University of Bedfordshire

CIS016-2 Object Oriented Programming and Software Engineering

Assignment 1: Control an Elevator – A C# Project

Student ID: 2216044

Name: Subarna Bajracharya

Table of Contents

Table of Figures

Table of Tables

Introduction

The Elevator Control System is developed for the purpose of operating a lift in a two floored office building. This system automates various functions like movement of the lift between different floors, managing the doors, emergency stop functions, along with storing a log of every action that happens in the database. A control panel is present inside the elevator and a button to call the elevator is present on the outside. A screen is also present that shows the status of the lift.

When the lift-call button present outside on each floor is pressed, the lift moves to that particular floor and upon reaching the door opens. The control panel inside the lift allows the passengers to select their desired floor, open and close doors, and also an emergency button.

The goal of this project is to create an efficient and reliable system that is easy to use, keeps people safe when using the lift, and also saves the log of every activity of the lift.

Problem Statement

The main challenge of this project is developing an elevator control system using OOP concept that works efficiently for a two floored office building. The lift must safely move between floors, the doors should function as requested, and overall give passengers a seamless and secure experience.

Main issues:

1. **Safety Mechanisms:** The elevator needs to consist of strong safety mechanisms. These mechanisms can include features like making sure the doors are closed when moving between different floors and having an emergency stop button in case of any kind of emergency.
2. **UI and Controls:** Both the control panel inside the elevator and the buttons outside on each floor should be easy to use and understandable for passengers.
3. **Handling Many Requests:** The system should handle multiple requests from floors at the same time and make sure that the system follows the best way.
4. **Keeping Records:** The system should display and record all the activity that the elevator does.
5. **Efficient Movement:** The elevator should not waste any time and should perform the desired action as fast and secure as possible, all while keeping the doors and movement in sync.

Proposed Solution

The designed solution uses an object-oriented concept to make the elevator smoothly work while keeping the passengers safe. This solution includes:

- **Elevator Movement Control:** The elevator should move up and down based on the button pressed by the passengers with state flags that limit and make sure the lift is moving in one direction at a time.
- **Automatic Door Operation**: Whenever the lift travels to any floor, the door should open and close automatically.
- **User Feedback:** The current activity status of the lift should be displayed to the passengers both inside the elevator and waiting outside the elevator.
- **Action logging:** Every activity of the elevator should be saved in the database.
- **Emergency Stop:** In emergency situations, the passenger can press the emergency button to stop the elevator and open all the doors, this also triggers a voice warning.

Aims and Objectives

Aim:

The project aims to develop an object-oriented application that manages the operations of an elevator in a two floored office building. This will make sure the lift works smoothly and securely travel between floors.

Objectives:

1. **Elevator Movement Control**: The elevator system should move the elevator up or down based on the buttons pressed by the passengers. The system should also make sure the lift travels smoothly between floors.
2. **Door Management:** The system should control when the elevator and floor doors open and close securely.
3. **User Interface**: The control panel and buttons should be understandable and easy to use and also the current activity should be displayed to passengers inside the elevator and passengers waiting outside.
4. **Emergency Handling**: The system should have an emergency button and upon pressed, it should stop the lift immediately, open all the doors, and send alerts.
5. **Activity Logging**: The system should also record all the activities of the elevator and record it in a database.
6. **Testing and Evaluation**: Each feature of the system should be thoroughly tested, and the secure working of the function should be confirmed.

Project Plan

The development plan of the elevator system is divided into six different phases, each phase will focus on certain parts of the project. This plan is designed to complete the project on time and achieve all the objectives.

Phase 1: Requirements Analysis (Week 1)

- **Task 1**: Analyze the project brief to fully gather all the requirements and all the necessary features.
- **Task 2:** Conduct research on similar elevator systems.

Phase 2: System Design (Week 2)

- **Task 1**: Design the prototype of the elevator system which includes the elevator itself, two floors, control panel, and a panel that shows the logs.
- **Task 2**: Design the UML diagrams for this application like Use case, Class, Activity and ER Diagram.

Phase 3: Application Development (Week 3 & 4)

- **Task 1:** Develop and implement the logic behind movement of elevators and the doors.
- **Task 2**: Implementing the logic for control panel, call buttons and emergency.
- **Task 3:** Integrate the database and activity record feature.

Phase 4: User Interface and Interaction (Week 4 & 5)

- **Task 1:** Develop the UI based on the prototype.
- **Task 2**: Ensure that the control panel and buttons UI is understandable and easy to use.

Phase 5: Testing and Debugging (Week 5)

- **Task 1:** Conduct test on every functionality of the movement of elevator, actions of doors, and inputs from the passengers.
- **Task 2**: Test the integration of functionality with control panel and buttons.
- **Task 3:** Debug any issue and optimize the performance of the application.

Phase 6: Documentation, Final Report, and Presentation (Week 6 & 7)

- **Task 1:** Document the code with detailed explanations of core logics used in the application.
- **Task 2:** Prepare the final report with all the resources and information included.
- **Task 3:** Review the project, finalize the report, and prepare for viva/presentation.

Use Case Diagram

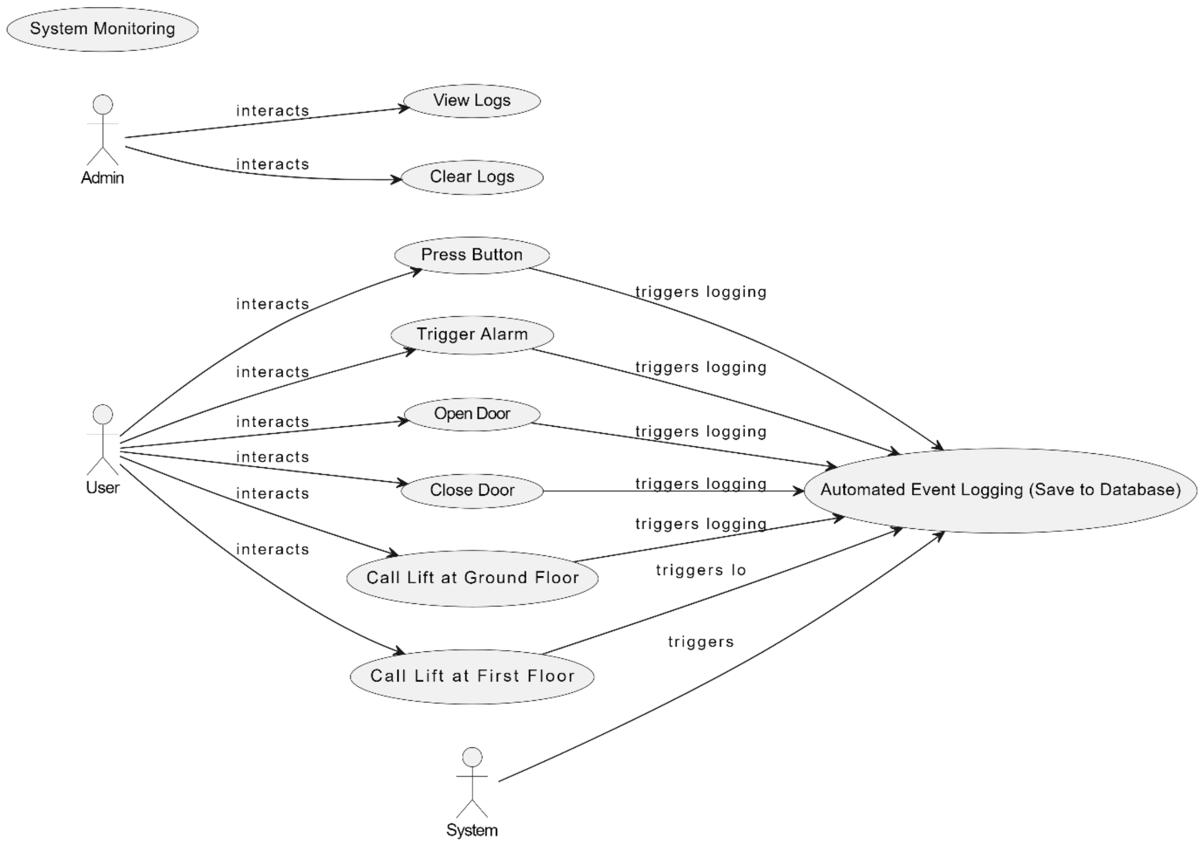
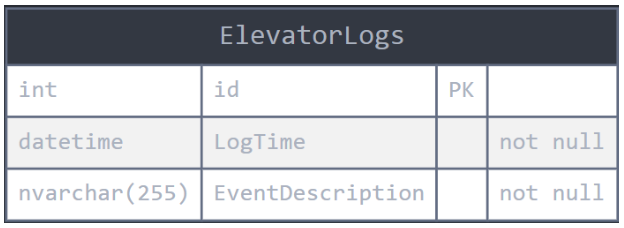


Figure 1: Use Case Diagram

The diagram shows all the interaction between the user, elevator system and admin. User can perform elevator operations by pressing buttons, opening/closing doors, triggering the alarm and calling the lift. The system will automatically record all activities of the elevator, while the admin can monitor the logs.

ER Diagram



ElevatorLogs			
int	id	PK	
datetime	LogTime		not null
nvarchar(255)	EventDescription		not null

Figure 2: ER Diagram

The ERD of the application consists of one entity; ElevatorLogs which will store the id, log time and the description of the event.

Class Diagram

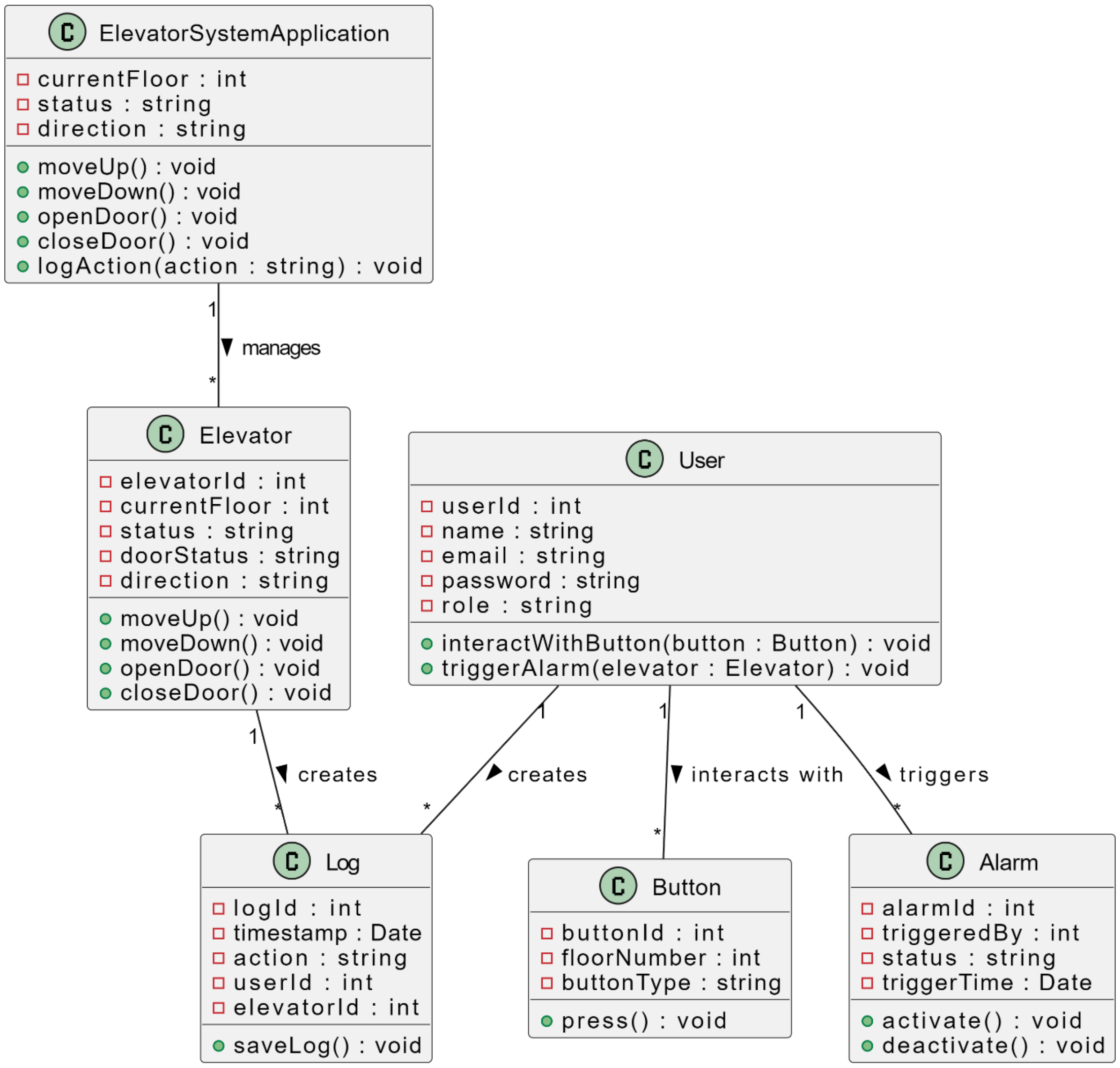


Figure 3: Class Diagram

The class diagram shows the form application managing all the elevator controls which includes the movement of the lift, door's opening/closing, and the action of the button. The application also stores all the logs in the database.

Activity Diagram

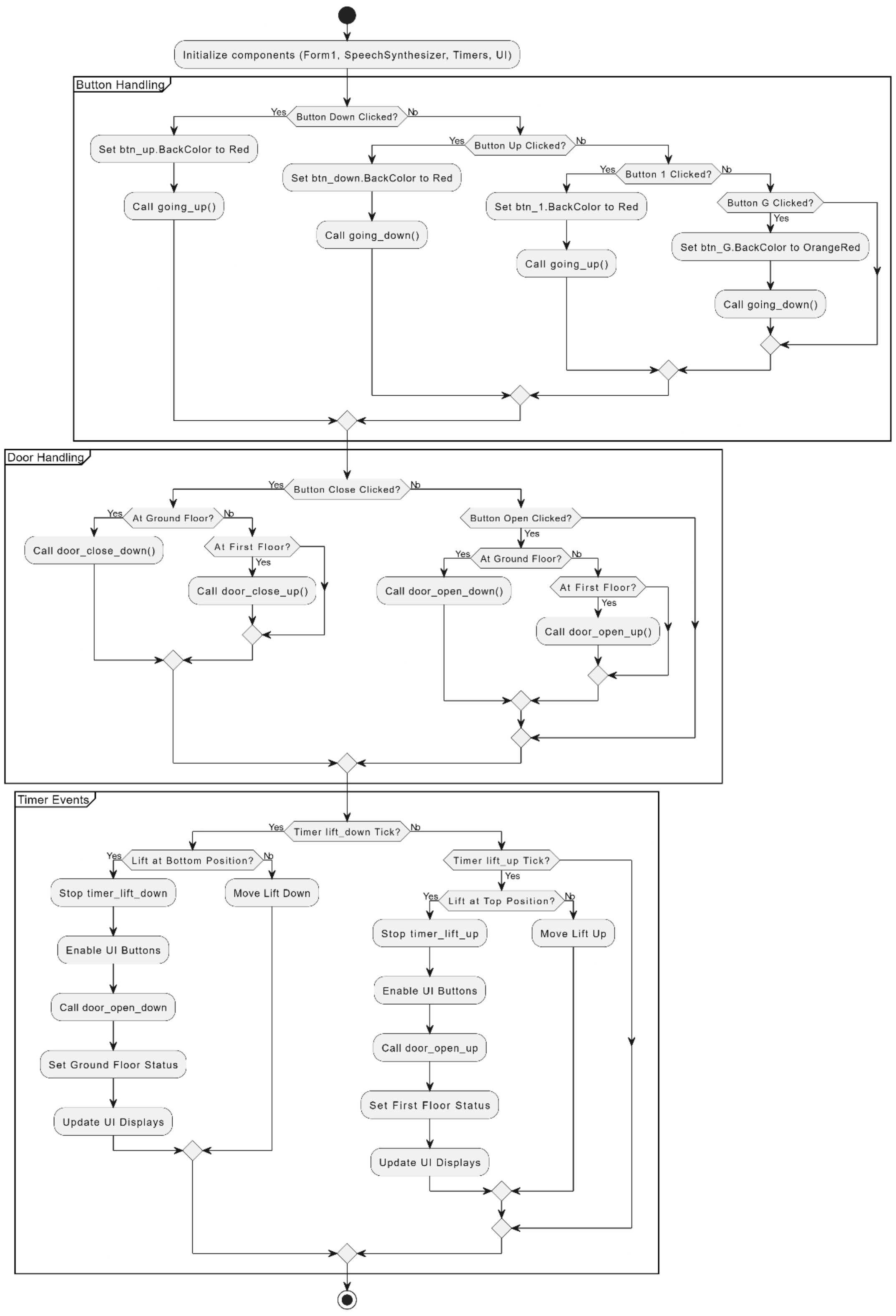


Figure 4: Activity Diagram

This activity diagram shows the processes of the whole elevator system. This includes handling the functioning of buttons for moving elevator, opening doors, and triggering alarm. Timer events manage how the elevator moves, how the door moves, and update the UI in real time.

Language and Database Used

The elevator system application is designed and developed using C# within the .NET Framework for a windows form application. C# was the ideal choice because of its strong OOP capabilities and easy GUI designing.

The database system used in this application is Microsoft SQL Server (MS SQL) as MS SQL is highly reliable with high performance and easy integration with C#.

Prototype

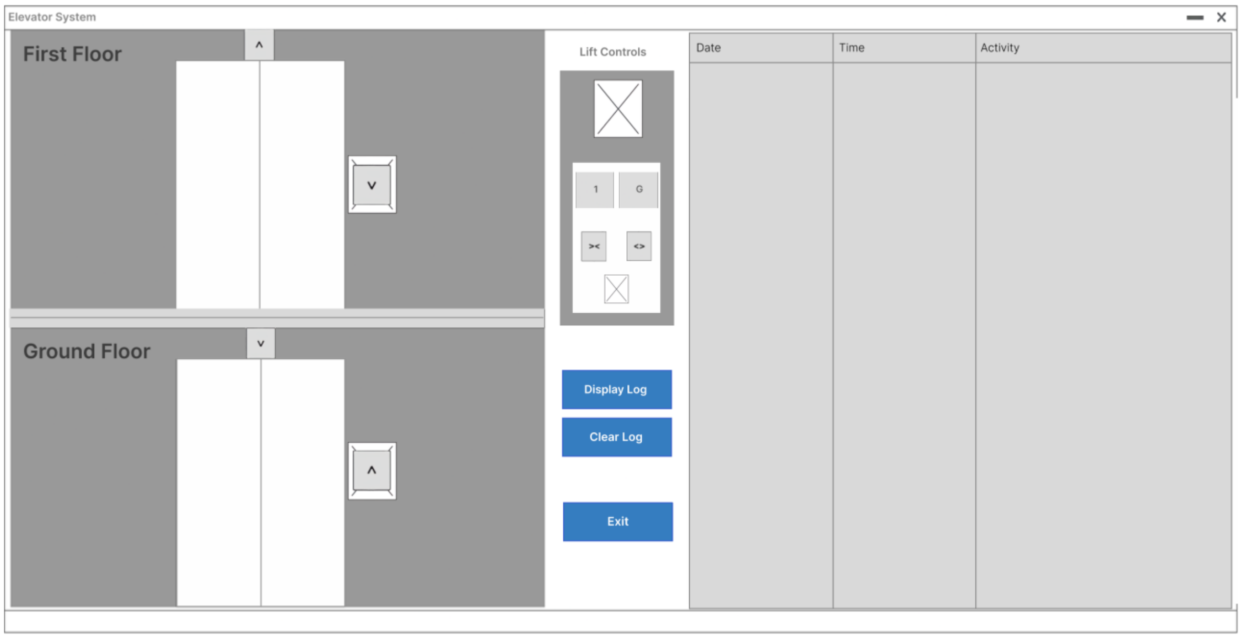
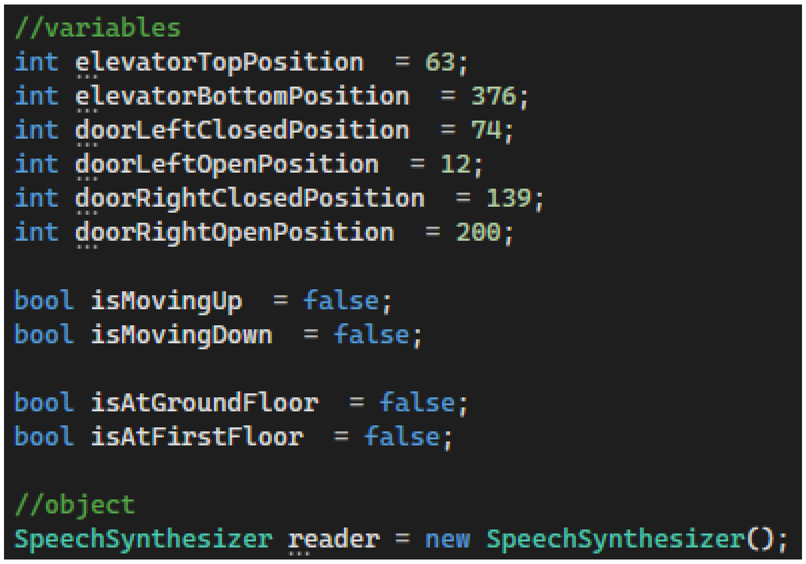


Figure 5: UI Prototype

The above picture shows the prototype of the planned elevator system application. This includes both the first floor and ground floor with call buttons outside each floor. The internal control panel is also included which consists of different controls like selecting the desired floor, opening/closing doors, and a display that shows the current activity status of the elevator. The prototype also includes a section which shows the activity logs, and some buttons related to it.

Implementation

1. Variable and Object Initialization



```
//variables
int elevatorTopPosition  = 63;
int elevatorBottomPosition  = 376;
int doorLeftClosedPosition  = 74;
int doorLeftOpenPosition  = 12;
int doorRightClosedPosition  = 139;
int doorRightOpenPosition  = 200;

bool isMovingUp  = false;
bool isMovingDown  = false;

bool isAtGroundFloor  = false;
bool isAtFirstFloor  = false;

//object
SpeechSynthesizer reader = new SpeechSynthesizer();
```

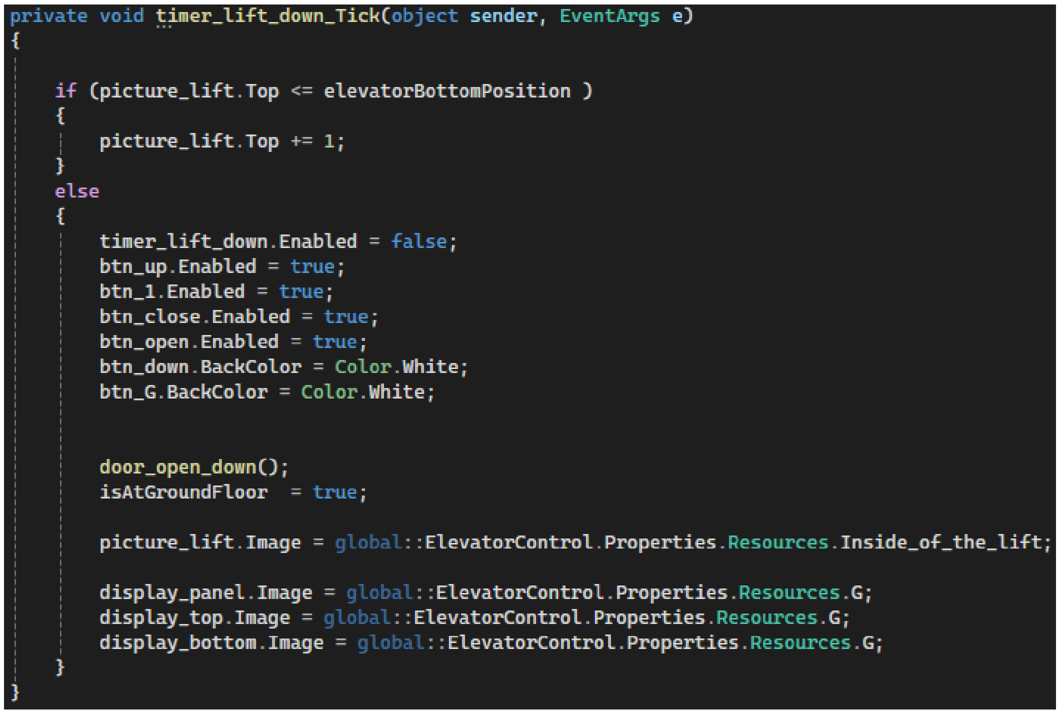
These variables defines the top / bottom positions of elevators and open/closed positions of doors.

Flags (isMovingUp, isMovingDown) determines the elevator’s direction whereas flags (isAtGroundFloor and isAtFirstFloor) tracks where the elevator currently is.

The object SpeechSynthesizer is used for providing passengers with voice feedback.

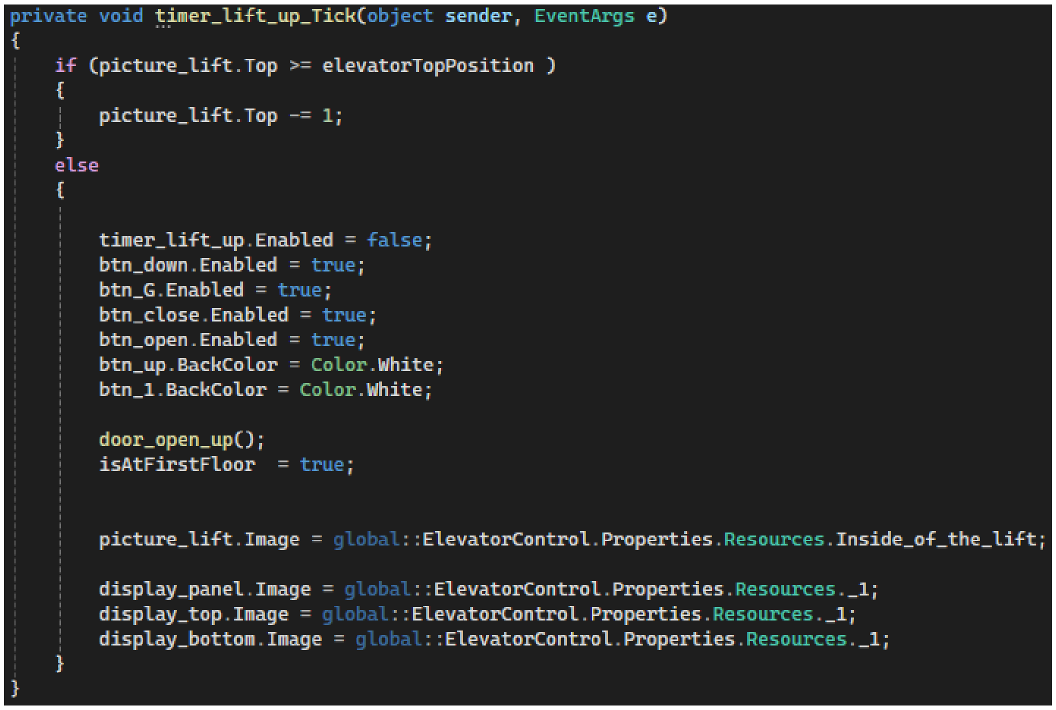
2. Elevator Movement Logic (Timer Ticks):

Moving Down (timer_lift_down_Tick):



```
private void timer_lift_down_Tick(object sender, EventArgs e)
{

    if (picture_lift.Top <= elevatorBottomPosition )
    {
        picture_lift.Top += 1;
    }
    else
    {
        timer_lift_down.Enabled = false;
        btn_up.Enabled = true;
        btn_1.Enabled = true;
        btn_close.Enabled = true;
        btn_open.Enabled = true;
        btn_down.BackColor = Color.White;
        btn_G.BackColor = Color.White;


        door_open_down();
        isAtGroundFloor  = true;

        picture_lift.Image = global::ElevatorControl.Properties.Resources.Inside_of_the_lift;

        display_panel.Image = global::ElevatorControl.Properties.Resources.G;
        display_top.Image = global::ElevatorControl.Properties.Resources.G;
        display_bottom.Image = global::ElevatorControl.Properties.Resources.G;
    }
}
```

Moving Up (timer_lift_up_Tick):



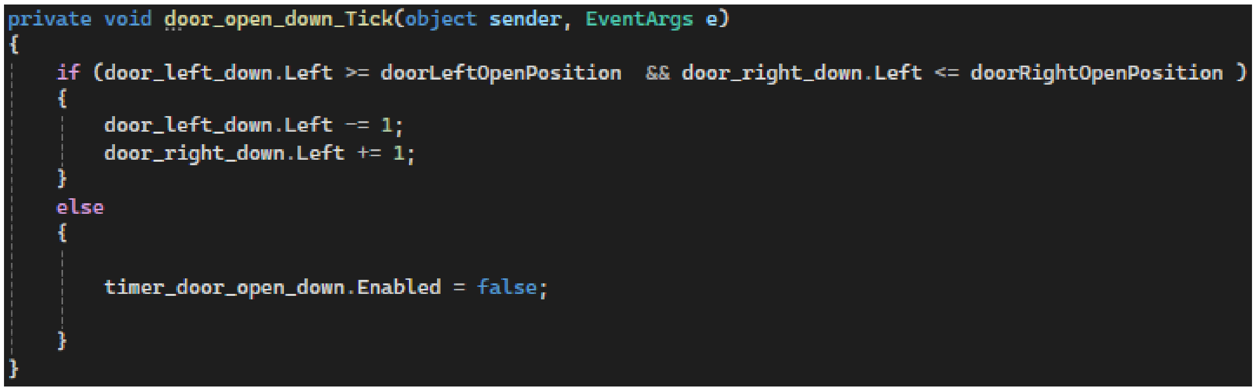
```
private void timer_lift_up_Tick(object sender, EventArgs e)
{
    if (picture_lift.Top >= elevatorTopPosition )
    {
        picture_lift.Top -= 1;
    }
    else
    {

        timer_lift_up.Enabled = false;
        btn_down.Enabled = true;
        btn_G.Enabled = true;
        btn_close.Enabled = true;
        btn_open.Enabled = true;
        btn_up.BackColor = Color.White;
        btn_1.BackColor = Color.White;

        door_open_up();
        isAtFirstFloor  = true;


        picture_lift.Image = global::ElevatorControl.Properties.Resources.Inside_of_the_lift;

        display_panel.Image = global::ElevatorControl.Properties.Resources._1;
        display_top.Image = global::ElevatorControl.Properties.Resources._1;
        display_bottom.Image = global::ElevatorControl.Properties.Resources._1;
    }
}
```

The movement of the elevator is controlled using timers (timer_lift_down, timer_lift_up) which moves the elevator up or down by changing the top property value of picture_lift.

3. **Door Open/Close Logic:**
Just like the position of the elevator, the door movement is also controlled using timers. There are separate timers for each direction:

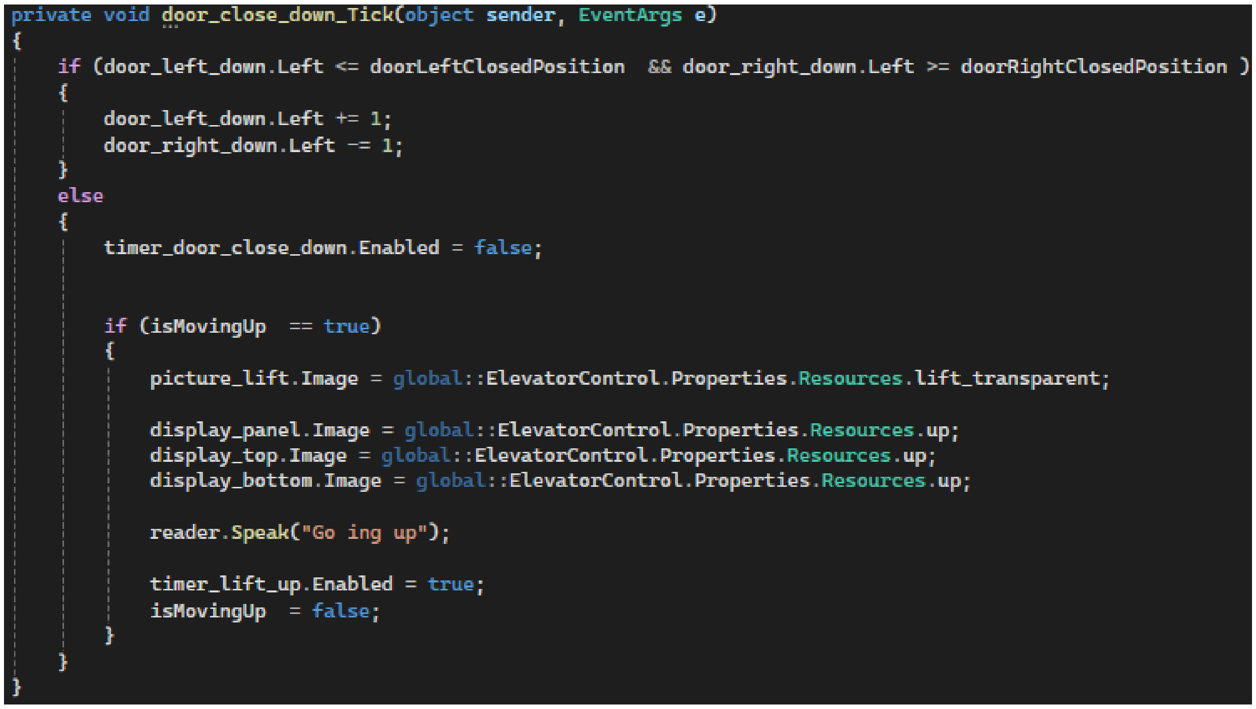
- door_open_down, door_close_down: This opens/closes the doors on the first floor.
- door_open_up, door_close_up: This opens/closes the doors on the ground floor.

Door Open Down (door_open_down):



```
private void door_open_down_Tick(object sender, EventArgs e)
{
    if (door_left_down.Left >= doorLeftOpenPosition  && door_right_down.Left <= doorRightOpenPosition )
    {
        door_left_down.Left -= 1;
        door_right_down.Left += 1;
    }
    else
    {

        timer_door_open_down.Enabled = false;

    }
}
```

Door Close Up (door_close_down):

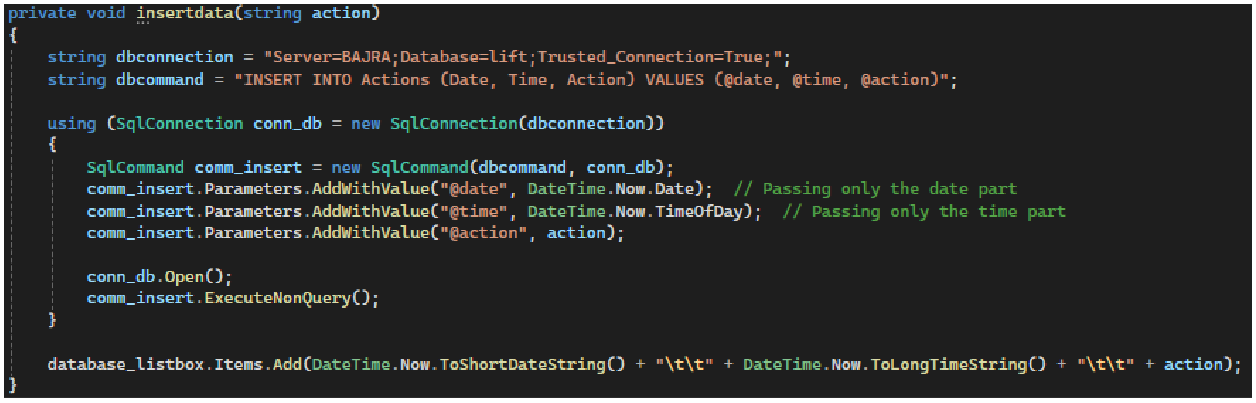


```
private void door_close_down_Tick(object sender, EventArgs e)
{
    if (door_left_down.Left <= doorLeftClosedPosition  && door_right_down.Left >= doorRightClosedPosition )
    {
        door_left_down.Left += 1;
        door_right_down.Left -= 1;
    }
    else
    {
        timer_door_close_down.Enabled = false;


        if (isMovingUp  == true)
        {
            picture_lift.Image = global::ElevatorControl.Properties.Resources.lift_transparent;

            display_panel.Image = global::ElevatorControl.Properties.Resources.up;
            display_top.Image = global::ElevatorControl.Properties.Resources.up;
            display_bottom.Image = global::ElevatorControl.Properties.Resources.up;

            reader.Speak("Go ing up");

            timer_lift_up.Enabled = true;
            isMovingUp  = false;
        }
    }
}
```

4. **Database Interaction**

The elevator system is designed to record and store every activity into the database using MS SQL Server and SQL command objects.

Inserting Data (insertdata):

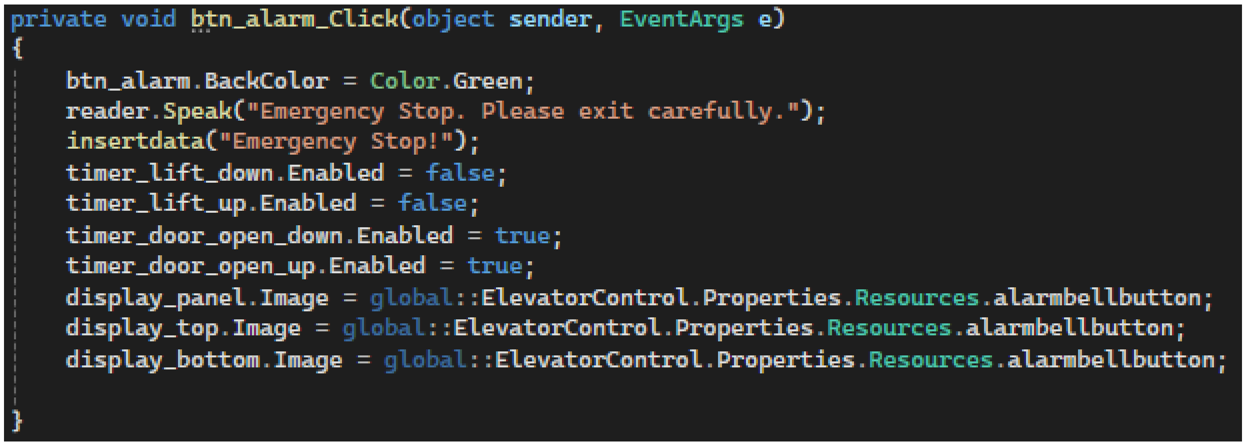


```
private void insertdata(string action)
{
    string dbconnection = "Server=BAJRA;Database=lift;Trusted_Connection=True;";
    string dbcommand = "INSERT INTO Actions (Date, Time, Action) VALUES (@date, @time, @action)";

    using (SqlConnection conn_db = new SqlConnection(dbconnection))
    {
        SqlCommand comm_insert = new SqlCommand(dbcommand, conn_db);
        comm_insert.Parameters.AddWithValue("@date", DateTime.Now.Date);  // Passing only the date part
        comm_insert.Parameters.AddWithValue("@time", DateTime.Now.TimeOfDay);  // Passing only the time part
        comm_insert.Parameters.AddWithValue("@action", action);

        conn_db.Open();
        comm_insert.ExecuteNonQuery();
    }

    database_listbox.Items.Add(DateTime.Now.ToShortDateString() + "\t\t" + DateTime.Now.ToLongTimeString() + "\t\t" + action);
}
```

5. **Emergency Stop Logic:**

During a time of emergency situations, the passengers can press the emergency button that stops the elevator, opens doors on both floors and a emergency message is said out loud by the system.

Emergency Stop (btn_alarm_Click):

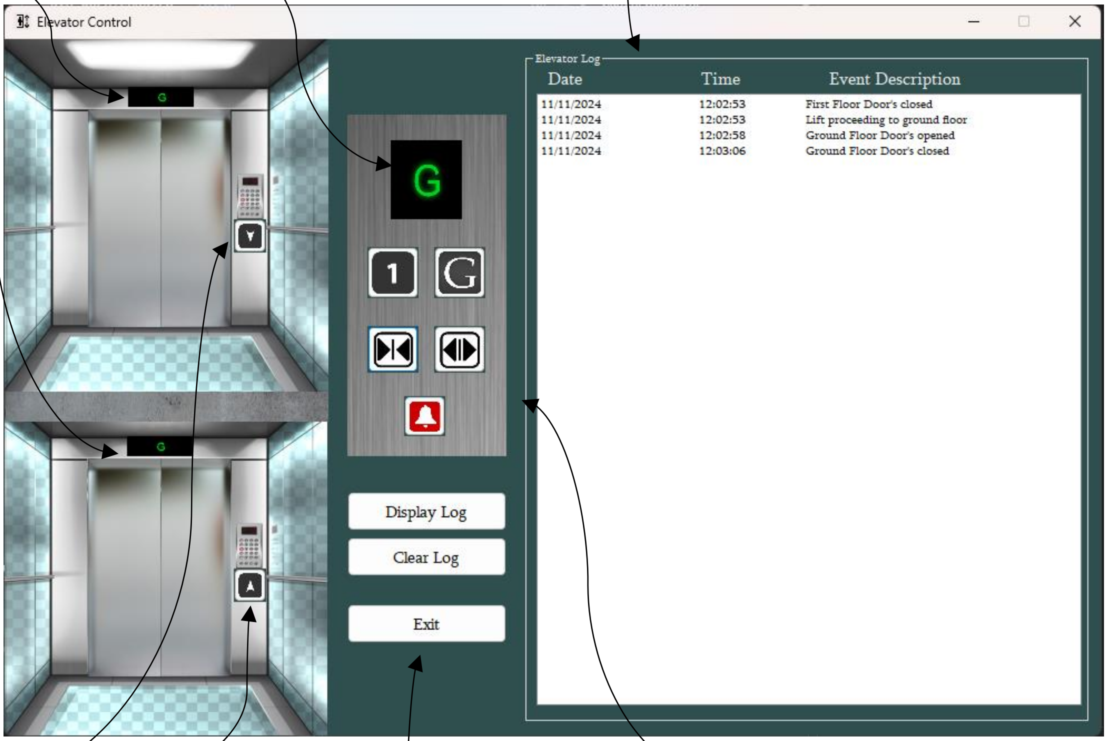


```
private void btn_alarm_Click(object sender, EventArgs e)
{
    btn_alarm.BackColor = Color.Green;
    reader.Speak("Emergency Stop. Please exit carefully.");
    insertdata("Emergency Stop!");
    timer_lift_down.Enabled = false;
    timer_lift_up.Enabled = false;
    timer_door_open_down.Enabled = true;
    timer_door_open_up.Enabled = true;
    display_panel.Image = global::ElevatorControl.Properties.Resources.alarmbellbutton;
    display_top.Image = global::ElevatorControl.Properties.Resources.alarmbellbutton;
    display_bottom.Image = global::ElevatorControl.Properties.Resources.alarmbellbutton;

}
```

UI Design explanation:

A database list box that displays the log of the elevator showing date, time and event description.

Shows the current activity of the lift



Elevator Log

Date	Time	Event Description
11/11/2024	12:02:53	First Floor Door's closed
11/11/2024	12:02:53	Lift proceeding to ground floor
11/11/2024	12:02:58	Ground Floor Door's opened
11/11/2024	12:03:06	Ground Floor Door's closed

Figure 6: UI Design Explanation

Lift Request Buttons present outside both floor's lift.

Control panel inside the elevator consisting of 5 buttons and a display panel. Buttons include first floor button, ground floor button, door close button, door open button, and an emergency button.

Display Log button which displays the whole log stored in the database, Clear log button which only clears the displayed log and Exit button which exits the application.

Note: The snapshots of the application under different functionality are present in the Appendix section.

Testing Table

Test Case ID	Description	Steps to Reproduce	Expected Result	Pass/ Fail	Notes/Encountered Issues
TC01	Test btn_down functionality	Click btn_down button, Observe the lift	Lift should move down; doors should close at ground floor	Pass	Ensure the lift moves seamlessly with no unexpected actions.
TC02	Test btn_up functionlity	Click btn_up button, Observe the lift	Lift should move up; doors should close at first floor	Pass	Check if the lift moves up consistently even if the button is pressed multiple times
TC03	Test door_open_down method	Manually call door_open_down()	Ground floor doors should open	Pass	There should be no delay.
TC04	Test door_close_up method	Manually call door_close_up()	First floor doors should close	Pass	
TC05	Test timer_lift_down functionality	Enable timer_lift_down	Lift should move towards the ground floor	Pass	This shouldn't cause any inconsistent movements.
TC06	Test timer_lift_up functionality	Enable timer_lift_up	Lift should move towards the first floor	Pass	Shouldn't cause any inconsistent movements.
TC07	Test insertdata method for logging	Call insertdata() with a string	Log entries should be created in the database.	Pass	
TC08	Testing btn_displaylog and btn_clearlog	Click btn_displaylog, btn_clearlog	Log data should be fetched and displayed. Displayed data should be cleared	Pass	
TC09	Test btn_alarm	Click btn_alarm	Emergency stop message should be spoken and the lift should be stopped, and all the doors should be opened	Pass	
TC10	Test elevator display panel images	Observe the display panel during lift movement	Appropriate images like going up, going down, G, 1 should appear	Pass	Ensure that the image transition is smooth and matches the current state of the lift.
TC11	Test door movement limits	Let doors move, Check end positions	Doors should stop moving at open/close positions	Pass	Ensure the door's stopping positions are accurate.

Table 1: Testing Table

Conclusion

In conclusion, it is safe to say that the elevator system application successfully meets all the goals and objectives by making sure the elevator moves swiftly, the doors operations work seamlessly, the panel displays the current activity of the elevator correctly, and the emergency stop button works securely. All of these features were tested thoroughly, and they are performed as expected.

While the task was to develop a system for just one elevator with two floors, future upgrades might include supporting more elevators and more floors. In summary, this project shows how C# can be used in such real-time systems and provides a very solid base for any future upgrades.

Self-Evaluation Table

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	10	Yes	TC01, TC02, TC10	10
	Skeleton of event handlers in place for all buttons	10	Yes	TC01–TC11	10
Task 2	All event handlers are functional	10	Yes	TC01-TC11	10
Task 3	Database (DB) is designed and can be connected	5	Yes	TC07, TC08	5
	Log Information can be retrieved from DB and displayed in the GUI	5	Yes	TC08	5
	When the log button is pressed, log information is sent to and stored in the DB	5	Yes	TC07, TC08	5
	Use the disconnected model rather than connected model (Data source is updated via DataAdapters Update() method instead of	5	Yes	Testing confirmed functionality	5

	ExecuteNonQuery() method)				
	Using relative path instead of absolute path	5	Yes	-	5
	Avoiding any duplication among the event handlers over the database related functions	5	Yes	TC07, TC08	5
Task 4	Events described in Task 2 animated using delegation and timer	10	Yes	TC01, TC02, TC05, TC06	10
Task 5	Eliminating logical errors and handling exceptions with try and catch	5	Yes	TC01-TC11	5
	Optimise the efficiency of GUI by implementing multiple tasks concurrently via BackgroundWorker	5	No	-	0
	Use state patterns instead of if-else statements to accommodate future changes of the requirement	10	No	-	0
Task 6	Testing report	10	Yes	TC01-TC11	10
Total		100			85

Table 2: Self Evalutation Table

References

Microsoft, 2021. *Windows Forms Application (C#)*. [online] Available at: https://docs.microsoft.com/en-us/dotnet/desktop/winforms/ [Accessed 11 November 2024].

Robustelli, D. (2019). *C# Programming: From Problem Analysis to Program Design*. 6th ed. Boston: Cengage Learning.

Eberle, A. and Gill, M., 2018. *Elevator Control Systems: A Practical Guide*. 2nd ed. New York: Springer.

draw.io, 2024. *draw.io: Online Diagram Software*. [online] Available at: https://app.diagrams.net/ [Accessed 11 November 2024].

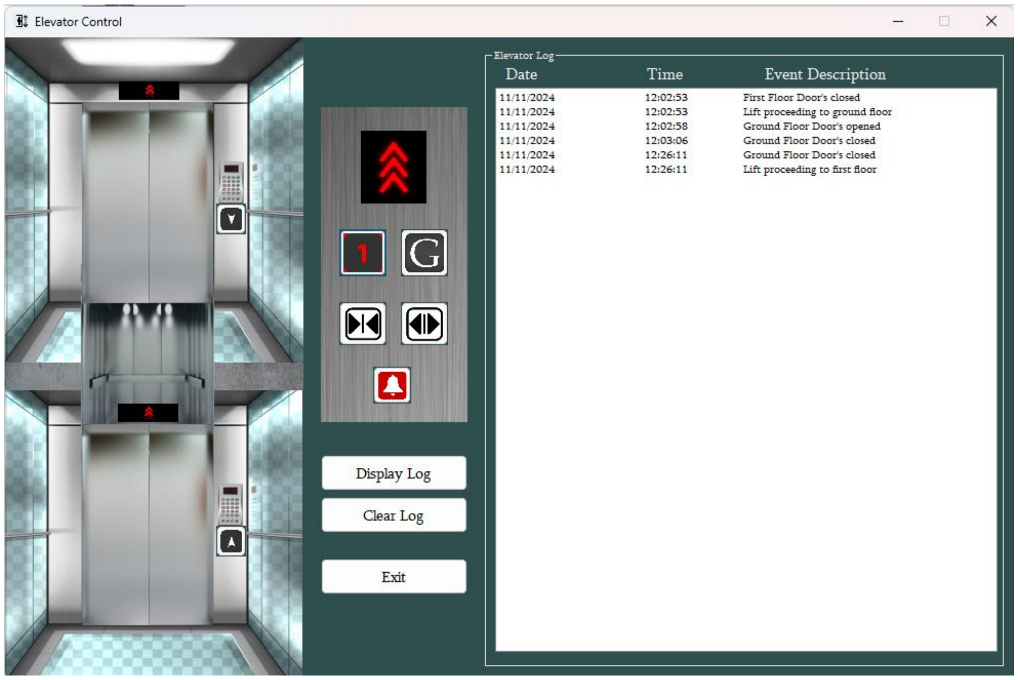
Figma, 2024. *Figma: The Collaborative Interface Design Tool*. [online] Available at: https://www.figma.com [Accessed 11 November 2024]

Appendix

The project seems to exceed the file limit in BREO so I uploaded the whole project in github. here's the Github link to the project: Click Me

Snapshots of application under different features

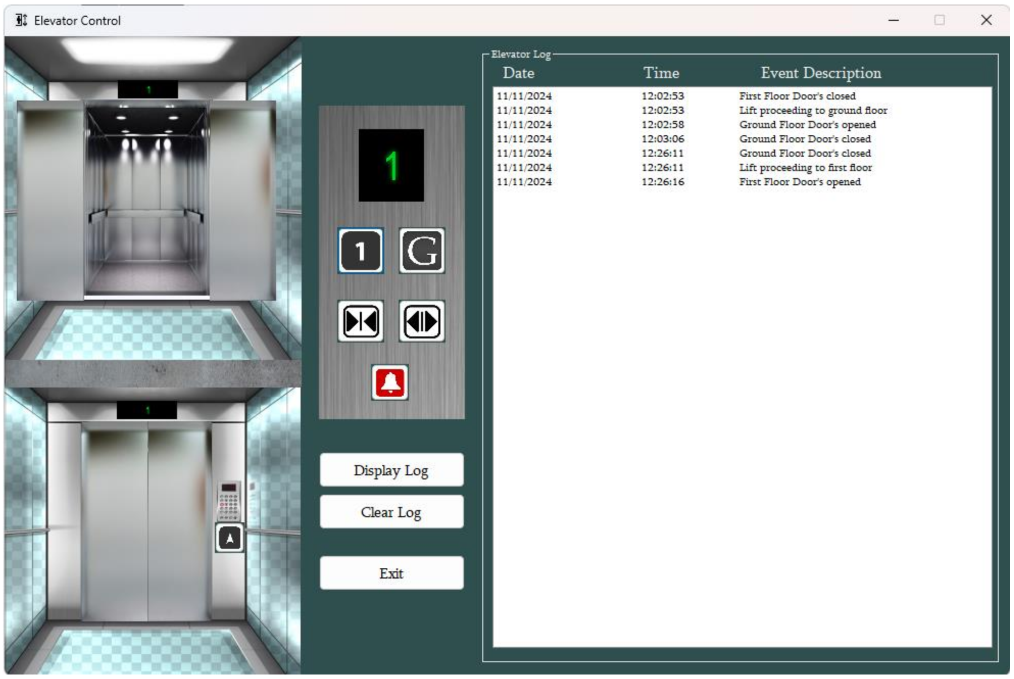
1. Elevator moving between floors (image shows elevator move to first floor):



Date	Time	Event Description
11/11/2024	12:02:53	First Floor Door's closed
11/11/2024	12:02:53	Lift proceeding to ground floor
11/11/2024	12:02:58	Ground Floor Door's opened
11/11/2024	12:03:06	Ground Floor Door's closed
11/11/2024	12:26:11	Ground Floor Door's closed
11/11/2024	12:26:11	Lift proceeding to first floor

Figure 7: UI 1

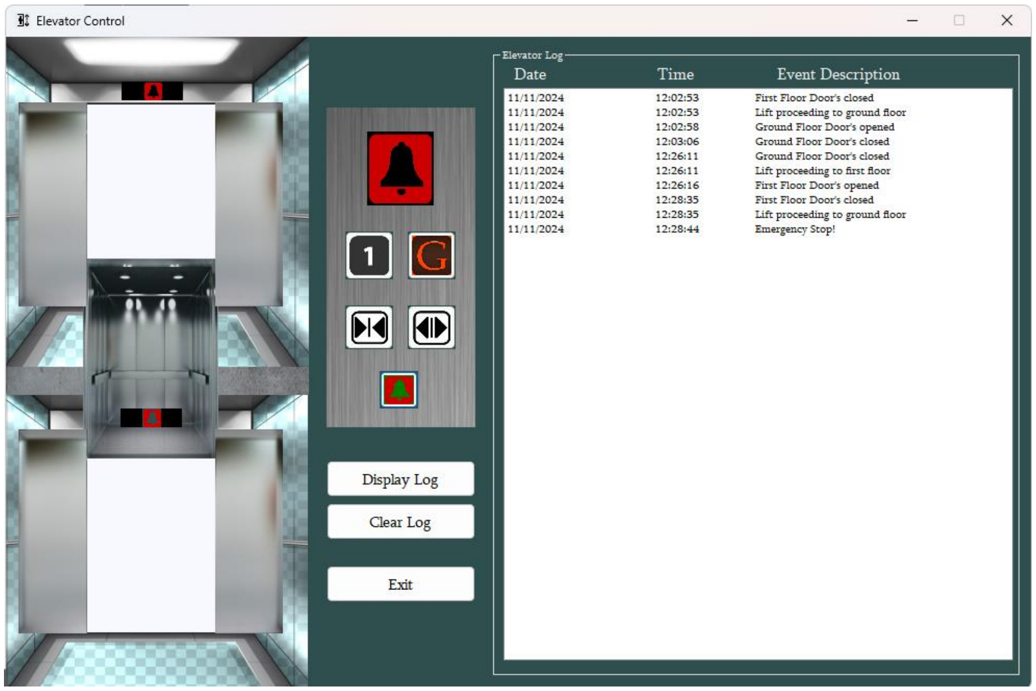
2. Elevator when doors are opened (image shows elevator is in first floor so door's of first floor is opened):



```
Elevator Log
Date          Time        Event Description
11/11/2024    12:02:53    First Floor Door's closed
11/11/2024    12:02:53    Lift proceeding to ground floor
11/11/2024    12:02:58    Ground Floor Door's opened
11/11/2024    12:03:06    Ground Floor Door's closed
11/11/2024    12:26:11    Ground Floor Door's closed
11/11/2024    12:26:11    Lift proceeding to first floor
11/11/2024    12:26:16    First Floor Door's opened
```

Figure 8: UI 2

3. Elevator when emergency button is pressed:



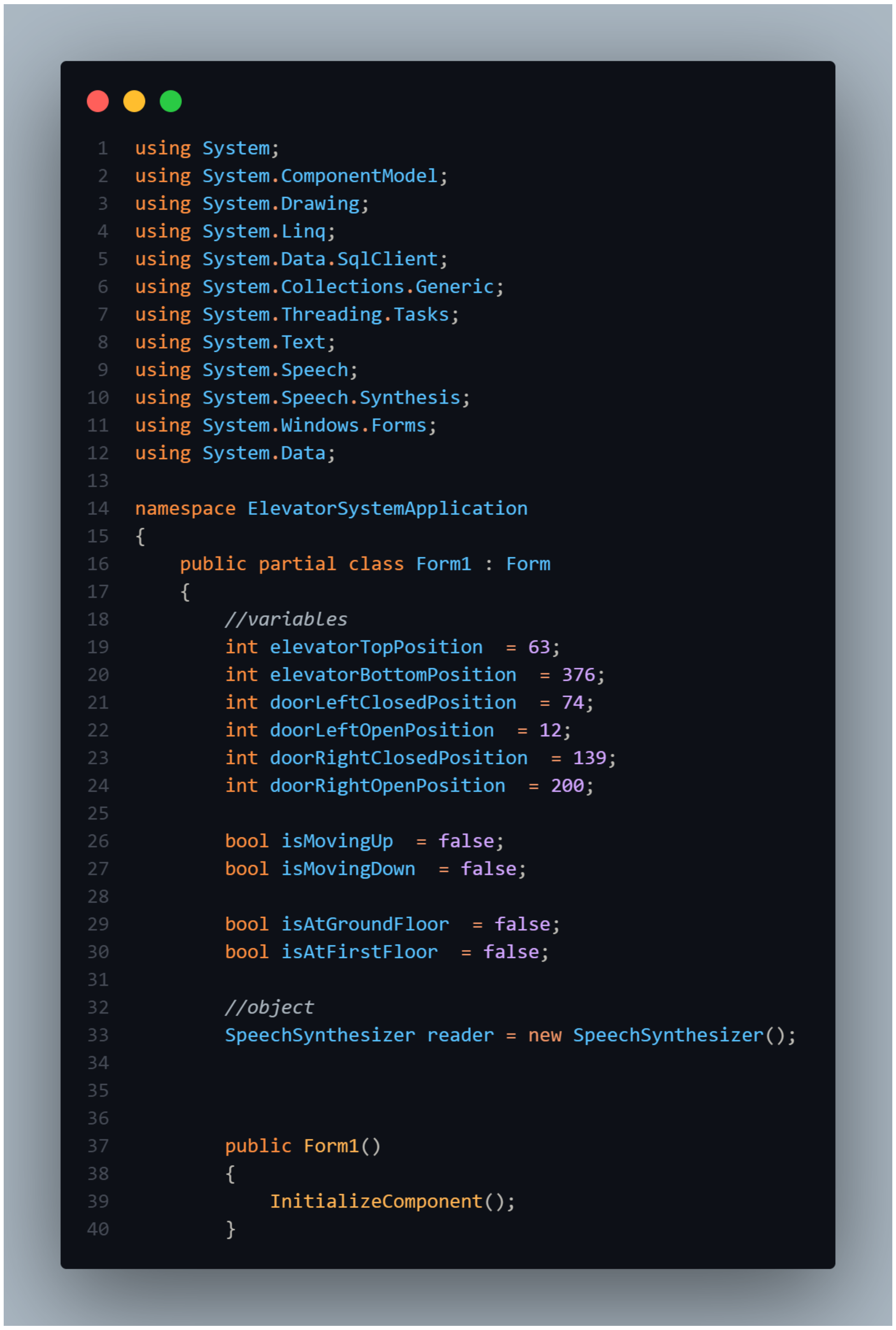
```
Elevator Log
Date          Time        Event Description
11/11/2024    12:02:53    First Floor Door's closed
11/11/2024    12:02:53    Lift proceeding to ground floor
11/11/2024    12:02:58    Ground Floor Door's opened
11/11/2024    12:03:06    Ground Floor Door's closed
11/11/2024    12:26:11    Ground Floor Door's closed
11/11/2024    12:26:11    Lift proceeding to first floor
11/11/2024    12:26:16    First Floor Door's opened
11/11/2024    12:28:35    First Floor Door's closed
11/11/2024    12:28:35    Lift proceeding to ground floor
11/11/2024    12:28:44    Emergency Stop!
```

Figure 9: UI 3

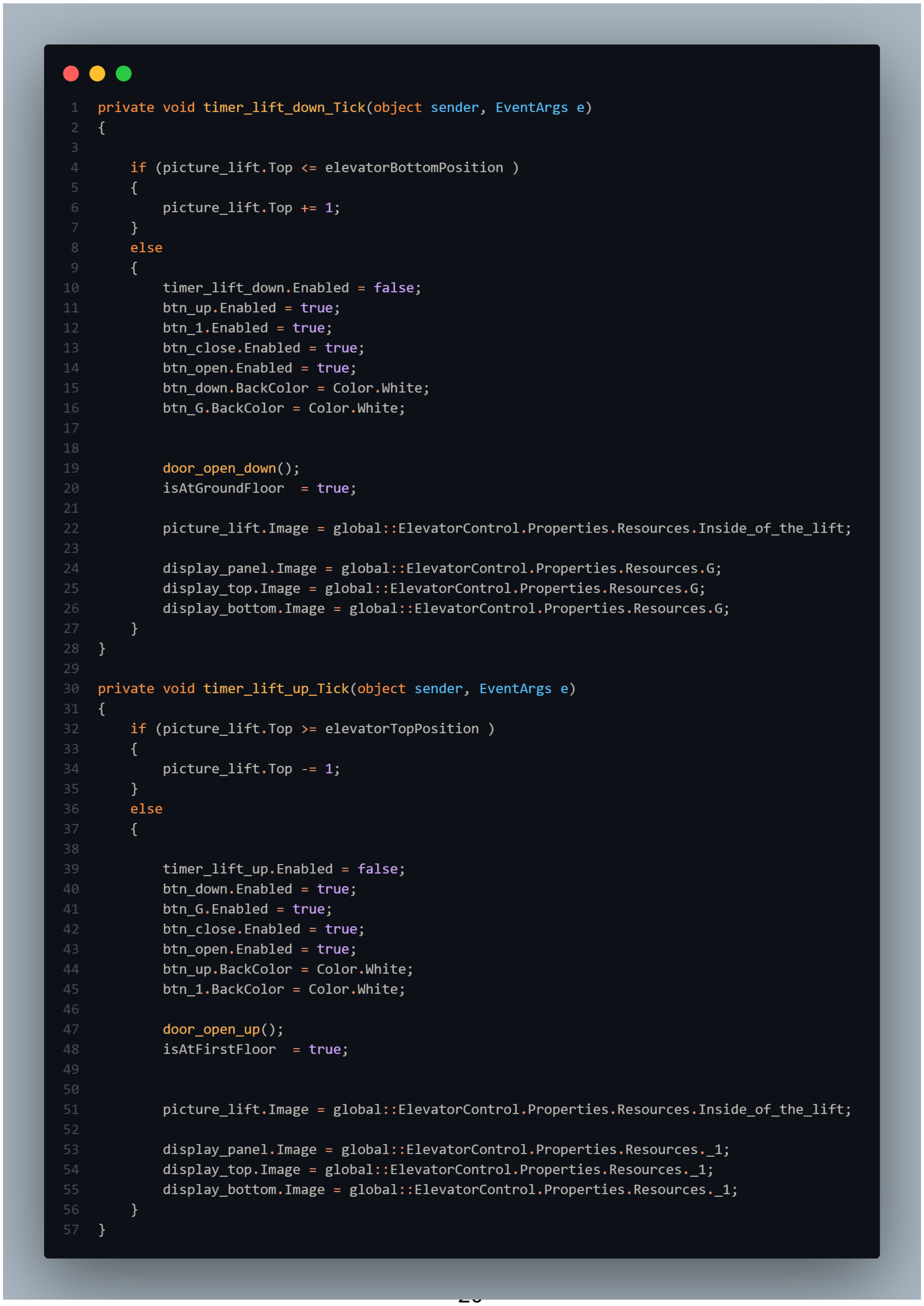
Complete source code

Github Link to complete source code: Click Me

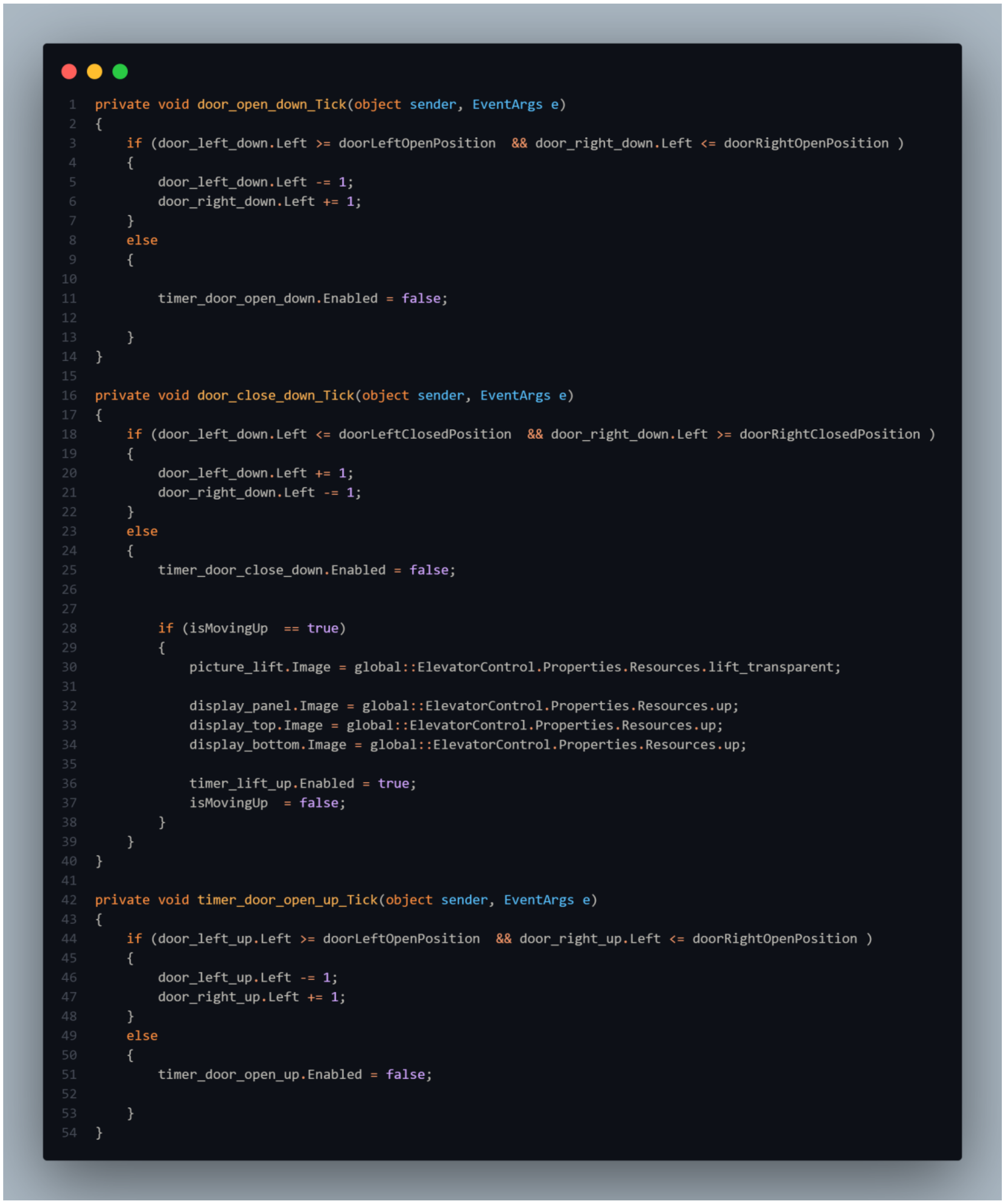
Form1.cs:



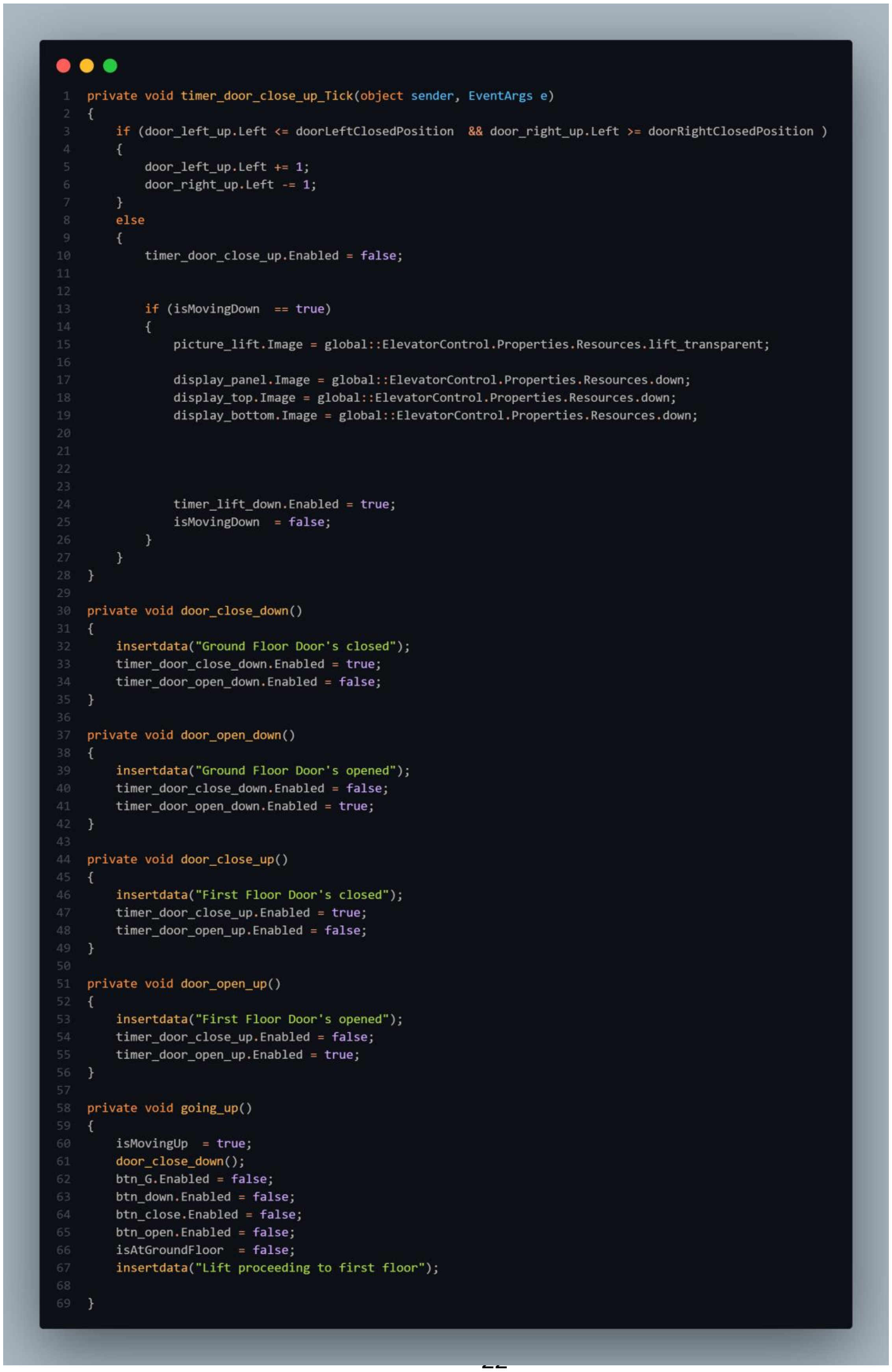
```
using System;
using System.ComponentModel;
using System.Drawing;
using System.Linq;
using System.Data.SqlClient;
using System.Collections.Generic;
using System.Threading.Tasks;
using System.Text;
using System.Speech;
using System.Speech.Synthesis;
using System.Windows.Forms;
using System.Data;

namespace ElevatorSystemApplication
{
    public partial class Form1 : Form
    {
        //variables
        int elevatorTopPosition  = 63;
        int elevatorBottomPosition  = 376;
        int doorLeftClosedPosition  = 74;
        int doorLeftOpenPosition  = 12;
        int doorRightClosedPosition  = 139;
        int doorRightOpenPosition  = 200;

        bool isMovingUp  = false;
        bool isMovingDown  = false;

        bool isAtGroundFloor  = false;
        bool isAtFirstFloor  = false;

        //object
        SpeechSynthesizer reader = new SpeechSynthesizer();



        public Form1()
        {
            InitializeComponent();
        }
```



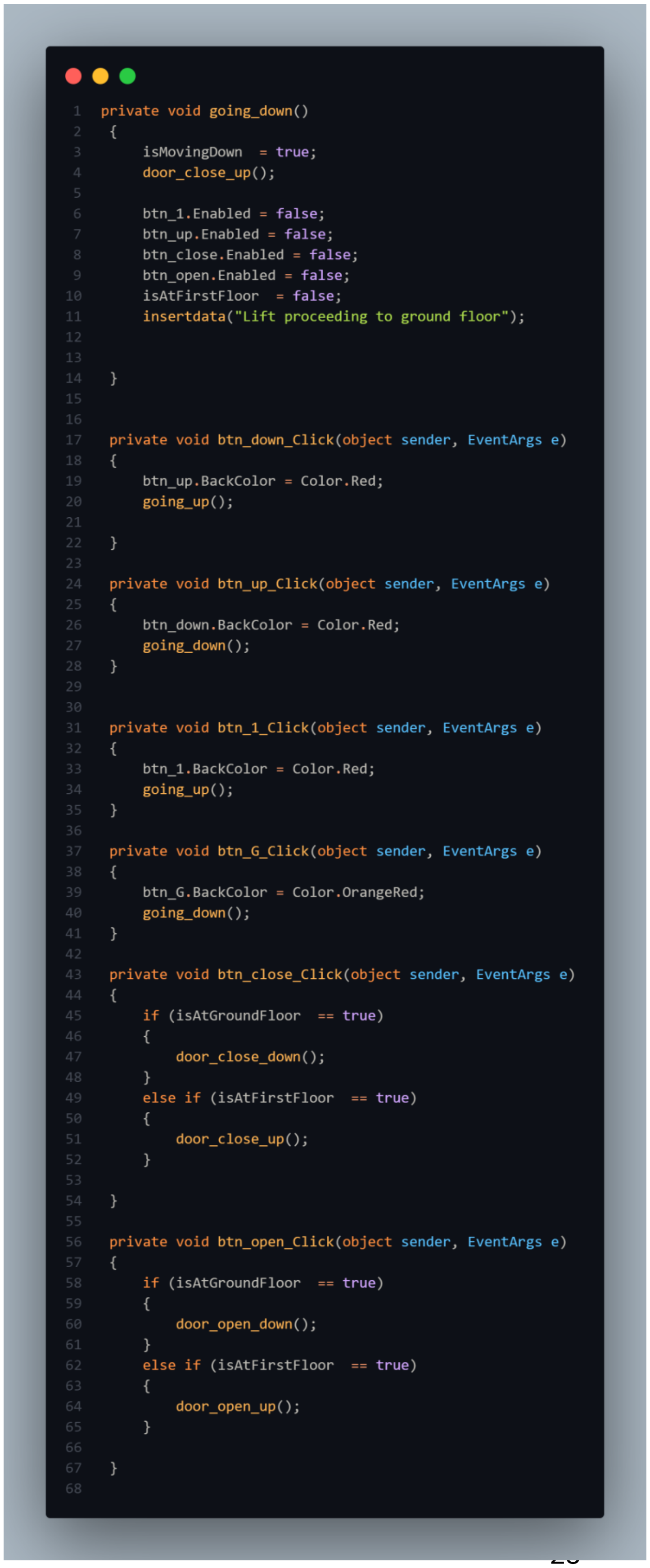
```
private void timer_lift_down_Tick(object sender, EventArgs e)
{

    if (picture_lift.Top <= elevatorBottomPosition )
    {
        picture_lift.Top += 1;
    }
    else
    {
        timer_lift_down.Enabled = false;
        btn_up.Enabled = true;
        btn_1.Enabled = true;
        btn_close.Enabled = true;
        btn_open.Enabled = true;
        btn_down.BackColor = Color.White;
        btn_G.BackColor = Color.White;


        door_open_down();
        isAtGroundFloor  = true;

        picture_lift.Image = global::ElevatorControl.Properties.Resources.Inside_of_the_lift;

        display_panel.Image = global::ElevatorControl.Properties.Resources.G;
        display_top.Image = global::ElevatorControl.Properties.Resources.G;
        display_bottom.Image = global::ElevatorControl.Properties.Resources.G;
    }
}

private void timer_lift_up_Tick(object sender, EventArgs e)
{
    if (picture_lift.Top >= elevatorTopPosition )
    {
        picture_lift.Top -= 1;
    }
    else
    {

        timer_lift_up.Enabled = false;
        btn_down.Enabled = true;
        btn_G.Enabled = true;
        btn_close.Enabled = true;
        btn_open.Enabled = true;
        btn_up.BackColor = Color.White;
        btn_1.BackColor = Color.White;

        door_open_up();
        isAtFirstFloor  = true;


        picture_lift.Image = global::ElevatorControl.Properties.Resources.Inside_of_the_lift;

        display_panel.Image = global::ElevatorControl.Properties.Resources._1;
        display_top.Image = global::ElevatorControl.Properties.Resources._1;
        display_bottom.Image = global::ElevatorControl.Properties.Resources._1;
    }
}
```



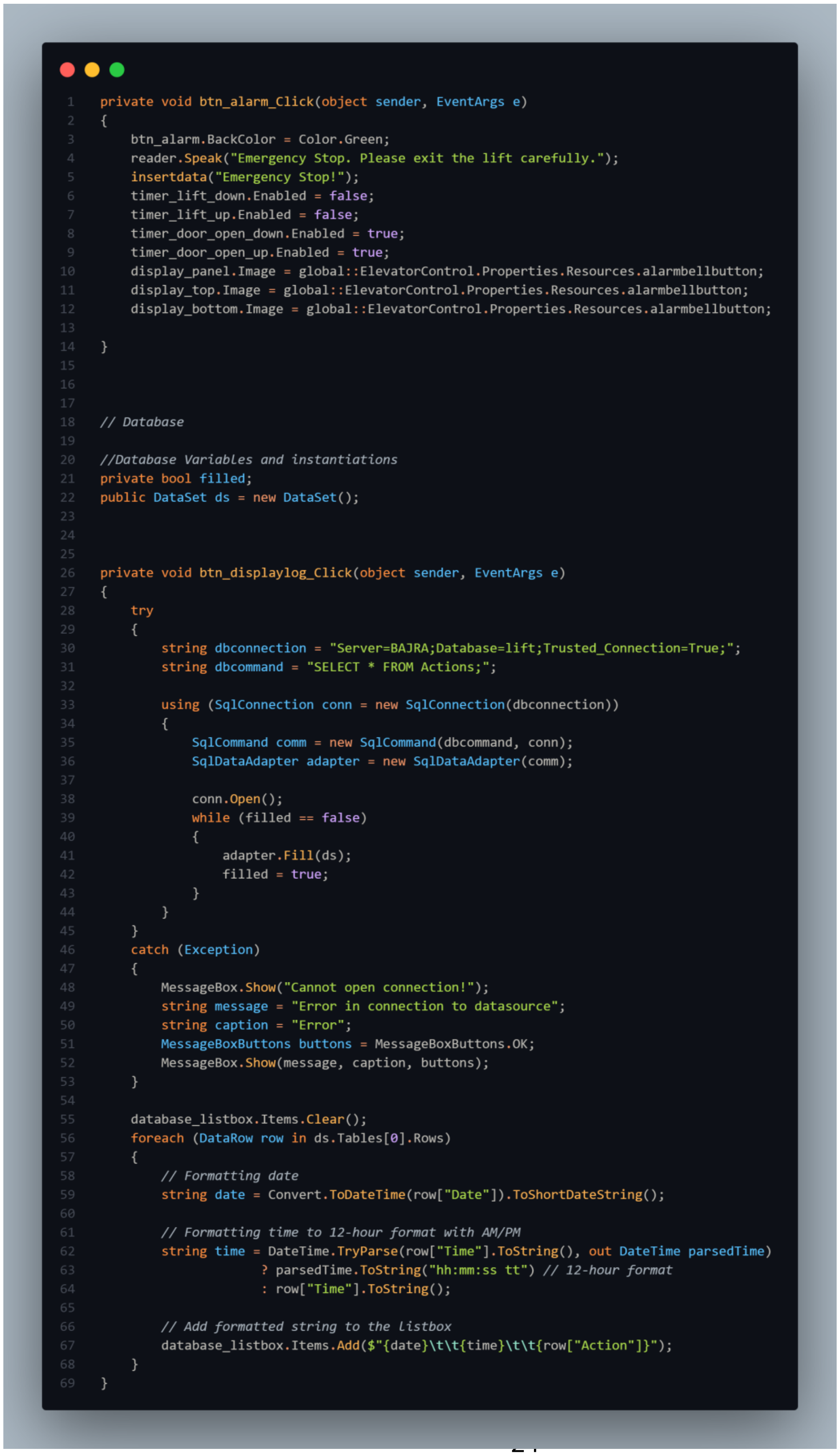
```
private void door_open_down_Tick(object sender, EventArgs e)
{
    if (door_left_down.Left >= doorLeftOpenPosition  && door_right_down.Left <= doorRightOpenPosition )
    {
        door_left_down.Left -= 1;
        door_right_down.Left += 1;
    }
    else
    {

        timer_door_open_down.Enabled = false;

    }
}

private void door_close_down_Tick(object sender, EventArgs e)
{
    if (door_left_down.Left <= doorLeftClosedPosition  && door_right_down.Left >= doorRightClosedPosition )
    {
        door_left_down.Left += 1;
        door_right_down.Left -= 1;
    }
    else
    {
        timer_door_close_down.Enabled = false;


        if (isMovingUp  == true)
        {
            picture_lift.Image = global::ElevatorControl.Properties.Resources.lift_transparent;

            display_panel.Image = global::ElevatorControl.Properties.Resources.up;
            display_top.Image = global::ElevatorControl.Properties.Resources.up;
            display_bottom.Image = global::ElevatorControl.Properties.Resources.up;

            timer_lift_up.Enabled = true;
            isMovingUp  = false;
        }
    }
}

private void timer_door_open_up_Tick(object sender, EventArgs e)
{
    if (door_left_up.Left >= doorLeftOpenPosition  && door_right_up.Left <= doorRightOpenPosition )
    {
        door_left_up.Left -= 1;
        door_right_up.Left += 1;
    }
    else
    {
        timer_door_open_up.Enabled = false;

    }
}
```



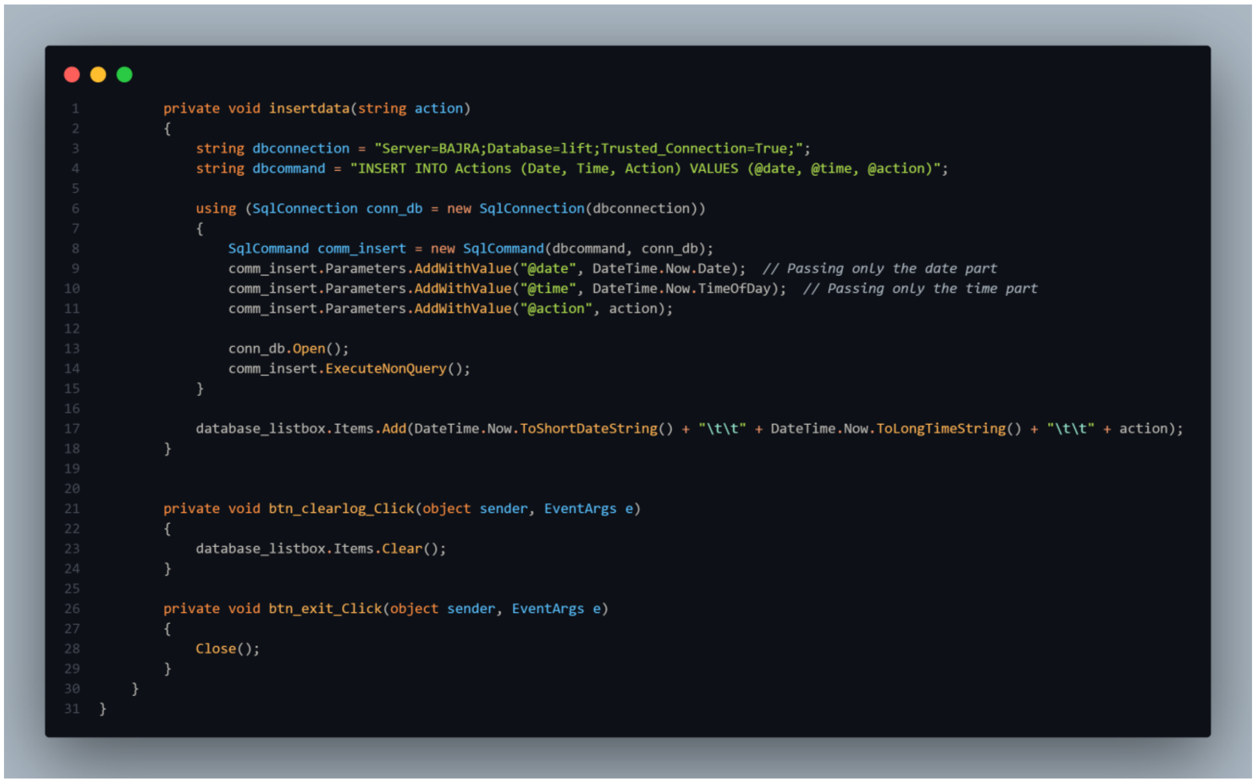
```
private void timer_door_close_up_Tick(object sender, EventArgs e)
{
    if (door_left_up.Left <= doorLeftClosedPosition  && door_right_up.Left >= doorRightClosedPosition )
    {
        door_left_up.Left += 1;
        door_right_up.Left -= 1;
    }
    else
    {
        timer_door_close_up.Enabled = false;


        if (isMovingDown  == true)
        {
            picture_lift.Image = global::ElevatorControl.Properties.Resources.lift_transparent;

            display_panel.Image = global::ElevatorControl.Properties.Resources.down;
            display_top.Image = global::ElevatorControl.Properties.Resources.down;
            display_bottom.Image = global::ElevatorControl.Properties.Resources.down;




            timer_lift_down.Enabled = true;
            isMovingDown  = false;
        }
    }
}

private void door_close_down()
{
    insertdata("Ground Floor Door's closed");
    timer_door_close_down.Enabled = true;
    timer_door_open_down.Enabled = false;
}

private void door_open_down()
{
    insertdata("Ground Floor Door's opened");
    timer_door_close_down.Enabled = false;
    timer_door_open_down.Enabled = true;
}

private void door_close_up()
{
    insertdata("First Floor Door's closed");
    timer_door_close_up.Enabled = true;
    timer_door_open_up.Enabled = false;
}

private void door_open_up()
{
    insertdata("First Floor Door's opened");
    timer_door_close_up.Enabled = false;
    timer_door_open_up.Enabled = true;
}

private void going_up()
{
    isMovingUp  = true;
    door_close_down();
    btn_G.Enabled = false;
    btn_down.Enabled = false;
    btn_close.Enabled = false;
    btn_open.Enabled = false;
    isAtGroundFloor  = false;
    insertdata("Lift proceeding to first floor");

}
```



```
private void going_down()
 {
     isMovingDown  = true;
     door_close_up();

     btn_1.Enabled = false;
     btn_up.Enabled = false;
     btn_close.Enabled = false;
     btn_open.Enabled = false;
     isAtFirstFloor  = false;
     insertdata("Lift proceeding to ground floor");


 }


 private void btn_down_Click(object sender, EventArgs e)
 {
     btn_up.BackColor = Color.Red;
     going_up();

 }

 private void btn_up_Click(object sender, EventArgs e)
 {
     btn_down.BackColor = Color.Red;
     going_down();
 }


 private void btn_1_Click(object sender, EventArgs e)
 {
     btn_1.BackColor = Color.Red;
     going_up();
 }

 private void btn_G_Click(object sender, EventArgs e)
 {
     btn_G.BackColor = Color.OrangeRed;
     going_down();
 }

 private void btn_close_Click(object sender, EventArgs e)
 {
     if (isAtGroundFloor  == true)
     {
         door_close_down();
     }
     else if (isAtFirstFloor  == true)
     {
         door_close_up();
     }

 }

 private void btn_open_Click(object sender, EventArgs e)
 {
     if (isAtGroundFloor  == true)
     {
         door_open_down();
     }
     else if (isAtFirstFloor  == true)
     {
         door_open_up();
     }

 }
```



```
private void btn_alarm_Click(object sender, EventArgs e)
{
    btn_alarm.BackColor = Color.Green;
    reader.Speak("Emergency Stop. Please exit the lift carefully.");
    insertdata("Emergency Stop!");
    timer_lift_down.Enabled = false;
    timer_lift_up.Enabled = false;
    timer_door_open_down.Enabled = true;
    timer_door_open_up.Enabled = true;
    display_panel.Image = global::ElevatorControl.Properties.Resources.alarmbellbutton;
    display_top.Image = global::ElevatorControl.Properties.Resources.alarmbellbutton;
    display_bottom.Image = global::ElevatorControl.Properties.Resources.alarmbellbutton;

}



// Database

//Database Variables and instantiations
private bool filled;
public DataSet ds = new DataSet();



private void btn_displaylog_Click(object sender, EventArgs e)
{
    try
    {
        string dbconnection = "Server=BAJRA;Database=lift;Trusted_Connection=True;";
        string dbcommand = "SELECT * FROM Actions;";

        using (SqlConnection conn = new SqlConnection(dbconnection))
        {
            SqlCommand comm = new SqlCommand(dbcommand, conn);
            SqlDataAdapter adapter = new SqlDataAdapter(comm);

            conn.Open();
            while (filled == false)
            {
                adapter.Fill(ds);
                filled = true;
            }
        }
    }
    catch (Exception)
    {
        MessageBox.Show("Cannot open connection!");
        string message = "Error in connection to datasource";
        string caption = "Error";
        MessageBoxButtons buttons = MessageBoxButtons.OK;
        MessageBox.Show(message, caption, buttons);
    }

    database_listbox.Items.Clear();
    foreach (DataRow row in ds.Tables[0].Rows)
    {
        // Formatting date
        string date = Convert.ToDateTime(row["Date"]).ToShortDateString();

        // Formatting time to 12-hour format with AM/PM
        string time = DateTime.TryParse(row["Time"].ToString(), out DateTime parsedTime)
                    ? parsedTime.ToString("hh:mm:ss tt") // 12-hour format
                    : row["Time"].ToString();

        // Add formatted string to the listbox
        database_listbox.Items.Add($"{date}\t\t{time}\t\t{row["Action"]}");
    }
}
```



```
        private void insertdata(string action)
        {
            string dbconnection = "Server=BAJRA;Database=lift;Trusted_Connection=True;";
            string dbcommand = "INSERT INTO Actions (Date, Time, Action) VALUES (@date, @time, @action)";

            using (SqlConnection conn_db = new SqlConnection(dbconnection))
            {
                SqlCommand comm_insert = new SqlCommand(dbcommand, conn_db);
                comm_insert.Parameters.AddWithValue("@date", DateTime.Now.Date);  // Passing only the date part
                comm_insert.Parameters.AddWithValue("@time", DateTime.Now.TimeOfDay);  // Passing only the time part
                comm_insert.Parameters.AddWithValue("@action", action);

                conn_db.Open();
                comm_insert.ExecuteNonQuery();
            }

            database_listbox.Items.Add(DateTime.Now.ToShortDateString() + "\t\t" + DateTime.Now.ToLongTimeString() + "\t\t" + action);
        }


        private void btn_clearlog_Click(object sender, EventArgs e)
        {
            database_listbox.Items.Clear();
        }

        private void btn_exit_Click(object sender, EventArgs e)
        {
            Close();
        }
    }
}
```