Calorify Design Documentation

Team Information

- Team name: Calorify
- Team members
  - Sohan Saimbhi
  - Hritish Mahajan
  - Christin Alex
  - Syed Basit Hussain
  - Eslam Tarrum

Executive Summary

Calorify is an e-commerce store which is dedicated to allowing the customers to choose and purchase healthy food products in order to cater to their calorie intake requirements. We want to provide a platform on which customers can, easily and safely, browse and purchase various healthy food products including salads, wraps, dips and our signature dishes. It is completely user-input based so that we can meet the needs of our customer and help them acheive their diet goals.

Purpose

Our purpose is to allow our customers to purchase high quality healthy food products to satisfy their daily calorie requirements with a considerable amount of freedom towards customization. The inventory manager or the owner of the store, their functionality includes that they can view, add, update, and delete products from the menu listing.

Glossary and Acronyms

Term	Definition
SPA	Single Page Application
User	Any person who accesses our calorify website
Buyer	A person who register a new account and logs in with the intention to purchase the food products
Admin	A person who logs in with the admin username and upkeeps the inventory of calorify
Shopping cart	A collection of food products chosen by the buyer for them to purchase at the checkout
Product	The food products which are sold in the estore
MVP	Minimum Viable Product
10% feature	An additional feature implemented on top of the MVP

Term	Definition
DAO	Data Access Object, within the persistence tier
HTTP	HyperText Transfer Protocol, a network protocol for specifications on how data should be transferred
CSS	Cascading Style Spreadsheets that describe how HTML elements are to be displayed on screen and deals with the styling of the website
UI	View section of the project to be shown to the user
API	Application Programming Interface, connection interface between computers or computer programs. It helps the client application communicate with the server application.

Requirements

The store will be based on Spring Boot as a backend and Angular JS as front end working together to provide the functionality of the Calorify store. We have considered the following functional and non-functional requirements:

1. Functional requirements:

- The website must allow the customer to create an account and login
- The website must not allow identical users to be registered
- The website must allow the customer to view the list of products and their nutritional information
- The application will support adding new products to the inventory, with only the administrator having access to this function.
- The website must allow admin to create, update, read and delete products from inventory
- The website must only allow specific pages to be viewed by admin
- The website must not allow specific pages to be viewed by customer
- The website must not allow specific pages to be viewed by unregistered user
- The website must allow the customer to filter products by price
- The website must allow customers to find products containing the user search term in their name
- The application will support sorting products by price, name, and calories.
- The website must allow customer to see details of a product of interest
- The website must allow the customer to read, add, remove, update and delete products from their cart
- The website must allow the customer to checkout and purchase the products in their cart
- The website must allow the customer to view their purchase history
- The website must allow the customer to provide feedback on the products

2. Non-functional requirements:

- The website must be secure and protect the customer's personal information
- The website must be responsive and work on all devices
- The website must be easy to use and navigate
- The website must have a modern and sleek design

Definition of MVP

The minimum viable product includes

Login and logout functionality, where an inventory manager can log in as "admin" and no passwords are required

Customer functionality, including searching for products, adding products to their shopping cart, and checking out Inventory management, which allows inventory managers to add, remove, and update products in the inventory Data persistence, so that the inventory, users, and user shopping carts are saved.

10% feature - An additional feature implemented on top of the MVP.

MVP Features

Epics

Product Lists

List Products on Customer Product Page Product Details

Inventory Management

Add to Inventory List Inventory Delete from Inventory Edit inventory item

Shopping Cart

Display Cart Add to Cart Remove from Cart Change Quantity Confirm Cart Cart Persistence

Login

Logout Login as User Login as E-Store Admin

Product API

Create New Product Get a single product Get Entire Inventory Search Product Update Product Delete a Single Product

User API

Create a New User Get a Single User Update User Delete User Get All Users

Registration Page

Entering and Creating User Enter Personal Details

User Profile

View User Profile Edit User Profile

Role API

Create a New Role Get All Roles Delete Role

Spring Security- Login API and User Roles

Login API Admin Registration Roles User Registration Roles Unregistered User Roles

Product Search Component

Search Food Items by Name Search Food Items by Price Search Food Items by Caloric Range

Sorting and Filters

Sort by Calories Sort by Price

Customer Checkout

Checkout from Shopping Cart List Shopping Cart Contents on Checkout Page Purchase Options Direct Checkout from Product Details

Make Your Own Salad

Add Ingredients Remove Ingredients Calorie Adder

Roadmap of Enhancements

Sprint 1

For the first sprint, we focused on getting the backend functional for the MVP. We worked on the backend for inventory management and product listing. It mainly consisted of working on the Product model, CRUD operations in the Controller and FileDAO and setting up the API endpoints. The end goal was to have a functioning backend for the MVP that was able to store and retrieve product data The User Stories and Enhancements we worked on are listed below.

- Create a New Product
- Get a Single Product
- Get All Products
- Update a Product
- Delete a Product
- Search for a Product

Sprint 2

We completed majority of our user stories and completed our MVP including the frontend and backend in this sprint. We worked on the frontend for inventory management for the Admin and product listing for the Customer. We also added the Shopping Cart functionality with persistence. Additionally, we added password hashing and JWT authentication using Spring Security to make our API secure and only accessible by the role assigned to the user logged in. The User Stories and Enhancements we worked on are listed below.

Admin Page

- Login as Admin

- Add to Inventory
- List Inventory
- Edit Inventory Item
- Delete from Inventory

E-Store Page

- Login as Customer
- View Products
- Search Product by Name
- View Product Details
- Add to Cart
- View Cart
- Edit Cart Item Quantity
- Remove from Cart
- Checkout

Backend

- Shopping Cart Model & Controller
- Register as Customer
- Password Hashing
- JWT Authentication
- Login API
- Role API

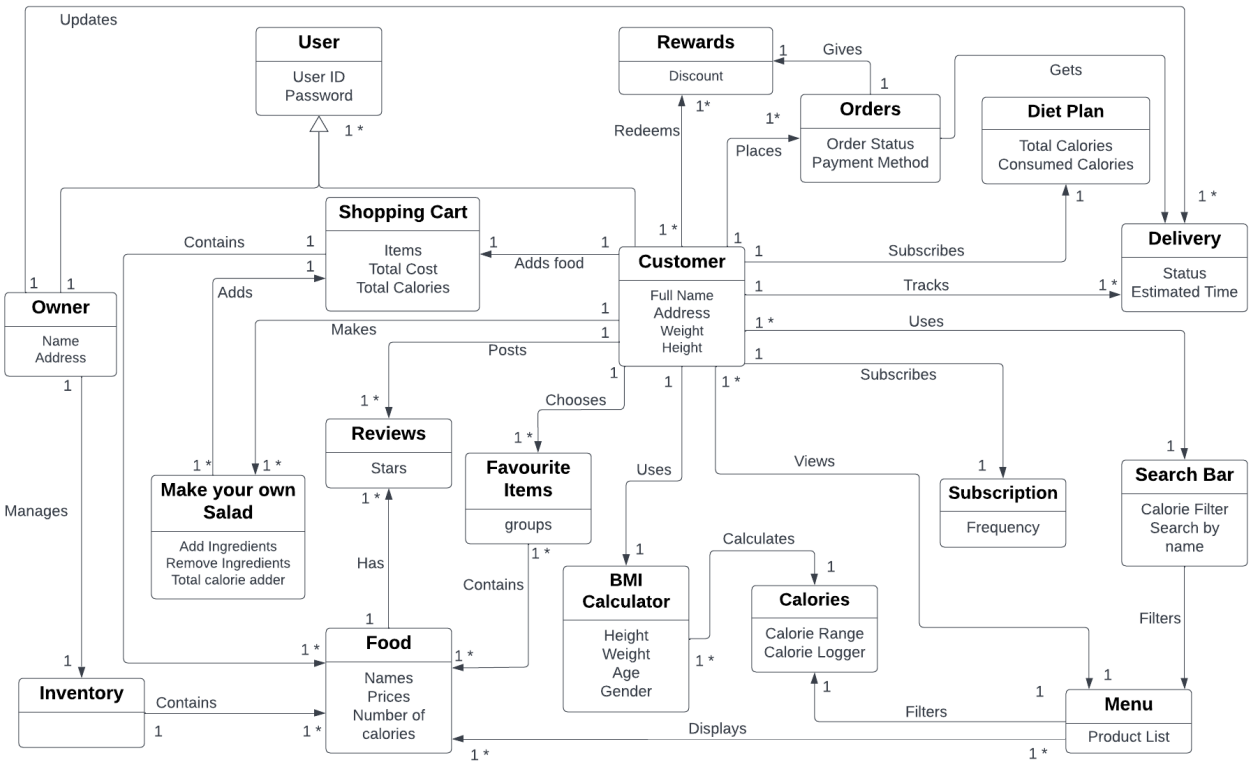
Sprint 3

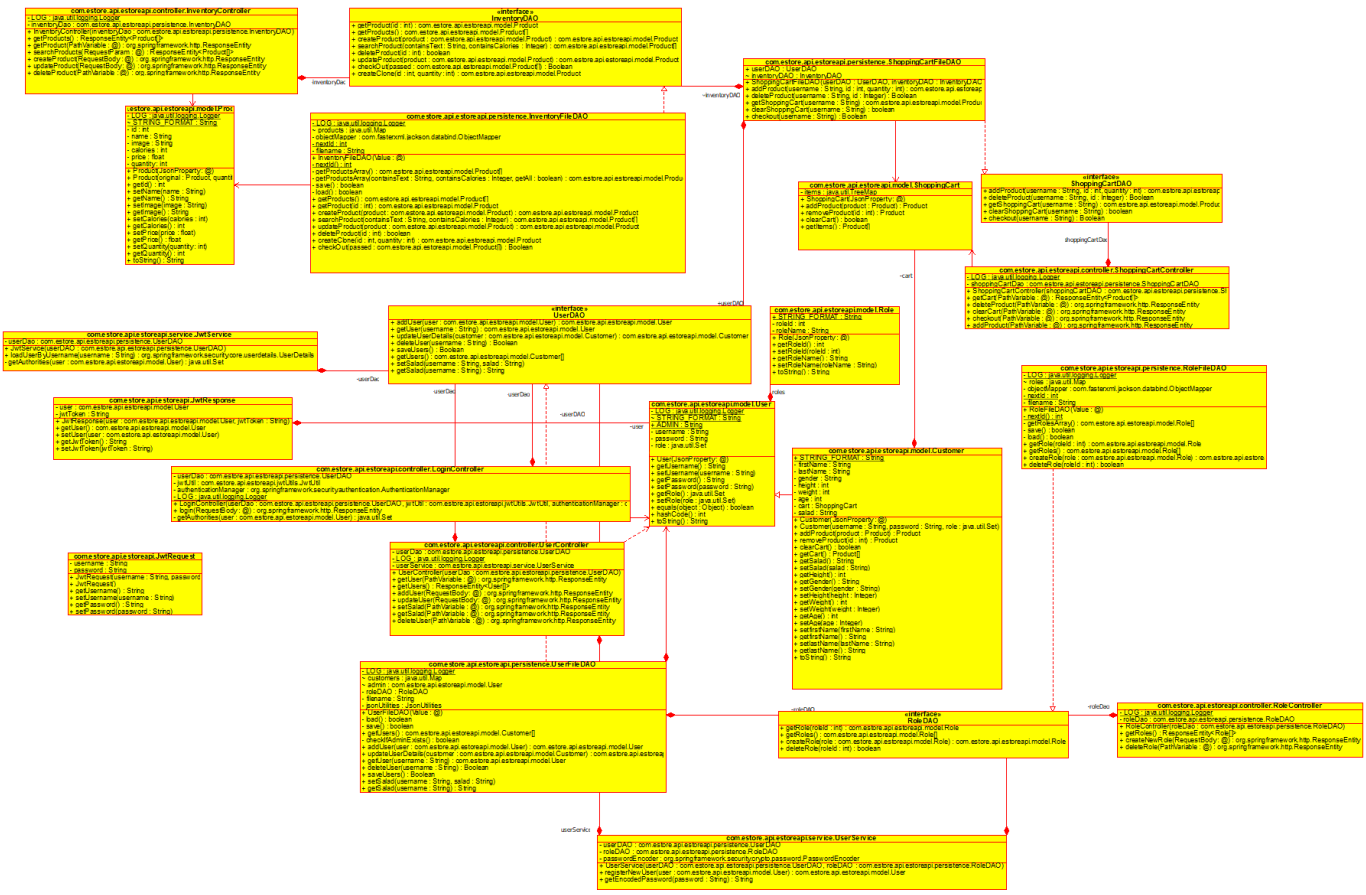
For the final sprint, we worked on our 10% features, namely the custom Salad Maker and BMI Calculator, as well as sorting and range filters for calories and price. We also added the ability to edit your User Profile and added a checkout page for the Customer to enter their details. Additionally, we worked on the user experience and overall design of our page and made it a lot more user friendly. We also added error messages to let the user know when something goes wrong. The User Stories and Enhancements we worked on are listed below.

- Custom Salad Maker
- BMI Calculator
- Sort by Calories
- Sort by Price
- Filter by Calories
- Filter by Price
- Edit User Profile
- Checkout Page
- Design and UX
- Error Messages

Application Domain

CALORIFY





There is a parent entity which represents the user, is then extended into two based on privileges of the role: 1) admin/owner 2) customer. Another crucial entity is the product/food that is being sold for purchase by a customer. The products being sold are kept inside inventory. The admin is able to modify the contents of this inventory. A customer has access to the menu which displays all the products present in the inventory. The admin is the only one allowed to add, remove, and update products in the inventory. The customer can also

search for products by name, price, and calories. If the customer wants to filter for a specific product or type of product, they use the Search bar. They can also use search filters to specify and narrow down the price or calories they want to intake. They can also sort the menu by price or calories.

The customer can add products from inventory to their shopping cart, which can be described as an attribute of User and the shopping cart is persistent across login sessions. The customer can checkout from shopping cart to place an order. For checking out, the customer has to enter an address and card details. The items are then subtracted from inventory. The customer can use ingredients in the inventory to make their own salad which will be a custom product created by the user that will be placed in the shopping cart and can be purchased in checkout. Finally the customers personal details can be usd to calculate BMI and the recommended calories. This can be used by the user to filter products based their suggested calorie intake.

Architecture and Design

The Estore-UI is the graphical user interface front end for our application. It is made up of HTML, CSS, and TypeScript code and uses the Angular framework. The Estore-UI communicates with the Estore-API to fetch data and to send data back to the backend for processing.

The Estore-API is the back end for our application. It is made up of Java code and uses the Spring framework. The Estore-API receives HTTP requests from the Estore-UI and responds with HTTP answers. The Estore-API uses I/O to communicate with the storage for read and write. JSON files constitute the storage and format of requests and response bodies

Our application consists of three main components: the model, the controller and the persistence.

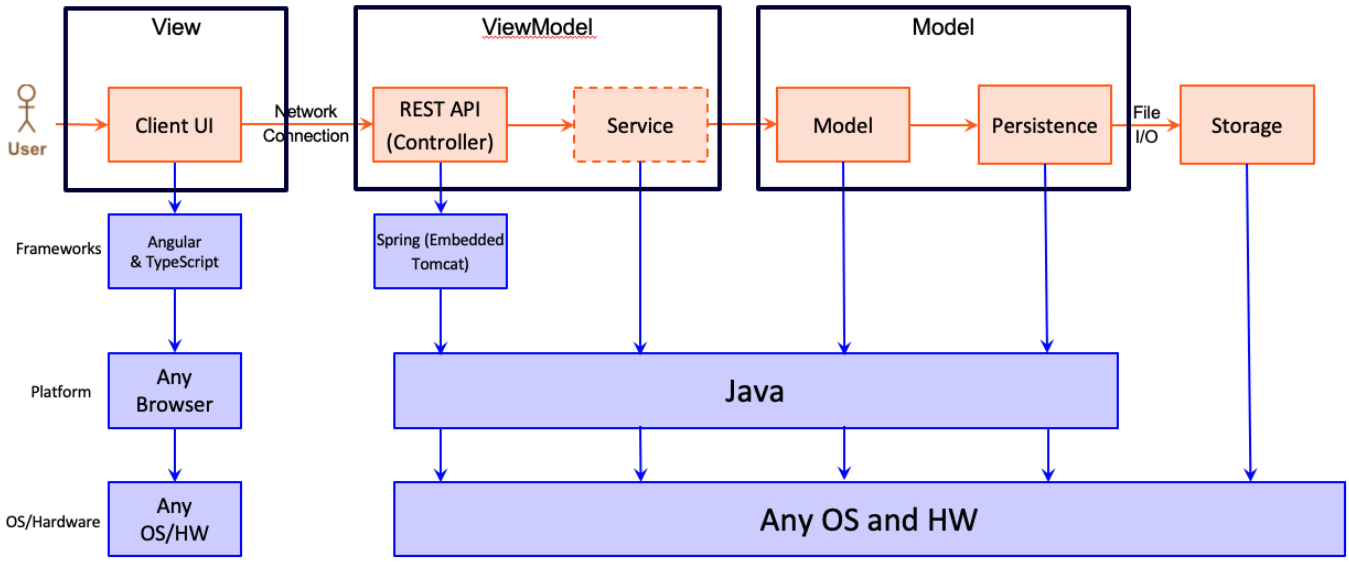
The model is the part of the application representing the data layer of our application with classes. The model is the format in which information may be saved in a persistent file. For example, a User that stores information about users in the form of Object

The controller is responsible for handling user requests and responses. It is what will call the Data Access Object when appropriate to retrieve and add information and perform logic on them. The controller is written in Java and makes use of the Spring framework.

The services are responsible for business logic.

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.

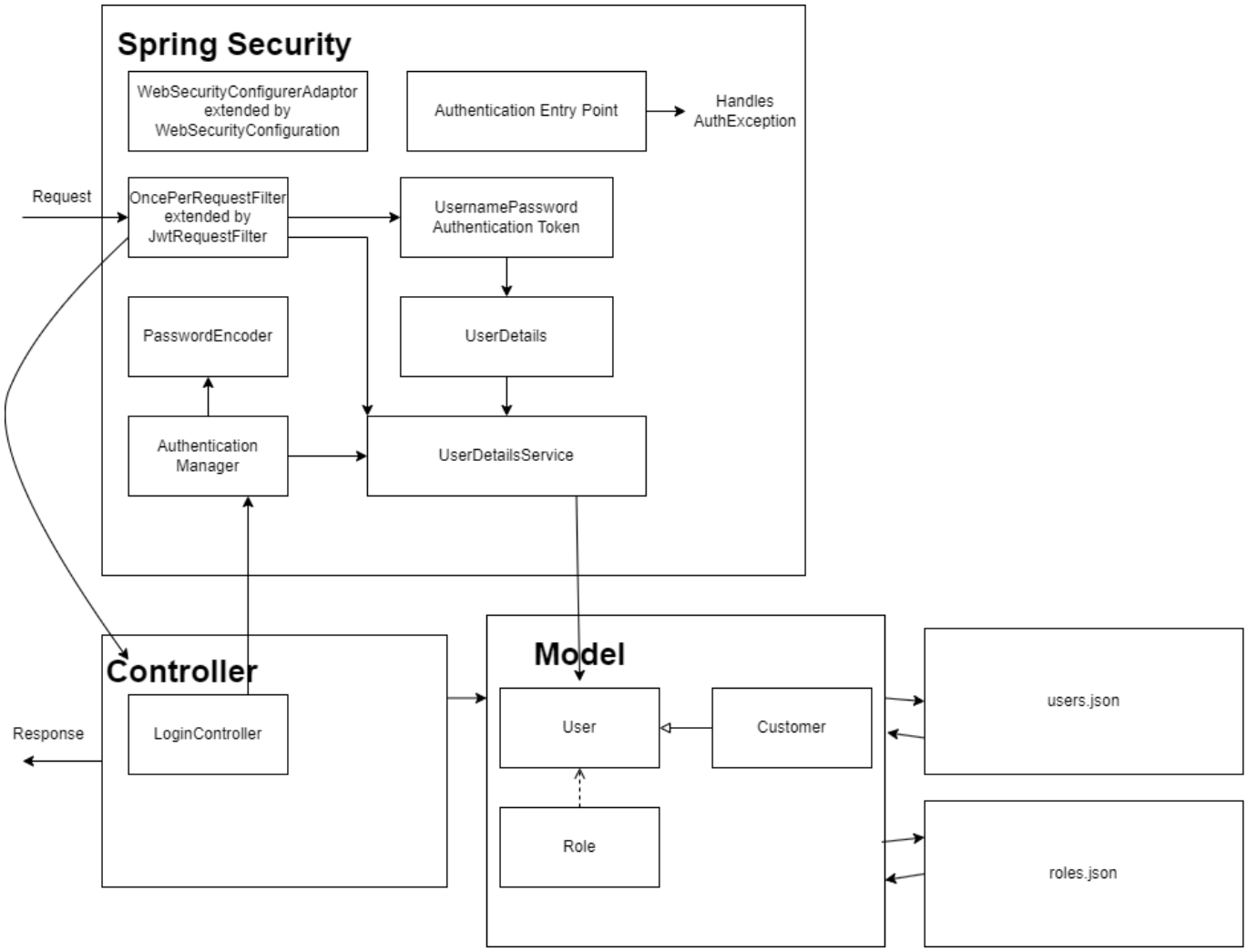
The Model stores the application data objects including any functionality to provide persistance.

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.

The Controller is written in Java and provides the logic for the application as well as the communication layer between the ViewModel and the Model.

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

Spring Security Architecture



Overview of User Interface

When a user first launches the application, they see the login page, where they are given the option to either login or to sign up. From the sign up option, the user is taken to a page which prompts for a username. The user can then either select to create their account, or to cancel. Cancel takes the user back to the login page. Create account will create a new account for the user with the given username, log them in, and bring them to the browse page unless there is already an account with that username, in which case, the user will receive a message alerting them of this issue. Now, when the user is on the login page, enters an existing username that is not “admin,” and selects the login option, they are logged into the application and are taken to the browse page, where they see the inventory of products below a menu bar, with the options of login, browse, and shopping cart, along with a search bar where they can search for products whose names contain the entered string. If they select a product from this browse page, they are taken to a page containing details of the product, which also gives them the option to add the item to their cart, and to personalize the item (i.e. add an engraving message, choose their varnish). If they select the shopping cart option from the menu, they are taken to a page which displays the items currently in their shopping cart. When the user is on the login page and enters the username “admin,” they are brought to the admin-browse page where they can view and add products to, or remove products from, the inventory. When they select a specific product, they are brought to the admin-product-detail page, where they can update the product’s information.

View Tier

The view tier is the user interface of the application. The View is the client-side SPA built with Angular utilizing HTML, CSS and TypeScript. It is responsible for displaying data to the user and for gathering input from the

user. The view tier consists of two types of components: UI components and presentation components. The UI components are responsible for generating the HTML that is sent to the client. They use the presentation components to get the data that they need to display. The presentation components are responsible for getting the data from the business tier and for formatting it for the UI components. The sequence diagram below shows a customer searching for an item and adding it to their cart. The customer starts by entering a search term into the search box. This sends an HTTP GET request to the server. The server then calls the business tier to get a list of items that match the search term. The business tier returns a list of items to the server. The server then calls the presentation tier to format the data for the UI components. The presentation tier returns the formatted data to the server. The server then calls the UI components to generate the HTML for the search results page. The UI components return the HTML to the server. The server then sends the HTML to the client. The client displays the HTML to the user.

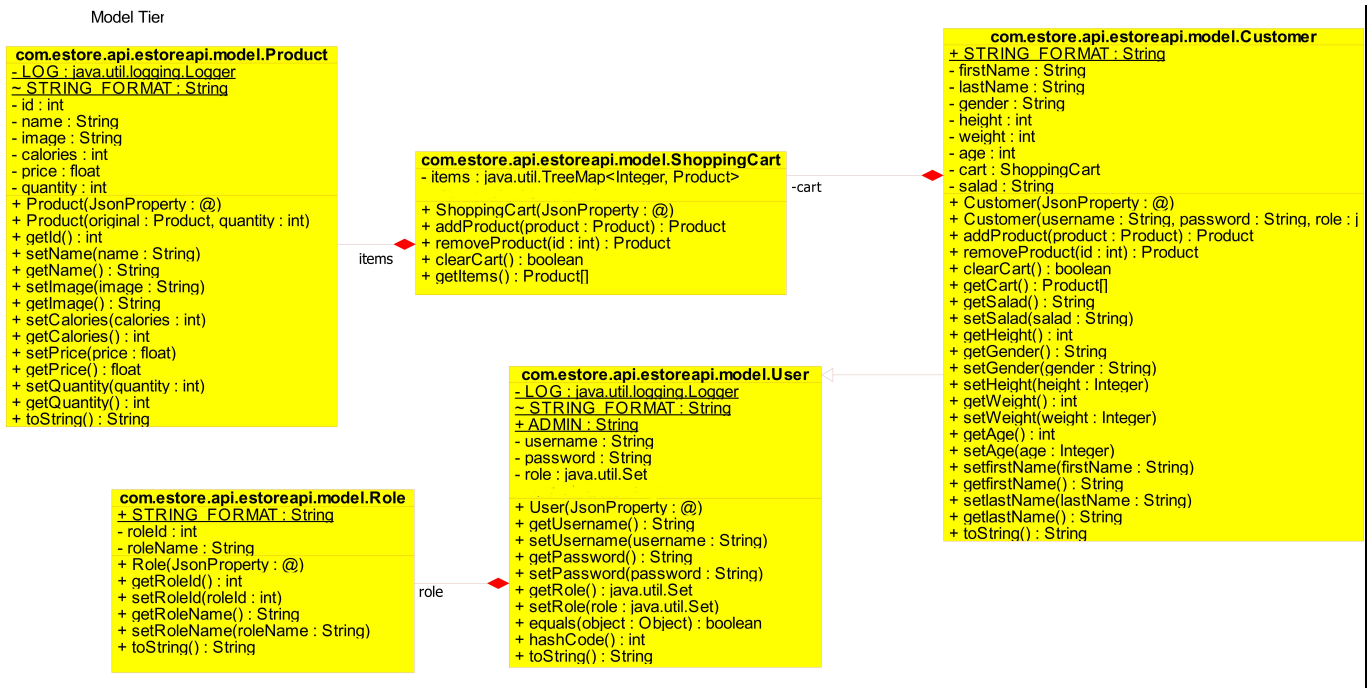
ViewModel Tier

The view model is the part of the application the user interacts with. The ViewModel provides RESTful APIs to the client (View) as well as any logic required to manipulate the data objects from the Model. It is made up of HTML, CSS, and TypeScript code and uses the Angular framework. The view model communicates with the controller to fetch data and to send data back to the backend for processing. View components are implemented as EJS templates which are rendered and served by the server. When a user navigates to the home page, the server will render and serve the home page template. The home page template contains static content as well as dynamic content that is populated by querying the database. For example, the home page template might contain a list of the most popular items on the site. This list is generated by querying the database and then passed to the home page template which renders the list in HTML.

The view tier of the e-store has the following components:

- Home Page
- Product Listing Page
- Product Details Page
- Shopping Cart Page
- Checkout Page
- Registration and Login Page

Model Tier

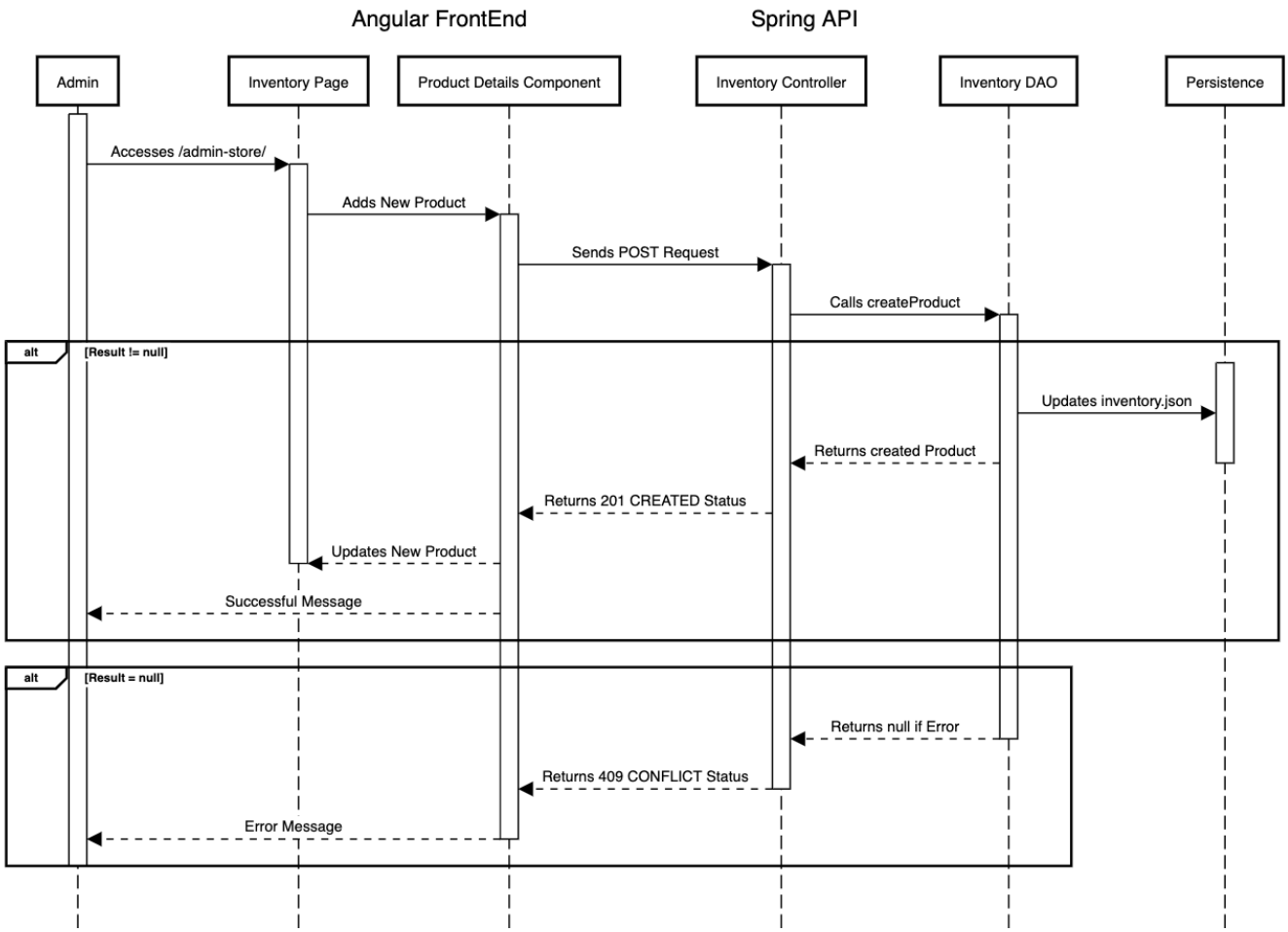


The model tier is responsible for storing the data for the application. The model tier consists of three types of components: data objects, data access objects, and business objects. The data objects are responsible for storing the data for the application. The data access objects are responsible for getting the data from the storage layer and for saving the data to the storage layer. The business objects are responsible for performing the business logic for the application. For example, a customer adding an item to their cart. The customer clicks on the add to cart button for an item. This sends an HTTP POST request to the server. The server then calls the business tier to add the item to the customer's cart. The business tier calls the data access tier to get the data for the item from the storage layer. The data access tier returns the data for the item to the business tier. The business tier then adds the item to the customer's cart. The business tier calls the data access tier to save the customer's cart to the storage layer. The data access tier saves the customer's cart to the storage layer.

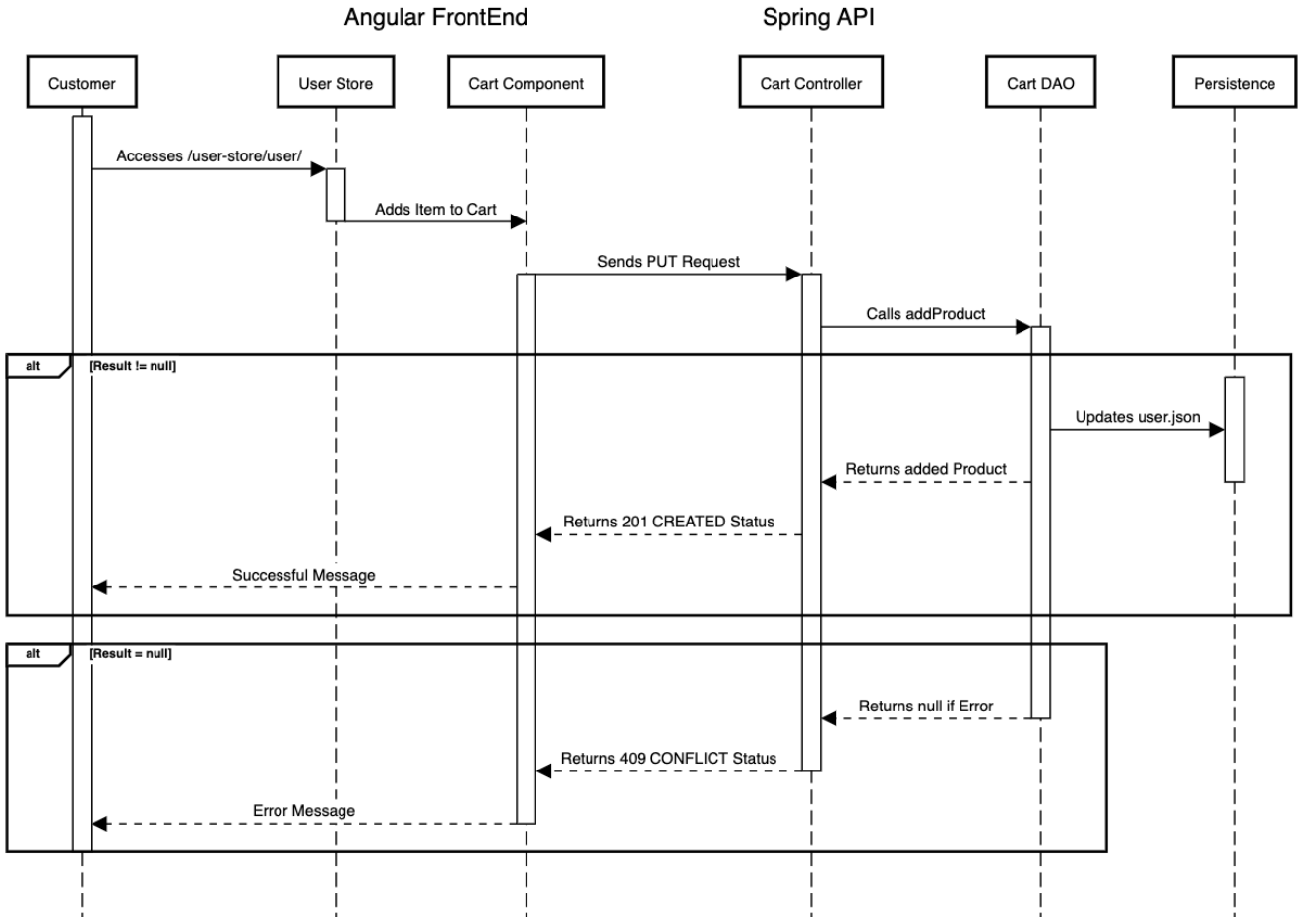
> *Provide a summary of this tier of your architecture. This section will follow the same instructions that are given for the View Tier above. At appropriate places as part of this narrative provide one or more static models (UML class diagrams) with some details such as critical attributes and methods.*

Sequence Diagrams

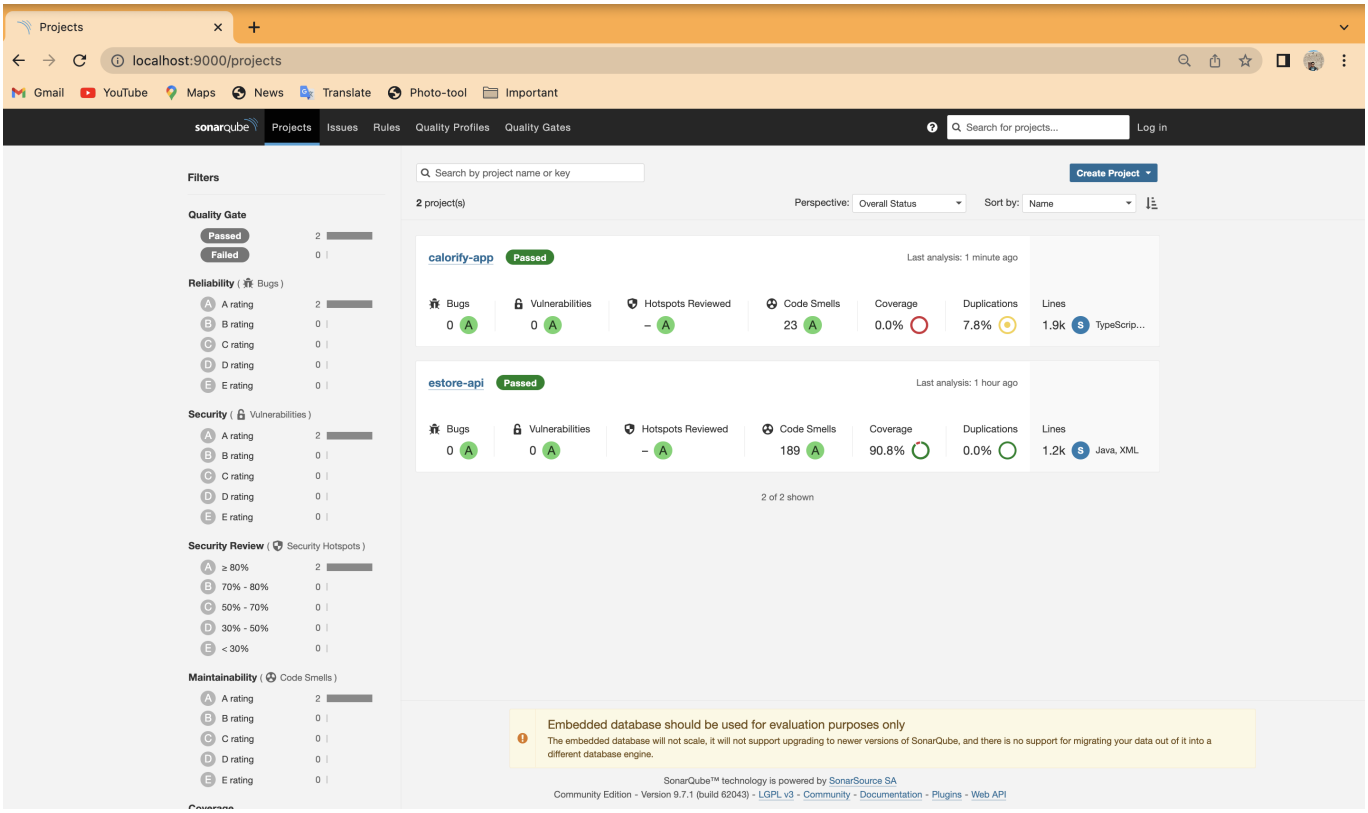
Create Product


Angular FrontEnd
Spring API
Admin
Inventory Page
Product Details Component
Inventory Controller
Inventory DAO
Persistence
Accesses /admin-store/
Adds New Product
Sends POST Request
Calls createProduct
alt
[Result != null]
Updates inventory.json
Returns created Product
Returns 201 CREATED Status
Updates New Product
Successful Message
alt
[Result = null]
Returns null if Error
Returns 409 CONFLICT Status
Error Message

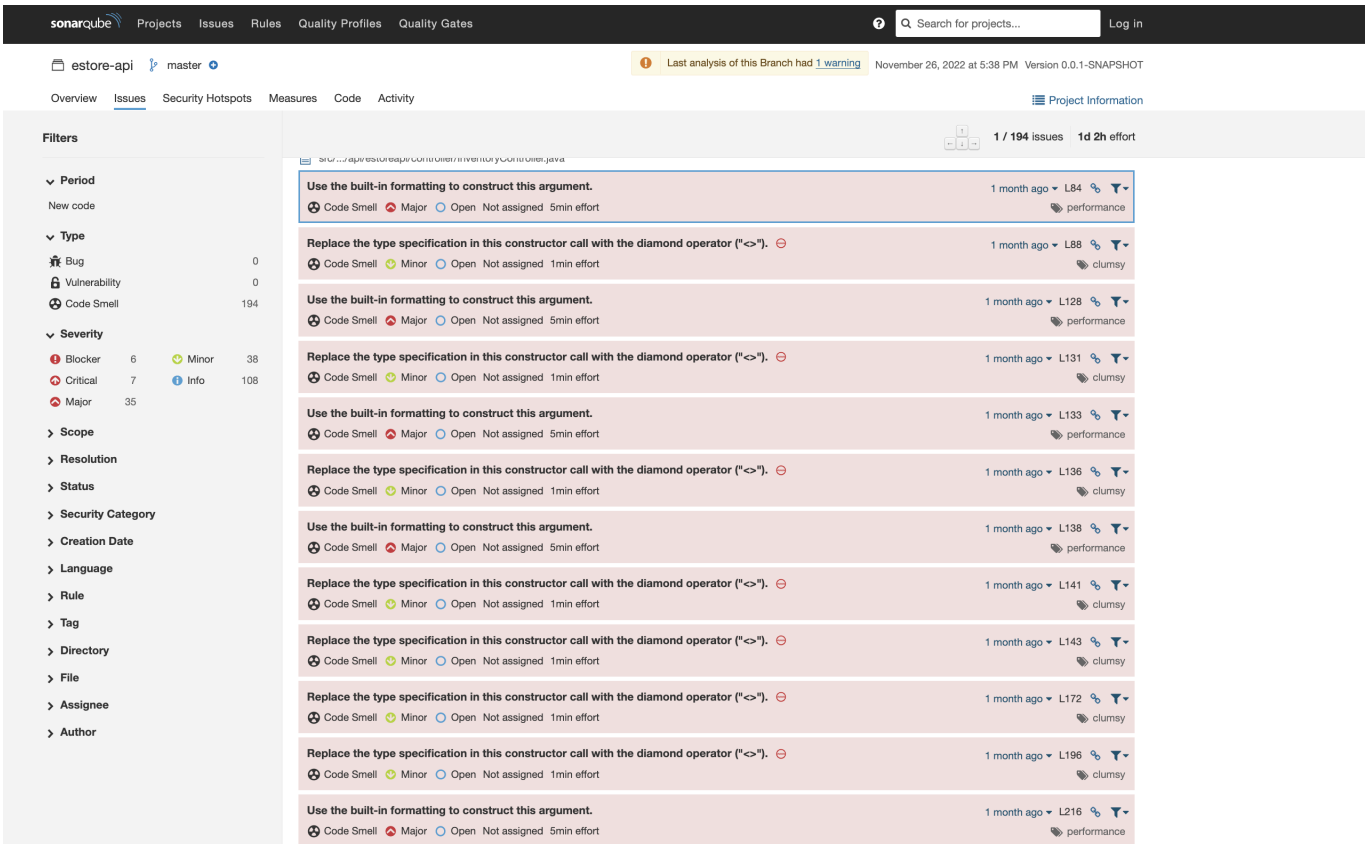
Add to Cart


Angular FrontEnd
Spring API
Customer
User Store
Cart Component
Cart Controller
Cart DAO
Persistence
Accesses /user-store/user/
Adds Item to Cart
Sends PUT Request
Calls addProduct
alt
[Result != null]
Updates user.json
Returns added Product
Returns 201 CREATED Status
Successful Message
alt
[Result = null]
Returns null if Error
Returns 409 CONFLICT Status
Error Message

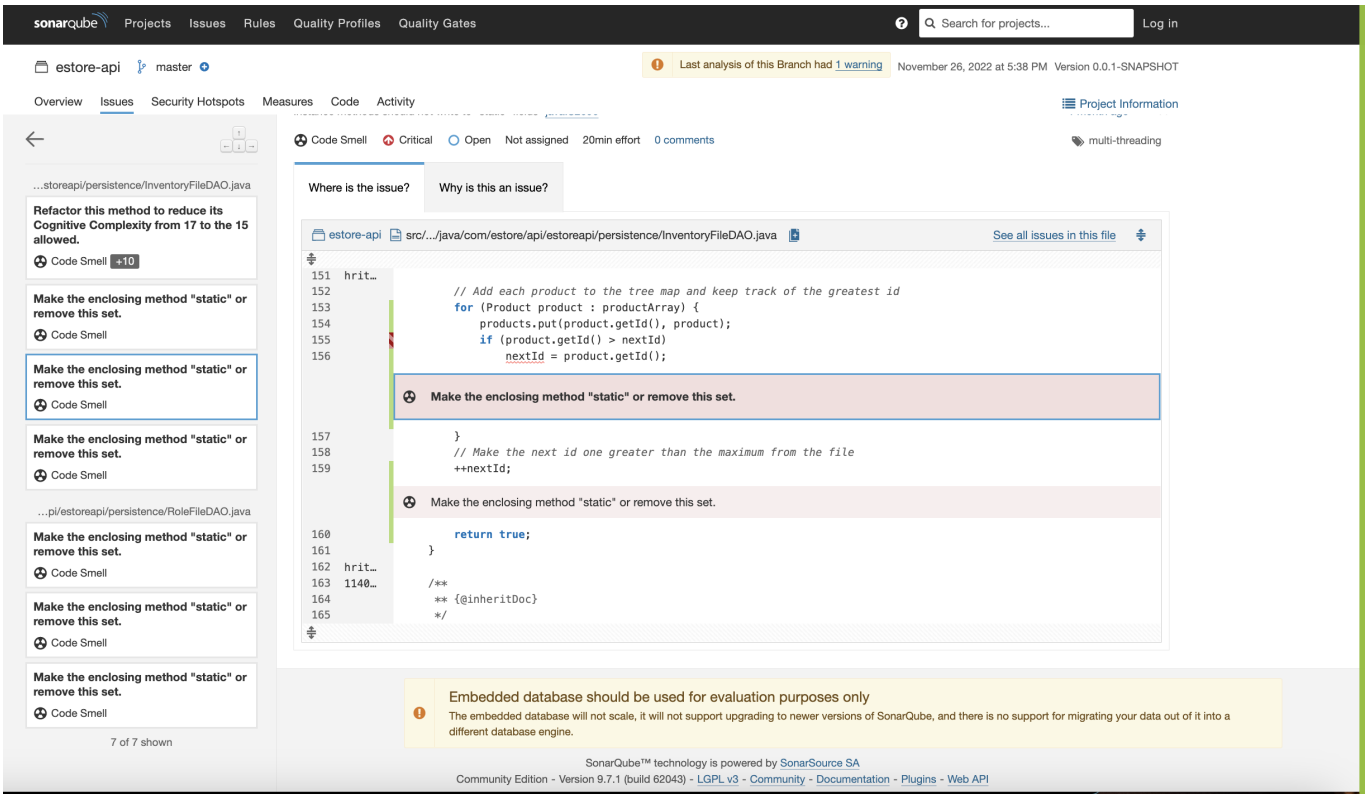
Static Code Analysis/Design Improvements



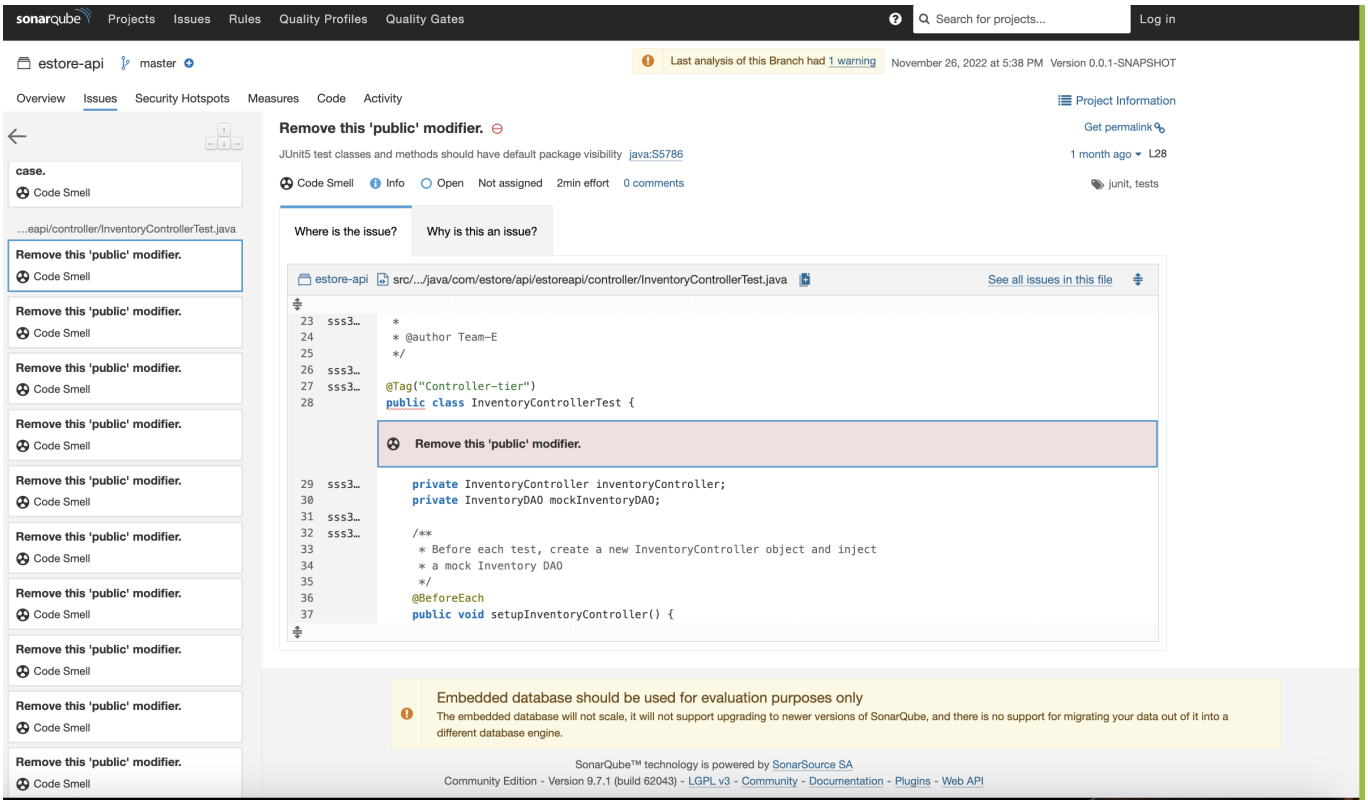
Static Code Analyis Issues - API



For the API, we had 194 issues shown with about an hour or two of estimated effort required to fix them. Almost all these issues are minor and repetitive and do not affect any functionality of our estore.

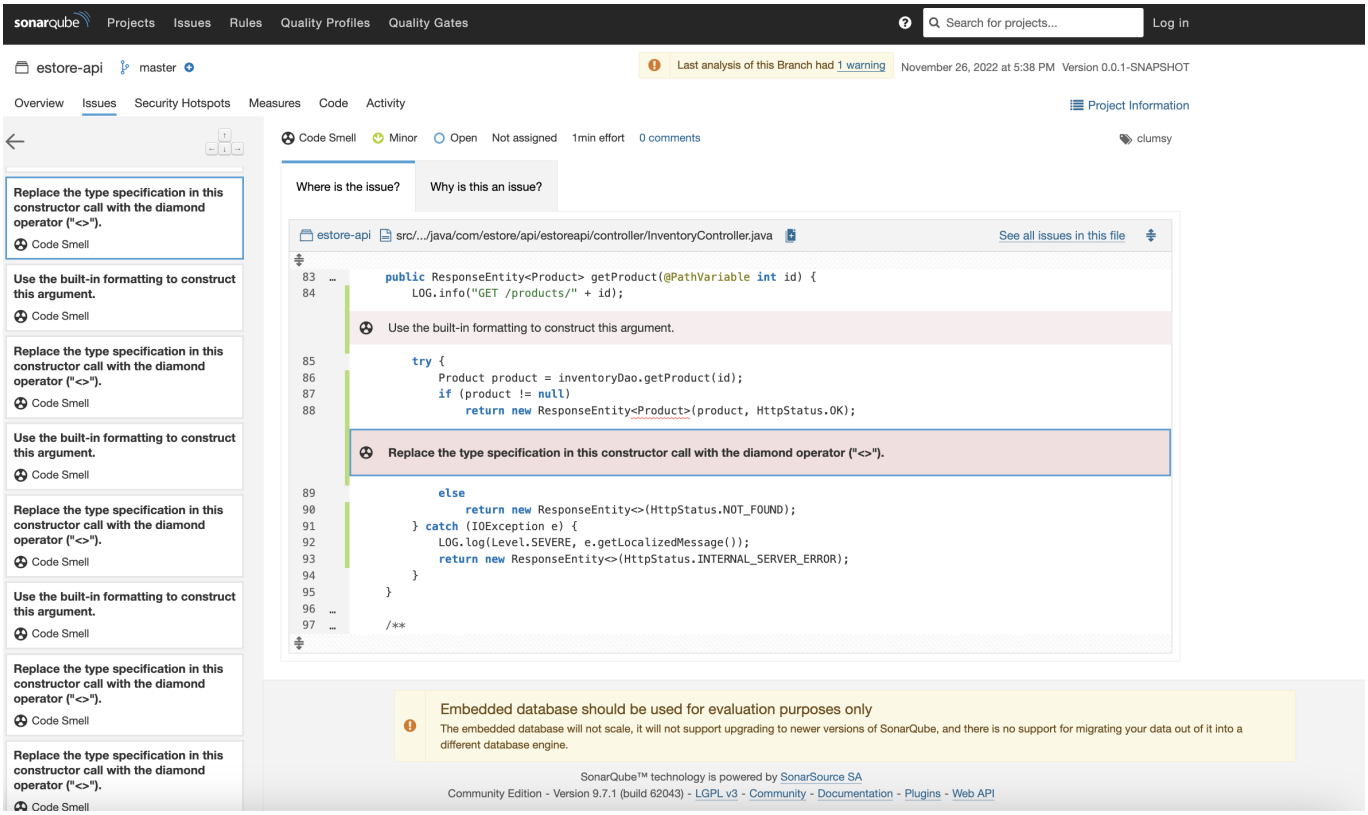


It can be challenging to update a static field correctly from a non-static method, and if there are numerous threads or instances of the class in use, errors are likely to result. Static fields should ideally only be updated by synchronized static methods. We have not made the methods static. We will have to include the static modifier in the method definition. This is an easy fix by just adding the static modifier to the method. We have about 6 instances of this and were marked as Critical.



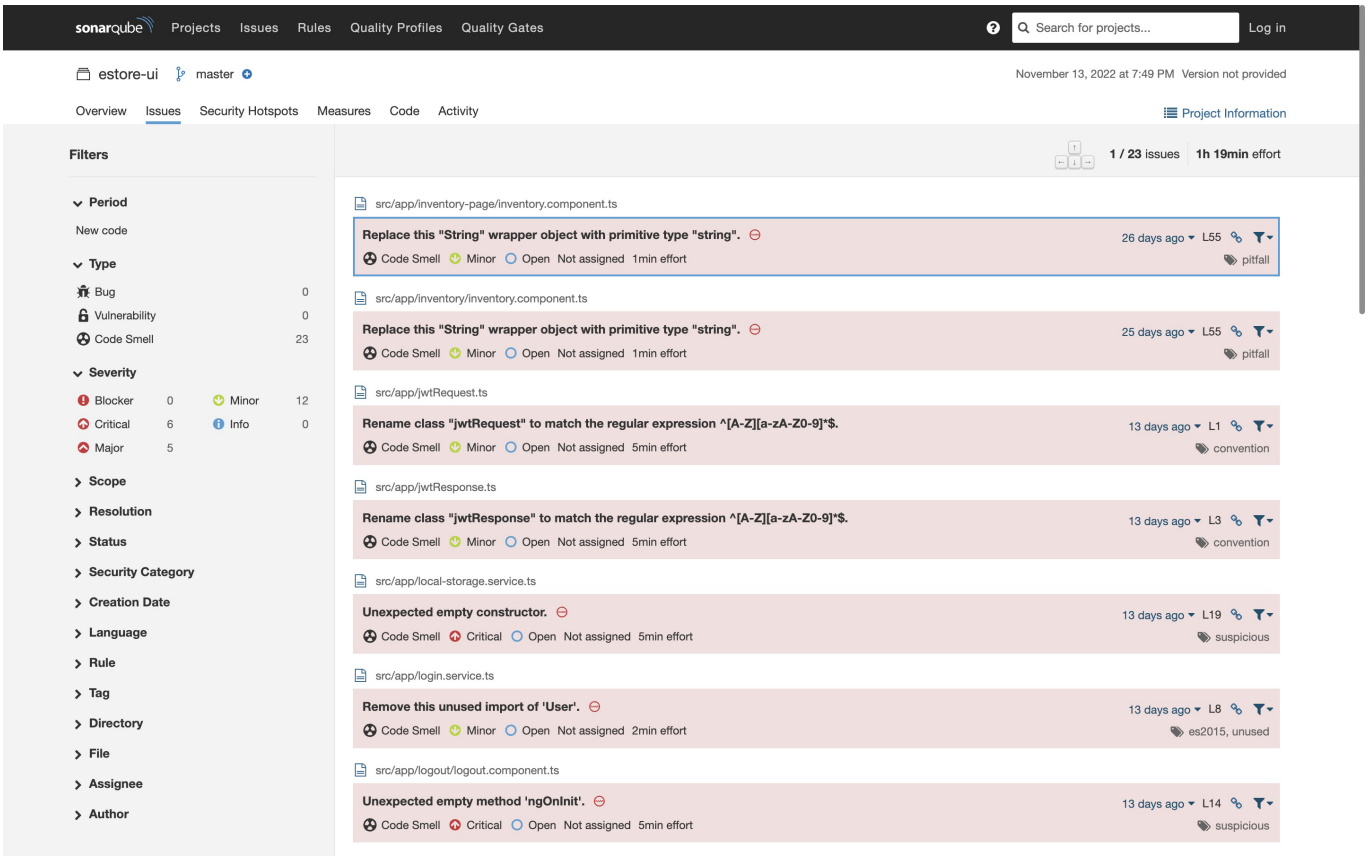
Although JUnit5 test classes can have any visibility other than private, using the default package visibility is better because it makes the code easier to read. We have specified the public modifier in the JUnit Test

Methods. However, this is redundant do not need to specify the public modifier for test methods. This is an easy fix by just removing the public modifier from the method. We have about 108 instances of this.

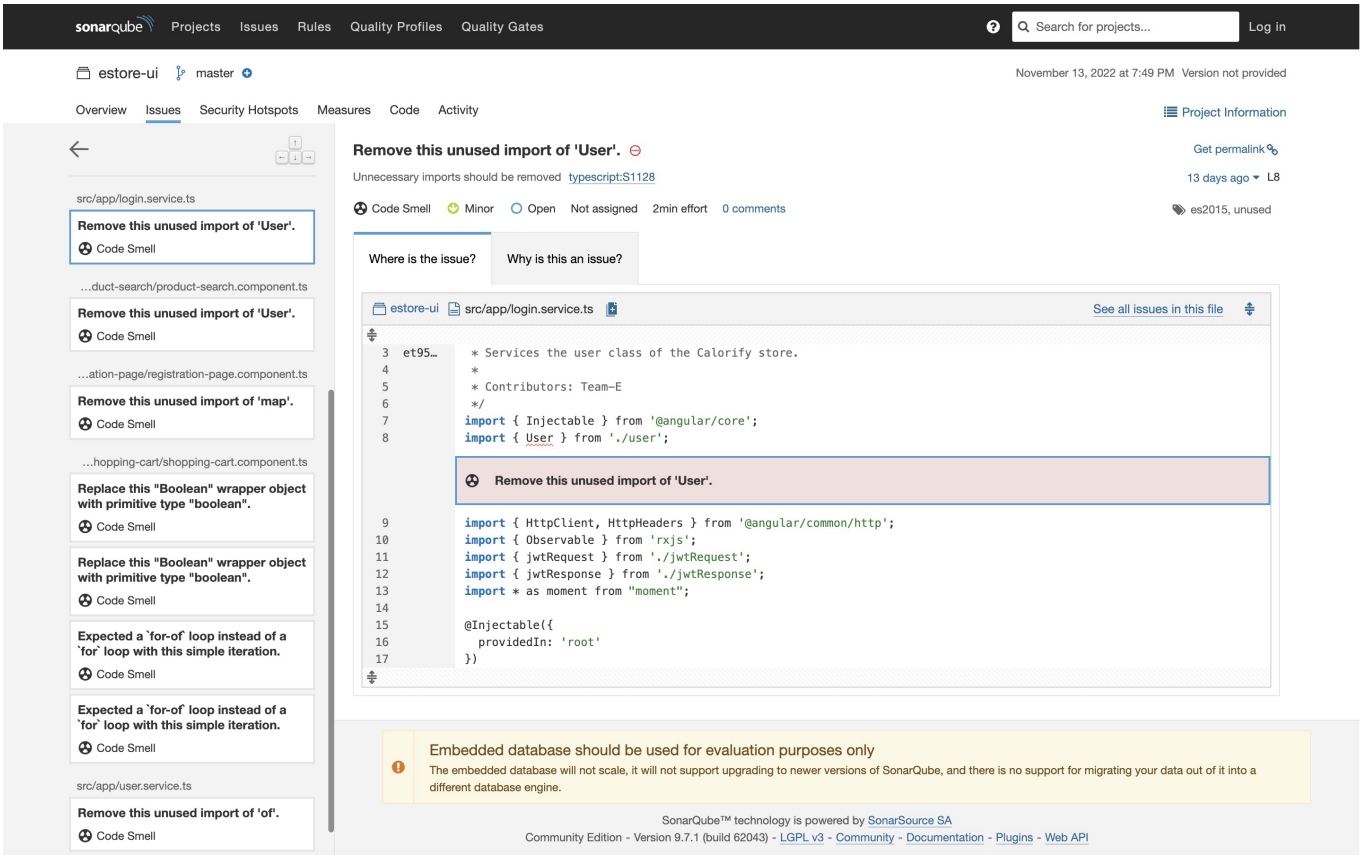


To make generics code less verbose, Java included the diamond operator (<>). For example, we can now simplify the constructor declaration with <>, and the compiler will infer the type, rather than having to state a List's type in both its declaration and its constructor. We have specified the Generic Type inside diamond operator in the return statement. However, this is redundant since we already defined it in the function definition. This is an easy fix by just removing the Generic Type from the diamond operator. We have about 24 instances of this.

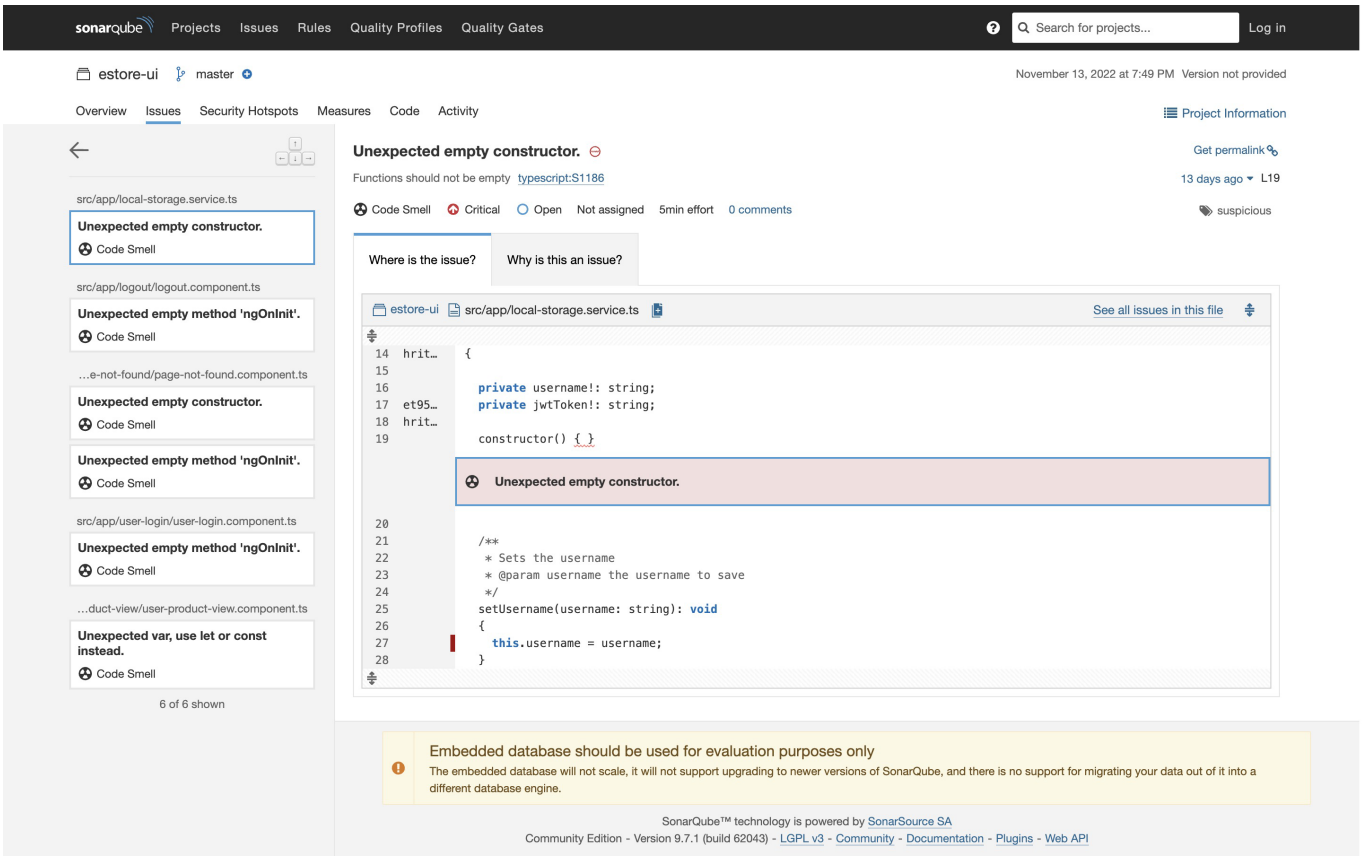
Static Code Analyis Issues - UI



For the UI, we had 23 issues shown with about an hour of estimated effort required to fix them. Almost all these issues are minor and do not affect any functionality of our estore.



We had 4 issues of unused imports, which were probably left there during us testing out different methods to implement UI components. The fix is simple, we just need to delete the lines that have these import statements.



We have 5 issues of empty constructors or methods, namely the ngOnInit and constructor methods in some components. This is another simple fix, where we just need to delete the lines that have these empty methods.

Recommendations for Design Improvements

- Order tracking history implemented in the backend
- Customer as well as Admin can access Placed Orders
- User Profile Details during Registration
- Change Shopping Cart API to store items by reference to ID to increase efficiency and reduce storage space
- Move Total Price and Quantity Calculation for Shopping Cart to Backend

Testing

Acceptance Testing

A total of 46 stories were tested and all of them are PASSING. None of the tested Stories are failing. We have a minor exception handling bug that will be fixed ASAP.

User Story	Acceptance Criterion	Sprint 2	Tester initials; date;	Sprint 3	Tester initials;
As a Customer I want to see a list of products so that I choose what to purchase.	GIVEN that I am on the e-store site WHEN I am not on the Products page THEN I must see a means to navigate to the Products page.	Pass	SBH; 11/4/22	Pass	SBH; 11/20/22
	GIVEN that I am not on the Products page WHEN I choose the Products page THEN I am taken to the Products page.	Pass	SBH; 11/4/22	Pass	SBH; 11/20/22
	GIVEN that I am on the Products page WHEN there are no products in the inventory I see a message indicating that that there are no products available.	Pass	SBH; 11/4/22	Pass	SBH; 11/20/22
	GIVEN that I am on the products page WHEN there are products in the inventory THEN I see each product and short description.	Pass	SBH; 11/4/22	Pass	SBH; 11/20/22
	GIVEN that I am on the products page WHEN there are products in the inventory THEN I see a means to add each product to my shopping cart.	Pass	SBH; 11/4/22	Pass	SBH; 11/20/22
As a customer, upon clicking the desired product in the product page, I should be able to see further details about the product so that I can finalize my decision	GIVEN product exists WHEN I select details about product THEN I expect to see the details of the selected product.	Pass	SBH; 11/4/22	Pass	SBH; 11/20/22
As a user I want to search for food items by name on Calorify product listing page so that I can view the specific food items i desire	GIVEN I want to filter the products by name WHEN I type a string into the search bar THEN I should be able to see products containing the string I entered.	Pass	CA; 11/4/22	Pass	CA; 11/20/22
As an E-Store Owner, I want to be able to add a new product to the store inventory after filling the required details so that I can introduce new products.	GIVEN I submit a product object WHEN no product with the given name exists in the inventory THEN the system should create the product, add it to the inventory, save to the persistent store, and return the product object and a status code of 201 (CREATED)	Pass	SS; 11/4/22	Pass	SS; 11/20/22
	GIVEN I submit a product object WHEN a product with the given name already exists in the inventory THEN the system should return a status code of 409 (CONFLICT)	Pass	SS; 11/4/22	Pass	SS; 11/20/22
As an E-Store Owner, I want to be able to see all products available in the current store system along with their properties so that I can decide what operation to perform.	GIVEN I submit a request for the inventory WHEN products exist in the inventory THEN the system should return a list of products in the inventory and a status code of 200 (OK)	Pass	SS; 11/4/22	Pass	SS; 11/20/22
	GIVEN I submit a request for the inventory WHEN no products exist in the inventory THEN the system should return an empty list of products and a status code of 200 (OK)	Pass	SS; 11/4/22	Pass	SS; 11/20/22
As an E-Store Owner, I want to be able to delete an existing product to from the store inventory after filling the required details so that I can introduce new products.	GIVEN I submit a product ID WHEN the product with the given ID exists in the inventory THEN the system should delete the product, remove it from the inventory, save to the persistent store, and return a status code of 200 (OK)	Pass	SS; 11/4/22	Pass	SS; 11/20/22
	GIVEN I submit a product ID WHEN a product with the given ID does not exist in the inventory THEN the system should return a status code of 404 (NOT FOUND)	Pass	SS; 11/4/22	Pass	SS; 11/20/22

As an E-Store Owner, I want to be able to edit existing product details so that I can update a products information.	GIVEN I submit a request for the inventory WHEN products exist in the inventory THEN the system should update the list of products as expected and return a confirmation and a status code of 200 (OK)	Pass	SS; 11/4/22	Pass	SS; 11/20/22
	GIVEN I submit a request for the inventory WHEN no products exist in the inventory THEN the system should return an empty list of products and a status code of 200 (OK)	Pass	SS; 11/4/22	Pass	SS; 11/20/22
As a Customer, I should be able to see what the current contents of my shopping cart are so that I can verify what is left to buy before checkout.	GIVEN cart is not empty WHEN I select to view items in my cart THEN I expect products in cart to displayed.	Pass	HM; 11/4/22	Pass	HM; 11/20/22
As a Customer, I want to be able to add my desired product to the shopping cart so that I can bring it to checkout for purchase later	GIVEN cart is empty WHEN I add product to cart THEN I expect product to show up in the cart.	Pass	HM; 11/4/22	Pass	HM; 11/20/22
As a Customer, I want to be able to remove a product from the shopping cart so I can remove those food items I no longer want to order.	GIVEN the cart is not empty WHEN I remove product from cart THEN I expect the product to not show up in the cart.	Pass	HM; 11/4/22	Pass	HM; 11/20/22
As a Customer I want to select the quantity of a food item so I can add multiple items at once to my shopping cart.	GIVEN I have a product in cart WHEN I increase or decrease the quantity of the product THEN I expect product's quantity to be updated in the cart.	Pass	HM; 11/4/22	Pass	HM; 11/20/22
As a user I want to confirm my shopping cart so that I can proceed to checkout.	GIVEN cart is not empty WHEN I select checkout THEN I expect my order to be confirmed and processed.	Pass	HM; 11/4/22	Pass	HM; 11/20/22
As a customer, items in my shopping cart should be preserved even after I logout so that I can continue shopping WHEN I log back in.	GIVEN cart is not empty WHEN I logout THEN later login THEN I expect products to still remain in the cart.	Pass	HM; 11/4/22	Pass	HM; 11/20/22
As a Customer I want to log in as a user so that I can make purchases.	GIVEN that I already have an account on the website, WHEN I enter my username password, THEN I expect to be able to access the user store page.	Pass	ET; 11/4/22	Pass	ET; 11/20/22
	GIVEN that I already have an account on the website, WHEN I enter my username, THEN I expect to be able to access my shopping cart	Pass	ET; 11/4/22	Pass	ET; 11/20/22
As an E-store Owner I want to log in as admin so I can manage the inventory.	GIVEN that I use admin as my username and password WHEN I login, THEN I expect to be able to access the inventory page.	Pass	ET; 11/4/22	Pass	ET; 11/20/22
As a user I want to register myself on Calorify so that I can product listing and select products to checkout	GIVEN I am in registration page WHEN I enter my details and register THEN I should have an account created with role user	Pass	CA; 11/4/22	Pass	CA; 11/20/22
	GIVEN I am a registered user with role user WHEN I access the product listing page THEN I should be able to add products to shopping cart/buy products	Pass	CA; 11/4/22	Pass	CA; 11/20/22
	GIVEN I am a registered user with role user WHEN I access the landing page THEN I should be able to access Calorify features such as BMI Calculator	Pass	CA; 11/4/22	Pass	CA; 11/20/22

	GIVEN I am a registered user with role user WHEN I access the landing page THEN I should be able to access Calorify features such as BMI Calculator	Pass	CA; 11/4/22	Pass	CA; 11/20/22
	GIVEN I am a registered user WHEN I try to access products listing page THEN I should be able to do so with no prompts	Pass	CA; 11/4/22	Pass	CA; 11/20/22
As an admin I want to register myself on Calorify so that I can update the product inventory, add users, create roles based on business needs	GIVEN I am in registration page WHEN I enter my details and register THEN I should have an account created with role Owner WHEN I log into this account	Pass	CA; 11/4/22	Pass	CA; 11/20/22
	GIVEN I am a user with role admin WHEN I access the landing page THEN I should be able to access the Inventory Management page where I add, remove, update products	Pass	CA; 11/4/22	Pass	CA; 11/20/22
	GIVEN I am a user with role admin WHEN I access any page on the webpage THEN I should be able to do so with no issue	Pass	CA; 11/4/22	Pass	CA; 11/20/22
	GIVEN I am in registration page WHEN I enter my details and register as an owner and another owner account already exists THEN I should not be able to create an account as only one owner is allowed	Pass	CA; 11/4/22	Pass	CA; 11/20/22
As an unregistered user, I should not be able to access product listing page, shopping cart and checkout functionality so that I must register/login to do so	GIVEN that I have not registered an account WHEN I look at the landing page THEN I shouldnt see pages accessible only by registered users	Pass	CA; 11/4/22	Pass	CA; 11/20/22
	GIVEN that I have not registered an account WHEN I try to access a Calorify feature THEN I should be prompted to register/login	Pass	CA; 11/4/22	Pass	CA; 11/20/22
	GIVEN that I have not registered an account WHEN I try to access products listing page THEN I should be prompted to register/login	Pass	CA; 11/4/22	Pass	CA; 11/20/22
	GIVEN that I have not registered an account WHEN I try to buy products in the product listing page THEN I should not be allowed purchase or add to shopping cart	Pass	CA; 11/4/22	Pass	CA; 11/20/22
As a developer, I want the user to be able to log in using login API endpoint with their username and password so that they can access calorify services	GIVEN I submit a request with username and password for an existing user of the system WHEN the password and username match what is in the system THEN the system should return the auTHENticated user and the jwt token for use for further requests with a status code of 200 (OK)	Pass	CA; 11/4/22	Pass	CA; 11/20/22
	GIVEN I submit a request with username and password for an existing user of the system WHEN the password and username don't match what is in the system THEN the system should return 401 (UNAUTHORIZED) response	Pass	CA; 11/4/22	Pass	CA; 11/20/22
	GIVEN I submit a request with username and password WHEN the user does not exist within the system THEN the system should return the status code of 404 (NOT FOUND)	Pass	CA; 11/4/22	Pass	CA; 11/20/22
AS an admin I WANT to submit a request to create a new Role (RoleName) SO THAT it is can be used for role based access to pages	GIVEN I submit a Role object WHEN no Role with the given roleName exists THEN the system should create the Role, add it to the role file, save to the persistent store, and return the user object and a status code of 201 (CREATED)	Pass	CA; 11/4/22	Pass	CA; 11/20/22
	GIVEN I submit a Role object WHEN a Role with the given roleName already		CA; 11/4/22		CA; 11/20/22

	GIVEN I submit a Role object WHEN a Role with the given roleName already exists THEN the system should return a status code of 409 (CONFLICT)	Pass	CA; 11/4/22	Pass	CA; 11/20/22	
AS a Developer I WANT to submit a request to get all roles SO THAT I have knowledge of all roles present in the system.	GIVEN I submit a request for all roles WHEN roles exist in the role storage THEN the system should return a list of all role with a status code of 200 (OK)	Pass	CA; 11/4/22	Pass	CA; 11/20/22	
	GIVEN I submit a request for all roles WHEN no roles exist in the role storage THEN the system should return an empty list of roles and a status code of 200 (OK)	Pass	CA; 11/4/22	Pass	CA; 11/20/22	
AS a Developer I want to submit a request to delete a role SO THAT it is no longer in the role storage file	GIVEN I submit a role ID WHEN the role with the role ID exists THEN the system should delete the role, remove it from storage, save, and return a status code of 200 (OK)	Pass	CA; 11/4/22	Pass	CA; 11/20/22	
	GIVEN I submit a role ID WHEN a role with the given ID does not exist THEN the system should return a status code of 404 (NOT FOUND)	Pass	CA; 11/4/22	Pass	CA; 11/20/22	
AS a Developer I WANT to submit a request to create a new user (username, password, role) SO THAT it is available to be used for registration.	GIVEN I submit a User object WHEN no User with the given username exists THEN the system should create the user, add it to the userData file, save to the persistent store, and return the user object and a status code of 201 (CREATED)	Pass	CA; 11/4/22	Pass	CA; 11/20/22	
	GIVEN I submit a User object WHEN a User with the given username already exists THEN the system should return a status code of 409 (CONFLICT)	Pass	CA; 11/4/22	Pass	CA; 11/20/22	
AS a Developer I WANT to submit a request to get a single user SO THAT I can access user details such as first name and last name	GIVEN I submit a request for a single user by id WHEN user exists in the user storage THEN the system should return the user and a status code of 200 (OK)	Pass	CA; 11/4/22	Pass	CA; 11/20/22	
	GIVEN I submit a request for the single user by id WHEN the product does not exist in the user storage THEN the system should return a status code of 404 (NOT FOUND)	Pass	CA; 11/4/22	Pass	CA; 11/20/22	
AS a Developer I WANT to submit a request to update user details SO THAT user detail changes are stored	GIVEN I submit a request for updating user with details of a specified id WHEN user exist in the user storage THEN the system should update details of user in the user storage and return the updated user with a status code of 200 (OK)	Pass	CA; 11/4/22	Pass	CA; 11/20/22	
	GIVEN I submit a request for updating user with details of a specified id WHEN no user exist in the user storage THEN the system should 404 error(NOT_FOUND)	Pass	CA; 11/4/22	Pass	CA; 11/20/22	
As a Developer I want to submit a request to delete a user SO THAT it is no longer in the user storage file.	GIVEN I submit a user ID WHEN the user with the user ID exists THEN the system should delete the user, remove it from the user storage, save, and return a status code of 200 (OK)	Pass	CA; 11/4/22	Pass	CA; 11/20/22	
	GIVEN I submit a user ID WHEN a user with the given ID does not exist THEN the system should return a status code of 404 (NOT FOUND)	Pass	CA; 11/4/22	Pass	CA; 11/20/22	
AS a Developer I WANT to submit a request to	GIVEN I submit a request for all users WHEN users exist in the user storage THEN the system should return a list of all users with a status code of 200 (OK)	Pass	CA; 11/4/22	Pass	CA; 11/20/22	

	200 (OK)				
As a user I want to be able to create an account by entering my username and password on the registration page on Calorify so that I can access the websites services	GIVEN that I am in the registration page WHEN I enter the username and password and click create THEN a new account should be created with that username	Pass	CA; 11/4/22	Pass	CA; 11/20/22
	GIVEN that I am in the registration page WHEN I enter invalid format for any of the fields THEN i should not be allowed to create user	Pass	CA; 11/4/22	Pass	CA; 11/20/22
	GIVEN that I am in the registration page WHEN I create an account THEN I should land in user profile page	Pass	CA; 11/4/22	Pass	CA; 11/20/22
	GIVEN that I am in the registration page WHEN I land in the user profile page THEN I should the details of user besides username and password should be null	Pass	CA; 11/4/22	Pass	CA; 11/20/22
As a user after registering my account with user name and password, I want to enter my personal details such as Full name, height, weight on the user profile page so that this information can be used in my profile and in Calorify services	GIVEN I am in user profile page WHEN I enter and update the user details i want THEN the user account details on my profile should be updated	Pass	CA; 11/4/22	Pass	CA; 11/20/22
As a user I want to be able to view my accounts details in the user profile page on Calorify so that I can verify what information Calorify is using	Given I enter the user profile page when the page loads then I should see my account details correctly			Pass	HM; 11/20/22
As a user I want to be able to edit my accounts and update details in the user profile page on Calorify so that I can update my account information	Given that I am in the personal detail input page when I enter personal details then it should submit and be saved for profile			Pass	HM; 11/20/22
As a user, I want to search for products from Calorify store based on a range of calories so that I may buy products based on my caloric needs	Given I am in product listing page when I enter a range of calories then I should see a list of products that are within that specified range			Pass	CA; 11/20/22
	Given I am in product listing page when I enter a range of calories then I should not see products that are outside that specified range			Pass	CA; 11/20/22
	Given I am in product listing page when I enter a specific calorie value then I should see a list of products that have calorie values less than or equal to specific calorie value			Pass	CA; 11/20/22
	Given I am in product listing page when I enter a specific calorie value then I should not see products that have calorie values greater than specific calorie value			Pass	CA; 11/20/22

As a user I want to sort food items from highest to lowest price or vice versa on Calorify product listing page so that I can easily view low or high price foods based on my spending budget	Given I am in product listing page when I select sort by lowest prices then I should see products listed with prices in ascending order(from lowest to highest)			Pass	CA; 11/20/22
	Given I am in product listing page when I select sort by highest prices then I should see products listed with prices in descending order(from highest to lowest)			Pass	CA; 11/20/22
As a user I want to filter the food items list based on categories so that I can get the list of foods suiting my preferences for types of foods	Given that I am in the product listing page when I check boxes of specified categories then I should see products of that category			Pass	SBH; 11/20/22
	Given that I am in the product listing page when I check boxes of specified categories then I should not see products outside of that category			Pass	SBH; 11/20/22
	Given that I am in the product listing page when I dont check any boxes of specified categories then I should see all products of all categories			Pass	SBH; 11/20/22
As a user I want to search for food items based on a price range on Calorify so that I can view the food items within my budget	Given I am in product listing page when I enter a specific price range then I should only see products within that price range			Pass	SS; 11/20/22
	Given I am in product listing page when I enter a specific price then I should only see products less than or equal to that value			Pass	SS; 11/20/22
	Given I am in product listing page when I enter a specific price range then I should not see products outside that range			Pass	SS; 11/20/22
	Given I am in product listing page when I enter a specific price then I should not see products greater than that value			Pass	SS; 11/20/22
As a user I want to sort food items by highest to lowest calories or vice versa on Calorify product listing page so that I can easily view low or high calorie foods based on my diet	Given I am in product listing page when I select sort by lowest calories then I should see products listed with calories in ascending order(from lowest to highest)			Pass	CA; 11/20/22
	Given I am in product listing page when I select sort by highest calories then				CA; 11/20/22

As a user, I should be able to input the range of calories in which I want food products from so that I can regulate my diet	Given I am in registration page when I enter my details and register then I should have an account created with role Owner when I log into this account			Pass	CA; 11/20/22
As a user I want to add ingredients to my salad bowl so that I can get that in my salad.	Given as a customer when I add ingredient to my salad then I expect the salad to have those added ingredients			Pass	HM; 11/20/22
As a user I want to remove ingredients from my salad bowl so that I cannot get that in my salad.	Given as a customer when I remove an ingredient to my salad then I expect the salad to have those ingredients removed.			Pass	HM; 11/20/22
As a customer, I should be able to go to checkout from shopping cart so that I can purchase what I have saved in the shopping cart	Given as a customer when I checkout my shopping cart then the system should checkout my cart successful			Pass	ET; 11/20/22
As a customer, I should be able to see what products I am about to purchase in the checkout page so that I can verify what I will buy	Given as a customer when I am on the checkout page then I expect the shopping cart contents on the checkout page.			Pass	ET; 11/20/22
As a customer, I should see whether my purchase has been successful so that I am notified that no error has occurred	Given something is in the cart when user checkouts out then the system should return a successful notification with no errors occured			Pass	SS; 11/20/22
As a user I want to enter my BMI details so my body mass index can be calculated and my calorie intake as well.	GIVEN I submit a put inputs to calculate WHEN i select the button to calculate THEN the system calculates and returns back the calculated BMI.			Pass	HM; 11/20/22
As a user I want to add all the calories of each ingredient so that I can see in total how many calories I intake from that salad bowl.	GIVEN as a user WHEN I add more ingredients THEN the system calculates and returns back the calculated total calories.			Pass	SBH; 11/20/22
As a user I want to enter my BMI details so my body mass index can be calculated and my calorie intake as well.	GIVEN I submit a put inputs to calculate WHEN i select the button to calculate THEN the system calculates and returns back the calculated BMI.			Pass	SS; 11/20/22
As a user I want to add all the calories of each ingredient so that I can see in total how many	GIVEN as a user WHEN I add more ingredients THEN the system calculates and returns back the calculated total calories.			Pass	SS; 11/20/22

Unit Testing and Code Coverage

Our unit testing strategy is to make sure that our tests are thorough, independent and readable. Thorough meaning that we are performing at a high code coverage - at least 90% overall. Independent meaning that we are testing one thing at a time. For example, we've decided that a unit is a class. We are testing each and every component of a class - its state and functions as well as those functions method arguments.

estore-api

estore-api

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.estore.api.estoreapi.persistence		93%		78%	15	63	13	159	1	31	0	3
com.estore.api.estoreapi.controller		94%		86%	6	51	11	174	1	29	0	4
com.estore.api.estoreapi.model		98%		78%	2	65	1	122	0	58	0	5
Total	95 of 1,966	95%	23 of 122	81%	23	179	25	455	2	118	0	12

We have achieved an overall code coverage of 95%, including 98% for the Model tier, 94% for the Controller Tier and 93% for the Persistence Tier.

Coverage: Controller Tier

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
UserController		78%		75%	2	13	11	46	1	9	0	1
InventoryController		95%		81%	4	19	4	64	0	8	0	1
ShoppingCartController		100%		100%	0	12	0	39	0	7	0	1
RoleController		100%		100%	0	7	0	25	0	5	0	1
Total	52 of 709	92%	6 of 44	86%	6	51	15	174	1	29	0	4

We have achieved 94% code coverage for the Controller Tier, which satisfies and exeeds the minimum 90% ideal requirement.

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

InventoryController

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
searchProducts(String, Integer)		87%		71%	4	8	4	22	0	1
createProduct(Product)		100%		100%	0	2	0	8	0	1
updateProduct(Product)		100%		100%	0	2	0	8	0	1
getProduct(int)		100%		100%	0	2	0	8	0	1
deleteProduct(int)		100%		100%	0	2	0	8	0	1
getProducts()		100%		n/a	0	1	0	6	0	1
InventoryController(InventoryDAO)		100%		n/a	0	1	0	3	0	1
static {...}		100%		n/a	0	1	0	1	0	1
Total	11 of 256	95%	4 of 22	81%	4	19	4	64	0	8

Analysis

For our Inventory Controller file, we have 95% coverage which is really good already. searchProducts is 87% instructions covered. All the other are 100% instructions covered. The ideal coverage for controller file should 90% and above so our controller satisfies the requirement Most of the missed instructions and branches are in searchProduct().

```
    @GetMapping("/")
    public ResponseEntity<Product[]> searchProducts(@RequestParam(required = false) String name,
            @RequestParam(required = false) Integer calories) {
        try {
            if (name != null && calories != null) {
                LOG.info("GET /products/?name=" + name + "&calories=" + calories);
                Product[] products = inventoryDao.searchProduct(name, calories);
                if (products.length != 0)
                    return new ResponseEntity<Product[]>(products, HttpStatus.OK);
            } else if (name != null) {
                LOG.info("GET /products/?name=" + name);
                Product[] products = inventoryDao.searchProduct(name, null);
                if (products.length != 0)
                    return new ResponseEntity<Product[]>(products, HttpStatus.OK);
            } else if (calories != null) {
                LOG.info("GET /products/?calories=" + calories);
                Product[] products = inventoryDao.searchProduct(null, calories);
                if (products.length != 0)
                    return new ResponseEntity<Product[]>(products, HttpStatus.OK);
            } else {
                return new ResponseEntity<Product[]>(HttpStatus.NOT_FOUND);
            }
        } catch (IOException e) {
            LOG.log(Level.SEVERE, e.getLocalizedMessage());
            return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
        }
        return null;
    }
```

As such, for proper coverage, we could write tests that cover a test that tests product.length in terms of whether, if product.length == 0, assert that the result is NOT_FOUND. We should also cover the code branches of when name != null or name =null in our tests. Along with this we can write tests to cover products != null or products = null

Coverage: Model Tier

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
User		92%		62%	2	15	1	25	0	11	0	1
Customer		100%		n/a	0	21	0	34	0	21	0	1
Product		100%		n/a	0	15	0	33	0	15	0	1
ShoppingCart		100%		100%	0	8	0	19	0	5	0	1
Role		100%		n/a	0	6	0	11	0	6	0	1
Total	6 of 465	98%	3 of 14	78%	2	65	1	122	0	58	0	5

We have achieved 98% code coverage for the Model Tier, which satisfies and exeeds the minimum 95% ideal requirement.

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

Customer

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
toString()		100%		n/a	0	1	0	1	0	1
Customer(String, String, Set)		100%		n/a	0	1	0	4	0	1
Customer(String, String, Set, ShoppingCart)		100%		n/a	0	1	0	4	0	1
addProduct(Product)		100%		n/a	0	1	0	1	0	1
removeProduct(int)		100%		n/a	0	1	0	1	0	1
setHeight(Integer)		100%		n/a	0	1	0	2	0	1
setWeight(Integer)		100%		n/a	0	1	0	2	0	1
setAge(Integer)		100%		n/a	0	1	0	2	0	1
clearCart()		100%		n/a	0	1	0	1	0	1
getCart()		100%		n/a	0	1	0	1	0	1
setSalad(String)		100%		n/a	0	1	0	2	0	1
setGender(String)		100%		n/a	0	1	0	2	0	1
setfirstName(String)		100%		n/a	0	1	0	2	0	1
setlastName(String)		100%		n/a	0	1	0	2	0	1
getSalad()		100%		n/a	0	1	0	1	0	1
getHeight()		100%		n/a	0	1	0	1	0	1
getGender()		100%		n/a	0	1	0	1	0	1
getWeight()		100%		n/a	0	1	0	1	0	1
getAge()		100%		n/a	0	1	0	1	0	1
getfirstName()		100%		n/a	0	1	0	1	0	1
getlastName()		100%		n/a	0	1	0	1	0	1
Total	0 of 135	100%	0 of 0	n/a	0	21	0	34	0	21

Analysis

It is very well tested file and all code is covered as we can above we achieved 100% code coverage

Coverage: Persistence Tier

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
RoleFileDAO		85%		71%	5	17	9	49	1	10	0	1
InventoryFileDAO		97%		82%	8	38	4	86	0	15	0	1
ShoppingCartFileDAO		98%		50%	2	8	0	24	0	6	0	1
Total	49 of 792	93%	14 of 64	78%	15	63	13	159	1	31	0	3

We have achieved 93% code coverage for the Persistence Tier, which satisfies and exeeds the minimum 90% ideal requirement.

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

InventoryFileDAO

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
checkOut(Product[])		93%		70%	3	6	2	12	0	1
getProductsArray(String, Integer, boolean)		94%		85%	3	11	1	18	0	1
createClone(int, int)		88%		50%	1	2	1	4	0	1
createProduct(Product)		100%		100%	0	3	0	11	0	1
load()		100%		75%	1	3	0	9	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
searchProduct(String, Integer)		100%		n/a	0	1	0	2	0	1
InventoryFileDAO(String, ObjectMapper)		100%		n/a	0	1	0	5	0	1
getProducts()		100%		n/a	0	1	0	2	0	1
nextId()		100%		n/a	0	1	0	3	0	1
getProductsArray()		100%		n/a	0	1	0	1	0	1
static {...}		100%		n/a	0	1	0	1	0	1
Total	12 of 456	97%	8 of 46	82%	8	38	4	86	0	15

Analysis We are only 2% short of the ideal requirement of 90%. We would have to test our getProduct array, createproduct elements in our InventoryFileDAO more thoroughly in order to achieve a decent coverage. As we can see, all the other elements have been covered 100%.

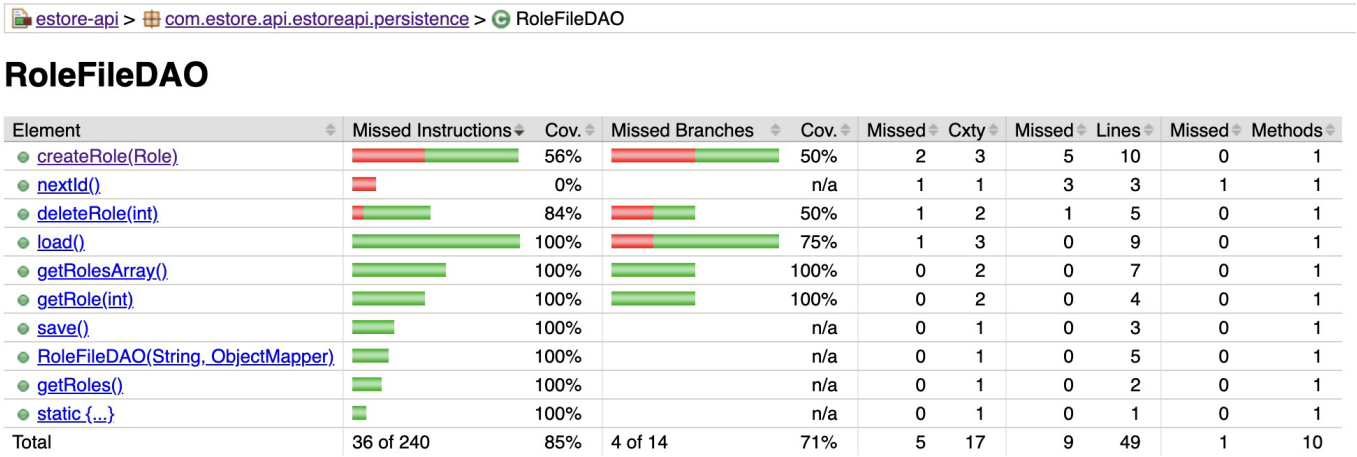
```
    private Product[] getProductsArray(String containsText, Integer containsCalories, boolean getAll) {
        ArrayList<Product> productArrayList = new ArrayList<>();
        // if containsText == null, no filter, (excluding because controller accounts
        // for no query parameter)
        for (Product product : products.values()) {
            if (getAll == true) {
                productArrayList.add(product);
            } else if (containsCalories == null) {
                // matches has been used instead of containsText to make search case insensitive
                // ?i switches to case insensitive mode, .* means any sequence of characters is
                // accepted
                if (product.getName().matches("(?i).*" + containsText + ".*"))
                    productArrayList.add(product);
            } else if (containsText == null) {
                if (product.getCalories() <= containsCalories)
                    productArrayList.add(product);
            } else if (containsCalories != null && containsText != null) {
                if (product.getName().matches("(?i).*" + containsText + ".*")
                        && product.getCalories() <= containsCalories)
                    productArrayList.add(product);
            }
        }

        Product[] productArray = new Product[productArrayList.size()];
        productArrayList.toArray(productArray);
        return productArray;
    }
```

We could add tests that take into account when containsCalories == null and containsCalories != null. We could add tests that account for if containsText == null and getCalories<=containsCalories, expect the product to be added to productArrayList. We could write the tests that are alternative to this. A final set of testcases could expect that product was added if containsCalories and containsText != null along with products name matches the text searched. One alternative to this would be expect product not to be added if these if conditions are not met.

Coverage: Poorly Tested Component

This is our analysis of a poorly-tested component.



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

RoleFileDAO

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
createRole(Role)		56%		50%	2	3	5	10	0	1
nextId()		0%		n/a	1	1	3	3	1	1
deleteRole(int)		84%		50%	1	2	1	5	0	1
load()		100%		75%	1	3	0	9	0	1
getRolesArray()		100%		100%	0	2	0	7	0	1
getRole(int)		100%		100%	0	2	0	4	0	1
save()		100%		n/a	0	1	0	3	0	1
RoleFileDAO(String, ObjectMapper)		100%		n/a	0	1	0	5	0	1
getRoles()		100%		n/a	0	1	0	2	0	1
static {...}		100%		n/a	0	1	0	1	0	1
Total	36 of 240	85%	4 of 14	71%	5	17	9	49	1	10

Analysis In RoleFileDAO element, it only covers 85% code and covers 71% branches and createRole element, it only covers 56% code. We will work on this poorly tested component as soon as possible.

Object Oriented Principle Adherence

Single Responsibility

Single Responsibility is very essential for our project. This principle is made use of in this project. Single Responsibility is one of the most important principles across all the principles. It simply means that each class defined must have only one single responsibility. It makes the application easier to maintain and also easier to understand. It is also done to ensure that in the future, if any changes are made to the code, it does not affect every class or dependent since each key functionality has its own separate class. Each class in the project has a very well-defined responsibility it must adhere to. The methods and attributes inside the class also contribute to the same. For example, the Product Class, holds information regarding every aspect of the product pertaining to that itself and has only that responsibility. It has many advantages, but the notable ones are that it is easier to understand the scope of a change in a class. It is easier to manage concurrent modifications or changes made.

Its main goal is the separation of responsibilities, which causes separate concerns to go into separate classes and also helps with testing the application, since it's simpler to test each short individual class. The project follows this object-oriented design principle. For example , the InventoryController Class had the sole purpose of handling API requests related to product inventory. If any particular request arrives, it has to communicate with that particular method and then provide the response. It handles API requests and returns back HTTP responses. This class has no apparent relationship or even is not concerned with the management of the Products data, or even the persistence aspect which is the storage mechanism for our project. The persistence directory has the InventoryDAO and the Inventory File DAO class, which has the responsibility of storing data.

The Inventory Controller Class delegates this management and storage responsibility to the InventoryDAO Class. Hence, the InventoryController class of our project is an apt example for single responsibility. Clustering every mechanism into this class would limit the reusability of the class, making it difficult to unit test the application. Solidifying the purpose of each class, so that each class has one responsibility leads to better adherence with this principle. Unnecessary coupling must also be avoided since single responsibility could lead to that.

Dependency Inversion

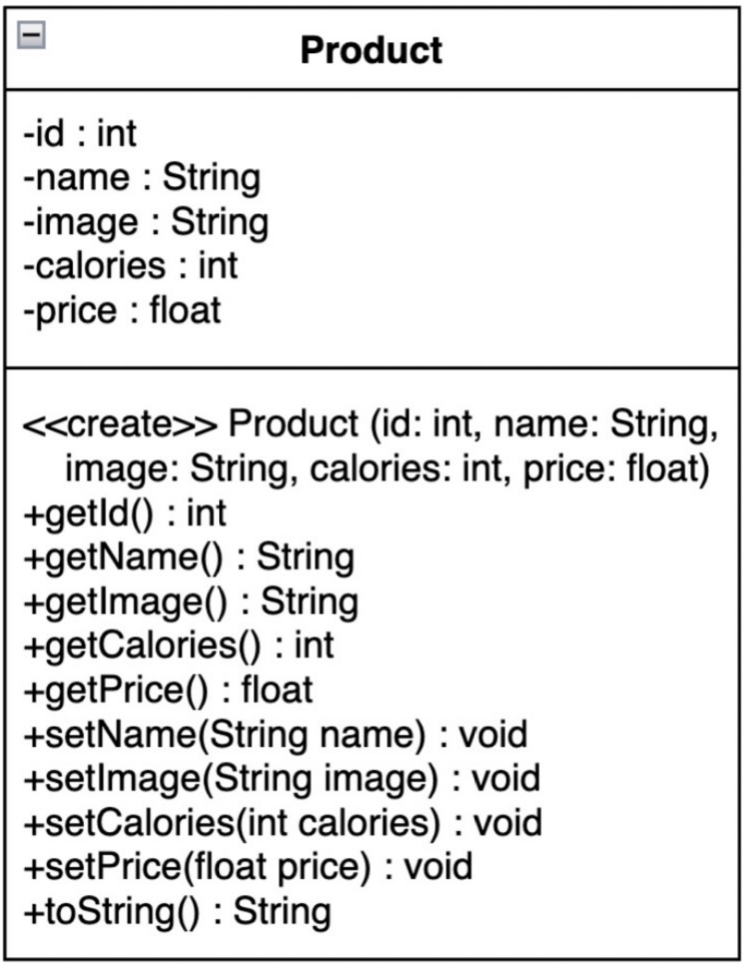
The Dependency Inversion Principle provides and allows room for looser coupling between dependent entities. Martin defines this principle as “High Level Modules should not depend on low-level modules. Both should depend on abstractions.". This project utilizes this principle. Dependency Inversion Principle reduces coupling between different pieces of code. It basically seeks to help with code reusability. Also, this principle is critical for doing unit tests since we can inject objects into the constructor. In this the low-level module has no responsibility to instantiate a dependent class. The high-level module injects the dependent elements and the low-level is dependent on the high-level’s abstraction and not its implementation. Dependency Injection can be defined as a principle that provides looser coupling between dependent entities. In other words, higher level classes should not depend on low-level classes. Both should depend on some layer of abstraction to communicate. Calorify's backend API is built on Spring Boot which provides what is necessary for successfully implementing dependency injection through the use of a configuration file. Dependency injection is implemented between InventoryController, and InventoryFileDAO through the interface InventoryDAO. Spring, via configuration, creates an InventoryFileDAO object.

For example, in this case the framework creates an InventoryFileDAO object. It then, injects this InventoryFileDAO object into the Inventory Controller when it’s called or instantiated. Our controller class, Inventory Controller only is told to deal with the abstraction which is the higher level InventoryDAO. This enhances the reusability of the code in addition to provide the ease of independently testing the Inventory Controller and the InventoryFileDAO class. As discussed in the above example, the InventoryController Class only deals with the higher level InventoryDAO abstraction. The benefit of this is that the lower-level implementation of storing and accessing/manipulating data can be updated at any point in time without having an effect on the InventoryController class’s responsibility. To illustrate this, the current underlying storing mechanism is a file using json objects representing an array of products. So, we know, that as long as the data access object is adhering to all the principles, our InventoryController does not need to change at all. In fact, even in angular the services aspect is a huge supporter of dependency injection. Since the project follows MVC (Model-View-Controller) Architecture, it already exclusively is designed to support the Object-Oriented Programming Design Principles.

Information Expert

Information Expert is a design pattern that states “Assign responsibility to the class that has the information needed to fulfill the responsibility.” According to this principle, responsibilities are assigned in the form of methods which are fields to classes. All behaviors that directly work with a class's attributes should be implemented in the class itself. Any operations that the client would perform using the attribute data or class methods should also be considered for implementation. The client shouldn’t be doing any “heavy-lifting”, all methods should be simplified for their use.

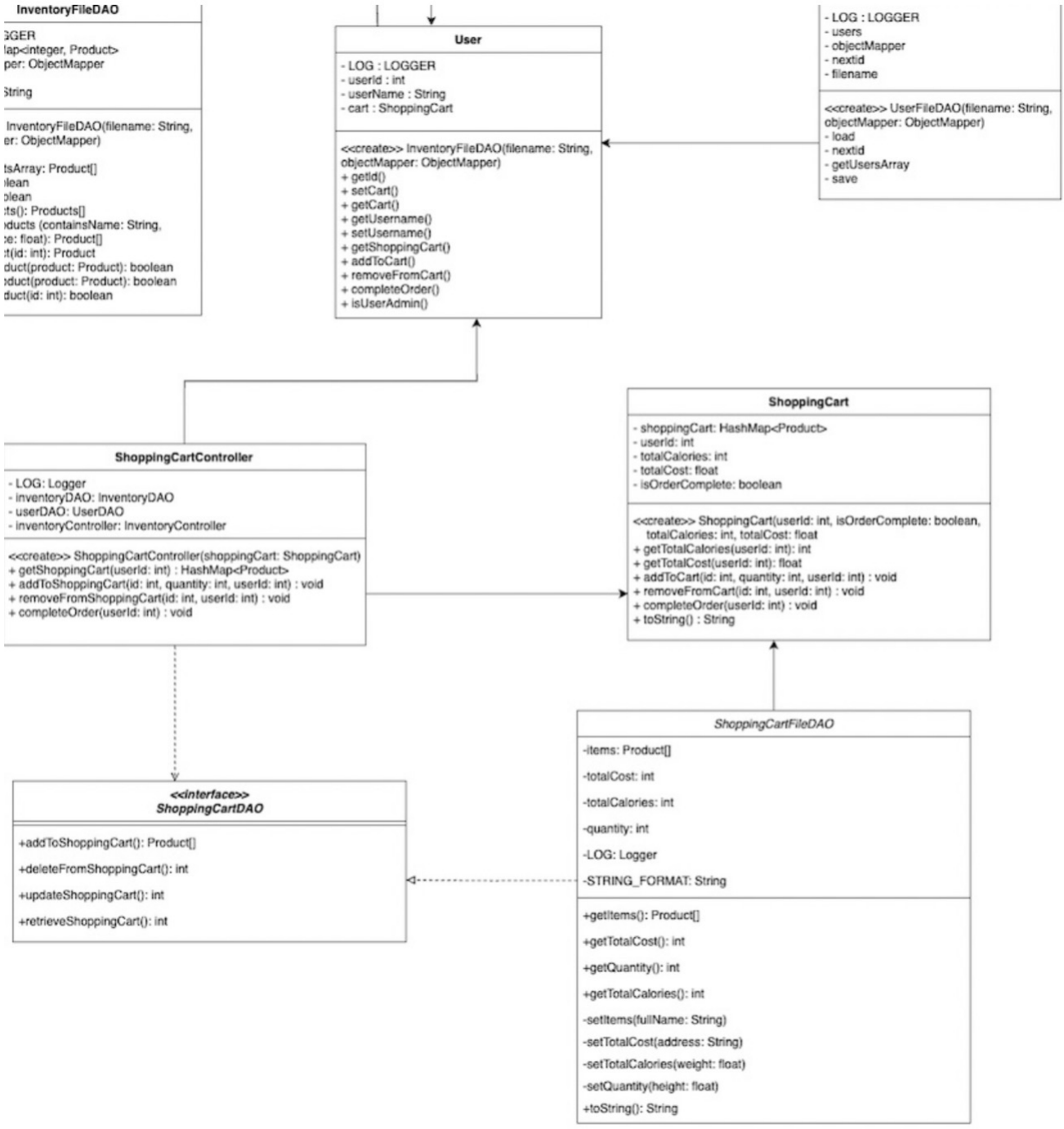
For example, this is the UML for our Product class:



All the getter and setter methods are provided which let the user retrieve and modify individual attributes of a product without fully deleting and defining a product again. The toString() method makes it easy to print all the details of a product without calling all the getter functions at once.

Low Coupling

Low coupling is the principle which aims to reduce the impact of any change in the system. If a change needs to be made somewhere, there shouldn't be any unnecessary coupling such that you may have to make many changes in different classes. For example, To make the Product class to work, an InventoryController class, an InventoryDAO interface, and the InventoryFileDAO class which implements it, are required, which are all connected together. There should be minimal number of connections for any class.



The User offloads its work to the ShoppingCartController which does all the work of creating a shopping cart for each user and interacting with its own DAO. Before this, we had designed the User to have a shopping cart as their own attribute, but that led to a lot of coupling as we needed to use a DAO for persistent storage of shopping carts for each user, and the user was already connected to an InventoryController, which handled the DAO for the Inventory.