System Requirements Specification

Index

For

Product Manager

Version 1.0

TABLE OF CONTENTS

PRODUCT MANAGER

System Requirements Specification

You need to only work on the backend part. Please ignore the frontend angular part.

BACKEND-EXPRESS RESTFUL APPLICATION

1 PROJECT ABSTRACT

"**Product Manager**" is an express js application designed to provide a seamless product management APIs. It leverages the ExpressJs with MongoDB as the database. This platform aims to provide APIs on the product, allowing users to browse, search for, and purchase a wide range of products.

Following is the requirement specifications:

Product Module Functionalities	
1	Create a new product
2	Get product by id
3	Update product by id
4	Delete product by id
5	Get all products
6	Get list of all top rated products
7	Search product by name or description

2 Assumptions, Dependencies, Risks / Constraints

2.1 Product Constraints

2.2 When creating, product name and price are mandatory fields, on failing it should throw a custom exception with message as “Failed to create product.”.

2.3 When fetching a product by ID, if the product ID does not exist, the operation should throw a custom exception with message as “Product not found.”.

2.4 When updating a product, if the product ID does not exist, the operation should throw a custom exception with message as “Product not found.”.

2.5 When removing a product, if the product ID does not exist, the operation should throw a custom exception with message as “Product not found.”.

2.6 When searching a product, if the name or description does not exist, it should throw a custom exception with message as “Failed to search for products.”.

Common Constraints

- All the database operations must be implemented in serviceImpl file only.
- Do not change, add, remove any existing methods in the service file.
- In the service layer, custom methods can be added as per requirements.
- All RestEndpoint methods and Exception Handlers must return data in json format.
- Any type of authentication and authorisation must be added in routes file only.

3 Rest Endpoints

Rest End-points to be exposed in the routes file and attached with controller method along with method details for the same to be created. Please note, that these all are required to be implemented.

3.1 Product Restpoints

URL Exposed	Purpose
1. /api/products/all Http Method: GET Parameter: - Return: list of products	Fetches all the products
2. /api/products/ Http Method: POST Parameter: - Return: newly created product	Creates a new product

3. /api/products/search Http Method: GET Parameter: - Return: searched products	Search the product by name or description
4. /api/products/top/:limit Http Method: GET Parameter: limit Return: list of products	Fetches the top rated products
5. /api/products/:id Http Method: GET Parameter: id Return: fetch product	Fetches the product by id
6. /api/products/:id Http Method: PUT Parameter: id Return: updated product	Updates the product by id
7. /api/products/:id Http Method: DELETE Parameter: id Return: deleted product	Deletes the product by id

4 TEMPLATE CODE STRUCTURE

4.1 MODULES/PRODUCTS: controller

Resources

ProductController (Class)	This is the controller class for the product module.	To be implemented

4.2 MODULES/PRODUCTS: dao

Resources

File	Description	Status
models/cart model **models/product model**	Models for cart and product	Already implemented

schemas/cart schema **schemas/product schema**	Schemas for cart and product	Already implemented

4.3 MODULES/PRODUCTS: routes

Resources

File	Description	Status
Product routes	Routes for product	Already implemented.

4.4 MODULES/PRODUCTS: service

Resources

Class	Description	Status
ProductService	• Defines ProductService	Already implemented.

4.5 MODULES/PRODUCTS: service/impl

Resources

Class	Description	Status
ProductServiceImpl	• Implements ProductService.	To be implemented.

5 METHOD DESCRIPTIONS

5.1 ProductController Class - Method Descriptions:

Method	Task	Implementation Details
getAllProducts	To retrieve all products	- The request type should be GET with URL /api/products/all. - The method should call productService.getAllProducts(). - Return the products in JSON format. - On failure, respond with status 500 and error message: 'Failed to retrieve products.'
createProduct	To create a new product	- The request type should be POST with URL /api/products/. - Extract product data using req.body. - Call productService.createProduct(req.body). - Return the created product with status 201 CREATED. - On error, respond with status 500 and error message: 'Failed to create product.'
searchProduct	To search products by name or description	- The request type should be GET with URL /api/products/search. - Use req.query to extract 'name' and 'description'. - Call productService.searchProduct(name, description). - Return the found products in JSON format. - On error, respond with status 500 and error message: 'Failed to search products.'
getTopRatedProducts	To retrieve top-rated products with a limit	- The request type should be GET with URL /api/products/top/:limit. - Extract limit using req.params.limit and convert to integer. - Call productService.getTopRatedProducts(limit). - Return the products in JSON format. - On error, respond with status 500 and error message: 'Failed to retrieve top rated products.'
getProduct	To get a specific product by ID	- The request type should be GET with URL /api/products/:id. - Extract product ID using req.params.id. - Call productService.getProduct(id). - Return the product data. - On error, respond with status 404 and error

		message: 'Product not found.'
updateProduct	To update a product by ID	- The request type should be PUT with URL /api/products/:id. - Extract product ID using req.params.id and update data using req.body. - Call productService.updateProduct(id, req.body). - Return the updated product. - On error, respond with status 404 and error message: 'Product not found.'
deleteProduct	To delete a product by ID	- The request type should be DELETE with URL /api/products/:id. - Extract product ID using req.params.id. - Call productService.deleteProduct(id). - Return the deleted product data. - On error, respond with status 404 and error message: 'Product not found.'

5.2 ProductServiceImpl Class - Method Descriptions:

Method	**Task**	**Implementation Details**
getAllProducts	To fetch all products from the database	- Fetch all product records using Product.find(). - Return the list of products. - On failure, throw error: 'Failed to get all products.'
createProduct	To create and save a new product	- Accept product data as input. - Use Product.create(productData) to save it. - Return the newly created product. - On failure, throw error: 'Failed to create product.'
searchProduct	To search products by name and/or description	- Build a query object using regex for name and description. - Use Product.find(query) to get matching products. - Return the matched products. - On failure, throw error: 'Failed to search for products.'

getProduct	To fetch a single product by its ID	- Use Product.findById(productId) to get the product. - If not found, throw error: 'Product not found.' - Return the product. - On failure, throw error: 'Failed to get product.'
updateProduct	To update product details by ID	- Use Product.findByIdAndUpdate(productId, updatedProduct, { new: true }). - If not found, throw error: 'Product not found.' - Return the updated product. - On failure, throw error: 'Failed to update product.'
deleteProduct	To delete a product by ID	- Use Product.findByIdAndDelete(productId). - If not found, throw error: 'Product not found.' - Return the deleted product. - On failure, throw error: 'Failed to delete product.'
getTopRatedProducts	To get a list of top-rated products	- Use Product.find().sort({ ratings: -1 }).limit(limit). - Return the sorted list of top-rated products. - On failure, throw error: 'Failed to get top rated products.'

EXECUTION STEPS TO FOLLOW FOR BACKEND

1. **All actions like build, compile, running application, running test cases will be through Command Terminal.**
2. **To open the command terminal the test takers, need to go to**

   Application menu (Three horizontal lines at left top) -> Terminal ->New Terminal.

3. **This editor Auto Saves the code.**
4. **These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.**
5. **To test any Restful application, the last option on the left panel of IDE, you can find ThunderClient, which is the lightweight equivalent of POSTMAN.**
6. **You can follow series of command to setup express environment once you are in your**

project-name folder:

a. **npm install -> Will install all dependencies -> takes 10 to 15 min**
b. **npm run start -> To compile and run the project.**
c. **npm run jest -> to run all test cases and see the summary of all passed and failed test cases.**
d. **npm run test -> to run all test cases and register the result of all test cases. It is mandatory to run this command before submission of workspace -> takes 5 to 6 min.**