Logo, company name

Description automatically generated

***PRIOJECT***

*Cab Management System*

***Subject:***

**dsa**

***Submitted by:***

SYED HAMZA

Sp22-BCS-099

USMAN AKRAM

Sp22-BCS-103

***Submitted to:***

Dr. Inayat ur Rehman

Our project involves various functionalities related to managing users, captains, and rides in a transportation system. The project includes the implementation of Dijkstra's algorithm to find the shortest route between different spots in the transportation network.

Structure Division:

Hamza-099

->Multiple Linked List

->Queue

Usman Akram-103

->Graph

->Stack

Key Features:

1. User Struct:
   * Represents a user in the transportation system.
   * Stores user-related information such as name, username, password, and contact number.
   * Provides a method to display user details.
2. Ride Struct:
   * Represents a ride in the transportation system.
   * Stores ride-related information such as distance, fare, rider name, route, and ride status.
   * Provides a method to display ride details.
3. Captain Struct:
   * Represents a captain (driver) in the transportation system.
   * Stores captain-related information such as name, car name, registration number, car type, and status.
   * Maintains a linked list of rides associated with each captain.
   * Provides methods to display captain details and associated rides.
4. Graph:
   * Defines a 2D array representation of the transportation network.
   * Contains the distances between different spots (vertices) in the network.
   * Utilizes Dijkstra's algorithm to find the shortest path between two spots.
5. Functions:
   * enqueue: Adds a ride to the queue.
   * initVisited: Initializes the visited array for DFS traversal.
   * dfs: Performs Depth-First Search (DFS) traversal to mark visited spots.
   * minDistance: Finds the minimum distance vertex from the set of vertices not yet included in the shortest path tree.
   * printPath: Recursively prints the shortest path from source to destination.
   * printSolution: Calls printPath to display all possible paths and their distances.
   * dijkstrasAlgorithm: Implements Dijkstra's algorithm to find the shortest route between source and destination.
   * addCaptain: Adds a captain to the transportation system.
   * displayCaptains: Displays details of all captains in the system.
   * displayEveryDetail: Displays details of all captains and their associated rides.
   * removeCaptain: Removes a captain from the system.
   * addUser: Adds a user to the transportation system.
   * displayUsers: Displays details of all users in the system.
   * removeUser: Removes a user from the system.

Usage:

1. User Management:
   * Users can be added using the **addUser** function by providing the necessary information.
   * Existing users can be displayed using the **displayUsers** function.
   * Users can be removed using the **removeUser** function by selecting the appropriate serial number.
2. Captain Management:
   * Captains can be added using the **addCaptain** function by providing the necessary information.
   * Existing captains can be displayed using the **displayCaptains** function.
   * Captains can be removed using the **removeCaptain** function by selecting the appropriate serial number.
3. Ride Management:
   * Rides can be enqueued using the **enqueue** function by providing the necessary ride details.
   * The shortest route between two spots can be calculated using the **dijkstrasAlgorithm** function.
   * The calculated route and distance can be displayed using the **printSolution** function.

Limitations and Future Enhancements:

1. The code lacks error handling and input validation, which may lead to unexpected behavior or crashes if incorrect input is provided.
2. The code does not include mechanisms for persistent storage of user, captain, and ride data. Therefore, the data will be lost once the program terminates.
3. The code does not implement features for modifying or updating user, captain, or ride information after their creation.