DSA Semester Project

Ride-Sharing Dispatch & Trip Management System

This project focuses on designing and implementing an in-memory ride-sharing system similar to popular platforms such as Uber or Careem. While the problem domain is familiar, the internal implementation requires advanced data structures, careful state management, and rollback handling.

Objective

The objective of this project is to enhance the understanding of graphs, state machines, rollback mechanisms, and modular software design by utilizing custom data structures. Students will also learn to collaborate effectively and use AI tools responsibly.

Team Structure

This is a 2-student group project.
Both students are responsible for system integration, testing, and documentation.

System Overview

The system simulates a ride-sharing service where riders request trips and drivers are assigned based on proximity. Trips move through a well-defined lifecycle and can be cancelled or rolled back. The system maintains a complete history of trips for analysis.

Core Functional Requirements

1. **Zone-Based City Partitioning**
    - **City represented as a** weighted graph
    - **Divide the city into**: Zones / sectors
        - Nodes = locations
        - Edges = roads with distance
    - **Implement**:
        - Shortest path algorithm
        - Custom adjacency structure
2. **Driver and Rider management**
    - **Drivers**:
        - Location
        - Availability
    - **Riders**:
        - Pickup & drop-off points

    - Drivers are:
    - Assigned primarily within their zone
    - Cross-zone assignment costs more
3. **Trip lifecycle state management**
    - Each trip transitions through
        - REQUESTED → ASSIGNED → ONGOING → COMPLETED
    - Or:
        - REQUESTED → CANCELLED
        - ASSIGNED → CANCELLED
    - State transitions must be valid and enforced.
4. **Trip cancellation and rollback**
    - Cancelling a trip must:
        - Restore driver availability
        - Restore system state
    - Roll back the last `k` operation
5. **Trip history and analytics**
    - Maintain a history of all trips
    - Support queries
        - Average trip distance
        - Driver utilization
        - Cancelled vs completed trips

Implementation Constraints

- No STL graph or map containers for core logic
- No external map or routing APIs
- No global variables
- Multi-file implementation is mandatory
- Header files must contain declarations only

Required File Structure

City.h / City.cpp
Driver.h / Driver.cpp
Rider.h / Rider.cpp
Trip.h / Trip.cpp
DispatchEngine.h / DispatchEngine.cpp
RollbackManager.h / RollbackManager.cpp
RideShareSystem.h / RideShareSystem.cpp
main.cpp

Testing Requirements

A minimum of 10 test cases must be implemented, including:

- Shortest path correctness
- Driver reassignment after cancellation
- Multiple trip rollbacks
- Invalid state transition handling
- Analytics correctness after rollback

Documentation

Students must submit a design.md file explaining:

- Graph representation and routing approach
- Trip state machine design
- Rollback strategy
- Time and space complexity

Evaluation Criteria

Evaluation will be based on correctness, data structure usage, modularity, collaboration, testing quality, and documentation depth.

Important Note on AI Usage

AI tools can be used however you want, but knowing what is happening in the project is important.