3) Traffic Flow Analysis and Optimization System

REST API | Validation | Authentication | Mongo Db

A city transport department needs to analyze traffic flow to improve road usage, reduce congestion, and optimize traffic light timings. They need an API to manage and analyze traffic data from various intersections and roads in the city. This data will include vehicle counts, traffic signal states, and times of day to perform meaningful analysis. Your task is to design an API system in Node.js that handles traffic survey data, performs analysis, and provides insights.

You are required to create the following API endpoints for the Traffic Flow Analysis and Optimization System:

1. Collect traffic survey data at various intersections, including vehicle counts, light states, and timestamp information.
2. Analyze traffic flow based on the number of vehicles at different times of the day.
3. Identify congested roads or intersections based on traffic data.
4. Generate reports on traffic patterns for different times, days, and locations.
5. Provide predictions on peak traffic hours based on historical data.

For each of the APIs, you must handle path variables, query parameters, and HTTP headers effectively. You will also need to implement various algorithms like searching, sorting, and relational comparisons.

Steps to Solve

Set up the Project:

- Initialize a Node.js project using `npm init`.
- Install required dependencies: Express, MongoDB (Mongoose), Body-Parser, Moment (for time management), etc.
- Set up the server using Express.

Design the Data Model:

- Create a traffic survey data schema with attributes:
  - `intersectionId` (String)
  - `vehicleCount` (Number)
  - `lightState` (Enum: 'green', 'yellow', 'red')
  - `timestamp` (Date)
  - `roadName` (String)

Create the APIs:

- **POST** `/api/traffic` - Add traffic survey data (Vehicle count, light state, etc.)
- **GET** `/api/traffic/{intersectionId}` - Get all traffic data for a specific intersection.
- **GET** `/api/traffic/summary` - Get a summary of traffic data (filtered by time, day, or road).
- **GET** `/api/traffic/peak-hours` - Identify peak traffic hours based on historical data.
- **PUT** `/api/traffic/{id}` - Update traffic survey data.
- **DELETE** `/api/traffic/{id}` - Delete a specific traffic survey entry.

Implement Traffic Analysis Algorithms:

- Sort the traffic data based on vehicle counts to determine congestion levels.
- Search for intersections with the highest traffic flow during specific times.
- Generate reports summarizing average traffic counts by day, time, and intersection.
- Identify patterns in traffic light states affecting congestion.

Sample Data: orders

```
[
 {
  "intersectionId": "I-101",
  "vehicleCount": 350,
  "lightState": "green",
  "timestamp": "2024-11-25T08:00:00Z",
  "roadName": "Main St."
 },
 {
  "intersectionId": "I-102",
  "vehicleCount": 200,
  "lightState": "red",
  "timestamp": "2024-11-25T08:05:00Z",
  "roadName": "Second Ave."
 },
 {
  "intersectionId": "I-103",
  "vehicleCount": 500,
  "lightState": "yellow",
  "timestamp": "2024-11-25T08:15:00Z",
  "roadName": "Broadway"
 }
]
```

Sample Input for Each API

POST /api/traffic
Request Body:

```
{
 "intersectionId": "I-104",
 "vehicleCount": 600,
 "lightState": "green",
 "timestamp": "2024-11-25T08:30:00Z",
 "roadName": "5th Ave."
}
```

GET /api/traffic/{intersectionId}
Example Request:
GET /api/traffic/I-101
Response:

```
[
 {
  "intersectionId": "I-101",
  "vehicleCount": 350,
  "lightState": "green",
    "timestamp": "2024-11-25T08:00:00Z",
    "roadName": "Main St."
  }
]
```

GET /api/traffic/summary
Example Request:
GET /api/traffic/summary?roadName=Main St.
Response:

```
{
  "roadName": "Main St.",
  "totalVehicleCount": 350,
  "peakTrafficTime": "08:00 AM",
  "lightState": "green"
}
```

GET /api/traffic/peak-hours
Example Request:
GET /api/traffic/peak-hours?date=2024-11-25
Response:

```
{
  "peakHours": [
    {"hour": "08:00", "vehicleCount": 350},
    {"hour": "08:30", "vehicleCount": 600}
  ]
}
```

Sample Output for Each API:

GET /api/traffic/{intersectionId}
For an intersection ID I-101, the system returns the traffic survey records for that intersection.
Sample Output:

```
{
  "intersectionId": "I-101",
  "vehicleCount": 350,
  "lightState": "green",
  "timestamp": "2024-11-25T08:00:00Z",
  "roadName": "Main St."
}
```

GET /api/traffic/summary
For a road Main St., the system will calculate the total vehicle count and the peak traffic times.
Sample Output:

```
{
  "roadName": "Main St.",
  "totalVehicleCount": 350,
  "peakTrafficTime": "08:00 AM",
  "lightState": "green"
}
```

GET /api/traffic/peak-hours
For the date 2024-11-25, the system identifies the peak traffic hours.
Sample Output:

```
{
  "peakHours": [
    {"hour": "08:00", "vehicleCount": 350},
    {"hour": "08:30", "vehicleCount": 600}
  ]
}
```

7) API Implementation Use Cases:

- Searching: Use query parameters to search for traffic data based on specific intersections, road names, or time ranges.
- Sorting: Sort traffic data by vehicle count to determine the busiest intersections or times.
- Relational Comparison: Compare traffic flows at different intersections during the same time to identify congestion patterns.

Example sorting of traffic data by vehicle count:

```
// Sorting traffic survey data by vehicleCount

const sortedTrafficData = trafficData.sort((a, b) => b.vehicleCount - a.vehicleCount);
```

Example comparison to identify congestion:

```
if (trafficData1.vehicleCount > trafficData2.vehicleCount) {
  console.log(`Intersection ${trafficData1.intersectionId} has more traffic than ${trafficData2.intersectionId}`);
}
```