# System Requirements Specification

## Index

For

## Product Manager

Version 1.0

## TABLE OF CONTENTS

BACKEND-EXPRESS NODE APPLICATION 3

1 Project Abstract 3

2 Assumptions, Dependencies, Risks / Constraints 4

2.1 Product Constraints 4

3 Rest Endpoints 5

3.1 ProductRoutes 5

4 Template Code Structure (modules) 6

4.1 controller 6

4.2 dao 6

4.3 routes 6

4.4 service 6

4.5 serviceImpl 7

5 Execution steps to follow for Backend 7

# 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.

2.3 When fetching a product by ID, if the product ID does not exist, the operation should throw a custom exception.

2.4 When updating a product, if the product ID does not exist, the operation should throw a custom exception.

2.5 When removing a product, if the product ID does not exist, the operation should throw a custom exception.

2.6 When adding a comment in a blog, if the blog ID does not exist, it should throw a custom exception.

2.7 When searching a product, if the name or description 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 PRODUCT RESTPOINTS

| URL Exposed | | Purpose |
|---|---|---|
| 1. /api/products/all | | Fetches all the products |
| Http Method | GET | |
| Parameter | - | |
| Return | list of products | |
| 2. /api/products/create | | Creates a new product |
| Http Method | POST | |
| Parameter | - | |
| Return | newly created product | |
| 3. /api/products/search | | Search the product by name or description |
| Http Method | GET | |
| Parameter | - | |
| Return | searched products | |
| 4. /api/products/top-rated/:limit | | Fetches the top rated products |
| Http Method | GET | |
| Parameter | limit | |
| Return | list of products | |
| 5. /api/products/:id | | Fetches the product by id |
| Http Method | GET | |
| Parameter | id | |
| Return | fetch product | |
| 6. /api/products/:id | | Updates the product by id |
| Http Method | PUT | |
| Parameter | id | |
| Return | updated product | |
| 7. /api/products/:id | | Deletes the product by id |
| Http Method | DELETE | |
| Parameter | id | |
| Return | deleted product | |

# 4 TEMPLATE CODE STRUCTURE

## 4.1 MODULES/PRODUCTS: controller

**Resources**

| | | |
|---|---|---|
| **ProductController**<br>**(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**<br><br>**models/product model** | Models for cart and product | Already implemented |
| **schemas/cart schema**<br><br>**schemas/product schema** | Schemas for cart and product | Already implemented |

## 4.3 MODULES/PRODUCTS: routes

**Resources**

| **File** | **Description** | **Status** |
|---|---|---|
| **Product routes** | Routes for product | Partially 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. |

## 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.