mirafra
TECHNOLOGIES

CAPSTONE PROJECT ON

# REAL TIME TICKET BOOKING SYSTEM

## ABSTRACT

The **Real-Time Ticket Booking System** is a command-line application developed to streamline the process of reserving tickets for buses, trains, and flights. This system enables users to securely register, log in, and manage their bookings with features like ticket search, booking, cancellation, and history tracking. Implemented in C++, it employs object-oriented programming principles and efficient data structures such as vectors and unordered maps to handle ticket inventory and user data effectively. The system supports real-time updates, allowing users to search for tickets based on transport type, origin, destination, and date while checking availability. A built-in payment module offers multiple options, including debit cards, UPI, and mobile payment apps, ensuring secure and convenient transactions. Booked tickets are generated as detailed text files, and users can access their booking and cancellation history for future reference. Data persistence is achieved through file handling, ensuring consistent and reliable storage of ticket information. The application incorporates robust input validation and error handling to enhance usability and minimize user errors. It emphasizes clarity and responsiveness in its interface, providing an efficient, user-friendly experience. This project showcases practical implementations of C++ concepts, including file I/O, data structures, and object-oriented design, offering a comprehensive solution to ticket booking challenges. It is ideal for demonstrating real-world programming skills and the ability to develop systems that integrate multiple functionalities seamlessly.

**Key Words**: Real-Time Ticket Booking System, C++ Programming, Command-Line Interface (CLI), Object-Oriented Programming (OOP), Data Structures, Ticket Management, File Handling, User Authentication, Payment Integration, Booking, Cancellation, Transportation Tickets, Ticket History Tracking, Secure Transactions, Input Validation, Data Persistence.

## INTRODUCTION

The **Real-Time Ticket Booking System** is a C++ application designed to facilitate the process of booking tickets for various modes of transportation, such as buses, trains, or flights. It provides users with a platform to search for available tickets, make reservations, cancel bookings, and view their booking history. The system is equipped with essential features like authentication, payment processing, and detailed record management, ensuring a smooth and secure user experience.

real-world ticket booking environment where users can:

1. **Search Tickets**: Filter available tickets based on transport type, origin, destination, and date.
2. **Book Tickets**: Secure a ticket by providing necessary details and making a payment.
3. **Cancel Bookings**: Cancel an already booked ticket and update its availability.
4. **View History**: Access previously booked or canceled tickets.
5. **Authenticate Users**: Register new users or allow existing users to log in securely.

## OBJECTIVES

- Develop a robust system for real-time ticket booking and cancellations.
- Provide users with an intuitive interface for ticket searches based on specific criteria such as date, origin, destination, and transport type.
- Include features like payment confirmation, ticket generation, and history tracking.
- Ensure data persistence and security through file handling and authentication mechanisms.
- Support a variety of transport modes, including buses, trains, and flights.

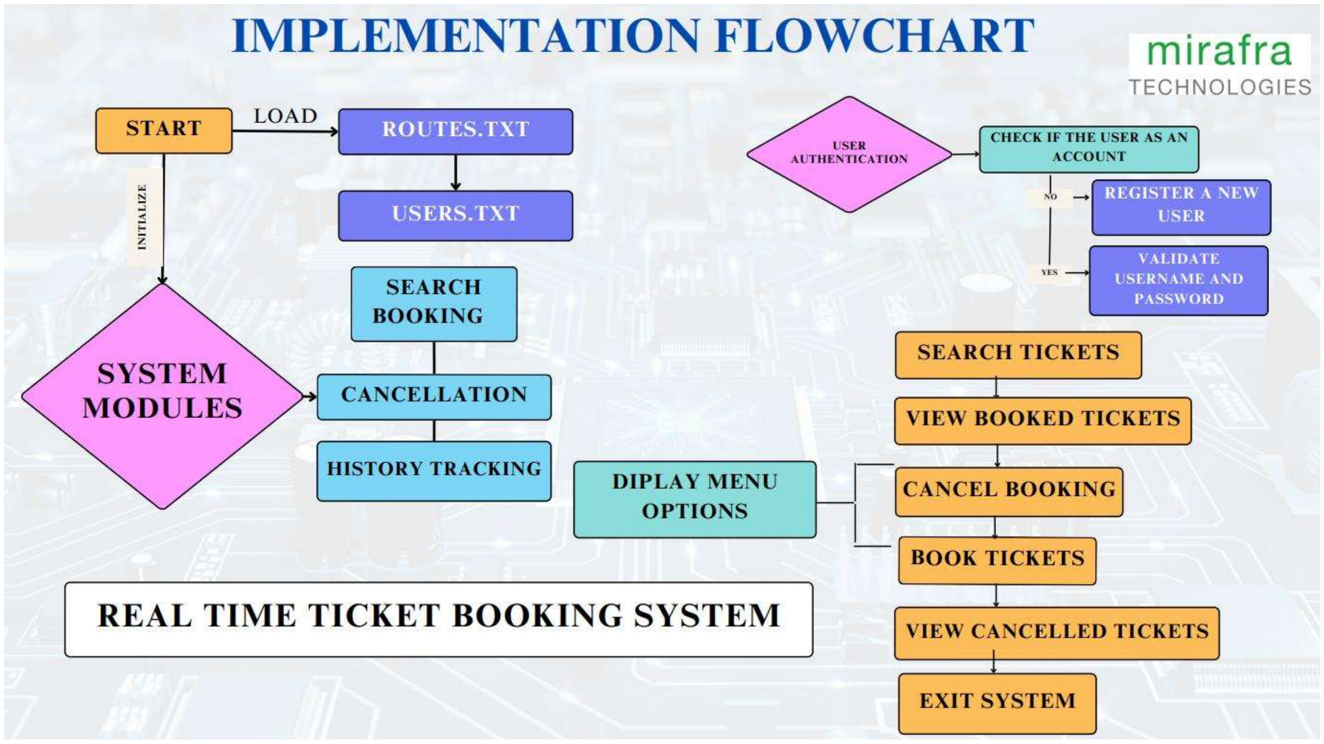
## FEATURES

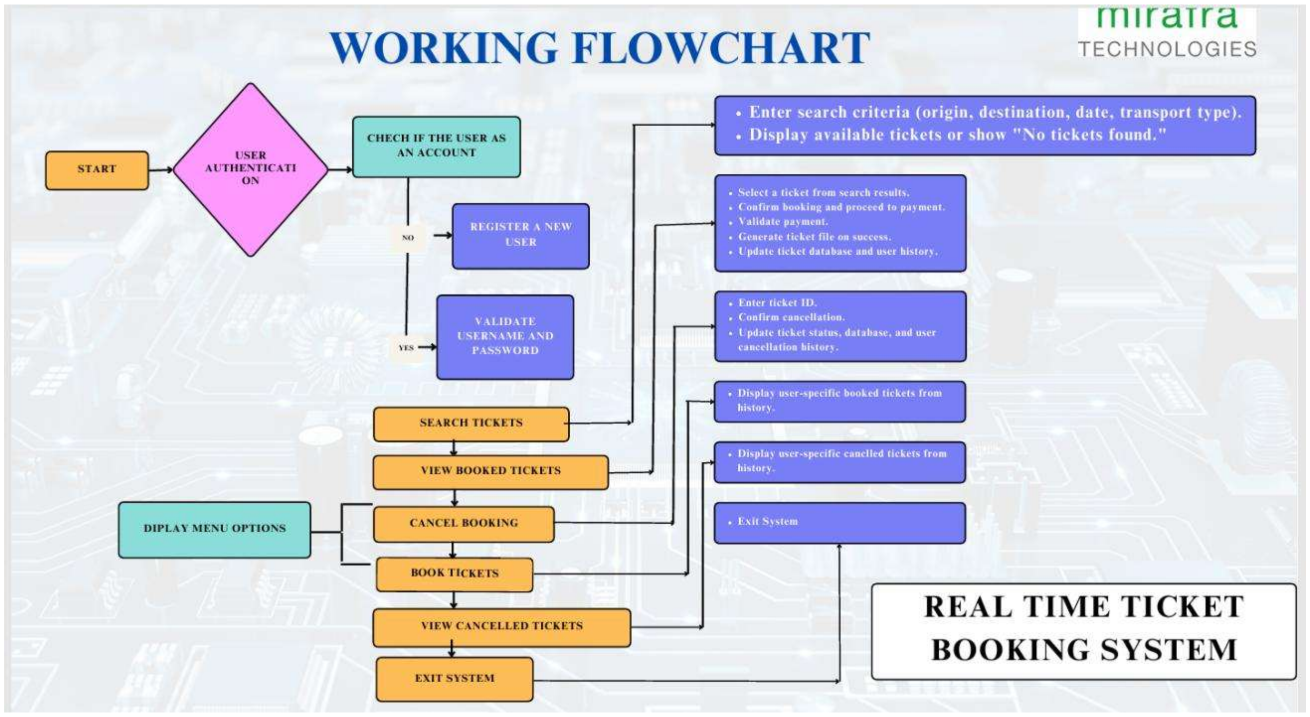
1. **User Authentication**
    - Registration and login functionality.
    - Secure credential management.
2. **Ticket Management**
    - Search tickets based on transport type, origin, destination, and date.
    - Book tickets and generate detailed ticket files.
    - Cancel bookings and update availability dynamically.
3. **Payment Gateway**
    - Integration of multiple payment methods, including Debit Card, UPI, and digital wallets.
    - Payment confirmation and processing.
4. **User Profiles**
    - View booked and canceled tickets history.
      Manage user-specific ticket details.

## IMPLEMENTATION

1. **Programming Language**
    - The system is developed using **C++** for its efficiency and support for file handling and object-oriented programming.
2. **File Handling**
    - `routes.txt`: Stores ticket details.
    - `<username>_booked.txt`: Stores booked tickets for each user.
    - `<username>_cancelled.txt`: Stores canceled tickets for each user.
    - `users.txt`: Stores user credentials for authentication.
3. **Classes and Modules**
    - **Ticket**: Represents ticket attributes and status.
    - **Booking System**: Manages ticket operations such as search, booking, cancellation, and payment.
    - **Authentication**: Handles user registration and login.
4. **Algorithms**
    - **Search Algorithm**: Filters tickets based on user inputs (origin, destination, transport type, and date).
    - **Payment Processing**: Simulates payment confirmation through user interaction.
    - **Dynamic Updates**: Updates ticket availability status after booking or cancellation.
5. **Data Structures**
    - `vector<Ticket>`: Stores ticket data in memory for real-time operations.
    - `unordered_map<int, string>`: Maps ticket IDs to customer names for tracking bookings.

## FLOWCHARTS





- Enter search criteria (origin, destination, date, transport type).
- Display available tickets or show "No tickets found."

- Select a ticket from search results.
- Confirm booking and proceed to payment.
- Validate payment.
- Generate ticket file on success.
- Update ticket database and user history.

- Enter ticket ID.
- Confirm cancellation.
- Update ticket status, database, and user cancellation history.

- Display user-specific booked tickets from history.

- Display user-specific canclled tickets from history.

- Exit System

## CODE IMPLEMENTATION

This is the code implementation of real time ticket booking system

```
#include <iostream>
#include <fstream>
#include <string>
#include <unordered_map>
#include <vector>
#include <iomanip>
#include <limits>
using namespace std;

// Class to manage tickets
class Ticket {
public:
    int ticketID;
    string origin;
    string destination;
    string date;
    bool isAvailable;
    string transportType;
    double price; // New attribute to store ticket price

    Ticket(int id, string org, string dest, string dt, string transport, double p, bool available = true)
        : ticketID(id), origin(org), destination(dest), date(dt), transportType(transport), price(p),
isAvailable(available) {}
};

class BookingSystem {
private:
    vector<Ticket> tickets;              // List of tickets
    unordered_map<int, string> bookings; // TicketID -> Customer Name

    // Load tickets from file
    void loadTickets() {
        ifstream file("routes.txt");
        if (!file.is_open()) {
            cerr << "Error: Could not open tickets file.\n";
            return;
        }

        int id, available;
        string org, dest, date, transport;
        double price;
        while (file >> id >> org >> dest >> date >> transport >> price >> available) {
            tickets.emplace_back(id, org, dest, date, transport, price, available == 1);
        }
        file.close();
    }

    // Save tickets to file
    void saveTickets() {
        ofstream file("routes.txt");
        if (!file.is_open()) {
            cerr << "Error: Could not save tickets file.\n";
            return;
        }

        for (const auto& ticket : tickets) {
            file << ticket.ticketID << " " << ticket.origin << " " << ticket.destination << " "
                 << ticket.date << " " << ticket.transportType << " " << ticket.price << " "
                 << (ticket.isAvailable ? 1 : 0) << "\n";
        }
        file.close();
    }

    // Function to generate a ticket after booking
    void generateTicket(const Ticket& ticket, const string& customerName) {
        // Create a file named ticket_<ticketID>.txt
        ofstream ticketFile("ticket_" + to_string(ticket.ticketID) + ".txt");
        if (!ticketFile.is_open()) {
            cerr << "Error generating ticket file.\n";
            return;
        }

        // Write ticket details into the file
        ticketFile << "***** TICKET *****\n";
        ticketFile << "Ticket ID: " << ticket.ticketID << "\n";
        ticketFile << "Customer: " << customerName << "\n";
        ticketFile << "Origin: " << ticket.origin << "\n";
        ticketFile << "Destination: " << ticket.destination << "\n";
        ticketFile << "Date: " << ticket.date << "\n";
        ticketFile << "Transport: " << ticket.transportType << "\n";
        ticketFile << "Price: $" << fixed << setprecision(2) << ticket.price << "\n";
        ticketFile << "Status: Booked\n";
        ticketFile << "*\n";

        ticketFile.close();
        cout << "Ticket generated successfully! Check 'ticket_" << ticket.ticketID << ".txt' for your ticket
details.\n";
    }
```

```
public:
   BookingSystem() {
      loadTickets();
   }

   ~BookingSystem() {
      saveTickets();
   }

   void searchTickets(const string& transport, const string& origin, const string& destination, const string&
date) {
      bool found = false;
      cout << "\nAvailable " << transport << " Tickets:\n";
      cout << setw(10) << "TicketID" << setw(15) << "Origin" << setw(15) << "Destination"
          << setw(15) << "Date" << setw(15) << "Transport" << setw(10) << "Price" << setw(10) << "Status"
<< "\n";

      for (const auto& ticket : tickets) {
         if (ticket.transportType == transport && ticket.origin == origin && ticket.destination == destination
            && ticket.date == date && ticket.isAvailable) {
            cout << setw(10) << ticket.ticketID << setw(15) << ticket.origin
                << setw(15) << ticket.destination << setw(15) << ticket.date
                << setw(15) << ticket.transportType << setw(10) << ticket.price
                << setw(10) << "Available" << "\n";
            found = true;
         }
      }

      if (!found) {
         cout << "No tickets available for the given criteria.\n";
      }
   }

   void bookTicket(const string& transport, const string& origin, const string& destination, const string&
date, const string& customerName) {
   bool found = false;
   for (auto& ticket : tickets) {
      if (ticket.transportType == transport && ticket.origin == origin && ticket.destination == destination
         && ticket.date == date && ticket.isAvailable) {
         cout << "\nTicket found: " << ticket.ticketID << " " << ticket.origin << " -> " << ticket.destination
             << " on " << ticket.date << " (" << ticket.transportType << ")\n";
         cout << "Price: $" << ticket.price << "\n";
         cout << "Do you want to book this ticket? (y/n): ";
         char choice;
         cin >> choice;

         // Handle invalid input for booking confirmation
```

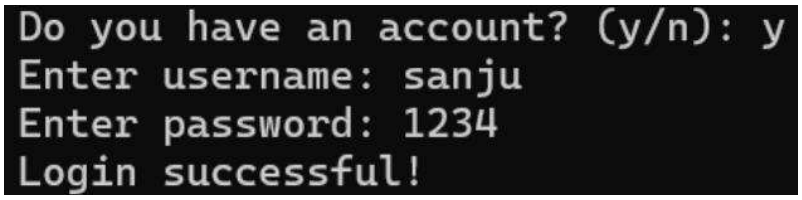
```
        while (choice != 'y' && choice != 'Y' && choice != 'n' && choice != 'N') {
          cout << "Invalid choice. Please enter 'y' for yes or 'n' for no: ";
          cin >> choice;
        }

        if (choice == 'y' || choice == 'Y') {
          // Ask for payment after booking
          if (payForTicket(ticket.price)) {
            ticket.isAvailable = false;
            bookings[ticket.ticketID] = customerName;
            cout << "Ticket booked successfully for " << customerName << "!\n";

            // Save to user's booked ticket history
            ofstream bookedFile(customerName + "_booked.txt", ios::app);
            if (bookedFile.is_open()) {
              bookedFile << ticket.ticketID << " " << ticket.origin << " -> " << ticket.destination
                         << " on " << ticket.date << " (" << ticket.transportType << ")\n";
              bookedFile.close();
            } else {
              cerr << "Error opening booked file for " << customerName << endl;
            }

            // Generate ticket after booking
            generateTicket(ticket, customerName);

            // Save the updated tickets to routes.txt
            saveTickets();

            found = true;
            break;
          } else {
            cout << "Payment failed. Booking was not completed.\n";
          }
        } else {
          cout << "Booking canceled.\n";
        }
      }
    }

    if (!found) {
      cout << "No available tickets found for the given criteria.\n";
    }
}


  bool payForTicket(double ticketPrice) {
    // Automatically set the payment to the ticket price
```

```
double payment = ticketPrice;

// Display the payment options
cout << "\nSelect a payment method:\n";
cout << "1. Debit Card\n";
cout << "2. UPI Payment\n";
cout << "3. Gpay/Phonepay\n";

int choice;
string paymentMethod;

while (true) { // Loop until a valid option is selected
    cout << "Enter your choice (1/2/3): ";
    cin >> choice;
     // Check for invalid input (non-numeric input)
    if (cin.fail()) {
        cin.clear(); // Clear the error flag
        cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Discard invalid input
        cout << "Invalid choice, please enter a valid option (1-3).\n";
    }

    if (choice == 1) {
        paymentMethod = "Debit Card";
        break;
    } else if (choice == 2) {
        paymentMethod = "UPI Payment";
        break;
    } else if (choice == 3) {
        paymentMethod = "Gpay/Phonepay";
        break;
    } else {
        cout << "Invalid choice. Please select a valid option (1/2/3).\n";
    }
}

// Confirm the payment
cout << "\nYou have selected " << paymentMethod << " as your payment method.\n";
cout << "The total payment amount is: $" << fixed << setprecision(2) << payment << "\n";
cout << "Do you confirm the payment of $" << fixed << setprecision(2) << payment << "? (y/n): ";

char confirm;
cin >> confirm;
// Handle invalid input for booking confirmation
        while (confirm != 'y' && confirm != 'Y' && confirm != 'n' && confirm != 'N') {
            cout << "Invalid choice. Please enter 'y' for yes or 'n' for no: ";
            cin >> confirm;
        }
    if (confirm == 'y' || confirm == 'Y') {
        cout << "\nPayment of $" << fixed << setprecision(2) << payment << " via " << paymentMethod << "
successful.\n";
        return true;  // Payment is successful
    } else {
        cout << "Payment canceled. Booking not completed.\n";
        return false;  // Payment is canceled
    }
}


    void cancelBooking(int ticketID, const string& customerName) {
        bool found = false;
        for (auto& ticket : tickets) {
            if (ticket.ticketID == ticketID && !ticket.isAvailable) {
                cout << "\nDo you want to cancel this booking? (y/n): ";
                char choice;
                cin >> choice;
                  // Handle invalid input for booking confirmation
            while (choice != 'y' && choice != 'Y' && choice != 'n' && choice != 'N') {
                cout << "Invalid choice. Please enter 'y' for yes or 'n' for no: ";
                cin >> choice;
            }
                if (choice == 'y' || choice == 'Y') {
                    ticket.isAvailable = true;
                    bookings.erase(ticket.ticketID);
              // Update user's booking history
                std::ofstream updateHistory(customerName + "_booked.txt", std::ios::app);
                if (updateHistory.is_open())
{
                    updateHistory << "Past Booking Ticket Cancelled\n";
                    updateHistory.close();
                } else {
                    std::cerr << "Error opening Booked file for " << customerName << std::endl;
                }

                    // Save to user's canceled ticket history
                    ofstream cancelledFile(customerName + "_cancelled.txt", ios::app);
                    if (cancelledFile.is_open()) {
                        cancelledFile << ticket.ticketID << " " << ticket.origin << " -> " << ticket.destination
                                     << " on " << ticket.date << " (" << ticket.transportType << ")\n";
                        cancelledFile.close();
                    } else {
                        cerr << "Error opening cancelled file for " << customerName << endl;
                    }
                cout << "Booking canceled.\n";

                // Save the updated tickets to routes.txt
                saveTickets();

                found = true;
            }
            break;
        }
    }

    if (!found) {
        cout << "Ticket not found or already available.\n";
    }
}

void viewBookedTickets(const string& customerName) {
    bool found = false;
    cout << "\nBooked Tickets for " << customerName << ":\n";
    cout << setw(10) << "TicketID" << setw(15) << "Origin" << setw(15) << "Destination"
         << setw(15) << "Date" << setw(15) << "Transport" << setw(10) << "Price" << "\n";

    // Display user's booked tickets from the file
    ifstream bookedFile(customerName + "_booked.txt");
    if (bookedFile.is_open()) {
        string line;
        while (getline(bookedFile, line)) {
            cout << line << "\n";
            found = true;
        }
        bookedFile.close();
    } else {
        cerr << "Error opening booked file for " << customerName << endl;
    }

    if (!found) {
        cout << "No booked tickets found for " << customerName << ".\n";
    }
}

void viewCancelledTickets(const string& customerName) {
    bool found = false;
    cout << "\nCancelled Tickets for " << customerName << ":\n";
    cout << setw(10) << "TicketID" << setw(15) << "Origin" << setw(15) << "Destination"
         << setw(15) << "Date" << setw(15) << "Transport" << setw(10) << "Price" << "\n";
        // Display user's cancelled tickets from the file
        ifstream cancelledFile(customerName + "_cancelled.txt");
        if (cancelledFile.is_open()) {
            string line;
            while (getline(cancelledFile, line)) {
                cout << line << "\n";
                found = true;
            }
            cancelledFile.close();
        } else {
            cerr << "Error opening cancelled file for " << customerName << endl;
        }

        if (!found) {
            cout << "No canceled tickets found for " << customerName << ".\n";
        }
    }
};

// User authentication system
class Authentication {
public:
    bool authenticate(string& currentUser) {
        unordered_map<string, string> users;
        ifstream infile("users.txt");
        string username, password;

        // Load users
        while (infile >> username >> password) {
            users[username] = password;
        }
        infile.close();

        char choice;
        while (true) {
            cout << "Do you have an account? (y/n): ";
            cin >> choice;

            // Check for valid input
            if (choice == 'y' || choice == 'Y') {
                // Login process
                while (true) {
                    cout << "Enter username: ";
                    cin >> username;
                    cout << "Enter password: ";
                    cin >> password;
                if (users.find(username) != users.end() && users[username] == password) {
                    currentUser = username;
                    cout << "Login successful!\n";
                    return true;
                } else {
                    cout << "Invalid credentials. Please enter valid credentials.\n";
                }
            }
        } else if (choice == 'n' || choice == 'N') {
            // Registration process
            cout << "Register a new account.\n";
            cout << "Enter username: ";
            cin >> username;
            cout << "Enter password: ";
            cin >> password;

            ofstream outfile("users.txt", ios::app);
            outfile << username << " " << password << "\n";
            outfile.close();

            currentUser = username;
            cout << "Account created successfully!\n";
            return true;
        } else {
            cout << "Invalid input. Please enter 'y' or 'n'.\n";
        }
    }
  }
};
int main() {
  string currentUser;
  Authentication auth;
  if (!auth.authenticate(currentUser)) {
    cerr << "Authentication failed.\n";
    return 1;
  }

  BookingSystem bookingSystem;

  while (true) {
    int choice;
    cout << "\n**** Ticket Booking System ****\n";
    cout << "1. Search Tickets\n2. Book Ticket\n3. Cancel Booking\n4. View Booked Tickets\n5. View
Canceled Tickets\n6. Exit\n";
    cout << "Enter your choice: ";
    cin >> choice;
```

```
        // Check for invalid input (non-numeric input)
        if (cin.fail()) {
           cin.clear(); // Clear the error flag
           cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Discard invalid input
           cout << "Invalid choice, please enter a valid option (1-6).\n";
        } else if (choice == 1) {
           string transport, origin, destination, date;
           cout << "Enter transport type: ";
           cin >> transport;
           cout << "Enter origin: ";
           cin >> origin;
           cout << "Enter destination: ";
           cin >> destination;
           cout << "Enter date (yyyy-mm-dd): ";
           cin >> date;
           bookingSystem.searchTickets(transport, origin, destination, date);
        } else if (choice == 2) {
           string transport, origin, destination, date, customerName;
           cout << "Enter transport type: ";
           cin >> transport;
           cout << "Enter origin: ";
           cin >> origin;
           cout << "Enter destination: ";
           cin >> destination;
           cout << "Enter date (yyyy-mm-dd): ";
           cin >> date;
           bookingSystem.bookTicket(transport, origin, destination, date, currentUser);
        } else if (choice == 3) {
           int ticketID;
           cout << "Enter ticket ID to cancel: ";
           cin >> ticketID;

           bookingSystem.cancelBooking(ticketID, currentUser);
        } else if (choice == 4) {
           bookingSystem.viewBookedTickets(currentUser);
        } else if (choice == 5) {
           bookingSystem.viewCancelledTickets(currentUser);
        } else if (choice == 6) {
           cout << "Exiting system.\n";
           break;
        } else {
           cout << "Invalid choice, please enter a valid option (1-6).\n";
        }
    }
  return 0;
}
```

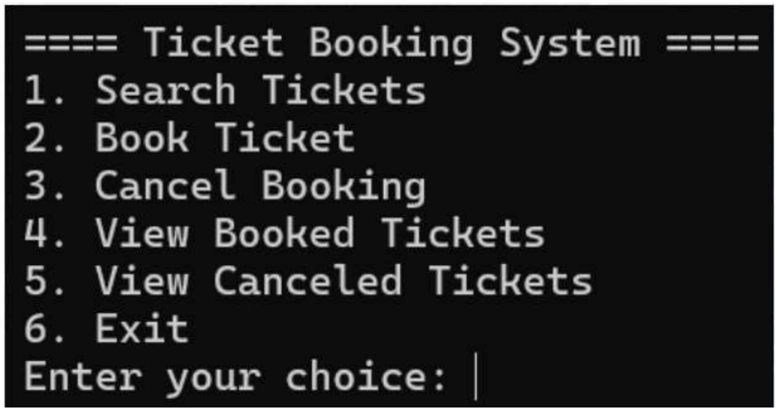
# OUTPUT

## AUTHENTICATION



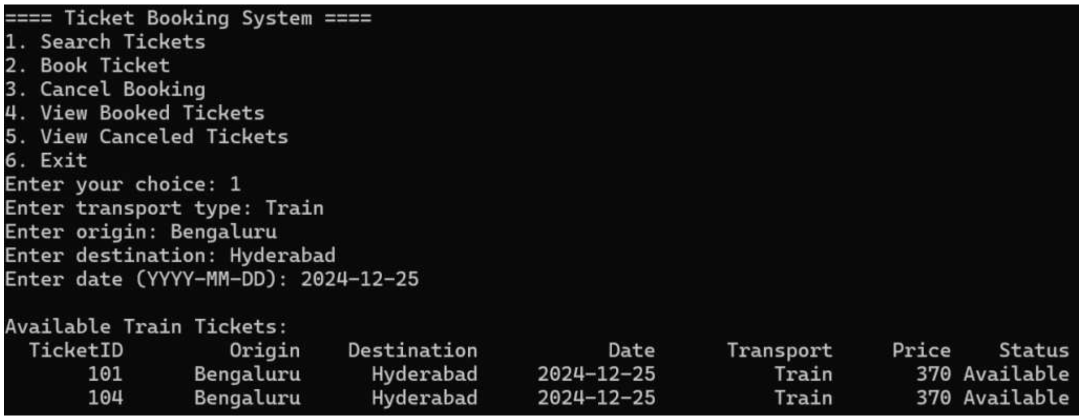
```
Do you have an account? (y/n): y
Enter username: sanju
Enter password: 1234
Login successful!
```

## MENU OPTIONS



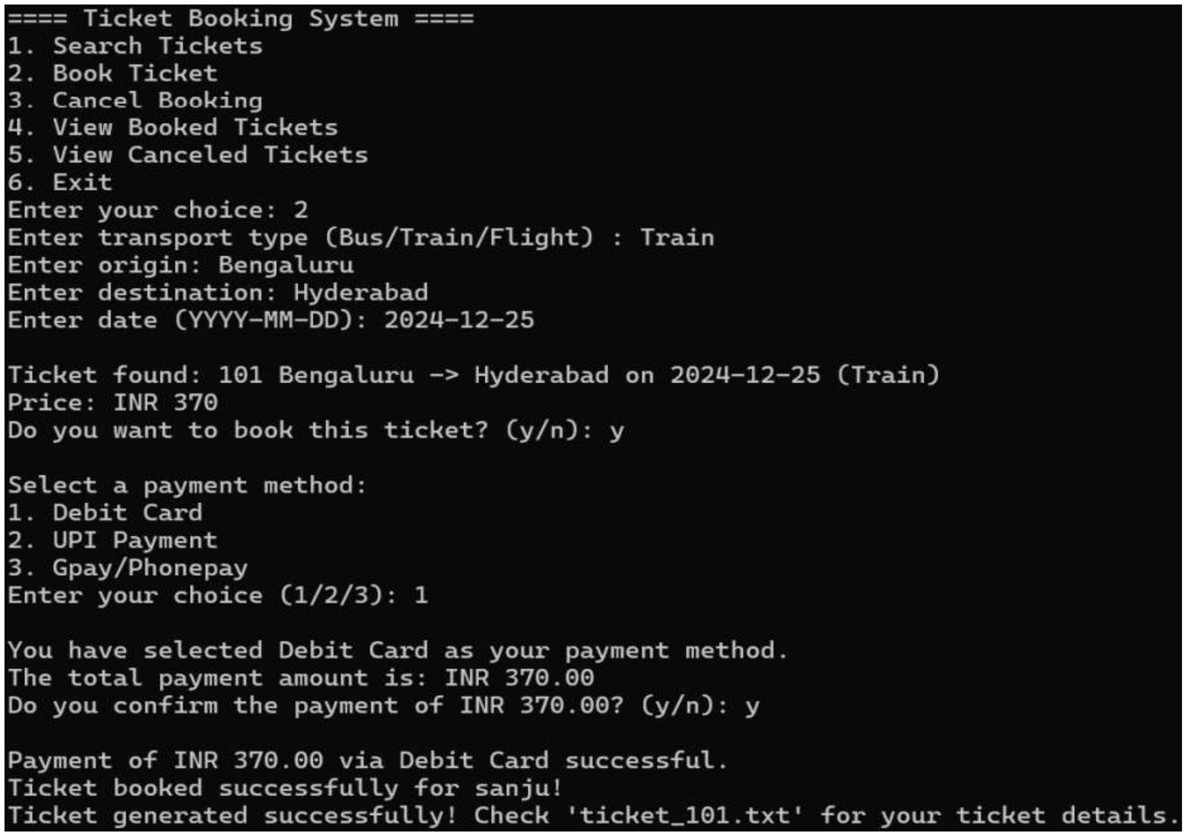
```
==== Ticket Booking System ====
1. Search Tickets
2. Book Ticket
3. Cancel Booking
4. View Booked Tickets
5. View Canceled Tickets
6. Exit
Enter your choice:
```

## SEARCH TICKETS



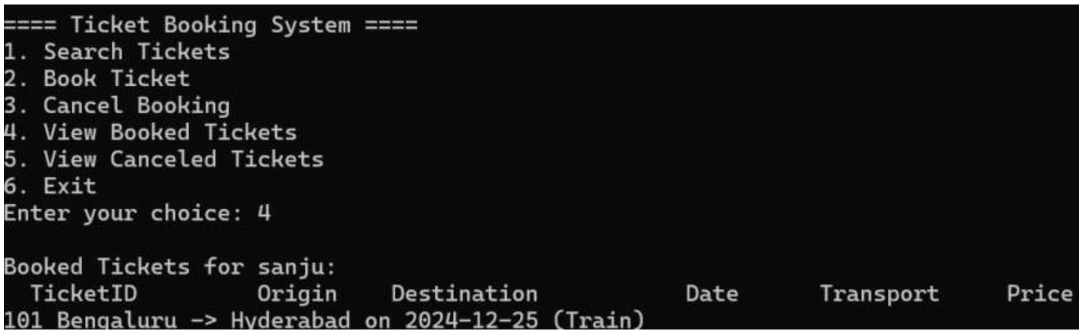
```
==== Ticket Booking System ====
1. Search Tickets
2. Book Ticket
3. Cancel Booking
4. View Booked Tickets
5. View Canceled Tickets
6. Exit
Enter your choice: 1
Enter transport type: Train
Enter origin: Bengaluru
Enter destination: Hyderabad
Enter date (YYYY-MM-DD): 2024-12-25

Available Train Tickets:
  TicketID         Origin    Destination          Date       Transport        Price    Status
       101      Bengaluru      Hyderabad    2024-12-25           Train          370 Available
       104      Bengaluru      Hyderabad    2024-12-25           Train          370 Available
```

## BOOK TICKETS



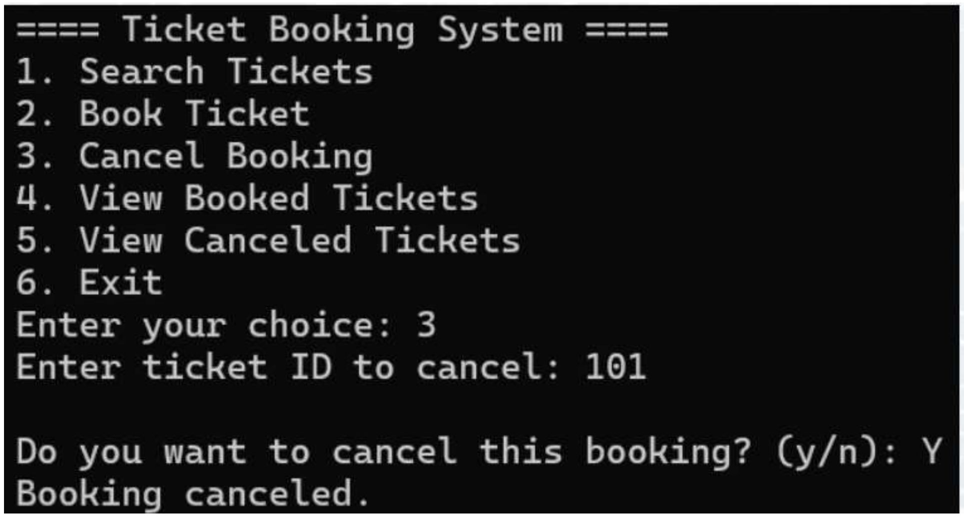
```
==== Ticket Booking System ====
1. Search Tickets
2. Book Ticket
3. Cancel Booking
4. View Booked Tickets
5. View Canceled Tickets
6. Exit
Enter your choice: 2
Enter transport type (Bus/Train/Flight) : Train
Enter origin: Bengaluru
Enter destination: Hyderabad
Enter date (YYYY-MM-DD): 2024-12-25

Ticket found: 101 Bengaluru -> Hyderabad on 2024-12-25 (Train)
Price: INR 370
Do you want to book this ticket? (y/n): y

Select a payment method:
1. Debit Card
2. UPI Payment
3. Gpay/Phonepay
Enter your choice (1/2/3): 1

You have selected Debit Card as your payment method.
The total payment amount is: INR 370.00
Do you confirm the payment of INR 370.00? (y/n): y

Payment of INR 370.00 via Debit Card successful.
Ticket booked successfully for sanju!
Ticket generated successfully! Check 'ticket_101.txt' for your ticket details.
```

## VIEW BOOKED TICKETS



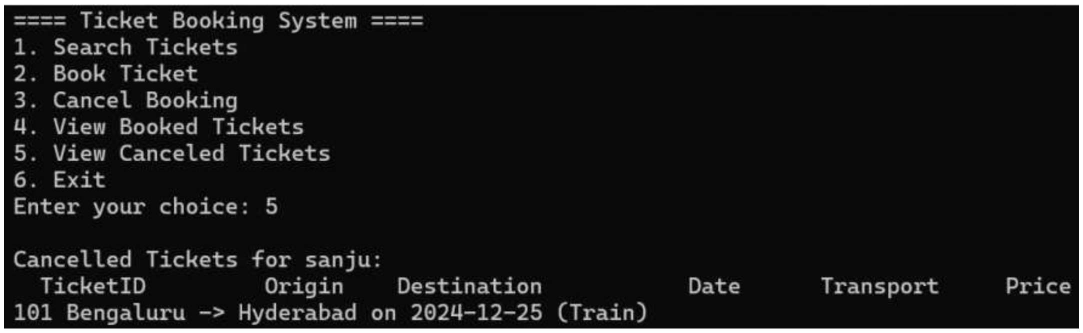
```
==== Ticket Booking System ====
1. Search Tickets
2. Book Ticket
3. Cancel Booking
4. View Booked Tickets
5. View Canceled Tickets
6. Exit
Enter your choice: 4

Booked Tickets for sanju:
  TicketID         Origin    Destination           Date      Transport     Price
101 Bengaluru -> Hyderabad on 2024-12-25 (Train)
```

**CANCEL TICKET**

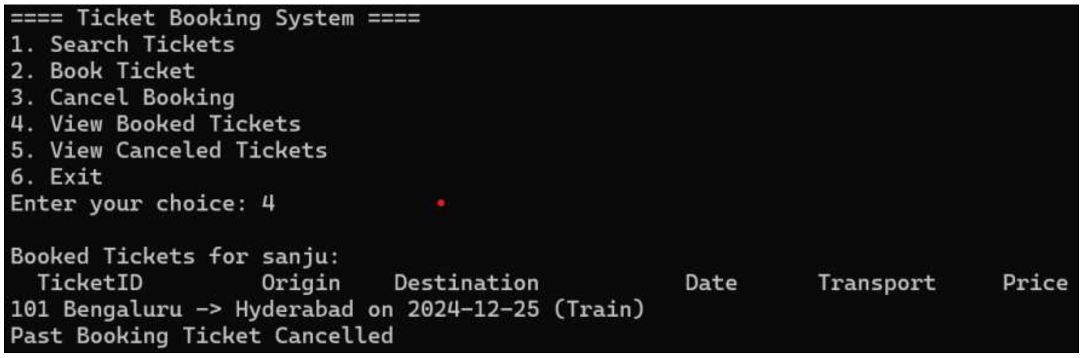


```
==== Ticket Booking System ====
1. Search Tickets
2. Book Ticket
3. Cancel Booking
4. View Booked Tickets
5. View Canceled Tickets
6. Exit
Enter your choice: 3
Enter ticket ID to cancel: 101

Do you want to cancel this booking? (y/n): Y
Booking canceled.
```

**VIEW CANCELLED TICKETS**



```
==== Ticket Booking System ====
1. Search Tickets
2. Book Ticket
3. Cancel Booking
4. View Booked Tickets
5. View Canceled Tickets
6. Exit
Enter your choice: 5

Cancelled Tickets for sanju:
  TicketID          Origin    Destination           Date      Transport     Price
101 Bengaluru -> Hyderabad on 2024-12-25 (Train)
```



```
==== Ticket Booking System ====
1. Search Tickets
2. Book Ticket
3. Cancel Booking
4. View Booked Tickets
5. View Canceled Tickets
6. Exit
Enter your choice: 4

Booked Tickets for sanju:
  TicketID          Origin    Destination           Date      Transport     Price
101 Bengaluru -> Hyderabad on 2024-12-25 (Train)
Past Booking Ticket Cancelled
```

## ADVANTAGES

- Simplifies ticket management for both users and administrators.
- Provides real-time updates to ticket availability.
- Offers flexibility in payment methods.
- Maintains user-specific records for accountability.
- Real-time ticket availability checks prevent overbooking
- The use of file-based storage eliminates the need for expensive database systems

## FUTURE ENHANCEMENT

- **Real-time Seat Availability:** Implement real-time updates for seat availability to prevent overbooking and ensure accurate bookings.

- **Payment Gateway Integration:** Integrate third-party payment systems for secure, real-time payment processing.

- **Mobile App Integration:** Develop a mobile app for Android/iOS to allow users to book and manage tickets on the go.

- **Admin Panel for Ticket Management:** Create an admin panel to manage routes, bookings, and cancellations efficiently.

## CONCLUSION

The Ticket Booking System developed as part of this project successfully enables users to search for, book, and cancel tickets for different modes of transportation such as buses, trains, and flights. The system ensures efficient management of ticket availability, payment handling, and booking history.
It provides a seamless user experience with features like real-time seat availability, booking confirmations, and detailed ticket generation. By implementing robust functionalities such as user authentication and file handling, this system showcases how technology can simplify ticket management processes. Future enhancements such as mobile app integration and real-time updates can further elevate the user experience, making the system even more efficient and user-friendly.

## REFERENCES

https://www.geeksforgeeks.org/
https://en.cppreference.com
https://github.com