System Requirements Specification Index

For

Urban Traffic Analysis Platform System

Version 1.0

IIHT Pvt. Ltd.
fullstack@iiht.com

TABLE OF CONTENTS

1 Project Abstract

The Municipal Transport Authority requires a traffic analysis platform to monitor and optimize traffic flow across a metropolitan area. The system will track intersection congestion, analyze traffic patterns by time of day, monitor incident reports, and generate recommendations for traffic light timing. City planners and traffic engineers will use the system to identify congestion hotspots, analyze accident-prone areas, and simulate the impact of road modifications before implementation.

2 Business Requirements:

Screen Name	Console input screen
Problem Statement	1. System needs to store and analyze different types of traffic data (intersection ID, location, traffic volume, congestion level, incidents, peak hours) 2. System must support filtering intersections by congestion level, traffic volume, time of day, or incident frequency 3. Console should handle different dictionary operations like Basic filtering (intersections by congestion level, incident frequency), Dictionary comprehension (for traffic volume ranges, peak hour patterns), Dictionary methods (update, get, items), Dictionary merging and transformation, Dictionary-based data analysis

3 Constraints

3.1 Input Requirements

1. Intersection Records:

- Must be stored as dictionaries with fields for id, name, coordinates, traffic_volume, congestion_level, peak_hours, nearby_landmarks, incident_history
- Must be stored in a dictionary variable with intersection ID as key
- Example: `"I001": {"name": "Main & Broadway", "coordinates": (40.7128, -74.0060), "traffic_volume": 1200, "congestion_level": "High", "peak_hours": ["07:00-09:00", "16:00-18:00"], "nearby_landmarks": ["Central Station", "City Hall", "Shopping Mall"], "incident_history": ["Accident", "Signal Failure"]}`

2. Congestion Level:
   - Must be one of: "Low", "Moderate", "High", "Severe", "Critical"
   - Must be stored as string in intersection's "congestion_level" field
   - Example: "High"
3. Traffic Volume:
   - Must be stored as integer in "traffic_volume" field
   - Must be greater than or equal to 0
   - Example: 1200 (vehicles per hour)
4. Predefined Intersections:
   - Must use these exact predefined intersections in the initial intersections dictionary:
     - `"I001": {"name": "Main & Broadway", "coordinates": (40.7128, -74.0060), "traffic_volume": 1200, "congestion_level": "High", "peak_hours": ["07:00-09:00", "16:00-18:00"], "nearby_landmarks": ["Central Station", "City Hall", "Shopping Mall"], "incident_history": ["Accident", "Signal Failure"]}`
     - `"I002": {"name": "Park & 5th", "coordinates": (40.7580, -73.9855), "traffic_volume": 800, "congestion_level": "Moderate", "peak_hours": ["07:30-09:30", "17:00-19:00"], "nearby_landmarks": ["Central Park", "Museum", "Office Complex"], "incident_history": ["Road Work"]}`
     - `"I003": {"name": "River & Market", "coordinates": (40.7020, -74.0160), "traffic_volume": 1500, "congestion_level": "Severe", "peak_hours": ["07:00-10:00", "15:30-19:30"], "nearby_landmarks": ["Financial District", "Ferry Terminal", "Restaurant Row"], "incident_history": ["Accident", "Flooding", "Traffic Light Outage"]}`
     - `"I004": {"name": "Highway Junction 42", "coordinates": (40.7310, -73.9980), "traffic_volume": 2000, "congestion_level": "Critical", "peak_hours": ["06:30-10:30", "15:00-20:00"], "nearby_landmarks": ["Shopping Center", "Industrial Park", "Residential Area"], "incident_history": ["Multiple Accidents", "Construction"]}`
     - `"I005": {"name": "University & College", "coordinates": (40.7480, -74.0010), "traffic_volume": 600, "congestion_level": "Low", "peak_hours": ["08:00-09:00", "14:00-16:00"], "nearby_landmarks": ["University Campus", "Student Housing", "Sports Arena"], "incident_history": []}`

5. New Intersections:

    - Must use these exact predefined items in the new intersections dictionary:

        - `"N001": {"name": "Airport Access Road", "coordinates": (40.6413, -73.7781), "traffic_volume": 1800, "congestion_level": "Severe", "peak_hours": ["05:00-07:00", "19:00-21:00"], "nearby_landmarks": ["International Terminal", "Parking Garage", "Hotel Zone"], "incident_history": ["Accident", "Road Work"]}`

        - `"N002": {"name": "Harbor & Waterfront", "coordinates": (40.7023, -74.0190), "traffic_volume": 750, "congestion_level": "Moderate", "peak_hours": ["08:00-10:00", "16:30-18:30"], "nearby_landmarks": ["Ferry Terminal", "Tourist Area", "Restaurant District"], "incident_history": ["Pedestrian Incident"]}`

3.2 Operations Constraints

1. Dictionary Creation:

- Must use proper dictionary creation syntax
- Example: `{"id": "I001", "name": "Intersection Name", ...}`

2. Dictionary Access:

- Must use proper key access methods
- Example: `intersection_data[intersection_id]` or `intersection_data.get(intersection_id)`

3. Dictionary Filtering:

- Must use dictionary comprehension
- Example: `{iid: intersection for iid, intersection in intersection_data.items() if intersection["congestion_level"] == "High"}`

4. Dictionary Merging:

- Must use dictionary unpacking
- Example: `{**existing_dict, **new_dict}`

5. Dictionary Transformation:

- Must use dictionary comprehension with modification
- Example: `{k: {**v, "needs_review": len(v["incident_history"]) > 1} for k, v in intersections.items()}`

6. Dictionary Methods:

- Must use dictionary methods (keys, values, items)
- Example: `intersection_data.items()`, `intersection_data.keys()`

7. Dictionary-based Analytics:
    - Must use dictionaries for storing and calculating statistics
    - Example: `congestion_counts = {}`
8. Error Handling:
    - Must check if keys exist before accessing
    - Example: `if intersection_id in intersection_data:`
9. Immutability:
    - Must create new dictionaries rather than modifying in place
    - Example: `updated_intersection_data = intersection_data.copy()`
10. Dictionary Comprehension:
    - Must use dictionary comprehension for filtering and transforming
    - Example: `{k: v for k, v in d.items() if condition}`

3.3 Output Constraints

1. Display Format:
    - Show intersection ID, name, coordinates, traffic volume, congestion level, peak hours, nearby landmarks, incident history
    - Format traffic volume with thousands separator
    - Format congestion level with color indicators (optional)
    - Each intersection must be displayed on a new line

2. Output Format:
    - Show "== URBAN TRAFFIC ANALYSIS PLATFORM =="
    - Show "Total Intersections: {count}"
    - Show "Congestion Levels: {levels}"
    - Show "Current Intersection Data:"
    - Show intersections with format: "{id} | {name} | Coordinates: {coordinates} | Traffic Volume: {traffic_volume} vph | Congestion: {congestion_level} | Peak Hours: {peak_hours} | Landmarks: {nearby_landmarks} | Incidents: {incident_history}"
    - Show "Filtered Results:" when displaying search results

4. Template code Structure:

1. Data Management Functions:
    - `initialize_data()` - creates the initial intersections and new intersections dictionaries

2. Dictionary Operation Functions:

- `filter_by_congestion_level(intersection_data, level)` - filters intersections by congestion level
- `filter_by_traffic_volume(intersection_data, min_volume, max_volume)` - filters intersections by traffic volume range
- `filter_by_peak_hour(intersection_data, time_period)` - filters intersections by peak hour
- `filter_by_incident_type(intersection_data, incident_type)` - filters intersections by incident type
- `find_intersections_near_landmark(intersection_data, landmark)` - finds intersections near a specific landmark
- `update_traffic_volume(intersection_data, intersection_id, new_volume)` - updates an intersection's traffic volume
- `update_congestion_level(intersection_data, intersection_id, new_level)` - updates congestion level
- `add_incident_record(intersection_data, intersection_id, incident)` - adds an incident to an intersection
- `merge_intersection_data(existing_intersections, new_intersections)` - merges two intersection dictionaries
- `calculate_congestion_distribution(intersection_data)` - counts intersections at each congestion level
- `calculate_total_traffic_volume(intersection_data)` - calculates total traffic volume across all intersections
- `find_high_incident_areas(intersection_data, threshold)` - finds intersections with incidents above threshold
- `create_volume_brackets(intersection_data)` - groups intersections into traffic volume brackets

3. Display Functions:

- `get_formatted_intersection(iid, intersection)` - formats an intersection for display
- `display_data(data, data_type)` - displays intersections or other data types

4. Program Control Functions:

- `main()` - main program function

5. EXECUTION STEPS TO FOLLOW:

1. Run the program
2. View the main menu
3. Select operations:
   - Option 1: View Intersection Data
   - Option 2: Filter Intersections
   - Option 3: Update Intersection Data
   - Option 4: Add New Intersections

   - Option 5: View Traffic Statistics
   - Option 0: Exit

4. Perform operations on the intersection data
5. View results after each operation
6. Exit program when finished