# “IMPLEMENTATION OF TRAIN TICKET RESERVATION SYSTEM”

# TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| **CHAPTER** | **TITLE** | **PAGE NO.** |
| **1.** | **INTRODUCTION** |  |
| **2.** | **ALGORITHM** |  |
| **3.** | **DESCRIPTION OF THE PROJECT** |  |
| **4.** | **CONCEPT INVOLVED** |  |
| **5.** | **TOOLS BASED APPROACH** |  |
| **6.** | **SOFTWARE BASED APPROACH** |  |
| **7.** | **CODE** |  |
| **8.** | **RESULTS** |  |
| **9.** | **CONCLUSION** |  |

**CHAPTER 1 INTRODUCTION**

The project is a console-based Railway Reservation System developed in C++, designed to handle the core functionalities of booking, modifying, and managing train reservations. It simulates real-world railway operations using an intuitive menu-driven interface, making it accessible and educational for both users and developers. This system integrates fundamental concepts such as data structures, dynamic memory, input/output streams, and exception handling.

The system stores train-related information using well-defined structures: Train, Coach, Seat, and Reservation. These encapsulate all necessary attributes like seat number, availability, fare, passenger details, and more. The user can interact with the system to perform tasks such as:

* Adding trains and associating them with coaches and seats.
* Booking tickets with preferences (gender, food/drinks), Tatkal options, and age-based fare.
* Modifying and cancelling reservations based on unique passenger identification (name and age).
* Checking seat availability across coaches.
* Auto-confirmation of RAC and Waiting List passengers when confirmed seats become available.
* Viewing and organizing reservation data, classified as Confirmed, RAC, Waiting, and Tatkal.

In a broader context, this project demonstrates the utility of software systems in managing complex reservation processes typically handled by centralized transportation networks. By mimicking real-life constraints such as limited seats, reservation preferences, dynamic fare changes, and prioritized waiting lists, the system provides both a learning opportunity and a foundation for more advanced software implementations.

# CHAPTER 2 ALGORITHM

1. **Add Train:**

* Collect train metadata (name, source, destination, departure).
* Loop through the number of coaches, then through the number of seats in each coach.
* Initialize each seat with details like reservation status and seat number.
* Store the complete Train structure in a train vector.

1. **Reserve Seat:**

* Accept user input for train name, coach, and reservation preferences.
* Validate seat availability and coach existence.
* Collect passenger details, including food and drink choices.
* Automatically calculate fare based on age (with discounts for children and seniors).
* Adjust pricing and categorization for Tatkal bookings.
* Store reservation in appropriate list (Confirmed/RAC/Waiting).

1. **Display All Trains**:

* Traverse through the vector of trains and print associated coaches, seat availability, and fare details.

1. **Display Reservations**:

* Display all reservations under a selected category with passenger, train, and fare details.

1. **Cancel Reservation**:

* Search for passenger by name and age.
* On match, update seat status and remove from reservation list.
* Optionally trigger auto-confirm logic for RAC/Waiting list.

1. **Modify Reservation:**

* Search for reservation.
* Allow updates to source, destination, gender, or other attributes.
* Recalculate fare and reflect changes.

1. **Check Seat Availability:**

* Search for specific train and coach.
* Print count and list of all available seats.

1. **Auto-Confirm Reservations:**

* Automatically upgrade RAC and Waiting List passengers based on newly available seats.
* Prioritize RAC before Waiting List.

1. **Main Menu Loop:**

* Continuously prompt the user with options.
* Use switch-case logic to invoke corresponding functions.
* Loop until exit is selected.

# CHAPTER 3 DESCRIPTION OF THE PROJECT

This railway reservation project simulates a simplified booking system that emphasizes practical problem-solving through structured programming. It introduces core components necessary for reservation systems used in real-world railways, albeit in a smaller, demonstrative format.

**Key Features:**

1. **Train and Coach Management:**

* Each train can have multiple coaches.
* Coaches have different fare structures and seat counts.
* Designed for scalability—more coaches and trains can be added dynamically.

1. **Seat Reservation Logic:**

* Passengers specify train, coach, personal details, food/drink preferences.
* Supports Tatkal booking mode with price surges.
* Fare varies based on passenger age (child, adult, senior).

1. **Detailed Reservation Management:**

* Each reservation is stored and classified as Confirmed, RAC, Waiting, or Tatkal.
* Supports cancellation and data modification for flexibility.
* Displays reservation data in a clear, formatted output.

1. **Seat Availability Check:**

* Real-time display of seats left unreserved in any coach.

1. **Auto-Confirmation System:**

* Automates seat reassignment from the RAC and waiting list.
* Ensures maximum seat occupancy and fairness.

1. **User-Friendly Interface:**

* Simple console-driven options.
* Clean output formatting using iomanip.

1. **Randomization:**

* Random number generator (rand()) used in seat assignment to mimic unpredictability of real-time systems.

Overall, the project effectively captures the essential workflows of a railway reservation backend system, providing a practical introduction to how such systems operate behind the scenes.

# CHAPTER 4 CONCEPT INVOLVED

The development of this railway reservation system integrates several core programming and software engineering concepts. These form the backbone of the application’s structure, functionality, and efficiency.

**1. Structures (structs):**

* **Usage:** Represent entities like Train, Coach, Seat, and Reservation.
* **Significance:** Helps in bundling related data into single units, enabling modular and readable code.

**2. Vectors (std::vector):**

* **Usage:** Store dynamic collections such as lists of coaches in a train or seats in a coach.
* **Significance:** Allows dynamic memory allocation and resizing, essential for handling variable passenger and train data.

**3. Random Number Generation:**

* **Usage:** Assigns random seat numbers during reservation.
* **Significance:** Simulates unpredictability, mimicking real-world scenarios where exact seat allocation is system-determined.

**4. Object-Oriented Programming (OOP):**

* **Usage:** Though not classes, the use of structs and modular functions follows OOP principles like encapsulation and abstraction.
* **Significance:** Improves code organization, maintainability, and logical grouping of attributes and methods.

**5. File I/O (Planned for Expansion):**

* **Usage:** Can be used to save/load reservations between sessions (persistence).
* **Significance:** Allows users to retain data beyond one run of the program, making the system more practical.

**6. Input/Output Streams:**

* **Usage:** cin for user input, cout for displaying output.
* **Significance:** Provides real-time interaction and feedback for a console-based UI.

**7. Exception Handling:**

* **Usage:** Throws errors when invalid train/coach names or operations are encountered.
* **Significance:** Increases program robustness and prevents crashes from bad input.

**8. Algorithmic Thinking:**

* **Usage:** Logical processes for adding trains, reserving, modifying, auto-confirming, etc.
* **Significance:** Ensures operations are efficient, structured, and predictable.

**9. Menu-Driven Interface:**

* **Usage:** Guides users through operations using numbered choices.
* **Significance:** Increases usability and simplifies navigation for non-technical users.

**10. Dynamic Memory Allocation:**

* **Usage:** Implicit via vectors.
* **Significance:** Adapts to real-time user input and scalable data structures.

# CHAPTER 5

**TOOLS STUDY BASED APPROACH**

The development environment and associated tools play a key role in creating, testing, and managing the codebase.

**1. Programming Language:**

* **C++** was selected for its performance, structure, and support for procedural and OOP paradigms.

**2. Integrated Development Environment (IDE):**

* Used **Visual Studio**, **Code::Blocks**, or **Dev-C++** for compiling, debugging, and running the code.

**3. Version Control:**

* Tools like **Git** or **GitHub** can track changes, support backups, and aid collaboration.

**4. Code Editors:**

* **Visual Studio Code** and **Sublime Text** were used for lightweight editing with syntax highlighting.

**5. Compiler:**

* **GCC (GNU Compiler Collection)** for compiling C++ code efficiently across platforms.

**6. Debugging Tools:**

* Integrated debuggers in IDEs or tools like **GDB** for tracing logical errors and checking variable states.

**7. Random Number Generator:**

* The rand() function from <cstdlib> facilitates pseudo-random behavior in seat assignment.

**8. Documentation Tools:**

* Tools like **Doxygen** (or manual inline comments) help explain code logic and purpose.

**9. Testing Frameworks:**

* Tools like **Google Test** or **Catch2** can be integrated for unit testing core functions.

**10. Static Code Analysis Tools:**

* **CppCheck**, **Clang Static Analyzer** for catching bugs, memory leaks, and code inefficiencies.

**11. Build Automation Tools:**

* Systems like **CMake**, **Makefiles**, or **Ninja** can be used for structured project builds

# CHAPTER 6 SOFTWARE BASED APPROACH

The software approach is grounded in designing a modular, extensible, and efficient reservation system. It combines logic-driven design with real-time interaction.

**1. Modular Design:**

* Separate functions handle train addition, seat reservation, cancellations, etc., making the code easy to read and maintain.

**2. Object-Oriented Features:**

* Data abstraction and encapsulation via structures keep related data and logic grouped.

**3. Dynamic Data Structures:**

* Vectors enable dynamic resizing and are ideal for collections like passengers, seats, and coaches.

**4. Console-Based Interface:**

* Uses cin and cout with menu-driven design for ease of use.

**5. Error Handling:**

* Input validation and basic exception handling protect against crashes and unexpected behavior.

**6. Memory Efficiency:**

* Implicit dynamic allocation via STL vectors avoids wastage and adapts to user input.

**7. User Experience:**

* Input prompts are clear and options are logically grouped for better interaction.

**8. Algorithm Implementation:**

* Algorithms control logic flow for all operations such as booking, modifying, and confirming.

**9. Scalability:**

* Easily expandable to support multiple trains, more coaches, additional passenger features.

**10. Potential Add-ons:**

* File-based Persistence, Admin Panel, Ticket Printouts, Graphical UI can enhance the system further.

# CHAPTER 7

# CODE

# #include <iostream>

# #include <vector>

# #include <string>

# #include <iomanip>

# #include <cstdlib>

# #include <ctime>

# using namespace std;

# struct Seat {

# int seatNumber;

# bool reserved;

# string passengerName;

# int passengerAge;

# string foodChoice;

# string drinkChoice;

# };

# struct Coach {

# string coachName;

# float fare;

# vector<Seat> seats;

# };

# struct Train {

# string trainName;

# string source;

# string destination;

# string departureTime;

# vector<Coach> coaches;

# };

# struct Reservation {

# string pnr;

# string trainName;

# string coachName;

# int seatNumber;

# string passengerName;

# int passengerAge;

# float fare;

# string foodChoice;

# string drinkChoice;

# string status; // Confirmed / Tatkal / RAC / Waiting

# };

# vector<Train> trains;

# vector<Reservation> confirmedReservations;

# vector<Reservation> tatkalReservations;

# vector<Reservation> racList;

# vector<Reservation> waitingList;

# string generatePNR() {

# string pnr = "PNR";

# for (int i = 0; i < 6; ++i)

# pnr += to\_string(rand() % 10);

# return pnr;

# }

# void displayMenu() {

# cout << "\n==============================\n";

# cout << " TRAIN RESERVATION SYSTEM\n";

# cout << "==============================\n";

# cout << "1. Add Train\n";

# cout << "2. Reserve Seat\n";

# cout << "3. Display All Trains\n";

# cout << "4. Display Reservations\n";

# cout << "5. Cancel Reservation\n";

# cout << "6. Check Seat Availability\n";

# cout << "7. Auto confirm RAC and Waiting List Reservations\n";

# cout << "8. Exit\n";

# cout << "Enter your choice: ";

# }

# void addTrain() {

# Train t;

# cout << "\nEnter Train Name: ";

# cin >> t.trainName;

# cout << "Enter Source: ";

# cin >> t.source;

# cout << "Enter Destination: ";

# cin >> t.destination;

# cout << "Enter Departure Time: ";

# cin >> t.departureTime;

# int numCoaches;

# cout << "Enter number of coaches: ";

# cin >> numCoaches;

# for (int i = 0; i < numCoaches; i++) {

# Coach c;

# cout << "Enter Coach Name: ";

# cin >> c.coachName;

# cout << "Enter Fare: ";

# cin >> c.fare;

# int numSeats;

# cout << "Enter number of seats: ";

# cin >> numSeats;

# for (int j = 0; j < numSeats; j++) {

# Seat s;

# s.seatNumber = j + 1;

# s.reserved = false;

# c.seats.push\_back(s);

# }

# t.coaches.push\_back(c);

# }

# trains.push\_back(t);

# cout << "Train added successfully!\n";

# }

# void displayAllTrains() {

# for (const auto& t : trains) {

# cout << "\nTrain: " << t.trainName << " | From: " << t.source << " To: " << t.destination

# << " | Departs at: " << t.departureTime << endl;

# for (const auto& c : t.coaches) {

# int available = 0;

# for (const auto& s : c.seats)

# if (!s.reserved) available++;

# cout << "Coach: " << c.coachName << " | Fare: " << c.fare << " | Available Seats: " << available << endl;

# }

# }

# }

# void reserveSeat() {

# string trainName, coachName;

# cout << "\nEnter Train Name: ";

# cin >> trainName;

# cout << "Enter Coach Name: ";

# cin >> coachName;

# Train\* selectedTrain = nullptr;

# Coach\* selectedCoach = nullptr;

# for (auto& t : trains) {

# if (t.trainName == trainName) {

# selectedTrain = &t;

# for (auto& c : t.coaches) {

# if (c.coachName == coachName) {

# selectedCoach = &c;

# break;

# }

# }

# break;

# }

# }

# if (!selectedTrain || !selectedCoach) {

# cout << "Invalid train or coach name!\n";

# return;

# }

# string name, food, drink;

# int age;

# cout << "Enter Passenger Name: ";

# cin.ignore(); getline(cin, name);

# cout << "Enter Age: ";

# cin >> age;

# // Food ordering part

# string foodPreference;

# cout << "Would you like to order food? (yes/no): ";

# string foodOrder;

# cin >> foodOrder;

# vector<string> vegFoods = {"Paneer Butter Masala", "Dal Makhani", "Mixed Veg Curry", "Chole Bhature"};

# vector<string> nonVegFoods = {"Chicken Tikka", "Chicken Biryani", "Butter Chicken", "Mutton Rogan Josh"};

# if (foodOrder == "yes" || foodOrder == "Yes") {

# cout << "Enter food preference (veg/non-veg): ";

# cin >> foodPreference;

# if (foodPreference == "veg" || foodPreference == "Veg") {

# cout << "Available veg food options: ";

# for (auto& f : vegFoods) cout << f << ", ";

# cout << "\nEnter your choice exactly as above: ";

# cin.ignore();

# getline(cin, food);

# } else if (foodPreference == "non-veg" || foodPreference == "Non-veg" || foodPreference == "nonveg") {

# cout << "Available non-veg food options: ";

# for (auto& f : nonVegFoods) cout << f << ", ";

# cout << "\nEnter your choice exactly as above: ";

# cin.ignore();

# getline(cin, food);

# } else {

# food = "No Food";

# }

# } else {

# food = "No Food";

# cin.ignore();

# }

# // Drink ordering part

# cout << "Would you like to order a drink? (yes/no): ";

# string drinkOrder;

# cin >> drinkOrder;

# vector<string> drinks = {"Coke", "Sprite", "Fanta", "Lemonade"};

# if (drinkOrder == "yes" || drinkOrder == "Yes") {

# cout << "Available drink options: ";

# for (auto& d : drinks) cout << d << ", ";

# cout << "\nEnter your drink choice exactly as above: ";

# cin.ignore();

# getline(cin, drink);

# } else {

# drink = "No Drink";

# cin.ignore();

# }

# Reservation r;

# r.pnr = generatePNR();

# r.passengerName = name;

# r.passengerAge = age;

# r.foodChoice = food;

# r.drinkChoice = drink;

# r.trainName = trainName;

# r.coachName = coachName;

# bool seatFound = false;

# for (auto& seat : selectedCoach->seats) {

# if (!seat.reserved) {

# seat.reserved = true;

# seat.passengerName = name;

# seat.passengerAge = age;

# seat.foodChoice = food;

# seat.drinkChoice = drink;

# r.seatNumber = seat.seatNumber;

# if (age < 12 || age > 60)

# r.fare = selectedCoach->fare \* 0.5;

# else

# r.fare = selectedCoach->fare;

# cout << "Is this a Tatkal reservation? (y/n): ";

# char tatkal;

# cin >> tatkal;

# if (tatkal == 'y' || tatkal == 'Y') {

# r.fare \*= 1.2;

# r.status = "Tatkal";

# tatkalReservations.push\_back(r);

# } else {

# r.status = "Confirmed";

# confirmedReservations.push\_back(r);

# }

# cout << "Reservation successful! \n"

# << "PNR: " << r.pnr << "\n"

# << "Train: " << r.trainName << "\n"

# << "Coach: " << r.coachName << "\n"

# << "Seat: " << r.seatNumber << "\n"

# << "Passenger: " << r.passengerName << "\n"

# << "Age: " << r.passengerAge << "\n"

# << "Food: " << r.foodChoice << "\n"

# << "Drink: " << r.drinkChoice << "\n"

# << "Fare: " << r.fare << "\n"

# << "Status: " << r.status << "\n";

# seatFound = true;

# break;

# }

# }

# if (!seatFound) {

# r.fare = selectedCoach->fare;

# r.seatNumber = 0;

# if (racList.size() < 5) {

# r.status = "RAC";

# racList.push\_back(r);

# cout << "No seats available. Added to RAC. PNR: " << r.pnr << endl;

# } else {

# r.status = "Waiting";

# waitingList.push\_back(r);

# cout << "No seats available. Added to Waiting List. PNR: " << r.pnr << endl;

# }

# }

# }

# void displayReservations(const vector<Reservation>& resList, const string& title) {

# cout << "\n" << title << ":\n";

# if (resList.empty()) {

# cout << "No reservations.\n";

# return;

# }

# for (const auto& r : resList) {

# cout << "PNR: " << r.pnr << ", Train: " << r.trainName << ", Coach: " << r.coachName

# << ", Seat: " << (r.seatNumber == 0 ? "N/A" : to\_string(r.seatNumber)) << ", Name: " << r.passengerName

# << ", Age: " << r.passengerAge << ", Fare: " << r.fare << ", Food: " << r.foodChoice

# << ", Drink: " << r.drinkChoice << ", Status: " << r.status << endl;

# }

# }

# void cancelReservation() {

# string pnr;

# cout << "\nEnter PNR to cancel: ";

# cin >> pnr;

# for (auto it = confirmedReservations.begin(); it != confirmedReservations.end(); ++it) {

# if (it->pnr == pnr) {

# for (auto& t : trains) {

# if (t.trainName == it->trainName) {

# for (auto& c : t.coaches) {

# if (c.coachName == it->coachName) {

# for (auto& s : c.seats) {

# if (s.seatNumber == it->seatNumber) {

# s.reserved = false;

# break;

# }

# }

# }

# }

# }

# }

# confirmedReservations.erase(it);

# cout << "Reservation canceled for PNR: " << pnr << endl;

# return;

# }

# }

# cout << "Reservation not found for PNR: " << pnr << endl;

# }

# void checkSeatAvailability() {

# string trainName, coachName;

# cout << "\nEnter Train Name: ";

# cin >> trainName;

# cout << "Enter Coach Name: ";

# cin >> coachName;

# for (const auto& t : trains) {

# if (t.trainName == trainName) {

# for (const auto& c : t.coaches) {

# if (c.coachName == coachName) {

# cout << "Available seats: ";

# bool anyAvailable = false;

# for (const auto& s : c.seats) {

# if (!s.reserved) {

# cout << s.seatNumber << " ";

# anyAvailable = true;

# }

# }

# if (!anyAvailable)

# cout << "None";

# cout << endl;

# return;

# }

# }

# }

# }

# cout << "Train or coach not found.\n";

# }

# void autoConfirmRACandWaitingList() {

# // Confirm RAC passengers

# for (auto it = racList.begin(); it != racList.end();) {

# bool confirmed = false;

# for (auto& t : trains) {

# if (t.trainName == it->trainName) {

# for (auto& c : t.coaches) {

# if (c.coachName == it->coachName) {

# for (auto& s : c.seats) {

# if (!s.reserved) {

# s.reserved = true;

# s.passengerName = it->passengerName;

# s.passengerAge = it->passengerAge;

# s.foodChoice = it->foodChoice;

# s.drinkChoice = it->drinkChoice;

# it->seatNumber = s.seatNumber;

# it->status = "Confirmed";

# confirmedReservations.push\_back(\*it);

# it = racList.erase(it);

# confirmed = true;

# break;

# }

# }

# }

# if (confirmed) break;

# }

# }

# if (confirmed) break;

# }

# if (!confirmed) ++it;

# }

# // Confirm Waiting list passengers

# for (auto it = waitingList.begin(); it != waitingList.end();) {

# bool confirmed = false;

# for (auto& t : trains) {

# if (t.trainName == it->trainName) {

# for (auto& c : t.coaches) {

# if (c.coachName == it->coachName) {

# for (auto& s : c.seats) {

# if (!s.reserved) {

# s.reserved = true;

# s.passengerName = it->passengerName;

# s.passengerAge = it->passengerAge;

# s.foodChoice = it->foodChoice;

# s.drinkChoice = it->drinkChoice;

# it->seatNumber = s.seatNumber;

# it->status = "Confirmed";

# confirmedReservations.push\_back(\*it);

# it = waitingList.erase(it);

# confirmed = true;

# break;

# }

# }

# }

# if (confirmed) break;

# }

# }

# if (confirmed) break;

# }

# if (!confirmed) ++it;

# }

# cout << "Auto confirmation of RAC and Waiting List reservations done.\n";

# }

# int main() {

# srand(time(0));

# int choice;

# do {

# displayMenu();

# cin >> choice;

# switch (choice) {

# case 1: addTrain(); break;

# case 2: reserveSeat(); break;

# case 3: displayAllTrains(); break;

# case 4:

# displayReservations(confirmedReservations, "Confirmed Reservations");

# displayReservations(tatkalReservations, "Tatkal Reservations");

# displayReservations(racList, "RAC List");

# displayReservations(waitingList, "Waiting List");

# break;

# case 5: cancelReservation(); break;

# case 6: checkSeatAvailability(); break;

# case 7: autoConfirmRACandWaitingList(); break;

# case 8: cout << "Exiting...\n"; break;

# default: cout << "Invalid choice!\n";

# }

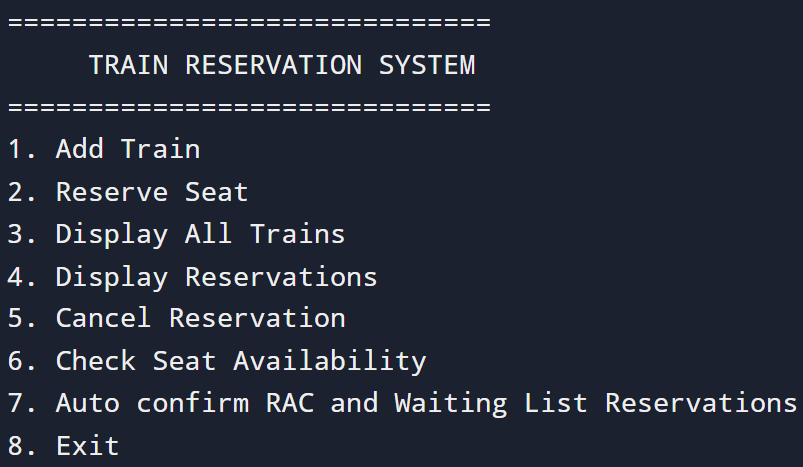
# } while (choice != 8);

# return 0;

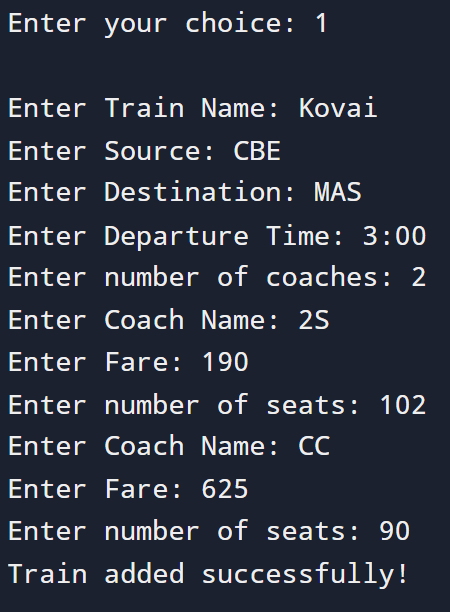
# }

# CHAPTER 8 RESULT

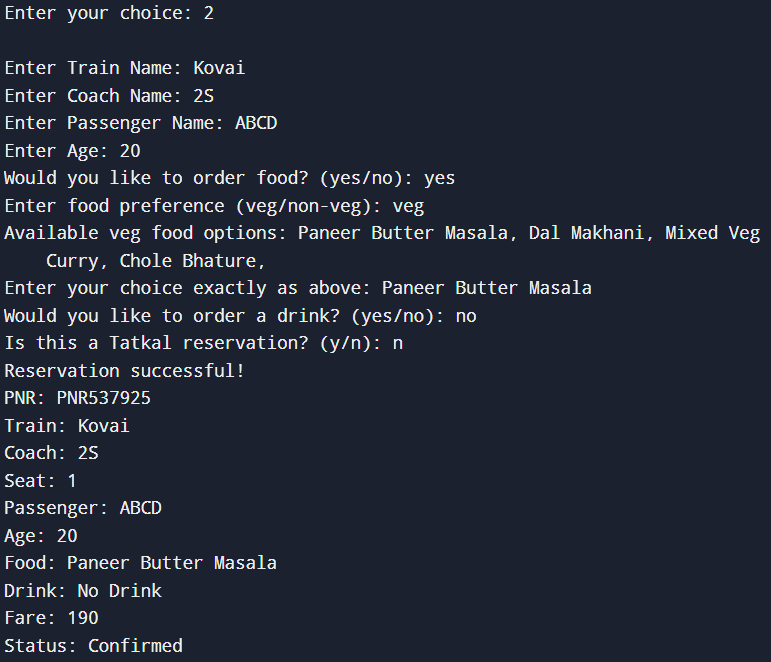
1. Display Panel



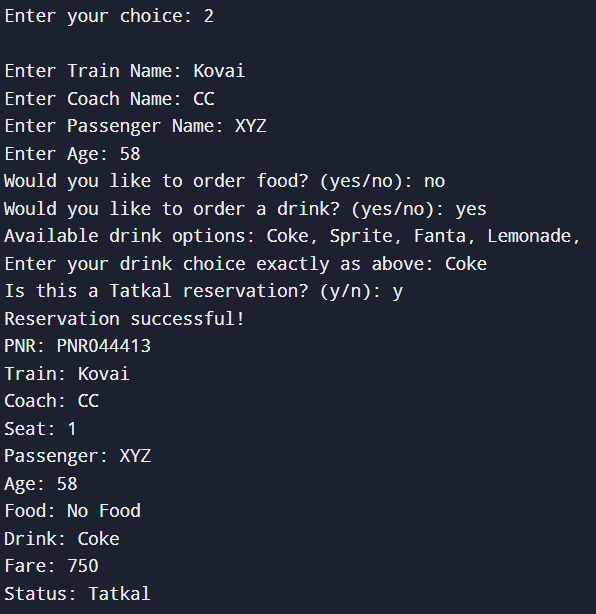
1. Adding Train Details



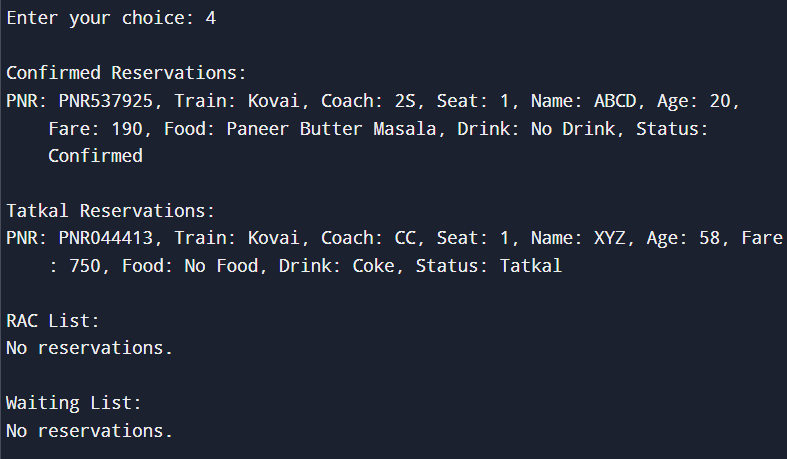
1. Booking Ticket (Normal):



1. Booking Ticket (Tatkal):



1. Display Reservation:



# CHAPTER 9

**CONCLUSION**

In conclusion, this Railway Reservation System built in C++ successfully demonstrates how core programming concepts can be applied to simulate a real-world application. It efficiently captures user input, maintains structured records, dynamically handles reservations, and models system behavior such as auto-confirmation from RAC and Waiting List.

From a development perspective, the system exhibits:

* Modular design
* Data abstraction
* Efficient memory handling
* Dynamic data structures
* User-friendly console interaction

It lays a solid foundation for a more comprehensive reservation system with potential for:

* GUI integration
* Database back-end
* Persistent file storage
* Online connectivity

The project not only serves as an excellent exercise in C++ programming, system design, and data structure application but also offers room for expansion into a production-grade software model. It illustrates how software can be a powerful tool in streamlining operations in industries like transportation and logistics.