**Tech Stack I used:**

**Node.js and Express.js:**

**Express.js: Simplifies the creation of RESTful API endpoints with its straightforward routing system.**

**Node.js: Efficiently manages concurrent API requests due to its non-blocking, asynchronous I/O model. This is crucial for handling high volumes of requests, such as those to the NHTSA API and interactions with the database.**

1. Asynchronous I/O: Efficiently handles multiple concurrent requests.

2. Performance: Fast response times with non-blocking operations.

3. Scalability: Easy to scale horizontally across multiple instances.

4. Express.js: Simplifies routing and middleware management.

5. JSON Handling: Native support for JSON data, ideal for API responses.

6. Middleware: Facilitates adding features like authentication and rate limiting.

8. Integration: Easy to integrate with third-party APIs and services.

**MONGODB**

1. Schema Flexibility: Handles varying data structures easily.

2. Scalability: Designed for horizontal scaling and large volumes of data.

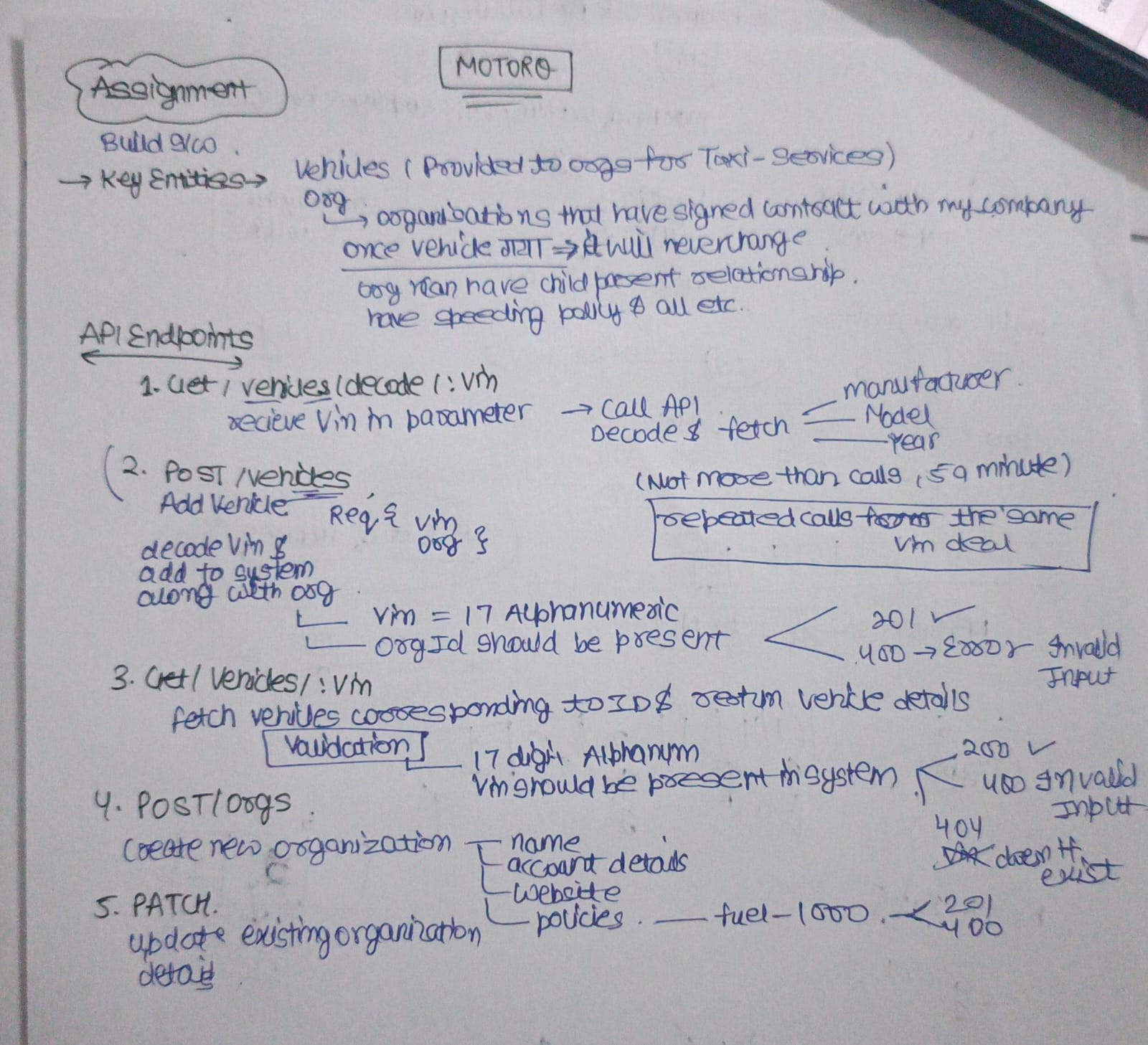
3. Document-Oriented: Stores data in JSON-like documents, aligning with API data formats.

4. Performance: Efficient for read and write operations with its in-memory processing.

5. Easy Integration: Works well with Node.js through libraries like Mongoose.

6. Aggregation Framework: Powerful querying and data processing capabilities.

**ANALYSIS**



**Functionality of Each File**

**1. app.js**

* **Purpose**: Sets up and configures the Express server.
* **Functionality**:
  + Loads environment variables from .env.
  + Configures middleware (express.json() to parse JSON request bodies).
  + Sets up routing for vehicles and organizations.
  + Starts the server on a specified port.

**2. .env**

* **Purpose**: Stores environment variables for configuration.
* **Functionality**:
  + PORT: Port number on which the server runs.
  + MONGO\_URI: MongoDB connection string.
  + NHTSA\_API\_URL: URL for the NHTSA API.
  + NHTSA\_API\_RATE\_LIMIT: Rate limit for NHTSA API calls.

**3. routes/orgRoutes.js**

* **Purpose**: Defines routes related to organizations.
* **Functionality**:
  + **POST /**: Maps to createOrg to create a new organization.
  + **PATCH /:id**: Maps to updateOrg to update an existing organization.
  + **GET /**: Maps to getAllOrgs to retrieve all organizations.

**4. routes/vehicleRoutes.js**

* **Purpose**: Defines routes related to vehicles.
* **Functionality**:
  + **GET /decode/:vin**: Maps to decodeVin to decode a VIN using the NHTSA API.
  + **POST /**: Maps to addVehicle to add a new vehicle to the system.
  + **GET /:vin**: Maps to getVehicleByVin to fetch vehicle details by VIN.

**5. controllers/orgController.js**

* **Purpose**: Contains business logic for handling organization-related requests.
* **Functionality**:
  + **createOrg(req, res)**: Creates a new organization and saves it to the database.
  + **updateOrg(req, res)**: Updates an existing organization's details.
  + **getAllOrgs(req, res)**: Retrieves all organizations from the database.

**6. controllers/vehicleController.js**

* **Purpose**: Contains business logic for handling vehicle-related requests.
* **Functionality**:
  + **decodeVin(req, res)**: Decodes a VIN using the NHTSA API, with rate limiting and caching.
  + **addVehicle(req, res)**: Adds a new vehicle to the system, validates the VIN, and associates it with an organization.
  + **getVehicleByVin(req, res)**: Retrieves vehicle details by VIN from the database.

**7. models/Org.js**

* **Purpose**: Defines the schema for the Organization entity in MongoDB.
* **Functionality**:
  + Defines the fields for an organization (e.g., name, account, website, fuelReimbursementPolicy, speedLimitPolicy).
  + Provides methods for interacting with the organizations collection in MongoDB.

**8. models/Vehicle.js**

* **Purpose**: Defines the schema for the Vehicle entity in MongoDB.
* **Functionality**:
  + Defines the fields for a vehicle (e.g., vin, org, manufacturer, model, year).
  + Provides methods for interacting with the vehicles collection in MongoDB.

These files and their functionalities work together to build a complete backend system for managing vehicles and organizations for the More Torque taxi service company.