**Unit code: CIS016-2**

**OBJECT ORIENTED PROGRAMMING AND SOFTWARE ENGINEERING**

**Assignment 1: Control an Elevator – A C# Project**

**Testing Report**

**Student name: Raimonds Buls**

**Student ID: 1912944**

Table of contents

|  |
| --- |
| Aim 3 |
| Objectives 3 |
| Task Description 3 |
| Design 4 |
| Program Code 4 |
| Testing 9 |
| Overview 11 |
| Marking Matrix with Self-Assessment table 11 |

Aim

This assignment aimed to put into practice knowledge gained in C sharp programming skills to solve real-life problems. It is essential to get used to other programming languages and advance in self-development. Enhancing in other software development languages will increase our capabilities to excel and match the requirements of the job market nowadays.

Objectives

The project requires to design and program two-floor elevator object orientated software application for company needs. The elevator contains an elevator room with a control panel, which has request buttons and display-window, elevator shaft, and request buttons with displays outside on each floor. The elevator doors should open on the arrival of the corresponding floor and close on departing.

Task Description

The project requires a software application to contain GUI (Graphical user interface). GUI contains two requested buttons for each floor, a control panel with two buttons and display window, display areas on each floor - for elevator status representation, and a log button, which will trigger historical information of elevator operations to be shown.

The program should respond to events activated on GUI. For example, when the button on the first floor has been pressed, the elevator should appear on the corresponding floor. The same applies to other buttons and display windows with elevator status.

The program should also include an animation implementation to show the visualization of elevator movement and doors opening/closing events.

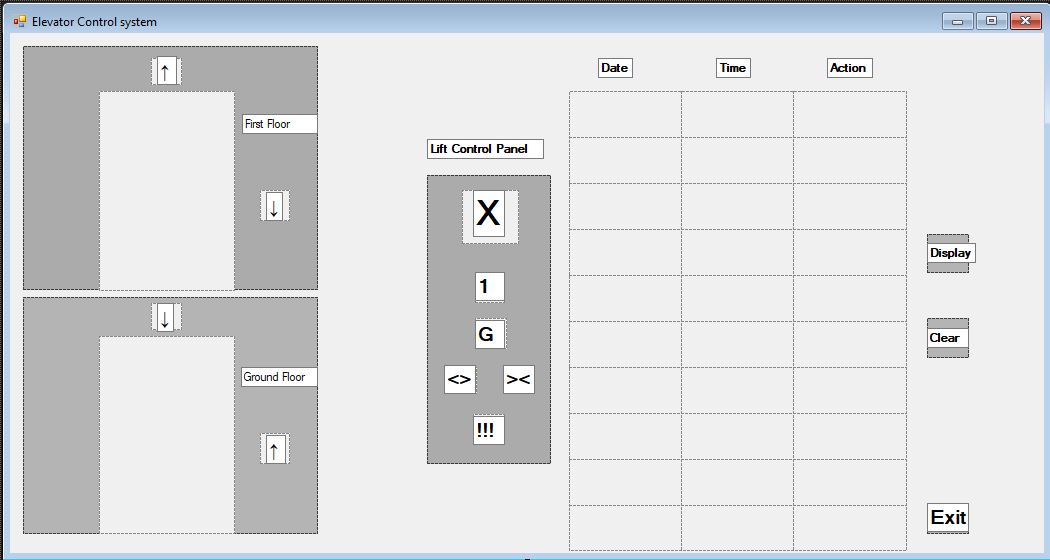
A log of elevator operations with time and actions stored in a database.

To optimize the programs development described above additional tasks should be included:

* 1 Optimize the portability by using relative path instead of absolute path
* 2 Optimize the maintainability by avoiding any duplication among the event handlers over the database related functions
* 3 Optimize the robustness by eliminating logical errors and handling exception
* 4 Optimize the efficiency of GUI by implementing multiple tasks concurrently via BackgroundWorker
* 5 The elevator might have more than two floors, apply the “state design pattern” in the program to accommodate future changes of the requirement. The state is expected to be dispatched dynamically rather than determined by if/switch

Design

Starting point of the project was skeleton of GUI shown below:

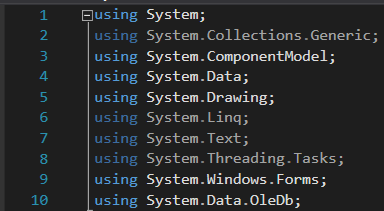


Core point here was to keep graphical user interface simple and intuitive to understand. That helped in coding part and made navigation easier. However, during code programming part some components were changed.

Program code

The program is separated in parts and will be explained below:

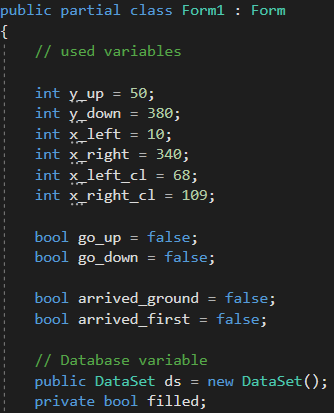




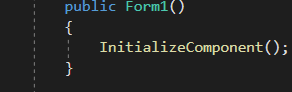
This part of code add references to the libraries program will use



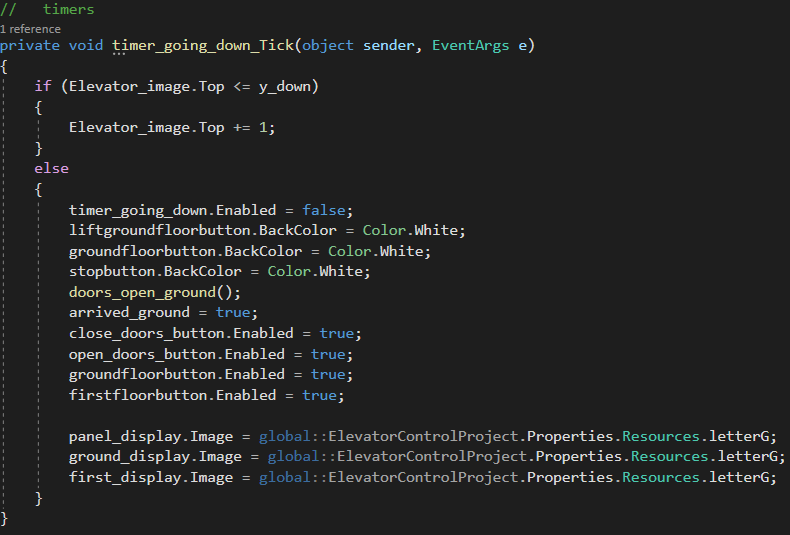
The part of code above will allow program to connect to database.



Code here includes variables used in program and database

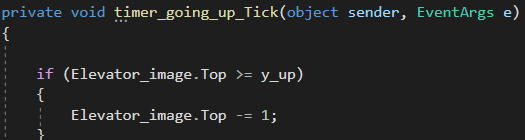
Major parts in code were commented, this helps to identify program code, and in a case for future expand navigate to directly to the part which is in need of changes.

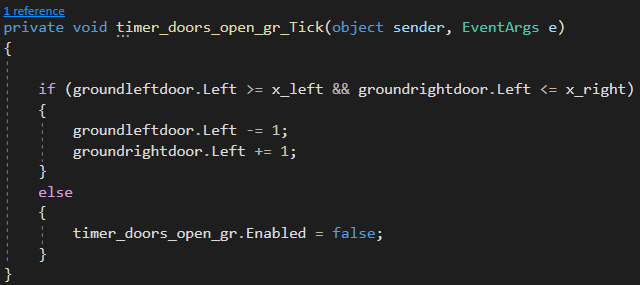
Function below calls initializeComponent(); which builds up user interface for program.



This function is used each time the timer interval is used and performs an animation of the elevator going down. When the elevator is on the ground floor than timer stops and other function to open doors is called. Elevator status is part of function and updates all three displays with ground floor status image on destination reach.

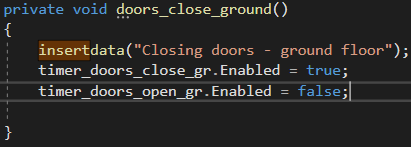
Also, when a button is being pressed color is being changed. That’s an extra feature adding the part that we can see in real life.



The function here by structure is identical to the one above, in this case, the elevator going up. Variables defined before as y\_down and y\_up allow the program to identify stop points for timer functions. As each object in GUI has its coordinates on the x and y-axis, we able to move these objects in a direction we need.

This function opens doors on the ground floor by being called each time interval. Here I used another set of variables to set up stop points for doors opening/closing animations.  

Same method was used in other functions.



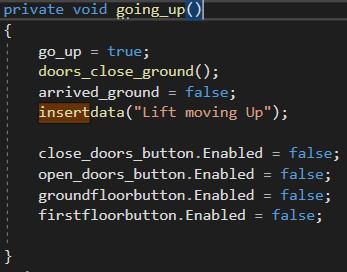
This function enables the timer on the ground floor to close doors there. It also contains insertdata() function. Which will add a string of information written in “ “ when the function was executed. This will be used in the database as information to represent elevator status and actions.







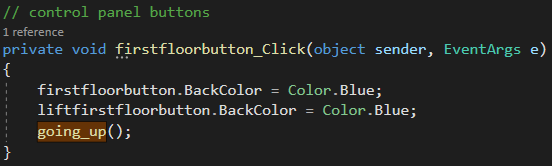
These functions operate in same way.



This function calls doors\_close\_ground() when elevator going up and insertdata().

Also, certain buttons are disabled here, except for the emergency stop one.

Same method here.



When the first flor button is being clicked this function is used as it’s the event handler.

The color of the button pressed is changed, also going\_up() method is called.

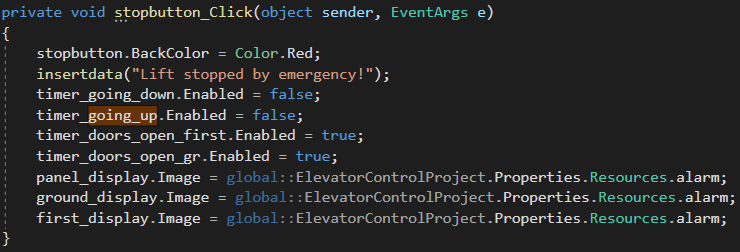


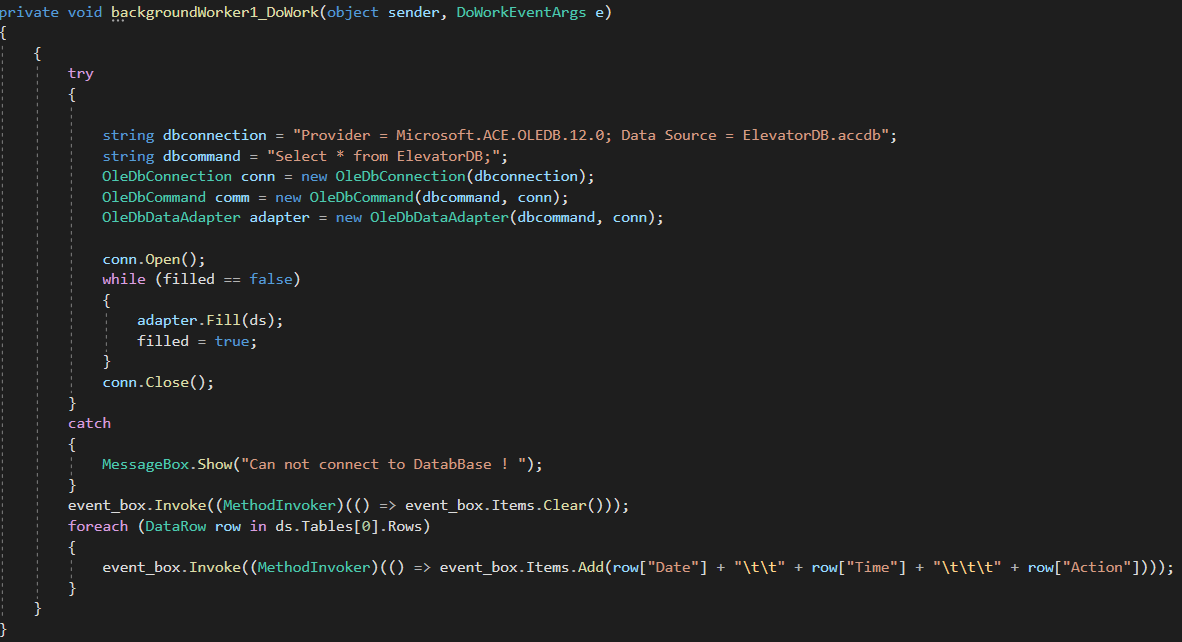






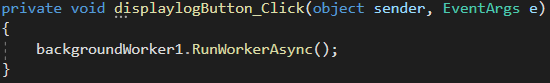
Other buttons work in similar way.

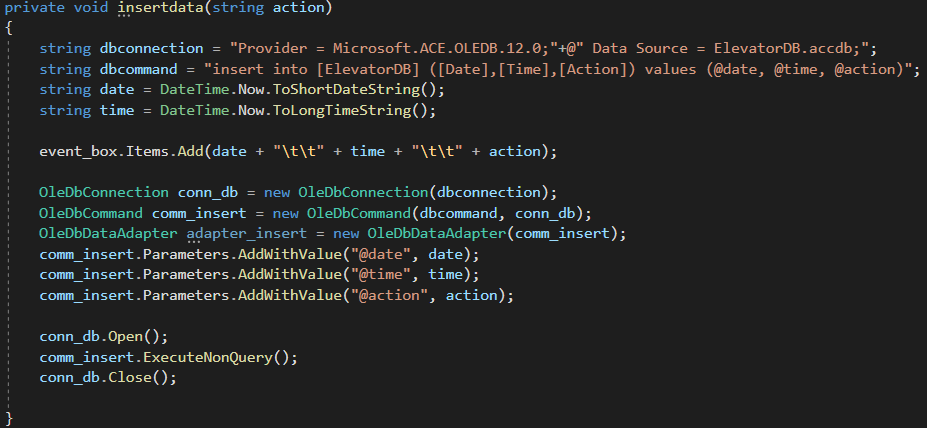


This function is used to stop the elevator, open doors on both floors, and change display status on displays. Insertdata() function is used here to pass information to the database later on. 

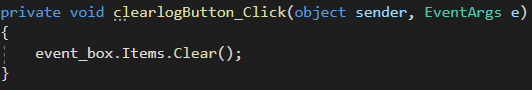
Taking in moderation, that database over time will be loaded with logs of elevator status and going to contain a lot of data it’s moved in a separate thread with this function. In other thread function connects to the database, if the connection was an unsuccessful message with “Cannot connect to DataBase! “will appear. If a connection was established then the function extracts database information and displays in event\_box. Each elevator event-triggered contains a Date, Time, and Action description. MS Access database was used, I already had experience with SQL but wanted to go with something different. 

By adding a database directly in the program folder /bin/debug, the relative path was achieved instead absolute one.



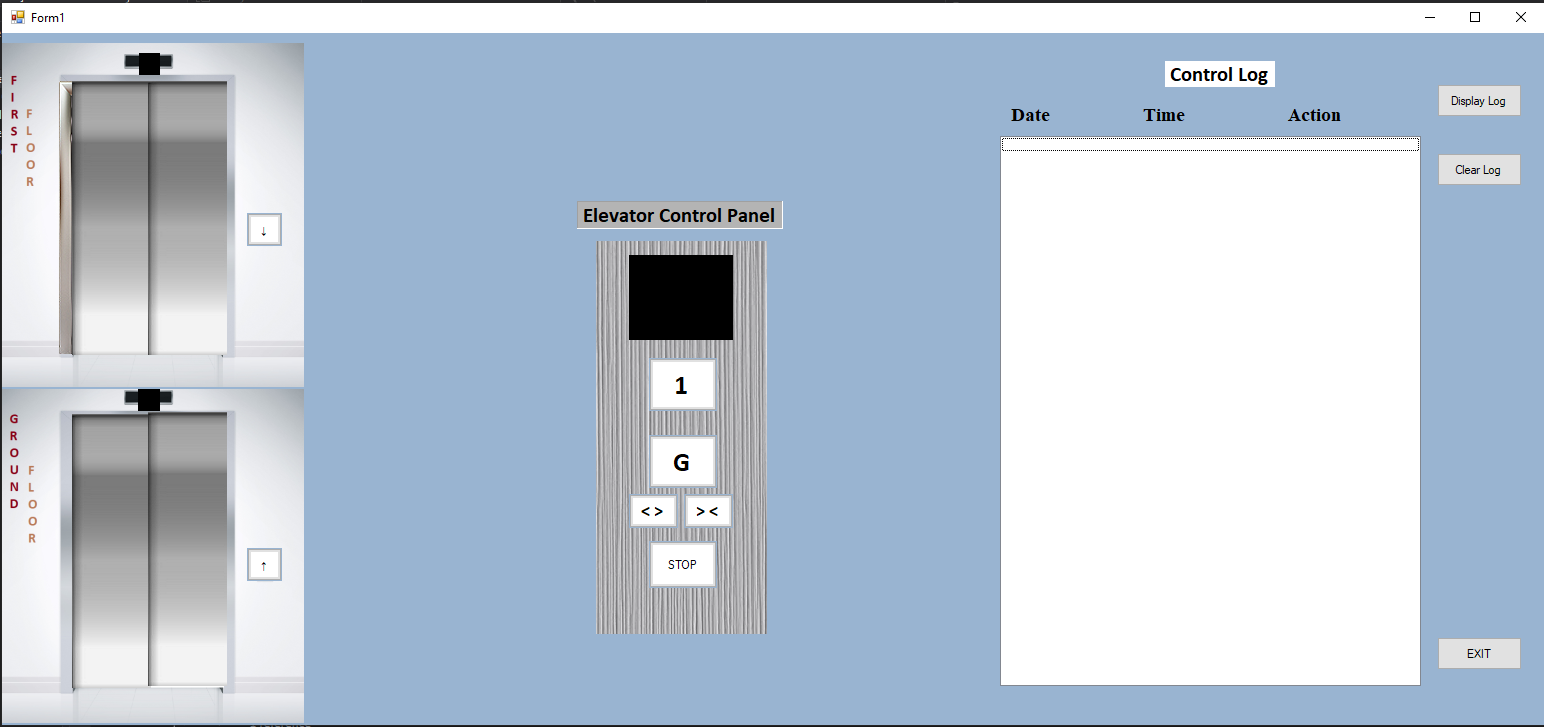
This function will display database logs on click, but since that information will be running in a separate thread – backgroundWorker1.RunWorkerAsync() is called.

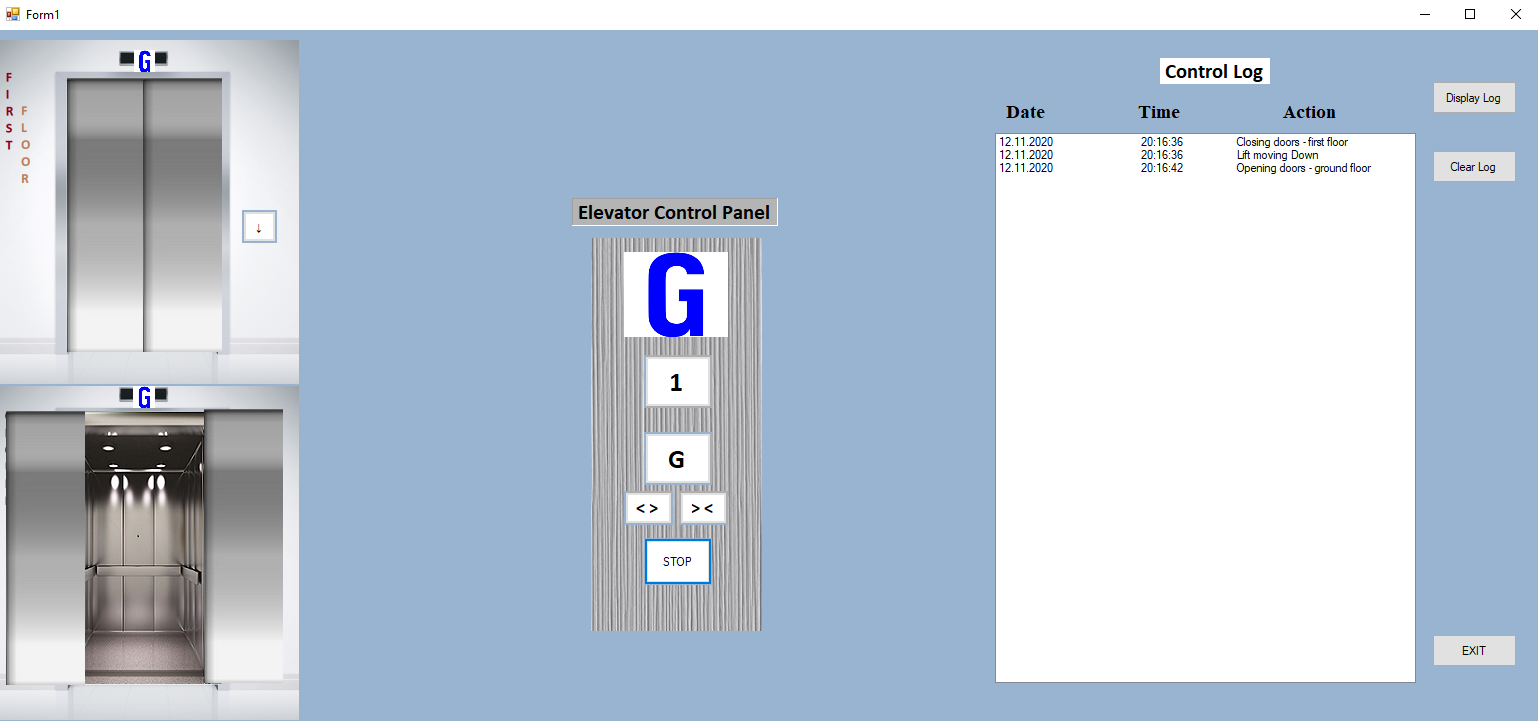
This function connects to the database, inserts data of elevator action events in the database, and saves it.

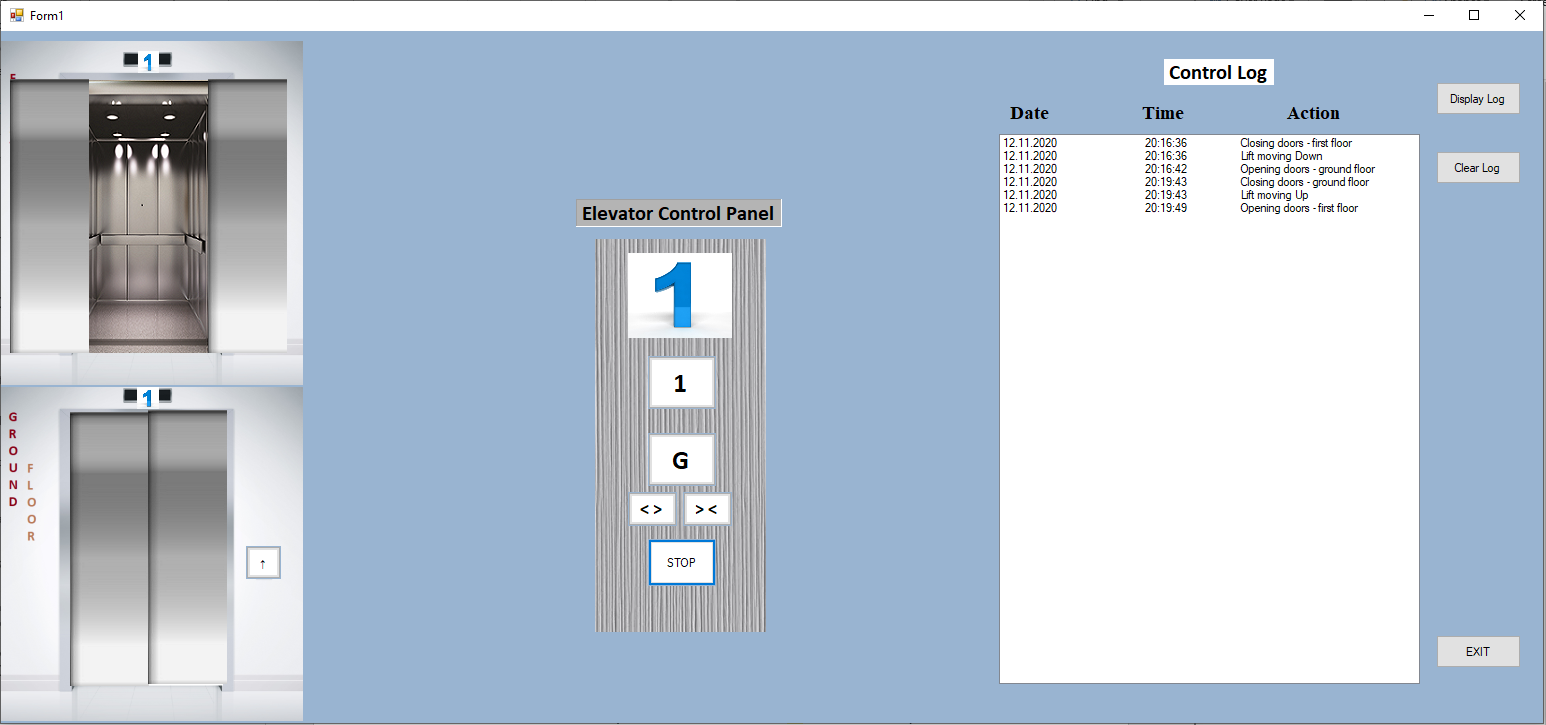


This function hides database logs from the display when the button is clicked.

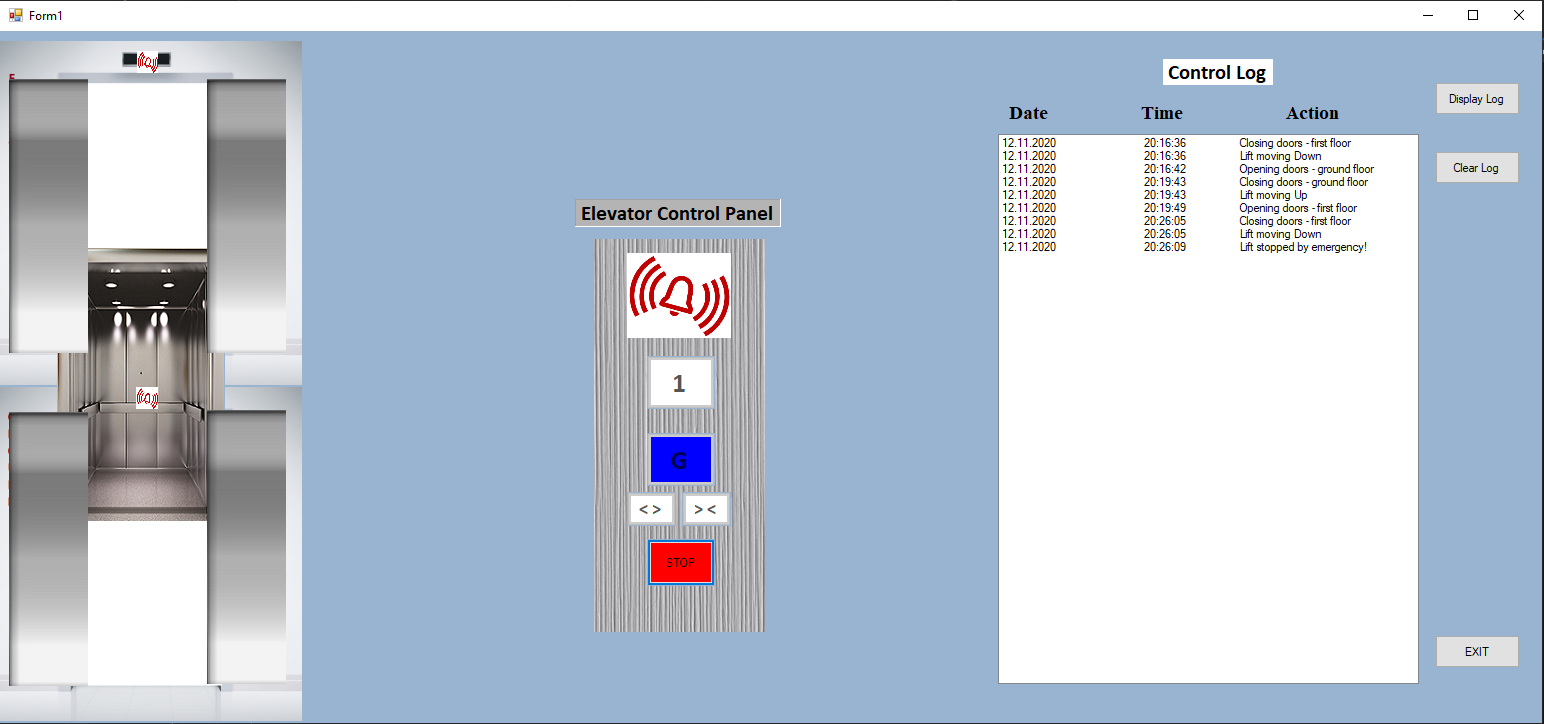
Testing

Final version of the program, when launched: 



When G (ground floor) button is pressed, the elevator went down. When the destination is reached doors opening and display windows show what floor the elevator is on. Database recent action activity – with date, time, and action information.

After the first-floor button was pressed, the elevator went up. Before movement took a place, ground floor doors were closed. After reaching the destination on the first floor, doors were opened, display windows were updated and additional information was stored in the database.



The ground floor button was pressed again, however, in this case, the emergency stop was pressed as well. The elevator stopped his movement in the middle of the path, doors on both floors were opened, display windows were updated with the elevator new status, and information was stored in the database.

Overview

Core requirements in the program were achieved. Software application able to perform actions in the task description, it also contains GUI, animations, and database. It was an interesting experience going from skeleton design to the final stage of the program, where multiple problems were encountered and required different solutions. To resolve them and find the proper approach I had to go back to lecture slides and what we were doing on practical exercises. Worth to mention, to advance and expand programming skills extra information has been taken from other sources as well, including youtube programming videos and some interesting solutions on stack overflow. As an example, by implementing a multi-thread method, event\_box caused the error, as it was part of the form in another thread. To solve that issue, I found the information to use the Invoke method. On the other hand, some advanced methods to make the system more robust were not designed. For instance, using a disconnected model (Data adapters Update() method instead ExecureNonQuery()) or system is still based on if-else statements instead of state patterns, which will affect future changes.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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 | Page 4 | 10 |
| Skeleton of event handlers in place for all buttons | 10 | yes | Page 4 | 10 |
| **Task 2** | All event handlers are functional | 10 | yes | Page 4-9 | 10 |
| **Task 3** | Database (DB) is designed and can be connected | 5 | yes | Page 8-9 | 5 |
| Log Information can be retrieved from DB and displayed in the GUI | 5 | yes | Page 8 | 5 |
| When the log button is pressed, log information is sent to and stored in the DB | 5 | yes | Page 8 | 5 |
| Use the disconnected model rather than connected model (Data source is updated via DataAdapters Update() method instead of ExecuteNonQuery() method) | 5 | yes | Page 11 | 0 |
| **Task 4** | Events described in Task 2 animated using delegation and timer | 10 | yes | Page 4-9 | 8 |
| **Task 5** | Using relative path instead of absolute path | 5 | yes | Page 8 | 5 |
| Avoiding any duplication among the event handlers over the database related functions | 5 | yes | Page 8 | 4 |
| Eliminating logical errors and handling exceptions with try and catch | 5 | yes | Page 8 | 3 |
| Optimise the efficiency of GUI by implementing multiple tasks concurrently via BackgroundWorker | 5 | yes | Page 8 | 4 |
| Use state patterns instead of if-else statements to accommodate future changes of the requirement | 10 | yes | Page 11 | 0 |
| **Task 6** | Test report | 10 | yes |  | 7 |
| **Total** |  | 100 |  |  | 76 |