

**Metro Ticket Booking System**

**A Project Report**

**Submitted in Partial fulfilment**

**of the Degree of**

**Bachelors of Computer Applications**

**Supervisor's Name – Dr. Kavita Mittal**

**Submitted By– Yash Maurya**

**Roll. No – 120923062**

**Semester: III**



**Jagan Nath University**

**Bahadurgarh (NCR)**

**(2023-26)**

## **PROJECT CERTIFICATE**

This is to certify that the project report entitled **Metro Ticket Booking System** submitted to **JaganNath University, Bahadurgarh** in partial fulfilment of the requirement for the award of the degree of **BACHELOR OF COMPUTER APPLICATIONS (BCA)**, is an original work carried out by Mr./Ms. **Yash Maurya** EnrolmentNo.:(university) **120923062** under the guidance of

**Dr. Kavita Mittal.**

The matter embodied in this project is a genuine work done by the student and has not been submitted whether to this University or to any other University / Institute for the fulfilment of the requirement of any course of study.

**Name of the Student :**

Yash Maurya

**Name of the Guide :**

Dr. Kavita Mittal

**Signature of the Student**

**Signature of the Guide**

**Enrolment No.:**

120923062

**Date :**

## Acknowledgement

We would like to express our heartfelt gratitude to everyone who has supported us in the successful completion of our **Metro Ticket Booking System** project.

First and foremost, we extend our sincere thanks to our project advisor, **Dr. Kavita Mittal**, whose guidance, expertise, and constructive feedback have been invaluable throughout the development process. Your encouragement and insightful suggestions have greatly contributed to the success of this project.

We are also deeply grateful to our Head of Department **Dr. Mohit Mathur**, for their unwavering support and encouragement. Your leadership and commitment to excellence have been a constant source of inspiration and motivation for us.

Additionally, we appreciate the valuable insights and support from our colleagues and peers, especially during the testing phase. Their collaborative spirit and willingness to assist have significantly enhanced the quality of our work.

We would like to acknowledge the assistance provided by the **JaganNath University** library and technical support staff. Their access to relevant resources and technical assistance was essential for overcoming various challenges during the project.

Finally, we would like to express our appreciation to our families and friends for their unwavering support and encouragement throughout this endeavor.

Thank you all for your contributions and support. This project would not have been possible without your involvement.

## Index

S.No	Topic	Page No.
1.	Introduction	1
2.	Features	3
3.	Tools/Environment	5
4.	Program code (Metro Ticket Booking System Source Code)	8
5.	Test Case and Validation	20
6.	Input and Output Screens	24
7.	Limitations of the Project	28
8.	Future Application of the Project	31

# Metro Ticket Booking System

## Introduction

**Project Title:** Metro Ticket Booking System

## Overview

The Metro Ticket Booking System is a comprehensive application designed to facilitate the booking of metro tickets, provide real-time information about metro operations, and manage booking records. This system aims to streamline the ticket booking process for users while offering additional functionalities such as timetable information, station details, and contact information for customer support. Additionally, it provides administrative capabilities for managing and reviewing booking records.

## Objectives

**User-Friendly Interface:** The application offers an intuitive interface for users to book metro tickets, view station details, and access timetable information.

**Booking Management:** Users can book tickets for different metro routes and journey types (one-way or return), with real-time fare calculation based on the chosen route.

**Timetable and Station Information:** The system provides up-to-date information about metro timetables and available stations.

**Administrative Functions:** The application includes an admin login feature that allows authorized personnel to view all booking records and manage the system effectively.

**Data Persistence:** Booking records are saved to and loaded from a file, ensuring that user data is preserved across sessions.

## Basic Functionality

- 1. Booking Functionality:** Users can select source and destination stations, specify the number of tickets, and choose the type of journey (one-way or return).
  - Fare calculation is automated based on the selected route and journey type.
  - Booking details are saved with a timestamp for future reference.**Timetable Information:** Provides detailed information about train frequencies, service times, and travel durations between stations.
- 2. Contact Information:** Displays contact details for customer support, including phone numbers, email addresses, and physical addresses.

3. **Admin Login:** Allows administrators to log in with a password and access detailed booking records. Admins can view all bookings, including information about the source and destination, ticket numbers, fare, and booking timestamps.
4. **Data Storage and Retrieval:** Booking data is stored in a file and loaded upon application start, ensuring data persistence.

### **Technical Specifications**

- Programming Language: C++
- Libraries Used: Standard I/O Streams, File Handling, Time Management, and Input Validation.
- Data Structures: Arrays for storing booking details, structures for organizing booking information, and file I/O for data persistence.

# Features

## Features of the Metro Ticket Booking System

### 1. User Interface:

- Main Menu: A clear and user-friendly main menu to navigate through the application's functionalities, including booking tickets, viewing timetables, checking station details, and accessing contact information. Input Validation:
- Ensures that user inputs are validated for correctness, providing error messages and prompts for re-entry if necessary.

### 2. Ticket Booking:

- Station Selection: Users can select from a list of available metro stations as the source and destination for their journey. ○ Number of Tickets: Users specify the number of tickets they wish to purchase. ○ Journey Type: Users choose between a one-way or return journey, with fares calculated accordingly.
- Fare Calculation: Automatic calculation of fares based on the selected route and journey type, ensuring accurate pricing.
- Booking Details Storage: Records details of each booking, including source and destination stations, number of tickets, journey type, fare, and timestamp.

### 3. Timetable Information:

- Operational Details: Provides information on train frequency, service times, and travel durations between stations.
- Service Hours: Displays the operating hours of the metro service, including off-peak frequencies.

### 4. Station Information:

- Available Stations List: Displays a list of metro stations, offering users an overview of the network.
- Future Expansion: Notifies users about upcoming station additions to the network.

### 5. Admin Features:

- Admin Login: Secure login system for administrators to access privileged features.
- View All Bookings: Allows administrators to review detailed records of all bookings, including fare, journey details, and timestamps.
- Password Protection: Ensures secure access to administrative functions with password authentication.

#### **6. Data Management:**

- **Booking Storage:** Saves booking data to a file, ensuring that all records are preserved across sessions.
- **Data Retrieval:** Loads saved booking data from the file when the application starts, maintaining continuity and data integrity.

#### **7. Feedback and Exit Options:**

- **Feedback Prompts:** Provides options for users to leave feedback or return to the main menu.
- **Exit Options:** Allows users to exit the application gracefully, with a prompt for feedback and thanks for visiting.

#### **8. Date and Time Management:**

- **Timestamp Recording:** Stores the date and time of each booking, using the tm structure to capture precise timing information.
- **Date and Time Display:** Formats and displays booking timestamps in a user-friendly format.



## Tools and Environment

**Tools and Environment use in this project code.**

- **Programming Language:** The program is developed by using C++.
- C++ support object oriented programing and we make a class of same attribute.
- **Visual Studio:** A robust IDE with extensive features for C++ development, including debugging and version control.
- **MinGW:** A Windows port of GCC, used to compile C++ code in Windows environments.
- **Operating System:** The program will be developed and executed on an operating systems [Window].
- **File Handling:** The program uses file handling techniques to store and manage employee payroll records. Text files (.txt) are used for data storage and retrieval.
- **Libraries:** Standard C++ libraries for input/output operations, file handling, class, function and basic arithmetic operations are utilized in this program.

# Data Dictionary

## Data Dictionary for Metro Class

Attribute/Variable Name	Data Type	Description	Example Value
from[100]	int[]	Array storing the source station codes for bookings (1-5).	1
to[100]	int[]	Array storing the destination station codes for bookings (1-5).	3
no[100]	int[]	Array storing the number of tickets for each booking.	2
type[100]	int[]	Array storing the journey type (1 = One-way, 2 = Return).	1
fare[100]	float[]	Array storing the fare for each booking.	21.00
ans[10]	char[]	Temporary array for storing user responses to simple questions like "YES/yes".	"YES"
bookingCount	int	Stores the total number of bookings made.	5
adminPassword	char[]	Stores the password for admin login.	"admin123"
nextTicketId	int	Counter used to assign unique ticket IDs.	101
Booking	struct	Structure that holds details about each booking, including ticket ID, stations, number of tickets, fare, journey type, and the time the booking was made.	(Details in the structure table below)
bookings[100]	Booking[]	Array of Booking structures to store all bookings.	

## Booking Structure Data Dictionary

Attribute Name	Data Type	Description	Example Value
id	int	Unique ID assigned to each booking.	1001
from	int	Source station code (1-5).	1
to	int	Destination station code (1-5).	3
no	int	Number of tickets for the booking.	2
type	int	Journey type (1 = One-way, 2 = Return).	1
fare	float	Total fare for the booking.	31.00
dateTime	tm	Date and time of the booking.	2024-09-10 14:30:00

## Metro Class Methods

Method Name	Return Type	Description
<b>Metro()</b>	Constructor	Initializes default values, including the admin password, and loads bookings from a file.
<b>menu()</b>	void	Displays the main menu and handles user choices.
<b>booking()</b>	void	Handles the booking process, including input validation and fare calculation.
<b>timetable()</b>	void	Displays metro timetable details.
<b>station()</b>	void	Displays the available stations.
<b>contact_us()</b>	void	Placeholder function to display contact information (currently not implemented).
<b>show_bookings()</b>	void	Displays the current user's bookings.
<b>adminLogin()</b>	void	Handles the admin login process.
<b>displayAllBookings()</b>	void	Displays all bookings (admin-only functionality).
<b>saveBookings()</b>	void	Saves current bookings to a file.
<b>loadBookings()</b>	void	Loads bookings from a file when the program starts.
<b>cancelBooking()</b>	void	Cancels a booking based on a ticket ID.

## Program code (Complete code)

```
#include <iostream>
#include <limits>
#include <ctime>
#include <iomanip>
#include <fstream>
#include <cctype>
#include <cstring>
#include <sstream>

using namespace std;

class Metro {
private:
    int from[100], to[100], no[100], type[100];
    float fare[100];
    char ans[10];
    int bookingCount;
    char adminPassword[20];
    int nextTicketId; // To generate unique ticket IDs

    // Structure to hold booking details with tm for date and time
    struct Booking {
        int id; // Unique ticket ID
        int from;
        int to;
        int no;
        int type;
        float fare;
        tm dateTime; // Use tm structure to hold date and time
    } bookings[100];
```

public:

```
Metro() : bookingCount(0), nextTicketId(1) {
```

```
    strcpy(adminPassword, "admin123");
```

```
    loadBookings();
```

```
}
```

```
void booking();
```

```
void timetable();
```

```
void station();
```

```
void contact_us();
```

```
void menu();
```

```
void show_bookings(); // Method to show booking details
```

```
void adminLogin(); // Admin login method
```

```
void displayAllBookings(); // Method to display all bookings
```

```
void saveBookings(); // Save bookings to file
```

```
void loadBookings(); // Load bookings from file
```

```
void cancelBooking(); // Method to cancel a booking
```

```
};
```

```
// Function to display the menu and handle user choices
```

```
void Metro::menu() {
```

```
    int x;
```

```
    cout << "\n\n\n\t\t__HOME__";
```

```
    cout << "\n\t\t1. Booking\n\t\t2. Timetable\n\t\t3. Stations\n\t\t4. Contact us\n\t\t5. Show Bookings\n\t\t6. Cancel Booking\n\t\t7. Admin Login\n\t\t8. Exit\n";
```

```
    cout << "Choose the Operation: ";
```

```
    cin >> x;
```

```
// Validate input
```

```
while (cin.fail() || x < 1 || x > 8) {
```

```
    cin.clear(); // Clear the error flag
```

```

cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Discard invalid input
cout << "Invalid choice. Please enter a number between 1 and 8: ";
cin >> x;
}

```

```

switch (x) {
    case 1:
        booking();
        break;
    case 2:
        timetable();
        break;
    case 3:
        station();
        break;
    case 4:
        contact_us();
        break;
    case 5:
        show_bookings(); // Show bookings
        break;
    case 6:
        cancelBooking(); // Cancel booking
        break;
    case 7:
        adminLogin(); // Admin login
        break;
    case 8:
        saveBookings(); // Save bookings to file before exiting
        cout << "Your feedback is valuable for us!\n\n";
        cout << "Please let us know what you think.\n\n";
        cout << "Thanks for visiting!!!!\n\n";
        break;
}

```

```

        default:

            cout << "Enter the correct operation.\n";

        }
    }

// Function to display available stations
void Metro::station() {
    cout << "\t\t\t___Available Stations: ___\n";
    cout << "\n\t\t\t1. Kashmere Gate\n\t\t\t2. Rohini East\n\t\t\t3. Karol Bagh\n\t\t\t4. Noida\n\t\t\t5. Rithala\n";

    cout << "\t\t\t___ We are coming soon to other areas ___\n";
    cout << "Do you want to go home (YES/yes): ";
    cin >> ans;

    if (strcmp(ans, "YES") == 0 || strcmp(ans, "yes") == 0 || strcmp(ans, "Yes") == 0)
        menu();
    else
        cout << "\t\t\t\t\t\t\tThanks for visiting-_\n";
}

// Function to display the timetable
void Metro::timetable() {
    cout << "Operational Section: Kashmere Gate to Rithala\n";
    cout << "Frequency of Trains: 10 mins\n";
    cout << "Off Peak Frequency: 15 mins\n";
    cout << "Service Time: 7 am to 10 pm (15 hrs)\n";
    cout << "Station Stopping Time: 30 Sec\n";
    cout << "Travel Time from Kashmere Gate to Rithala: 25 mins\n";
    cout << "Do you want to go home (YES/yes): ";
    cin >> ans;

    if (strcmp(ans, "YES") == 0 || strcmp(ans, "yes") == 0 || strcmp(ans, "Yes") == 0)
        menu();
}

```





```

    }

    // Fare calculation
    if ((from[bookingCount] == 1 && to[bookingCount] == 2) || (from[bookingCount] ==
2 && to[bookingCount] == 1)) {
        fare = (10.50 * no[bookingCount]) * type[bookingCount];
    } else if ((from[bookingCount] == 2 && to[bookingCount] == 3) ||
(from[bookingCount] == 3 && to[bookingCount] == 2)) {
        fare = (17.50 * no[bookingCount]) * type[bookingCount];
    } else if ((from[bookingCount] == 2 && to[bookingCount] == 4) ||
(from[bookingCount] == 4 && to[bookingCount] == 2)) {
        fare = (17.50 * no[bookingCount]) * type[bookingCount];
    } else if ((from[bookingCount] == 3 && to[bookingCount] == 4) ||
(from[bookingCount] == 4 && to[bookingCount] == 3)) {
        fare = (7.00 * no[bookingCount]) * type[bookingCount];
    } else if ((from[bookingCount] == 1 && to[bookingCount] == 3) ||
(from[bookingCount] == 3 && to[bookingCount] == 1)) {
        fare = (21.00 * no[bookingCount]) * type[bookingCount];
    } else if ((from[bookingCount] == 1 && to[bookingCount] == 4) ||
(from[bookingCount] == 4 && to[bookingCount] == 1)) {
        fare = (21.00 * no[bookingCount]) * type[bookingCount];
    } else if ((from[bookingCount] == 1 && to[bookingCount] == 5) ||
(from[bookingCount] == 5 && to[bookingCount] == 1)) {
        fare = (31.00 * no[bookingCount]) * type[bookingCount];
    } else if ((from[bookingCount] == 2 && to[bookingCount] == 5) ||
(from[bookingCount] == 5 && to[bookingCount] == 2)) {
        fare = (31.00 * no[bookingCount]) * type[bookingCount];
    } else if ((from[bookingCount] == 3 && to[bookingCount] == 5) ||
(from[bookingCount] == 5 && to[bookingCount] == 3)) {
        fare = (31.00 * no[bookingCount]) * type[bookingCount];
    } else if ((from[bookingCount] == 4 && to[bookingCount] == 5) ||
(from[bookingCount] == 5 && to[bookingCount] == 4)) {
        fare = (21.00 * no[bookingCount]) * type[bookingCount];
    } else {

```







```

        saveBookings(); // Save changes to file
        return;
    }
}

cout << "Ticket ID not found.\n";
}

// Function to handle admin login
void Metro::adminLogin() {
    char password[20];
    cout << "Enter Admin Password: ";
    cin >> password;

    if (strcmp(password, adminPassword) == 0) {
        displayAllBookings();
    } else {
        cout << "Invalid password.\n";
    }
}

// Function to save bookings to a file
void Metro::saveBookings() {
    ofstream file("bookings.dat", ios::binary);
    if (file.is_open()) {
        file.write(reinterpret_cast<char*>(&bookingCount), sizeof(bookingCount));
        file.write(reinterpret_cast<char*>(&nextTicketId), sizeof(nextTicketId));
        file.write(reinterpret_cast<char*>(bookings), sizeof(bookings));
        file.close();
    } else {
        cout << "Unable to open file for saving bookings.\n";
    }
}

```



```
    return 0;  
}
```

## Test Case and Validation

- **Menu Selection**

Test valid inputs: 1, 2, 3, 4, 5, 6, 7.

Test invalid inputs: values outside the range 1-7, non-numeric inputs.

- **Station Selection**

Test valid inputs: 1, 2, 3, 4, 5.

Test invalid inputs: values outside the range 1-5, non-numeric inputs.

- **Ticket Booking Details**

Test valid inputs:

SOURCE STATION: 1 to 5.

DESTINATION STATION: 1 to 5.

NUMBER OF TICKETS: positive integer.

JOURNEY TYPE: 1 (One Way) or 2 (Return).

- **Test invalid inputs:**

SOURCE STATION and DESTINATION STATION outside 1-5.

NUMBER OF TICKETS: zero or negative numbers.

JOURNEY TYPE other than 1 or 2.

- **Admin Login**

Test valid password (e.g., admin123).

Test invalid password:

Incorrect password.

Empty password.

Test scenarios for failed attempts.

- **Functionality Testing**

Booking Function

Book tickets with valid details and check if fare calculation is correct.

Attempt booking with invalid details and ensure the system handles errors gracefully.

Timetable and Station Information

Ensure the correct timetable and station information is displayed.



Show Bookings

After booking tickets, ensure the bookings are correctly listed.

Validate the format and correctness of the displayed booking details, including date and time.

- **Admin Functionality**

Admin should be able to view all bookings correctly after successful login.

Ensure that the admin functionality is inaccessible without proper login.

- **File Operations**

**Saving Bookings**

Test if bookings are correctly saved to the file bookings.txt.

Check file content manually or programmatically to verify correctness.

**Loading Bookings**

Test if the program correctly loads previously saved bookings from the file.

Ensure that the system can handle cases where the file is empty or does not exist.

- **Booking Count Limits**

Test adding bookings when bookingCount is at its maximum (100 in this case).

Ensure the program handles the maximum limit gracefully.

**File Handling Edge Cases**

Test with a corrupted or malformed bookings.txt file.

Ensure the program handles file read/write errors properly.

- **User Experience**

**Error Messages**

Verify that error messages are clear and guide the user to correct input mistakes.

**User Prompts**

Check that user prompts (e.g., "Do you want to go home") are functioning as expected and respond to both affirmative and negative inputs.

## Sample Test Cases

### 1. Valid Menu Selection

- Input: 1
- Expected: Program should navigate to the booking function.

### 2. Invalid Menu Selection

- Input: 8 or non-numeric value

```
SOURCE STATION (1-5): 1
DESTINATION STATION (1-5): 6
Number of Tickets : 2
Journey type:
5. Rithala
SOURCE STATION (1-5): 1
DESTINATION STATION (1-5): 6
5. Rithala
SOURCE STATION (1-5): 1
5. Rithala
5. Rithala
SOURCE STATION (1-5): 1
DESTINATION STATION (1-5): 6
Number of Tickets : 2
Journey type:
1. One Way
2. Return
2
Invalid input. Please enter valid values.
```

- Expected: Program should prompt for valid input.

### 3. Valid Ticket Booking

- Inputs:
  - SOURCE STATION: 1
  - DESTINATION STATION: 2
  - NUMBER OF TICKETS: 2
  - JOURNEY TYPE: 1
  - Expected: Correct fare calculation and booking details saved.

### 4. Invalid Ticket Booking

- Inputs:
  - SOURCE STATION: 6 (out of range)
  - Expected: Error message and prompt to re-enter valid input.

```

      ____HOME____
      1. Booking
      2. Timetable
      3. Stations
      4. Show Bookings
      5. Cancel Booking
      6. Admin Login
      7. Exit

Choose the Operation: 8
Invalid choice. Please enter a number between 1 and 7: 
```

### 5. Admin Login Success

- o Input: admin123
- o Expected: Display all bookings.

### 6. Admin Login Failure

- o Inputs:
- o Incorrect Password

```

      ____HOME____
      1. Booking
      2. Timetable
      3. Stations
      4. Show Bookings
      5. Cancel Booking
      6. Admin Login
      7. Exit

Choose the Operation: 6
Enter Admin Password: admin432
Invalid password.
```

### 7. File Saving and Loading

- o Save bookings to bookingss.txt and then reload.
- o Expected: Data integrity is maintained across save and load operations.

## Input and Output Screens

Choose Operation : Now we have to choose operation which we wants to perform on the program .Take input from 1-7 to see the result.

For Booking press 1 : If we wants to book ticket from Source station to Destination .

```
WELCOME TO DELHI METRO
Developed By JCC ROHINI SECTOR 3, NEW DELHI

-----HOME-----
1. Booking
2. Timetable
3. Stations
4. Contact us
5. Show Bookings
6. Admin Login
7. Exit

Choose the Operation: |
```

Source Station : 1(Kashmere Gate)

Destination Station: 4(Noida)

Number of tickets: 1

Journey Type: press 1 for one way and 2 for Return

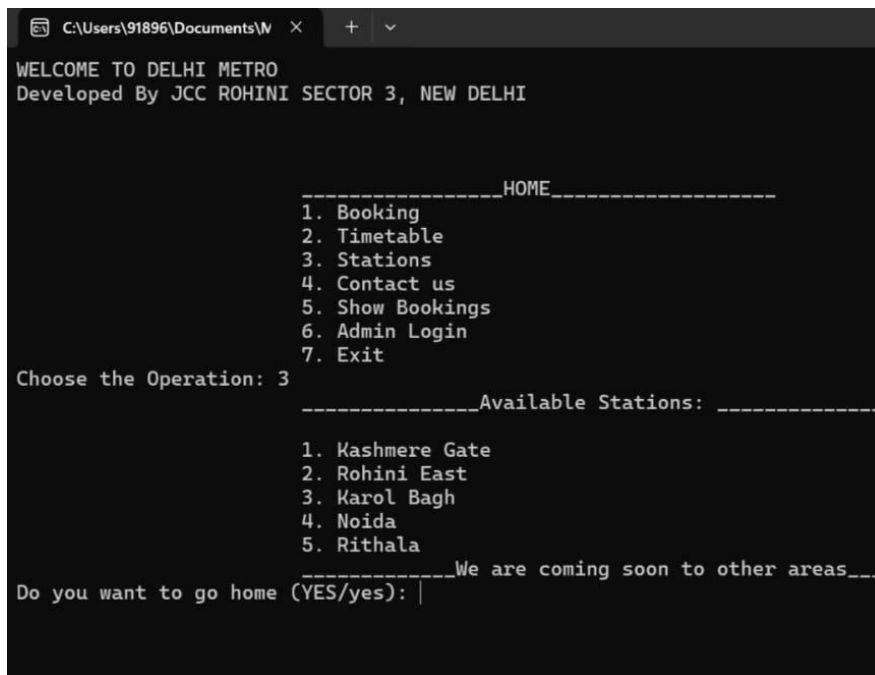
And now it will calculate ticket fare .

```
C:\Users\91896\Documents\W x + v
WELCOME TO DELHI METRO
Developed By JCC ROHINI SECTOR 3, NEW DELHI

-----HOME-----
1. Booking
2. Timetable
3. Stations
4. Contact us
5. Show Bookings
6. Admin Login
7. Exit

Choose the Operation: 2
Operational Section: Kashmere Gate to Rithala
Frequency of Trains: 10 mins
Off Peak Frequency: 15 mins
Service Time: 7 am to 10 pm (15 hrs)
Station Stopping Time: 30 Sec
Travel Time from Kashmere Gate to Rithala: 25 mins
Do you want to go home (YES/yes): |
```

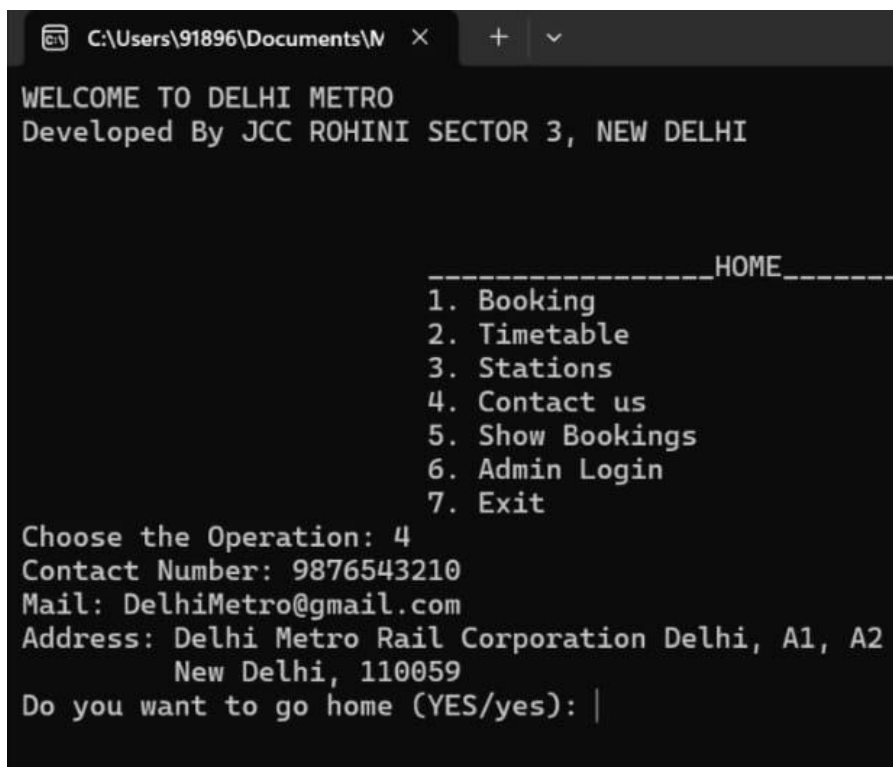
Choose Operation 2 : if we enter option 2 it will show us timetable , Service timing , Duration etc.



```
C:\Users\91896\Documents\W x + v
WELCOME TO DELHI METRO
Developed By JCC ROHINI SECTOR 3, NEW DELHI

-----HOME-----
1. Booking
2. Timetable
3. Stations
4. Contact us
5. Show Bookings
6. Admin Login
7. Exit
Choose the Operation: 3
-----Available Stations: -----
1. Kashmere Gate
2. Rohini East
3. Karol Bagh
4. Noida
5. Rithala
-----We are coming soon to other areas-----
Do you want to go home (YES/yes): |
```

Choose operation : If we choose operation 3 it will show us all the station that are available



```
C:\Users\91896\Documents\W x + v
WELCOME TO DELHI METRO
Developed By JCC ROHINI SECTOR 3, NEW DELHI

-----HOME-----
1. Booking
2. Timetable
3. Stations
4. Contact us
5. Show Bookings
6. Admin Login
7. Exit
Choose the Operation: 4
Contact Number: 9876543210
Mail: DelhiMetro@gmail.com
Address: Delhi Metro Rail Corporation Delhi, A1, A2
New Delhi, 110059
Do you want to go home (YES/yes): |
```

Choose operation : If we choose operation 4 ,it will give us all the Booking Details such as number of ticket , Journey time , Fare, Source Station , Destination Station.

```
HOME
1. Booking
2. Timetable
3. Stations
4. Show Bookings
5. Cancel Booking
6. Admin Login
7. Exit
Choose the Operation: 5
Enter Ticket ID to Cancel: 2
Booking with Ticket ID 2 has been cancelled.
-----
Process exited after 68.27 seconds with return value 0
Press any key to continue . . .
```

Choose operation : If we choose operation 5, it will cancel tickets by giving ticket id .

```
C:\Users\91896\Documents\W x + v
WELCOME TO DELHI METRO
Developed By JCC ROHINI SECTOR 3, NEW DELHI

-----HOME-----
1. Booking
2. Timetable
3. Stations
4. Contact us
5. Show Bookings
6. Admin Login
7. Exit
Choose the Operation: 5
-----BOOKING DETAILS-----
Booking 1:
From: 3
To: 1
Number of Tickets: 3
Journey Type: Return
Fare: 126 rupees
Date and Time: 2024-08-07 22:05:47
-----
Booking 2:
From: 1
To: 4
Number of Tickets: 1
Journey Type: One Way
Fare: 21 rupees
Date and Time: 2024-08-08 21:31:44
-----
Do you want to go back to menu (YES/yes): |
```

Choose operation : If we enter operation 6 , It will ask for Admin Log in Details such as Password to show all the Booking Details .

```
C:\Users\91896\Documents\W x + v
WELCOME TO DELHI METRO
Developed By JCC ROHINI SECTOR 3, NEW DELHI

-----HOME-----
1. Booking
2. Timetable
3. Stations
4. Contact us
5. Show Bookings
6. Admin Login
7. Exit
Choose the Operation: 7
Your feedback is valuable for us!

Please let us know what you think.

Thanks for visiting!!!!

-----
Process exited after 6.815 seconds with return value 0
Press any key to continue . . . |
```

Choose operation : If we enter 7 , It will ask for Feedback and Exit .

## **Limitation of the Project**

### **1. Array Size Limits**

- Limitation: The code uses fixed-size arrays (from[100], to[100], etc.) to store booking details, which means the maximum number of bookings is limited to 100.
- Improvement: Use dynamic data structures such as `std::vector` to handle a variable number of bookings.

### **2. No Validation for User Input**

- Limitation: Although there is some input validation in the booking method, there is no comprehensive input validation for other methods or for all possible edge cases.
- Improvement: Implement robust input validation for all user inputs across different methods.

### **3. Error Handling**

- Limitation: Error handling is minimal. For instance, the program might crash or behave unexpectedly if file operations fail or if invalid data is read from files.
- Improvement: Add more error handling, especially for file operations and invalid inputs. Use exceptions or error codes to manage errors more gracefully.

### **4. Fixed Fare Calculation Limitation:**

- Fare calculation is hard-coded and limited to specific routes. If fare changes or new routes are added, the code must be manually updated.
- Improvement: Use a more flexible approach for fare calculation, such as reading fare data from a configuration file or database.

### **5. Security Concerns**

- Limitation: The admin password is stored as plain text, and there is no mechanism for securing or hashing passwords.
- Improvement: Implement password hashing and secure storage mechanisms. Consider using a more secure authentication system.

### **6. Time Handling**

- Limitation: The code uses the `tm` structure for date and time, which is a C-style approach. It can be prone to errors and lacks modern conveniences.
- Improvement: Use C++11 `<chrono>` library or other modern date/time libraries for better handling and formatting of date and time.



## **7. Code Duplication**

- Limitation: There is duplicated logic for checking user input and formatting output in multiple methods.
- Improvement: Refactor the code to use helper functions or classes to handle repetitive tasks and reduce code duplication.

## **8. User Interface**

- Limitation: The user interface is text-based and may not be very userfriendly or intuitive.
- Improvement: Consider developing a graphical user interface (GUI) for a better user experience, or improve the text-based UI with clearer prompts and error messages.

## **9. File Handling Issues**

- Limitation: The code uses plain text files for saving and loading bookings, which might not be secure or efficient.
- Improvement: Consider using binary files or a database for better performance and security.

## **10. Memory Management**

- Limitation: While the current code doesn't have explicit dynamic memoryallocation issues, using fixed-size arrays might waste memory or be inefficient.
- Improvement: Use dynamic containers like `std::vector` to manage memory more efficiently and flexibly.

## **11. Ticket ID Generation**

- Limitation: The ticket ID generation is a simple integer increment, which might not be unique in a multi-user environment.
- Improvement: Use a more robust ticket ID generation strategy, such as UUIDs or a database-generated ID.

## **12. User Feedback and Usability**

- Limitation: The system asks for user feedback but does not actually handle or store it.
- Improvement: Implement a system for collecting and storing user feedback if needed.

### **13. Code Documentation and Readability**

- Limitation: The code lacks comments and documentation, making it harder to understand and maintain.
- Improvement: Add comments and documentation to explain the purpose of different methods, classes, and logic.

### **14 Function Decomposition**

- Limitation: Some methods like `booking()` handle multiple responsibilities, including input validation and fare calculation.
- Improvement: Decompose complex methods into smaller, single-responsibility functions to improve readability and maintainability.

## Future Application of the Project

### 1. Integration with Real-Time Data

- **Real-Time Scheduling:** Integrate the system with real-time train schedules and updates. This could include train delays, cancellations, and dynamic fare adjustments based on peak times.
- **Live Updates:** Implement live updates for train timings and station statuses using APIs from metro service providers.

### 2. Mobile and Web Application Interface

- **Mobile App:** Develop a mobile application for both iOS and Android platforms, allowing users to book tickets, view schedules, and manage their bookings from their smartphones.
- **Web Portal:** Create a web-based interface where users can perform the same operations as the desktop application, providing greater accessibility.

### 3. Enhanced User Experience

- **User Accounts and Profiles:** Implement user accounts with personal profiles, booking history, and saved payment methods for quicker future transactions.
- **Push Notifications:** Send notifications about booking confirmations, reminders, and updates via SMS or email.

### 4. Advanced Fare Calculation

- **Dynamic Pricing:** Implement dynamic pricing based on demand, time of day, or season to optimize revenue and manage crowd control.
- **Discounts and Offers:** Introduce discount schemes for regular commuters, special offers, or promotional codes.

### 5. Additional Booking Features

- **Seat Selection:** Allow users to select specific seats or compartments, especially for long journeys.
- **Multimodal Integration:** Integrate with other transportation modes like buses or taxis, offering a unified ticketing system for a seamless travel experience.

### 6. Administrative and Reporting Tools

- **Dashboard for Admins:** Develop a comprehensive admin dashboard for monitoring bookings, generating reports, and managing operational aspects.

- **Analytics and Reporting:** Implement advanced analytics to track booking trends, peak travel times, and user demographics for better decision-making.

## 7. Accessibility and Localization

- **Multi-Language Support:** Offer the application in multiple languages to cater to a diverse user base.
- **Accessibility Features:** Ensure the application is accessible to users with disabilities by following WCAG (Web Content Accessibility Guidelines).

## 8. Security and Privacy Enhancements

- **Secure Transactions:** Implement secure payment gateways with encryption to protect user payment information.
- **Data Protection:** Ensure user data privacy and comply with regulations like GDPR (General Data Protection Regulation) for data handling and storage.

## 9. Scalability and Performance

- **Scalable Architecture:** Design the system to handle a large number of concurrent users and bookings efficiently, using cloud services or distributed systems.
- **Performance Optimization:** Optimize code and database queries to improve the performance and responsiveness of the application.

## 10. Integration with Payment Gateways

- **Online Payment:** Integrate with various payment gateways (e.g., PayPal, Stripe) to allow online payments for ticket bookings.
- **Wallet Integration:** Allow integration with digital wallets and payment apps for a smoother transaction process.

## Bibliography

### 1. Books:

- "Programming: Principles and Practice Using C++" by Bjarne Stroustrup:  
Written by the creator of C++, this book introduces programming concepts in a clear and structured manner. It covers basic programming principles and provides a solid foundation in C++.

### 2. Online Resources:

- Google (<https://www.google.com>) – A vast repository of knowledge and problem-solving resources, Google was instrumental in finding solutions to specific coding issues, debugging techniques, and understanding best practices in C++ programming.
- GeeksforGeeks (<https://www.geeksforgeeks.org>) –GeeksforGeeks offered numerous tutorials, code examples, and explanations of C++ concepts and algorithms that were beneficial for solving specific problems and improving the overall code structures.