

PROJECT Design Documentation

Team Information

- Team name: Myriad Desserts
- Team members
 - Giselle
 - Chaitanya
 - Avasyu
 - Ammar
 - Mohammed

Executive Summary

This project entails the development of an e-store for Myriad Desserts, a confectionary shop. Myriad Desserts is a modernized version of a traditional confectionary store where customers can choose a product and customize certain ingredients to their liking.

Purpose

The objective of this project is to create and launch an online store for Myriad Desserts Confectionary. Customers and employees are the primary users of this e-store. Customers will be able to view a list of products, choose a product, and customize the ingredients of a specific product through this e-store. Employees of the store are another group of users who can use the e-store interface to manage inventory of required supplies.

Glossary and Acronyms

Term	Definition
API	Application programming interface
JSON	Java Script Object Notation
OOP	Object Oriented Programming
URL	Unifrom Resource Locator
REST	Representational State Transfer
SPA	Single page
MVP	Minimum Viable Product

Requirements

- As an owner I want to be able to view inventory details to maintain stock.
- As a Customer, I want to create an account so that I can order products.
- As a customer I need to add and remove products from the cart so that i can purchase what I want.
- As a Customer, I want to login into my account so that I can view a list of products

- As a customer I want to add and remove ingredients from my selected product to make edited product.
- As a customer I want to be able to use checkout on my cart so that I can place an order
- As a customer I want to be able to choose products to be in the cart in order to purchase them.
- As an owner I want a notification when stock reaches low level so that I can resupply inventory when needed.
- As a customer I want to see a Create Account option so that I can successfully complete Account Registration with the e-store.
- As a Customer I want to be able to edit the ingredients so that I can make a customized product.
- As an owner I want to get notifications so that I know which ingredients are running low on stock.
- As an owner I want authentication methods set up so that customers and employees can authenticate themselves.
- As a Employee, I want to login into the admin account so that I can perform inventory management

Definition of MVP

The MVP must have the ability to create new products and display them for the customer to be able to see and place the order. It also needs to be able to search for a specific product instead of manually reading through the whole list. MVP should also provide us the ability to update a product information by the product owner in case of any critical situations manual override is possible. Finally we need to be able to remove the product either if it's not in demand or if the making charges of the product is too expensive to continue.

MVP Features

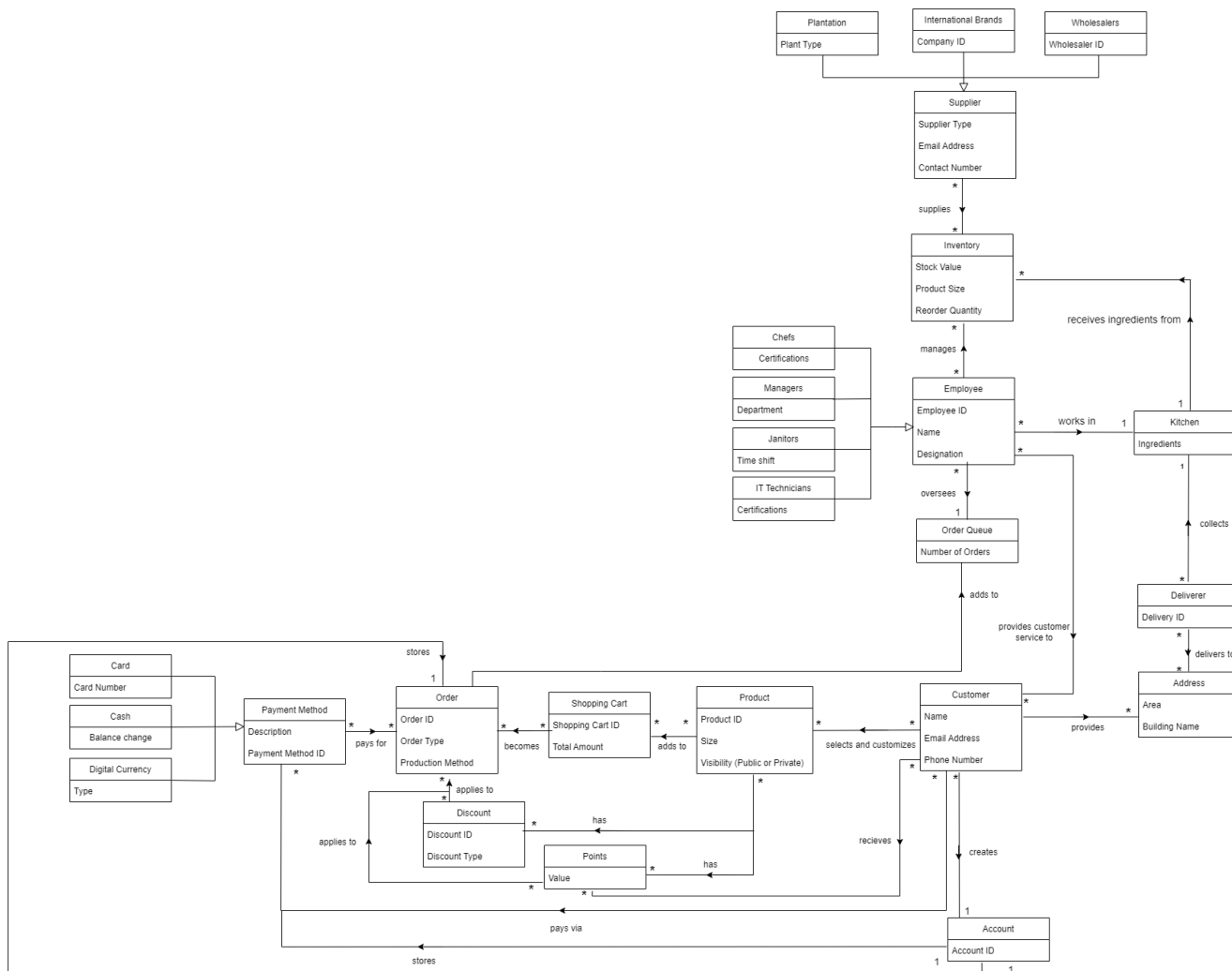
- Creating new products
- Searching for a product
- Getting the entire inventory
- Update a product
- Get a single product
- Delete a single product
- Customer Log in
- Employee Log in
- Add to cart
- Delete item from cart
- remove and add ingredients from product

Roadmap of Enhancements

- Customer accounts: allows customer to create accounts to store their information for future orders, all accounts are password protected.
- Store management: allows owner to manage inventory, add and remove products, and offer discounts on products
- Checkout: allows customers to make orders and have them delivered to the location linked to the user account. it also sends the order to the staff so that they can start preparing it.

Application Domain

This section describes the application domain.



The domain model shown above highlights key entities such as Customer, Account, Inventory, Order, and Shopping Cart. The relationships between these entities are crucial and should run smoothly in order for the store to function as smoothly as possible. The e-store's functionality primarily involves entities such as Customer, Product, Shopping Cart, Order and Order.

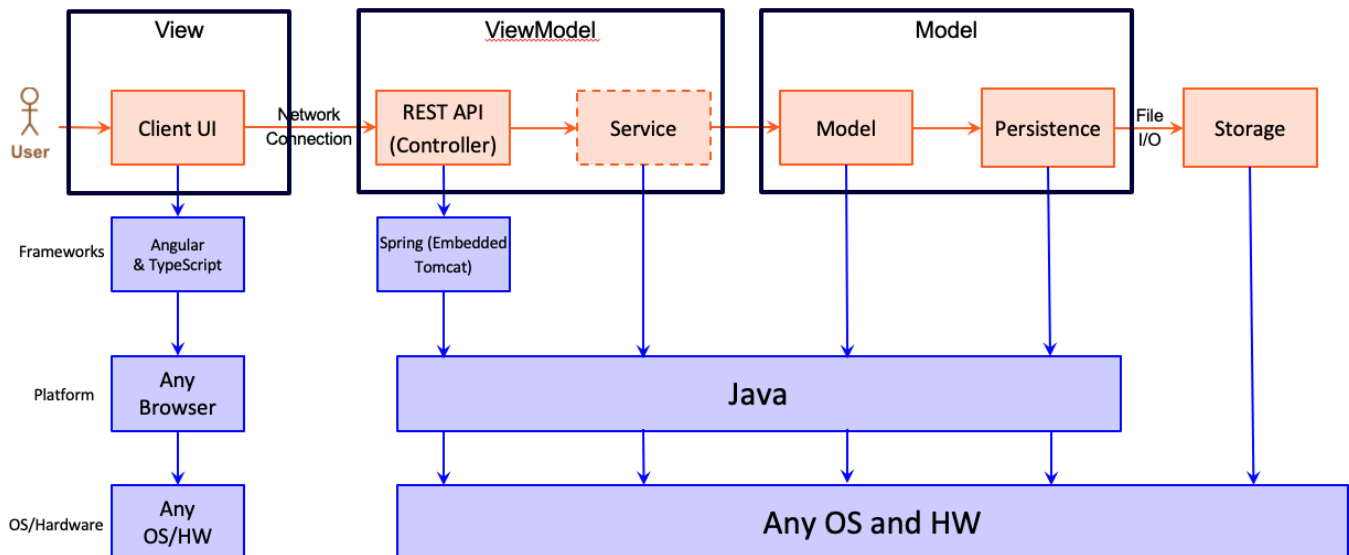
The following is the flow and relationship between the various important domain entities: The customer first logs into their account, where their name, username and password is stored. The Account is used to direct customers to the login page where they can see the Product. The Product is then selected and added to shopping Cart, or the Product can be customized using ingredients available in the inventory and then add the customized product to the cart. The Customer can checkout of the shopping cart when they have finalized the selected items. Upon checking out of the shopping cart, Customers will be notified of the total price and an Order is placed. Meanwhile, Employees manage the inventory system, which is supplied by respective suppliers to replenish the stock.

Architecture and Design

This section describes the application architecture.

Summary

The following Tiers/Layers model shows a high-level view of the webapp's architecture.



The e-store web application, is built using the Model–View–ViewModel (MVVM) architecture pattern.

The Model stores the application data objects including any functionality to provide persistence.

The View is the client-side SPA built with Angular utilizing HTML, CSS and TypeScript. The ViewModel provides RESTful APIs to the client (View) as well as any logic required to manipulate the data objects from the Model.

Both the ViewModel and Model are built using Java and Spring Framework. Details of the components within these tiers are supplied below.

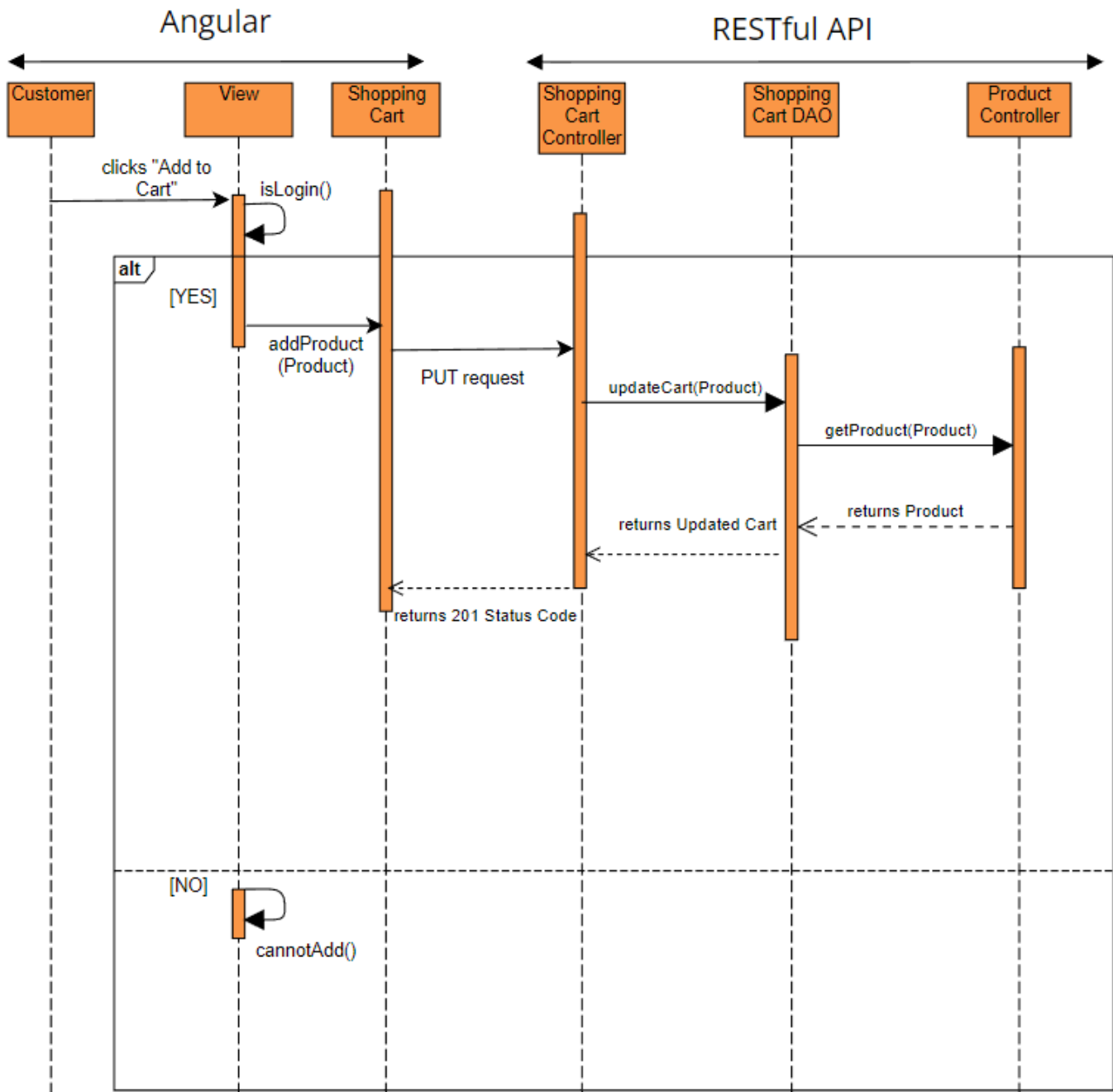
Overview of User Interface

Once the angular webpage is running the users are greeted with the login page. If the user hasn't created an account yet they will have to click the sign up button. Once the user signs up and logs in with the user credentials then they would see the products page. Here the user can add the products to the cart and customize and then the user can see the cart once again and checkout after which they would be returned to the main menu. On the other hand if the product manager logs in as an admin then he can see the products page and can also change the quantity price and other aspects of each product and also add or remove products itself.

View Tier

This view tier is where the user interacts with the application. This tier is responsible for outputting necessary information to the user and prompting input information from the user as well. The UI components mainly deal with showing essential components required for login, showing products, showing shopping cart and so on. For a specific component like shopping cart, the User can see a list of products and can choose to add to the cart. When they click on add to cart, an HTTP request is sent to the server. The server then gets the list of items that have been added to the shopping cart. The presentation tier then formats and represents this list of items to the client.

Below is the sequence diagram for the user adding an item to the shopping cart:



ViewModel Tier

The user first comes across the login page, for which a json server is running in the background to authenticate the credentials inputted. The page template contains static and dynamic content that is populated using the json server.

The components that exist in the view tier of the e-store are:

- 1. Login Page
- 2. Dashboard
- 3. Shopping Cart page

Model Tier

Provide a summary of this tier of your architecture. This section will follow the same instructions that are given for the View Tier above.

At appropriate places as part of this narrative provide one or more static models (UML class diagrams) with some details such as critical attributes and methods.

Static Code Analysis/Design Improvements

Discuss design improvements that you would make if the project were to continue. These improvement should be based on your direct analysis of where there are problems in the code base which could be addressed with design changes, and describe those suggested design improvements.

With the results from the Static Code Analysis exercise, discuss the resulting issues/metrics measurements along with your analysis and recommendations for further improvements. Where relevant, include screenshots from the tool and/or corresponding source code that was flagged.

Testing

This section will provide information about the testing performed and the results of the testing.

Acceptance Testing

In our Acceptance Test plan there is a total of 13 user stories and 28 Acceptance criterion. All of have the PASS status. None of the tested User stories are failing or experiencing bugs.

Below are User Stories and Acceptance Criterion as seen in the Acceptance Test Plan. [AT part 1](#)







As a Customer, I want to create an account so that I can order products.	Given that I am not yet a registered customer when I see a Sign up option then I must be able to create an account.			Pass	MM; 15/11/22
As a Customer, I want to login with my account so I can see the list of products	Given that I am a registered customer when I see the Login Option then I must be able to login.			Pass	AB; 15/11/22
	Given that I am not yet a registered customer when I enter the credentials and click on Login then I must be able to see Login failed message.			Pass	AK; 15/11/22
	Given that I am not yet a registered customer when I see the login page then I must be able to see a Signup option on the login page.			Pass	MM; 15/11/22
	Given that the user has entered their credentials when they click Login, they should be redirected to Dashboard.			Pass	AB; 15/11/22
	Given that I have not signed in when I see a Login option then I must be able to login using admin			Pass	AK; 15/11/22
As a Employee, I want to login into the admin account so that I can perform inventory management	Given that I have entered the admin credentials when I click on Login then I must be able to see			Pass	GC; 15/11/22
As a customer, I want to be able to choose products to be in the cart in order to purchase them.	Given that I see a product I like when I select to add this product to the cart then I expect a notification to pop up saying the an item is added to the cart.			Pass	SC; 17/11/22
	Given that I have clicked on add to cart when I click on the shopping cart icon then I expect to be			Pass	AB; 17/11/22
	Given that I have clicked on shopping cart icon when I am redirected to the shopping cart page then I expect to see the item I selected in the cart.			Pass	AK; 17/11/22
As a customer, I want to edit the shopping cart so I can make changes in my order preferences.	Given that I see the option to delete button for a specific item when I click on the delete button then I expect to see the item be removed from the cart.			Pass	MM; 18/11/22
	Given that I see the option to input quantity for a item in the cart when I enter a number then I expect the quantity for a specific item to be updated.			Pass	GC; 18/11/22
	Given that I am in the shopping cart and I see the go Back option when I click on the Go Back button then I expect the items on the cart to stay and I be redirected to the products page			Pass	SC; 18/11/22
	Given that I see a clear cart button when I click this button then I expect all items in the cart to be cleared.			Pass	AB; 18/11/22
As a Customer I want to be able to edit the ingredients so that I can make a customized product.	Given that I see the customize button when I click on it then I expect to see a list of available ingredients that I use to customize the product			Pass	SC; 19/11/22
	Given that I want to add ingredients to my selected product when I click on select button for a specific ingredient then I must see the product is customized with this ingredient in the shopping			Pass	GC; 19/11/22
	Given that I want to remove ingredients to my selected product when I click on deselect button for a specific ingredient then I must see the product is not customized with this ingredient in the			Pass	MM; 19/11/22
	Given that I see the option of preview checkout on the shopping cart, when I click it, I expect that the total price of items in the cart to show			Pass	AK; 19/11/22
As a customer, I want to finalize my preferences shown on the shopping cart so I can confirm my order.	Given that I see the option of checkout on the shopping cart, when I click it, I expect that my			Pass	AB; 19/11/22
	Given that I see the option of checkout on the shopping cart, when I click it, I expect a			Pass	GC; 19/11/22
	Given that I see the option of checkout on the shopping cart, when I click it, I expect that all			Pass	SC; 20/11/22

Unit Testing and Code Coverage

Controller Tier Analysis







estore-api > com.estore.api.estoreapi.controller

com.estore.api.estoreapi.controller

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
ProductController		75%		71%	3	15	10	54	1	8	0	1
ItemController		100%		100%	0	10	0	36	0	6	0	1
CartController		100%		100%	0	7	0	17	0	5	0	1
Total	57 of 444	87%	4 of 26	84%	3	32	10	107	1	19	0	3

estore-api > com.estore.api.estoreapi.controller

com.estore.api.estoreapi.controller

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
ProductController		75%		71%	3	15	10	54	1	8	0	1
ItemController		100%		100%	0	10	0	36	0	6	0	1
CartController		100%		100%	0	7	0	17	0	5	0	1
Total	57 of 444	87%	4 of 26	84%	3	32	10	107	1	19	0	3

The Controller tier has a coverage percentage of only 87%. This is mainly due to the fact of fewer test conditions were implemented to test potential failure scenarios. One of the fail scenarios includes passing a null object as a parameter to the method.





A few suggestions to improve the coverage rates for this specific tier could be :

1. Focus on implementing test conditions for areas of code that are most likely to have bugs or disrupt the functionality of the code, like passing a null object as a parameter
2. Create tests for specific parts of the code that support the main functionality first, and debug any errors related to the test code before moving on to the next important part of the code. This eases the debugging process if required.

Model Tier Analysis

estore-api > com.estore.api.estoreapi.model

com.estore.api.estoreapi.model


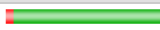
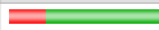

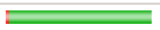
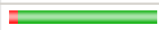



Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Item		47%	n/a	n/a	4	11	4	17	4	11	0	1
Product		100%	n/a	n/a	0	10	0	15	0	10	0	1
Cart		100%	n/a	n/a	0	6	0	9	0	6	0	1
Request		100%	n/a	n/a	0	4	0	7	0	4	0	1
Total	42 of 202	79%	0 of 0	n/a	4	31	4	48	4	31	0	4

The results of our model tier tests show that we have validated each and every one of the functions that make up our method, including all of the exceptions and any other potential failure areas, except for Item. Apart from Item, the results of our tests on the model tier suggest that there is 100% coverage and indicates that all methods have been tested and functioned as intended. The methodology and structure for testing this particular class could be used to improve the coverage rates for Item in this tier.

Persistence Tier Analysis

 `estore-api` >  `com.estore.api.estoreapi.persistence`

com.estore.api.estoreapi.persistence

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
 CartFileDAO		94%		77%	4	17	7	58	0	8	0	1
 ItemFileDAO		96%		93%	2	20	3	49	1	12	0	1
 ProductFileDAO		100%		93%	1	21	0	54	0	13	0	1
Total	22 of 767	97%	6 of 50	88%	7	58	10	161	1	33	0	3

According to the report, there is 97% coverage for the persistence tier. The remaining percentage occurs due to a lack of test statements to test scenarios where a null object might be passed as a parameter, where a Status code of Not Found would ideally be outputted. The persistence tier is mainly constitutes the ProductFileDAO, which contains methods about the product that is important for our software. Therefore, it is essential that we add the missing tests to make it reach a 100% coverage rate in order to avoid having any unexpected bugs in the future.










As suggested before, we could implement tests that are more directed towards testing the main functionalities of the code, which could include designing tests intended to deal with scenarios where the program could fail.

Another suggestion for improving coverage rates would be setting a proper methodology of how to implement tests. To ensure consistency, a common checklist could be created that could be applicable to similar methods, to test the most common scenarios.

Well-tested Component


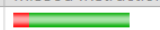






















 `estore-api` >  `com.estore.api.estoreapi.persistence`

com.estore.api.estoreapi.persistence

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
 CartFileDAO		94%		77%	4	17	7	58	0	8	0	1
 ItemFileDAO		96%		93%	2	20	3	49	1	12	0	1
 ProductFileDAO		100%		93%	1	21	0	54	0	13	0	1
Total	22 of 767	97%	6 of 50	88%	7	58	10	161	1	33	0	3

























 `estore-api` >  `com.estore.api.estoreapi.persistence` >  [CartFileDAO](#)

CartFileDAO

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
 getProducts(int)		86%		66%	2	4	1	7	0	1
 addProduct(Request)		94%		100%	0	2	2	13	0	1
 removeProduct(Request)		94%		100%	0	2	2	13	0	1
 CartFileDAO(String, ObjectMapper)		81%		n/a	0	1	2	8	0	1
 load()		100%		100%	0	2	0	5	0	1
 getCartArray(int)		100%		66%	2	4	0	8	0	1
 save(int)		100%		n/a	0	1	0	3	0	1
 static {...}		100%		n/a	0	1	0	1	0	1
Total	14 of 260	94%	4 of 18	77%	4	17	7	58	0	8








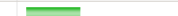




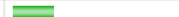













estore-api > com.estore.api.estoreapi.persistence > ItemFileDAO

ItemFileDAO

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
nextId()		0%		n/a	1	1	3	3	1	1
load()		100%		75%	1	3	0	9	0	1
getItemsArray(String)		100%		100%	0	4	0	8	0	1
updateItem(Item)		100%		100%	0	2	0	6	0	1
deleteItem(int)		100%		100%	0	2	0	5	0	1
getItem(int)		100%		100%	0	2	0	4	0	1
save()		100%		n/a	0	1	0	3	0	1
ItemFileDAO(String, ObjectMapper)		100%		n/a	0	1	0	5	0	1
findItems(String)		100%		n/a	0	1	0	2	0	1
getItems()		100%		n/a	0	1	0	2	0	1
static {...}		100%		n/a	0	1	0	1	0	1
getItemsArray()		100%		n/a	0	1	0	1	0	1
Total	8 of 238	96%	1 of 16	93%	2	20	3	49	1	12

estore-api > com.estore.api.estoreapi.persistence > ProductFileDAO

ProductFileDAO

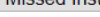







Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
load()		100%		75%	1	3	0	9	0	1
getProductsArray(String)		100%		100%	0	4	0	8	0	1
createProduct(Product)		100%		n/a	0	1	0	5	0	1
updateProduct(Product)		100%		100%	0	2	0	6	0	1
deleteProduct(int)		100%		100%	0	2	0	5	0	1
getProduct(int)		100%		100%	0	2	0	4	0	1
save()		100%		n/a	0	1	0	3	0	1
ProductFileDAO(String, ObjectMapper)		100%		n/a	0	1	0	5	0	1
findProducts(String)		100%		n/a	0	1	0	2	0	1
getProducts()		100%		n/a	0	1	0	2	0	1
nextId()		100%		n/a	0	1	0	3	0	1
static {...}		100%		n/a	0	1	0	1	0	1
getProductsArray()		100%		n/a	0	1	0	1	0	1
Total	0 of 269	100%	1 of 16	93%	1	21	0	54	0	13

The Product class under the Model tier has been well-tested. As the figures above shows, there is a small percentage of missed instructions or any missed branches, and there is good testing coverage of all the methods shown. This helps indicate that the code is working as intended. The code for this component represents well-tested code that is efficient, dependable and consistent CI/CD.

Poorly-test Component

estore-api > com.estore.api.estoreapi.model

com.estore.api.estoreapi.model

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Item		47%		n/a	4	11	4	17	4	11	0	1
Product		100%		n/a	0	10	0	15	0	10	0	1
Cart		100%		n/a	0	6	0	9	0	6	0	1
Request		100%		n/a	0	4	0	7	0	4	0	1
Total	42 of 202	79%	0 of 0	n/a	4	31	4	48	4	31	0	4

Item

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods
toString()	<div><div></div></div>	0%	n/a	n/a	11	11	1
setQuantity(Integer)	<div><div></div></div>	0%	n/a	n/a	11	11	1
setPrice(Double)	<div><div></div></div>	0%	n/a	n/a	11	11	1
setName(String)	<div><div></div></div>	0%	n/a	n/a	11	11	1
Item(int, String, int, double, String[])	<div><div></div></div>	100%	n/a	n/a	01	07	01
static {...}	<div><div></div></div>	100%	n/a	n/a	01	01	01
getIngredients()	<div><div></div></div>	100%	n/a	n/a	01	01	01
getId()	<div><div></div></div>	100%	n/a	n/a	01	01	01
getName()	<div><div></div></div>	100%	n/a	n/a	01	01	01
getQuantity()	<div><div></div></div>	100%	n/a	n/a	01	01	01
getPrice()	<div><div></div></div>	100%	n/a	n/a	01	01	01
Total	42 of 80	47%	0 of 0	n/a	411	417	411

Although few gaps in coverage have been noticed across most tiers, the class Item under the Model tier has the most significant gap in terms of coverage. Item has only 47% coverage. This percentage of missed branches and missed instructions are the highest compared to other classes in other tiers.