Sprint-2 User story document

Sprint Document - Designing Backend APIs and Implementation

Document History

Author	Version	Date	Description	Comments
Giridharan Sankaran	**0.1**	**11-Feb-2024**	**Functionality and Technical document update**	
Ashwin Krishna Mani and Reshma Shaji	**0.2**	**15-Feb-2024**	**Test results document updated**	
Yogesharvind Nedumaran	**1.0**	**15-Feb-2024**	**Reviewed and Baselined**	

1. **Background**

In the ever-evolving landscape of retail, effective management of product information stands as a linchpin for operational excellence, regulatory adherence, and the facilitation of seamless customer interactions. Our current sprint is dedicated to crafting a robust backend API Server system, fortified with state-of-the-art Login Security, leveraging the power of SpringBoot and Java.

Our paramount goal in this sprint is to engineer a backend infrastructure capable of extracting precise, abstracted data, facilitating seamless updates to existing records, and providing comprehensive functionalities accessible to end users through front-end UI interactions.

Through the utilization of SpringBoot, we aim to streamline backend interactions, allowing us to prioritize the design of APIs capable of executing intricate operations and furnishing pertinent statistics and features. Furthermore, we're instituting separate authentication mechanisms for Clerks and Managers, affording the latter additional permissions for overseeing Warehouse Stock, thereby ensuring a secure and tailored experience for each user role.

2. **Proposed Functional Solution**

For now, the proposed solution is to follow the Model-View-Controller [MVC] Pattern and design for all the needed collections from the database namely Product, Expiry, Discount, and Supplier [May vary as designs become more clear]. The Product Class is used to manage and catalog all the products available in the store, Expiry tracks dates for all the products and tells us which products are on the brink of expiry and which aren’t. Discount Class is used to help us select a percentage discount to apply on products and Supplier class has the B2B Details which will be used for placing Re-orders.

To implement this solution, we are broadly using four packages - Models, Repos, Services, and Controllers. Each table would need four classes from these packages to represent or modify the data in these tables.
For Eg: The Inventory Table has Product Model, ProductRepo, ProductService, and ProductController.

4. **Design Analysis and Solution:**

4.1 Technical changes:

4.1.1 Design Analysis:

As part of the technical changes, the implementation will use SpringBoot and Java to design backend APIs for Product, Supplier, Discount, and Expiry, and add authentication mechanisms separately for Clerk and Manager.

4.1.2 Implementation for Inventory Collection:
4.1.3 Implementation for Supplier Collection:
4.1.4 Implementation for Discount Collection:
4.1.5 Implementation for Expiry Collection:
4.1.6 How is Login Done? For Clerk and Manager

4.1.7 API testing using Post-Man:
All the APIs are hit and tested using Postman, details of which shall be attached to this document.
Additionally, from the next sprint, we plan on writing all unit and integration tests and adding a clear CI/CD pipeline to run builds every time we add changes in the code base.

5. **Use Test Cases:**

S.NO	**Collection Tested**	**API Tested**	**Description**	**URL and Request Body**	**Expected Response**
1.	Discount	findingAll Discounts()	To know which items are labelled for discounts.	• /discount • None	Returns items with discounts
2.	Discount	findingDiscountWithUPCID()	To find if certain items are discounted or not based on their UPCID	• /discount/UPCID/{UPCID} • None	Returns an item with a particular UPCID if it has a discount
3.	Discount	updateDiscountPrice()	To enable the Store Manager to Change the Discount for items.	• /discount/{UPCID} • { “discountPrice” : 7.0 }	Changes the discountPrice for the selected item by UPCID. ** Currently returns all the discounted items after update. Need to fix this. Bug for next sprint.
4.	Expiry	findallExpiring Products()	To know which products are expiring based on certain criteria.	• /expiry • None	Returns all the products nearing expiry
5.	Expiry	addingExpiry()	Categorizing products when they satisfy the expiring threshold set for a particular item	• /expiry •	

6.	Expiry	findExpiryWithUPCID()	Find out the expiry date for certain products.	• /expiry/UPCID/{UPCID}	Returns the object with expiry date. ** Should we only return expiry date? Need to discuss.
7.	Expiry	findExpiryWithDate()	Find out which products are expiring within a particular date?	• /expirydate •	
8.	Supplier	addingBrand()	Lets you add new businesses to place orders with	• /supplier • {id:, brandName:, supplierEmail:, supplierContact:}	Returns all the details of suppliers after adding. **NEED to change this.
9.	Supplier	findAllBrands()	Provide all the connected brands and their contact details.	• /supplier • None	Returns the details of all the brands.
10.	Supplier	gettingSuppliersWithBrandNames()	Provide details of suppliers who sell a particular brand.	• /supplier • /brandName{brandName}	Returns the Suppliers who can provide a particular brand's products.
11.	Product	addProduct()	Adding New Products to the catalog()	• /create • All details of a Product	Returns all the products list updated() **NEED to discuss this.
12.	Product	getProductByUPCID()	find products by their UPCID()	• /UPCID/{UPCID}	Returns the product detail.
13.	Product	getAllProducts()	Get All Products in catelog	• /all	Returns all the products
14.	Product	getProductsUsingBrandName()	Find Products using Brand Name	• Brand/{Brand}	Returns all the products with that brand name.
15.	Product	getProductsUsingCategory()	find Products of the same category	Category/{Category}	Returns all products with similar category.
16.	Product	getProductsUsingProcurementDate()	find Products using the Date Of Procurement	DateOfP/{dateOfP}	Returns all products based

					on their date of procurement. **Need to change this name to Date Of Manufacturing
17.	Product	getProductsUsingExpiryDate()	Find Products nearing Expiry	DateOfE/{dateOfE}	Returns Products nearing expiry.

6. **Test Results Document:** Testing Report Using PostMan

7. **Assumptions:**
   - We still haven't used Authentication. Assuming all the APIs are accessible by both clerk and manager.
   - Certain Create APIs are returning all the records. Will fix it in the next sprint.
   - All APIs were tested using Postman.

8. **Exclusions (Specific Functionality that is not covered by this Development):**
   A. Thorough Unit and Integration Testing.
   B. Redundant details to be removed.
   C. Cleaning up code.
   D. Adding Comments where necessary.
   E. Authentication

9. **KHD:**

<To be attached>