System Requirements Specification

Index

For

E-Commerce

Version 1.0

TABLE OF CONTENTS

E-COMMERCE

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

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

Following is the requirement specifications:

	E-Commerce
Modules	
1	Product
2	Order
3	Middleware

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

Order Module Functionalities	
1	Get all orders
2	Create an order
3	Get order by id
4	Update an order it’s id
5	Delete an order by it’s id
6	Get order for user
7	Cancel any order
8	Get the payment details of any order
9	Process the payment for any order
10	Get the analytics of orders with data for totalOrders, average order amount, highest and lowest order amount
11	Generate the invoice for any order with order id, order date and total amount details
12	Track the order shipment by returning status and tracking number
13	Get the revenue analytics which returns the total, highest and lowest revenue
14	Check whether the product is ordered or not

Middleware Module Functionalities	
1	Implement logic to check and validate token in authUserMiddlware file.
2	Implement logic to check whether a logged in user is admin or not in authAdminMiddleware.

2 Assumptions, Dependencies, Risks / Constraints

2.1 Order Constraints

1. When creating an order userId, product, totalAmount, paymentDetails and shipmentDetails are mandatory fields, on failing it should throw a custom exception.
2. When fetching an order by ID, if the order ID does not exist, the operation should throw a custom exception.
3. When updating an order, if the order ID does not exist, the operation should throw a custom exception.
4. When removing an order, if the order ID does not exist, the operation should throw a custom exception.
5. When canceling an order, if the order ID does not exist, it should throw a custom exception.
6. When fetching payment details for an order, if the order ID does not exist, it should throw a custom exception.
7. When processing the payment for an order, if the order ID does not exist, it should throw a custom exception.
8. When generating an invoice for an order, if the order ID does not exist, it should throw a custom exception.
9. When tracking an order, if the order ID does not exist, it should throw a custom exception.

2.2 Product Constraints

1. When creating product name and price are mandatory fields, on failing it should throw a custom exception.
2. When fetching a product by ID, if the product ID does not exist, the operation should throw a custom exception.
3. When updating a product, if the product ID does not exist, the operation should throw a custom exception.
4. When removing a product, if the product ID does not exist, the operation should throw a custom exception.

5. When adding a comment in a blog, if the blog ID does not exist, it should throw a custom exception.
6. When searching a product, if the name or description does not exist, it should throw a custom exception.
7. When applying some discount on a product, if the discount percentage does not exist, it should throw a custom exception.
8. When checking out a cart, if the payment method and address does not exist, it should throw a custom exception.
9. When adding a product in cart, if the user, product, quantity and price does not exist, it should throw a custom exception.
10. When fetching cart details, if the user does not exist, it should throw a custom exception.
11. When updating a cart item, if the user, item and quantity does not exist, it should throw a custom exception.
12. When removing an item from the cart, if the user and item does not exist, it should throw a custom exception.

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 ORDER RESTPOINTS

Note: All order routes must be authenticated and only accessible on valid token.

URL Exposed		Purpose
1. /api/orders/all		Fetches all the orders
Http Method	GET	
Parameter	-	
Return	list all orders	
2. /api/orders/create		Creates a new order
Http Method	POST	
Parameter	-	
Return	newly created order	
3. /api/orders/:id		Fetches an order by id
Http Method	GET	
Parameter	id	
Return	get an order	
4. /api/orders/:id		Updates an order by id
Http Method	PUT	
Parameter	id	
Return	updates an order	
5. /api/orders/:id		Deletes an order by id
Http Method	DELETE	
Parameter	id	
Return	deletes an order	
6. /api/orders/user/:userId		Fetches an order for that user
Http Method	GET	
Parameter	userId	
Return	order	
7. /api/orders/cancel/:id		Cancels an orders
Http Method	DELETE	
Parameter	id	
Return	canceled order	

8. /api/orders/:id/payment		Get the payment details of any order
Http Method	GET	
Parameter	-	
Return	return the payment details	

9. /api/orders/:id/pay		Process the payment for any order
Http Method	GET	
Parameter	-	
Return	message whether payment is processed or not	

10. /api/orders/analytics		Get the analytics of orders with data for totalOrders, average order amount, highest and lowest order amount
Http Method	GET	
Parameter	-	
Return	order analytics	

11. /api/orders/:id/invoice		Generate the invoice for any order with order id, order date and total amount details
Http Method	GET	
Parameter	-	
Return	invoice	

12. /api/orders/:id/shipment		Track the order shipment by returning status and tracking number
Http Method	GET	
Parameter	-	
Return	shipment details	

13. /api/orders/revenue		Get the revenue analytics which returns the total, highest and lowest revenue
Http Method	GET	
Parameter	-	
Return	analytics	

14. /api/orders/ordered/:userId/:productId		Check whether the product is ordered or not
Http Method	GET	
Parameter	userId, productId	
Return	returns order	

3.2 PRODUCT RESTPOINTS

Note:

1. **Any route which must be authenticated (there must be valid token) are marked with authMiddleware**
2. **Any route which must be authorized only for admin (user must be of type admin) are marked with adminMiddleware.**
3. **Any route which must be both authenticated and authorized are marked with authMiddleware and adminMiddleware.**

URL Exposed	Purpose
1. /api/products/all Http Method: GET Parameter: - Return: list of products	Fetches all the products
2. /api/products/create Http Method: POST Parameter: - Return: newly created product	Creates a new product **[authMiddleware, adminMiddleware]**
3. /api/products/search Http Method: GET Parameter: - Return: searched products	Search the product by name or description
4. /api/products/top-rated/:limit Http Method: GET Parameter: limit Return: list of products	Fetches the top rated products **[authMiddleware]**
5. /api/products/discount/:userId Http Method: POST Parameter: userId Return: updated cart	Apply discount on cart **[authMiddleware]**
6. /api/products/cart/:userId Http Method: GET Parameter: userId Return: returns the cart	View the cart for that user **[authMiddleware]**

7. /api/products/cart/add/:userId		Adds item in the cart **[authMiddleware]**
Http Method	POST	
Parameter	userid	
Return	updated cart	

8. /api/products/cart/checkout/:userId		Checkout the cart **[authMiddleware]**
Http Method	POST	
Parameter	userId	
Return	return successful message with created order id	

9. /api/products/cart/update/:useId/:itemId		Updates an item in cart **[authMiddleware]**
Http Method	PUT	
Parameter	userId, itemId	
Return	updated cart	

10. /api/products/cart/remove/:userId/:itemId		Remove an item in cart **[authMiddleware]**
Http Method	DELETE	
Parameter	userId, itemid	
Return	removed item	

11. /api/products/:id		Gets the product by id
Http Method	GET	
Parameter	id	
Return	product	

12. /api/products/:id		Updated the product by id
Http Method	PUT	
Parameter	id	
Return	updated product	

13. /api/products/:id		Deletes the product by id **[authMiddleware, adminMiddleware]**
Http Method	DELETE	
Parameter	id	
Return	deleted product	

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.

.

4.2 Orders code structure

1) MODULES/ORDER: controller

Resources

OrderController (Class)	This is the controller class for the order module.	To be implemented

2) MODULES/ORDER: dao

Resources

File	Description	Status
models/order model	Model for order	Already implemented
schemas/order schema	Schema for order	Already implemented

3) MODULES/ORDER: routes

Resources

File	Description	Status
Order routes	Routes for order	Partially implemented.

4) MODULES/ORDER: service

Resources

Class	Description	Status
OrderService	• Defines OrderService	Already implemented.

5) MODULES/ORDER: service/impl

Resources

Class	Description	Status
OrderServiceImpl	• Implements OrderService.	To be implemented.

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. **If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page) then you need to use CTRL+Shift+B-command compulsorily on code IDE. This will push or save the updated contents in the internal git/repository. Else the code will not be available in the next login.**
5. **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.**
6. **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.**

7. **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**
8. **You need to use CTRL+Shift+B - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.**