PROJECT Design Documentation

Team Information

- Team name: 8h-8ball
- Team members
  - Aadith Charugundla
  - Cullen Walsh
  - Jessica Phillips
  - Zachary Rutherford
  - Frank Andes

Executive Summary

Purpose

This project will an e-store website to allow a small business owner to sell products to consumers. This specific e-store will feature a varitiey of coffee and tea blends to be sold to customers. The e-store will allow the manage inventory and add, modify, or remove products to be sold, while the customer will be able to browse the products and purchase them through the e-store.

Glossary and Acronyms

Term	Definition
SPA	Single Page
HTML	Hypertext Markup Language
MVP	Minimal Viable Product
MVVM	Model-View-ViewModel
CSS	Cascading Style Sheets
REST API	Representational State Transfer Application Programming Interface
HW	Hardware
OO	Object-Oriented
OS	Operating System

Requirements

Features of the application include browsing products, purchasing tea or coffee, adding products to a cart and proceeding to checkout, selecting custom packaging, inventory management, admin managment, and managing customer orders.

Definition of MVP

The MVP includes minimal authentication, which includes customer and admin user accounts for logging in and out of the site. As well as customer and e-store admin fuctionality. The required functionality involves searching for products, selecting the size of a product, modifying personal information, and addind or removing products from the cart, for customers. The admin aspect of the functionality involves adding, removing, and editing product and ingredient data, and adding or removing other admins. Admins will not have access to a shopping cart like a customer would.

MVP Features

> ***[Sprint 4]*** *Provide a list of top-level Epics and/or Stories of the MVP.*

- Prepackaged Tea/Coffee
- Customer Orders
- Single Cart
- Checkout
- Remove Products
- Modify Products
- Add Products
- Browse Products
- Add Admins
- Remove Admins
- Manage Ingredients
- User Login/Register
- Modify User

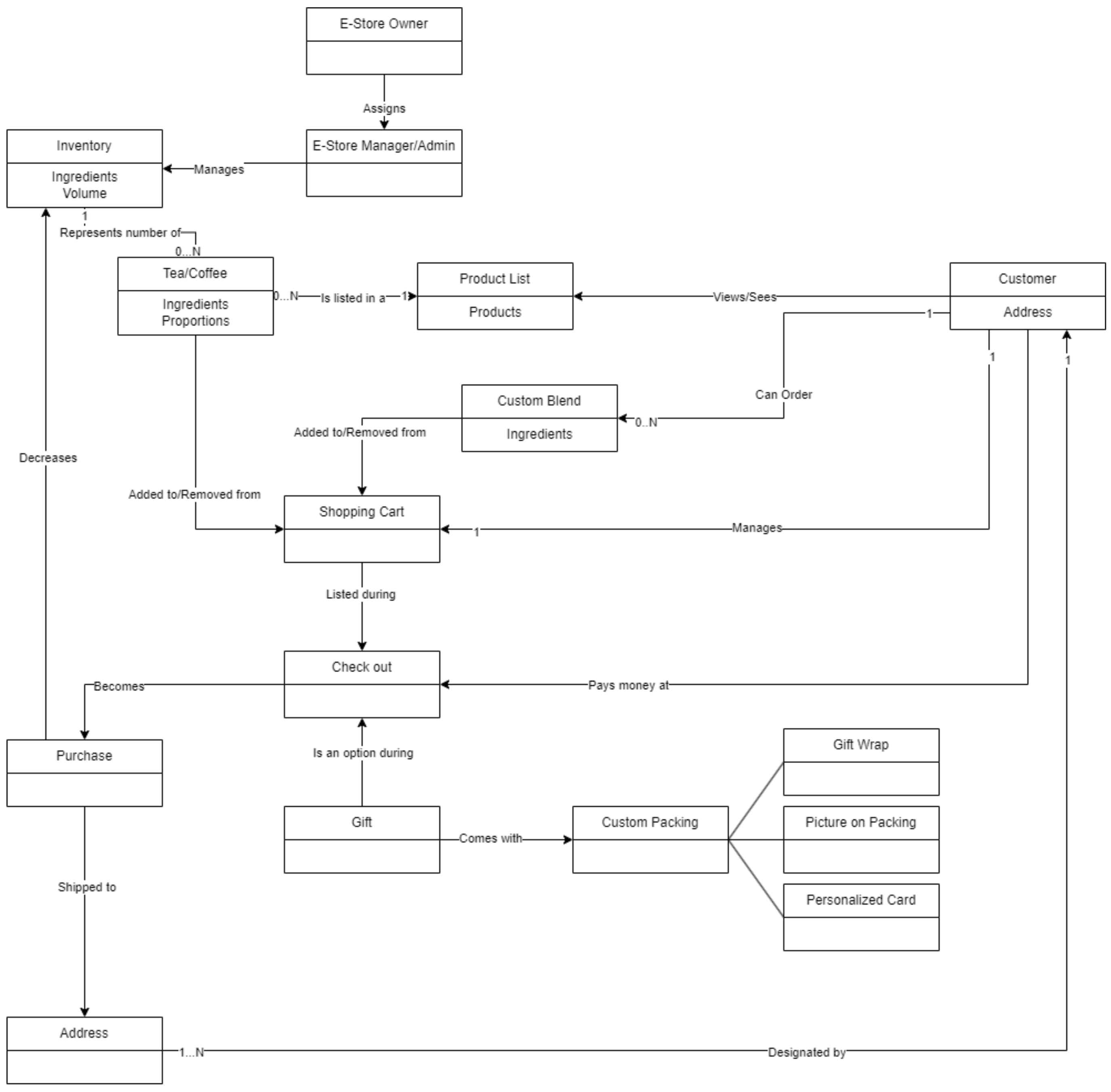
Enhancements

> ***[Sprint 4]*** *Describe what enhancements you have implemented for the project.*

- Gift Wrapping
- Package Picture
- Personalized Card
- Custom Blend
- Multiple Addresses
- Modify Admin Priviledges

Application Domain

This section describes the application domain.



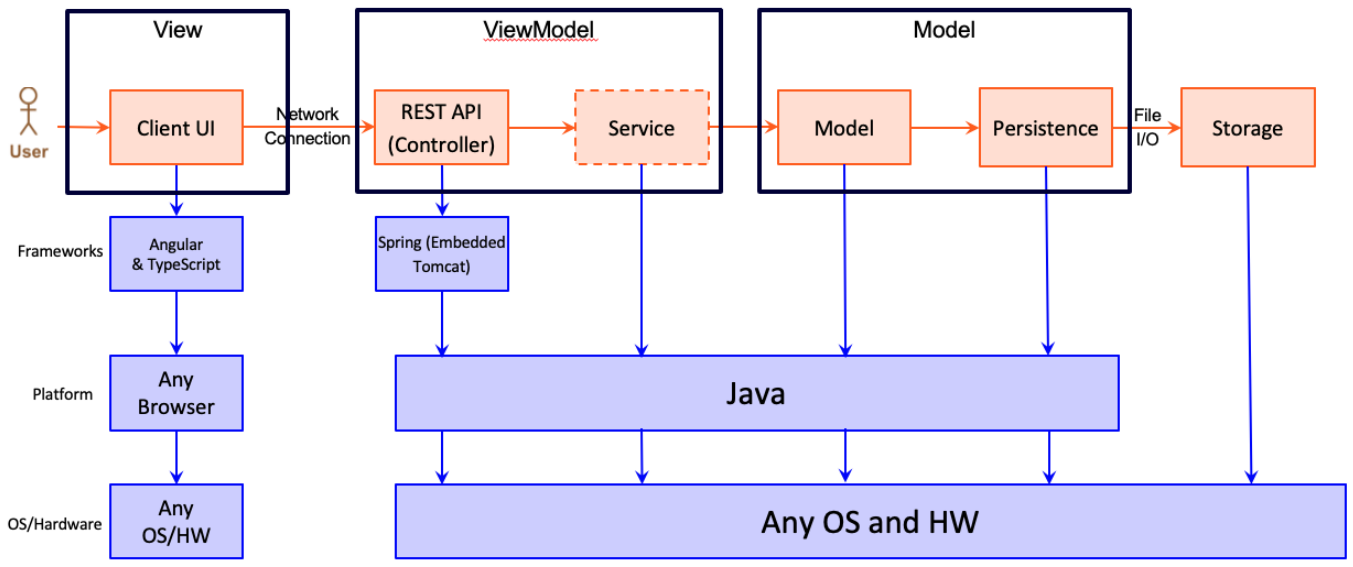
The customer manages the cart which contains various products from the product list. The cart is then listed during checkout where the user has the option to designate whether it's a gift and whether they want to add anything extra to it. Once they've made a purchase, the product is shipped to the user's address.

> ***[Sprint 2 & 4]*** *Provide a high-level overview of the domain for this application. You can discuss the more important domain entities and their relationship to each other.*

Architecture and Design

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.

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

The very top left of our webpage displays the text "Coffee/Tea E-store" and is prevalent on all the pages. Right underneath this text are the buttons "Back" and "Logout", since we believed these to be the most important. This notion is reflected in almost all websites and browsers. Notice that Chrome for example has the back arrow on the top left of the window. In the same row but on the far right, we have the "Settings" button, which brings the user to the user settings component. We belived that this also belonged near the top because one of the first things a new user would have to do is change their personal information by adding a credit card number, address, etc. A bit below that on the next row, we have the "Home", Dashboard", and "Store" buttons next to each other. The home button naturally takes the user to the homepage which is currently empty, but will later have more information on it. The Dashboard button brings the user to the dashboard where they can see a list of top products as well as search for products. The Store button leads to the main store which displays a table whose columns are Name, Blend Type, Ingredients, and Price. Right above this table is a search bar which can be used to search for individual products. The user also has the option to sort products by increasing or decreasing alphabetical order by clicking on the "Name" column. Once the user clicks on a product they are taken to a separate compoent which shows the details of the product such as the name, blend, sizes, price, and ingredients. The user has a choice when it comes to what size they want (from the options available since some sizes might not have enough ingredients to make) and once they've picked one they can click add to cart, which opens the cart on the side of the screen. This has a close button, as well as the option to remove items from the cart. Once the user is done creating their cart, they can click the Checkout button which takes them to the checkout component. Here they can see the products they have in their cart, as well as the address and payment method. When they are ready, they can click Order and will be brought back to the home page with an empty cart. If the user clicks logout they are taken back to the login page where they are prompted to input a username and password before clicking Login. If they wish, they can also register as a new user. If the user signs in as an administrator, they will see that they do not have access to the cart and instead have the ability to manage the store by managing the products (creating/deleting/modifying) as well as managing the ingredients (creating/deleting/modifying). They also have the option to view the current admins and add/remove other admins. In the orders page you can see a list of user orders that need to be completed, and if you click the fulfilled button you can see all the orders that have been completed.

View Tier

> ***[Sprint 4]*** *Provide a summary of the View Tier UI of your architecture. Describe the types of components in the tier and describe their responsibilities. This should be a narrative description, i.e. it has a flow or "story line" that the reader can follow.*

> ***[Sprint 4]*** *You must provide at least* ***2 sequence diagrams*** *as is relevant to a particular aspects of the design that you are describing. For example, in e-store you might create a sequence diagram of a customer searching for an item and adding to their cart. As these can span multiple tiers, be sure to include an relevant HTTP requests from the client-side to the server-side to help illustrate the end-to-end flow.*

> ***[Sprint 4]*** *To adequately show your system, you will need to present the* ***class diagrams*** *where relevant in your design. Some additional tips:*
>
> - *Class diagrams only apply to the* ***ViewModel*** *and* ***Model*** *Tier*
> - *A single class diagram of the entire system will not be effective. You may start with one, but will be need to break it down into smaller sections to account for requirements of each of the Tier static models below.*
> - *Correct labeling of relationships with proper notation for the relationship type, multiplicities, and navigation information will be important.*
> - *Include other details such as attributes and method signatures that you think are needed to support the level of detail in your discussion.*

ViewModel Tier

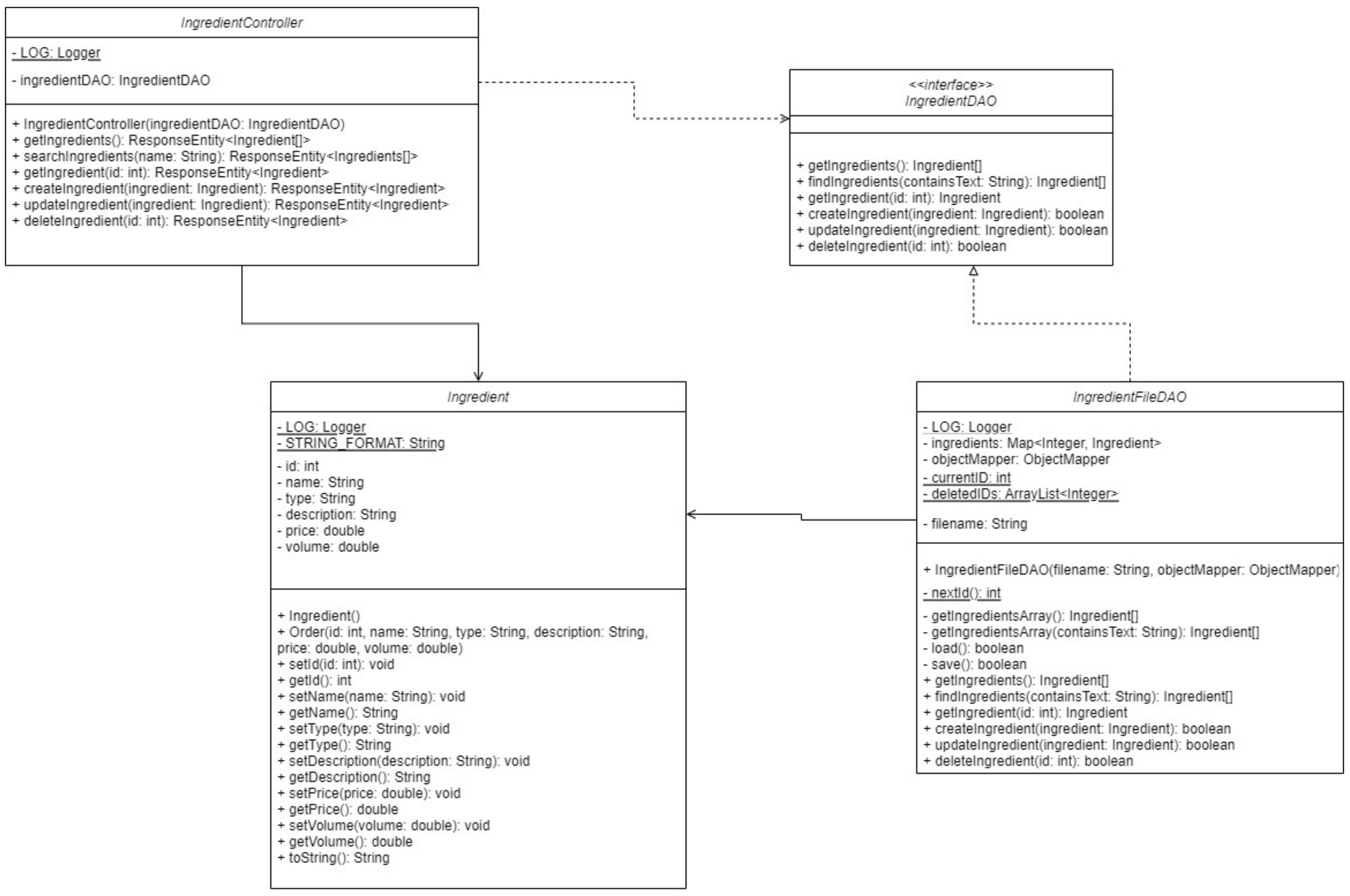
> ***[Sprint 4]*** *Provide a summary of this tier of your architecture. This section will follow the same instructions that are given for the View Tier above.*

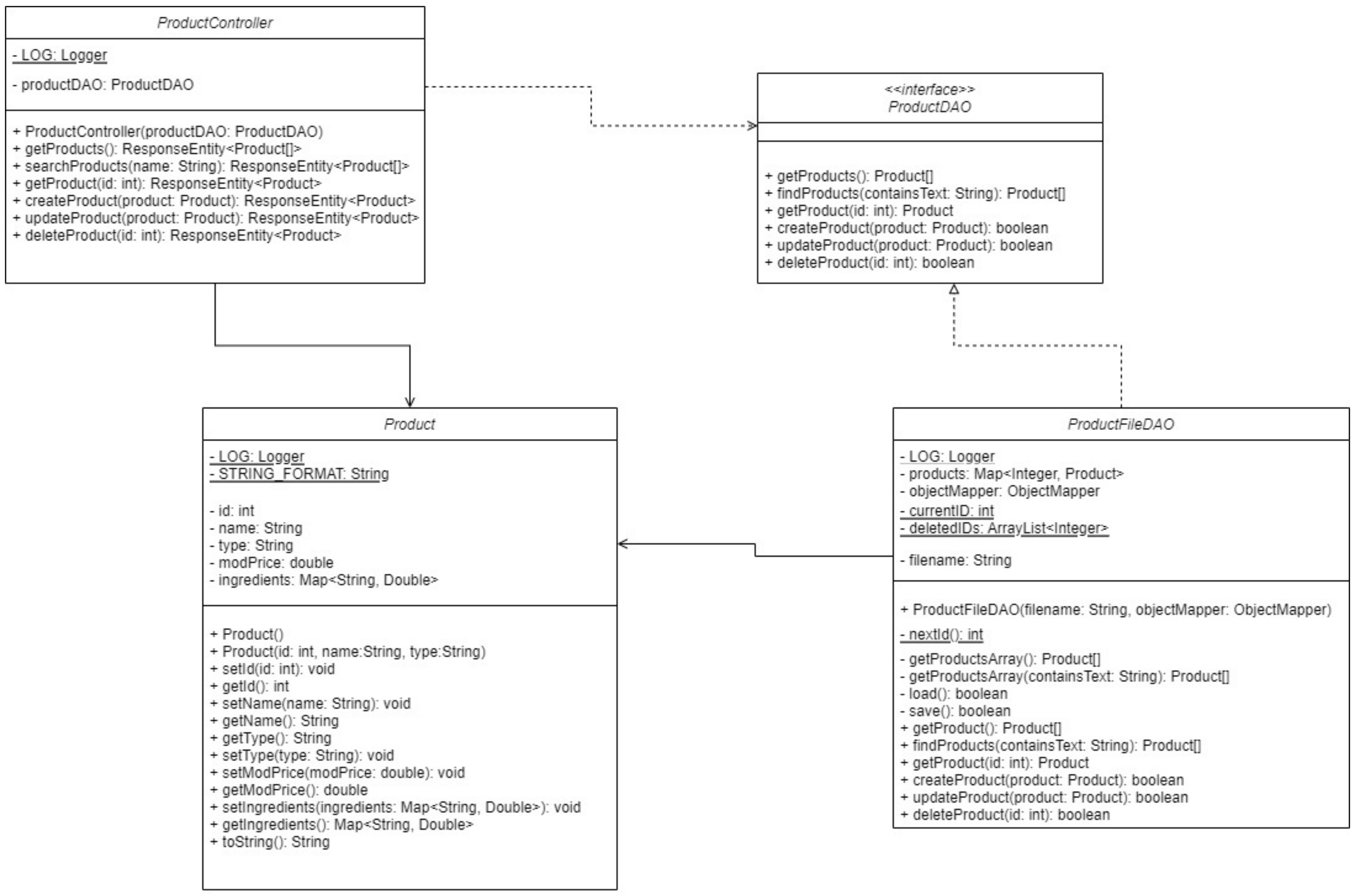
> *At appropriate places as part of this narrative provide* ***one*** *or more updated and* ***properly labeled*** *static models (UML class diagrams) with some details such as critical attributes and methods.*

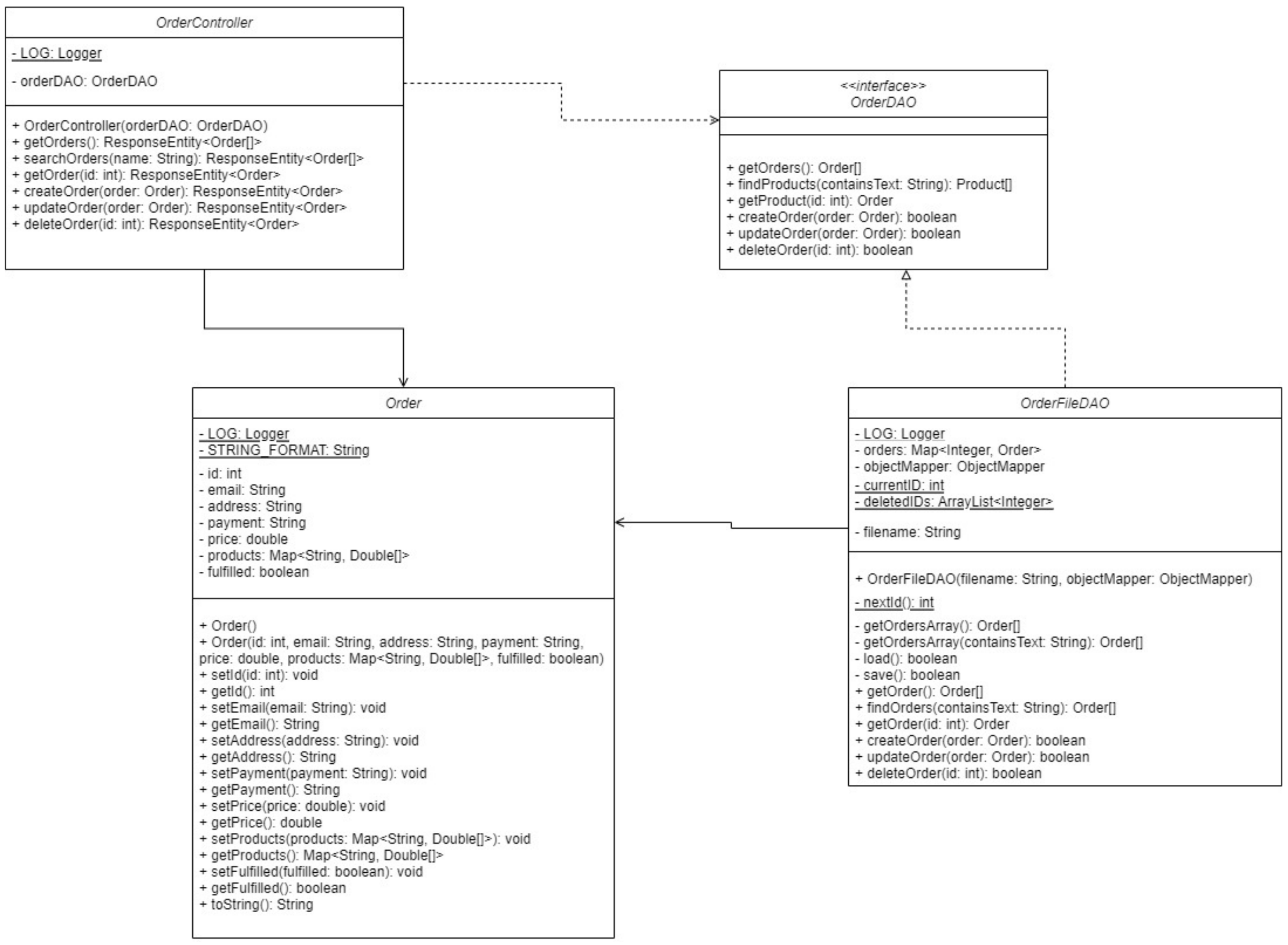
Replace with your ViewModel Tier class diagram 1, etc.

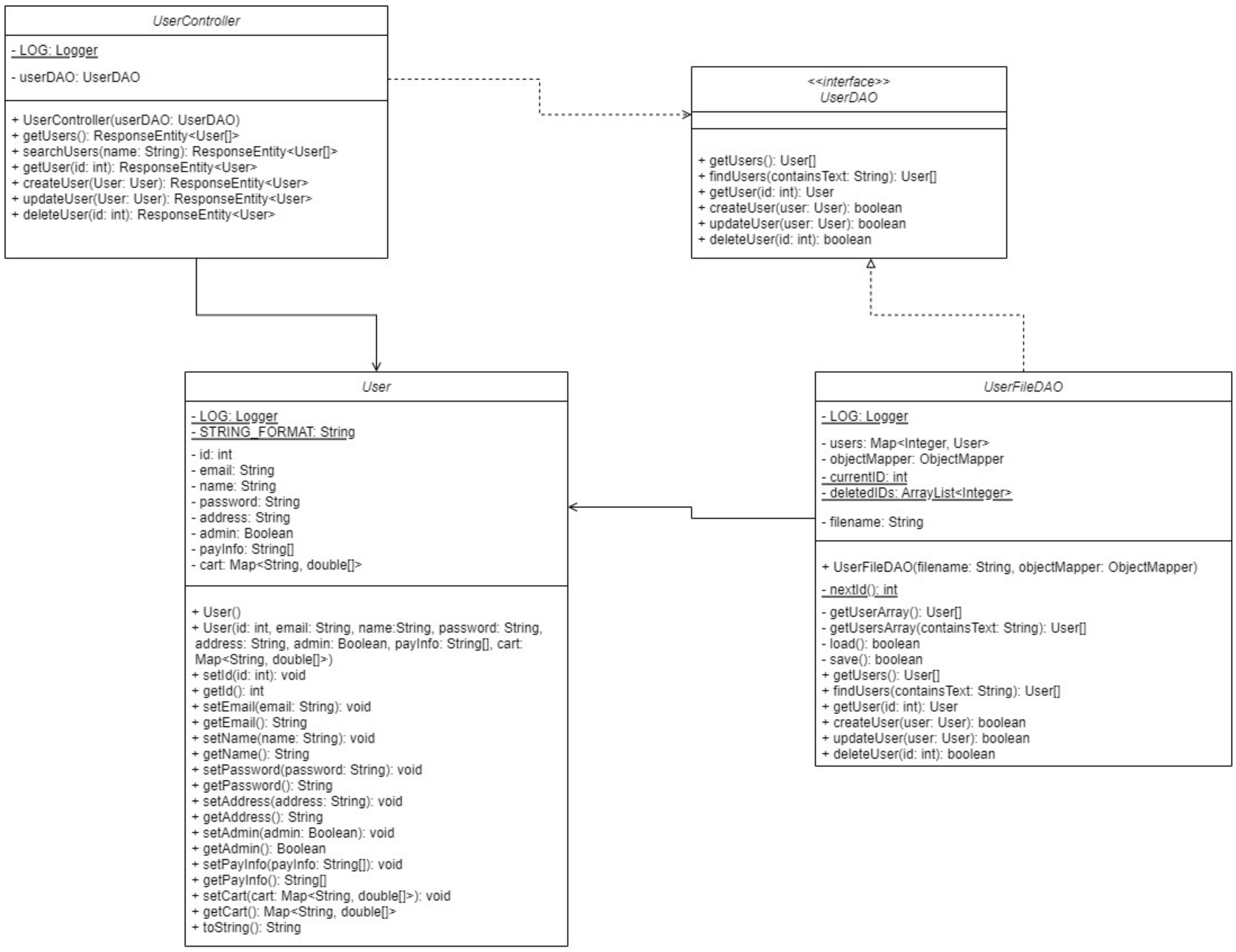
Model Tier

> ***[Sprint 2, 3 & 4]*** *Provide a summary of this tier of your architecture. This section will follow the same instructions that are given for the View Tier above.*


IngredientController
- LOG: Logger
- ingredientDAO: IngredientDAO
+ IngredientController(ingredientDAO: IngredientDAO)
+ getIngredients(): ResponseEntity<Ingredient[]>
+ searchIngredients(name: String): ResponseEntity<Ingredients[]>
+ getIngredient(id: int): ResponseEntity<Ingredient>
+ createIngredient(ingredient: Ingredient): ResponseEntity<Ingredient>
+ updateIngredient(ingredient: Ingredient): ResponseEntity<Ingredient>
+ deleteIngredient(id: int): ResponseEntity<Ingredient>
<<interface>>
IngredientDAO
+ getIngredients(): Ingredient[]
+ findIngredients(containsText: String): Ingredient[]
+ getIngredient(id: int): Ingredient
+ createIngredient(ingredient: Ingredient): boolean
+ updateIngredient(ingredient: Ingredient): boolean
+ deleteIngredient(id: int): boolean
Ingredient
- LOG: Logger
- STRING_FORMAT: String
- id: int
- name: String
- type: String
- description: String
- price: double
- volume: double
+ Ingredient()
+ Order(id: int, name: String, type: String, description: String, price: double, volume: double)
+ setId(id: int): void
+ getId(): int
+ setName(name: String): void
+ getName(): String
+ setType(type: String): void
+ getType(): String
+ setDescription(description: String): void
+ getDescription(): String
+ setPrice(price: double): void
+ getPrice(): double
+ setVolume(volume: double): void
+ getVolume(): double
+ toString(): String
IngredientFileDAO
- LOG: Logger
- ingredients: Map<Integer, Ingredient>
- objectMapper: ObjectMapper
- currentID: int
- deletedIDs: ArrayList<Integer>
- filename: String
+ IngredientFileDAO(filename: String, objectMapper: ObjectMapper)
- nextId(): int
- getIngredientsArray(): Ingredient[]
- getIngredientsArray(containsText: String): Ingredient[]
- load(): boolean
- save(): boolean
+ getIngredients(): Ingredient[]
+ findIngredients(containsText: String): Ingredient[]
+ getIngredient(id: int): Ingredient
+ createIngredient(ingredient: Ingredient): boolean
+ updateIngredient(ingredient: Ingredient): boolean
+ deleteIngredient(id: int): boolean


ProductController
- LOG: Logger
- productDAO: ProductDAO
+ ProductController(productDAO: ProductDAO)
+ getProducts(): ResponseEntity<Product[]>
+ searchProducts(name: String): ResponseEntity<Product[]>
+ getProduct(id: int): ResponseEntity<Product>
+ createProduct(product: Product): ResponseEntity<Product>
+ updateProduct(product: Product): ResponseEntity<Product>
+ deleteProduct(id: int): ResponseEntity<Product>
<<interface>>
ProductDAO
+ getProducts(): Product[]
+ findProducts(containsText: String): Product[]
+ getProduct(id: int): Product
+ createProduct(product: Product): boolean
+ updateProduct(product: Product): boolean
+ deleteProduct(id: int): boolean
Product
- LOG: Logger
- STRING_FORMAT: String
- id: int
- name: String
- type: String
- modPrice: double
- ingredients: Map<String, Double>
+ Product()
+ Product(id: int, name:String, type:String)
+ setId(id: int): void
+ getId(): int
+ setName(name: String): void
+ getName(): String
+ getType(): String
+ setType(type: String): void
+ setModPrice(modPrice: double): void
+ getModPrice(): double
+ setIngredients(ingredients: Map<String, Double>): void
+ getIngredients(): Map<String, Double>
+ toString(): String
ProductFileDAO
- LOG: Logger
- products: Map<Integer, Product>
- objectMapper: ObjectMapper
- currentID: int
- deletedIDs: ArrayList<Integer>
- filename: String
+ ProductFileDAO(filename: String, objectMapper: ObjectMapper)
- nextId(): int
- getProductsArray(): Product[]
- getProductsArray(containsText: String): Product[]
- load(): boolean
- save(): boolean
+ getProduct(): Product[]
+ findProducts(containsText: String): Product[]
+ getProduct(id: int): Product
+ createProduct(product: Product): boolean
+ updateProduct(product: Product): boolean
+ deleteProduct(id: int): boolean


OrderController
- LOG: Logger
- orderDAO: OrderDAO
+ OrderController(orderDAO: OrderDAO)
+ getOrders(): ResponseEntity<Order[]>
+ searchOrders(name: String): ResponseEntity<Order[]>
+ getOrder(id: int): ResponseEntity<Order>
+ createOrder(order: Order): ResponseEntity<Order>
+ updateOrder(order: Order): ResponseEntity<Order>
+ deleteOrder(id: int): ResponseEntity<Order>
<<interface>>
OrderDAO
+ getOrders(): Order[]
+ findProducts(containsText: String): Product[]
+ getProduct(id: int): Order
+ createOrder(order: Order): boolean
+ updateOrder(order: Order): boolean
+ deleteOrder(id: int): boolean
Order
- LOG: Logger
- STRING_FORMAT: String
- id: int
- email: String
- address: String
- payment: String
- price: double
- products: Map<String, Double[]>
- fulfilled: boolean
+ Order()
+ Order(id: int, email: String, address: String, payment: String, price: double, products: Map<String, Double[]>, fulfilled: boolean)
+ setId(id: int): void
+ getId(): int
+ setEmail(email: String): void
+ getEmail(): String
+ setAddress(address: String): void
+ getAddress(): String
+ setPayment(payment: String): void
+ getPayment(): String
+ setPrice(price: double): void
+ getPrice(): double
+ setProducts(products: Map<String, Double[]>): void
+ getProducts(): Map<String, Double[]>
+ setFulfilled(fulfilled: boolean): void
+ getFulfilled(): boolean
+ toString(): String
OrderFileDAO
- LOG: Logger
- orders: Map<Integer, Order>
- objectMapper: ObjectMapper
- currentID: int
- deletedIDs: ArrayList<Integer>
- filename: String
+ OrderFileDAO(filename: String, objectMapper: ObjectMapper)
- nextId(): int
- getOrdersArray(): Order[]
- getOrdersArray(containsText: String): Order[]
- load(): boolean
- save(): boolean
+ getOrder(): Order[]
+ findOrders(containsText: String): Order[]
+ getOrder(id: int): Order
+ createOrder(order: Order): boolean
+ updateOrder(order: Order): boolean
+ deleteOrder(id: int): boolean


UserController
- LOG: Logger
- userDAO: UserDAO
+ UserController(userDAO: UserDAO)
+ getUsers(): ResponseEntity<User[]>
+ searchUsers(name: String): ResponseEntity<User[]>
+ getUser(id: int): ResponseEntity<User>
+ createUser(User: User): ResponseEntity<User>
+ updateUser(User: User): ResponseEntity<User>
+ deleteUser(id: int): ResponseEntity<User>
<<interface>> UserDAO
+ getUsers(): User[]
+ findUsers(containsText: String): User[]
+ getUser(id: int): User
+ createUser(user: User): boolean
+ updateUser(user: User): boolean
+ deleteUser(id: int): boolean
User
- LOG: Logger
- STRING_FORMAT: String
- id: int
- email: String
- name: String
- password: String
- address: String
- admin: Boolean
- payInfo: String[]
- cart: Map<String, double[]>
+ User()
+ User(id: int, email: String, name:String, password: String, address: String, admin: Boolean, payInfo: String[], cart: Map<String, double[]>)
+ setId(id: int): void
+ getId(): int
+ setEmail(email: String): void
+ getEmail(): String
+ setName(name: String): void
+ getName(): String
+ setPassword(password: String): void
+ getPassword(): String
+ setAddress(address: String): void
+ getAddress(): String
+ setAdmin(admin: Boolean): void
+ getAdmin(): Boolean
+ setPayInfo(payInfo: String[]): void
+ getPayInfo(): String[]
+ setCart(cart: Map<String, double[]>): void
+ getCart(): Map<String, double[]>
+ toString(): String
UserFileDAO
- LOG: Logger
- users: Map<Integer, User>
- objectMapper: ObjectMapper
- currentID: int
- deletedIDs: ArrayList<Integer>
- filename: String
+ UserFileDAO(filename: String, objectMapper: ObjectMapper)
- nextId(): int
- getUserArray(): User[]
- getUsersArray(containsText: String): User[]
- load(): boolean
- save(): boolean
+ getUsers(): User[]
+ findUsers(containsText: String): User[]
+ getUser(id: int): User
+ createUser(user: User): boolean
+ updateUser(user: User): boolean
+ deleteUser(id: int): boolean

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

Replace with your Model Tier class diagram 1, etc.

OO Design Principles

> ***[Sprint 2, 3 & 4]** Discuss at least **4 key OO Principles** in your current design. This should be taken from your work in "Adherence to Architecture and Design Principles" that you have completed in a previous Sprint. Be sure to include any diagrams (or clearly refer to ones elsewhere in your Tier sections above) to support your claims.*

Single Responsibility: The Controller class does not have methods to change or retrieve the name, id, or type of a product. Rather its only purpose is to add, remove, update, or delete products that already exist.

Pure Fabrication: Pure Fabrication appears within the FileDAO classes. The FileDAO extracts functionality that could have been contained within the model classes, but to maintain other OOP priniciples, functionality is split between the two.

Controller: A Controller can be defined as the first object beyond the UI level that receives and controls a set of operations. The Controller is the class that deals with the HTTP requests and is the first layer beyond the "UI": also known as the Terminal or Localhost page.

Open/Closed: The ProductController class is open for extension without modifying the base functionality. The ProductController class will be extened to other parts of the website but it will not be modified further.

> ***[Sprint 3 & 4]** OO Design Principles should span across **all tiers.***

Static Code Analysis/Future Design Improvements

> ***[Sprint 4]** With the results from the Static Code Analysis exercise, **Identify 3-4** areas within your code that have been flagged by the Static Code Analysis Tool (SonarQube) and provide your analysis and recommendations.*
> *Include any relevant screenshot(s) with each area.*

> ***[Sprint 4]** Discuss **future** refactoring and other design improvements your team would explore if the team had additional time.*

Testing

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

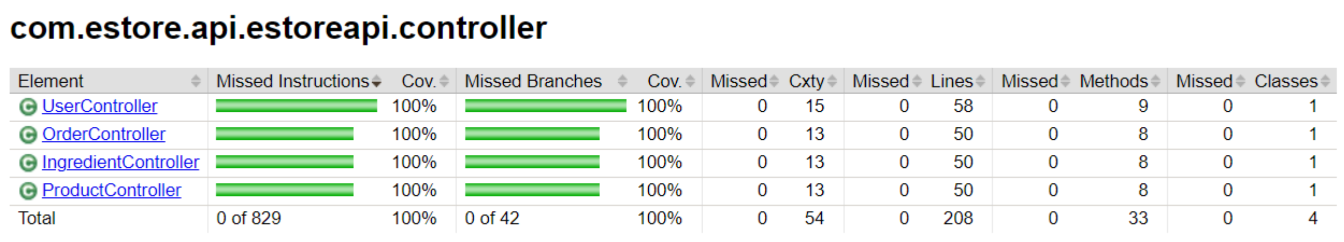
Acceptance Testing

> ***[Sprint 2 & 4]** Report on the number of user stories that have passed all their acceptance criteria tests, the number that have some acceptance criteria tests failing, and the number of user stories that have not had any testing yet. Highlight the issues found during acceptance testing and if there are any concerns.*

There are 13 stories that have passed their acceptance criteria tests. This is 100 percent of the the stories that have been developed. There are 0 stories that have failed their acceptance criteria tests, and 6 stories that have not yet been tested because they have not yet been developed.

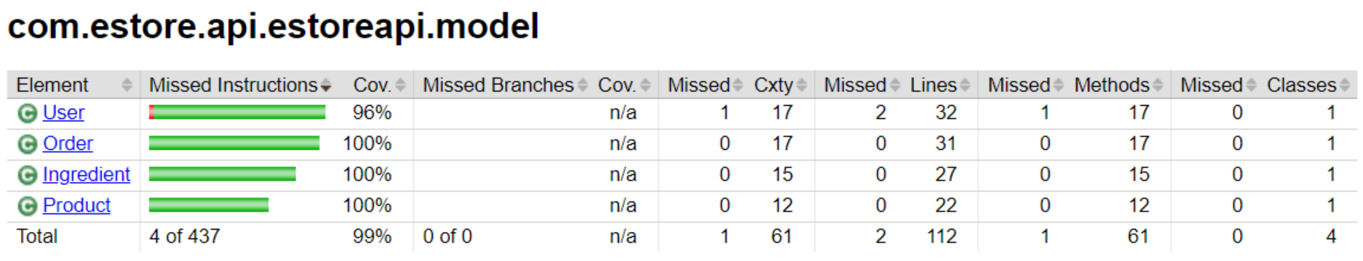
Unit Testing and Code Coverage

> ***[Sprint 4]** Discuss your unit testing strategy. Report on the code coverage achieved from unit testing of the code base. Discuss the team's coverage targets, why you selected those values, and how well your code coverage met your targets.*



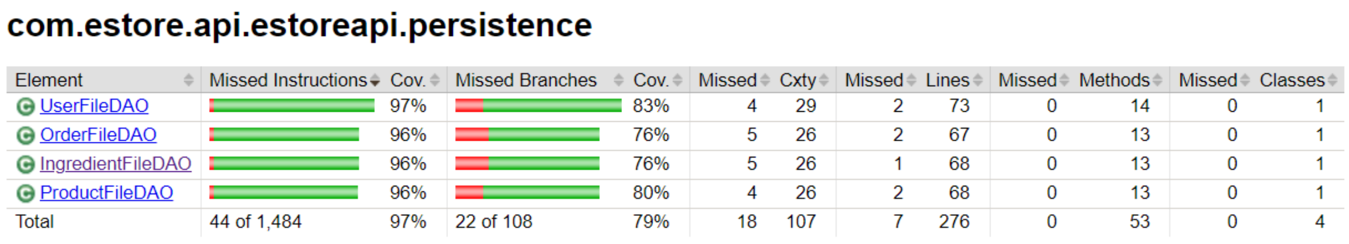
com.estore.api.estoreapi.controller

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
UserController		100%		100%	0	15	0	58	0	9	0	1
OrderController		100%		100%	0	13	0	50	0	8	0	1
IngredientController		100%		100%	0	13	0	50	0	8	0	1
ProductController		100%		100%	0	13	0	50	0	8	0	1
Total	0 of 829	100%	0 of 42	100%	0	54	0	208	0	33	0	4



com.estore.api.estoreapi.model

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
User		96%		n/a	1	17	2	32	1	17	0	1
Order		100%		n/a	0	17	0	31	0	17	0	1
Ingredient		100%		n/a	0	15	0	27	0	15	0	1
Product		100%		n/a	0	12	0	22	0	12	0	1
Total	4 of 437	99%	0 of 0	n/a	1	61	2	112	1	61	0	4



com.estore.api.estoreapi.persistence

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
UserFileDAO		97%		83%	4	29	2	73	0	14	0	1
OrderFileDAO		96%		76%	5	26	2	67	0	13	0	1
IngredientFileDAO		96%		76%	5	26	1	68	0	13	0	1
ProductFileDAO		96%		80%	4	26	2	68	0	13	0	1
Total	44 of 1,484	97%	22 of 108	79%	18	107	7	276	0	53	0	4

These images depict the code coverage for the e-store. The Controller, Model, and Persistence Tiers are covered at 100%, 99%, and 97% respectively. This is well above the acceptable rate of 95% for the Model Tier and 90% for other tiers. The code coverage results indicates that the code is being fully tested by the unit tests in place.

> ***[Sprint 2 & 4] Include images of your code coverage report.*** *If there are any anomalies, discuss those.*