PROJECT Design Documentation

Team Information

- Team name: Team 3 - Bookstore
- Team members
  - Rylan Arbour
  - Cole DenBleyker
  - Ryan Robison
  - Jack Hunsberger
  - Xin Huang

Executive Summary

The main intention of this project is to act as an e-store, with a focus in selling books. Users are able to login and add products to their shopping cart. They are also able to remove products from their shopping carts. If a user logs in as an admin then they gain access to edit products - being able to change the price, cost, quantity, name, and description. Admins can also add and remove products. The e-store also has promotional codes that admins can create and remove, and users can apply to get a discount on the total cost of their shopping cart. Users can also view their purcahse history.

Purpose

To create a robust E-store service in which customers can interact with an inventory of available items to add to their cart and checkout with.

Glossary and Acronyms

> ***[Sprint 2 & 4]*** *Provide a table of terms and acronyms.*

Term	Definition
SPA	Single Page

Requirements

This section describes the features of the application.

> *In this section you do not need to be exhaustive and list every story. Focus on top-level features from the Vision document and maybe Epics and critical Stories.*

Definition of MVP

> ***[Sprint 2 & 4]*** *Provide a simple description of the Minimum Viable Product.*

MVP Features

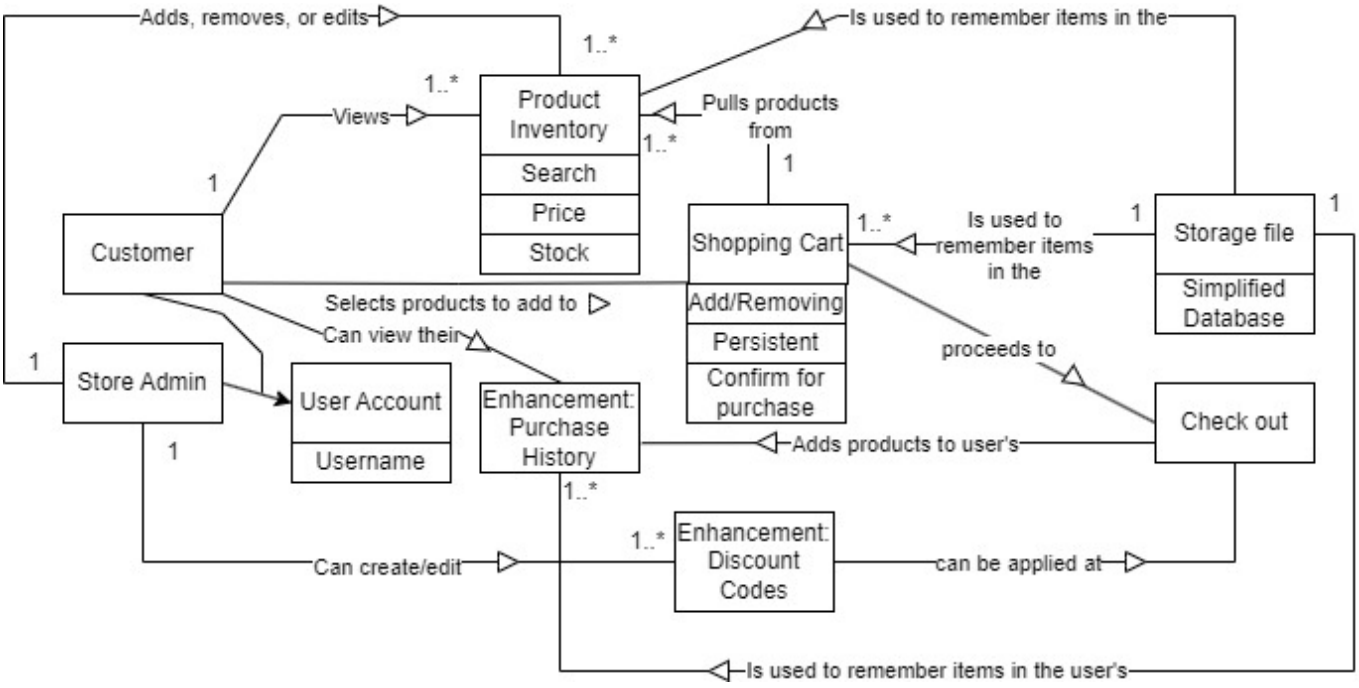
> ***[Sprint 4]*** *Provide a list of top-level Epics and/or Stories of the MVP.*

Enhancements

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

Application Domain

This section describes the application domain.



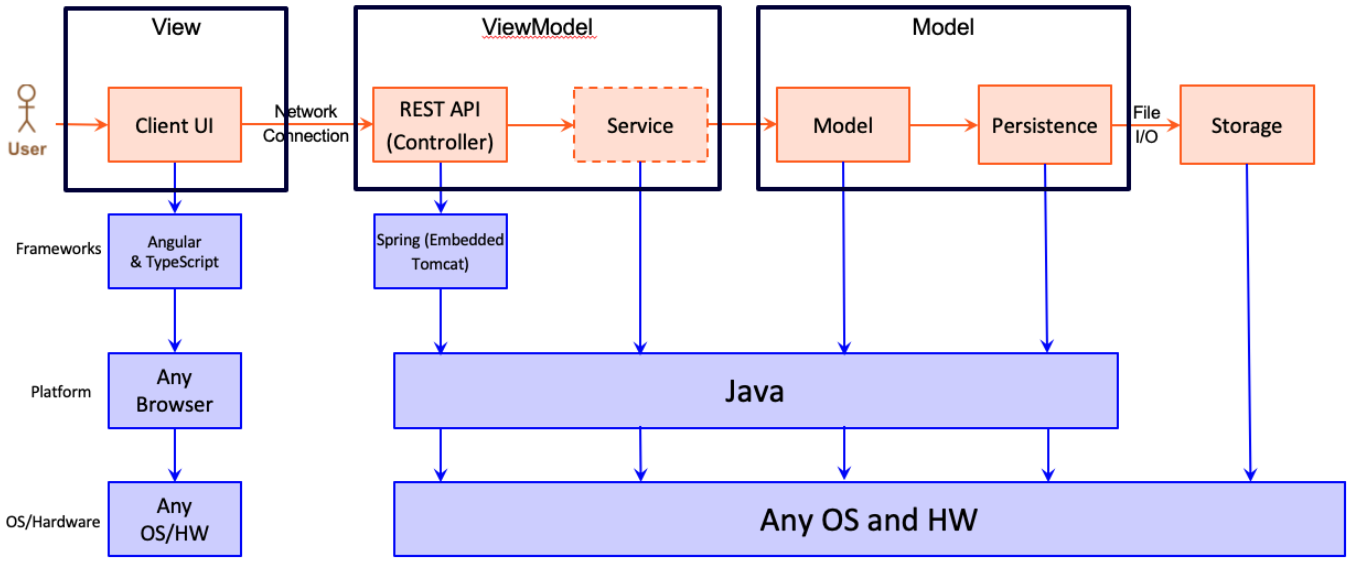
A customer has their own user account associated with a product history and shopping cart. In the estore, the customer can add items from the inventory to their cart, apply discount codes, and check out. Each product has a price, description, and stock that can be changed by the admin user.

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 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

This section describes the web interface flow; this is how the user views and interacts with the e-store application.

> *Provide a summary of the application's user interface. Describe, from the user's perspective, the flow of the pages in the web application.*

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

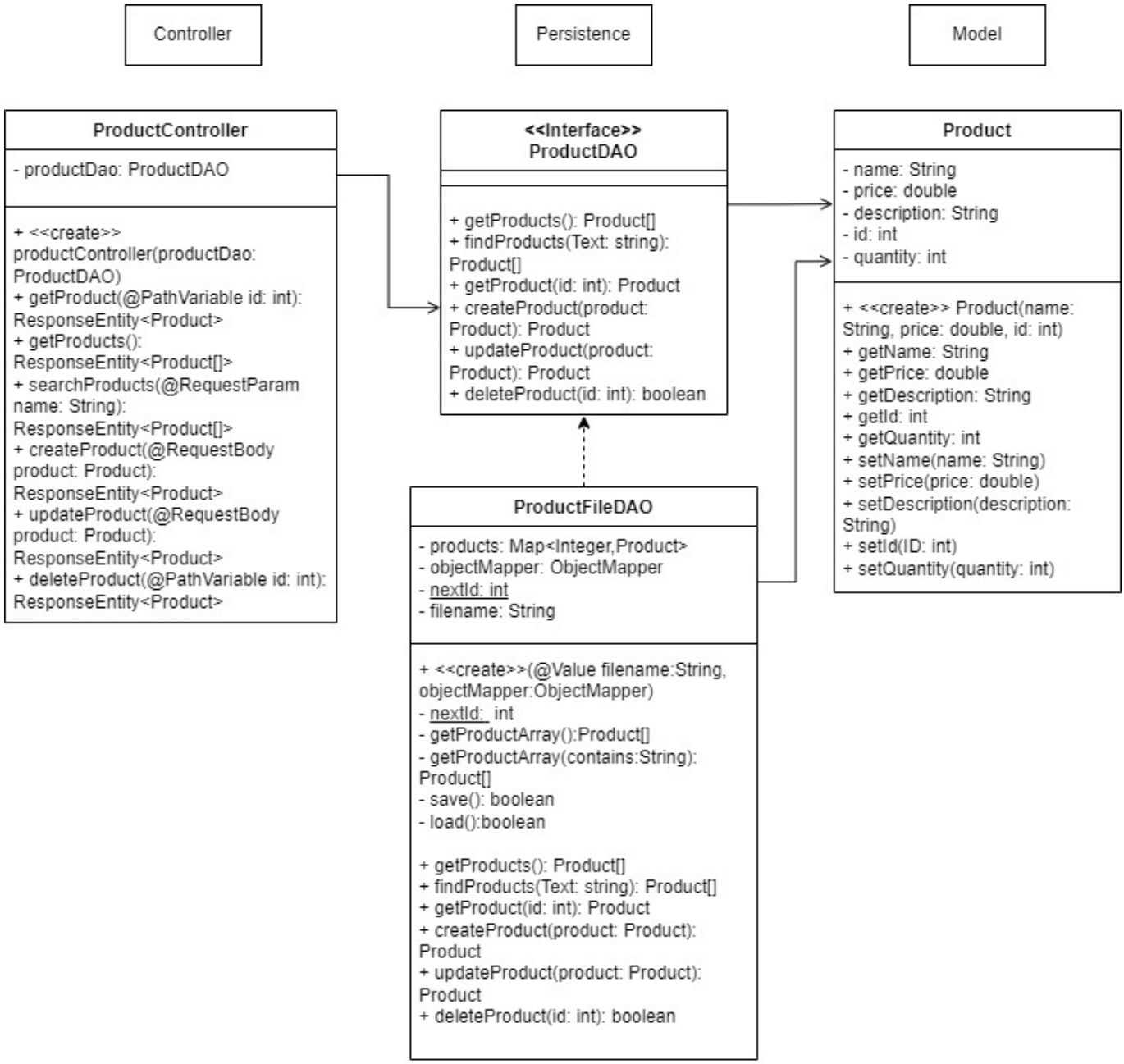
> ***[Sprint 4]** Provide a summary of this tier of your architecture. This section will follow the same instructions that are given for the View Tier above.*

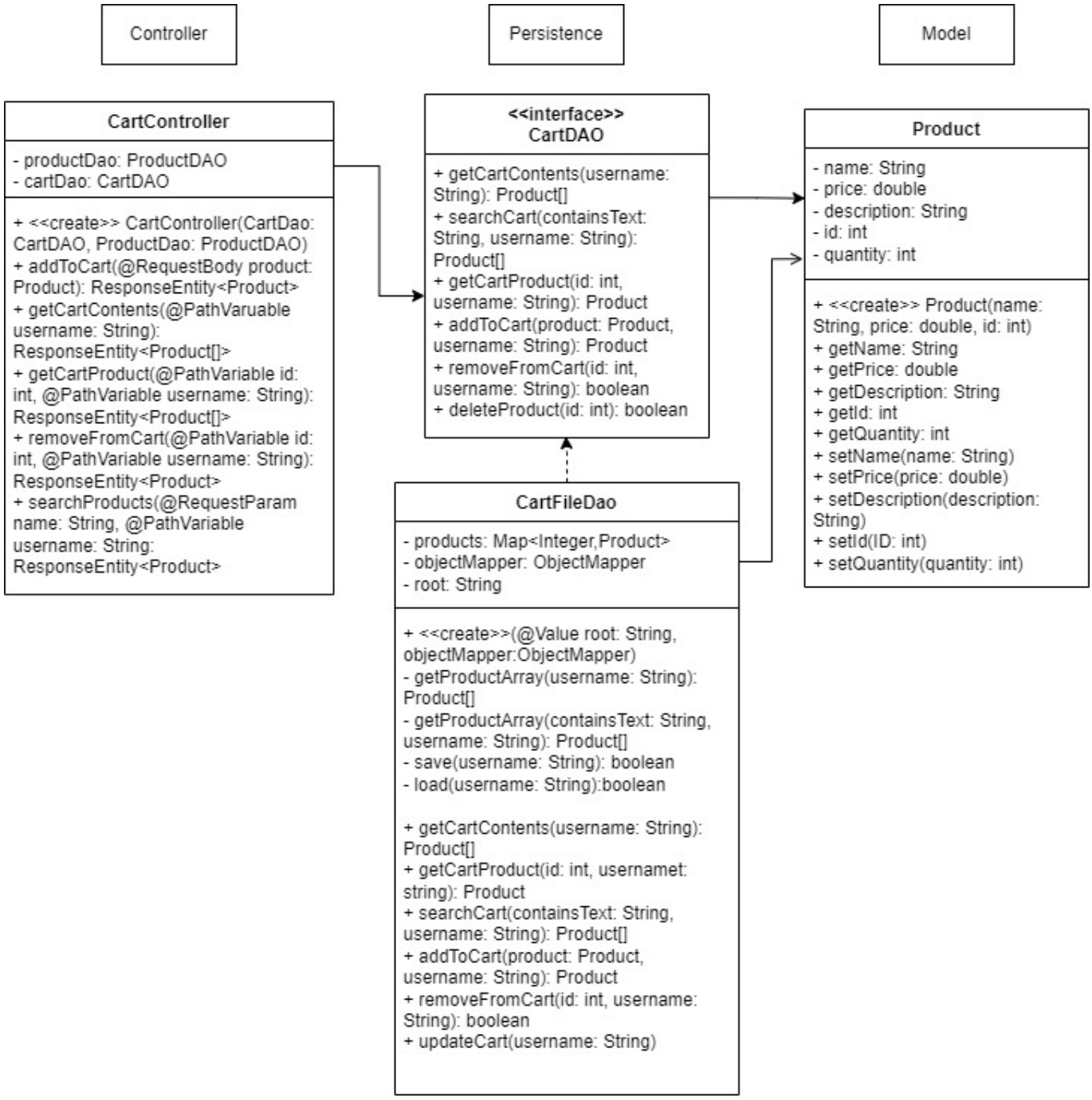
> *At appropriate places as part of this narrative provide **one** or more updated and **properly labeled** static models (UML class diagrams) with some details such as critical attributes and methods.*

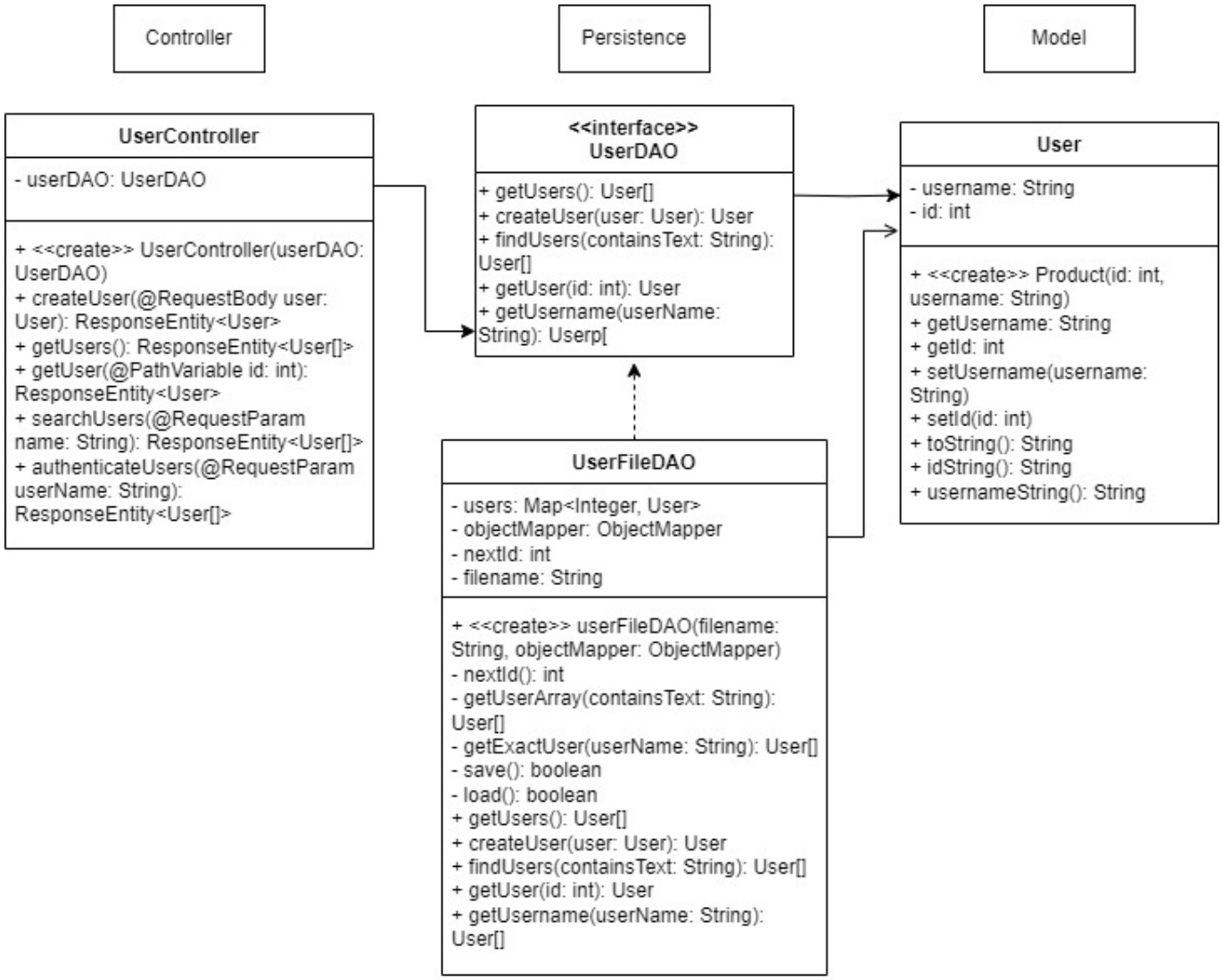
Replace with your ViewModel Tier class diagram 1, etc.

Model Tier

The following UML class diagrams provide an overview of how the model tier interacts with the other tiers of the program.


Controller
Persistence
Model
ProductController
- productDao: ProductDAO
+ <<create>> productController(productDao: ProductDAO)
+ getProduct(@PathVariable id: int): ResponseEntity<Product>
+ getProducts(): ResponseEntity<Product[]>
+ searchProducts(@RequestParam name: String): ResponseEntity<Product[]>
+ createProduct(@RequestBody product: Product): ResponseEntity<Product>
+ updateProduct(@RequestBody product: Product): ResponseEntity<Product>
+ deleteProduct(@PathVariable id: int): ResponseEntity<Product>
<<Interface>>
ProductDAO
+ getProducts(): Product[]
+ findProducts(Text: string): Product[]
+ getProduct(id: int): Product
+ createProduct(product: Product): Product
+ updateProduct(product: Product): Product
+ deleteProduct(id: int): boolean
Product
- name: String
- price: double
- description: String
- id: int
- quantity: int
+ <<create>> Product(name: String, price: double, id: int)
+ getName: String
+ getPrice: double
+ getDescription: String
+ getId: int
+ getQuantity: int
+ setName(name: String)
+ setPrice(price: double)
+ setDescription(description: String)
+ setId(ID: int)
+ setQuantity(quantity: int)
ProductFileDAO
- products: Map<Integer,Product>
- objectMapper: ObjectMapper
- nextId: int
- filename: String
+ <<create>>(@Value filename:String, objectMapper:ObjectMapper)
- nextId: int
- getProductArray():Product[]
- getProductArray(contains:String): Product[]
- save(): boolean
- load():boolean
+ getProducts(): Product[]
+ findProducts(Text: string): Product[]
+ getProduct(id: int): Product
+ createProduct(product: Product): Product
+ updateProduct(product: Product): Product
+ deleteProduct(id: int): boolean


Controller
Persistence
Model
CartController
- productDao: ProductDAO
- cartDao: CartDAO
+ <<create>> CartController(CartDao: CartDAO, ProductDao: ProductDAO)
+ addToCart(@RequestBody product: Product): ResponseEntity<Product>
+ getCartContents(@PathVaruable username: String): ResponseEntity<Product[]>
+ getCartProduct(@PathVariable id: int, @PathVariable username: String): ResponseEntity<Product[]>
+ removeFromCart(@PathVariable id: int, @PathVariable username: String): ResponseEntity<Product>
+ searchProducts(@RequestParam name: String, @PathVariable username: String: ResponseEntity<Product>
<<interface>>
CartDAO
+ getCartContents(username: String): Product[]
+ searchCart(containsText: String, username: String): Product[]
+ getCartProduct(id: int, username: String): Product
+ addToCart(product: Product, username: String): Product
+ removeFromCart(id: int, username: String): boolean
+ deleteProduct(id: int): boolean
Product
- name: String
- price: double
- description: String
- id: int
- quantity: int
+ <<create>> Product(name: String, price: double, id: int)
+ getName: String
+ getPrice: double
+ getDescription: String
+ getId: int
+ getQuantity: int
+ setName(name: String)
+ setPrice(price: double)
+ setDescription(description: String)
+ setId(ID: int)
+ setQuantity(quantity: int)
CartFileDao
- products: Map<Integer,Product>
- objectMapper: ObjectMapper
- root: String
+ <<create>>(@Value root: String, objectMapper:ObjectMapper)
- getProductArray(username: String): Product[]
- getProductArray(containsText: String, username: String): Product[]
- save(username: String): boolean
- load(username: String):boolean
+ getCartContents(username: String): Product[]
+ getCartProduct(id: int, usernamet: string): Product
+ searchCart(containsText: String, username: String): Product[]
+ addToCart(product: Product, username: String): Product
+ removeFromCart(id: int, username: String): boolean
+ updateCart(username: String)



The product class is used for any product that the store admin wants to add to the store inventory. It contains information about the product such as its name, price, and description. An id is associated with each product in order to make sure each product is treated differently by the system.

Additionally, each user is also associated with an instnace of the User class. The user class contains the user's username and the unique id connected with the that username.

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.*

> ***[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.*

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.*

code_coverage

There are no major anomalies for the unit test code coverage.