## **Product Management API Documentation**

The code provided implements a basic product management system using MongoDB, Express, and Node.js. Here’s an explanation of each file and its functionality, along with how to test it.

### **1. App/add.js**

This file is used to add a set of products to the database. It connects to the MongoDB database, drops the existing products collection (using drop.js), and then inserts a new set of products.

* **How it works:**
  + Connects to MongoDB using connectToDB.
  + Drops the products collection (calls dropProductsCollection).
  + Inserts a list of product objects into the products collection.
* **How to test:**
  + Run the file using the command node add.js. This will drop the products collection and insert the provided set of products into MongoDB.

### **2. App/drop.js**

This file handles dropping the products collection from the MongoDB database. If the collection doesn't exist, it handles the NamespaceNotFound error gracefully.

* **How it works:**
  + Connects to the database.
  + Attempts to drop the products collection.
  + Logs either success or a message indicating the collection does not exist.
* **How to test:**
  + You can import and run this function in isolation or include it in add.js, as demonstrated, to see the collection being dropped before adding new products.

### **3. App/read.js**

This file is responsible for reading (retrieving) all products from the products collection in MongoDB.

* **How it works:**
  + Connects to the MongoDB database.
  + Queries all documents from the products collection.
  + Logs the retrieved products to the console.
* **How to test:**
  + Run node read.js to view all products in the collection.

### **4. App/remove.js**

This file deletes a product with id: 1 from the MongoDB collection.

* **How it works:**
  + Connects to the database.
  + Deletes a product based on its id (in this case, id: 1).
  + Logs a success message.
* **How to test:**
  + Run node remove.js. Afterward, use the read.js file to verify that the product with id: 1 has been removed.

### **5. App/update.js**

This file updates the product with id: 1, changing its price to 11.99 and units to 90.

* **How it works:**
  + Connects to the database.
  + Finds the product with id: 1 and updates its price and units.
  + Logs a success message.
* **How to test:**
  + Run node update.js to update the product. Use read.js to confirm the product's new values.

### **6. server/controllers/productController.js**

This file contains the logic for handling product-related API routes. It provides functionalities such as fetching all products, getting a product by ID, adding a product, updating a product, and deleting a product.

* **Functions:**
  + **getProducts**: Retrieves all products from the database.
  + **getProductById**: Fetches a product by its MongoDB \_id.
  + **addProduct**: Adds a new product if the id is unique.
  + **updateProduct**: Updates an existing product based on its MongoDB \_id.
  + **deleteProduct**: Deletes a product based on its MongoDB \_id.
* **How to test:**
  + These can be tested using API testing tools like Postman or cURL. You can make GET, POST, PUT, and DELETE requests to the respective API routes (discussed below).

### **7. models/product.js**

This file defines the MongoDB schema for the Product model using Mongoose. It includes fields for name, description, price, units, and id.

* **How it works:**
  + Defines a Mongoose schema and model for products, specifying field types and optional validation (e.g., maxlength for name and description).
* **How to test:**
  + This schema is used when adding or updating products via the API (e.g., using the addProduct function in productController).

### **8. routes/productRoutes.js**

This file defines the API endpoints for product management. It maps URL paths to the corresponding controller functions.

* **Endpoints:**
  + GET /api/products: Fetch all products.
  + GET /api/products/:id: Fetch a single product by its MongoDB \_id.
  + POST /api/products: Add a new product.
  + PUT /api/products/:id: Update an existing product by its \_id.
  + DELETE /api/products/:id: Delete a product by its \_id.
* **How to test:**
  + These routes can be tested using Postman or cURL, sending requests to the local server (e.g., http://localhost:3000/api/products).

### **9. server/server.js**

This is the main entry point of the application. It sets up the Express server, connects to MongoDB, and applies middleware like CORS and bodyParser. It also loads the product routes.

* **How it works:**
  + Uses connectToDB to establish a connection with MongoDB.
  + Attaches the database connection to each request (req.db).
  + Loads product-related routes from routes/productRoutes.js.
  + Starts the Express server on port 3000.
* **How to test:**
  + Run node server.js to start the server. You can then use Postman or cURL to make requests to the API routes (/api/products).

### **Testing the API Using Postman**

1. **Start the server**: Run node server.js.
2. **Testing Endpoints**:
   * **GET /api/products**: Fetch all products.
     + Method: GET
     + URL: http://localhost:3000/api/products
     + Expected response: An array of all products in the database.
   * **GET /api/products/**: Fetch a product by its ID.
     + Method: GET
     + URL: http://localhost:3000/api/products/{PRODUCT\_ID}
     + Expected response: The product with the given ID.
   * **POST /api/products**: Add a new product.
     + Method: POST
     + URL: http://localhost:3000/api/products

Body (JSON):  
{

"id": 4,

"name": "Product 4",

"description": "Desc 4",

"price": 39.99,

"units": 25

}

* + - Expected response: The newly added product.
  + **PUT /api/products/**: Update an existing product.
    - Method: PUT
    - URL: http://localhost:3000/api/products/{PRODUCT\_ID}

Body (JSON):  
  
{

"price": 49.99,

"units": 20,  
 "name": "Product 4",

"description": "Desc 4"  
}

* + - Expected response: The updated product.
  + **DELETE /api/products/**: Delete a product by its ID.
    - Method: DELETE
    - URL: http://localhost:3000/api/products/{PRODUCT\_ID}
    - Expected response: Confirmation of the deleted product.