System Requirements Specification

Index

For

E-Commerce – Product Module

Version 1.0

TABLE OF CONTENTS

E-COMMERCE – Product Module

System Requirements Specification

BACKEND-EXPRESS RESTFUL APPLICATION

1 PROJECT ABSTRACT

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

Following is the requirement specifications:

	E-Commerce
Modules	
1	Product

Product Module Functionalities	
1	Get all products
2	Create a new product
3	Search the products
4	Get top rated products
5	Apply discount on cart
6	View the cart
7	Add item to the cart
8	Checkout the cart
9	Update an item in cart
10	Remove an item in cart
11	Get a product
12	Update a product
13	Delete a product

2 ASSUMPTIONS, DEPENDENCIES, RISKS / CONSTRAINTS

2.1 PRODUCT CONSTRAINTS

1. 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. 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.”.
3. When updating a product, if the product ID does not exist, the operation should throw a custom exception with message as “Product not found.”.
4. When removing a product, if the product ID does not exist, the operation should throw a custom exception with message as “Product not found.”.
5. 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.”.
6. When applying some discount on a product, if the discount percentage is invalid (0 > discount or discount > 100), it should throw a custom exception with message as “Invalid discount percentage.”.
7. When applying some discount on a product, if the discount percentage does not exist, it should throw a custom exception with message as “Failed to apply discount.”.
8. When checking out a cart, if the payment method and address does not exist, it should throw a custom exception with message as “Failed to complete checkout.”.
9. When adding a product in cart, if the user, product, quantity and price does not exist, it should throw a custom exception with message as “Failed to add products to the cart.”.
10. When fetching cart details, if the user does not exist, it should throw a custom exception with message as “Failed to view the contents of the cart.”.
11. When updating a cart item, if the user, item and quantity does not exist, it should throw a custom exception with message as “Failed to update item in the cart.”.
12. When removing an item from the cart, if the user and item does not exist, it should throw a custom exception with message as “Failed to remove item from the cart.”.

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.

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/discount/:userId Http Method: POST Parameter: userId Return: updated cart	Apply discount on cart
6. /api/products/cart/:userId Http Method: GET Parameter: userId Return: returns the cart	View the cart for that user

7. /api/products/cart/add/:userId

Http Method	POST
Parameter	userid
Return	updated cart

Adds item in the cart

8. /api/products/cart/checkout/:userId

Http Method	POST
Parameter	userId
Return	return successful message with created order id

Checkout the cart

9. /api/products/cart/update/:useId/:itemId

Http Method	PUT
Parameter	userId, itemId
Return	updated cart

Updates an item in cart

10. /api/products/cart/remove/:userId/:itemId

Http Method	DELETE
Parameter	userId, itemid
Return	removed item

Remove an item in cart

11. /api/products/:id

Http Method	GET
Parameter	id
Return	product

Gets the product by id

12. /api/products/:id

Http Method	PUT
Parameter	id
Return	updated product

Updated the product by id

13. /api/products/:id

Http Method	DELETE
Parameter	id
Return	deleted product

Deletes the product by id

4 TEMPLATE CODE STRUCTURE

4.1 Products code structure

1) MODULES/PRODUCTS: controller

Resources

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

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

3) MODULES/PRODUCTS: routes

Resources

File	Description	Status
Product routes	Routes for product	Partially implemented.

4) MODULES/PRODUCTS: service

Resources

Class	Description	Status
ProductService	• Defines ProductService	Already implemented.

5) MODULES/PRODUCTS: service/impl

Resources

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

.

5 METHOD DESCRIPTIONS

5.1 PRODUCT - Method Descriptions:

1. ProductController Class - Method Descriptions(Based on Route Mapping):

Method	Task	Implementation Details
createProduct	To create a new product	- The request type should be POST with URL /api/products. - Call productService.createProduct(req.body). - Return 201 status with created product. - On error, respond with 500 and message: 'Failed to create product.'
getProduct	To retrieve a product by ID	- The request type should be GET with URL /api/products/:id. - Call productService.getProduct(req.params.id). - Return the product. - On error, respond with 404 and message: 'Product not found.'
updateProduct	To update a product by ID	- The request type should be PUT with URL /api/products/:id. - Call productService.updateProduct(req.params.id, req.body). - Return the updated product. - On error, respond with 404 and message: 'Product not found.'

deleteProduct	To delete a product by ID	- The request type should be DELETE with URL /api/products/:id. - Call productService.deleteProduct(req.params.id). - Return the deleted product. - On error, respond with 404 and message: 'Product not found.'
getAllProducts	To fetch all products	- The request type should be GET with URL /api/products/all. - Call productService.getAllProducts(). - Return all products. - On error, respond with 500 and message: 'Failed to retrieve products.'
getTopRatedProducts	To get top-rated products with limit	- The request type should be GET with URL /api/products/top/:limit. - Call productService.getTopRatedProducts(limit). - Return top-rated products. - On error, respond with 500 and message: 'Failed to retrieve top rated products.'
searchProduct	To search products by name/description	- The request type should be GET with URL /api/products/search. - Call productService.searchProduct(name, description). - Return matching products. - On error, respond with 500 and message: 'Failed to search products.'
applyDiscount	To apply discount for a user's cart	- The request type should be POST with URL /api/products/discount/:userId. - Call productService.applyDiscount(userId, discountPercentage). - Return discount result. - On error, respond with 500 and message: 'Failed to apply discount.'
checkoutCart	To checkout a user's cart	- The request type should be POST with URL /api/products/cart/checkout/:userId. - Call productService.checkoutCart(userId, paymentMethod, address). - Return checkout result. - On error, respond with 500 and message: 'Failed to checkout cart.'

addToCart	To add a product to a user's cart	- The request type should be POST with URL /api/products/cart/add/:userId. - Call productService.addToCart(userId, productId, quantity, price). - Return updated cart. - On error, respond with 500 and message: 'Failed to add to cart.'
viewCart	To view the contents of a user's cart	- The request type should be GET with URL /api/products/cart/:userId. - Call productService.viewCart(userId). - Return cart details. - On error, respond with 500 and message: 'Failed to view cart.'
updateCartItem	To update quantity of a cart item	- The request type should be PUT with URL /api/products/cart/update/:userId/:itemId. - Call productService.updateCartItem(userId, itemId, quantity). - Return update result. - On error, respond with 500 and message: 'Failed to update cart item.'
removeCartItem	To remove an item from a cart	- The request type should be DELETE with URL /api/products/cart/remove/:userId/:itemId. - Call productService.removeCartItem(userId, itemId). - Return removal result. - On error, respond with 500 and message: 'Failed to remove cart item.'

2. ProductServiceImpl Class - Method Descriptions:

Method	Task	Implementation Details
createProduct	To create a new product	- Accept product data and create a new product using Product.create(). - Return the newly created product. - On error, throw: 'Failed to create product.'

getProduct	To retrieve a product by its ID	- Use Product.findById(productId). - If not found, throw: 'Product not found.' - Return the found product. - On error, throw: 'Failed to get product.'
updateProduct	To update an existing product by ID	- Use Product.findByIdAndUpdate(productId, updatedProduct, { new: true }). - If not found, throw: 'Product not found.' - Return the updated product. - On error, throw: 'Failed to update product.'
deleteProduct	To delete a product by ID	- Use Product.findByIdAndDelete(productId). - If not found, throw: 'Product not found.' - Return the deleted product. - On error, throw: 'Failed to delete product.'
searchProduct	To search products based on name and/or description	- Build query object using regex for name and description fields. - Use Product.find(query) to search. - Return the list of matching products. - On error, throw: 'Failed to search for products.'
getTopRatedProducts	To retrieve top-rated products up to a limit	- Use Product.find().sort({ ratings: -1 }).limit(limit). - Return the list of top-rated products. - On error, throw: 'Failed to get top rated products.'
getAllProducts	To retrieve all products	- Use Product.find() to retrieve all products. - Return the list of all products. - On error, throw: 'Failed to get all products.'
applyDiscount	To apply a discount to all products in the user's cart	- Validate the cart and discount percentage. - Apply discount to item prices and save the cart. - Return a success message. - On error, throw: 'Failed to apply discount.'

checkoutCart	To place an order and clear the cart	- Retrieve user's cart and calculate total amount. - Find cart as per userId, if not found throw “Cart not found.”. - Delete the user's cart. - On error, throw: 'Failed to complete checkout.'
addToCart	To add a product to a user's cart	- If item exists, increment quantity. - Else, add new item to the cart using $addToSet. - Return the updated cart. - On error, throw: 'Failed to add products to the cart.'
viewCart	To view a user's cart and product details	- Use Cart.findOne({ userId }).populate('items.product'). - Return the user's cart with product details. - On error, throw: 'Failed to view the contents of the cart.'
updateCartItem	To update quantity of a cart item	- Use Cart.findOneAndUpdate() with itemId and quantity. - Return the updated cart. - On error, throw: 'Failed to update item in the cart.'
removeCartItem	To remove an item from the cart	- Use $pull to remove item by itemId. - Return the updated cart. - On error, throw: 'Failed to remove item from the cart.'

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**