UML Documentation

This UML diagram describes the main classes and
their interfaces for a C++ project that manages a map of
cities and roads, allowing users to find the shortest path between cities,
update the map, and traverse it using various algorithms. The system is modular,
with each class responsible for a specific aspect of the program.

Class Summaries

GraphManager

Manages the core graph data using an adjacency list.
Provides methods to add/delete cities and edges, display the graph, and perform shortest path algorithms (Dijkstra, A*, Floyd).
Handles loading and saving graph data to files for persistence.

Dijkstra

Implements Dijkstra’s algorithm for shortest path calculation.
Stores distances, visited status, and previous nodes for path reconstruction.

GraphDisplay

Responsible for displaying the graph and shortest paths in a user-friendly format.

GraphTraverse

Implements traversal algorithms: Breadth-First Search (BFS) and Depth-First Search (DFS).

GraphController

Provides an alternative interface for managing the graph, including adding/removing cities and edges, displaying the graph, and finding shortest paths.

UserLogin

Manages user authentication and registration.
Validates user credentials and handles login/logout functionality.

Project Documentation (Summary)

This project is a C++ console application that allows users to model a map as a graph (using an adjacency list), representing cities as nodes and roads as weighted edges.
Users can:

Add, update, and delete cities and roads.
Display the current map.
Traverse the map using BFS or DFS.
Find the shortest path between two cities using Dijkstra’s algorithm (with total distance and path display).
Save and load the map data to/from files for persistence.
Log in and manage user sessions.
The system uses multiple data structures (unordered_map, list, vector, etc.) to efficiently support the required algorithms and operations.
The design is modular,
separating concerns such as graph management, traversal, display, and user authentication

The system uses multiple data structures (unordered_map, list, vector, etc.) to efficiently support the required algorithms and operations.
The design is modular, separating concerns such as graph management, traversal, display, and user authentication.

Note:
The UML diagram provides a clear overview of each class’s interface (public methods and attributes), which aligns with the project requirements for modularity, extensibility, and clean code. Each class encapsulates a specific responsibility, making the system easy to maintain and extend.

```
+---------------------------------------+
|           GraphManager                |
+---------------------------------------+
| - unordered_map<string, int> dist     |
| - unordered_map<string, string> prev  |
| - unordered_map<string, list<pair<    |
|     string, int>>> adjacencyList      |
+---------------------------------------+
| + AddCity(cityName: string): void     |
| + AddEdge(city1: string, city2:       |
|     string, distance: int): void      |
| + DeleteEdge(city1: string, city2:    |
|     string): void                     |
| + DeleteCity(cityName: string): void  |
| + GraphManager():                     |
| + ~GraphManager():                    |
| + DisplayGraph(): void                |
| + loadFromJson(filename: string):     |
|     void                              |
| + saveToJson(filename: string) const:|
|     void                              |
| + dijkstra(start: string, end:        |
|     string): void                     |
| + aStar(start: string, end: string):  |
|     void                              |
| + floyd(start: string&, end: string&):|
|     void                              |
| + getAdjacencyList() const: const     |
|     unordered_map<string, list<       |
|     pair<string, int>>>&              |
| + PrintSolution(start: string, end:   |
|     string): void                     |
+---------------------------------------+
+----------------------------------------------------+
|                   Dijkstra                         |
+----------------------------------------------------+
| - unordered_map<string, int> dist                  |
| - unordered_map<string, bool> visited              |
| - unordered_map<string, string> prev               |
+----------------------------------------------------+
| + dijkstra(graph: const GraphManager&,             |
|     startCity: string, endCity: string): void      |
| + printSolution(startCity: string,                 |
|     endCity: string) const: void                   |
+----------------------------------------------------+


+----------------------------------------------------+
|                GraphDisplay                        |
+----------------------------------------------------+
| - const GraphManager& graphManager                 |
+----------------------------------------------------+
| + GraphDisplay(gm: const GraphManager&)            |
| + displayGraph() const: void                       |
| + displayShortestPath(path: const vector<string>&,|
|     distance: double) const: void                  |
+----------------------------------------------------+



+----------------------------------------------------+
|                GraphTraverse                       |
+----------------------------------------------------+
| - const GraphManager& graphManager                 |
+----------------------------------------------------+
| + GraphTraverse(gm: const GraphManager&)           |
| + BFS(StartCity: string): void                     |
| + DFS(graph: unordered_map<string, string>,        |
|     path: vector<string>,                          |
|     paths: vector<vector<string>>&,                |
|     startcity: string): void                       |
+----------------------------------------------------+
+-----------------------------------------------------+
|                GraphController                      |
+-----------------------------------------------------+
| - unordered_map<string, list<pair<string, int>>>    |
|     mainMap                                         |
+-----------------------------------------------------+
| + getMap() const: unordered_map<string, list<       |
|     pair<string, int>>>&                            |
| + GraphController()                                 |
| + ~GraphController()                                |
| + addCity(cname: string): void                      |
| + addEdge(from: string, to: string, weight: int):   |
|     void                                            |
| + deleteEdge(from: string, to: string): void        |
| + removeCity(cname: string): void                   |
| + display(): void                                   |
| + findShortestPath(start: string, end: string):     |
|     void                                            |
| + DFS(graph: unordered_map<string, string>,         |
|     path: vector<string>,                           |
|     paths: vector<vector<string>>&,                 |
|     startcity: string): void                        |
| + BFS(start: string): void                          |
+-----------------------------------------------------+
```

```
+-----------------------------------------------------+
|                  UserLogin                          |
+-----------------------------------------------------+
| - usersDatabase: unordered_map<string, User>        |
| - currentUser: User*                                |
+-----------------------------------------------------+
| + validateMobileNumber(mobileNumber: string):       |
|     bool                                            |
| + login(phone: string, password: string): bool      |
| + signUp(firstName: string, lastName: string,       |
|     phone: string, password: string): void          |
| + userExists(phone: string) const: bool             |
| + getCurrentUser() const: const User&               |
| + isLoggedIn() const: bool                          |
| + logout(): void                                    |
| + isValidPhone(phone: string) const: bool           |
| + isValidPassword(password: string) const: bool     |
| + hashPassword(password: string) const: string      |
| + getValidPhoneFromUser() const: string             |
+-----------------------------------------------------+
```