Methodology

**1. Requirement Analysis**

* Identifying the geographic region to be covered.
* Defining the scope and different modes of transportation to be covered (walking, driving, cycling, public transport)
* Determining real-time traffic data.

**2. Algorithm Selection**

* **Graph Representation**: Represent the map as a graph where nodes are locations and edges are paths/roads.
* **Working on various Shortest Path Algorithms like Dijkstra's Algorithm, Floyd-Warshall etc.**
* **Data Structures**: Use priority queues, adjacency lists, or matrices depending on the algorithm and graph size.

**3. API Integration**

* Integrate mapping APIs to fetch real-time data and calculate distances.
* Ensure your API keys are secured and handle errors properly.

**4. Implementation**

* Start with backend development: Set up the server, database, and integrate APIs.
* Develop the frontend: Create the interface and ensure it communicates correctly with the backend.
* Implement the shortest path algorithm to process user input and return the optimal route.

**5. Testing**

* Test individual components of your app.

Data structures

**Data Structures Used**:

* **Graph**: **Graphs** are fundamental for representing the map where locations are **nodes** (or vertices) and paths/roads between them are **edges.**
* **Priority Queue**: Used primarily in **Dijkstra's** algorithms to efficiently select the next node with the smallest tentative distance.
* **HashMap**: Used for quickly accessing nodes and their associated data.
* **Arrays and List:** for storing sequences of data, such as distances, predecessors, or visited nodes.
* **Set**: Used to keep track of visited nodes efficiently.

Algorithms

1. **Dijkstra's algorithm**

One of the most popular and straightforward algorithms for finding the shortest path between two nodes in a graph, particularly when all edge weights are non-negative.

**Advantages:**

* Simple to implement and understand.
* Efficient for small to medium-sized graphs.

1. **Breadth-First Search (BFS)**

**Breadth-First Search (BFS)** is the simplest algorithm for finding the shortest path in an unweighted graph. BFS explores all nodes at the present depth before moving on to nodes at the next depth level.

**Advantages:**

* Very simple to implement.
* Guarantees the shortest path in unweighted graphs.

1. **Bellman-Ford Algorithm**

**Bellman-Ford** is similar to Dijkstra's but can handle graphs with negative weights, making it a versatile option.

**Advantages**

* Handles negative weights.
* Can be used to find the shortest path from a single source to all other nodes.