|  |
| --- |
|  |
| Assignment One Report |
|  |

|  |
| --- |
| 1807956 |

Table of Contents

[Self-assessment 2](#_Toc24575261)

[Project structure overview 3](#_Toc24575262)

[Task 1 3](#_Toc24575263)

[GUI 3](#_Toc24575264)

[Event handlers skeleton 3](#_Toc24575265)

[Task 2 3](#_Toc24575266)

[Functionality of event handlers 3](#_Toc24575267)

[Task 3 3](#_Toc24575268)

[Database 3](#_Toc24575269)

[Logging information and retrieving information from the database 3](#_Toc24575270)

[Data source updates 4](#_Toc24575271)

[Task 4 4](#_Toc24575272)

[Animation of the elevator and the doors 4](#_Toc24575273)

[Task 5 4](#_Toc24575274)

[Relative path usage 4](#_Toc24575275)

[Database related duplications 4](#_Toc24575276)

# 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 | Task 1 section |  |
| Skeleton of event handlers in place for all buttons | 10 | Yes | Task 1 section |  |
| **Task 2** | All event handlers are functional | 10 | Yes | Task 2 section |  |
| **Task 3** | Database (DB) is designed and can be connected | 5 | Yes | Task 3 section |  |
| Log Information can be retrieved from DB and displayed in the GUI | 5 | Yes | Task 3 section |  |
| When the log button is pressed, log information is sent to and stored in the DB | 5 | Yes | Task 3 section |  |
| Data source is updated via DataAdapters Update() method instead of ExecuteNonQuery() method | 5 | Yes | Task 3 section |  |
| **Task 4** | Events described in Task 2 animated using delegation and timer | 10 | Yes | Task 4 section |  |
| **Task 5** | Using relative path instead of absolute path | 5 | Yes | Task 5 section |  |
| Avoiding any duplication among the event handlers over the database related functions | 5 | Yes | Task 5 section |  |
| Eliminating logical errors and handling exceptions with try and catch | 5 | Yes | Task 5 |  |
| Optimise the efficiency of GUI by implementing multiple tasks concurrently via BackgroundWorker | 5 | Yes | Task 5 section |  |
| Use state patterns instead of if-else statements to accommodate future changes of the requirement | 10 | Yes | Task 5 section |  |
| **Total** |  | 100 | 100 |  |  |

# Project structure overview

The assignment code is made of two projects in the solution. The first one is named ElevatorUI and contains designed and functional user interface it is also the start project for the application. The second project in the solution is ElevatorCore which is a type of the Class Library. It consists of all the classes responsible for the desired functionality of the elevator, connection with the database and the application settings. To open the solution double click the Elevator.sln file in the main directory.

# Task 1

## GUI

The graphical user interface is implemented inside the ElevatorUI folder. The MainForm files contain the interface of the main window. Inside the ElevatorUI folder, there is a subfolder named Controls that contains the controls that represent the single floor and the elevators panel control.

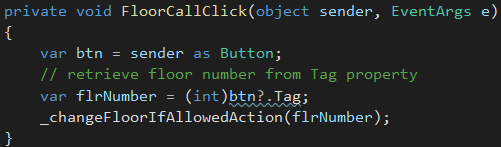
## Event handlers skeleton

All the buttons have assigned event handlers to them. It can be seen in the files containing code for Graphical User Interface.

# Task 2

## The functionality of event handlers

All buttons in the UI have functioning events that can be seen while using the program.



# Task 3

## Database

This project uses SQLite 3 as the database. The designed database is placed in two places. The first one as in per requirements in the assignment brief, in the Bin/Debug folder in the main directory of the project. The file itself is names Elevator.sqlite. The other place is the ElevatorUI folder in the main directory. The file is placed in 2 places in case the bin/Debug folder has been removed. In this case, the database file will be automatically copied to the bin/Debug directory when the project is built.

## Logging information and retrieving information from the database

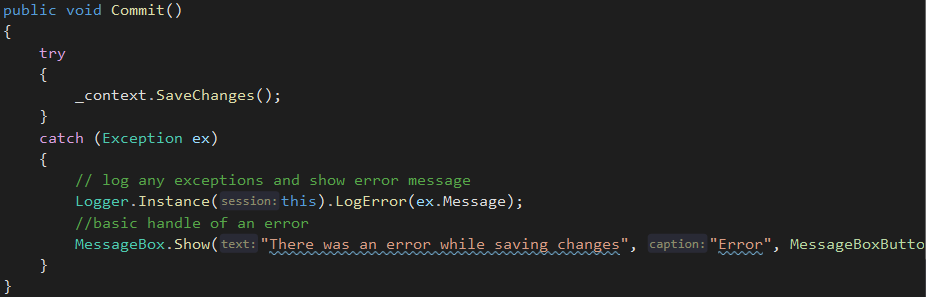
The project utilizes Entity Framework 6 which is an ORM extension to C# language. It allows assigning the database records to the object that contains all properties that are present in the table in the database. Also, it allows to easily add, edit and delete records from the database.

The data from the database is retrieved and loaded into DataGridView control, at the moment when the user presses the “Show Logs” button in the top left corner of the application.

The click on the “Show Logs” button also triggers save of the changes made to the database. The logs are automatically created and added to the dataset while making operations on the elevator. As a result of this, we can see logs with the exact time when the log was created and descriptions what at the time happened to the elevator. In addition when the programs close all the logs are saved automatically.

## Data source updates

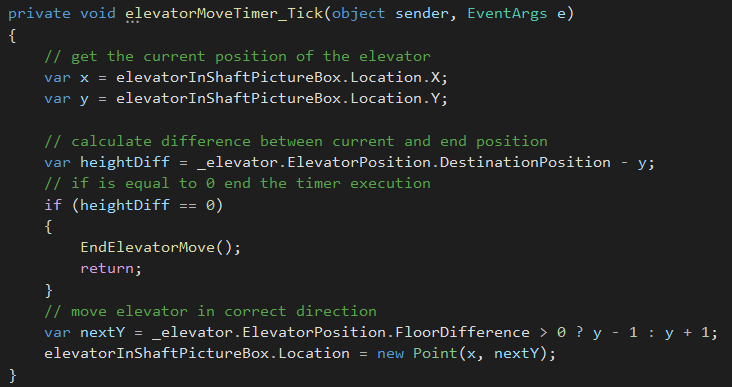
The created database connection utilizes the Repository and Unit Of Work patterns. Also, the use of the Entity Framework utilizes a disconnected database connection mode. Which allows saving many changes using one command. The class that is the concrete implementation of the Unit Of Pattern, named “Session”, contains the “Commit“ method, which will save all the changes to the database. The implementation of this class is in the ElevatorCore/DataAccess/Concrete catalogue.



# Task 4

## Animation of the elevator and the doors

The fact that the elevator in the shaft is animated as well as the doors can be seen while using the application. The function that is responsible for moving the elevator is shown on the screenshot below.



# Task 5

## Relative path usage

The use of a relative path is placed in the App.config file in the ElevatorUI project.

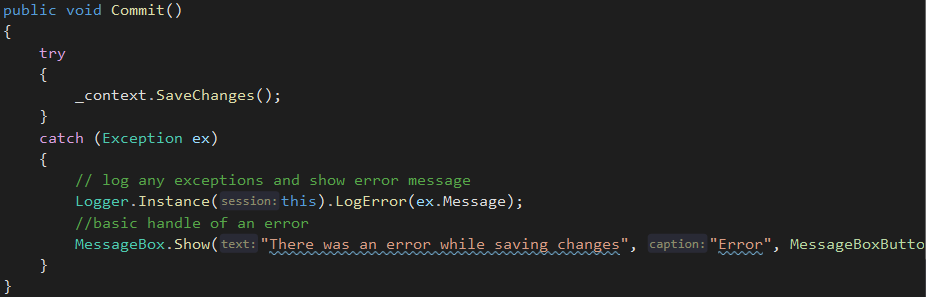


## Database related duplications

All the database related functions are placed in the ElevatorCore project in DataAccess/Concrete folder. They use Unit Of Work and Repository patterns to manage the whole connection process. The object of the Session class is initialized when the user specifies the number of floors he wants to have at the start of the application.

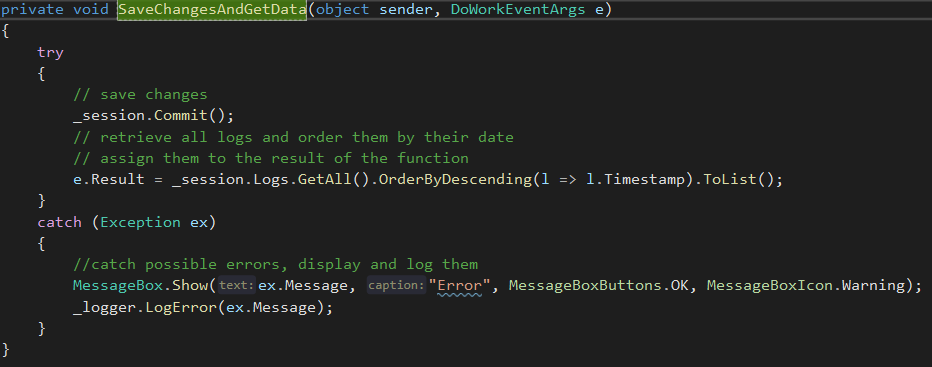
## Error handling

Logical errors are eliminated and possible errors that may happen are handled in try-catch blocks.



## Background worker

Background worker is used to commit changes to the database and retrieve information to the DataGridView. Retrieved and ordered data are assigned to the e.Result value of the function and later set as the data source of DataGridView.



## State patterns

State patterns are used to manage the behaviour of the elevator and its doors. They are implemented in the “ElevatorCore/Elevator/Concrete” folder.

The application uses four states:

* ElevatorDoorsClosingState
* ElevatorDoorsOpeningState
* ElevatorMovingState
* ElevatorStationaryState

The example implementation of one of the states:

