

Wooden Home Decor e-store Design Documentation

Team Information

- Team name: Phoenixes
- Team members
 - Mohammad Hadi Elabed
 - Khaled Al Dasouki
 - Aya Alqawasmi
 - Hrishikesh Kumar
 - Maksim Rashchupkin

Executive Summary

Our project is a Wooden Home Decor store, which is designed for our Client who is an experienced Wood Artisan. The store will allow customers to buy the products our client produces.

Purpose

The Wooden Home Decor e-store project will be a website that allows the owner to create and manage an inventory of wooden home decorations crafted by a skilled Japanese carpenter. Customers can browse the available products and manage their cart to decide what items they would like to purchase.

Glossary and Acronyms

Term	Definition
SPA	Single Page
WHD	Wooden Home Decor
DAO	Data Access Object

Requirements

This section describes the features of the application.

Definition of MVP

The MVP will consist of accounts (for customers and an owner) that manage an inventory system which stores products. Customers can browse products and add them to shopping carts, which can then be used to checkout and purchase the products.

MVP Features

- Account and Login System
- Product Browsing, Searching and Viewing
- Inventory Management
- Shopping Cart System

- ## Roadmap of Enhancements

1. View, Track and Manage Orders
2. Location Addresses associated with an account
3. Account password
4. Wishlist system

Application Domain

[illegible]

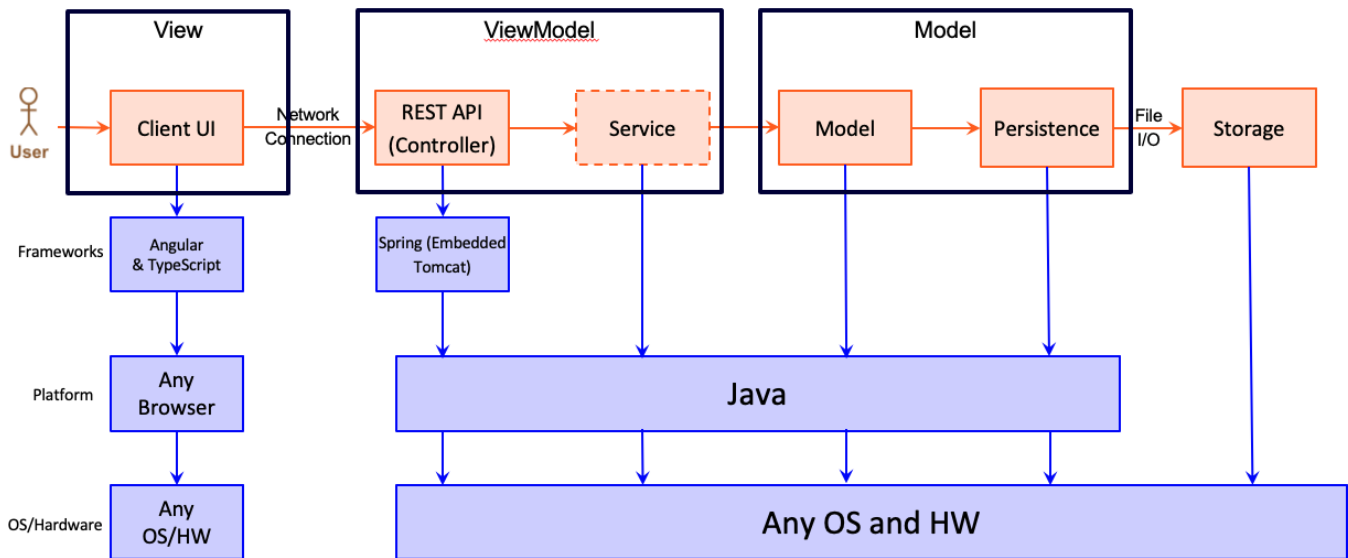
The estore will have accounts that represent the customers and the owner. Customers are able to browse products (the WHD) and add them to a cart while the owner is able to manage the inventory and products.

Architecture and Design

This section describes the application architecture.

Summary

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



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

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

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

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

Overview of User Interface

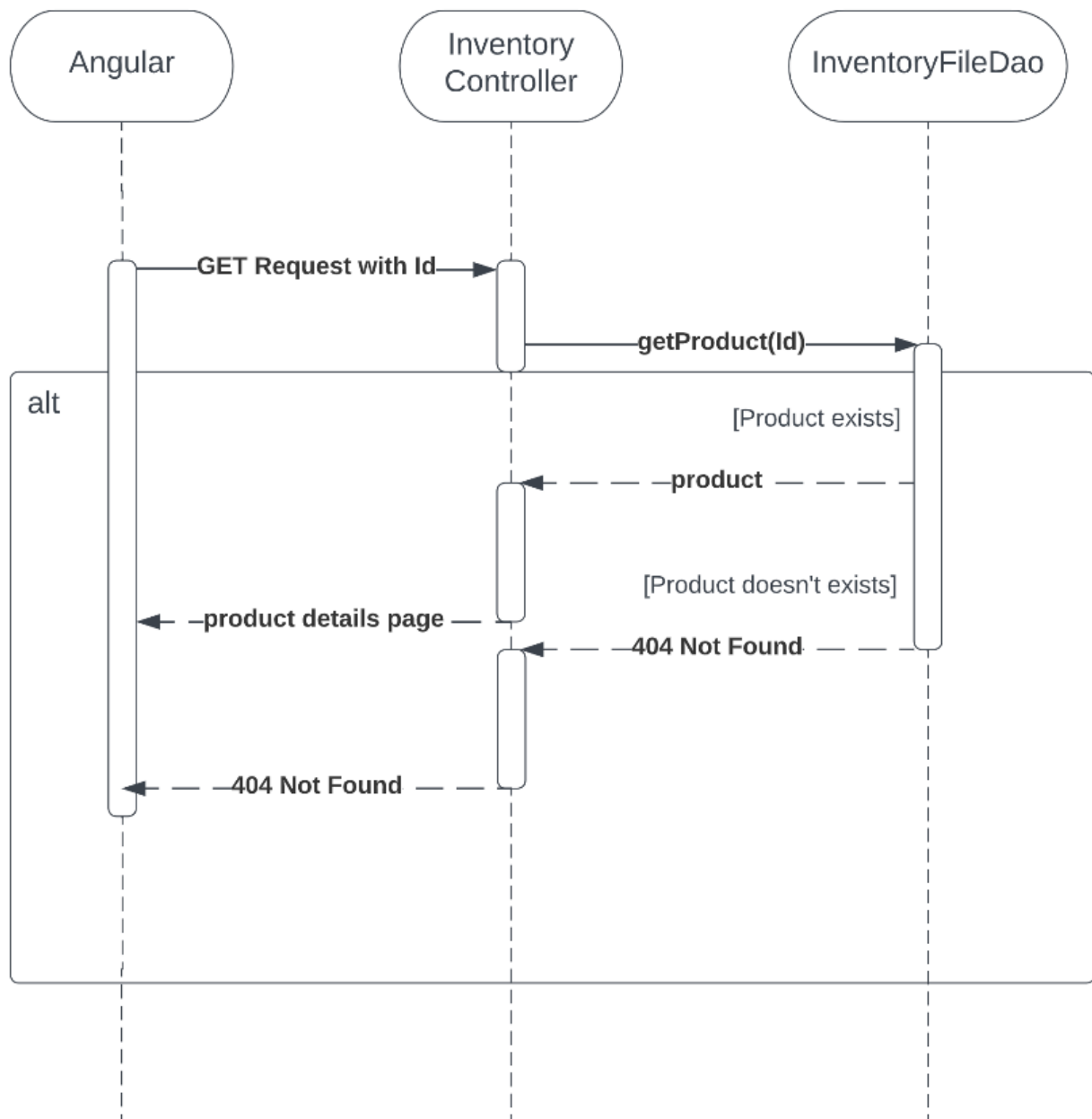
The system describes the applications user interface.

View Tier

Users will be required to login before entering the website dashboard, which shows a list of top products, a search bar to search for products by name, and the navigation buttons. The user can navigate to the products page, sending a GET request, to see a list of all products, the owner/admin of the website will also be able to delete or add products from this page by sending DELETE or POST requests respectively.

When a user clicks on a product, they send a GET request and go to the product details page, where they can see the details of the product they want. Admins will be able to send PUT requests in order to update the products using input boxes.

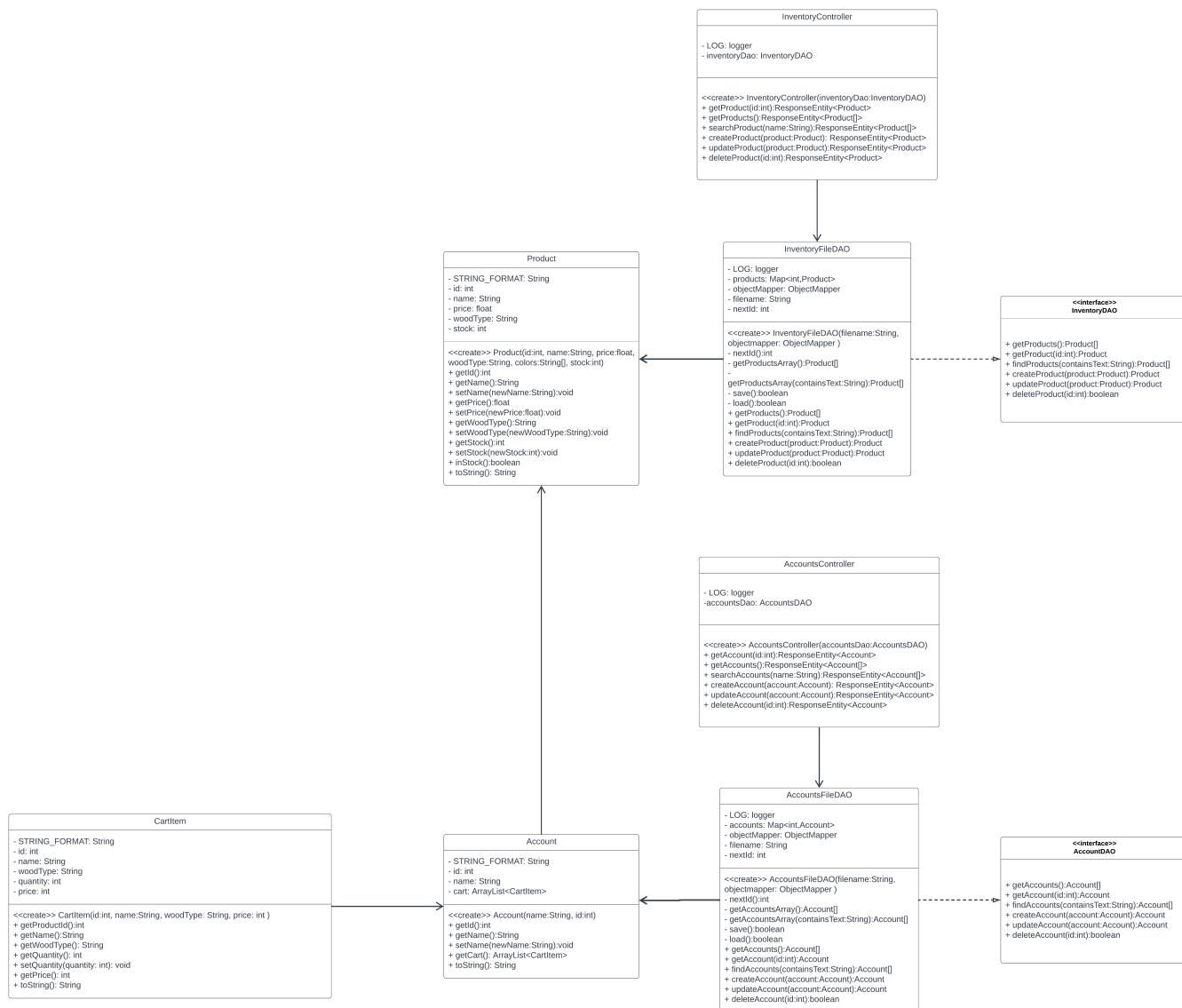
The following sequence diagram shows the process of a user adding an item to cart



Users can navigate to their shopping cart where they will have a list of all the products inside their cart, which they can remove from. Once a customer is satisfied with their cart they can press the checkout button which takes them to the checkout screen, which includes a list of the products they're going to purchase, and buttons to select the payment method. The customer then chooses whether or not the order is a gift for someone, writes the delivery address and finally creates the order.

ViewModel Tier

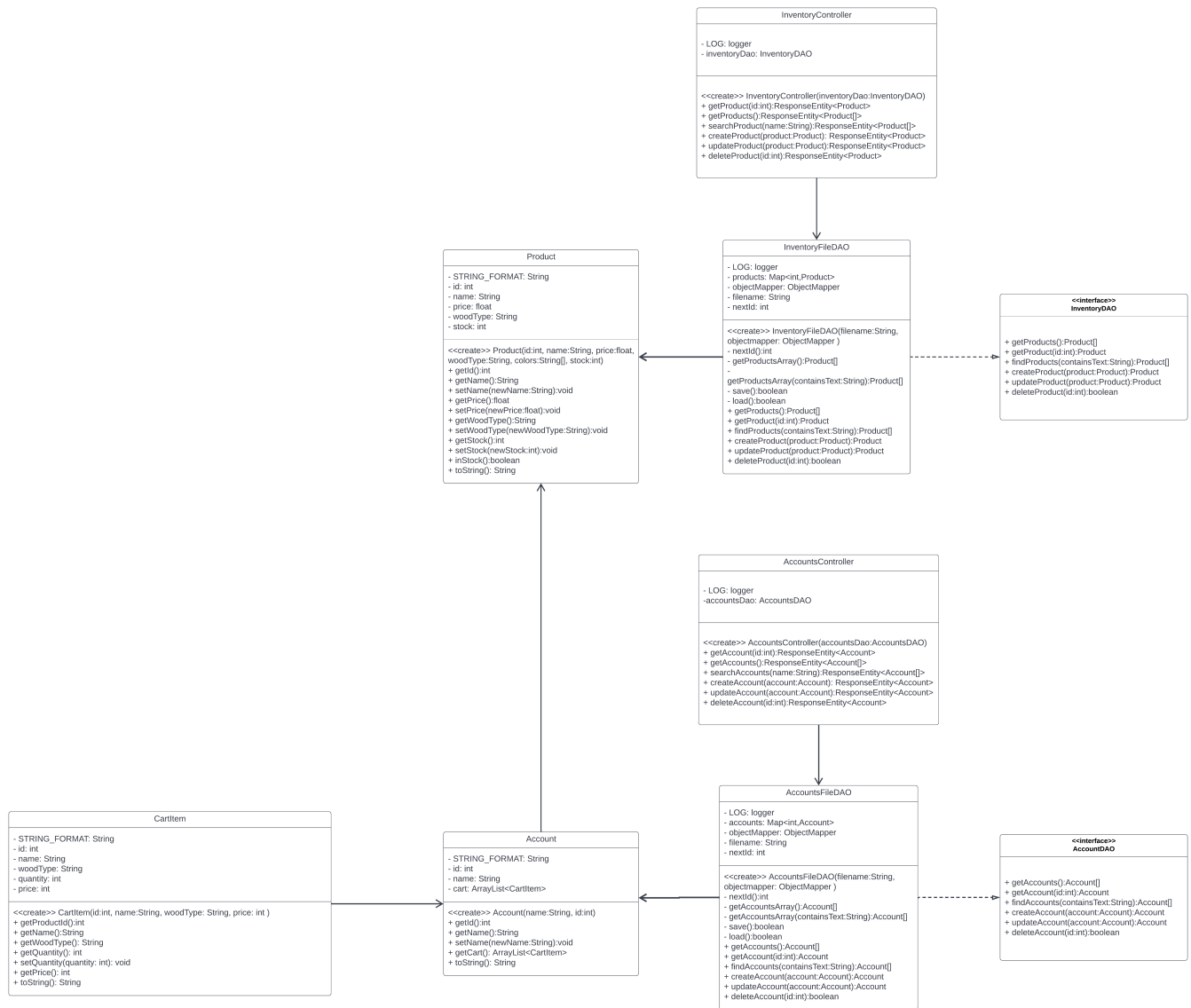
When users send HTTP requests, the controller layer acts accordingly and calls the DAO files which are responsible for communicating with the persistence layer, updating the JSON files or grabbing the information the user wanted (ex. a single product's details).



Model Tier

Users create and use Accounts to access the website and keep their data persistent across sessions, all operations related to the accounts are done through the **AccountsFileDao**. Accounts now have carts as an **ArrayList** of **CartItem**s, which take a name and id of a product, but track the selected woodtype, quantity, and total price (quantity x individual price)

Products each have their own properties that are stored in the Inventory JSON file, which is accessed by the **InventoryFileDao** whenever a user would like to get the details of a product for example, or if an owner would like to manage the inventory.



Static Code Analysis/Design Improvements

Our Design applies many of the OO Design Principles:

1. Single Responsibility: All our classes have one sole purpose.
2. Low Coupling: Our design draws only the necessary relationships between different classes and keeps the design as simple as possible.
3. Information Expert: Classes each have methods that deal with their information rather than relying on a different class to gather/manipulate that information.
4. Dependency Inversion: Java classes are injected in the appropriate place. Typescript services are also injected in the UI part.
5. Controller: Our system uses 2 controllers to manage coordinate actions dealing with accounts and products in the backend.
6. Law of Demeter: methods and attributes are only accessible to the classes that need them in both Java and Typescript parts of the application.
7. Pure Fabrication: FileDAO's are responsible for getting information as well as updating it, maintaining and supporting the Single Responsibility and Low Coupling principles.

An important feature that we should implement if the project is to continue would be security, as for this project we are only requiring a username to login, meaning that there is no password to protect users from

entering each others accounts. We could also further improve the data persistence for certain situations, for example currently the stock of a product is reduced when it's added to cart, when it should be deducted when an order is created.

Another improvement could be adding an account management screen for the admin so they can manage accounts, for example changing account names or deleting them. Currently this is only possible with CURL/PostMan

Testing

This section describes the preformed testing and the results.

Acceptance Testing

All completed user stories have had their testing completed successfully, with no failures or issues being encountered.









1	User Story	Acceptance Criterion	Sprint 2	Tester initials; date; comments (required if test failed)	Sprint 3	Tester initials; date; comments (required if test failed)
2	As a user I want to get a single product so that I can view the information of that product	Given I click on a product when the product exists then I should be directed to that products page and shown it's information	Pass	MR 22/10	Pass	MR 20/11
3	As a user I want to see a list of all products on the website so that I can browse them and decide what I want to buy	GIVEN I click on the products list WHEN I'm not on the products list page THEN I should be redirected to the products list page GIVEN I click on the products list WHEN I'm the products list page THEN I should stay on the page	Pass	AA 22/10	Pass	AA 20/11
4	As a user I want to search for products so that I can find products related to what I want more easily	GIVEN that I input an search term to be searched WHEN items with the search term in their name exist THEN the website should display the list of those products GIVEN that I input an search term to be searched WHEN no items with the search term in their name exist THEN the website should show a message indicating that there are no search results	Pass	HK 22/10	Pass	HK 20/11
5	As an owner I want to create a new product so that it becomes available for purchase on the website	GIVEN I enter attributes of a product WHEN no product with the given name exists THEN the system should create the product, add it to the inventory, save to the persistent store, and now show it on the user interface products list GIVEN I enter attributes of a product WHEN a product with the given name already exists THEN the system should display a message indicating an error	Pass	KA 22/10	Pass	KA 20/11
6	As an owner I want to submit a request to delete a single product so that it is no longer in the inventory.	GIVEN I click a button WHEN the product exists THEN the system should delete the product from the inventory, update the persistence file and display a message indicating it was successfully deleted	Pass	KA 22/10	Pass	KA 20/11
7	As an owner I WANT to submit a request to update a product in the inventory SO THAT I can change a property of the product	GIVEN I enter new properties for a product WHEN the properties are valid THEN the product should be updated in the inventory and the system should display a message indicating it was successfully updated GIVEN I enter new properties for a product WHEN the properties are invalid THEN the product should not be updated and the system should display a message indicating that the product was not updated successfully	Pass	ME 22/10	Pass	ME 20/11
8	As a buyer I want to add products to my shopping cart so that I register what I want to purchase.	Given that I press "add to cart" when the product is not in my cart and I'm logged in then the product should appear in my cart	Pass		Pass	HK 20/11
9	As a buyer I want to remove products from my shopping cart so that I remove what I don't want to purchase.	Given that I press "remove from cart" when the product is in my cart then the product should be removed from the cart	Pass		Pass	AA 20/11
10	As a buyer I want to purchase products in my shopping cart so that I receive them.	Given that I press "Checkout" when there are products in my cart then I should proceed to the payment system	Pass		Pass	ME 20/11
11	As a buyer I want to order products in my shopping cart as gifts so that I send them to another person with nice decorations.	Given that I try press "Checkout" when there are no products in my cart then I receive a message indicating that there's no items in my cart Given that I try tick "Order is a gift" when entering the order details then the order should be considered a gift order	Pass		Pass	ME 20/11
12	As a user I want to choose a wood type on the product so that it's something I like.	Given I choose a wood type when it's in stock then the product can be added to cart	Pass		Pass	KA 20/11
13	As a user I want to logout so that I'm no longer logged in	Given I press logout when I'm logged in then I expect to be logged out	Pass		Pass	MR 20/11
14	Given I submit my username when my username is not in the system then I expect to be prompted to create an account	Given I submit my username when my username is in the system then I expect to be logged in and moved to the homepage	Pass		Pass	KA 20/11
15	Given I submit my username when my username is not in the system then I expect to be prompted to create an account	Given I submit my username when my username is not in the system then I expect to be prompted to create an account	Pass		Pass	KA 20/11

Unit Testing and Code Coverage

All Unit Tests pass successfully.

Below is the Code Coverage for the whole program:

















estore-api

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.estore.api.estoreapi.controller		91%		83%	4	28	8	98	0	16	0	2
com.estore.api.estoreapi		88%		n/a	1	4	2	7	1	4	0	2
com.estore.api.estoreapi.model		99%		50%	1	31	0	49	0	30	0	3
com.estore.api.estoreapi.persistence		100%		90%	3	42	0	112	0	26	0	2
Total	42 of 1,272	96%	8 of 58	86%	9	105	10	266	1	76	0	9

Controller

Both account and inventory controllers have the exact same code coverage found below:

AccountsController

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
updateAccount(Account)		66%		100%	0	2	3	8	0	1
searchAccounts(String)		84%		50%	1	2	1	8	0	1
createAccount(Account)		96%		75%	1	3	0	9	0	1
getAccount(int)		100%		100%	0	2	0	8	0	1
deleteAccount(int)		100%		100%	0	2	0	7	0	1
getAccounts()		100%		n/a	0	1	0	5	0	1
AccountsController(AccountsDAO)		100%		n/a	0	1	0	3	0	1
static { ... }		100%		n/a	0	1	0	1	0	1
Total	18 of 219	91%	2 of 12	83%	2	14	4	49	0	8

The missing branches are also the same, the first one being no tests for a search with no results:

```

84.     @GetMapping("/")
85.     public ResponseEntity<Account[]> searchAccounts(@RequestParam String name) {
86.         LOG.info("GET /accounts/?name="+name);
87.
88.         try {
89.             Account[] accountsArray = accountsDao.findAccounts(name);
90.             if (accountsArray != null)
91.                 return new ResponseEntity<Account[]>(accountsArray, HttpStatus.OK);
92.             else
93.                 return new ResponseEntity<>(HttpStatus.NOT_FOUND);
94.         }
95.         catch(IOException e) {
96.             LOG.log(Level.SEVERE, e.getLocalizedMessage());
97.             return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
98.         }
99.
100.     }
101.

```

The second being if an exception is thrown while trying to update a product/account:


```
142.     @PutMapping("")
143.     public ResponseEntity<Account> updateAccount(@RequestBody Account account) {
144.         LOG.info("PUT /accounts " + account);
145.
146.         // Replace below with your implementation
147.         try {
148.             Account updatedAccount = accountsDao.updateAccount(account);
149.             if (updatedAccount != null)
150.                 return new ResponseEntity<Account>(updatedAccount, HttpStatus.OK);
151.             else
152.                 return new ResponseEntity<>(HttpStatus.NOT_FOUND);
153.         }
154.         catch(IOException e) {
155.             LOG.log(Level.SEVERE,e.getLocalizedMessage());
156.             return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
157.         }
158.     }
159. }
160.
```

To improve the test coverage

EstoreAPI

The code coverage for the EstoreApiApplication because we do not have a test for the main() method, which as far as we can tell is untestable since it just starts the application:

```
estore-api > com.estore.api.estoreapi > EstoreApiApplication.java
```

EstoreApiApplication.java

```
1. package com.estore.api.estoreapi;
2.
3. import org.springframework.boot.SpringApplication;
4. import org.springframework.boot.autoconfigure.SpringBootApplication;
5.
6. @SpringBootApplication
7. public class EstoreApiApplication {
8.
9.     public static void main(String[] args) {
10.         SpringApplication.run(EstoreApiApplication.class, args);
11.     }
12.
13. }
```

Model

The model tier has almost perfect code coverage:

```
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
Product	<div><div></div></div>	98%	<div><div></div></div>	50%	1 14	0 19	0 13	0 1
CartItem	<div><div></div></div>	100%	n/a		0 9	0 16	0 9	0 1
Account	<div><div></div></div>	100%	n/a		0 8	0 14	0 8	0 1
Total	1 of 237	99%	1 of 2	50%	1 31	0 49	0 30	0 3

The only missing branch not being covered is for the `inStock()` method inside the `Product` class which returns true or false:

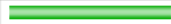
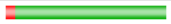


```
53.    /**inStock returns true if a product is in stock and false otherwise*/
54.    public boolean inStock(){return this.stock > 0;}
```

To improve the code coverage here we should add a test for checking that `inStock()` returns false when an item is out of stock.

Persistence

The persistence tier has perfect testing:

com.estore.api.estoreapi.persistence

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
InventoryFileDAO		100%		93%	1	21	0	58	0	13	0	1
AccountsFileDAO		100%		87%	2	21	0	54	0	13	0	1
Total	0 of 554	100%	3 of 32	90%	3	42	0	112	0	26	0	2

The untested branches are untestable since they are just single case scenarios (if condition is met do something, otherwise nothing), below is an example of this:

```
78.    private Account[] getAccountsArray(String containsText) { // if containsText == null, no filter
79.        ArrayList<Account> accountsArrayList = new ArrayList<>();
80.
81.        for (Account account : accounts.values()) {
82.            if (containsText == null || account.getName().toLowerCase().equals(containsText.toLowerCase())) {
83.                accountsArrayList.add(account);
84.            }
85.        }
86.
87.        Account[] accountArray = new Account[accountsArrayList.size()];
88.        accountsArrayList.toArray(accountArray);
89.        return accountArray;
90.    }
91.
```