Public Transportation System: Project Documentation

Tran Le Dung (MSSV: 24110084)

September 14, 2025

1 Project Overview

This Public Transportation System (PTS) is a C++ program I built to handle all sorts of transportation tasks, like managing vehicles, stations, routes, passengers, and schedules. It's designed with Object-Oriented Programming (OOP) concepts in mind, showcasing things like encapsulation, inheritance, and polymorphism. The system has a user-friendly menu that lets you add vehicles, book tickets, manage stations, and even check revenue reports.

2 System Features

Here's what the system can do:

- **Vehicle Management**: Add and view vehicles like ExpressBus, Metro, Train, and Bike, each with details like route, capacity, and ticket price.
- **Station Management**: Create and list stations, including their names and locations.
- **Route Management**: Set up and display routes that connect different stations.
- **Passenger Management**: Book tickets for passengers on specific vehicles, with checks to avoid overbooking.
- **Schedule Management**: Create and show schedules that link vehicles to stations and times.
- **Revenue Reporting**: Track ticket sales and calculate total revenue for all vehicles.
- **Travel Time Calculation**: Figure out travel times based on distance and vehicle-specific speeds (like a custom speed for ExpressBus).

3 Object-Oriented Design

The system is built around OOP principles to keep things organized and flexible:

- **Encapsulation**: I kept attributes like `route`, `capacity`, and `bookedSeats` private inside classes, with public methods like `bookTicket` to control access.
- **Inheritance**: There's a `Vehicle` base class that other classes (`ExpressBus`, `Metro`, `Train`, `Bike`) inherit from to share common features.

- **Polymorphism**: Methods like `calculateTravelTime` and `displayInfo` are virtual, so each vehicle type can have its own version of them.
- **Abstraction**: The `Vehicle` class simplifies things by providing a common way to handle all vehicle types using pointers.

The main classes are:

- `Vehicle`: The parent class with attributes like `route` and `ticketPrice`, plus methods for booking and displaying info.
- `ExpressBus`, `Metro`, `Train`, `Bike`: Child classes that tweak things like travel time calculations or display formats.
- `Station`, `Route`, `Passenger`, `Schedule`: Standalone classes to handle other parts of the system.

4 Code Structure

The code is split into neat, manageable chunks:

- **Class Definitions**: The `Vehicle` hierarchy and classes like `Station` and `Route` are set up with clear roles.

```
class Vehicle {
protected:
    string route;
    int capacity;
    int bookedSeats;
    double ticketPrice;
public:
    Vehicle(string r, int c, double price);
    virtual double calculateTravelTime(double distance);
    virtual void displayInfo();
    bool bookTicket();
};
```

- **Global Data**: I used vectors to store `Vehicle` pointers and objects for `Station`, `Route`, `Passenger`, and `Schedule`, making it easy to manage everything dynamically.
- **Menu System**: The main menu and sub-menus (like `vehicleMenu` and `passengerMenu`) let users interact with the system, with `system("cls")` to keep the screen tidy.

```
void vehicleMenu() {
    int choice;
    do {
        system("cls");
        cout << "\n===== Vehicle Menu =====\n";
        cout << "1. Add Vehicle\n";
        cout << "2. View All Vehicles\n";
        cout << "3. Display Tickets\n";
        cout << "0. Back\n";
        cin >> choice;
        // Handle choices
    } while (choice != 0);
```

```
}
```

- **Initialization**: The `initData` function sets up some sample vehicles, stations, routes, and schedules to test the system.

5 Sample Output

I tested the system with some initial data and user inputs. Here's what it looks like:

- **Vehicle List**:

```
=== Vehicle List ===
[ExpressBus] Route: RouteA | Speed: 60 | Capacity: 2 | Booked: 0 | Ti
[Metro] Route: RouteB | Capacity: 100 | Booked: 0 | Ticket: $1.5
[Train] Route: RouteC | Capacity: 200 | Booked: 0 | Ticket: $10
[Bike] Route: RouteD | Capacity: 1 | Booked: 0 | Ticket: $0.5
```

  This shows how each vehicle type displays its own details, thanks to polymorphism.

- **Ticket Booking**:

```
Enter passenger name: Bob
Choose vehicle index:
0. [ExpressBus] Route: RouteA | Speed: 60 | Capacity: 2 | Booked: 0 |
1
 Ticket booked successfully for route RouteA
```

  This confirms the system checks capacity and books tickets correctly.

- **Revenue Report**:

```
=== Revenue Report ===
[Tickets] Route: RouteA | Sold: 1 | Remaining: 1 | Capacity: 2 | Reve
[Tickets] Route: RouteB | Sold: 0 | Remaining: 100 | Capacity: 100 |
[Tickets] Route: RouteC | Sold: 0 | Remaining: 200 | Capacity: 200 |
[Tickets] Route: RouteD | Sold: 0 | Remaining: 1 | Capacity: 1 | Reve
Total Revenue: $5
```

  This proves the system tracks ticket sales and calculates revenue accurately.

6 AI Usage

I got some help from Grok (made by xAI) for a few things:

- Figuring out the vehicle class hierarchy by asking: *"What's a good inheritance structure for vehicles in a transportation system?"*
- Getting ideas for making the menu system user-friendly and validating inputs.
- Brainstorming ways to handle revenue calculations and display different vehicle types.

Grok gave me a starting point for the `Vehicle` classes and menu layout, but I tailored everything to fit my project, like adding `ExpressBus` and the `calculateTravelTime` method. I wrote and tested all the code myself to make sure it worked.

7 Conclusion

This Public Transportation System project was a great way to dive into OOP and build something useful. Using inheritance and polymorphism let me handle different vehicle types smoothly while keeping the code clean and organized. The menu system and dynamic data handling made the program easy to use and flexible. Working on this taught me a ton about encapsulation, abstraction, and how objects interact, and tackling challenges like memory management and input validation really leveled up my C++ skills. I'm proud of how the system came together, and it feels like a solid base for adding more features down the road. This project showed me how powerful OOP can be for real-world applications like transportation management.