LONDON METROPOLITAN UNIVERSITY

Module Code & Module Title

CS6004NT – Application Development

Assessment Weightage & Type

30% Individual Coursework

Year and Semester

2021-22 Autumn

Student Name: Prashant Nepal

London Met ID: 19033562

College ID: NP05CP4S200019

Assignment Due Date: 3rd January

Assignment Submission Date: 3rd January

Academic Supervisor: Mr. Rabi Rouniyar

I confirm that I understand my coursework needs to be submitted online via Google Classroom under the relevant module page before the deadline for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission, and a mark of zero will be awarded.

Table of Contents

Table of Figures

Table of Tables

1. Introduction

Recreation Center, an amusement companion for most of the people living inside the hustling and bustling city, has been providing outstanding services and offers. Along with the skyrocketed population, the demand for entertainment sources has surged. The number of visitors based upon various categories is enormously increasing too.

Unfortunately, staff and admins cannot conduct the business smoothly due to an outdated paper-based manual system. As a result, a desktop application has been proposed to be developed using **.Net** Framework based on **C#** programming language. And finally, the developed system should allow the admin and the staff to be logged into the system independently. The admin should be able to set the ticket rate in various categories. In contrast, employees should register the visitor's entry and exit time and calculate the total price. The admin and the staff should generate, analyze, and evaluate the daily and weekly reports.

C-Sharp, a programming language, was developed by Microsoft, applicable in .Net Framework for developing a desktop application, web-based application, games, IoT, and many more. C# is a static typing programming language that supports object-oriented paradigm, strong typing, declarative functions, null safety features, etc. On the other hand, the .Net Framework, which includes a large number of class libraries for delivering language interoperability features. The architecture of the .Net Framework consists of the CLR (Common Language Runtime) (for handling threads, garbage collection, exception handling, and type-safety) and the Class Library (that provides datatypes and APIs for managing functionalities like connecting to the database, file handling, etc.). Due to its most prominent features, it has been widely praised and used by developers to develop systems.

Visual Studio 2019, an IDE fully packed with libraries and features, has been implied to develop an application using .Net Framework. It is convenient for coding, debugging, developing, deploying, analyzing, and collaborating the software applications

with rich features (plugins, extensions). It is applicable in various platforms like Windows API, Microsoft Silverlight, etc., and supports 36 different programming languages.

Further moving into the project, .Net Framework based on the C# programming language will develop the desktop application for the Recreation Center. The Visual Studio 2019 IDE will be used for coding, debugging, and analyzing the codes.

2. Operation Instructions to Run the Program

2.1. Opening Project and Starting Application

Initially, the project solution file can be located by navigating to the project folder. After which, it should be opened with Microsoft Visual Studio 2019.

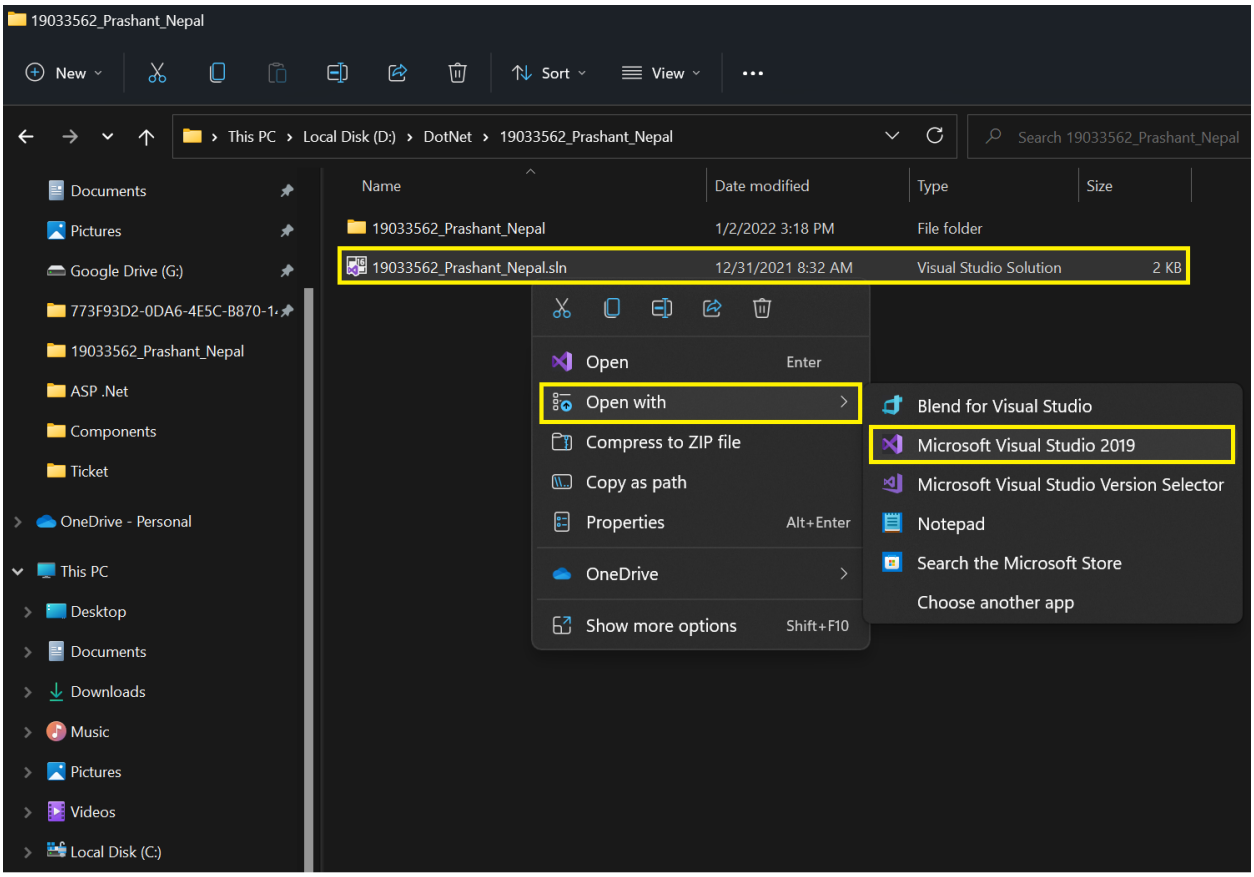

Figure 1: Opening Project in Visual Studio

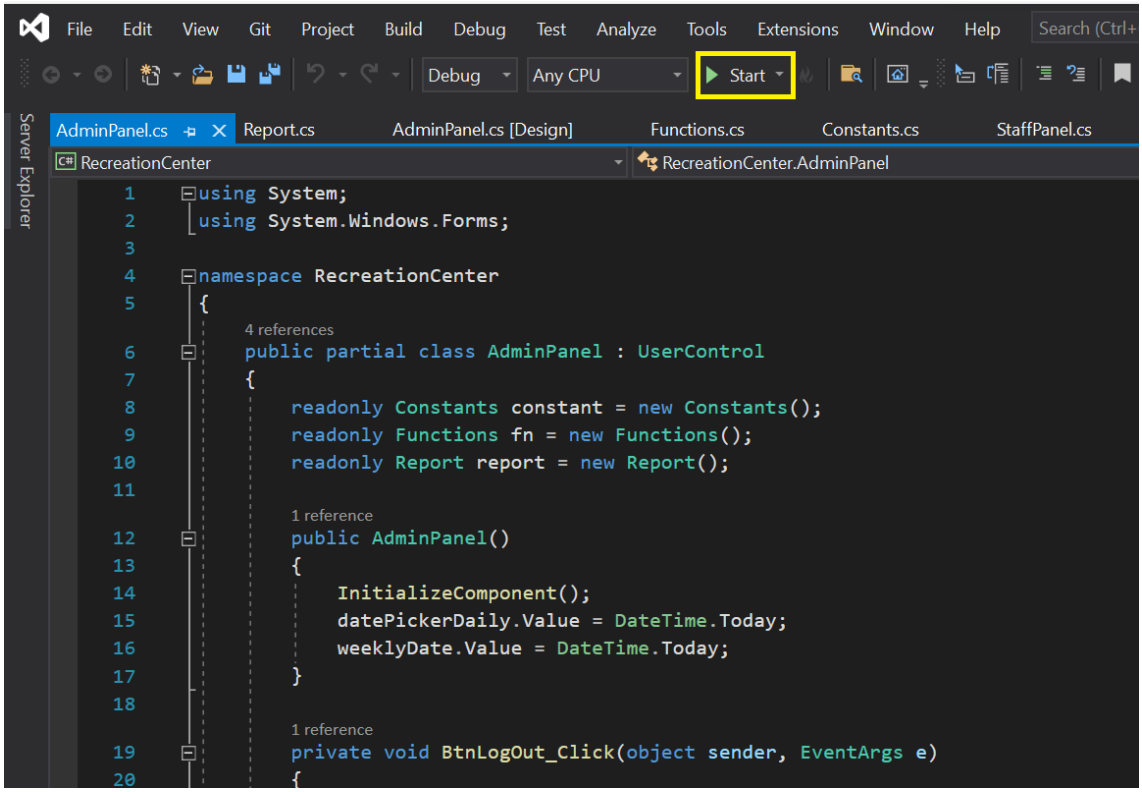

Figure 2: Starting Application

The start button should be clicked from the IDE, which will lunch the project's primary method and fire up the application. At first, the login page will be displayed.

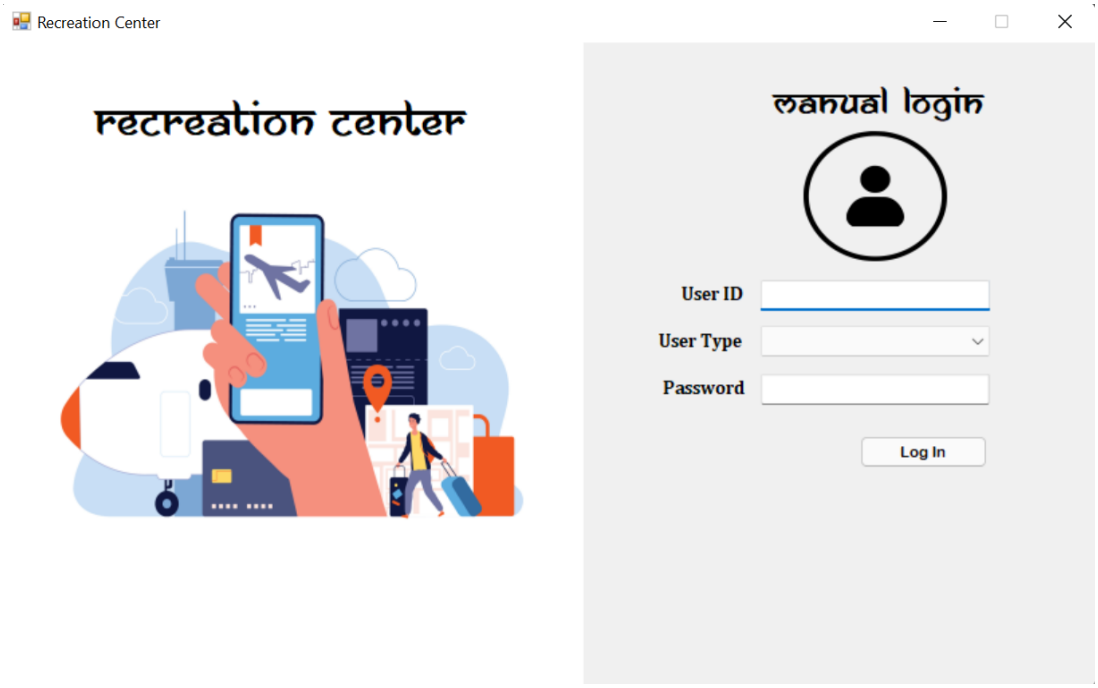

Figure 3: Login Page

2.2. Logging in as an Admin

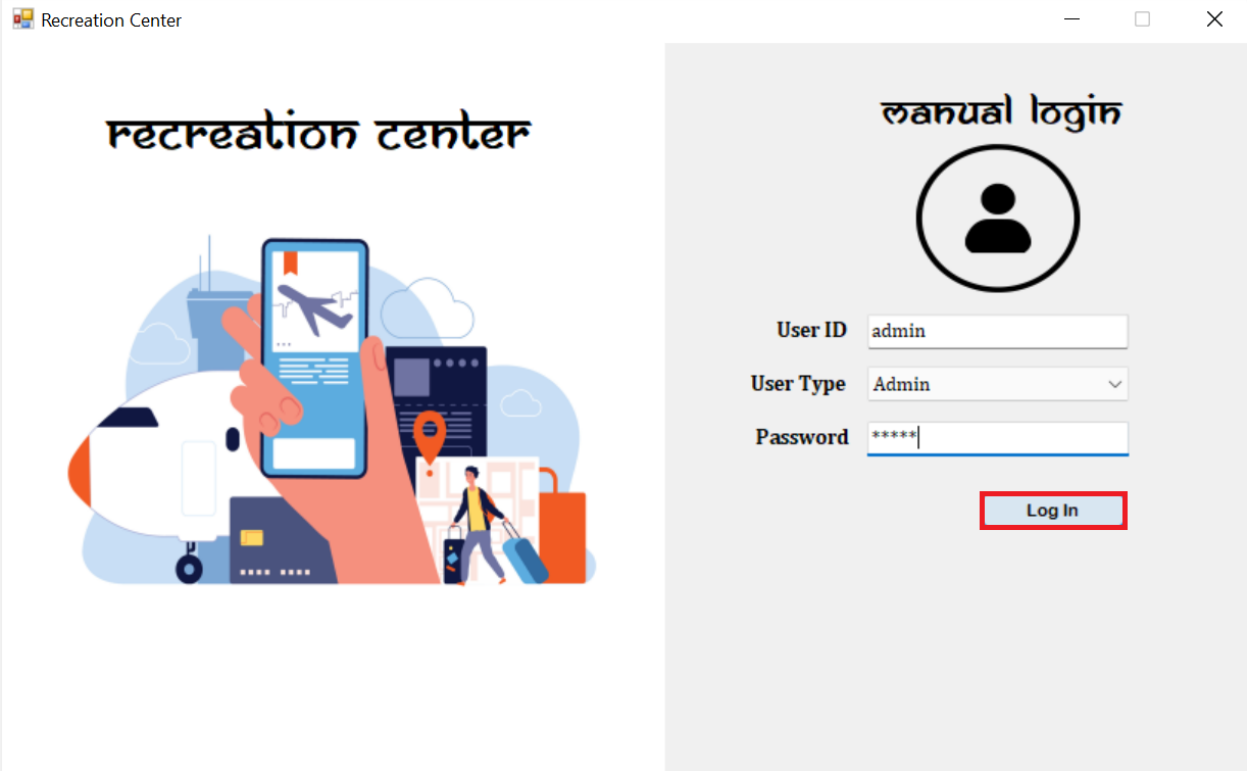

Figure 4: Login as an Admin

The credentials of the Admin should be inserted for opening the admin dashboard. The username and password for Admin are Admin and Admin, respectively.

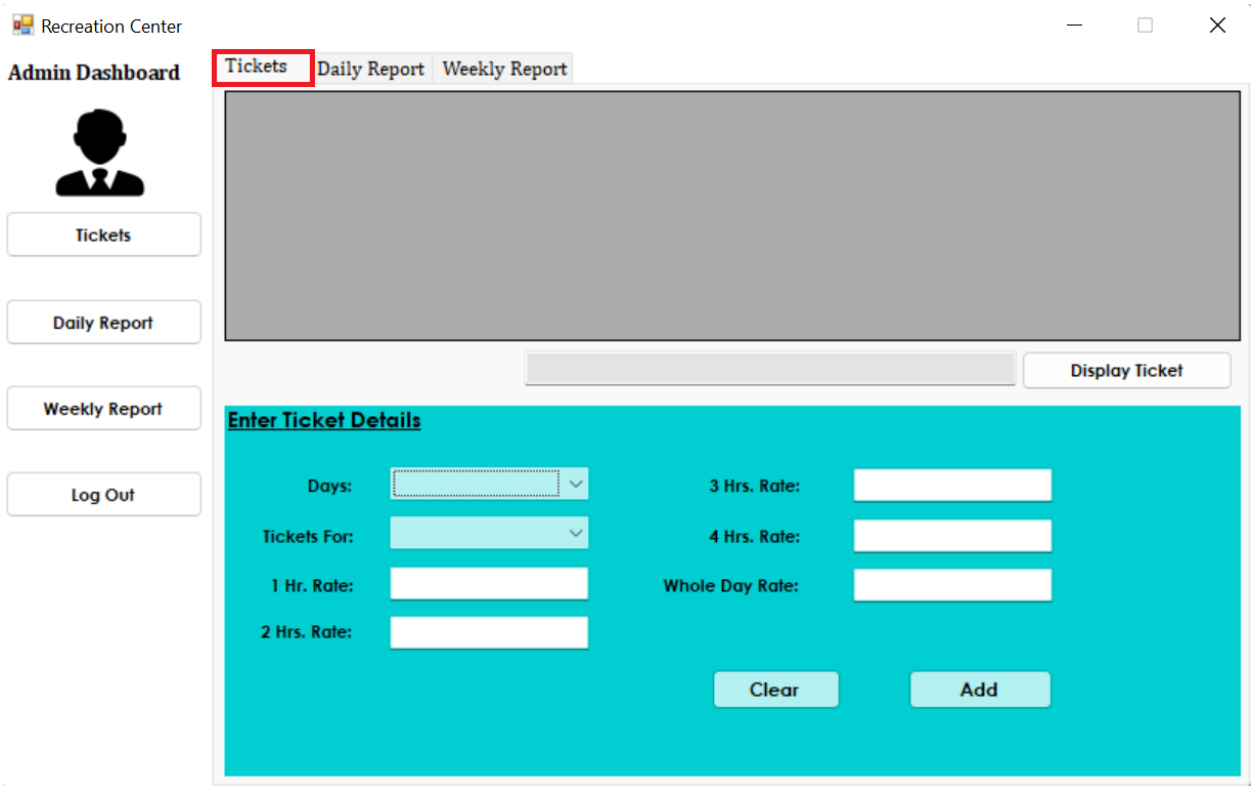

Figure 5: Admin Dashboard

2.3. Adding Ticket Price

The Admin can set the ticket price for the mentioned categories from the admin dashboard.

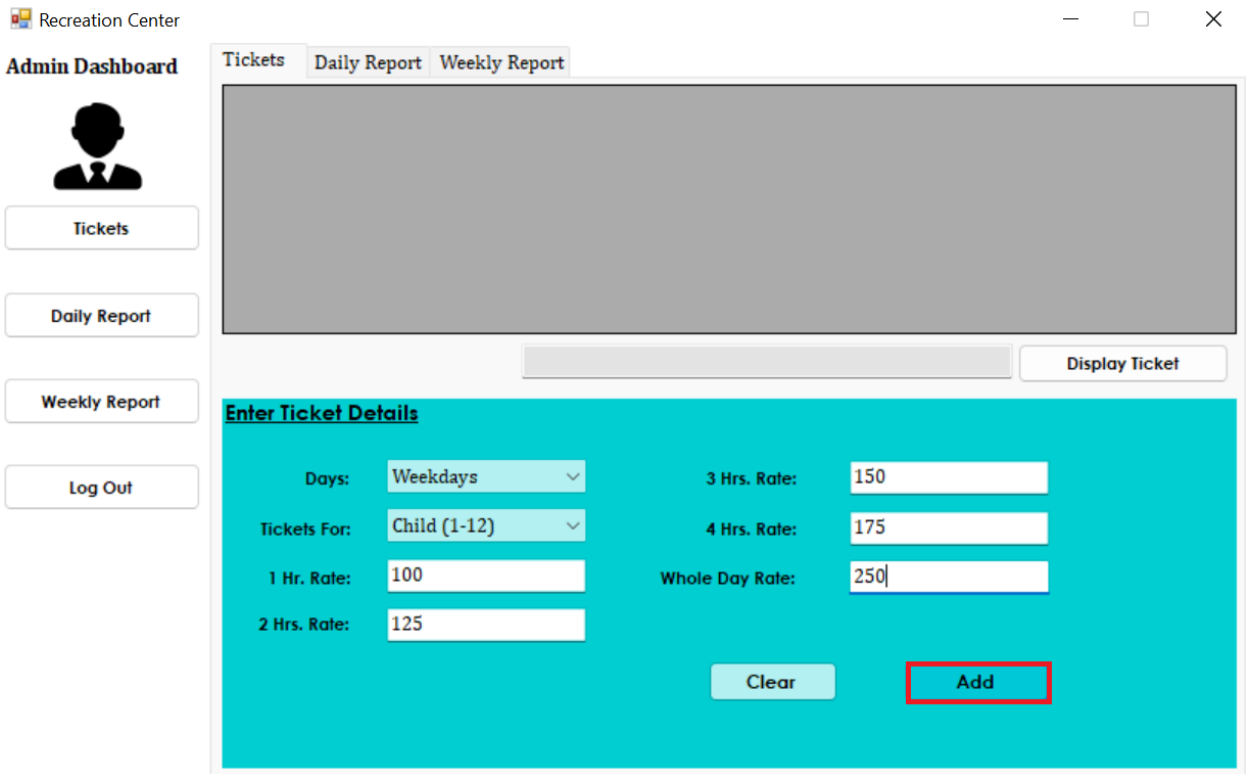

Figure 6: Adding Ticket Prices

2.4. Displaying Ticket Price.

After clicking the 'Display Ticket' Button, the dialog box will be appeared, from where a CSV file of a ticket can be selected to be displayed in the grid view.

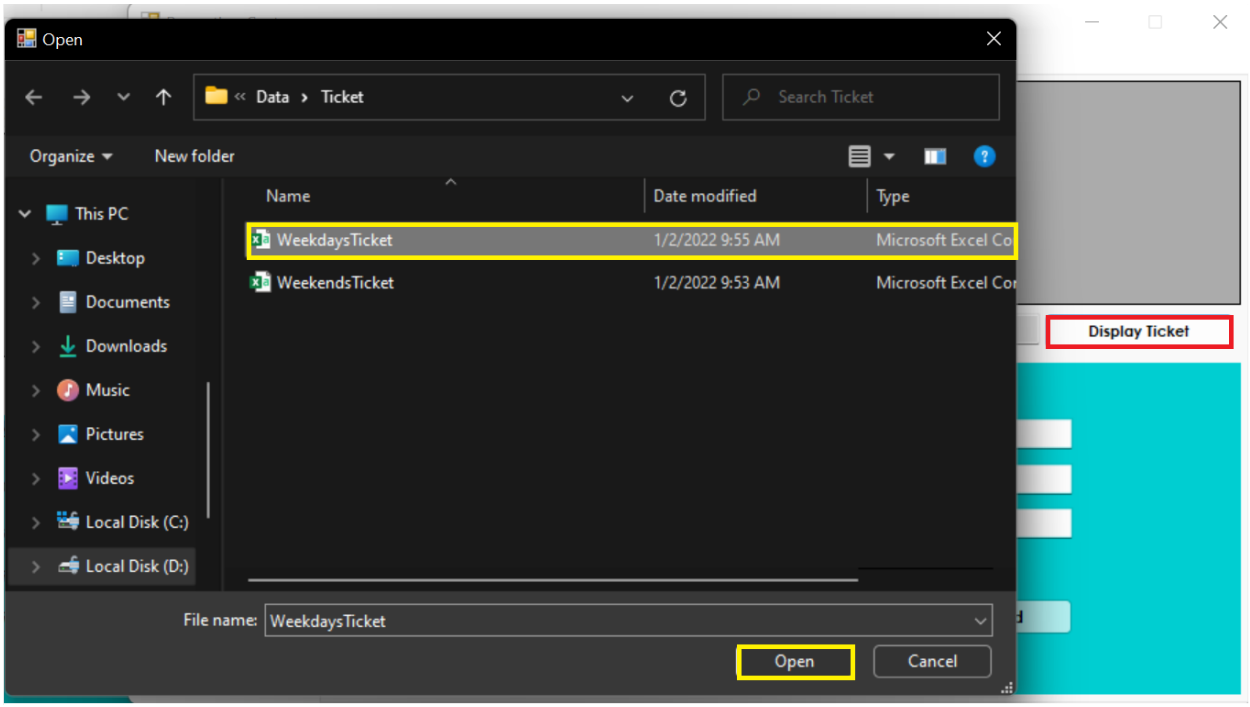

Figure 7: Displaying Ticket Prices

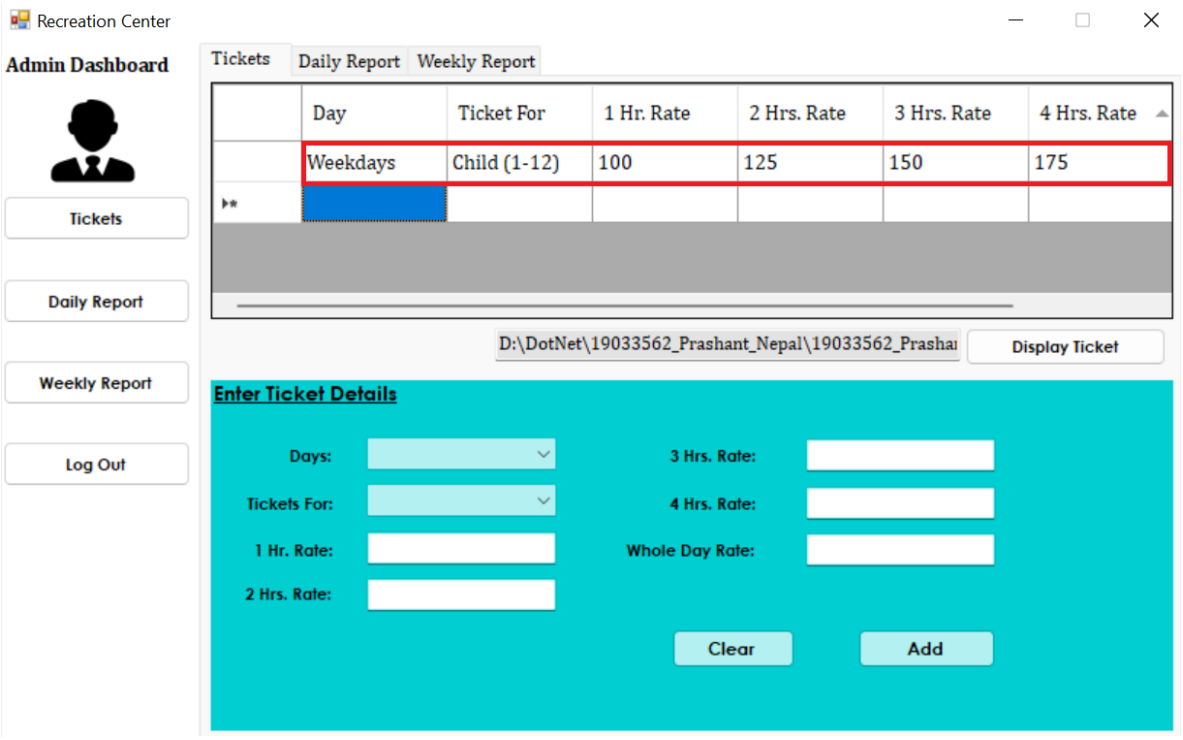

Figure 8: Recently Added Ticket Prices

2.5. Logging in as a staff

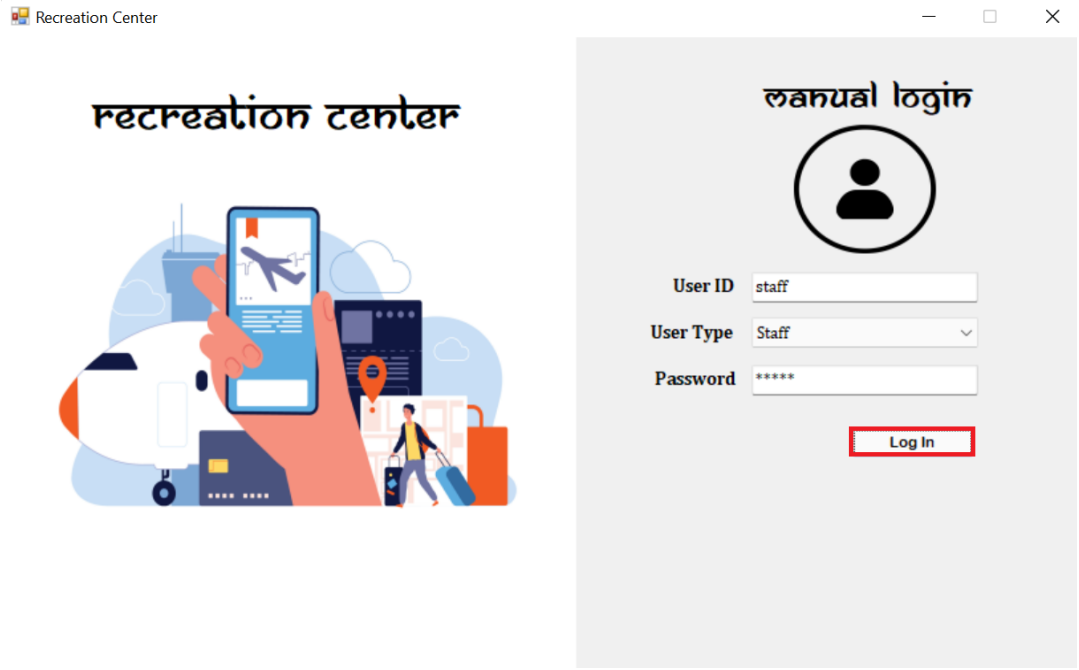

Figure 9: Logging in as a staff

Similarly, the staff credentials should be inserted for opening the staff dashboard. The username and password for the staff are staff and staff, respectively.

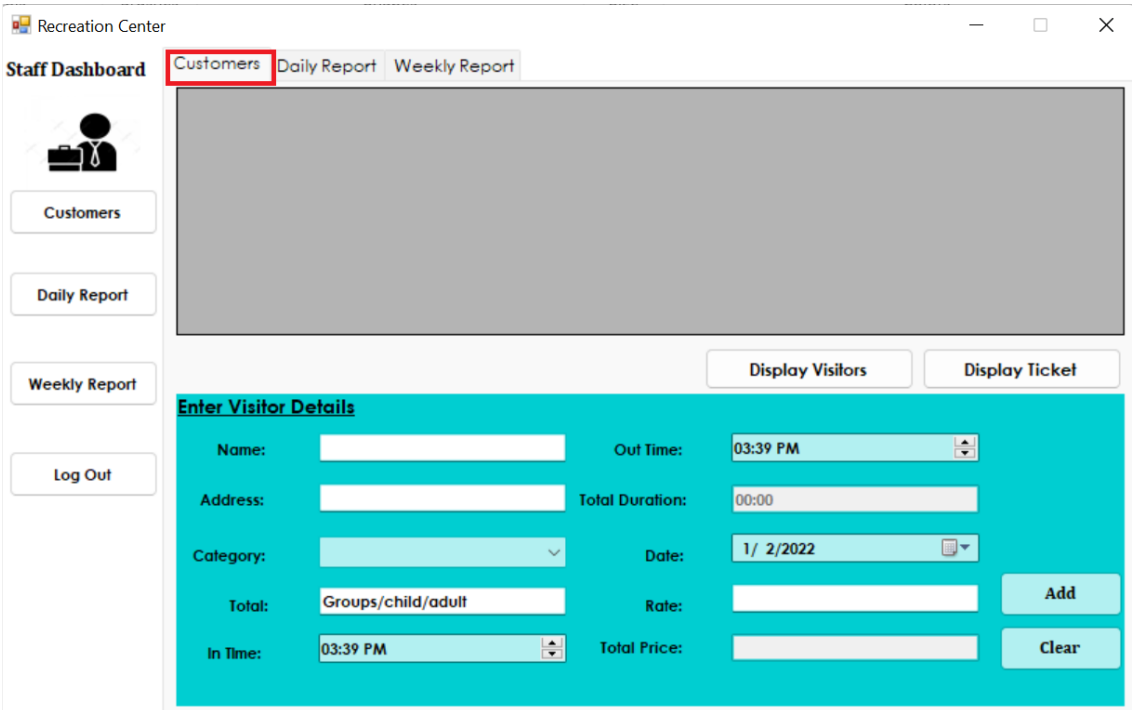

Figure 10: Staff Dashboard

2.6. Displaying Tickets

From the staff dashboard, by clicking the 'Display Ticket' button, the ticket price set by the Admin will be displayed in the grid view. The weekends and weekdays tickets will be automatically determined and depicted by the System.

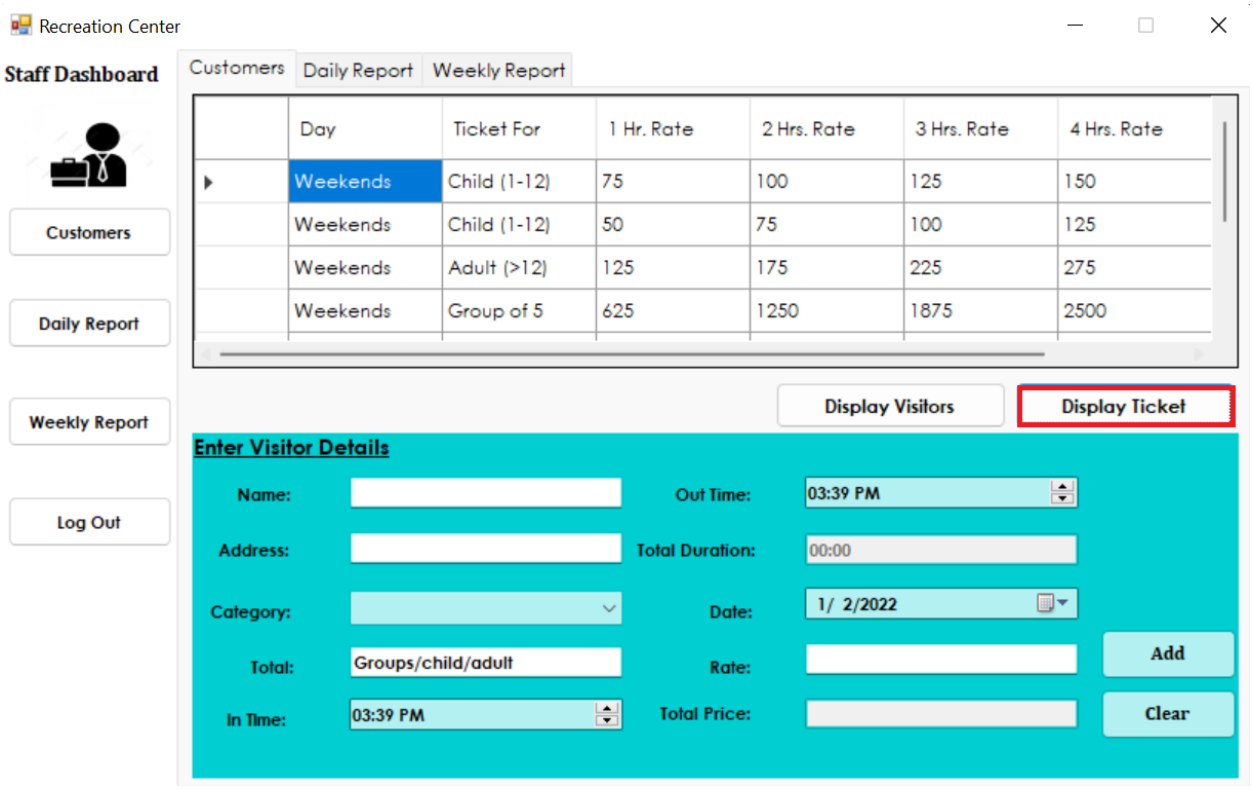

Figure 11: Displaying Tickets

2.7. Inserting Customer Details

Their relevant details should be inserted in the appropriate fields for registering the visitors. The System will automatically set the total duration by measuring entry and exit time. Further, the system will also calculate the total price by determining hours and minutes.

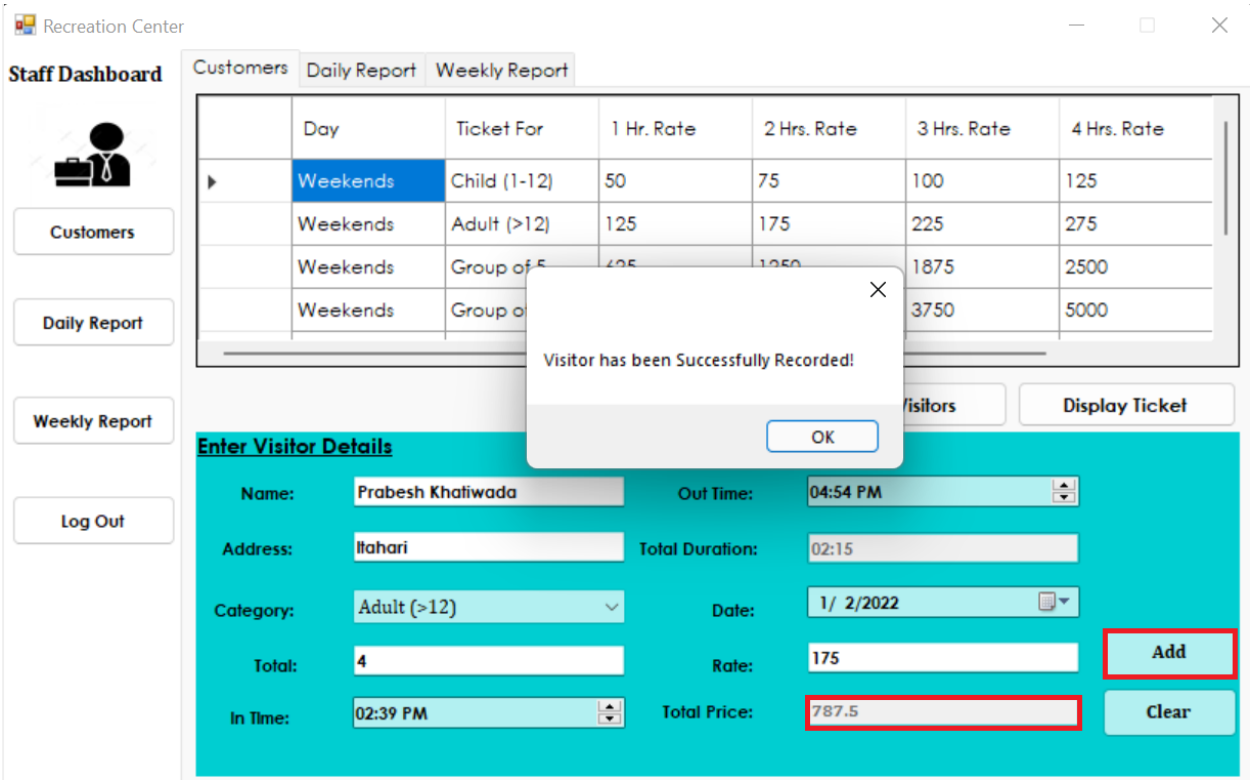

Figure 12: Inserting Customer Details

2.8. Displaying Visitors

The inserted customer details will be portrayed while clicking the 'Display Visitors' button.

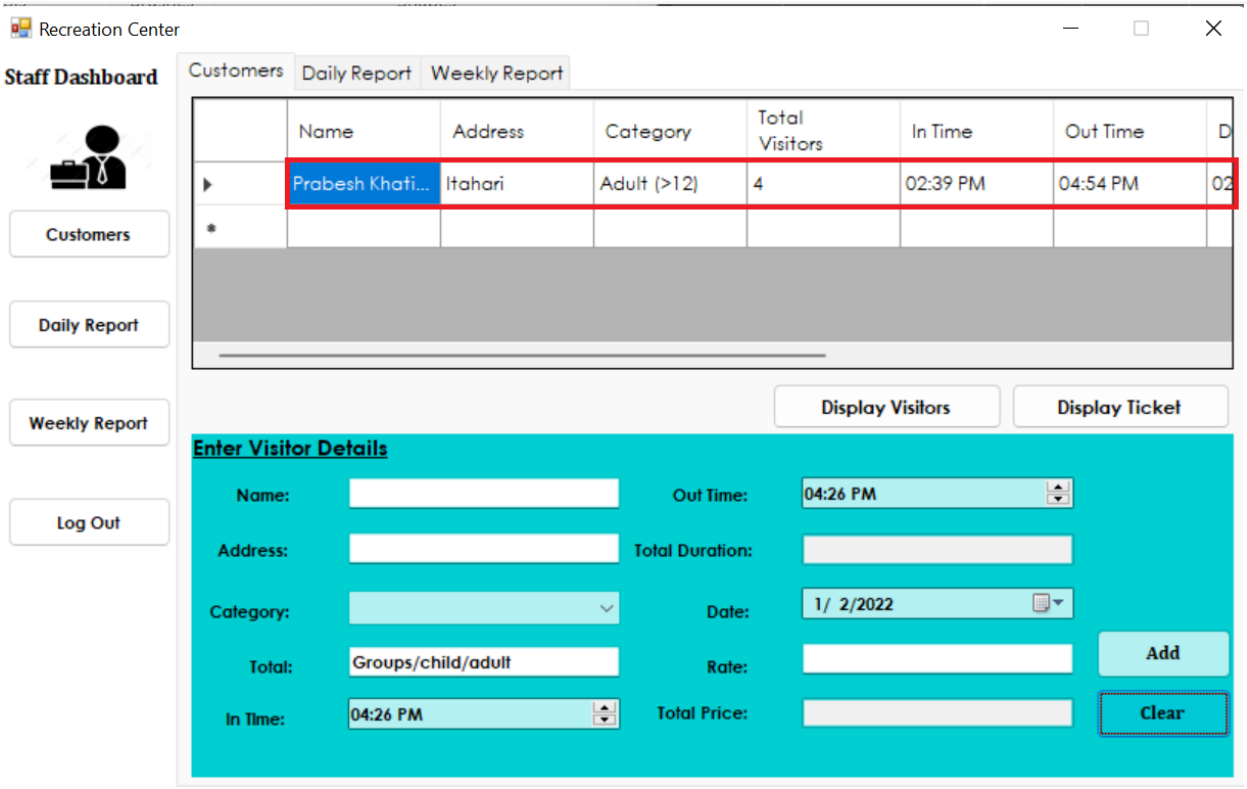

Figure 13: Displaying Recently Added Customers

2.9. Displaying Daily Report

By picking the date from the DateTimePicker, the daily report will be displayed in the grid view and the chart. Further, the 'View All Details' button displays overall details of customers of a specific date.

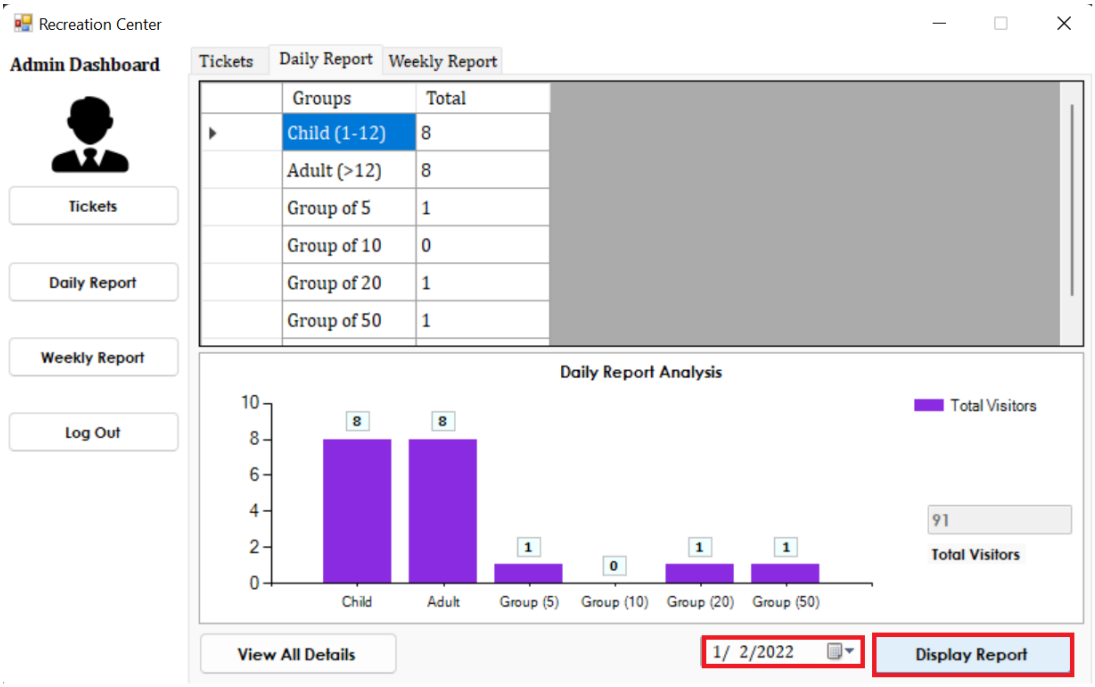

Figure 14: Displaying Daily Report

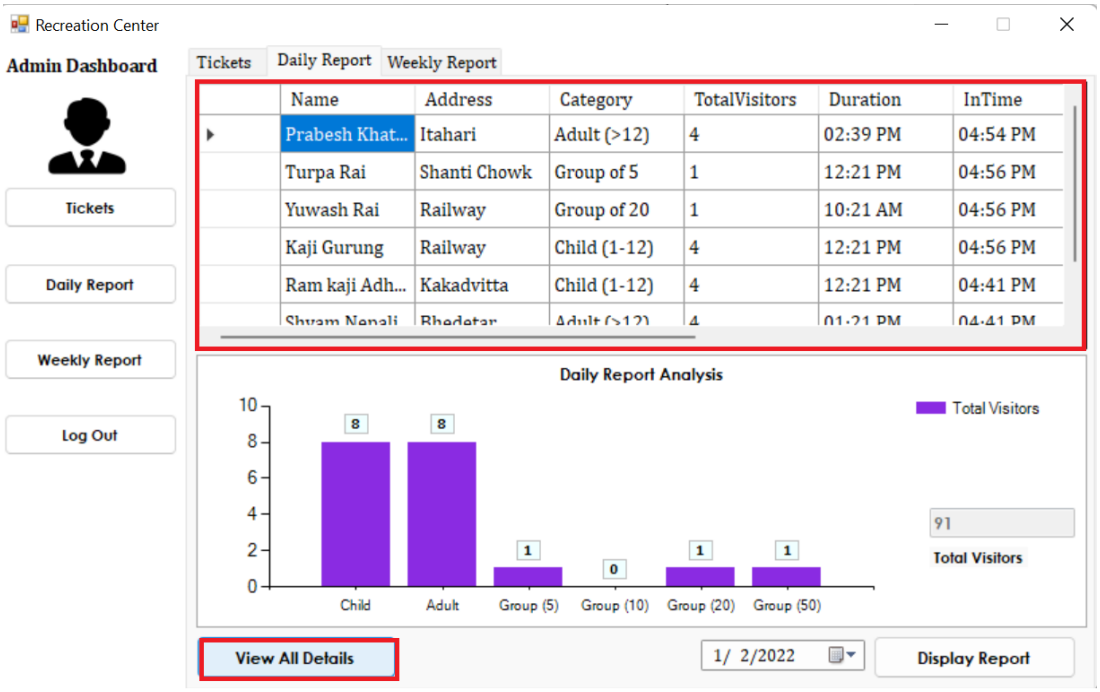

Figure 15: Displaying overall details of visitors

2.10. Displaying Weekly Report

The weekly report tab will display the overall visitors' count and earnings for a week. The weekly report of the selected date from the date time picker will be highlighted.

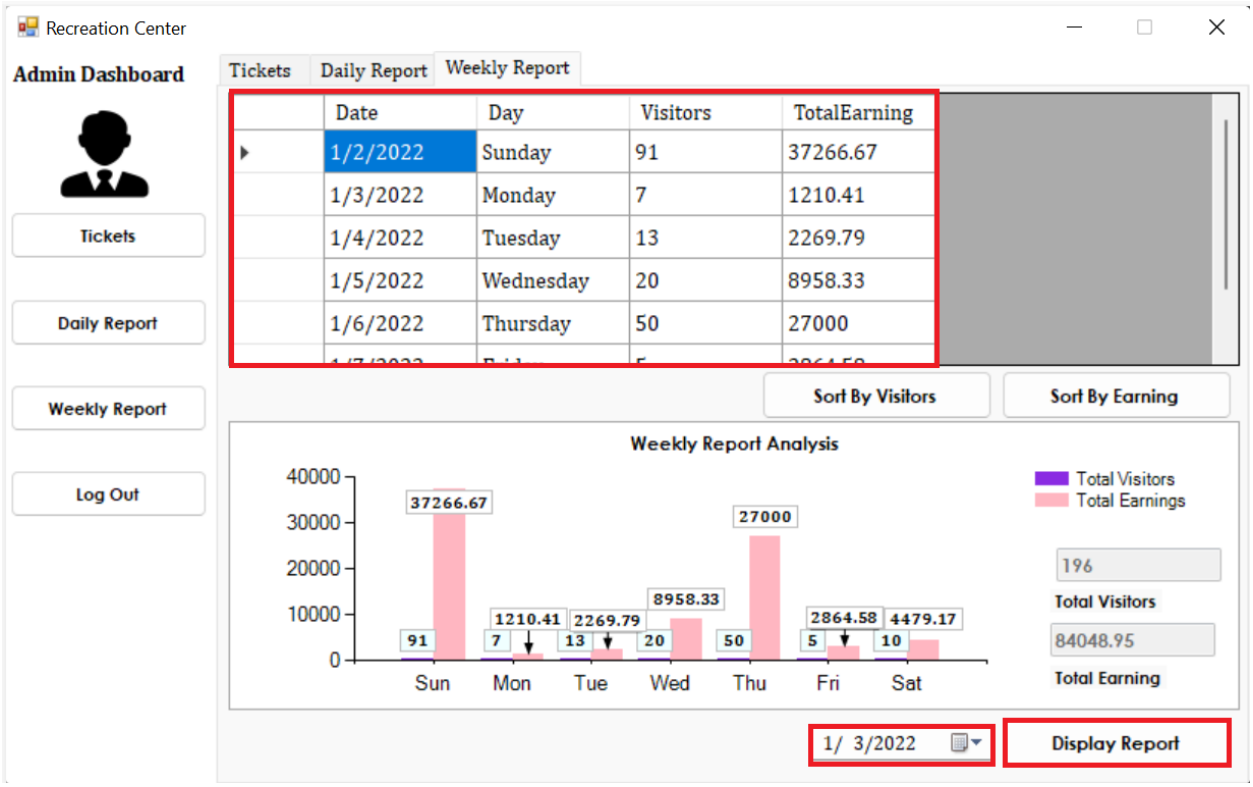

Figure 16: Displaying Weekly Report

2.11. Sorting by Number of Visitors

After displaying the weekly report, the data can be sorted according to the total visitors' count by clicking the 'Sort By Visitors' button.

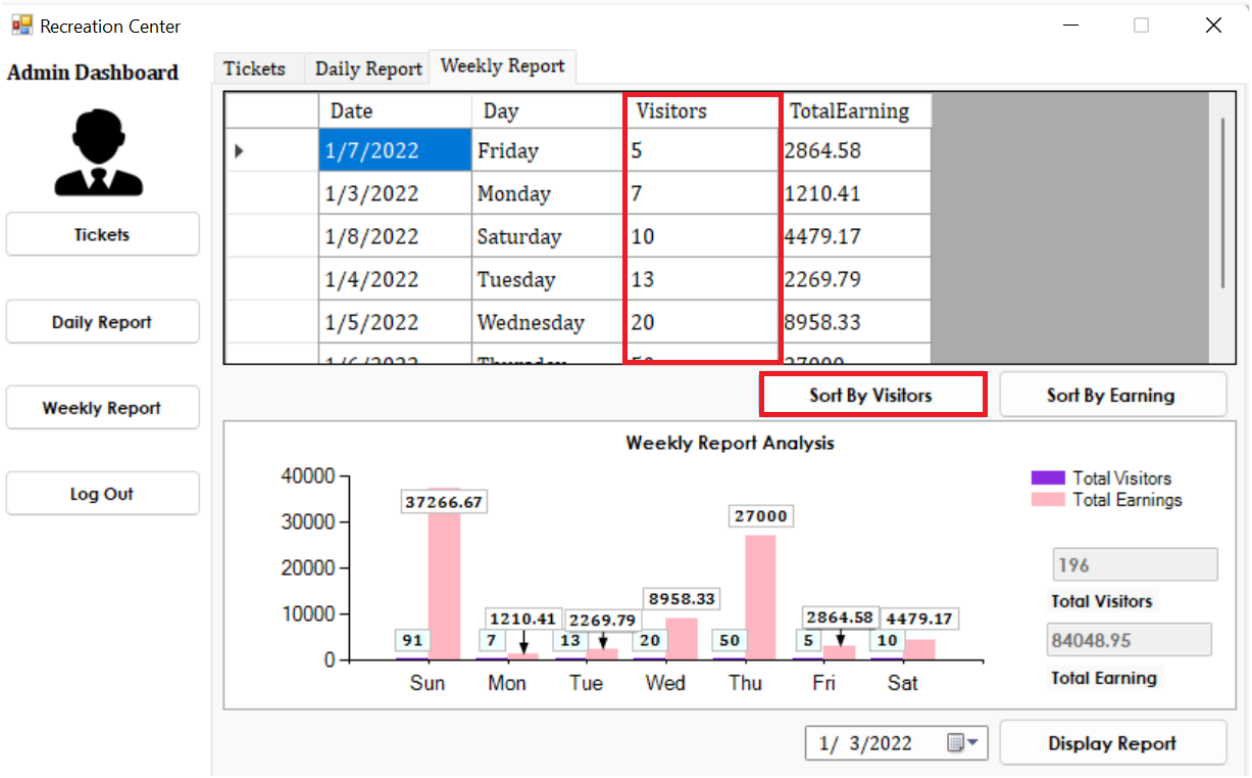

Figure 17: Sorting weekly data by visitors count

2.12. Sort by Earnings

Also, the displayed data will be sorted according to the earning while the 'Sort By Earning' button is clicked.

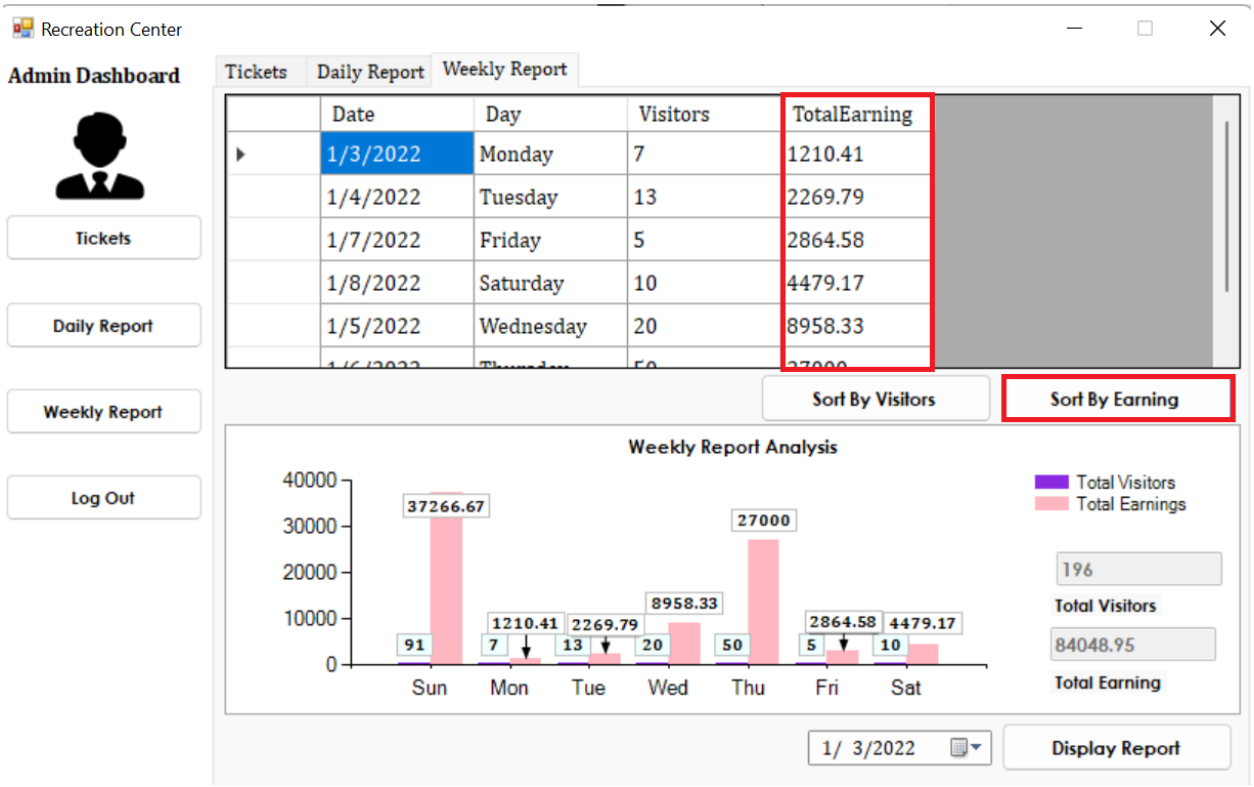

Figure 18: Sorting weekly data by earnings

3. Software Architecture Diagram

The software architecture diagram assists one in abstracting the overall outline of the application. The constraints, relationships, and boundaries between the elements can be figured out with the help of the software architecture diagram. Overall, it provides the physical structure of the software deployment and its evolution roadmap.

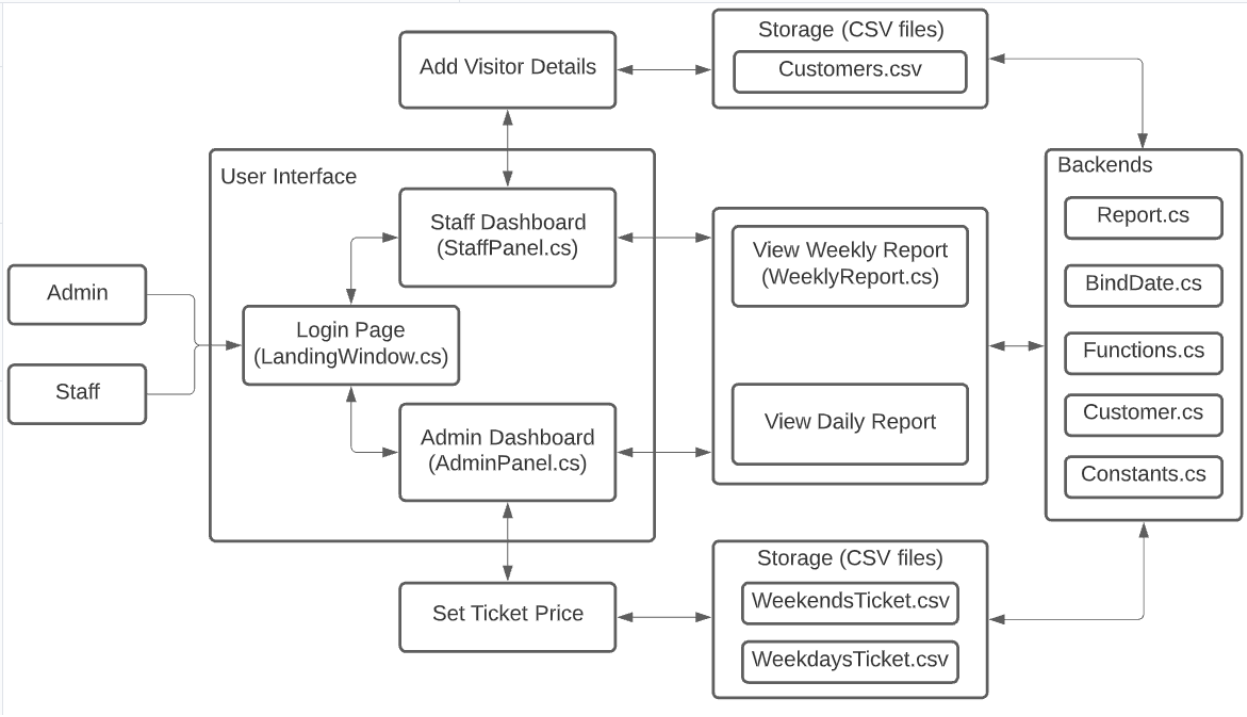

Figure 19: Software Architecture Diagram

From the above figure of the software architecture diagram for the desktop application, the System's flow and relationship can be predicted. There are two types of users, i.e., an Admin and a Staff. Both users are prompted towards the login page after launching the program, which is a Windows Form. The class 'LandingWindow' provides both design and functionality of the login.

Secondly, users will be navigated towards their respective dashboards after logging in. The class 'StaffPanel' (a user control form) provides functionalities and designs of the staff dashboard, while the category 'AdminPanel' (a user control form) does the same for the admin dashboard. The Admin can set ticket prices and save them to the CSV files located in the project folder path, whereas staff can add and save visitor details on the CSV file.

Thirdly, both users can view the daily and the weekly report. The POCO class 'WeeklyReport', stores the objects of the weekly report based on dates, days, total visitors, and total earnings. For displaying reports and carrying out other operations, CSV files and other classes depend on the backend.

The 'Report' class contains the functions that handle creating and displaying the weekly and daily reports in the backend. The class 'BindData' assists in binding and displaying data from the CSV file to the grid view. On the other hand, the 'Functions' class implements the functionalities used in more than one class like, adding data to the POCO class, writing data on the CSV files, and sorting data. Next, the 'Customer' is a POCO class for storing the object of the type customer. Finally, the' Constants' class has all variables used throughout the other courses. The convenience of changing data at only one location has supported implementing this class.

Also, the inbuilt MSDN classes like File, LINQ, and more have been utilized for different operations like handling, queries, etc.

4. Class Diagram

The class diagram of the overall classes implemented in the project, along with their relation, is mentioned below:

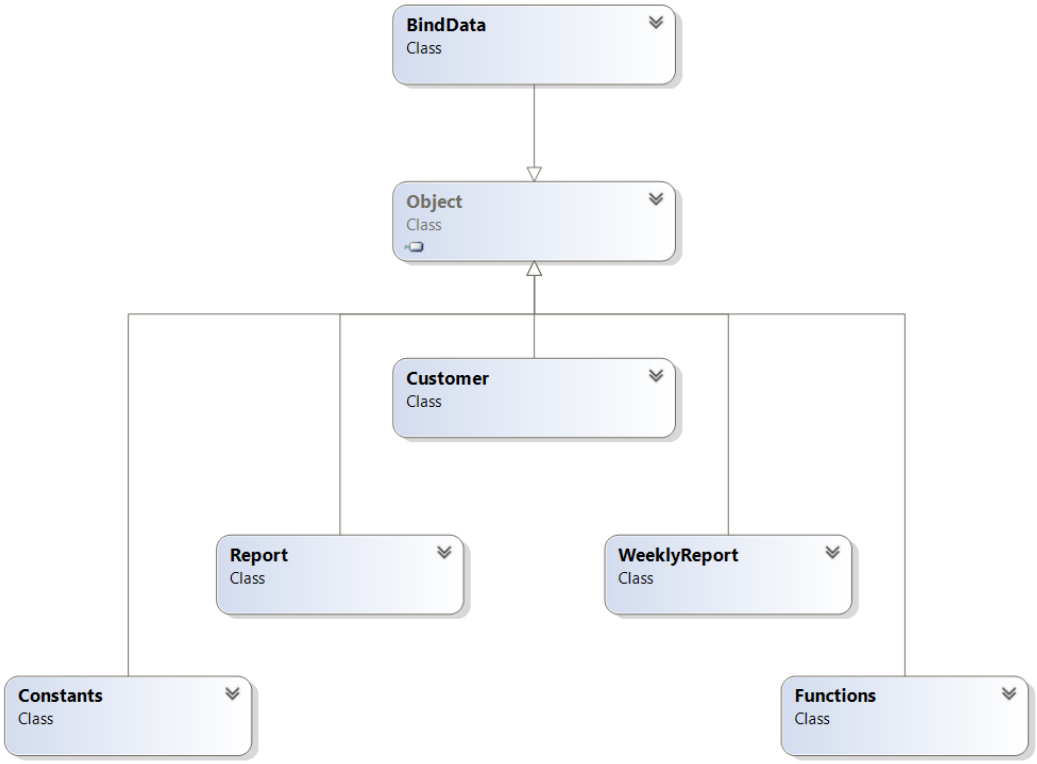

Figure 20: Class Diagram A

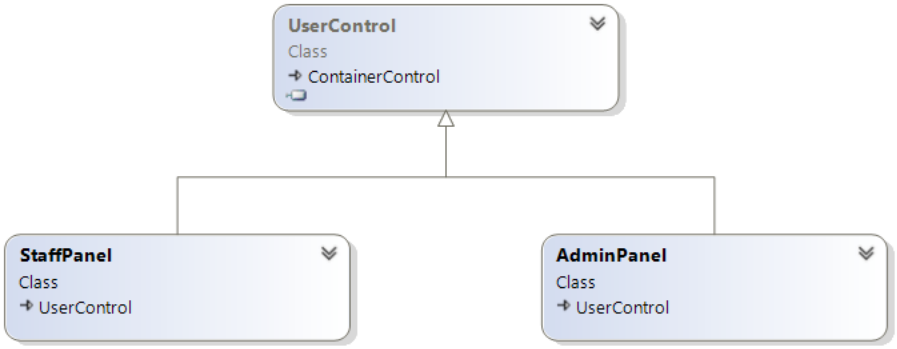

Figure 21: Class Diagram B

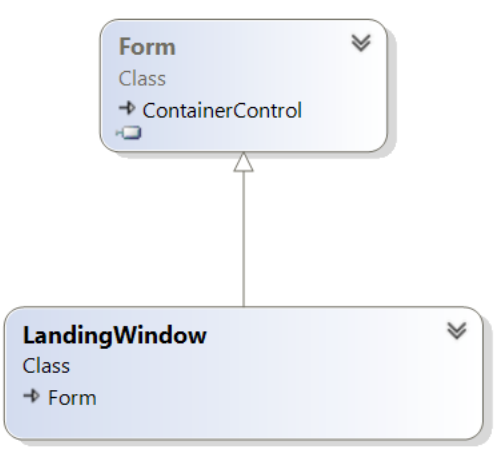

Figure 22: Class Diagram C

4.1. Program.cs

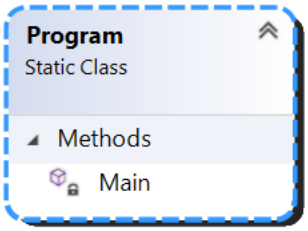

Figure 23: Class Diagram (Program.cs)

4.2. LandingWindow.cs

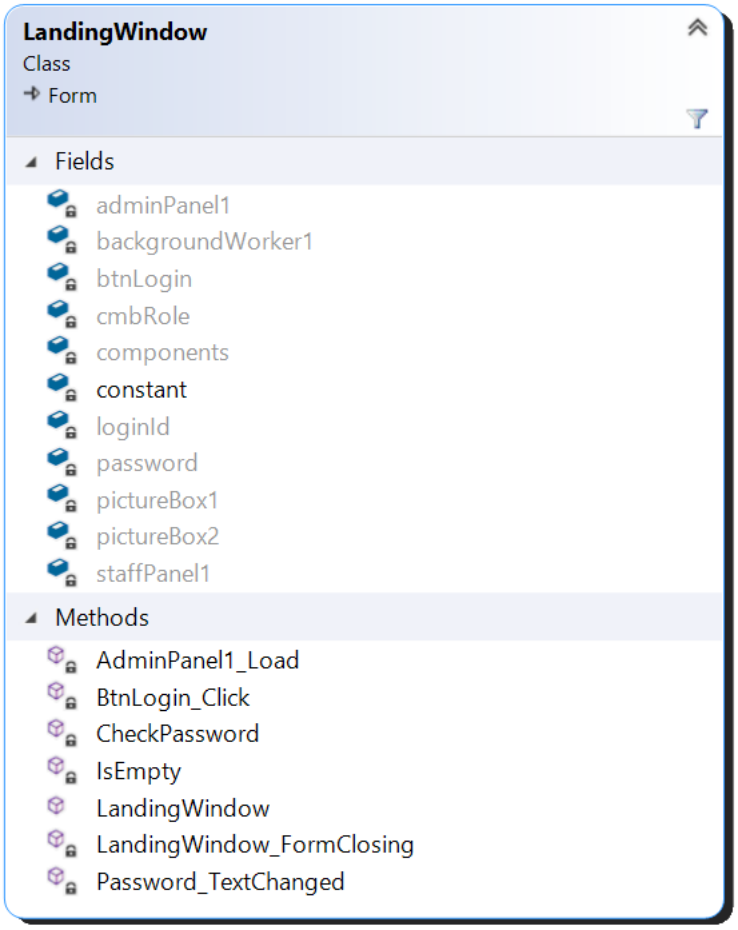

Figure 24: Class Diagram (LandingWindow.cs)

4.3. AdminPanel.cs

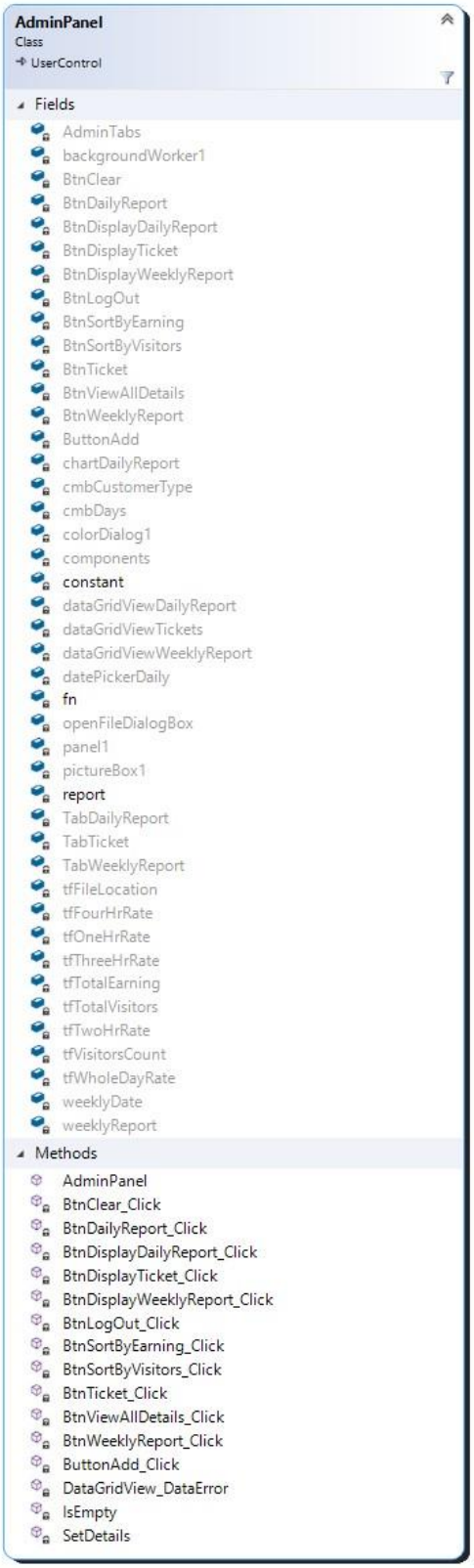

Figure 25: Class Diagram (AdminPanel.cs)

4.4. StaffPanel.cs

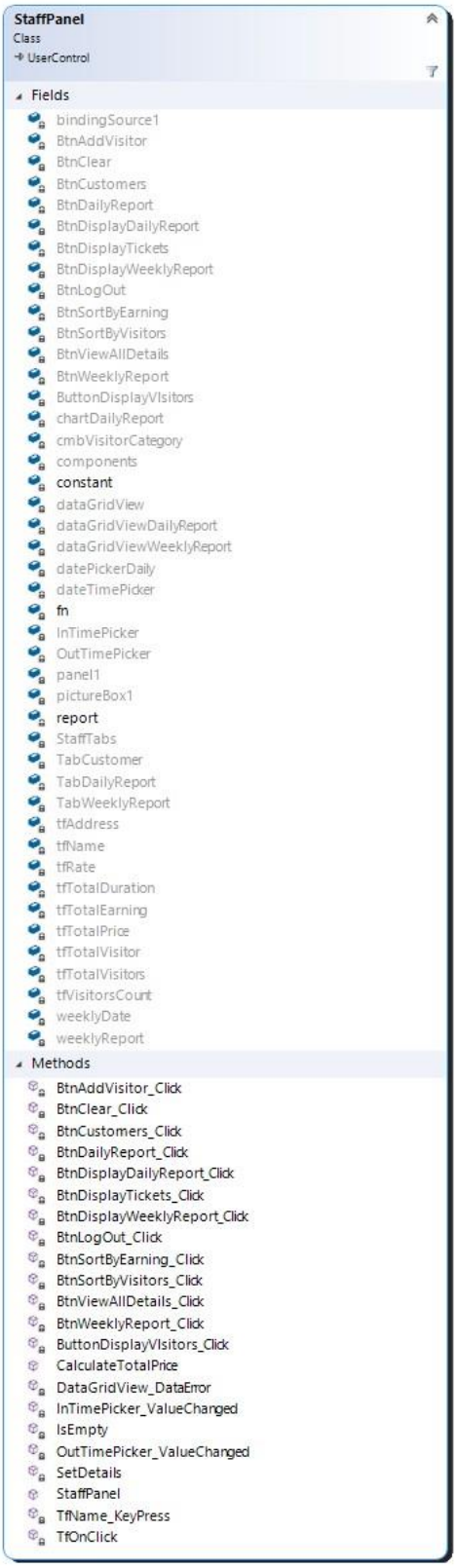

Figure 26: Class Diagram (StaffPanel.cs)

4.5. BindData.cs

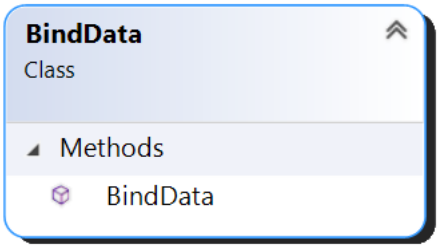

Figure 27: Class Diagram (BindData.cs)

4.6. Customer.cs

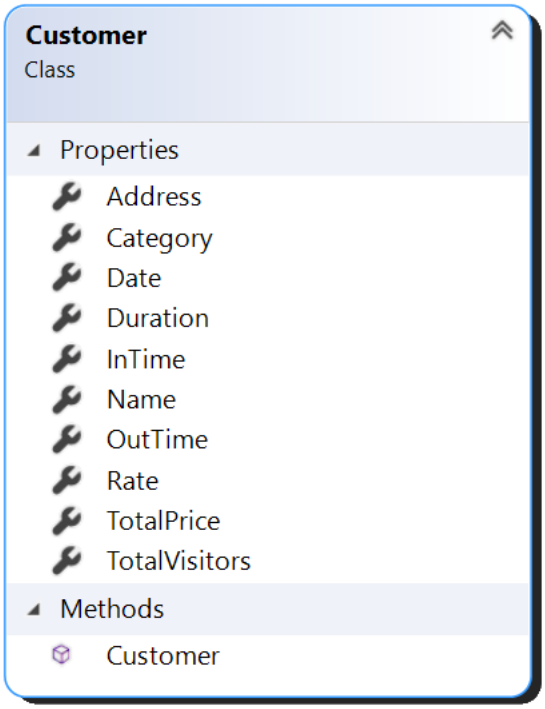

Figure 28: Class Diagram (Customer.cs)

4.7. WeeklyReport.cs

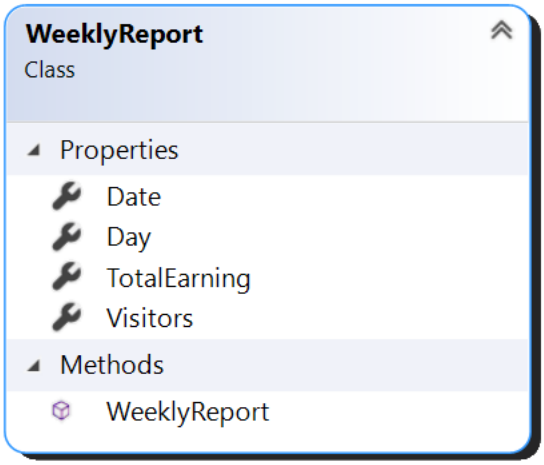

Figure 29: Class Diagram (WeeklyReport.cs)

4.8. Constants.cs

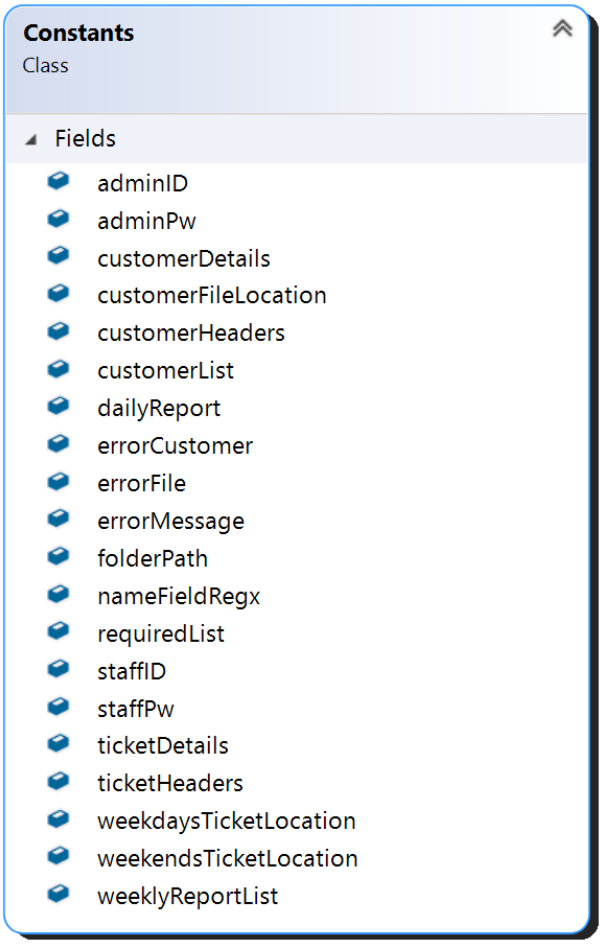

Figure 30: Class Diagram (Constants.cs)

4.9. Functions.cs

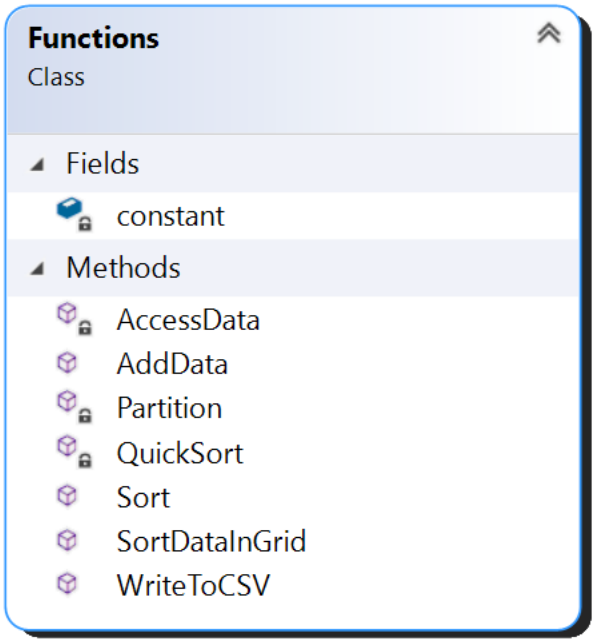

Figure 31: Class Diagram (Functions.cs)

4.10. Report.cs

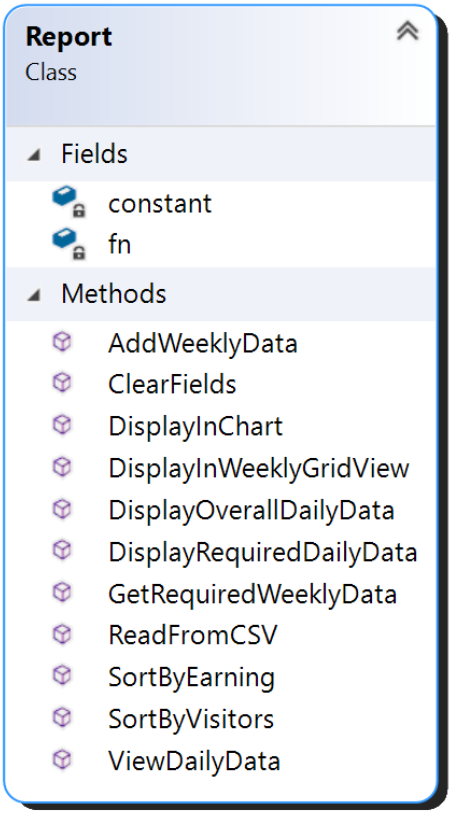

Figure 32: Class Diagram (Report.cs)

5. Method Description

A detailed description of the methods according to each of the class are mentioned below:

5.1. AdminPanel

Access Modifier and Return Type	Method and Description
private and void	**BtnClear_Click(object sender, EventArgs e)** This method clears the overall text fields and combo boxes when the' Clear' button is pressed.
private and void	**BtnDailyReport_Click(object sender, EventArgs e)** When the 'Daily Report' button is pressed, this method opens the tab displaying daily reports.
private and void	**BtnDisplayDailyReport_Click(object sender, EventArgs e)** When the 'Display Report' is clicked, this method calls functions from the 'Report' class for clearing grid view and chart. Further, the data from the CSV file will be abstracted and displayed.
private and void	**BtnDisplayTicket_Click(object sender, EventArgs e)** When the 'Display Ticket' button is pressed, this method opens the dialog box and sends the selected file's location to the 'BindData' class for portraying in the grid view.
private and void	**BtnDisplayWeeklyReport_Click(object sender, EventArgs e)** When this method is invoked, functions from the 'Report' class will be called for displaying a weekly report in the grid view and chart.
private and void	**BtnLogOut_Click(object sender, EventArgs e)** When the 'Log out' button is pressed, this method hides the current admin panel.
private and void	**BtnSortByEarning_Click(object sender, EventArgs e)**

	When the 'Sort By Earning' button is pressed, this method calls the 'SortByEarning' method from the report class for sorting the data based on earning.
private and void	**BtnSortByVisitors_Click(object sender, EventArgs e)** When the 'Sort By Visitors' button is pressed, this method calls the 'SortByVisitors' method from the report class for sorting the data based on visitors count.
private and void	**BtnTicket_Click(object sender, EventArgs e)** When the 'Ticket' button is pressed, this method opens the tab for adding ticket prices.
private and void	**BtnViewAllDetails_Click(object sender, EventArgs e)** When the 'View All Details' button is pressed, this method calls functions for displaying the overall daily report, including all details of customers.
private and void	**BtnWeeklyReport_Click(object sender, EventArgs e)** When the 'Weekly Report' button is pressed, this method opens a tab displaying the daily report.
private and void	**ButtonAdd_Click(object sender, EventArgs e)** When the 'Add' button is pressed, this method validates broad fields for inserting ticket details and adds the provided information to the CSV file.
private and void	**DataGridView_DataError(object sender, DataGridViewDataErrorEventArgs e)** This method handles error that may cause due to external data-parsing.
private and Boolean	**IsEmpty()** This method checks for the empty fields and returns' true' if any.
private and void	**SetDetails()** This method concatenates strings for adding into the CSV file.

Table 1: Method Descriptions (AdminPanel)

5.2. StaffPanel

Access Modifier and Return Type	Method and Description
private and void	**BtnAddVisitor_Click(object sender, EventArgs e)** When the 'Add' button is pressed, this method validates the input data and calls the function for writing into the CSV file.
private and void	**BtnClear_Click(object sender, EventArgs e)** When the 'Clear' button is pressed, this method clears the text fields, combo boxes, and date-time pickers.
private and void	**BtnCustomers_Click(object sender, EventArgs e)** This method displays a tab for inserting and viewing the customer details and opening ticket rates.
private and void	**BtnDailyReport_Click(object sender, EventArgs e)** This button displays the tab for viewing the daily report.
private and void	**BtnDisplayDailyReport_Click(object sender, EventArgs e)** When the 'Display Report' is clicked, this method calls functions from the 'Report' class for clearing grid view and chart. Further, the data from the CSV file will be abstracted and displayed.
private and void	**BtnDisplayTickets_Click(object sender, EventArgs e)** When the 'Display Ticket' button is pressed, this method displays the weekends or weekdays ticket by checking days.
private and void	**BtnDisplayWeeklyReport_Click(object sender, EventArgs e)** When this method is invoked, functions from the 'Report' class will be called for displaying a weekly report in the grid view and chart.
private and void	**BtnLogOut_Click(object sender, EventArgs e)** When the 'Log out' button is pressed, this method hides the current admin panel.
private and void	**BtnSortByEarning_Click(object sender, EventArgs e)**

	When the 'Sort By Earning' button is pressed, this method calls the 'SortByEarning' method from the report class for sorting the data based on earning.
private and void	**BtnSortByVisitors_Click(object sender, EventArgs e)** When the 'Sort By Visitors' button is pressed, this method calls the 'SortByVisitors' method from the report class for sorting the data based on visitors count.
private and void	**BtnViewAllDetails_Click(object sender, EventArgs e)** When the 'View All Details' button is pressed, this method calls functions for displaying comprehensive daily reports including all details of customers.
private and void	**BtnWeeklyReport_Click(object sender, EventArgs e)** This method opens the tab for witnessing the weekly report.
private and void	**ButtonDisplayVIsitors_Click(object sender, EventArgs e)** This method displays the visitor details in the grid view.
private and double	**CalculateTotalPrice(double rate, int totalVisitors)** This method returns the total price by calculating it with total time and rate.
private and void	**DataGridView_DataError(object sender, DataGridViewDataErrorEventArgs e)** This method handles error that may cause due to external data-parsing.
private and void	**InTimePicker_ValueChanged(object sender, EventArgs e)** This method calculates and displays total duration by determining the entry and exit times.
private and Boolean	**IsEmpty()** This method checks for the empty fields and returns' true' if any.
private and void	**OutTimePicker_ValueChanged(object sender, EventArgs e)** This method calculates and displays total duration by determining the entry and exit times.
private and void	**SetDetails()**

	This method concatenates strings for adding into the CSV file.
private and void	**TfName_KeyPress(object sender, KeyPressEventArgs e)** This method consumes any literal characters except alphabets for validating the name field.
private and void	**TfOnClick(object sender, EventArgs e)** This method clears the text box while clicking.

Table 2: Method Descriptions (StaffPanel)

5.3. LandingWindow

Access Modifier and Return Type	Method and Description
private and void	**AdminPanel1_Load(object sender, EventArgs e)** This method initially sets the visibility of an admin and a staff panel to false.
private and void	**BtnLogin_Click(object sender, EventArgs e)** This method checks for empty fields and passwords from the login page.
private and Boolean	**CheckPassword()** This method verifies the input credentials and displays either Admin or staff panel.
private and Boolean	**IsEmpty()** This method checks for the empty fields and returns' true' if any.
private and void	**LandingWindow_FormClosing(object sender, FormClosingEventArgs e)** This method displays confirmation box before closing the form.
private and void	**Password_TextChanged(object sender, EventArgs e)** This method changes the input password to the '*' character.

Table 3: Method Descriptions (LandingWindow)

5.4. Functions

Access Modifier and Return Type	Method and Description
private and dynamic	**AccessData(WeeklyReport list, string sortBy)** This method returns either visitors numbers or earnings from the list according to the given string property.
public and Customer	**AddData(string line)** This method creates the instance of Customer class from the given line of CSV files' data and returns the specific instance of Customer.
private and int	**Partition(List<WeeklyReport> weekData, int left, int right, string sortBy)** This method swaps the list data by comparing it with the pivot elements for sorting.
private and void	**QuickSort(List<WeeklyReport> week data, int left, int right, string sortBy)** This method calls the 'Partition' method and recursively calls itself for partitioning the provided list to be sorted.
public and void	**Sort(List<WeeklyReport> weekData, string sortBy)** This method calls the 'QuickSort' method by passing a list and its' ending and starting points and the property to be sorted.
public and void	**SortDataInGrid(List<WeeklyReport> weekData, string sortingProperty, DataGridView gridView)** This method checks if the provided grid view is empty and calls the 'Sort' method if not so.
public and void	**WriteToCSV(string path, string headers, string details)** This method writes into the CSV file for the given location and appends heading if the file does not exist initially.

Table 4: Method Descriptions (Functions)

5.5. Report

Access Modifier and Return Type	Method and Description
public and void	**AddWeeklyData(DateTime weekStart, DateTime weekEnd)** By filtering the current week's data, this method calculates total visitors and earnings. And finally, adds an instance of the 'WeeklyReport' class to the list.
public and void	**ClearFields(DataGridView gridView, Chart chart)** This method clears the grid view, chart, and different lists.
public and void	**DisplayInChart(Chart weeklyReport, TextBox visitors, TextBox earning)** By calculating earnings and visitors of each day, this method populates the chart with the data.
public and void	**DisplayInWeeklyGridView(DataGridView gridView)** This method displays the list to the grid view.
public and void	**DisplayOverallDailyData(DataGridView gridView)** This method displays overall customer details in the grid view.
public and void	**DisplayRequiredDailyData(DateTimePicker dateTimePicker, TextBox textBox, Chart chart)** This method displays the required data into the daily chart by calculating the weekly customer data.
public and void	**GetRequiredWeeklyData(DateTime weekStart, DateTime weekEnd)** This method filters the current week dates and adds them to the required list.
public and void	**ReadFromCSV()** This method reads the customer data from the CSV file and adds the list of customers with the data.
public and void	**SortByEarning(DataGridView gridView)**

	This method calls the 'SortDataInGrid' method from the 'Function' class for sorting the data by earnings.
public and void	**SortByVisitors(DataGridView gridView)** This method calls the 'SortDataInGrid' method from the 'Function' class for sorting the data by visitors.
public and void	**ViewDailyData(DataGridView gridView)** This function displays the dictionary of a daily report into the data grid view by converting it to the array.

Table 5: Method Descriptions (Report)

5.6. BindData

Access Modifier and Return Type	Method and Description
public	**BindData(string filePath, DataGridView dataGrid)** The constructor of the 'BindData' class binds the data from the CSV file to the data grid view.

Table 6: Method Descriptions (BindData)

6. Flowchart

The flowchart of implementing an algorithm for displaying the weekly report has been generated and depicted in the following section.

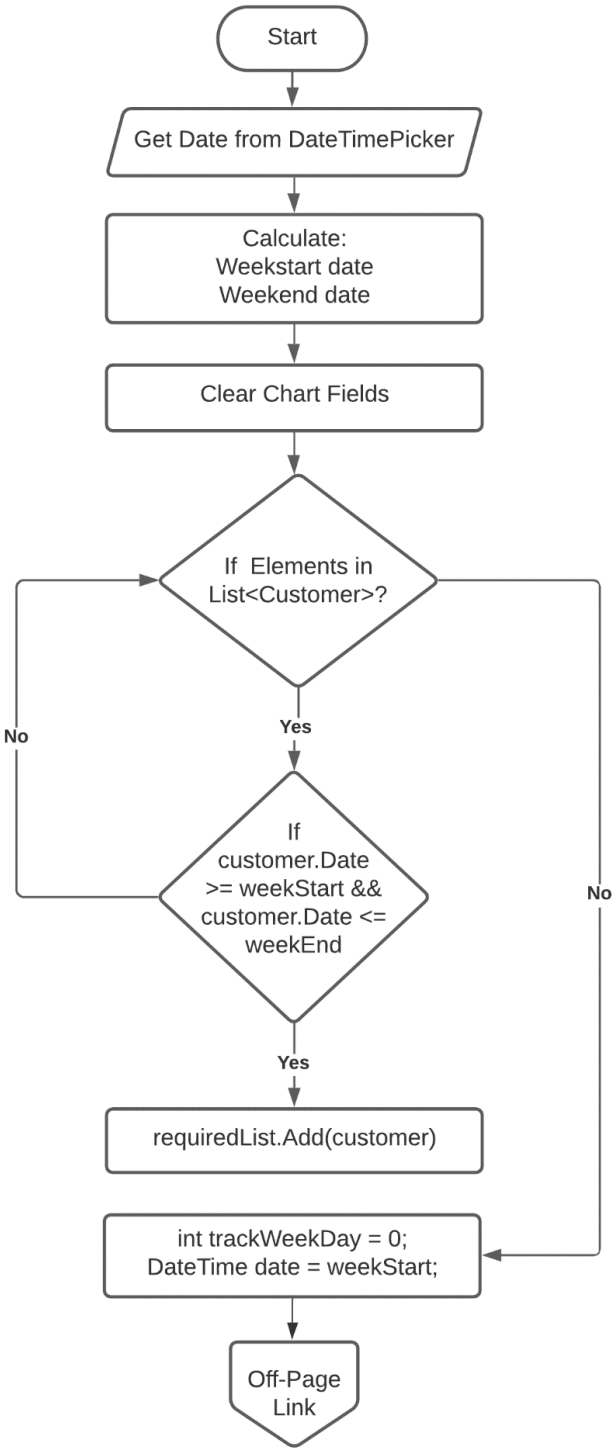

Figure 33: Flowchart A

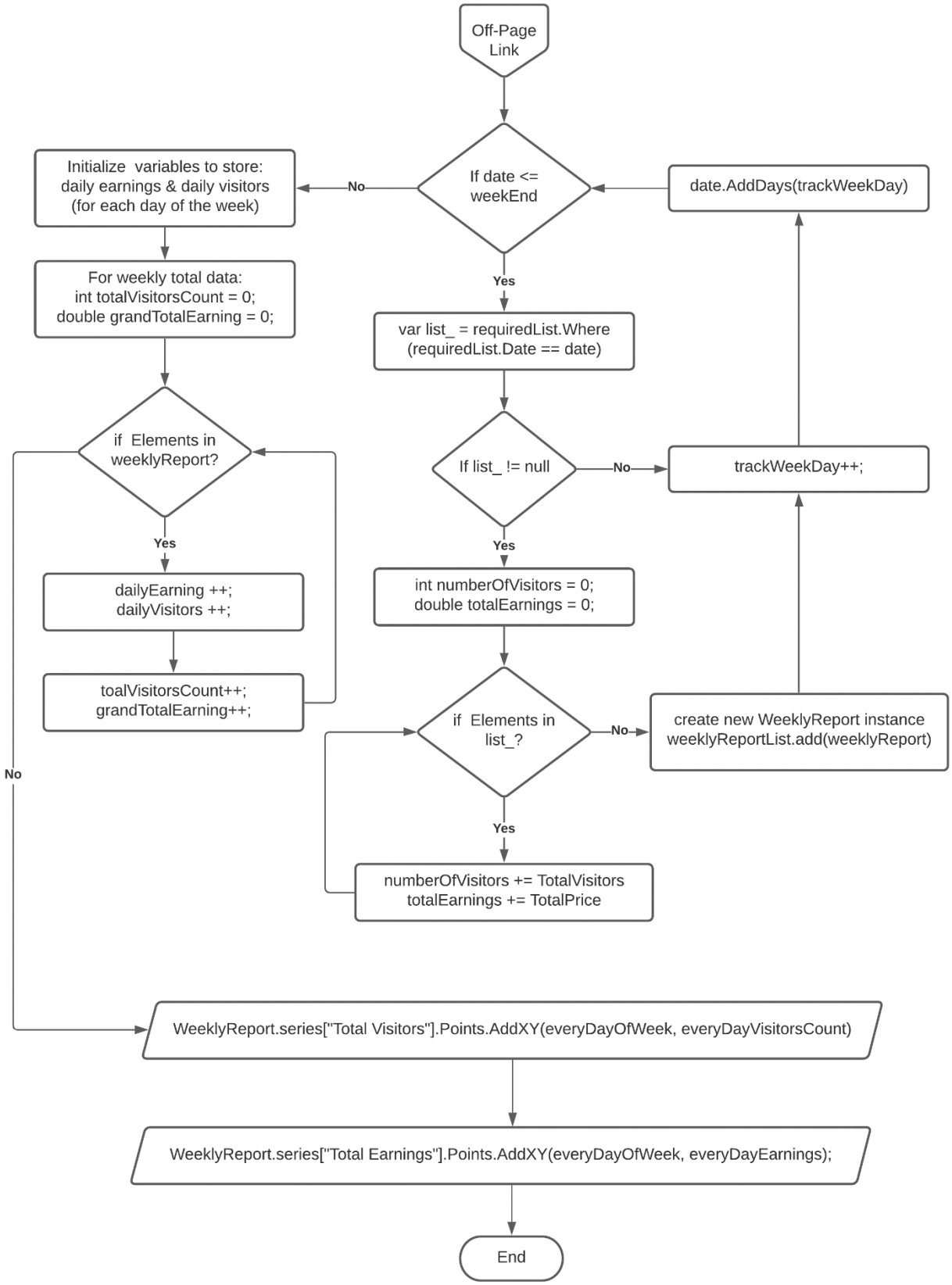

Figure 34: Flowchart B

7. Data Structures and Algorithm

7.1. Datatypes

Several data types have been utilized to store the appropriate types of data to perform various operations in the project. Some of the applied datatypes are briefly described below:

- **String**

  It stores character bounded by the double quotation. This datatype has been used for initializing and storing names, words, and texts.

- **Int**

  It is used for storing the whole number. This datatype has been used to keep the total number of visitors and other numerical digits for logical operations.

- **Double**

  It stores decimal numbers and can store up to 15 decimal points. This datatype has been used to keep the total price and the rate in the project.

- **Boolean**

  It stores either true or false decisions, which comes in handy while performing logical operations. This datatype has been used to check conditions in various program sections.

- **Array**

  This datatype can be used for storing multiple values of specific data types. It is used for storing lines of the CSV file by removing commas.

- **Dictionary**

  This data type stores data in the form of key-value pairs, which store and display daily reports' data.

- **List**

  This data type stores the collection of the objects. It has been utilized to store instances of POCO classes like Customer and WeeklyReport.

- **DateTime**

  It is the inbuilt data type for storing detailed date and time information. It has been used for manipulating, storing, and searching objects based on specific times.

- **Dynamic**

  This data type can store any kind of data to prevent errors during compile time. It has been used for storing and returning dynamic types of data based on input like double total price or total int visitors.

- **Var**

  This datatype implicitly stores any type of data based upon its initial value. It has been implied for temporarily storing an object or collection of things.

7.2. Quick Sort Algorithm.

Quicksort algorithm, a sorting algorithm, is based on the divide and conquer approach. Although the time complexity of this algorithm is **$O(n^2)$,** it can be efficiently utilized for small datasets.

Initially, a pivot element is selected from the list, and based upon the component, the list will be divided into subarrays. The features more minor than the pivot are kept on the left side, whereas greater ones are retained on the right side.

The exact process continues for every subarray until the subarrays become single. At this stage, the elements are sorted and must be merged.

The working mechanism of the quick sort algorithm is portrayed from the figure below:

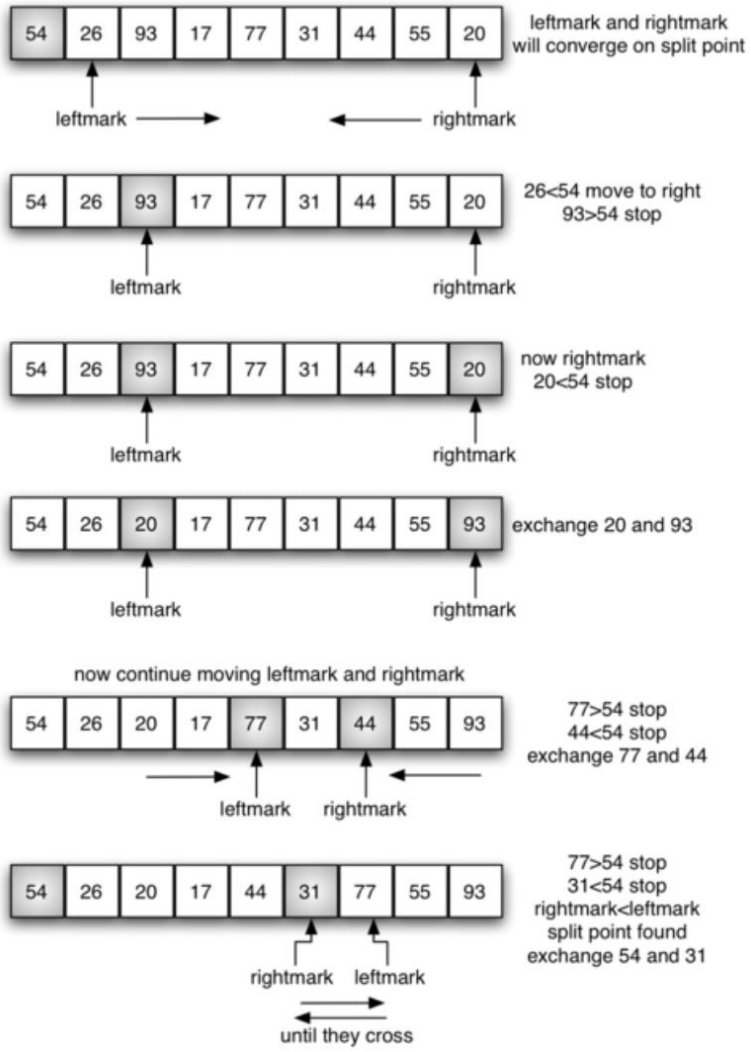

Figure 35: Quicksort

8. Personal Reflection and Conclusion

All in all, the complete coursework, which seemed unfeasible while handing, has been fluently rectified through immense dedication and intense research. The project-based desktop application development by employing the .Net Framework built on the C# programming language has been accomplished. The functionalities like facilitating the admin for setting ticket prices, the staff recording the visitors' details, and visualizing daily and weekly reports in the tabular and chart format have been achieved throughout the project.

Initially, adhering to C# and dealing with its complications was not easy. But gradually, with the project's flow, the core concept behind the C# like null safety, object-oriented, strong typing, etc., enlighten the knowledge of programming concepts. The automatic garbage collection feature frees the unreferenced storage claimed by the variables and makes the program efficient during run time. Other features, including better integrity, backward compatibility, make coding accessible and reflective. Since the syntax for C# is analogous to that of Java, the coding in this language becomes comfortable and straightforward.

Secondly, the IDE, i.e., Visual Studio 2019, has been used for the development of UI components and writing code for performing backends and connection with the storage files. The drag and drop features for developing UI components help rapidly implement user interfaces. The real-time error displaying part prevents many compile time errors and saves colossal time. The features include developing class diagrams, debugging, showing CPU usage during run time. Calculating metrics of the code based on maintainability index, depth of inheritance, cyclomatic complexity, class coupling, calculating lines of source code assisted in analyzing the performance of the developed system. Nevertheless, it was a remarkable experience using Visual Studio as an IDE and its' features for the project's development.

The system architecture has been designed by placing functions and variables of significant operations in different classes to be accessed from other classes when

required. By implementing POCO (Plain Old CLR Objects) types, the concept of object orientation and encapsulation has been achieved. Further, developing classes' lists helped navigate between objects, their properties, and data sorting.

During the developmental phases, problems regarding the incompatibility of data with different datatypes, figuring out appropriate datatype for storing objects and values, utilizing algorithms for sorting, playing with DateTime datatype, generating reports, and more were aroused. But intense enthusiasm for solving problems by researching and going through numerous online forums supported overcoming the issues.

Overall, it can be concluded that by eradicating the barriers from the loops of research, the confidence for developing similar projects based on the .Net Framework has been evolved. Finally, it can be assessed that the assessment has aided in enriching the skills regarding programming concepts, software architecture, and desktop application development. The wisdom gained from the project can indeed be employed in real-world scenarios' problems in the future.

References and Bibliography

Aggarwal, A. (2021) *Introduction to.NET Framework* [Online]. Available from: https://www.geeksforgeeks.org/introduction-to-net-framework/ [Accessed 29 December 2021].

Clouder, A. (2020) *How to Create an Effective Technical Architectural Diagram?* [Online]. Available from: https://www.alibabacloud.com/blog/how-to-create-an-effective-technical-architectural-diagram_596100 [Accessed 29 December 2021].

Microsoft. (2021) *It's how you make software* [Online]. Available from: https://visualstudio.microsoft.com/ [Accessed 29 December 2021].

MIcrosoft. (2021) *What is.NET Framework?* [Online]. Available from: https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet-framework [Accessed 29 December 2021].

Programiz. (2021) *Quicksort Algorithm* [Online]. Available from: https://www.programiz.com/dsa/quick-sort [Accessed 29 December 2021].

Singh, S. (2020) *var keyword in C#* [Online]. Available from: https://www.geeksforgeeks.org/var-keyword-in-c-sharp/ [Accessed 29 December 2021].

Skeet, J. (2019) *C# in Depth*. 4th ed. Manning Publications.

Stellman, A. & Greene, J. (2020) *Head First C#*. 4th ed. O'Reilly Media, Inc.

Troelsen, A. (2012) *Pro C# 5.0 and the.NET 4.5 Framework*. 6th ed. Apress.

Troelsen, A. & Japikse, P. (2015) *C# 6.0 and the.NET 4.6 Framework*. 7th ed. Apress.

Tutorialspoint. (2021) *C# - Structures* [Online]. Available from: https://www.tutorialspoint.com/csharp/csharp_struct.htm [Accessed 29 December 2021].

TutorialsPoint. (2021) *C# Language advantages and applications* [Online]. Available from: https://www.tutorialspoint.com/Chash-Language-advantages-and-applications [Accessed 19 December 2021].

Appendix A (Testing)

1. Test 1: Login

Objective	To log into the Admin and Staff dashboard, respectively.
Action	i.) Run the program. ii.) Provide a username for Admin and password. iii.) Logout from the Admin dashboard. iv.) Provide username and password for staff.
Expected Output	The admin dashboard should be displayed, logged out from the dashboard, and finally logged into the staff dashboard.
Actual Output	The admin dashboard was displayed, successfully logged out, and logged into the staff dashboard.
Result	Test Successful.

Table 7: Test Case 1

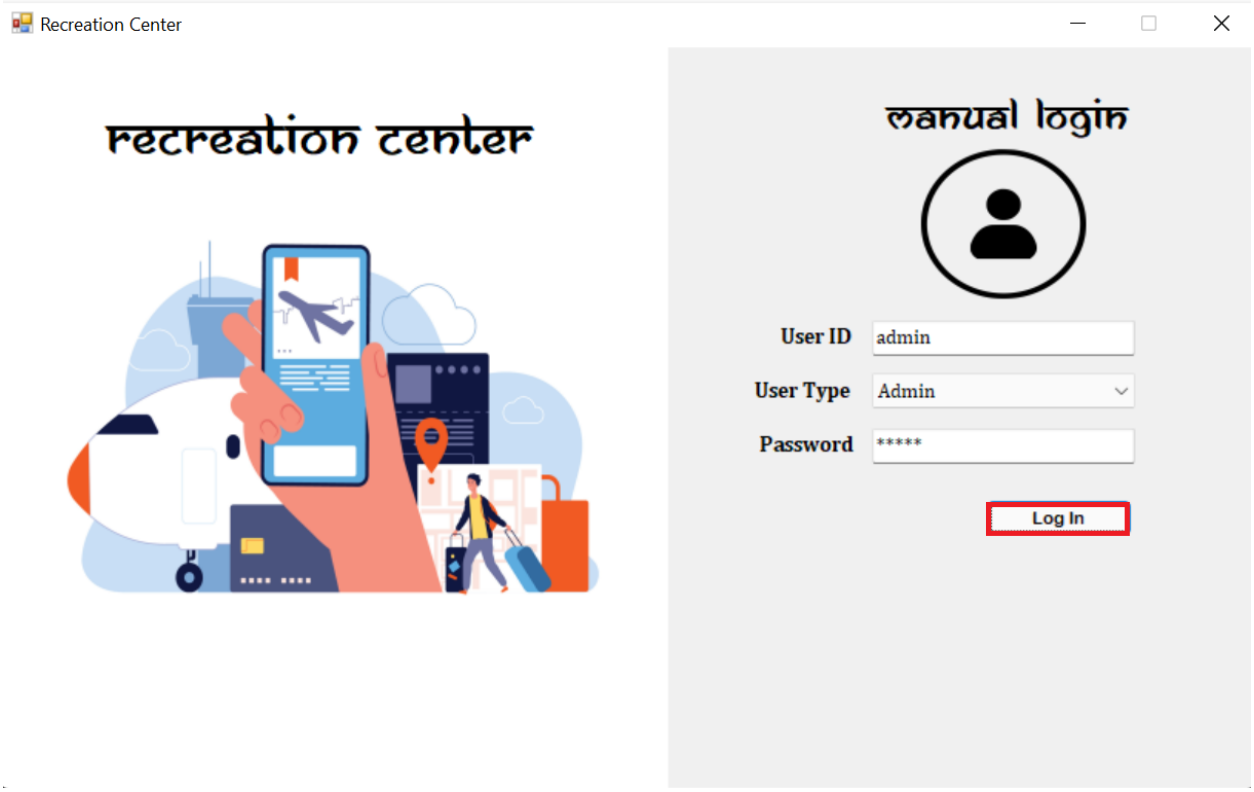

Figure 36: Inserting the Admin login Credentials

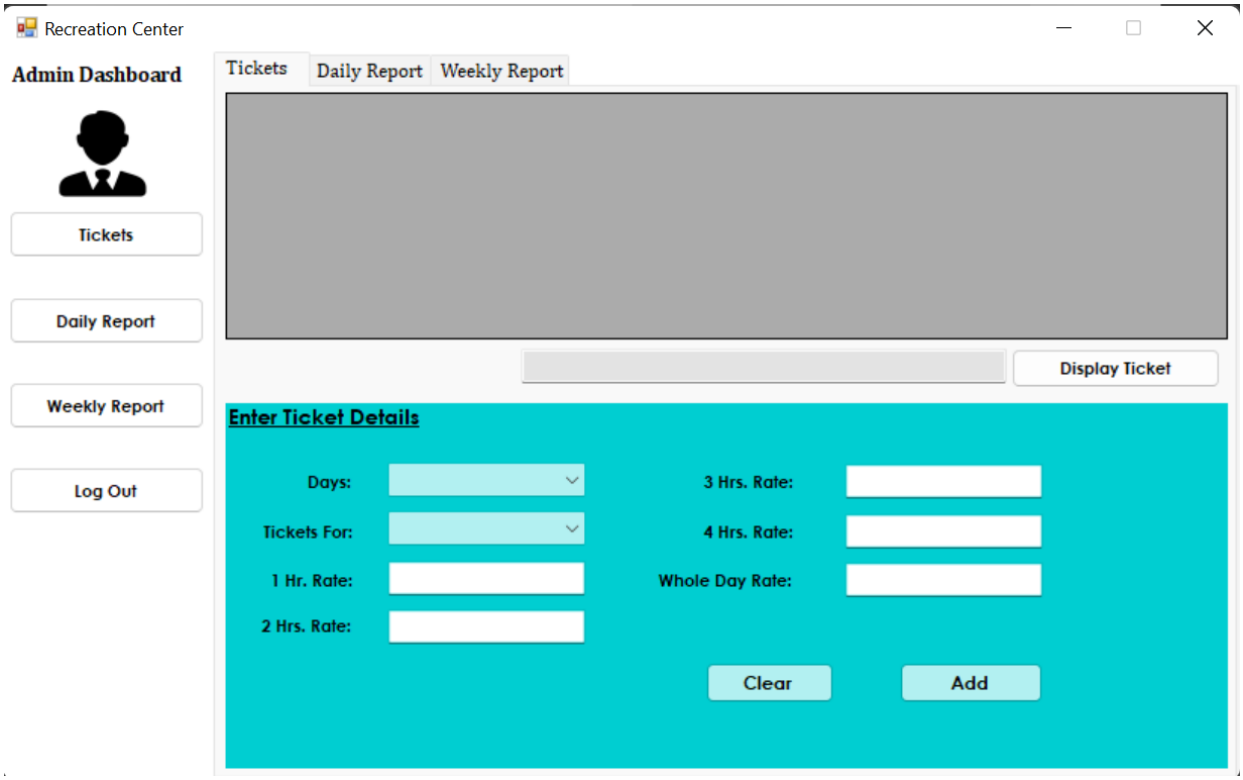

Figure 37: Admin Dashboard

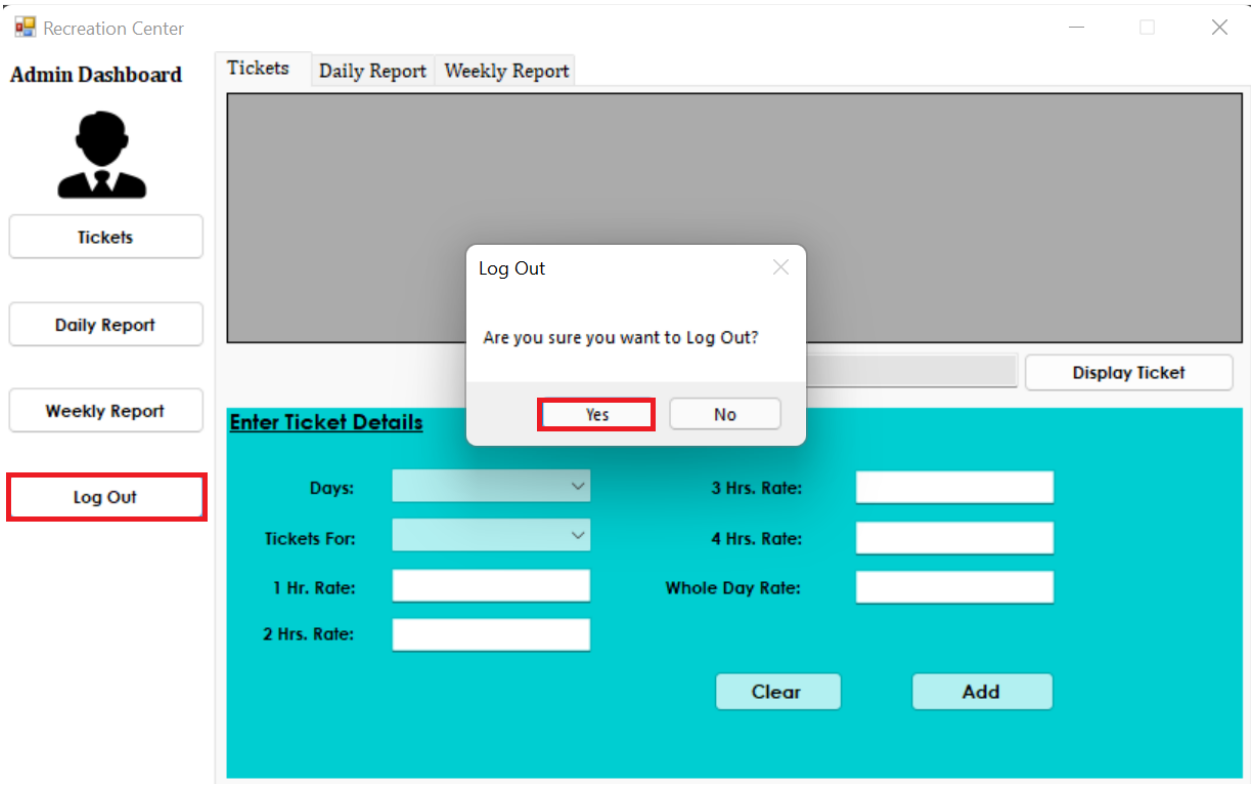

Figure 38: Logging out from the dashboard.

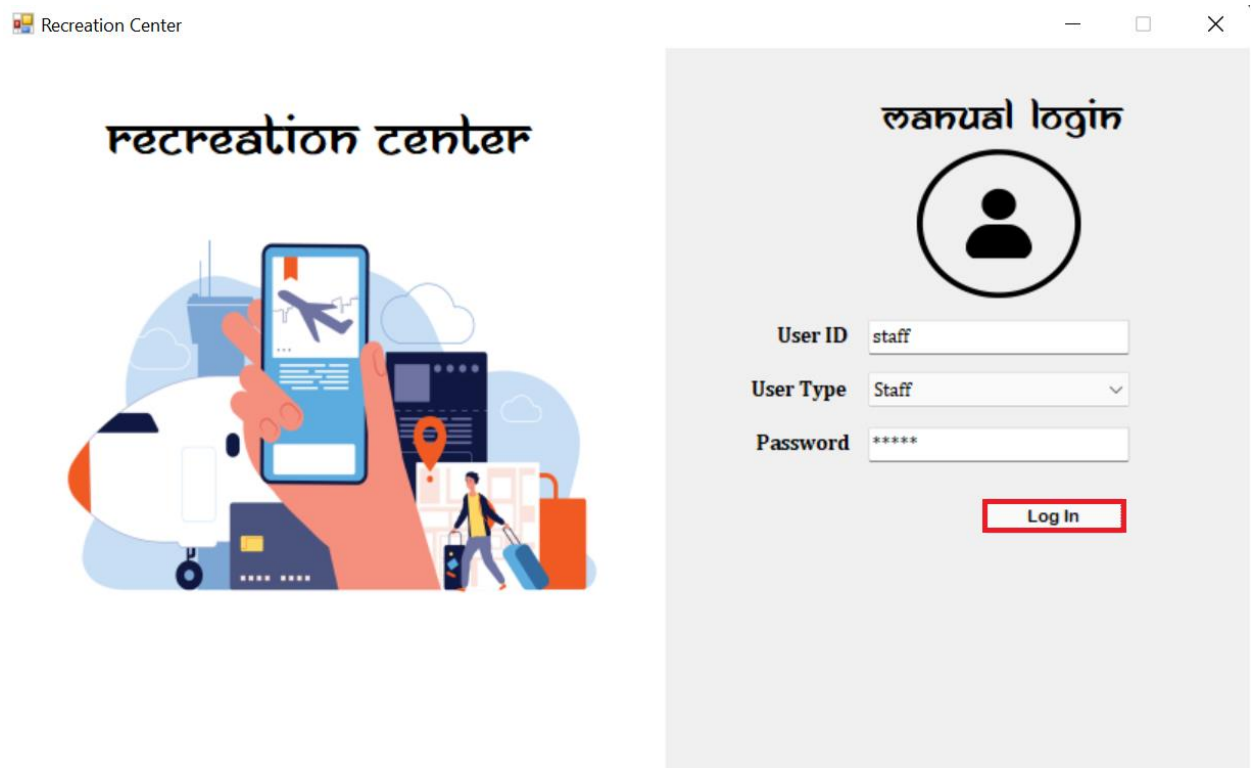

Figure 39: Inserting Staff Login Credentials

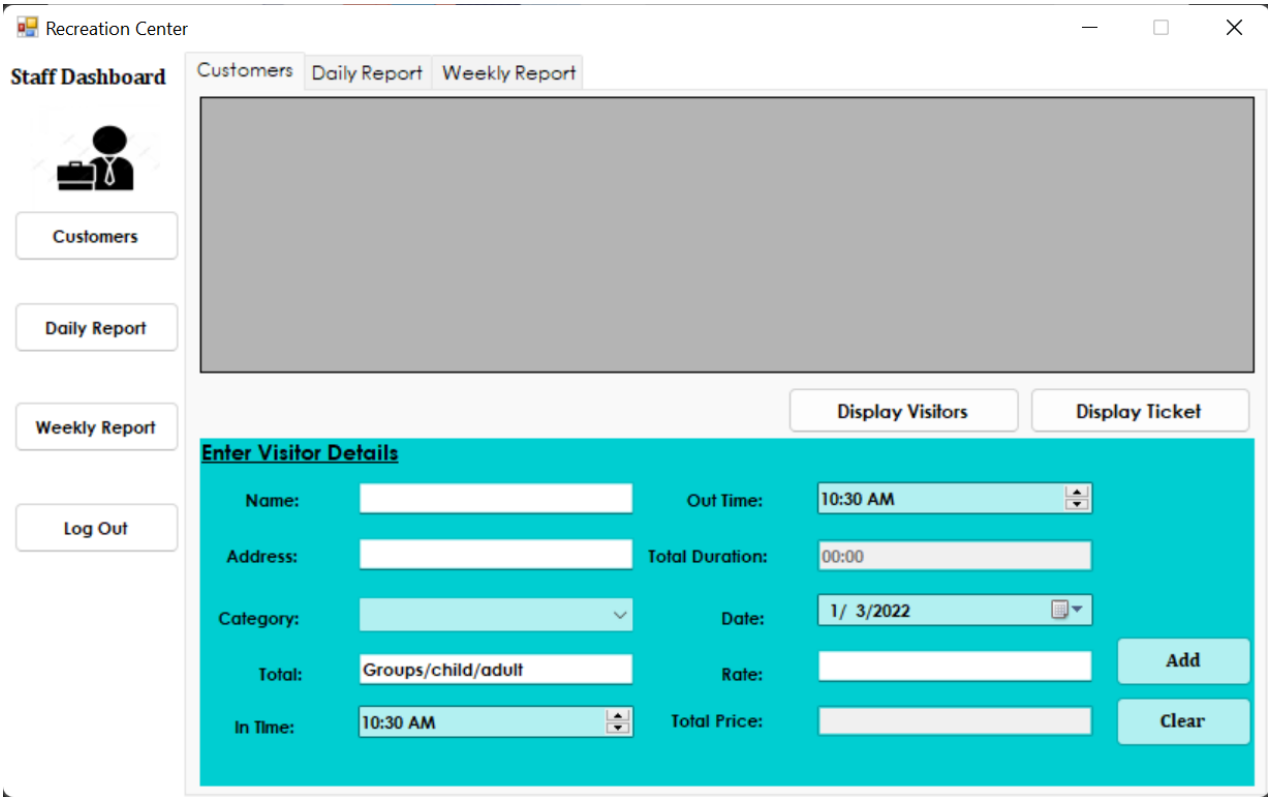

Figure 40: Staff Dashboard

2. Test 2: Ticket Price Validation

Objective	To add random figures or leave empty fields for rates.
Action	i.) Add random price rates like string and negative number ii.) Leave empty fields before adding ticket price
Expected Output	The error message should be displayed, and the error should be focused.
Actual Output	The error message was displayed, and the error messages were displayed
Result	Test Successful.

Table 8: Test Case 2

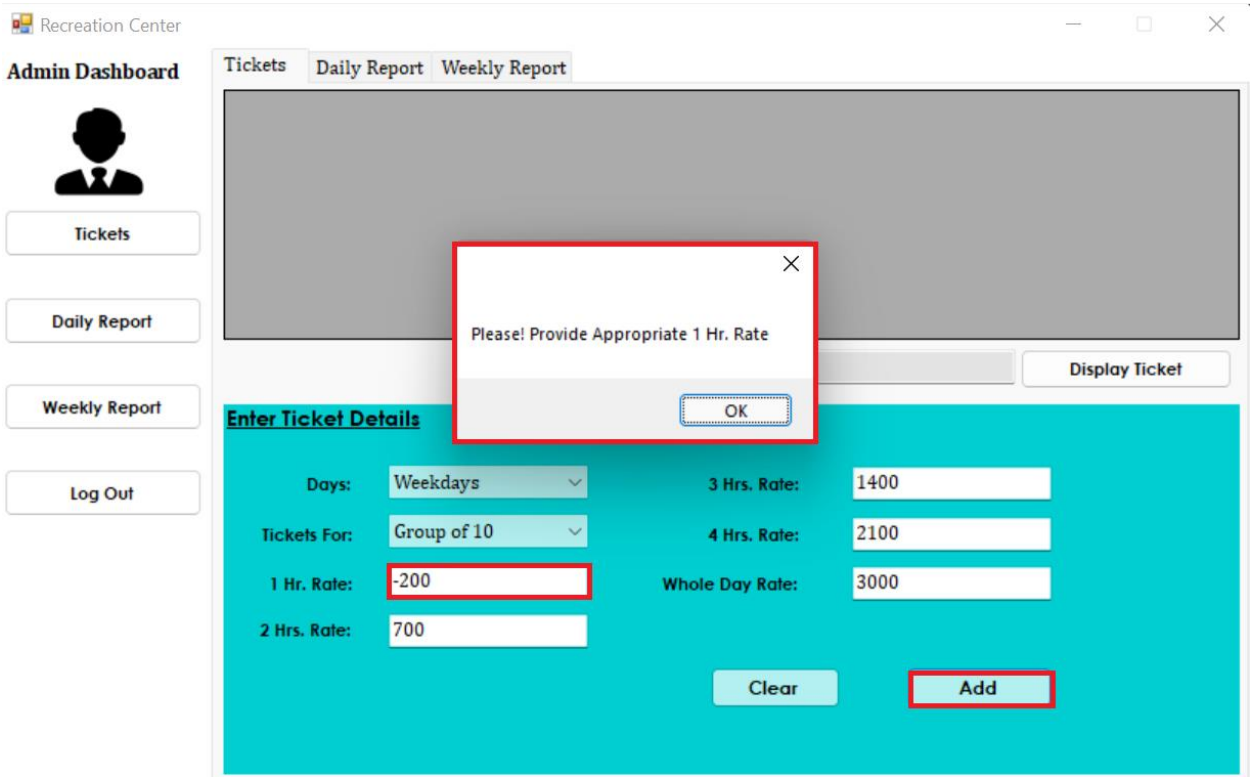

Figure 41: Error Message regarding invalid ticket rate.

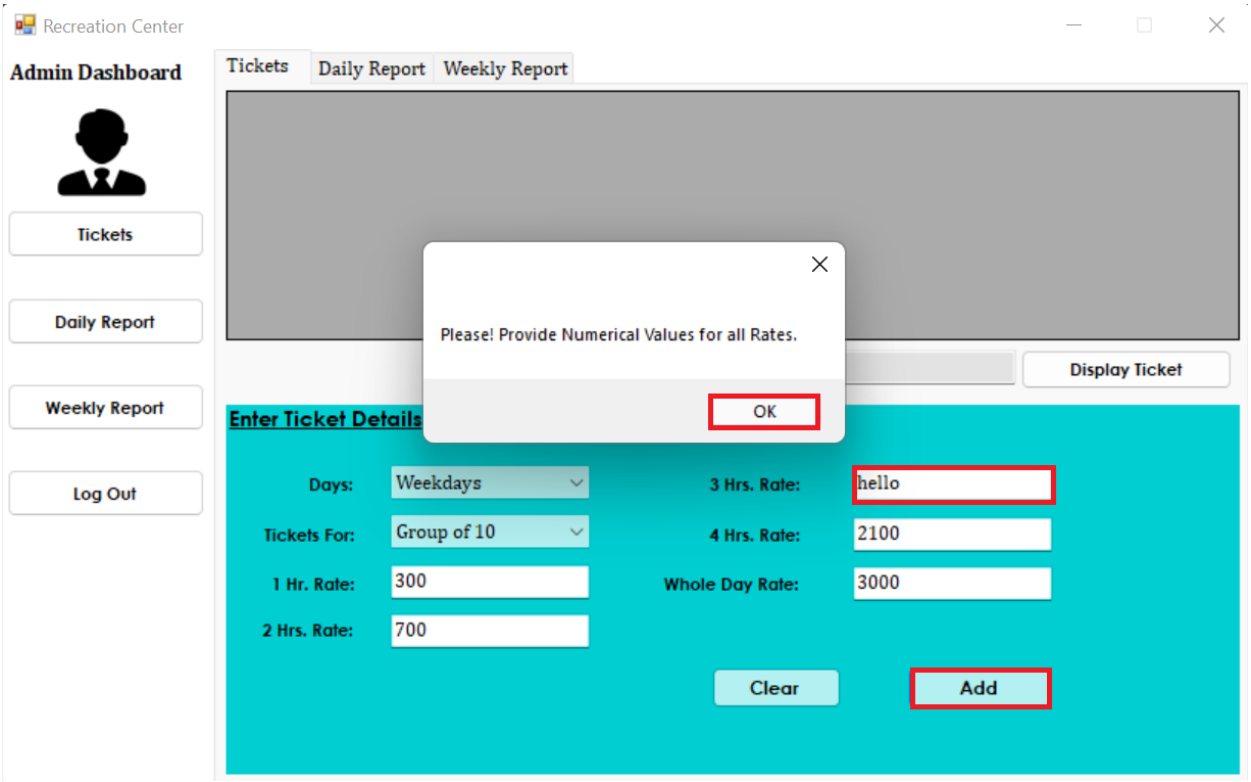

Figure 42: Providing character for the ticket rate

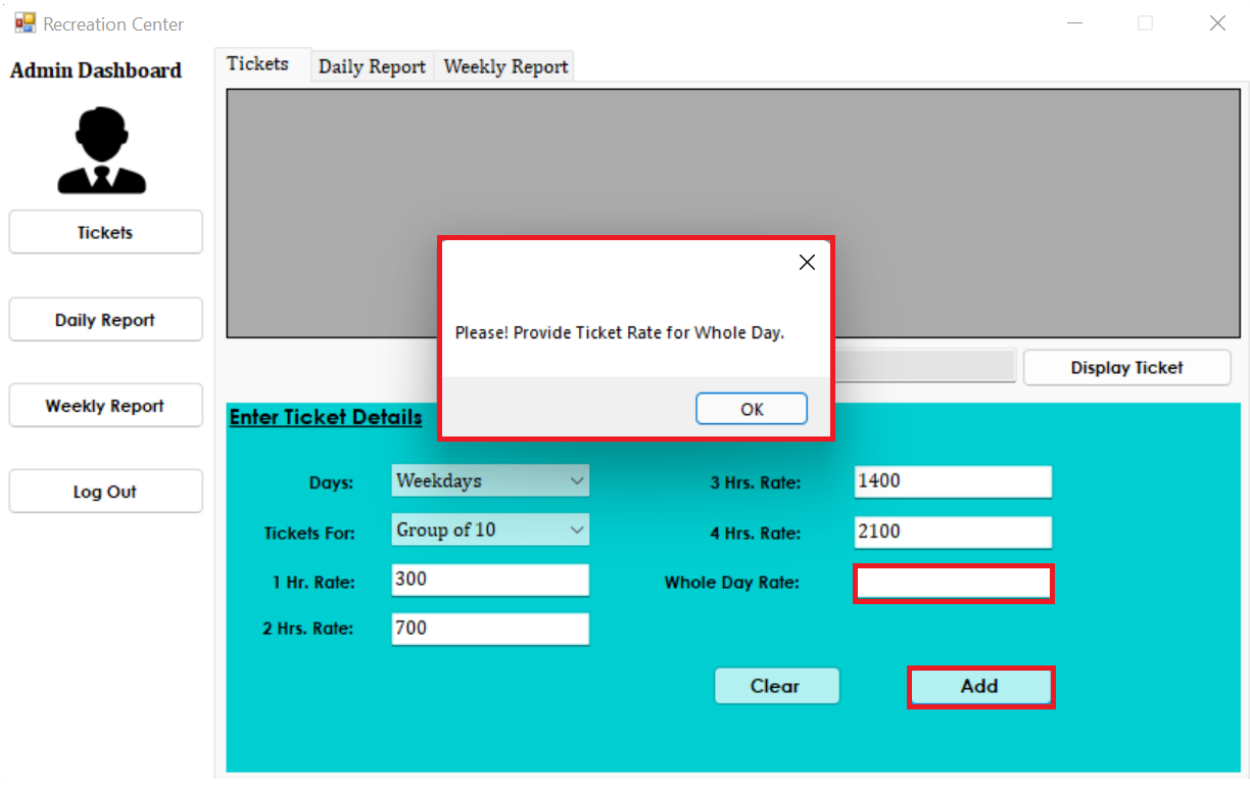

Figure 43: Leaving empty text field

3. Test 3: Visitors' Registration Validation

Objective	To add random visitors' details while registering.
Action	i.) Add visitors by providing random names, invalid entry, and exit times.
Expected Output	The staff should not press random keywords for names, and errors should be displayed in case of entry and exit time.
Actual Output	The staff could not press random keywords for name, and error was displayed in the case of entry and exit time.
Result	Test Successful.

Table 9: Test Case 3

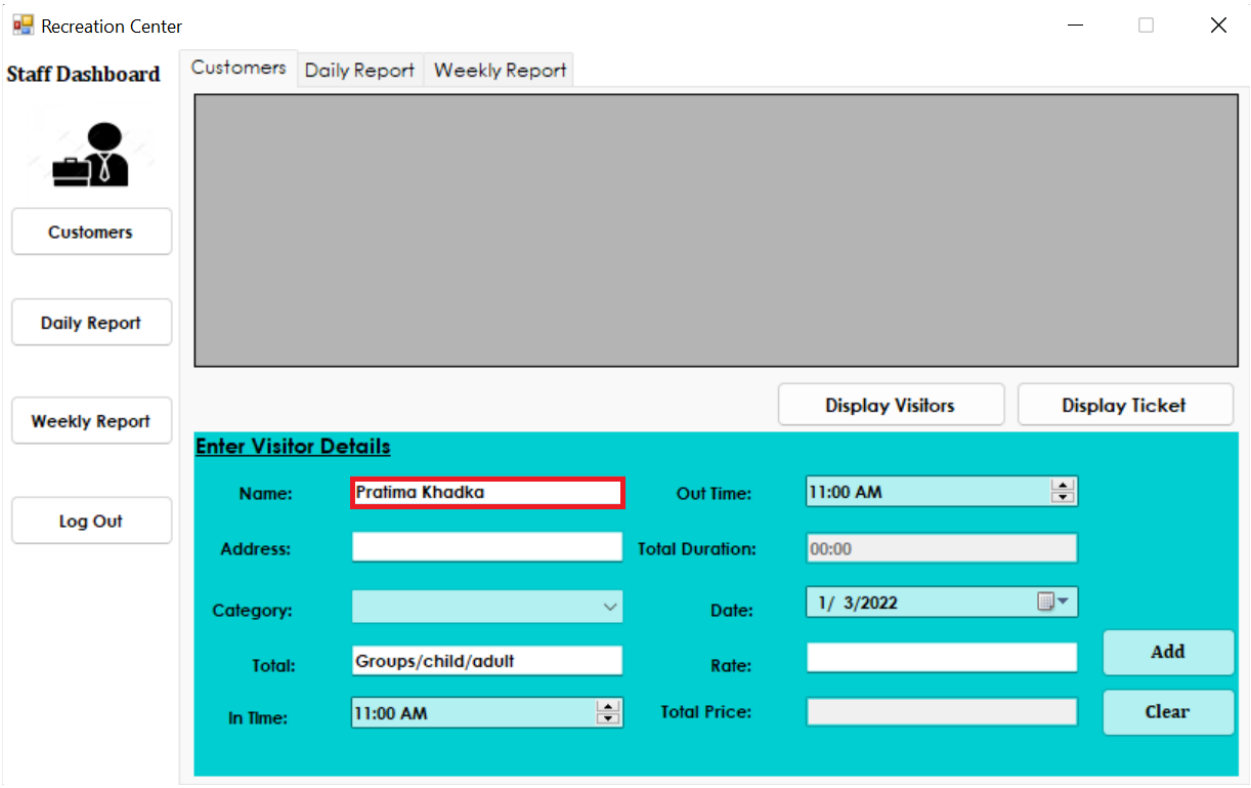

Figure 44: Unable to type any other characters or numbers except alphabets.

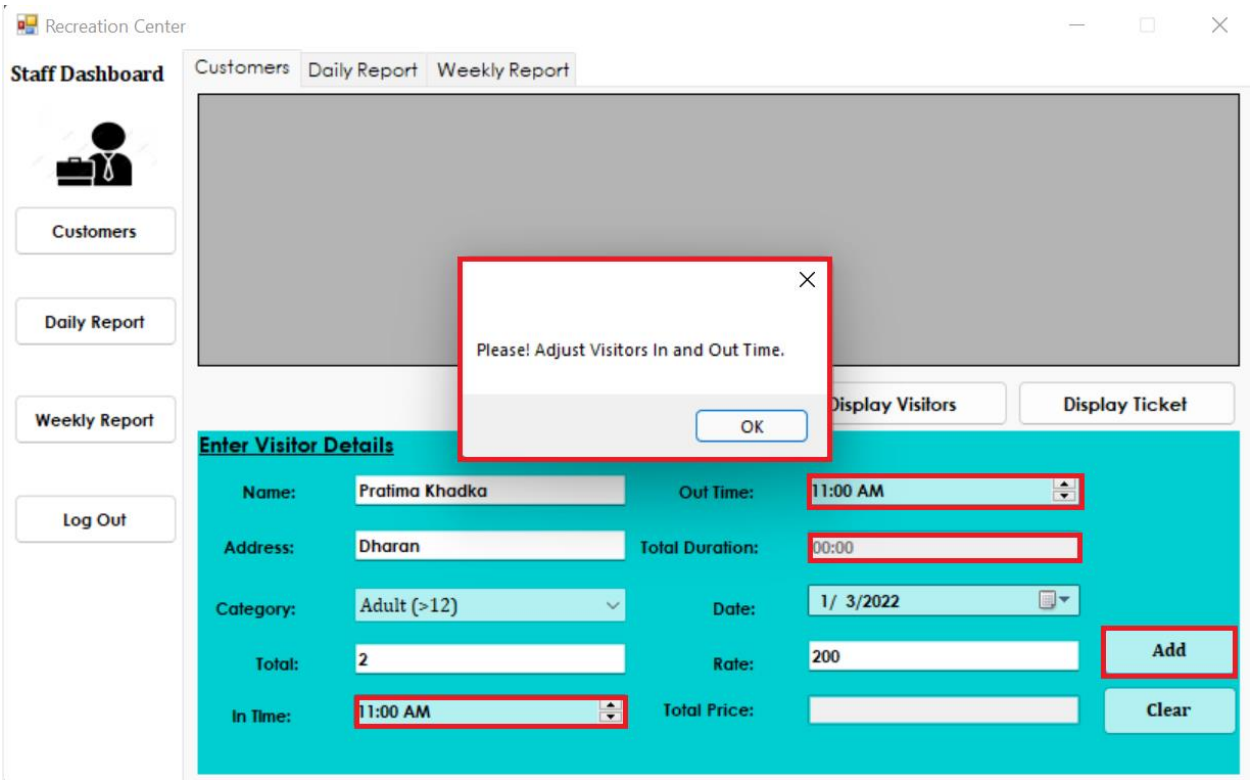

Figure 45: Error while adding visitors without adjusting time.

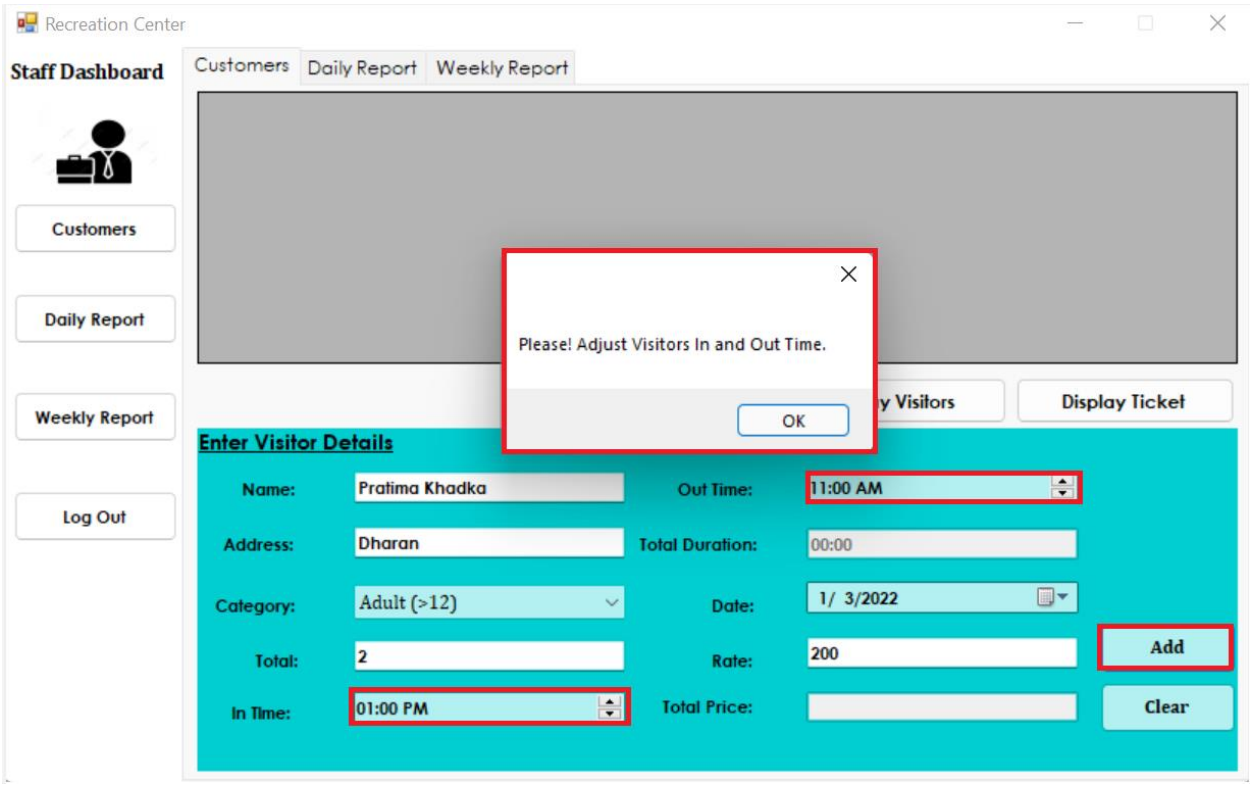

Figure 46: Error when entry time is more significant than exit time.

4. Test 4: Auto Calculation for total duration and price.

Objective	To figure out the total duration and the ticket price
Action	i.) Adjust entry and exit time and add the visitors
Expected Output	The total duration should be calculated from entry and exit time, and the total price should be automatically calculated for the entire period.
Actual Output	The total duration was calculated, and the full price was based on the time spent.
Result	Test Successful.

Table 10: Test Case 4

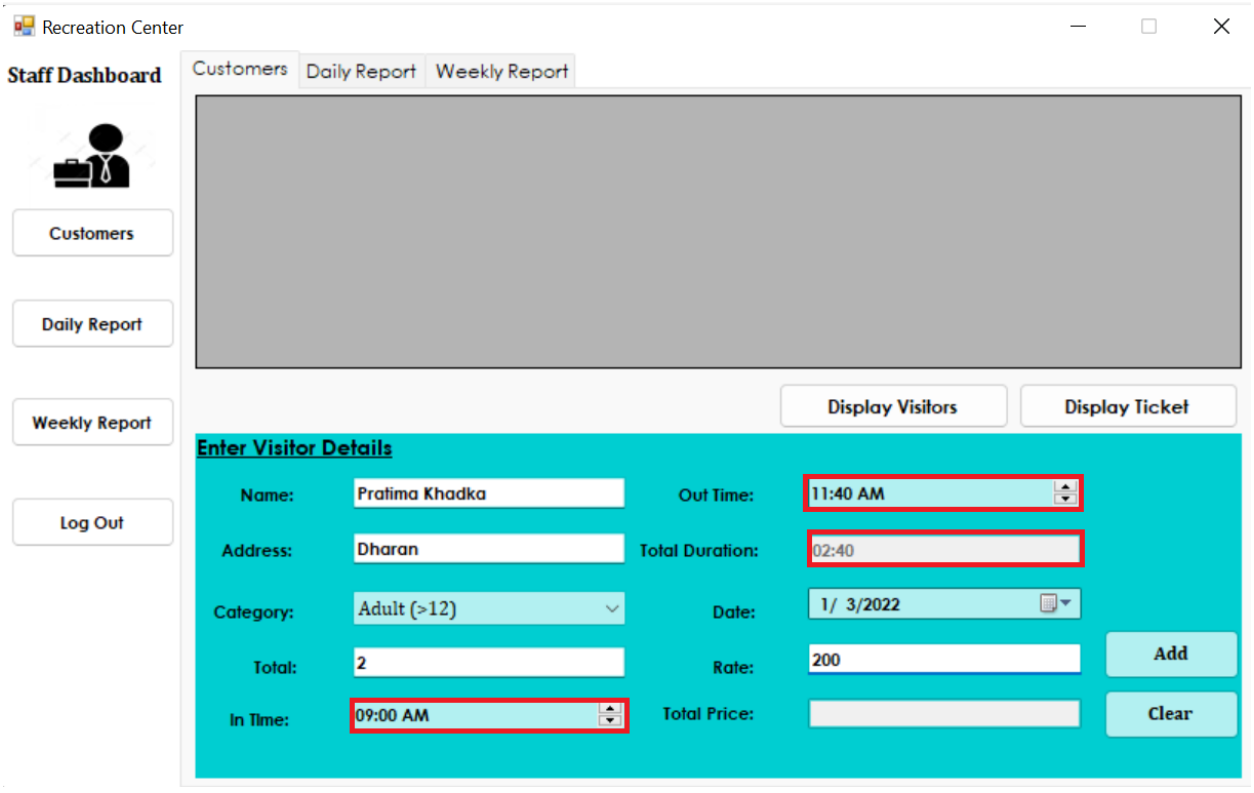

Figure 47: Calculation of total duration when entry and exit time are changed.

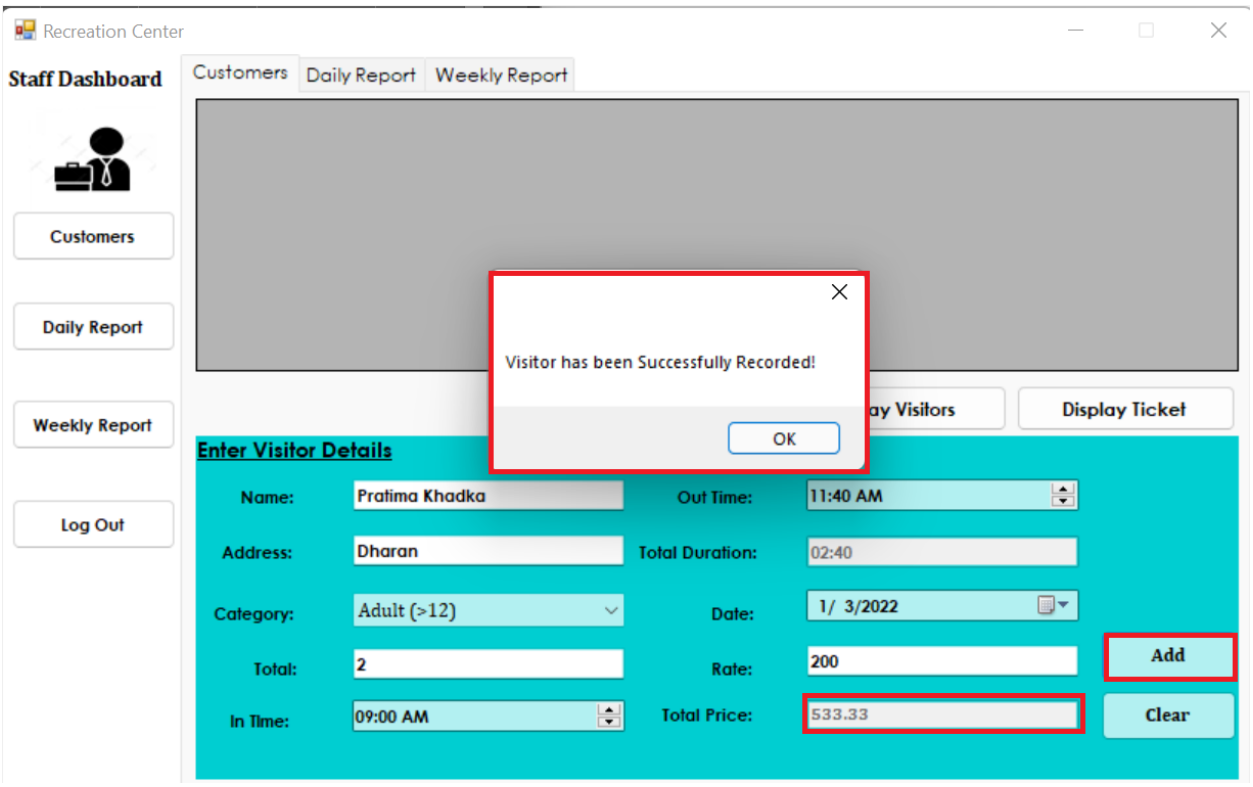

Figure 48: Automatic price calculation for the total duration.

5. Test 5: Displaying Visitors' Details and Ticket prices

Objective	If available, display visitors' details and ticket prices in the grid view.
Action	i.) Press the 'Display Visitors' button by removing and reading the Customer.csv file from the location. ii.) Press the 'Display Ticket' button.
Expected Output	The error message should be displayed if the 'Customer.csv' file is not found, and if found, details should be shown in the grid view. Similarly, the fair ticket price should be displayed by checking todays' date.
Actual Output	The error message was displayed when the 'Customer.csv' file was not found, and details as shown in the grid view when found. Similarly, a reasonable ticket price was revealed.
Result	Test Successful.

Table 11: Test Case 5

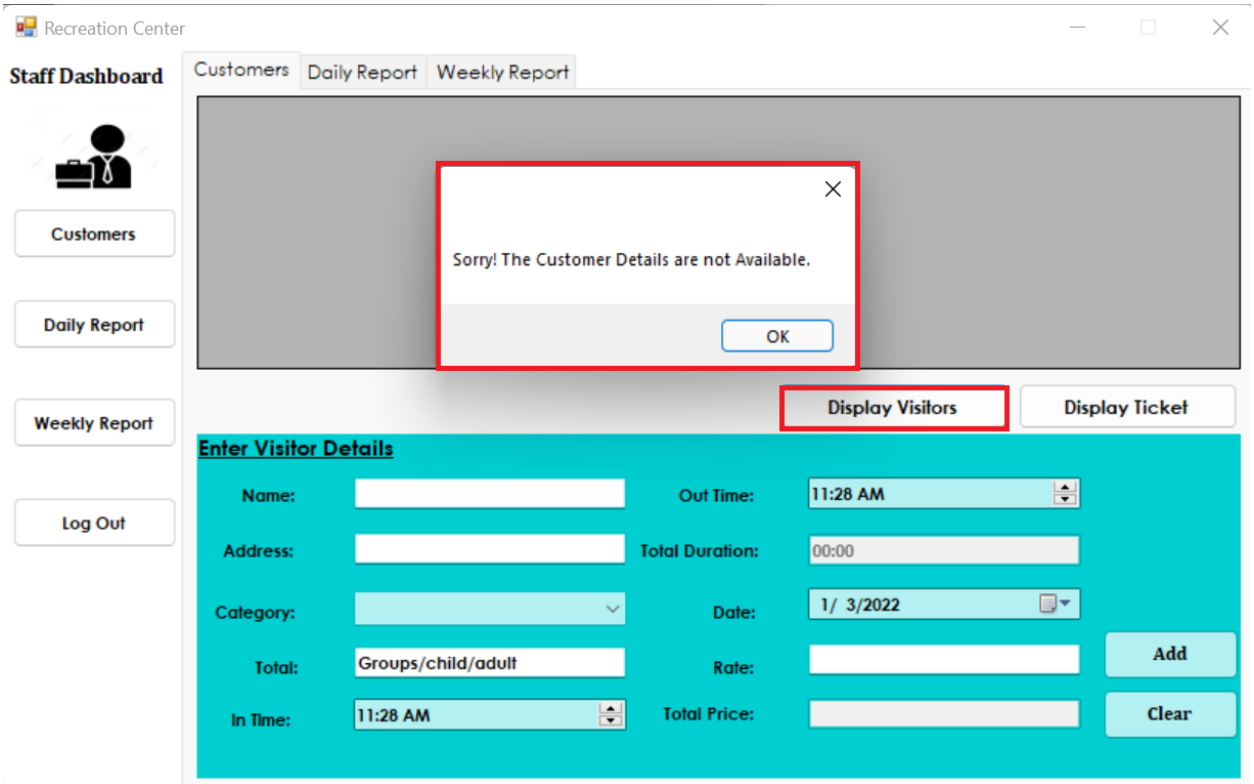

Figure 49: Error message when the file was not found.

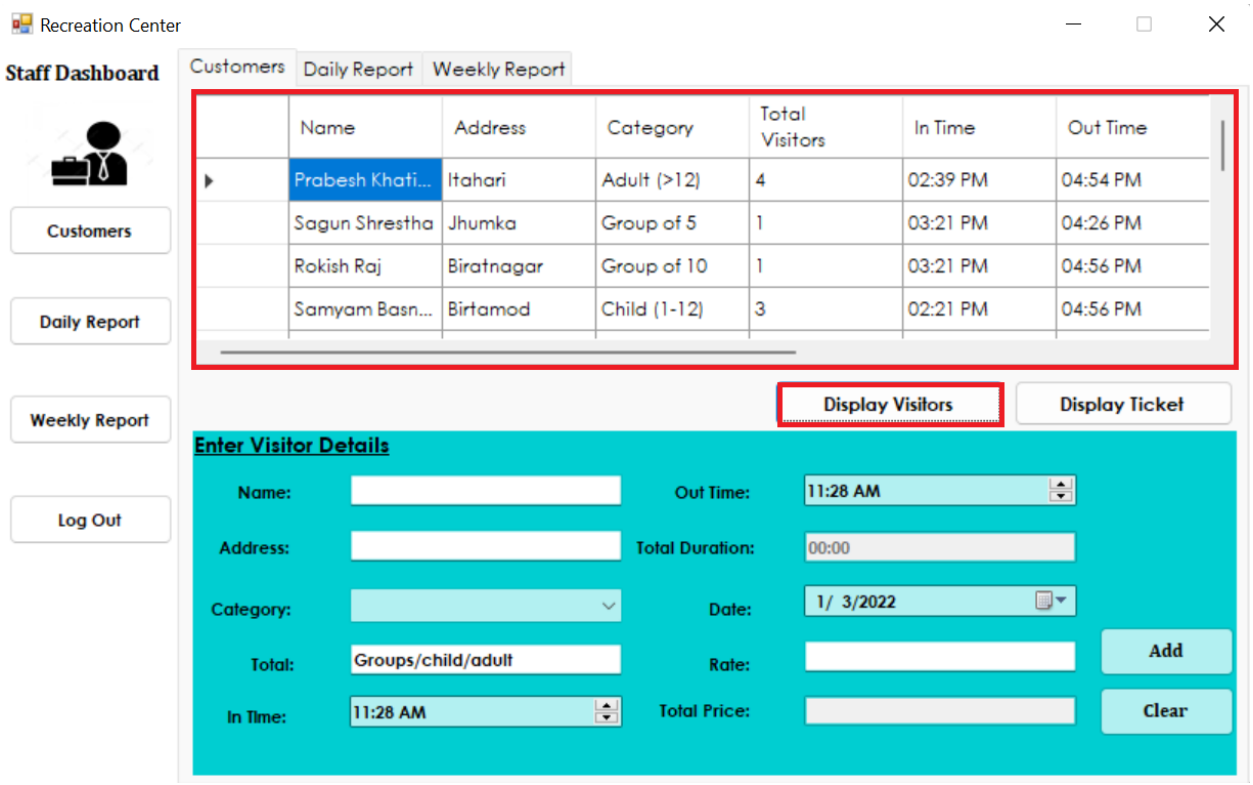

Figure 50: Displaying visitors' detail when the file is found.

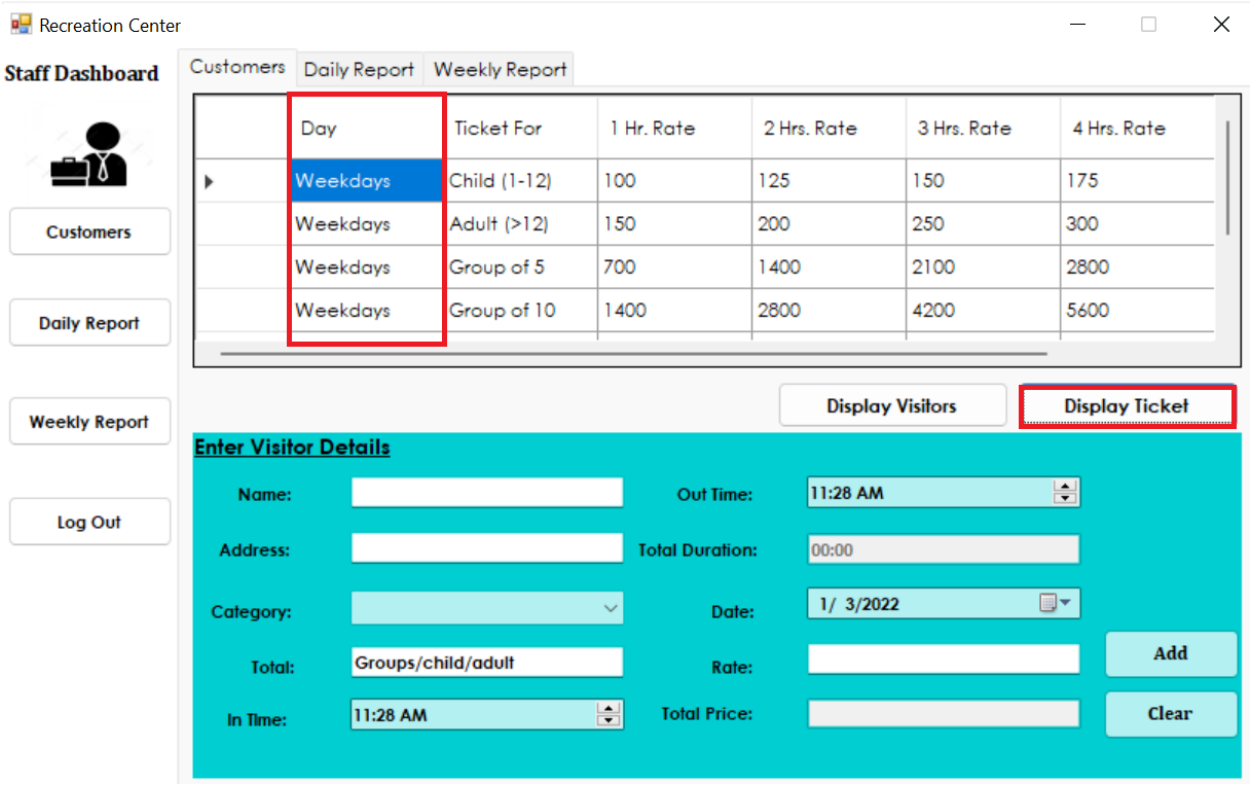

Figure 51: Displaying the ticket prices determining the specific day

6. Test 6: Displaying file into the grid view

Objective	To display the CSV file into the grid view
Action	i.) Press the 'Display Ticket' button and select the appropriate file in the grid view.
Expected Output	The dialog box should appear, and the selected file should be displayed in the grid view.
Actual Output	The dialog box was popped, and the selected file was displayed in the grid view.
Result	Test Successful.

Table 12: Test Case 6

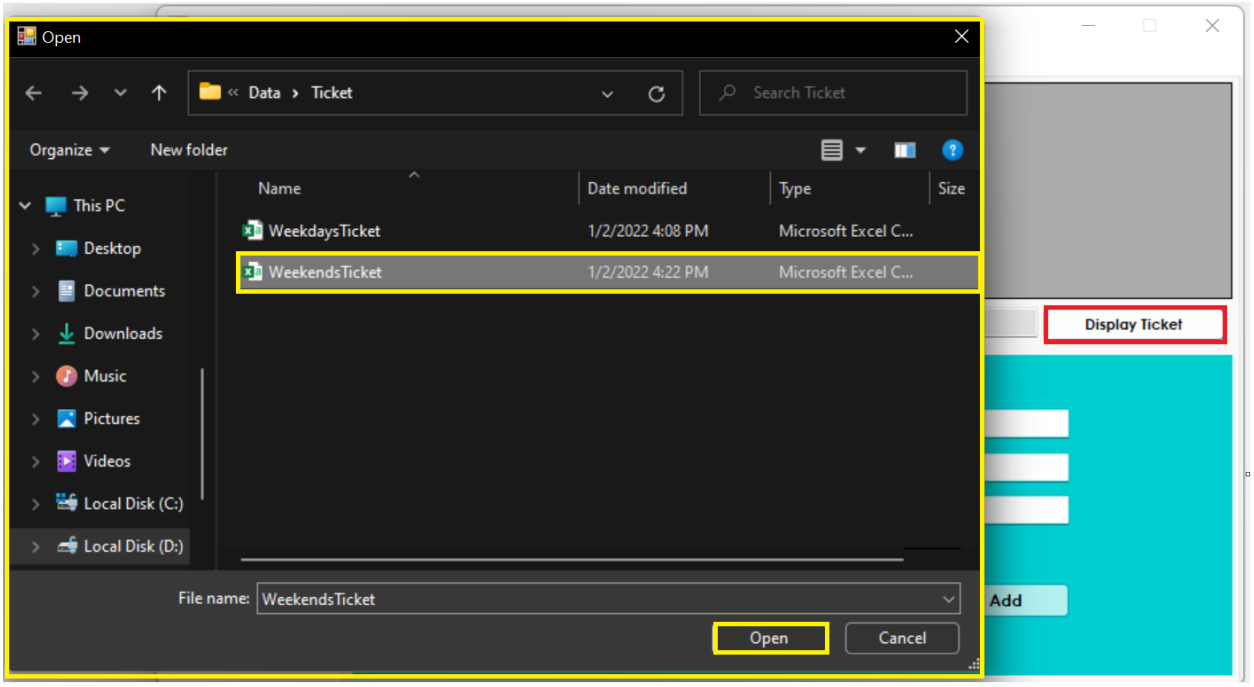

Figure 52: Selecting the CSV file from the dialog box.

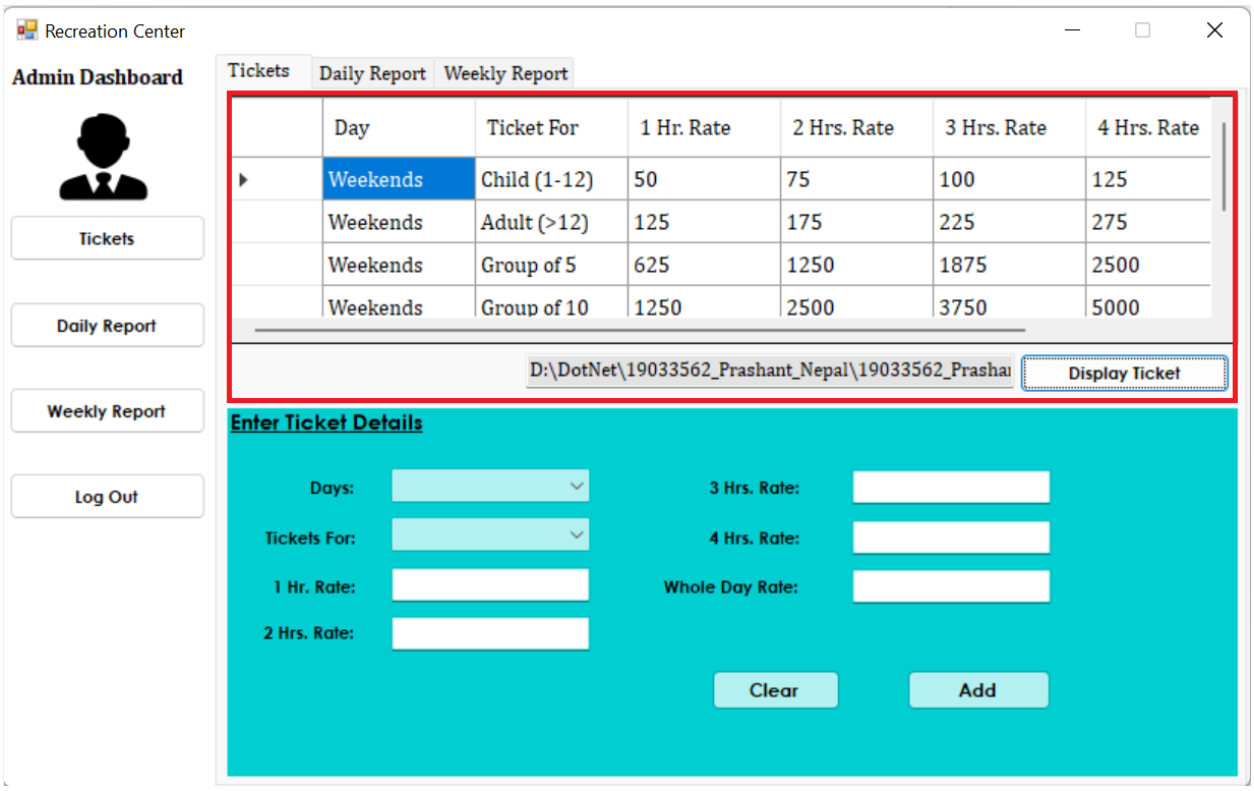

Figure 53: Displaying the selected CSV file in the grid view

7. Test 7: Displaying the daily Report

Objective	To display the daily report in the grid view and the chart
Action	i.) Press the 'Daily Report' button by selecting the appropriate date. ii.) Press the 'View All Details' button.
Expected Output	The daily report data of visitors for specific dates should be displayed. Further, the overall details of visitors should be displayed too.
Actual Output	The daily report data of visitors was displayed. Further, the overall details of visitors were also portrayed.
Result	Test Successful.

Table 13: Test Case 7

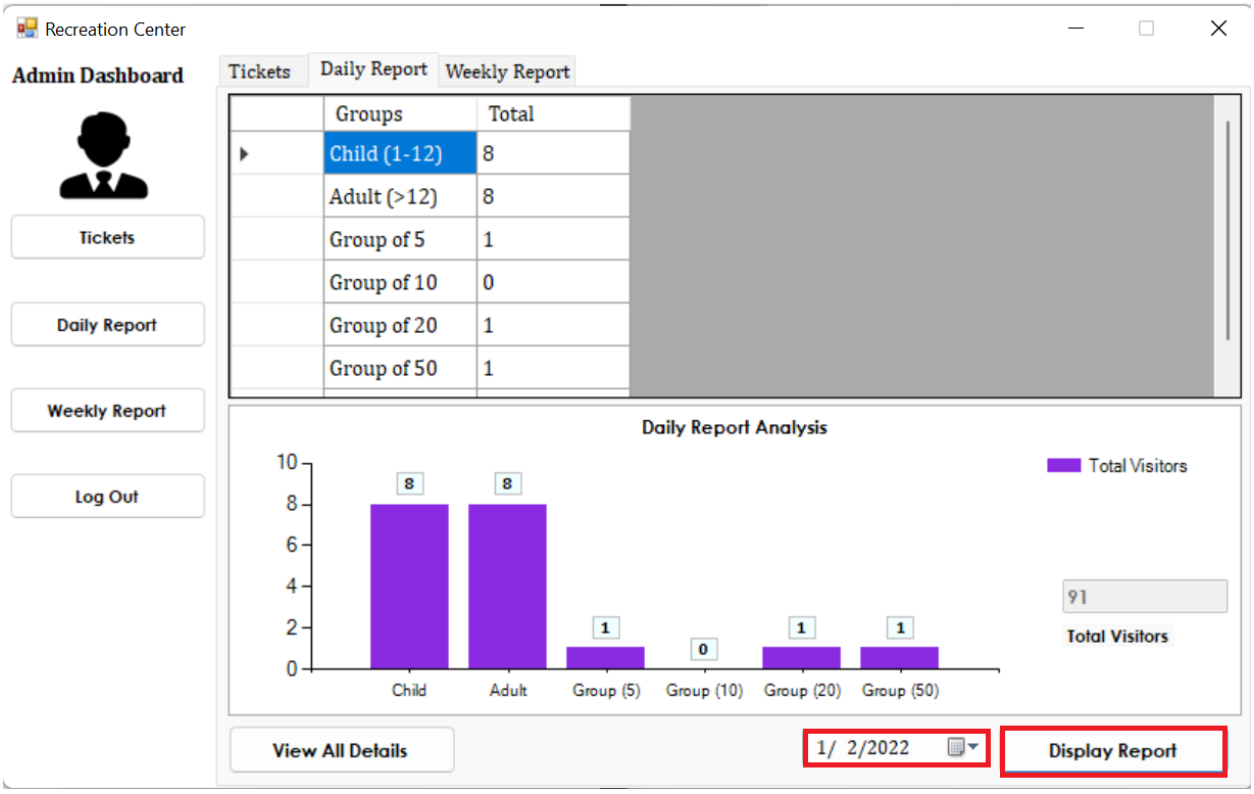

Figure 54: Displaying the daily report for the selected date.

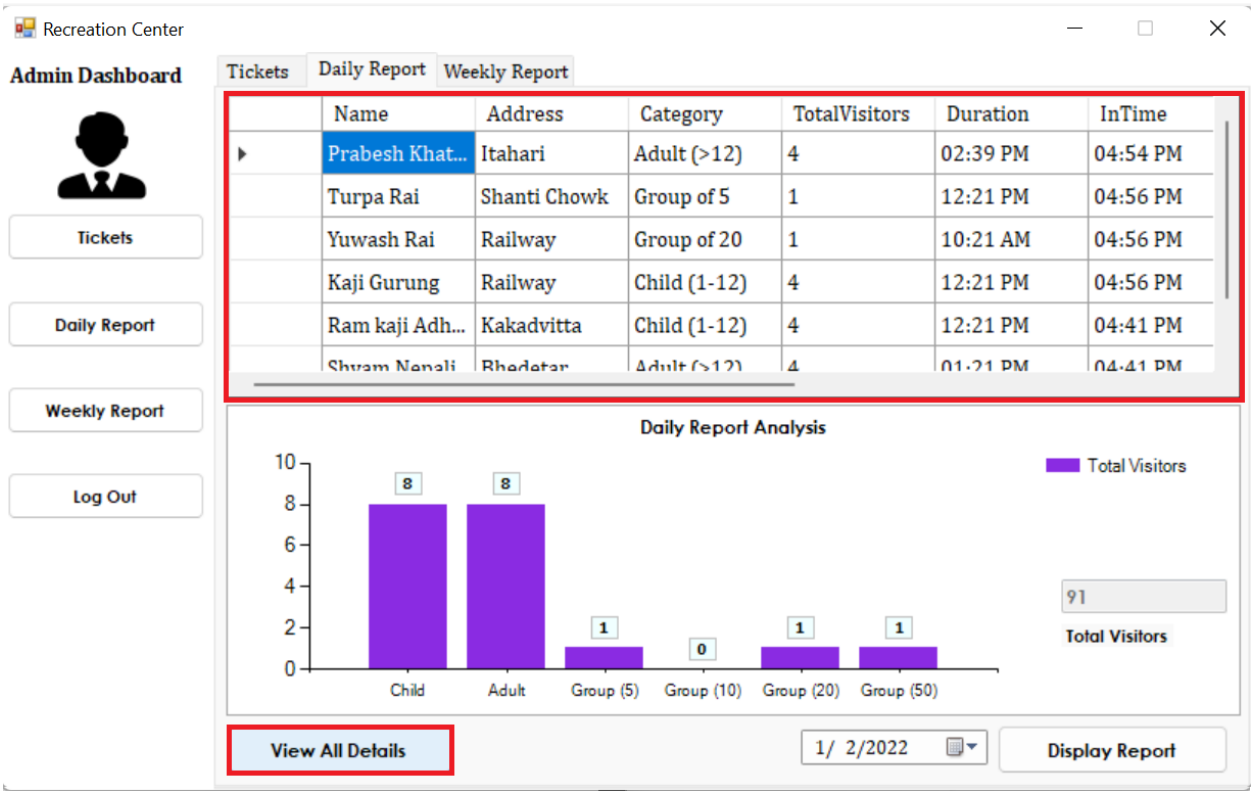

Figure 55: Displaying entire details of visitors

8. Test 8: Viewing the weekly report.

Objective	To view the weekly report data.
Action	i.) The 'Display Report' button should be pressed by selecting the appropriate date.
Expected Output	The weekly report for the selected date should be displayed.
Actual Output	The weekly report for the selected date was displayed.
Result	Test Successful.

Table 14: Test Case 8

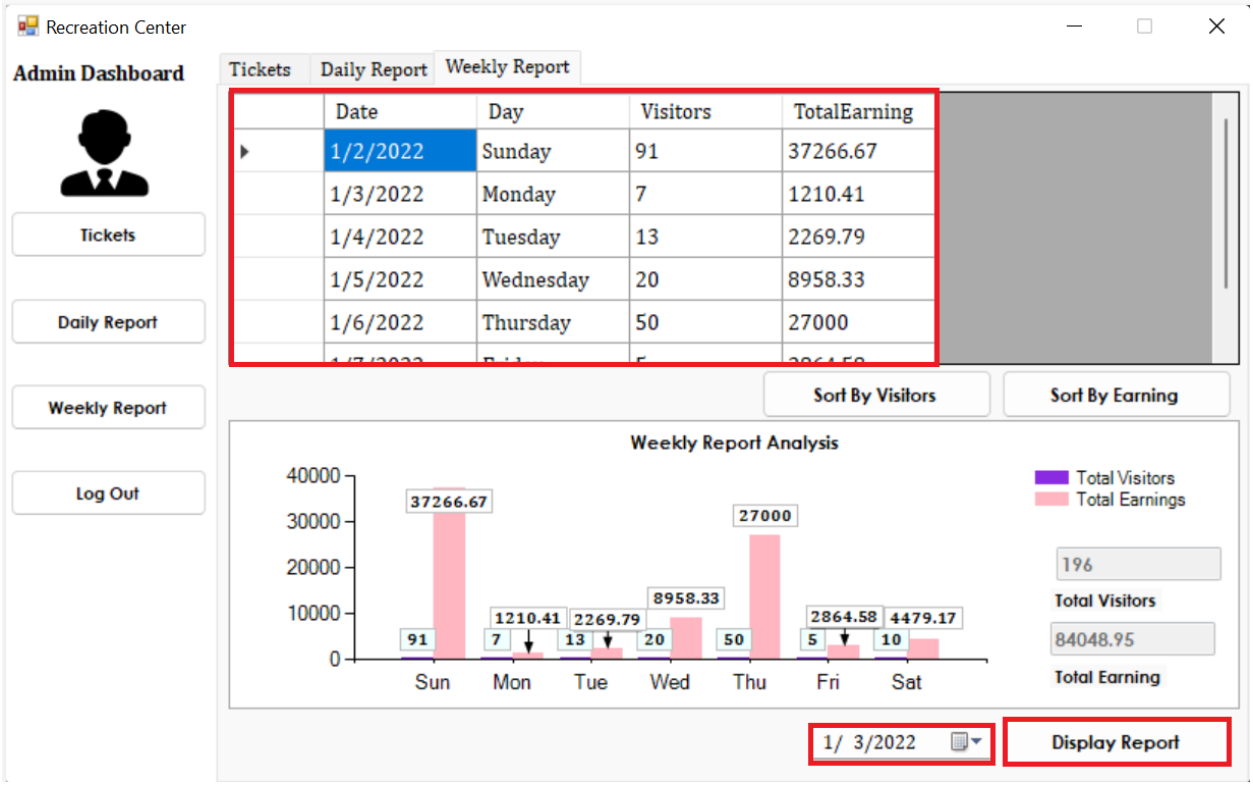

Figure 56: Displaying the weekly report data for the specific date.

9. Test 9: Sorting Data in the empty grid view

Objective	To sort data when the grid view is empty
Action	i.) Press the 'Sort By Visitors' button when the grid view is empty.
Expected Output	The error message should be displayed indicating no data available in the grid.
Actual Output	The error message was displayed.
Result	Test Successful.

Table 15: Test Case 9

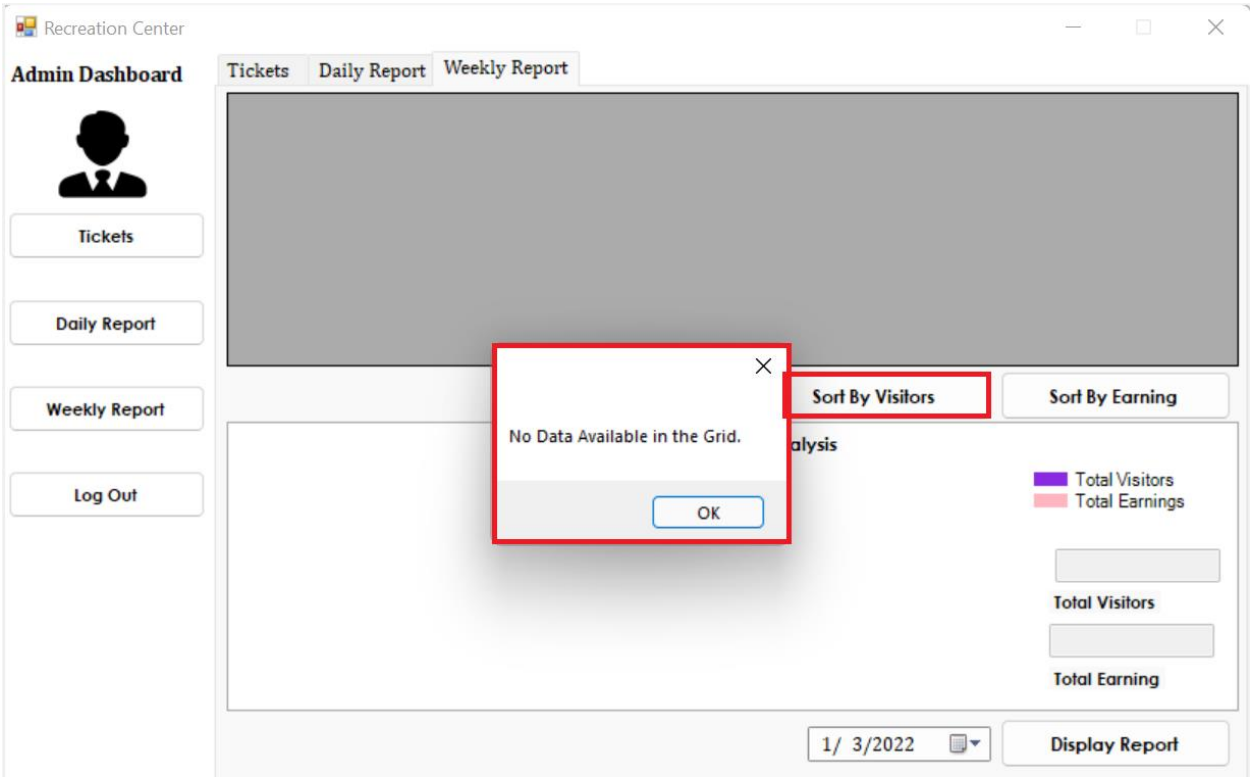

Figure 57: Error message while sorting data for the empty grid view.

10. Test 10: Sorting Data in the grid view

Objective	To sort data of the grid view.
Action	i.) Press the 'Sort By Visitors' button. ii.) Press the 'Sort By Earning' button.
Expected Output	The data from the grid view should be sorted according to earnings and total visitors accordingly.
Actual Output	The data from the grid view was sorted.
Result	Test Successful.

Table 16: : Test Case 10

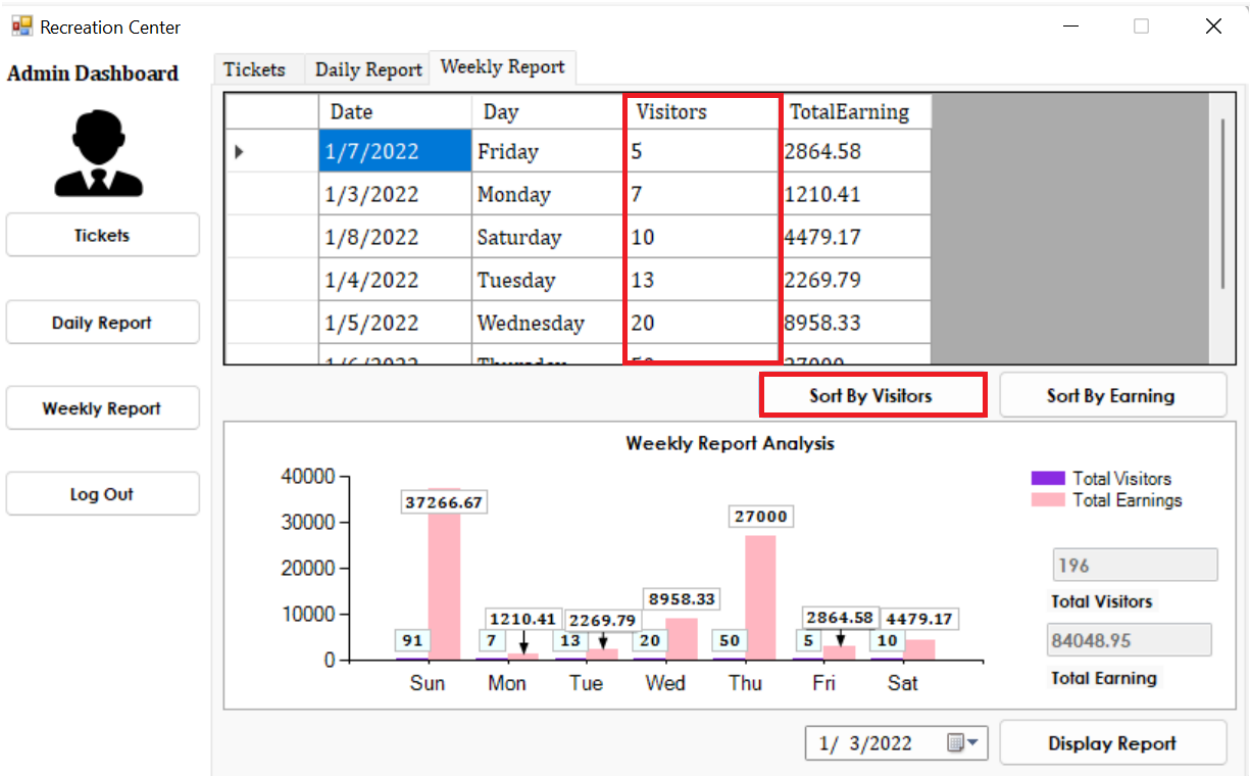

Figure 58: Displaying the sorted data according to total visitors.

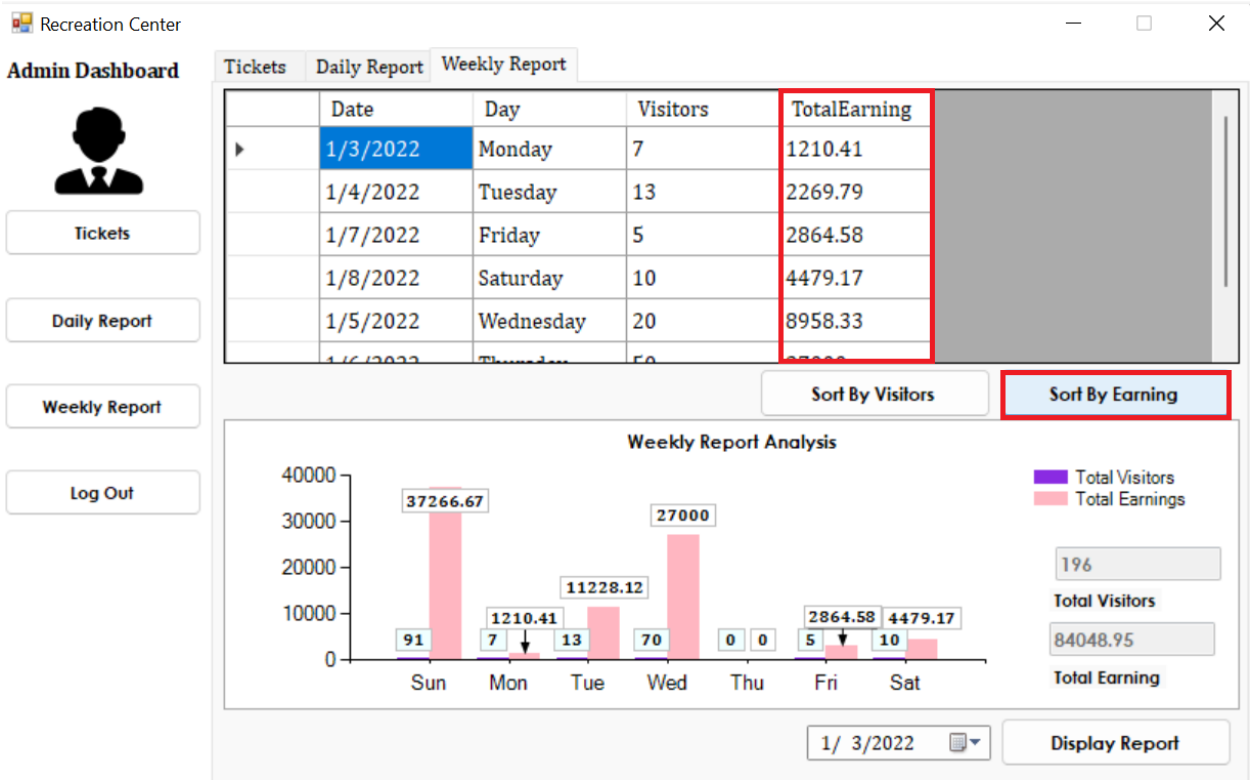

Figure 59: Displaying the sorted weekly data according to total earning

11. Test 11: Viewing details from opened files.

Objective	To display the details from the opened CSV file.
Action	i.) Open the 'Customer.csv' file in excel ii.) Press the 'Display Report' button.
Expected Output	The error message should be displayed indicating that the other application is opening the file.
Actual Output	The error message was displayed.
Result	Test Successful.

Table 17: Test Case 11

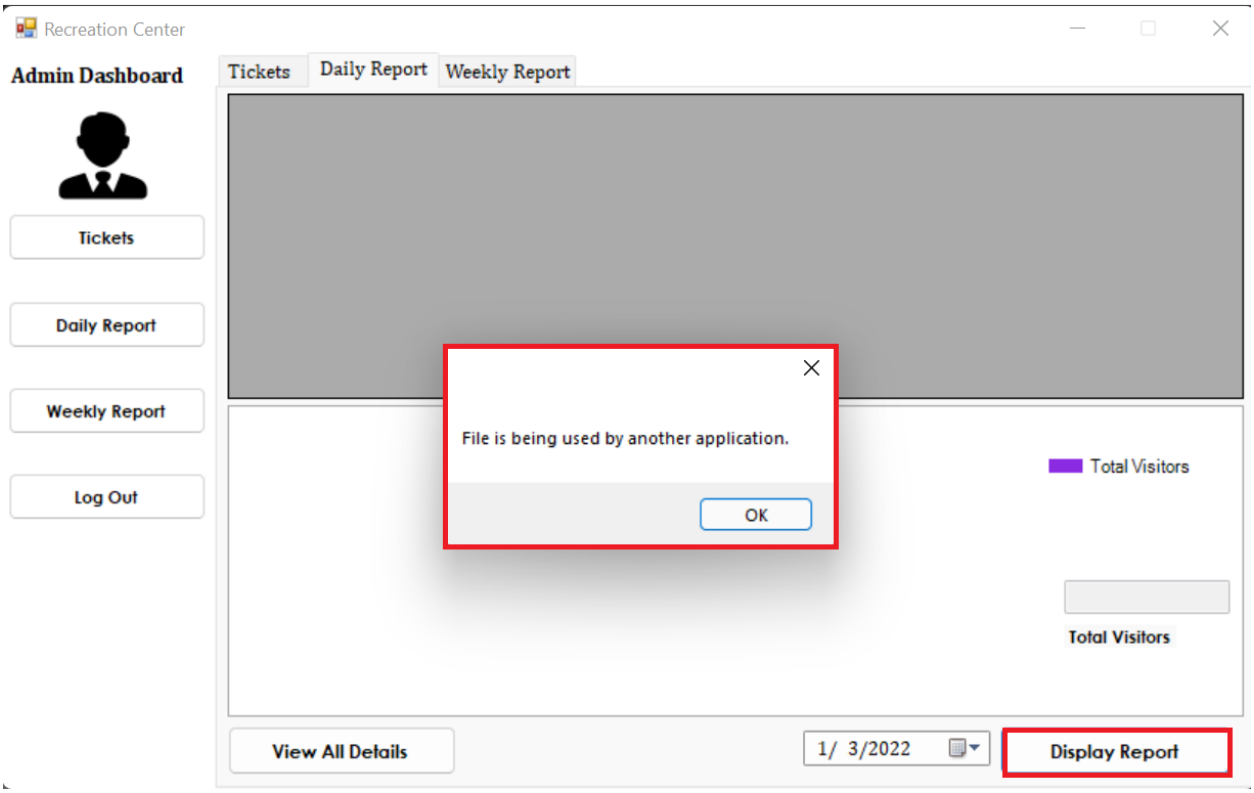

Figure 60: Displaying the error message when the file is opened in another application.

Appendix B (Code)

1. LandingWindow.cs

```
using System;
using System.Windows.Forms;

namespace RecreationCenter
{
    public partial class LandingWindow : Form
    {
        readonly Constants constant = new Constants();

        public LandingWindow()
        {
            InitializeComponent();
            this.MaximizeBox = false; //disabling window to maximizing
            this.AutoSizeMode = System.Windows.Forms.AutoSizeMode.GrowAndShrink;
//disabling window to increase or decrease
        }

        private void Password_TextChanged(object sender, EventArgs e)
        {
            password.PasswordChar = '*'; //changing the typed keywords into '*'
        }

        private Boolean IsEmpty()
        {
            //checking for the empty fields
            if (loginId.Text == "")
            {
                MessageBox.Show("Please! Provide User ID.");
                loginId.Focus();
                return true;
            }
            else if (cmbRole.SelectedIndex == -1)
            {
                MessageBox.Show("Please! Select User Type.");
                cmbRole.Focus();
                return true;
            }
            else if (password.Text == "")
            {
                MessageBox.Show("Please! Provide a Password.");
```

```
                password.Focus();
                return true;
            }
            return false;
        }

        private Boolean CheckPassword()
        {
            //checking for the id and password
            if (loginId.Text == constant.adminID && password.Text ==
constant.adminPw && cmbRole.SelectedIndex == 1)
            {
                adminPanel1.Visible = true; //showing admin panel
                password.Text = ""; //clearing the password field
                return true;
            }
            else if (loginId.Text == constant.staffID && password.Text ==
constant.staffPw && cmbRole.SelectedIndex == 0)
            {
                staffPanel1.Visible = true; //showing the staff panel
                password.Text = "";
                return true;
            }
            return false;
        }

        private void BtnLogin_Click(object sender, EventArgs e)
        {
            if (!IsEmpty()) //checking for empty fields
            {
                if (!CheckPassword()) //checking for password
                {
                    MessageBox.Show("Invalid User Name or Password.");
                }
            }
        }

        private void LandingWindow_FormClosing(object sender,
FormClosingEventArgs e)
        {
            //displaying dialog box before closing the form
            if (MessageBox.Show("Are you sure you want to stop application and
exit?", "Confirm Exit", MessageBoxButtons.YesNoCancel) != DialogResult.Yes)
            {
                e.Cancel = true;
```

```
            }
        }

        private void AdminPanel1_Load(object sender, EventArgs e)
        {
            //initially hiding the both panels
            adminPanel1.Visible = false;
            staffPanel1.Visible = false;
        }
    }
}
```

2. AdminPanel.cs

```
using System;
using System.Windows.Forms;

namespace RecreationCenter
{
    public partial class AdminPanel : UserControl
    {
        readonly Constants constant = new Constants();
        readonly Functions fn = new Functions();
        readonly Report report = new Report();

        public AdminPanel()
        {
            InitializeComponent();
            datePickerDaily.Value = DateTime.Today;
            weeklyDate.Value = DateTime.Today;
        }

        private void BtnLogOut_Click(object sender, EventArgs e)
        {
            if (MessageBox.Show("Are you sure you want to Log Out?", "Log Out",
MessageBoxButtons.YesNo) == DialogResult.Yes)
            {
                this.Visible = false; //hiding the admin panel
            }
        }

        private void BtnTicket_Click(object sender, EventArgs e)
        {
            AdminTabs.SelectedTab = TabTicket;
        }

        private void BtnDailyReport_Click(object sender, EventArgs e)
        {
            AdminTabs.SelectedTab = TabDailyReport;
        }

        private void BtnWeeklyReport_Click(object sender, EventArgs e)
        {
            AdminTabs.SelectedTab = TabWeeklyReport;
        }

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

```
            //clearing the text fields
            cmbDays.SelectedIndex = -1;
            cmbCustomerType.SelectedIndex = -1;
            tfOneHrRate.Text = "";
            tfTwoHrRate.Text = "";
            tfThreeHrRate.Text = "";
            tfFourHrRate.Text = "";
            tfWholeDayRate.Text = "";
        }

        private void DataGridView_DataError(object sender,
DataGridViewDataErrorEventArgs e)
        {
            //handling error that may cause from parsing
            e.Cancel = true;
        }

        private Boolean IsEmpty()
        {
            //checking if the fields are empty
            if (cmbDays.SelectedIndex == -1)
            {
                MessageBox.Show("Please! Provide Days");
                cmbDays.Focus();
                return true;
            }
            else if (cmbCustomerType.SelectedIndex == -1)
            {
                MessageBox.Show("Please! Provide Ticket Type");
                cmbCustomerType.Focus();
                return true;
            }
            else if (tfOneHrRate.Text == "")
            {
                MessageBox.Show("Please! Provide Ticket Rate for 1 Hr.");
                tfOneHrRate.Focus();
                return true;
            }
            else if (tfTwoHrRate.Text == "")
            {
                MessageBox.Show("Please! Provide Ticket Rate for 2 Hrs.");
                tfTwoHrRate.Focus();
                return true;
            }
            else if (tfThreeHrRate.Text == "")
```

```
            {
                MessageBox.Show("Please! Provide Ticket Rate for 3 Hrs.");
                tfThreeHrRate.Focus();
                return true;
            }
            else if (tfFourHrRate.Text == "")
            {
                MessageBox.Show("Please! Provide Ticket Rate for 4 Hrs.");
                tfFourHrRate.Focus();
                return true;
            }
            else if (tfWholeDayRate.Text == "")
            {
                MessageBox.Show("Please! Provide Ticket Rate for Whole Day.");
                tfWholeDayRate.Focus();
                return true;
            }
            return false;
        }

        private void SetDetails()
        {
            //assining data from text fields to the variable
            constant.ticketDetails = cmbDays.Text + "," + cmbCustomerType.Text +
"," + tfOneHrRate.Text + "," +
                                        tfTwoHrRate.Text + "," + tfThreeHrRate.Text
+ "," + tfFourHrRate.Text + "," +
                                        tfWholeDayRate.Text + "\n";
        }

        private void ButtonAdd_Click(object sender, EventArgs e)
        {
            if (!IsEmpty()) //checking if the field is empty
            {
                Double oneHrRate, twoHrRate, threeHrRate, fourHrRate,
wholeDayRate;
                try
                {
                    //validating the rates as double
                    oneHrRate = Double.Parse(tfOneHrRate.Text);
                    twoHrRate = Double.Parse(tfTwoHrRate.Text);
                    threeHrRate = Double.Parse(tfThreeHrRate.Text);
                    fourHrRate = Double.Parse(tfFourHrRate.Text);
                    wholeDayRate = Double.Parse(tfWholeDayRate.Text);
                }
```

```
                catch (Exception)
                {
                    MessageBox.Show("Please! Provide Numerical Values for all
Rates.");
                    tfOneHrRate.Focus();
                    return;
                }

                //checking the range of rates
                if (oneHrRate > 0 && oneHrRate < 10000000)
                {
                    if (twoHrRate > 0 && twoHrRate < 10000000)
                    {
                        if (threeHrRate > 0 && threeHrRate < 10000000)
                        {
                            if (fourHrRate > 0 && fourHrRate < 10000000)
                            {
                                if (wholeDayRate > 0 && wholeDayRate < 10000000)
                                {
                                    SetDetails();
                                    if (cmbDays.SelectedIndex == 0)
                                    {
                                        //calling function for writing on the CSV
file
                                        fn.WriteToCSV(constant.weekdaysTicketLoca
tion, constant.ticketHeaders, constant.ticketDetails);
                                    }
                                    else
                                    {
                                        fn.WriteToCSV(constant.weekendsTicketLoca
tion, constant.ticketHeaders, constant.ticketDetails);
                                    }

                                    MessageBox.Show("Ticket Successfully Added");
                                }
                                else
                                {
                                    MessageBox.Show("Please! Provide Appropriate
Whole Day Rate");
                                    tfWholeDayRate.Focus();
                                }
                            }
                            else
                            {
```

```
                                MessageBox.Show("Please! Provide Appropriate 4
Hrs. Rate");
                                tfFourHrRate.Focus();
                            }
                        }
                        else
                        {
                            MessageBox.Show("Please! Provide Appropriate 3 Hrs.
Rate");
                            tfThreeHrRate.Focus();
                        }
                    }
                    else
                    {
                        MessageBox.Show("Please! Provide Appropriate 2 Hrs.
Rate");
                        tfTwoHrRate.Focus();
                    }
                }
                else
                {
                    MessageBox.Show("Please! Provide Appropriate 1 Hr. Rate");
                    tfOneHrRate.Focus();
                }
            }
        }

        private void BtnDisplayTicket_Click(object sender, EventArgs e)
        {
            try
            {
                //opning dialog box for selecting ticket
                openFileDialogBox.ShowDialog();
                tfFileLocation.Text = openFileDialogBox.FileName;
                _ = new BindData(tfFileLocation.Text, dataGridViewTickets);
            }
            catch (Exception)
            {
                MessageBox.Show(constant.errorFile);
            }
        }

        private void BtnDisplayDailyReport_Click(object sender, EventArgs e)
        {
            //function for clearing the gridview and chart details
```

```
            report.ClearFields(dataGridViewDailyReport, chartDailyReport);

            //function for abstracting data from the CSV file
            report.ReadFromCSV();

            //calling function for displaying data into the chart
            report.DisplayRequiredDailyData(datePickerDaily, tfVisitorsCount,
chartDailyReport);

            //calling function for displaying data into the gridview
            report.ViewDailyData(dataGridViewDailyReport);
        }

        private void BtnViewAllDetails_Click(object sender, EventArgs e)
        {
            report.ClearFields(dataGridViewDailyReport, chartDailyReport);

            report.ReadFromCSV();

            report.DisplayRequiredDailyData(datePickerDaily, tfVisitorsCount,
chartDailyReport);

            //calling function for dispalying overall customer details
            report.DisplayOverallDailyData(dataGridViewDailyReport);
        }

        private void BtnDisplayWeeklyReport_Click(object sender, EventArgs e)
        {
            //calculating weekstart and weekend date of the chosen date
            DateTime weekStart = weeklyDate.Value.AddDays(-
(int)weeklyDate.Value.DayOfWeek);
            DateTime weekEnd = weeklyDate.Value.AddDays(6 -
(int)weeklyDate.Value.DayOfWeek);

            report.ClearFields(dataGridViewWeeklyReport, weeklyReport);
            weeklyReport.Series["Total Earnings"].Points.Clear(); //clearing
series of the chart

            report.ReadFromCSV();

            //calling function to get all data of a specific week
            report.GetRequiredWeeklyData(weekStart, weekEnd);

            //adding data into the list
            report.AddWeeklyData(weekStart, weekEnd);
```

```
            //displaying data into the gridview
            report.DisplayInWeeklyGridView(dataGridViewWeeklyReport);

            //displaying weekly data into the chart
            report.DisplayInChart(weeklyReport, tfTotalVisitors, tfTotalEarning);
        }

        private void BtnSortByVisitors_Click(object sender, EventArgs e)
        {
            //calling function for sorting data
            report.SortByVisitors(dataGridViewWeeklyReport);
        }

        private void BtnSortByEarning_Click(object sender, EventArgs e)
        {
            report.SortByEarning(dataGridViewWeeklyReport);
        }
    }
}
```

3. StaffPanel.cs

```
using System;
using System.IO;
using System.Text.RegularExpressions;
using System.Windows.Forms;

namespace RecreationCenter
{
    public partial class StaffPanel : UserControl
    {
        readonly Constants constant = new Constants();
        readonly Functions fn = new Functions();
        readonly Report report = new Report();

        public StaffPanel()
        {
            InitializeComponent();
            dateTimePicker.Value = DateTime.Today;
            datePickerDaily.Value = DateTime.Today;
            weeklyDate.Value = DateTime.Today;
            OutTimePicker.Value = DateTime.Now;
            InTimePicker.Value = DateTime.Now;
        }

        private void BtnLogOut_Click(object sender, EventArgs e)
        {
            if (MessageBox.Show("Are you sure you want to Log Out?", "Log Out",
MessageBoxButtons.YesNo) == DialogResult.Yes)
            {
                this.Visible = false; //hiding the staff panel
            }
        }

        private void BtnCustomers_Click(object sender, EventArgs e)
        {
            StaffTabs.SelectedTab = TabCustomer;
        }

        private void BtnDailyReport_Click(object sender, EventArgs e)
        {
            StaffTabs.SelectedTab = TabDailyReport;
        }

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

```
            StaffTabs.SelectedTab = TabWeeklyReport;
        }

        private void DataGridView_DataError(object sender,
DataGridViewDataErrorEventArgs e)
        {
            //handling parsing errors
            e.Cancel = true;
        }

        private void TfName_KeyPress(object sender, KeyPressEventArgs e)
        {
            //defining regex for taking alphabets only
            Regex regex = new Regex(constant.nameFieldRegx);

            //disabling from typing if matched regular expression
            if (!regex.IsMatch(e.KeyChar.ToString()))
            {
                e.Handled = true;
            }
        }

        private void BtnClear_Click(object sender, EventArgs e)
        {
            //clearing the text fields
            tfName.Text = "";
            tfAddress.Text = "";
            cmbVisitorCategory.SelectedIndex = -1;
            InTimePicker.Value = DateTime.Now;
            OutTimePicker.Value = DateTime.Now;
            tfTotalDuration.Text = "";
            tfTotalVisitor.Text = "Groups/child/adult";
            tfRate.Text = "";
            tfTotalPrice.Text = "";
            dateTimePicker.Value = DateTime.Today;
        }

        private Boolean IsEmpty()
        {
            //checking if any field is empty
            if (tfName.Text == "")
            {
                MessageBox.Show("Please! Provide Visitor Name.");
                tfName.Focus();
                return true;
```

```
            }
            else if (tfAddress.Text == "")
            {
                MessageBox.Show("Please! Provide Visitor's Address.");
                tfAddress.Focus();
                return true;
            }
            else if (cmbVisitorCategory.SelectedIndex == -1)
            {
                MessageBox.Show("Please! Select Visitor Category");
                cmbVisitorCategory.Focus();
                return true;
            }
            else if (tfTotalVisitor.Text == "")
            {
                MessageBox.Show("Please! Provide Total Number of Visitors.");
                tfTotalVisitor.Focus();
                return true;
            }
            else if (tfTotalDuration.Text == "" || tfTotalDuration.Text ==
"00:00")
            {
                MessageBox.Show("Please! Adjust Visitors In and Out Time.");
                InTimePicker.Focus();
                return true;
            }
            else if (tfRate.Text == "")
            {
                MessageBox.Show("Please! Provide Ticket Rate.");
                tfRate.Focus();
                return true;
            }
            return false;
        }

        private void SetDetails()
        {
            //assigning values from the field to the string
            constant.customerDetails = tfName.Text + "," + tfAddress.Text + "," +
cmbVisitorCategory.Text + "," + tfTotalVisitor.Text + "," +
                                       InTimePicker.Text + "," +
OutTimePicker.Text + "," + tfTotalDuration.Text + "," + dateTimePicker.Text + ","
+
                                       tfRate.Text + "," + tfTotalPrice.Text +
"\n";
```

```
        }

        private void OutTimePicker_ValueChanged(object sender, EventArgs e)
        {
            //validating the entry and the exit time
            if (OutTimePicker.Value > InTimePicker.Value)
            {
                //displaying the difference of time in the text field
                tfTotalDuration.Text = (OutTimePicker.Value -
InTimePicker.Value).ToString(@"hh\:mm");
            }
            else
            {
                tfTotalDuration.Text = "00:00";
            }
        }

        private void InTimePicker_ValueChanged(object sender, EventArgs e)
        {
            if (OutTimePicker.Value > InTimePicker.Value)
            {
                tfTotalDuration.Text = (OutTimePicker.Value -
InTimePicker.Value).ToString(@"hh\:mm");
            }
            else
            {
                tfTotalDuration.Text = "00:00";
            }
        }

        public double CalculateTotalPrice(double rate, int totalVisitors)
        {
            //getting first two characters from string as hour
            int hour = Int32.Parse(tfTotalDuration.Text.Substring(0, 2));

            //getting last two caracters from the string as minute
            int minute = Int32.Parse(tfTotalDuration.Text.Substring(3));
            double standardRate = rate * totalVisitors;

            //checking if minute and hours for adding rates
            if (minute > 0 && hour < 5 && hour > 0)
            {
                //formula for calculating amount for total spent minutes
                double minuteRate = (minute / (double)(hour * 60)) * rate *
totalVisitors;
```

```
                return Math.Round(standardRate + minuteRate, 2); //rounding
double to two decimal points
            }
            return standardRate;
        }

        private void BtnAddVisitor_Click(object sender, EventArgs e)
        {
            if (!IsEmpty())
            {
                int totalVisitors;
                double rate;

                try
                {
                    //converting string to int
                    totalVisitors = Int16.Parse(tfTotalVisitor.Text);

                    try
                    {
                        //converting to double
                        rate = Double.Parse(tfRate.Text);

                        //checking for valid number of visitors
                        if (totalVisitors > 0 && totalVisitors < 2000)
                        {
                            //checking for the range of rates
                            if (rate > 0 && rate < 1000000)
                            {
                                double total = CalculateTotalPrice(rate,
totalVisitors);

                                tfTotalPrice.Text = total.ToString();

                                SetDetails();

                                //writing data into the CSV file
                                fn.WriteToCSV(constant.customerFileLocation,
constant.customerHeaders, constant.customerDetails);
                                MessageBox.Show("Visitor has been Successfully
Recorded!");
                            }
                            else
                            {
                                MessageBox.Show("Please! Provide Appropriate
Ticket Rate.");
```

```
                            tfRate.Focus();
                        }
                    }
                    else
                    {
                        MessageBox.Show("Please! Provide Appropriate Number
of Visitors");
                        tfTotalVisitor.Focus();
                    }
                }
                catch (Exception)
                {
                    MessageBox.Show("Please! Provide Numerical Value for
Ticket Rate.");
                    tfRate.Focus();
                }
            }
            catch (Exception)
            {
                MessageBox.Show("Please! Provide Numerical Value for Number
of Visitors.");
                tfTotalVisitor.Focus();
            }
        }
    }

    private void BtnDisplayTickets_Click(object sender, EventArgs e)
    {
        try
        {
            if (File.Exists(constant.weekdaysTicketLocation) ||
File.Exists(constant.weekendsTicketLocation)) //checking for tickets
            {
                if (DateTime.Now.ToString("ddd") == "Sat" ||
DateTime.Now.ToString("ddd") == "Sun") //checking for weekends ticket
                {
                    if (File.Exists(constant.weekendsTicketLocation))
                    {
                        //displaying weekends ticket if found
                        _ = new BindData(constant.weekendsTicketLocation,
dataGridView);
                        return;
                    }
                    else
                    {
```

```
                            MessageBox.Show("Sorry! The Weekends Ticket Prices
are not Available.");
                        }
                    }
                    //dispalying weekdays ticket
                    _ = new BindData(constant.weekdaysTicketLocation,
dataGridView);
                }
                else
                {
                    MessageBox.Show("Ticket Prices are Currently Not
Available.");
                }
            }
            catch (Exception)
            {
                MessageBox.Show(constant.errorFile);
            }
        }

        private void ButtonDisplayVIsitors_Click(object sender, EventArgs e)
        {
            try
            {
                //checking for the file location
                if (File.Exists(constant.customerFileLocation))
                {
                    //displaying data into gridview
                    _ = new BindData(constant.customerFileLocation,
dataGridView);
                }
                else
                {
                    MessageBox.Show(constant.errorCustomer);
                }
            }
            catch (Exception)
            {
                MessageBox.Show(constant.errorFile);
            }
        }

        private void TfOnClick(object sender, EventArgs e)
        {
            //clearing the text field
```

```
            tfTotalVisitor.Clear();
        }

        private void BtnDisplayDailyReport_Click(object sender, EventArgs e)
        {
            //function for clearing the gridview and chart details
            report.ClearFields(dataGridViewDailyReport, chartDailyReport);

            //function for abstracting data from the CSV file
            report.ReadFromCSV();

            //calling function for displaying data into the chart
            report.DisplayRequiredDailyData(datePickerDaily, tfVisitorsCount,
chartDailyReport);

            //calling function for displaying data into the gridview
            report.ViewDailyData(dataGridViewDailyReport);
        }

        private void BtnViewAllDetails_Click(object sender, EventArgs e)
        {
            report.ClearFields(dataGridViewDailyReport, chartDailyReport);

            report.ReadFromCSV();

            report.DisplayRequiredDailyData(datePickerDaily, tfVisitorsCount,
chartDailyReport);

            //calling function for dispalying overall customer details
            report.DisplayOverallDailyData(dataGridViewDailyReport);
        }

        private void BtnDisplayWeeklyReport_Click(object sender, EventArgs e)
        {
            //calculating weekstart and weekend date of the chosen date
            DateTime weekStart = weeklyDate.Value.AddDays(-
(int)weeklyDate.Value.DayOfWeek);
            DateTime weekEnd = weeklyDate.Value.AddDays(6 -
(int)weeklyDate.Value.DayOfWeek);

            report.ClearFields(dataGridViewWeeklyReport, weeklyReport);
            weeklyReport.Series["Total Earnings"].Points.Clear(); //clearing
series of the chart

            report.ReadFromCSV();
```

```
            //calling function to get all data of a specific week
            report.GetRequiredWeeklyData(weekStart, weekEnd);

            //adding data into the list
            report.AddWeeklyData(weekStart, weekEnd);

            //displaying data into the gridview
            report.DisplayInWeeklyGridView(dataGridViewWeeklyReport);

            //displaying weekly data into the chart
            report.DisplayInChart(weeklyReport, tfTotalVisitors, tfTotalEarning);
        }

        private void BtnSortByVisitors_Click(object sender, EventArgs e)
        {
            //calling the function for sorting the data
            report.SortByVisitors(dataGridViewWeeklyReport);
        }

        private void BtnSortByEarning_Click(object sender, EventArgs e)
        {
            report.SortByEarning(dataGridViewWeeklyReport);
        }
    }
}
```

4. BindData.cs

```
using System;
using System.Data;
using System.IO;
using System.Windows.Forms;

namespace RecreationCenter
{
    class BindData
    {
        public BindData(string filePath, DataGridView dataGrid)
        {
            DataTable dt = new DataTable(); //creating new instance of DataTable

            string[] lines = File.ReadAllLines(filePath); //getting all lines of
the CSV file

            if (lines.Length > 0)
            {
                //creating the first line of CSV file as headers
                string firstLine = lines[0];
                string[] headers = firstLine.Split(',');

                foreach (string header in headers)
                {
                    dt.Columns.Add(new DataColumn(header)); //adding headers to
the column
                }

                //looping through each line of the CSV file and adding row to the
DataTable
                for (int i = 1; i < lines.Length; i++)
                {
                    string[] dataWords = lines[i].Split(',');
                    DataRow dr = dt.NewRow();
                    int columnIndex = 0;
                    foreach (String header in headers)
                    {
                        dr[header] = dataWords[columnIndex++];
                    }
                    dt.Rows.Add(dr);
                }
            }
            if (dt.Rows.Count > 0)
            {
```

```
                dataGrid.DataSource = dt; //displaying the DataTable in the
gridview
            }
        }
    }
}
```

5. Customer.cs

```
using System;

namespace RecreationCenter
{
    class Customer
    {
        //creating private variables with getter and setter methods
        public string Name { set; get; }
        public string Address { set; get; }
        public string Category { set; get; }
        public int TotalVisitors { set; get; }
        public string Duration { set; get; }
        public string InTime { set; get; }
        public string OutTime { set; get; }
        public DateTime Date { set; get; }
        public double Rate { set; get; }
        public double TotalPrice { set; get; }

        public Customer(string name, string address, string category, int
totalVisitors, string duration, string intTime, string outTIme, DateTime date,
double rate, double totalPrice)
        {
            //initializing the variables through the constructor
            this.Name = name;
            this.Address = address;
            this.Category = category;
            this.TotalVisitors = totalVisitors;
            this.Duration = duration;
            this.InTime = intTime;
            this.OutTime = outTIme;
            this.Date = date;
            this.Rate = rate;
            this.TotalPrice = totalPrice;
        }

    }
}
```

6. WeeklyReport.cs

```
using System;

namespace RecreationCenter
{
    class WeeklyReport
    {
        //creating private variables with getter and setter methods
        public DateTime Date { get; set; }
        public string Day { get; set; }
        public int Visitors { get; set; }
        public double TotalEarning { get; set; }

        public WeeklyReport(DateTime date, string daysOfWeeks, int
numberOfVisitors, double totalEarning)
        {
            //initializing the variables through the constructor
            this.Date = date;
            this.Day = daysOfWeeks;
            this.Visitors = numberOfVisitors;
            this.TotalEarning = totalEarning;
        }
    }
}
```

7. Constants.cs

```
using System;
using System.Collections.Generic;
using System.IO;

namespace RecreationCenter
{
    class Constants
    {
        //initializing user id and password for admin and staff
        public readonly string adminID = "admin";
        public readonly string staffID = "staff";
        public readonly string adminPw = "admin";
        public readonly string staffPw = "staff";

        //defining regular expression for validating name text box
        public readonly string nameFieldRegx = @"[A-Za-z\s\b]";

        //getting relative path for the project folder
        public static string folderPath =
Path.GetFullPath(Path.Combine(Environment.CurrentDirectory, @"..\..\Data\"));

        //providing relative pathname for each of the CSV file
        public readonly string weekendsTicketLocation = folderPath +
@"\Ticket\WeekendsTicket.csv";
        public readonly string weekdaysTicketLocation = folderPath +
@"\Ticket\WeekdaysTicket.csv";
        public readonly string customerFileLocation = folderPath +
@"\Customer\Customers.csv";

        //defining headers
        public readonly string customerHeaders = "Name" + "," + "Address" + "," +
"Category" + "," + "Total Visitors" + "," +
                                                 "In Time" + "," + "Out Time" +
"," + "Duration" + "," + "Date" + "," +
                                                 "Rate" + "," + "Total Price" +
Environment.NewLine;

        public readonly string ticketHeaders = "Day" + "," + "Ticket For" + "," +
"1 Hr. Rate" + "," + "2 Hrs. Rate" + "," +
                                               "3 Hrs. Rate" + "," + "4 Hrs.
Rate" + "," + "Whole Day Rate" + Environment.NewLine;

        public string customerDetails;
        public string ticketDetails;
```

```
        //initializing lists
        public List<Customer> customerList = new List<Customer>();
        public List<Customer> requiredList = new List<Customer>();
        public List<WeeklyReport> weeklyReportList = new List<WeeklyReport>();

        //initializing dictionary
        public Dictionary<string, int> dailyReport = new Dictionary<string,
int>();

        //defining error messages
        public readonly string errorMessage = "Sorry! The Customer File has been
Edited.";
        public readonly string errorFile = "File is being used by another
application.";
        public readonly string errorCustomer = "Sorry! The Customer Details are
not Available.";
    }
}
```

8. Functions.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Windows.Forms;

namespace RecreationCenter
{
    class Functions
    {
        readonly Constants constant = new Constants();
        public void WriteToCSV(string path, string headers, string details)
        {
            try
            {
                if (!File.Exists(path))
                {
                    //writing header in the CSV file
                    File.WriteAllText(path, headers);
                }
                File.AppendAllText(path, details); //appending data in the file
            }
            catch (Exception)
            {
                MessageBox.Show("File is being used by another application.");
            }
        }

        public Customer AddData(string line)
        {
            try
            {
                string[] details = line.Split(','); //removing comma and adding
to line into array

                //creating instance of Customer
                Customer data = new Customer(
                                details[0],
                                details[1],
                                details[2],
                                Convert.ToInt32(details[3]),
                                details[4],
                                details[5],
                                details[6],
                                Convert.ToDateTime(details[7]),
```

```
                                    Convert.ToDouble(details[8]),
                                    Convert.ToDouble(details[9])
                                    );
                return data;
            }
            catch (Exception)
            {
                MessageBox.Show(constant.errorMessage);
                throw new Exception(constant.errorMessage);
            }
        }

        public void Sort(List<WeeklyReport> weekData, string sortBy)
        {
            QuickSort(weekData, 0, weekData.Count - 1, sortBy);//calling
Quicksort function
        }

        private void QuickSort(List<WeeklyReport> weekData, int left, int right,
string sortBy)
        {
            if (left < right)
            {
                //calculating pivot point by partioning list
                int pivotPoint = Partition(weekData, left, right, sortBy);

                //calling function recursively
                QuickSort(weekData, left, pivotPoint - 1, sortBy);
                QuickSort(weekData, pivotPoint + 1, right, sortBy);
            }
        }

        private dynamic AccessData(WeeklyReport list, string sortBy)
        {
            //returning either int or double according to the string parameter
            return typeof(WeeklyReport).GetProperty(sortBy).GetValue(list);
        }

        private int Partition(List<WeeklyReport> weekData, int left, int right,
string sortBy)
        {
            dynamic pivot = AccessData(weekData[right], sortBy); //dynamically
accessing data from the list
            int i = left;
            for (int j = left; j <= right - 1; j++)
```

```
            {
                if (AccessData(weekData[j], sortBy) < pivot) //comparing value
between two indices of the list
                {
                    //swapping objects
                    var temp = weekData[i];
                    weekData[i] = weekData[j];
                    weekData[j] = temp;
                    i++;
                }
            }
            //swapping objects
            var tmp = weekData[right];
            weekData[right] = weekData[i];
            weekData[i] = tmp;
            return i;
        }

        public void SortDataInGrid(List<WeeklyReport> weekData, string
sortingProperty, DataGridView gridView)
        {
            if (gridView.DataSource != null) //checking if gridview is empty
            {
                gridView.DataSource = null; //clearing the gridview

                Sort(weekData, sortingProperty); //calling function for sorting

                gridView.DataSource = weekData; //displaying list into the
gridview
            }
            else
            {
                MessageBox.Show("No Data Available in the Grid.");
            }
        }
    }
}
```

9. Report.cs

```
using System;
using System.IO;
using System.Linq;
using System.Windows.Forms;
using System.Windows.Forms.DataVisualization.Charting;

namespace RecreationCenter
{
    class Report
    {
        readonly Constants constant = new Constants();
        readonly Functions fn = new Functions();
        public void ClearFields(DataGridView gridView, Chart chart)
        {
            //clearing lists
            constant.customerList.Clear();
            constant.requiredList.Clear();
            constant.weeklyReportList.Clear();
            constant.dailyReport.Clear();
            gridView.DataSource = null; //clearing the gridview
            chart.Series["Total Visitors"].Points.Clear(); //clearing the chart
        }

        public void ReadFromCSV()
        {
            try
            {
                if (File.Exists(constant.customerFileLocation))
                {
                    //adding data from csv file into customerList
                    constant.customerList =
File.ReadAllLines(constant.customerFileLocation)
                                            .Skip(1) //skipping the header of csv
file
                                            .Select(line => fn.AddData(line))
//lambda operation to return an instance of customer
                                            .ToList(); //converting to the list
                }
                else
                {
                    MessageBox.Show(constant.errorCustomer);
                }
            }
            catch (Exception)
```

```
            {
                MessageBox.Show(constant.errorFile);
            }
        }

        public void DisplayRequiredDailyData(DateTimePicker dateTimePicker,
TextBox textBox, Chart chart)
        {
            int child = 0;
            int adult = 0;
            int groupofFive = 0;
            int groupofTen = 0;
            int groupofTwenty = 0;
            int groupofFifty = 0;

            foreach (Customer customer in constant.customerList)
            {
                if (customer.Date.Equals(dateTimePicker.Value)) //comparing input
date with date from the list
                {
                    constant.requiredList.Add(customer);

                    //checking the category and summing the value
                    switch (customer.Category)
                    {
                        case "Child (1-12)":
                            child += customer.TotalVisitors;
                            break;
                        case "Adult (>12)":
                            adult += customer.TotalVisitors;
                            break;
                        case "Group of 5":
                            groupofFive += customer.TotalVisitors;
                            break;
                        case "Group of 10":
                            groupofTen += customer.TotalVisitors;
                            break;
                        case "Group of 20":
                            groupofTwenty += customer.TotalVisitors;
                            break;
                        default:
                            groupofFifty += customer.TotalVisitors;
                            break;
                    }
                }
```

```
            }

            //calculating total visitors according to the category
            int totalVisitors = child + adult + groupofFive * 5 + groupofTen * 10
+ groupofTwenty * 20 + groupofFifty * 50;

            //adding daily data into dictionary for displaying in gridview
            constant.dailyReport.Add("Child (1-12)", child);
            constant.dailyReport.Add("Adult (>12)", adult);
            constant.dailyReport.Add("Group of 5", groupofFive);
            constant.dailyReport.Add("Group of 10", groupofTen);
            constant.dailyReport.Add("Group of 20", groupofTwenty);
            constant.dailyReport.Add("Group of 50", groupofFifty);
            constant.dailyReport.Add("Total Visitors", totalVisitors);

            //for displaying total number of visitors
            textBox.Text = (totalVisitors).ToString();

            //for displaying in graph
            chart.Series["Total Visitors"].Points.AddXY("Child", child);
            chart.Series["Total Visitors"].Points.AddXY("Adult", adult);
            chart.Series["Total Visitors"].Points.AddXY("Group (5)",
groupofFive);
            chart.Series["Total Visitors"].Points.AddXY("Group (10)",
groupofTen);
            chart.Series["Total Visitors"].Points.AddXY("Group (20)",
groupofTwenty);
            chart.Series["Total Visitors"].Points.AddXY("Group (50)",
groupofFifty);
        }

        public void ViewDailyData(DataGridView gridView)
        {
            //setting the name for key and value of the dictionary
            var reportData = from line in constant.dailyReport select new {
Groups = line.Key, Total = line.Value };
            gridView.DataSource = reportData.ToArray(); //displaying in gridview
by converting to an array
        }

        public void DisplayOverallDailyData(DataGridView gridView)
        {
            //displaying objects from desiredList in gridview
            gridView.DataSource = constant.requiredList;
        }
```

```
        public void DisplayInWeeklyGridView(DataGridView gridView)
        {
            //displaying report data into the gridview
            gridView.DataSource = constant.weeklyReportList;
        }

        public void GetRequiredWeeklyData(DateTime weekStart, DateTime weekEnd)
        {
            //traversing through the list of Customer
            foreach (Customer customer in constant.customerList)
            {
                //comparing the date with starting and ending week dates
                if (customer.Date >= weekStart && customer.Date <= weekEnd)
                {
                    constant.requiredList.Add(customer);
                }
            }
        }

        public void AddWeeklyData(DateTime weekStart, DateTime weekEnd)
        {
            int trackWeekDay = 0;

            //looping through the each day of the week
            for (DateTime date = weekStart; date.AddDays(trackWeekDay) <=
weekEnd;)
            {
                //getting overall objects of the specific date
                var list_ = constant.requiredList.Where(day =>
day.Date.Equals(date.AddDays(trackWeekDay)));

                if (list_ != null)
                {
                    int numberOfVisitors = 0;
                    double totalEarnings = 0;
                    foreach (Customer customer in list_) //traversing through the
one day data
                    {
                        //summing up total visitors according to the category
                        switch (customer.Category)
                        {
                            case "Child (1-12)":
                                numberOfVisitors += customer.TotalVisitors;
                                break;
```

```
                            case "Adult (>12)":
                                numberOfVisitors += customer.TotalVisitors;
                                break;
                            case "Group of 5":
                                numberOfVisitors += customer.TotalVisitors * 5;
                                break;
                            case "Group of 10":
                                numberOfVisitors += customer.TotalVisitors * 10;
                                break;
                            case "Group of 20":
                                numberOfVisitors += customer.TotalVisitors * 20;
                                break;
                            default:
                                numberOfVisitors += customer.TotalVisitors * 50;
                                break;
                        }
                        totalEarnings += customer.TotalPrice; //summing up the
total daily earnings
                    }

                    //instiating the WeeklyReport class
                    WeeklyReport weeklyReport = new
WeeklyReport(date.AddDays(trackWeekDay),
date.AddDays(trackWeekDay).DayOfWeek.ToString(), numberOfVisitors,
totalEarnings);
                    constant.weeklyReportList.Add(weeklyReport); //adding into
the list of type WeeklyReport
                }
                else
                {
                    WeeklyReport weeklyReport = new
WeeklyReport(date.AddDays(trackWeekDay),
date.AddDays(trackWeekDay).DayOfWeek.ToString(), 0, 0);
                    constant.weeklyReportList.Add(weeklyReport);
                }
                trackWeekDay++;
            }
        }

        public void DisplayInChart(Chart weeklyReport, TextBox visitors, TextBox
earning)
        {
            int sun = 0; double sunEarning = 0;
            int mon = 0; double monEarning = 0;
            int tue = 0; double tueEarning = 0;
```

```
            int wed = 0; double wedEarning = 0;
            int thu = 0; double thuEarning = 0;
            int fri = 0; double friEarning = 0;
            int sat = 0; double satEarning = 0;
            int totalVisitorsCount = 0; double grandTotalEarning = 0;

            //traversing through the overall weeklyReport list
            foreach (WeeklyReport wkReport in constant.weeklyReportList)
            {
                //calculating total visitors and total earnings for each day of
the week
                switch (wkReport.Day)
                {
                    case "Sunday":
                        sun += wkReport.Visitors;
                        sunEarning += wkReport.TotalEarning;
                        break;
                    case "Monday":
                        mon += wkReport.Visitors;
                        monEarning += wkReport.TotalEarning;
                        break;
                    case "Tuesday":
                        tue += wkReport.Visitors;
                        tueEarning += wkReport.TotalEarning;
                        break;
                    case "Wednesday":
                        wed += wkReport.Visitors;
                        wedEarning += wkReport.TotalEarning;
                        break;
                    case "Thursday":
                        thu += wkReport.Visitors;
                        thuEarning += wkReport.TotalEarning;
                        break;
                    case "Friday":
                        fri += wkReport.Visitors;
                        friEarning += wkReport.TotalEarning;
                        break;
                    default:
                        sat += wkReport.Visitors;
                        satEarning += wkReport.TotalEarning;
                        break;
                }
                totalVisitorsCount += wkReport.Visitors;
                grandTotalEarning += wkReport.TotalEarning;
            }
```

```
            //for displaying in graph
            weeklyReport.Series["Total Visitors"].Points.AddXY("Sun", sun);
            weeklyReport.Series["Total Earnings"].Points.AddXY("Sun",
sunEarning);

            weeklyReport.Series["Total Visitors"].Points.AddXY("Mon", mon);
            weeklyReport.Series["Total Earnings"].Points.AddXY("Sun",
monEarning);

            weeklyReport.Series["Total Visitors"].Points.AddXY("Tue", tue);
            weeklyReport.Series["Total Earnings"].Points.AddXY("Tue",
tueEarning);

            weeklyReport.Series["Total Visitors"].Points.AddXY("Wed", wed);
            weeklyReport.Series["Total Earnings"].Points.AddXY("Wed",
wedEarning);

            weeklyReport.Series["Total Visitors"].Points.AddXY("Thu", thu);
            weeklyReport.Series["Total Earnings"].Points.AddXY("Thu",
thuEarning);

            weeklyReport.Series["Total Visitors"].Points.AddXY("Fri", fri);
            weeklyReport.Series["Total Earnings"].Points.AddXY("Fri",
friEarning);

            weeklyReport.Series["Total Visitors"].Points.AddXY("Sat", sat);
            weeklyReport.Series["Total Earnings"].Points.AddXY("Sat",
satEarning);

            //displaying totals in the text field.
            visitors.Text = totalVisitorsCount.ToString();
            earning.Text = grandTotalEarning.ToString();
        }

        public void SortByEarning(DataGridView gridView)
        {
            //passing property to be sorted accordingly
            fn.SortDataInGrid(constant.weeklyReportList, "TotalEarning",
gridView);
        }

        public void SortByVisitors(DataGridView gridView)
        {
            fn.SortDataInGrid(constant.weeklyReportList, "Visitors", gridView);
```

```
        }
    }
}
```