**1. app.js**

This file sets up the Express server, configures middleware, and connects your routes.

javascript

Copy code

const express = require('express');

const mongoose = require('mongoose');

const dotenv = require('dotenv');

const vehicleRoutes = require('./routes/vehicleRoutes');

const orgRoutes = require('./routes/orgRoutes');

// Load environment variables

dotenv.config();

// Initialize Express app

const app = express();

// Middleware to parse JSON bodies

app.use(express.json());

// Connect to MongoDB

mongoose.connect(process.env.MONGO\_URI, {

useNewUrlParser: true,

useUnifiedTopology: true,

})

.then(() => console.log('MongoDB connected successfully'))

.catch((err) => console.error('MongoDB connection error:', err));

// Mount routes

app.use('/vehicles', vehicleRoutes);

app.use('/orgs', orgRoutes);

// Start the server

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => {

console.log(`Server is running on port ${PORT}`);

});

**2. .env**

This file holds your environment variables like the MongoDB connection string and the port number.

makefile

Copy code

PORT=5000

MONGO\_URI=mongodb://localhost:27017/more-torque

NHTSA\_API\_URL=https://vpic.nhtsa.dot.gov/api/

NHTSA\_API\_RATE\_LIMIT=5

**Explanation:**

* **app.js**:
  + **Express Setup**: Sets up the Express server and middleware.
  + **MongoDB Connection**: Uses mongoose to connect to your MongoDB instance using a connection string stored in the .env file.
  + **Routes**: Links the route files (vehicleRoutes.js and orgRoutes.js) to specific paths.
  + **Server Start**: Starts the server on a port defined in the .env file or defaults to port 5000.
* **.env**:
  + **PORT**: The port on which your Express server will run.
  + **MONGO\_URI**: The MongoDB connection string. Replace this with your actual MongoDB URI if different.
  + **NHTSA\_API\_URL**: Base URL for the NHTSA API.
  + **NHTSA\_API\_RATE\_LIMIT**: Sets the rate limit for API calls to NHTSA to 5 per minute.

With this setup, you're laying the groundwork for your project, ensuring that your server can connect to MongoDB, handle routes, and enforce API rate limits.

**Next Step: Setting Up Routes and Controllers**

Now that you have your app.js and .env files set up, the next step is to create the route and controller files. Let’s start with the vehicleRoutes.js and vehicleController.js files.

**1. routes/vehicleRoutes.js**

This file will define the routes for vehicle-related operations and map them to the corresponding controller functions.

javascript

Copy code

const express = require('express');

const router = express.Router();

const vehicleController = require('../controllers/vehicleController');

// Route to decode VIN and get vehicle details

router.get('/decode/:vin', vehicleController.decodeVin);

// Route to add a new vehicle

router.post('/', vehicleController.addVehicle);

// Route to fetch a vehicle by VIN

router.get('/:vin', vehicleController.getVehicleByVin);

module.exports = router;

**2. controllers/vehicleController.js**

This file will handle the business logic for vehicle-related operations.

javascript

Copy code

const axios = require('axios');

const Vehicle = require('../models/Vehicle'); // Assuming you have a Vehicle model defined

const Org = require('../models/Org'); // Assuming you have an Org model defined

// Cache to store decoded VINs and limit API calls

let vinCache = {};

let apiCallCount = 0;

const rateLimit = process.env.NHTSA\_API\_RATE\_LIMIT || 5;

let lastApiCallTime = Date.now();

async function decodeVin(req, res) {

const vin = req.params.vin;

// Simple validation

if (!/^[A-HJ-NPR-Z0-9]{17}$/.test(vin)) {

return res.status(400).json({ error: 'Invalid VIN format' });

}

// Check cache first

if (vinCache[vin]) {

return res.json(vinCache[vin]);

}

// Rate limiting logic

const currentTime = Date.now();

if (currentTime - lastApiCallTime > 60000) {

apiCallCount = 0;

lastApiCallTime = currentTime;

}

if (apiCallCount >= rateLimit) {

return res.status(429).json({ error: 'Rate limit exceeded. Please try again later.' });

}

// Call NHTSA API

try {

const response = await axios.get(`${process.env.NHTSA\_API\_URL}vehicles/DecodeVin/${vin}?format=json`);

const vehicleData = {

manufacturer: response.data.Results[7].Value,

model: response.data.Results[9].Value,

year: response.data.Results[10].Value

};

// Cache the result

vinCache[vin] = vehicleData;

apiCallCount++;

res.json(vehicleData);

} catch (error) {

res.status(500).json({ error: 'Error decoding VIN' });

}

}

async function addVehicle(req, res) {

const { vin, org } = req.body;

// Validate VIN and Org

if (!/^[A-HJ-NPR-Z0-9]{17}$/.test(vin)) {

return res.status(400).json({ error: 'Invalid VIN format' });

}

const organization = await Org.findById(org);

if (!organization) {

return res.status(400).json({ error: 'Invalid organization ID' });

}

// Decode VIN

const vehicleData = vinCache[vin] || await decodeVin({ params: { vin } }, res);

if (!vehicleData.manufacturer) {

return; // If decodeVin sends a response, stop execution here

}

// Create and save vehicle

const vehicle = new Vehicle({

vin,

org,

manufacturer: vehicleData.manufacturer,

model: vehicleData.model,

year: vehicleData.year

});

try {

const savedVehicle = await vehicle.save();

res.status(201).json(savedVehicle);

} catch (error) {

res.status(500).json({ error: 'Error saving vehicle' });

}

}

async function getVehicleByVin(req, res) {

const vin = req.params.vin;

// Simple validation

if (!/^[A-HJ-NPR-Z0-9]{17}$/.test(vin)) {

return res.status(400).json({ error: 'Invalid VIN format' });

}

try {

const vehicle = await Vehicle.findOne({ vin });

if (!vehicle) {

return res.status(404).json({ error: 'Vehicle not found' });

}

res.json(vehicle);

} catch (error) {

res.status(500).json({ error: 'Error fetching vehicle' });

}

}

module.exports = {

decodeVin,

addVehicle,

getVehicleByVin

};

**Explanation:**

1. **vehicleRoutes.js**:
   * **router.get('/decode/:vin')**: Maps the /vehicles/decode/:vin endpoint to the decodeVin controller function.
   * **router.post('/')**: Maps the /vehicles endpoint to the addVehicle controller function.
   * **router.get('/:vin')**: Maps the /vehicles/:vin endpoint to the getVehicleByVin controller function.
2. **vehicleController.js**:
   * **decodeVin(req, res)**: Handles the decoding of a VIN by calling the NHTSA API and caching the result to minimize API calls.
   * **addVehicle(req, res)**: Decodes the VIN, validates the organization, and adds the vehicle to the system.
   * **getVehicleByVin(req, res)**: Fetches a vehicle by its VIN from the database.

**Next Steps:**

1. **Create Models:**
   * Implement MongoDB models for Vehicle and Org in the models/ directory.
2. **Org Routes and Controller:**
   * Set up routes and controllers for handling organization-related operations.

Once these are set up, you can start testing the endpoints using tools like Postman to ensure everything is working correctly before moving on to the database implementation.

**Create routes/orgRoutes.js**

This file will map organization-related routes to the corresponding controller functions.

javascript

Copy code

// routes/orgRoutes.js

const express = require('express');

const router = express.Router();

const orgController = require('../controllers/orgController');

// Route to create a new organization

router.post('/', orgController.createOrg);

// Route to update an organization's details

router.patch('/:id', orgController.updateOrg);

// Route to fetch all organizations

router.get('/', orgController.getAllOrgs);

module.exports = router;

**3. Update orgController.js**

Now, let's implement the business logic for the organization-related operations.

javascript

Copy code

// controllers/orgController.js

const Org = require('../models/Org');

// Create a new organization

async function createOrg(req, res) {

const { name, account, website, fuelReimbursementPolicy = 1000, speedLimitPolicy } = req.body;

try {

const newOrg = new Org({

name,

account,

website,

fuelReimbursementPolicy,

speedLimitPolicy

});

const savedOrg = await newOrg.save();

res.status(201).json(savedOrg);

} catch (error) {

res.status(400).json({ error: 'Error creating organization' });

}

}

// Update an organization's details

async function updateOrg(req, res) {

const { id } = req.params;

const updateData = req.body;

try {

const org = await Org.findById(id);

if (!org) {

return res.status(404).json({ error: 'Organization not found' });

}

Object.keys(updateData).forEach((key) => {

org[key] = updateData[key];

});

const updatedOrg = await org.save();

res.status(200).json(updatedOrg);

} catch (error) {

res.status(400).json({ error: 'Error updating organization' });

}

}

// Get all organizations

async function getAllOrgs(req, res) {

try {

const orgs = await Org.find();

res.status(200).json(orgs);

} catch (error) {

res.status(400).json({ error: 'Error fetching organizations' });

}

}

module.exports = {

createOrg,

updateOrg,

getAllOrgs

};

**4. Update app.js to include the organization routes**

You already have this line to include vehicle routes:

javascript

Copy code

app.use('/orgs', orgRoutes);

MODELS

**To Make It Fully Functional:**

1. **Create the models/Vehicle.js and models/Org.js files:**

javascript

Copy code

// models/Vehicle.js

const mongoose = require('mongoose');

const vehicleSchema = new mongoose.Schema({

vin: { type: String, required: true, unique: true },

org: { type: mongoose.Schema.Types.ObjectId, ref: 'Org', required: true },

manufacturer: String,

model: String,

year: String,

});

module.exports = mongoose.model('Vehicle', vehicleSchema);

javascript

Copy code

// models/Org.js

const mongoose = require('mongoose');

const orgSchema = new mongoose.Schema({

name: { type: String, required: true },

account: String,

website: String,

fuelReimbursementPolicy: { type: String, default: '1000' },

speedLimitPolicy: String,

parent: { type: mongoose.Schema.Types.ObjectId, ref: 'Org' },

});

module.exports = mongoose.model('Org', orgSchema);

1. **Install Required Packages:**

Ensure you have the necessary npm packages installed:

bash

Copy code

npm install express mongoose axios dotenv

1. **Run and Test:**

Run your server using:

bash

Copy code

node app.js

Use tools like Postman to test your API endpoints and ensure everything works as expected.