# A black and red text Description automatically generated

**PROJECT AND TEAM INFORMATION**

## Project Title

|  |
| --- |
| METRO RAIL ROUTE SYSTEM |

## Student/Team Information

|  |  |
| --- | --- |
| Team Name:  Team # | Metro Minds  DAA-IV-T085 |
| Team member 1 (Team Lead) | Kunwri,Jahnavi-23022824  [23022824@geu.ac.in](mailto:23022824@geu.ac.in) |
| Team member 2  Team member 3 | Vats,Disha-23022703  [23022703@geu.ac.in](mailto:23022703@geu.ac.in)    Rawat,Keshav- 230211351  [230211351@geu.ac.in](mailto:230211351@geu.ac.in) |

**PROJECT PROGRESS DESCRIPTION**

## Project Abstract

|  |
| --- |
| The **Delhi Metro Route Planner** is a console-based C++ application designed to help commuters find the shortest and most efficient route between metro stations in Delhi. The system models the metro network as a weighted undirected graph, where each station represents a node and the travel time between stations forms the weighted edges.  The core functionality employs **Dijkstra’s algorithm** to compute the shortest travel time between any two stations. The program also calculates the fare based on a per-minute rate, making the solution practical for fare estimation. The user interface, built on the Windows console, is enhanced with colored text for better visual clarity and user experience. This project demonstrates the application of graph theory and priority queues for solving real-world routing problems with clear, interactive console menus. |

## Updated Project Approach and Architecture

|  |
| --- |
| The Delhi Metro Route Planner models the entire metro network as a weighted undirected graph using an adjacency list, where each node represents a metro station, and edges carry the travel time between connected stations. Station names are stored in a vector for easy lookup, and the connections are represented as pairs of adjacent station IDs with their associated travel times. The core routing algorithm is **Dijkstra’s shortest path**, which efficiently computes the minimum travel time between any two given stations using a min-priority queue to select the next closest station during traversal.  The system is encapsulated within the MetroSystem class, which handles station data management, route calculation, fare computation based on a per-minute rate, and user interaction through a console-based interface. The interface is enhanced with colored text for clarity and improved user experience, implemented using Windows console API functions. The modular design of the project allows for easy extension, such as adding new stations, updating fare rates dynamically, and potentially integrating a graphical user interface or file-based data persistence in future versions.    The class diagram illustrates the structure of a C++ Metro Rail Route System, showcasing the relationships between different classes. At the top is the "C++ Metro Rail Route System" class, which has three child classes: "StationManager," "RouteManager," and "TrainManager." "RouteManager" and "TrainManager" further branch into a "MainController" class, which then leads to "UserInterface" and "Database/FileManager" classes. This structure visualizes how the system is organized, with the top-level class managing the overall system and its components. |

## 

Tasks Completed

|  |  |
| --- | --- |
| Task Completed | Team Member |
| Graph structure setup and station mapping  Dijkstra's algorithm for shortest path  Console GUI with color formatting  Fare calculation system  Interactive menu logic  Input handling and validation  Testing and debugging | Jahnavi Kunwri  Disha Vats  Keshav Rawat  Disha Vats  Keshav Rawat  Jahnavi Kunwri  All members |

## Challenges/Roadblocks

|  |
| --- |
| Modeling the Delhi Metro network accurately was challenging due to multiple lines and interchange stations. Representing this complex structure as a weighted graph needed careful design to handle connections and station indexing properly.  Implementing Dijkstra’s algorithm efficiently required optimizing priority queue operations and debugging distance updates. Ensuring the algorithm consistently found the shortest path demanded extensive testing.  Handling user inputs was difficult because station names could be entered with typos or different cases. Developing a reliable input validation system to manage these variations without crashing the program was essential.  Calculating fares based on travel time was another challenge. The fare module had to integrate tightly with the route planner to ensure the price reflected the exact shortest travel duration, requiring precise synchronization.  Adding colored text to improve the console interface introduced platform-specific issues. Using Windows API for color control reduced portability and needed careful handling to avoid display problems.  Memory management was a concern as all data was stored in memory. Scaling the system for larger datasets or real-time updates would require additional storage solutions like databases, increasing complexity.  Balancing simplicity and extensibility was a continuous challenge. Planning for future features such as GUI or live updates meant designing a flexible architecture without overcomplicating the initial implementation. |

## Tasks Pending

|  |  |
| --- | --- |
| Task Pending | Team Member (to complete the task) |
| Add file-based or database station saving  Expand metro network to 50+ stations  Add path logging/export feature  Route history or favorite path system  Add error handling for non-integer input  Connect future enhancements via files | Jahnavi Kunwri  Disha Vats  Keshav Rawat  Jahnavi Kunwri  Disha Vats  All members |

## 

## Project Outcome/Deliverables

|  |
| --- |
| The primary outcome of the Delhi Metro Route Planner project is a fully functional C++ application that models a real-world metro route system. It allows users to view all available metro stations, find the shortest route between any two stations using Dijkstra's algorithm, and calculate the travel fare based on a per-minute rate. The application successfully implements graph data structures, a priority queue, and robust input handling, all within a console-based user interface. It provides a smooth and interactive experience with colored prompts, ensuring ease of use for both technical and non-technical users.  Additional deliverables include modular code through the MetroSystem class, support for dynamic fare rate modification, and complete route visualization from source to destination. The project serves as a practical demonstration of how core data structures and algorithms can be applied to solve real-life routing problems. Although optional features like GUI and persistent data storage are not yet implemented, the existing application is ready for those extensions. Overall, the deliverables meet the initial objectives and lay a solid foundation for further development or integration into a larger metro navigation system. |

# Progress Overview

|  |
| --- |
| Approximately **85% of the core system functionality** has been completed. The main features, including the implementation of Dijkstra’s algorithm for shortest route calculation, the station graph structure, the console-based user interface, and the fare calculation module, are all fully developed and tested. The application is capable of handling different route scenarios, validating user inputs, and providing accurate travel times and fare estimates. We have also ensured that the interface is user-friendly through the use of colored text formatting for better readability and interaction, which enhances the overall experience despite being a console-based program.  The remaining tasks focus primarily on **optional enhancements** that can make the system more scalable and robust. These include upgrading from a console interface to a graphical user interface (GUI), integrating file handling for persistent station and route data, and optimizing the codebase for larger metro networks. While these additions are not essential for the core functionality, they will significantly improve usability, maintainability, and future extensibility. Once these features are integrated, the application will be more polished and capable of handling real-world complexity with ease. The team is confident that the final deliverable will not only meet the initial objectives but also demonstrate thoughtful design and adaptability |

# Codebase Information

|  |
| --- |
| The Delhi Metro Route Planner codebase is primarily developed in C++, leveraging the Standard Template Library (STL) for efficient data management. Core data structures include unordered\_map for storing station nodes and their connections, vector for adjacency lists representing graph edges, and priority\_queue to implement Dijkstra’s shortest path algorithm. The modular design separates graph construction, user input handling, route calculation, and fare computation into distinct functions to maintain clarity and ease of maintenance.  User interaction is handled through a simple console interface, allowing users to input source and destination stations by name. The program processes inputs with case-insensitive checks and basic validation to ensure robustness. Output includes the shortest route with intermediate stations, total travel time, and corresponding fare. The codebase is structured to support easy extension for features such as GUI integration or real-time data updates, making it adaptable for future enhancements.  GitHub link-  https://github.com/disha07075/METRO-RAIL-ROUTE-SYSTEM/tree/main |

## 

## Testing and Validation Status

|  |  |  |
| --- | --- | --- |
| Test Type | Status (Pass/Fail) | Notes |
| Route calculation accuracy  Fare calculation  Input handling  Edge case: same start/end  No-path scenario | Pass  Pass  Pass  Pass  Pass | Correct route identified using sample inputs  Accurate calculation for all route scenarios  Catches invalid station IDs and bad inputs  Message shown; route not calculated  "No route found" message displayed properly |

# Deliverables Progress

|  |
| --- |
| C++ Route Planner Core- Completed  Fare Calculation Logic- Completed  Console-based User Interface- Completed  Data Persistence/Export Features- In progress |