UNIVERSITY OF MINES AND TECHNOLOGY (UMAT)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



A DOCUMENTATION ON AN VEHICLE PARTS MANAGEMENT SYSTEM

GROUP 2

APRIL, 2024

This is a RESTful API built with Node.js, Express, and MongoDB for managing a vehicle part system. The API provides endpoints for handling Accessorise, Suspension parts, Car Parts , Exhaust Part, Motor Parts and User Authentication.

## Environment Setup

To run this application locally, you'll need to have Node.js and MongoDB installed on your machine. Follow these steps:

1. Clone the repository
2. Create a .env file in the root directory and add the following environment variables:

* PORT: The port number for the server (e.g., 3000)
* MONGODB\_URI: The MongoDB connection URI (e.g., mongodb://localhost:27017/vehicleParts\_ms)
* TEST\_MONGODB\_URI: The MongoDB connection URI for running tests (e.g., mongodb://localhost:27017/vehicleParts\_ms-test)
* MONGODB ATLAS\_URI: The MongoDB Atlas connection URI (e.g., mongodb+srv://phavour:phavour123@cluster0.jhkqkgk.mongodb.net/VehiclePart\_MS?retryWrites=true&w=majority&appName=Cluster0')

1. Install dependencies by running npm install
2. Start the server with npm start/nodemon app

## Project Structure

The project follows a modular structure with separate directories for models, controllers, routes, and utilities.

* app.js: The entry point of the application, where the Express app is configured and connected to MongoDB.
* index.js: The file that starts the server and listens on the specified port.
* models/: Directory containing Mongodb models for different entities.
* controllers/: Directory containing Express router modules for handling different routes.
* utils/: Directory containing utility files for configuration, logging, and middleware functions.

## API Endpoints

### **Accessories**

* GET /api/Accessories: Get all accessories.
* GET /api/ Accessories /:id: Get a single accessories by ID.
* POST /api/ Accessories /: Create a new accessories (admin only).
* DELETE /api/ Accessories /:id: Delete an accessories and their associated accessories.
* Patch /api/Accessories/:id: Update a accessories Parts.

### **Body\_Parts**

* GET /api/Body\_Parts/: Get all Body Parts with filtering, pagination, and sorting options.
* GET /api/Body\_Parts/:id: Get a single Body Parts by ID.
* POST /api/Body\_Parts/: Create a new Body Parts.
* DELETE /api/Body\_Parts/:id: Delete a Body Parts.
* Patch /api/Body\_Parts/:id: Update a Body Parts.

### **Car Parts**

* POST /api/Car\_Parts/: Create a new car part.
* GET /api/Car\_Parts/:Car\_Parts­Id: Get a car part by ID.
* DELETE /api/Car\_Parts/:Car\_PartsId: Delete a car part and its associated cart items.
* GET /api/Car\_Parts/:Car\_Parts: Get all car part items for a cart.
* GET /api/ Car\_Parts/: Car\_PartsId: Get a specific car part item.
* PATCH /api/ Car\_Parts/: Car\_PartsId/: Update the quantity of a car part item.

### **Exhaust Part**

* POST /api/Exhaust\_Parts/: Create a new exhaust part.
* GET /api/ Exhaust\_Parts/: Exhaust\_Parts­Id: Get a exhaust part by ID.
* DELETE /api/ Exhaust\_Parts/: Delete a exhaust part and its associated exhaust part.
* GET /api/ Exhaust\_Parts/Exhaust\_PartsId/: Get all exhaust part items for a exhaust part.
* POST /api/ Exhaust\_Parts/Exhaust\_Parts/: Add items to a exhaust part.
* GET /api/Exhaust\_PartsId/items/: Get a specific exhaust part item.
* PATCH/api/ Exhaust\_Parts/Exhaust\_PartsId/: Update the quantity of a exhaust part item.
* DELETE /api/ Exhaust\_Parts/Exhaust\_PartsId/: Delete a exhaust part item.

### **Suspension Part**

* POST /api/Suspension\_Parts/: Create a new Suspension part.
* GET /api/ Suspension\_Parts/Suspension\_Parts­Id: Get a Suspension part by ID.
* DELETE /api/ Suspension\_Parts/Suspension\_PartsId: Delete a Suspension part and its associated Suspension items.
* GET/api/ Suspension\_Parts/: Get all Suspension part items for the database.
* POST/api/ Suspension\_Parts/Suspension\_Parts/: Add items to a Suspension part.
* GET/api/ Suspension\_PartsId/: Get a specific Suspension part item.
* PATCH/api/ Suspension\_PartsId/: Update the quantity of a Suspension part item.
* DELETE/api/ Suspension\_Parts/Suspension\_PartsItemId: Delete a Suspension part item.

### **Motor Parts**

* GET/api/Motor\_Parts/: Get all Motor Parts collection.
* GET/api/ Motor\_Parts/Motor\_PartsId: Get a specific Motor Parts by ID.
* POST/api/ Motor\_Parts /: Add a new Motor Parts to the motor part collection.
* PATCH/api/ Motor\_PartsId/: Update the quantity of a Suspension part item.
* DELETE /api/ Motor\_Parts/Motor\_PartsId: Delete a Motor part item.

### **Users**

* GET/api/users/: Get all users or the authenticated user's data.
* POST/api/users/: Create a new user.
* PATCH/api/users/me: Update the authenticated user's data.
* DELETE /api/users/me: Delete the authenticated user's account.

## Utilities

### **Config**

The config.js file is responsible for handling environment variables. It exports the PORT and MONGODB\_URI variables, with the latter being set based on the NODE\_ENV (either using the TEST\_MONGODB\_URI or the regular MONGODB\_URI).

### **Logger**

The logger.js file provides a simple logging utility with two functions:

* info(...params): Logs the provided parameters to the console using console.log.
* error(...params): Logs the provided parameters to the console using console.error.

### **Middleware**

The app.js file contains various middleware functions used in the application:

* extractDuplicatedKey(errorMessage): A helper function that extracts the duplicated key from a MongoDB error message related to unique constraint violations.
* validateUser(request, response, next): Validates the user input for creating a new user by checking if the username, password, and email fields are present, if the username and password are at least 5 characters long, and if the email format is valid.
* requestLogger(request, response, next): Logs the request method, path, and body to the console.
* unknownEndpoint(request, response): Handles unknown endpoints by responding with a 404 Not Found error.
* errorHandler(error, request, response, next): Handles various types of errors, including CastError, ValidationError, MongoServerError (including duplicate key errors), JsonWebTokenError, and TokenExpiredError, and responds with an appropriate error message and status code.

## **Error Handling**

The application uses the errorHandler middleware to handle various types of errors. It logs the error message using the logger utility and responds with an appropriate error message and status code based on the error type. If the error type is not recognized, it passes the error to the default Express error handler.

## Group Members

1. Ebenezer Ainoo Tutu
2. Moses Tetteh
3. Araba Fenyiwah Turkson
4. Agyeiwaa Bernice Yirenkyi