PROJECT Design Documentation

Team Information

- Team name: Toronto Maple Leafs
- Team members
  - Domenic Lo Iacono
  - Niccolls Evsseef
  - Claire Kreisel
  - Ming Creekmore

Executive Summary

This project is a full stack application including a persistent storage, backend, and user interface. The goal of this application is to host a jersey store for the Toronto Maple Leafs. As an online store, the admin is able to manage the site by adding and removing jerseys, editing existing jerseys, and other site features. Furthermore, users that visit the site are able to browse through the jerseys available for sale, view specific jersey's attributes like cost, search for jerseys, and lastly add items to their shopping cart and buy the items in that cart.

Purpose

This website has an admin which is able to modify the stock of jerseys that are for sale by adding jerseys, removing jerseys, and editing existing jerseys. The website also has users which have a username and a shopping cart which is persistent after logout. Users are able to add and remove items from their shopping cart.

Glossary and Acronyms

Term	Definition
Admin	Website owner with special privlidges related to the inventory of jerseys.
User	A customer that interacts with the website which logs into the website and has a username and shopping cart.
Jersey	The type of product that is being sold at the E-Store
SPA	Single Page
DAO	Data access object

Requirements

This section describes the features of the application.

The application allows for the admin and user to log in and for new users to register for their account. It allows for the admin to preform the CRUD operations on a stock of jerseys to which the users will able to browse and view and add to their shopping cart. It allows for the user to add and delete items from their

shopping cart and 'buy' the items once they are ready. The website also allows quick access to every product via a search bar.

Definition of MVP

The admin and users must be able to login. New users are able to register new accounts. The admin must be able to preform CRUD operations on the stock of jerseys which is persistently stored. The user must be able to add, delete, and checkout their shopping cart which also must be persistently stored. The website must allow quick and easy access to the products with a postive user experince.

MVP Features and the names of their stories

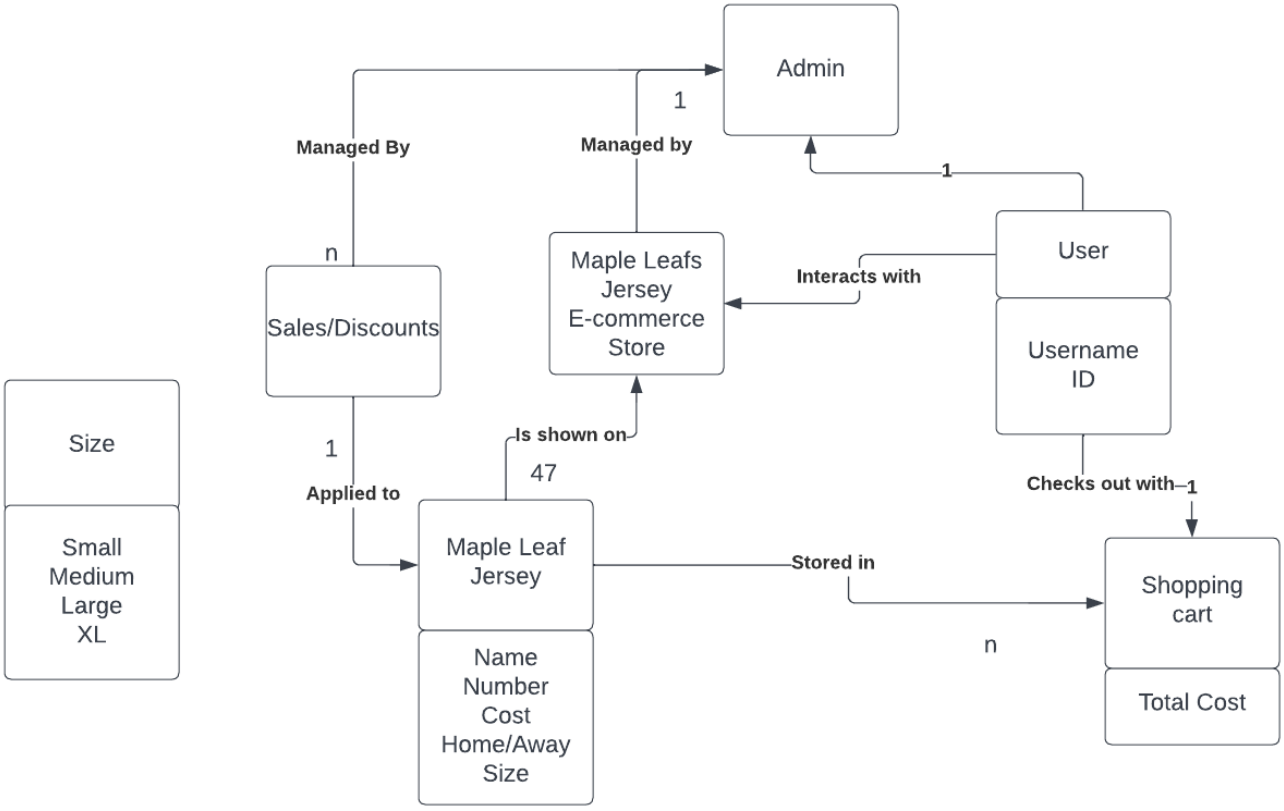
Login Admin and its users Register new users Admin Add jerseys Admin Delete jerseys Admin Update jerseys User Add to cart User Delete from cart User Checkout cart Search jerseys and view

Roadmap of Enhancements

Login Admin and its users Register new users Admin Add jerseys Admin Delete jerseys Admin Update jerseys User Add to cart User Delete from cart User Checkout cart Search jerseys and view

Application Domain

This section describes the application domain.



This is a Noun/Verb analysis relating to our domain model which is now represented in code through various classes.

Nouns: Jerseys Number Name Home/away Cost Size XS Small Medium Large XL

Store

User Admin Customer

Shopping Carts Number of items Total cost

Sales/discounts

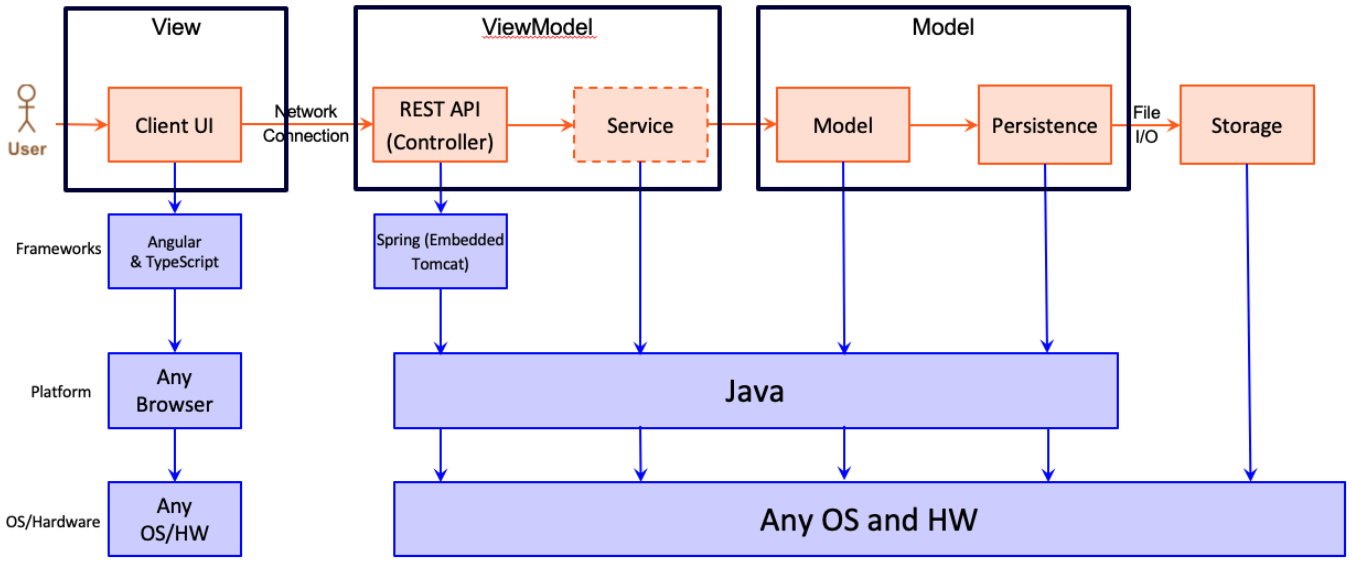
Verbs: Adding (Jerseys to store inventory and shopping cart) Editing (Jerseys to store inventory) Deleting (Jerseys from store inventory and shopping cart) Searching for jerseys Browsing the site/jerseys Checkout the shopping cart

Architecture and Design

Using Spring boot, the backend has persistent storage and a consistent model of objects using the controller, model, and persistence setup. Jerseys and Users have their respective controller, model, and persistence files which allow for operations and storage through JSON files. The frontend is built using angular and through the jersey.service.ts and user.service.ts files are able to access the backend data and use the data to deliver a UI that can support persistent storage.

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 user and admin have almost identical flows through the application. The user is only different than that of the admin by the fact that they are unable to see the CRUD operation buttons when viewing a jersey and that they have a shopping car. Any person first lands on a login page when visiting the site. Upon login, after registration if neccessary, a user is directed to a page that displays the stock of jerseys that can be filtered using the search bar, a shopping cart icon, a logout icon, and the username of whoever is currently logged in. From here they are able to view jerseys and add them to their cart. They are also able to view their shopping cart and remove items from cart, or buy all the items in the cart.

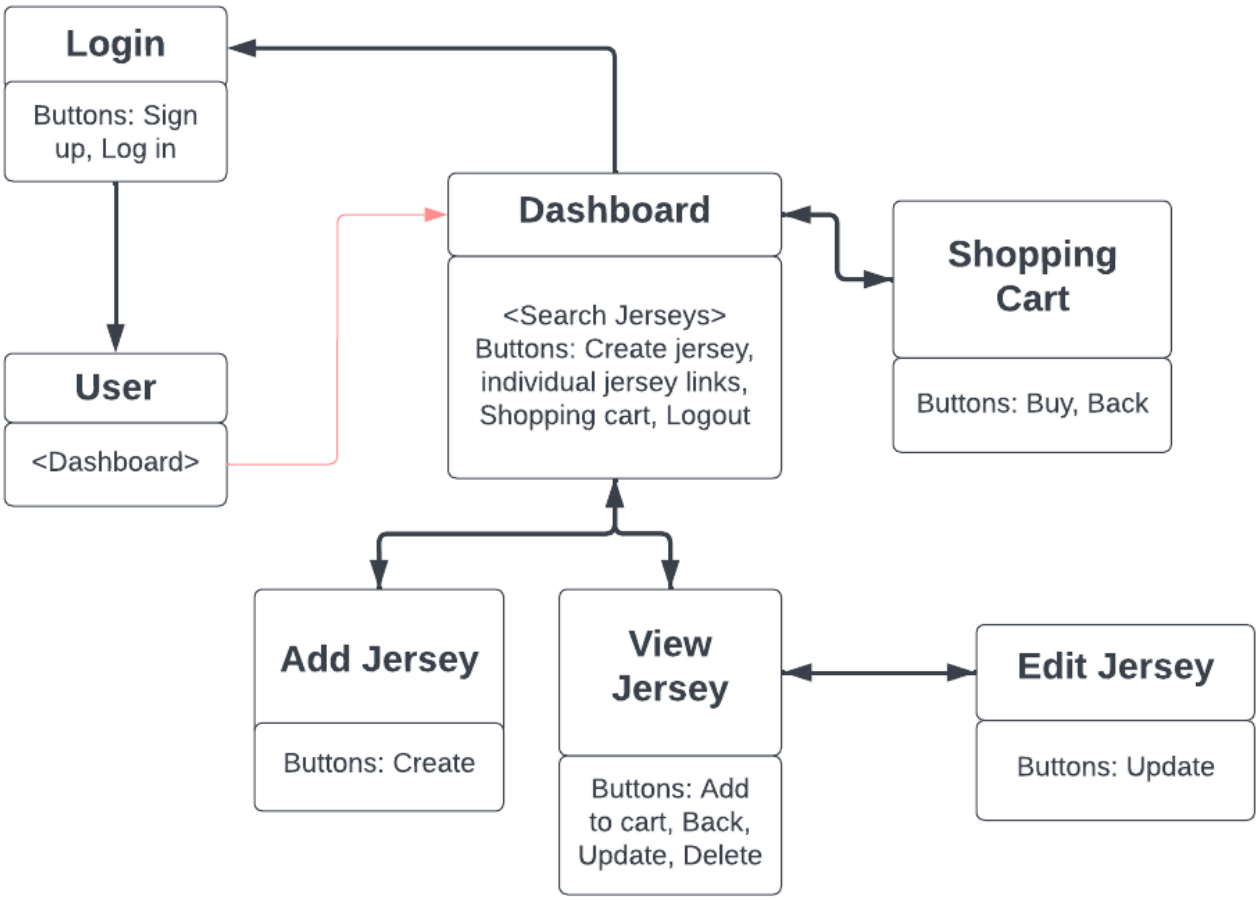
As stated before, admin has an almost identical website experience as a user. The difference is that on the browsing jersey page, there is no shopping cart. Instead, there is a create jersey button. Furthermore, when viewing an individual jersey, they can edit or delete it. They cannot buy an individual jersey because they don't have a shopping cart.

View Tier

Note: The app-components admin and browse-jersey are not used in the web-app. The components in the tier that are used include login, user, dashboard, search-jerseys, add-jersey, view-jersey, shopping-cart, and edit-jersey.

Login is where the user is directed to when they first enter the website. It has a username and login screen with basic authentication. When the user is authenticated, then the app routes to the user component, which routes to the dashboard component. The dashboard component consists of the search-jerseys component, an add jersey button, a shopping cart button, and logout button. These buttons route to the respective component names. The search-jerseys component all jerseys that fit the filtered name, with spaces returning all jerseys. Each jersey name displayed, routes to that specific jerseys view-jersey page.

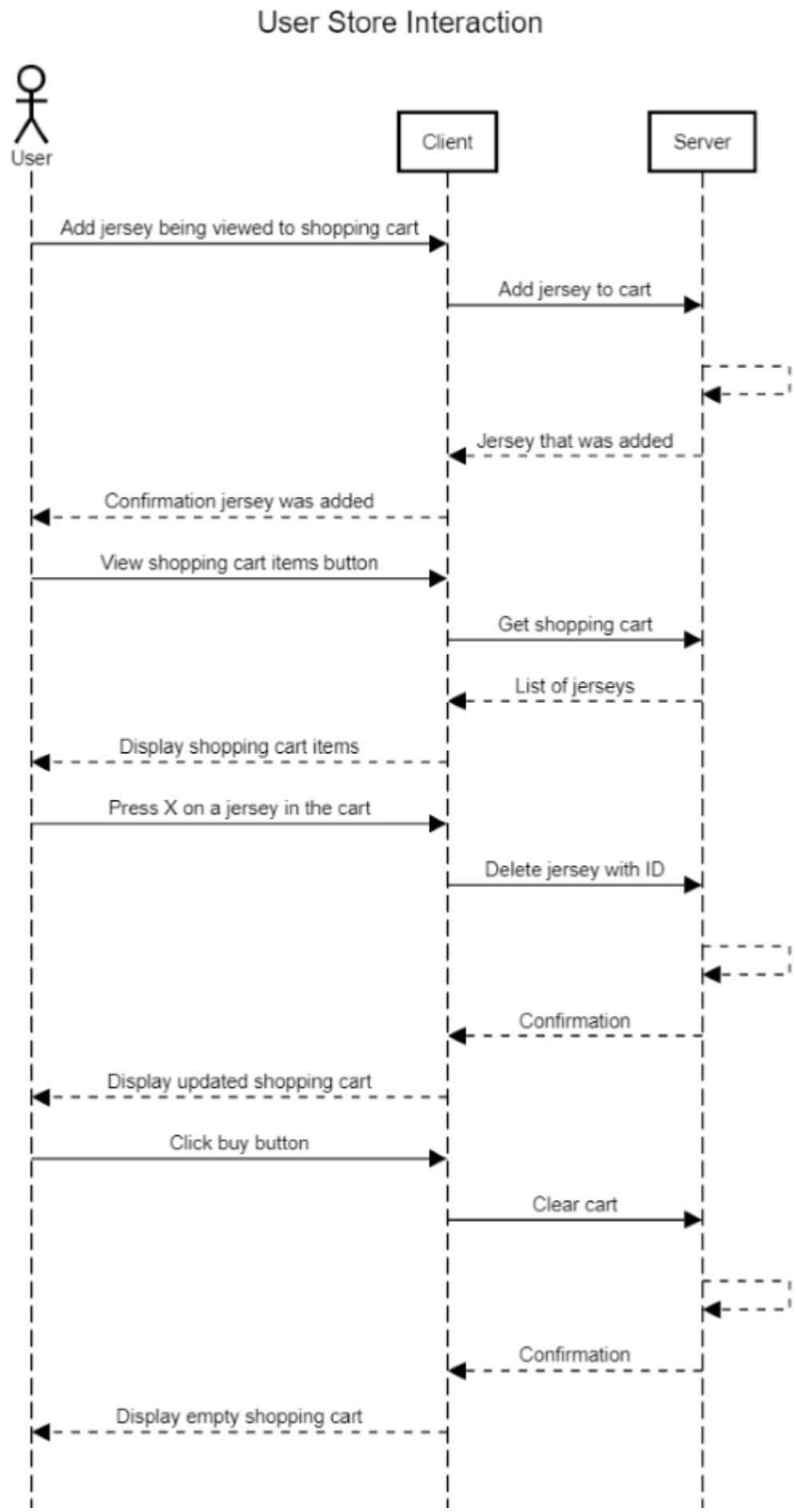
The view jersey page consists of the buttons: Back, Add to Cart, Update, and Delete. Update routes to edit-jersey component, then back to view-jersey. The shopping cart component consists of the buttons, Back and Buy. The diagrams below show the high-end diagram of the view tier routing just described

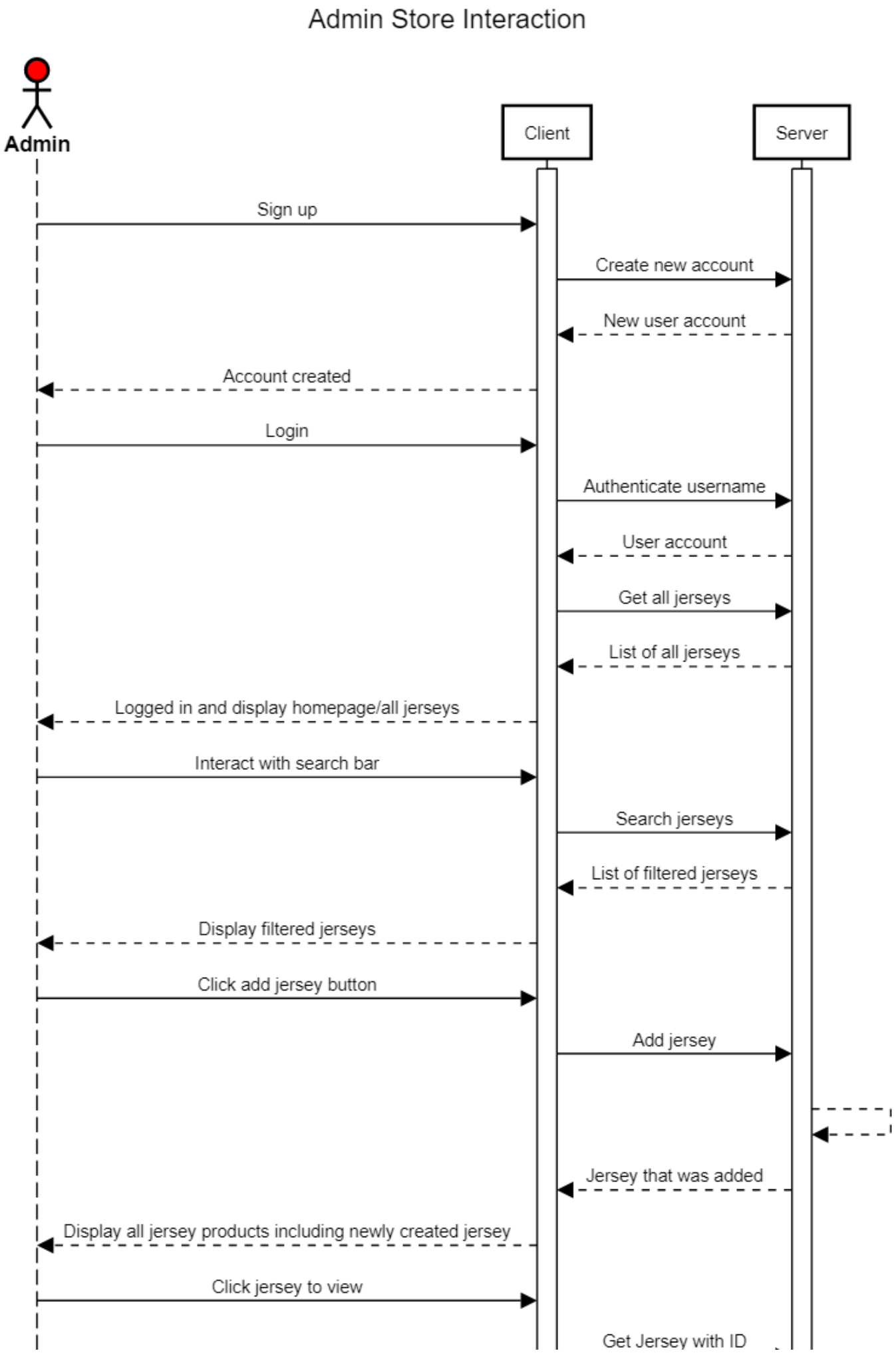


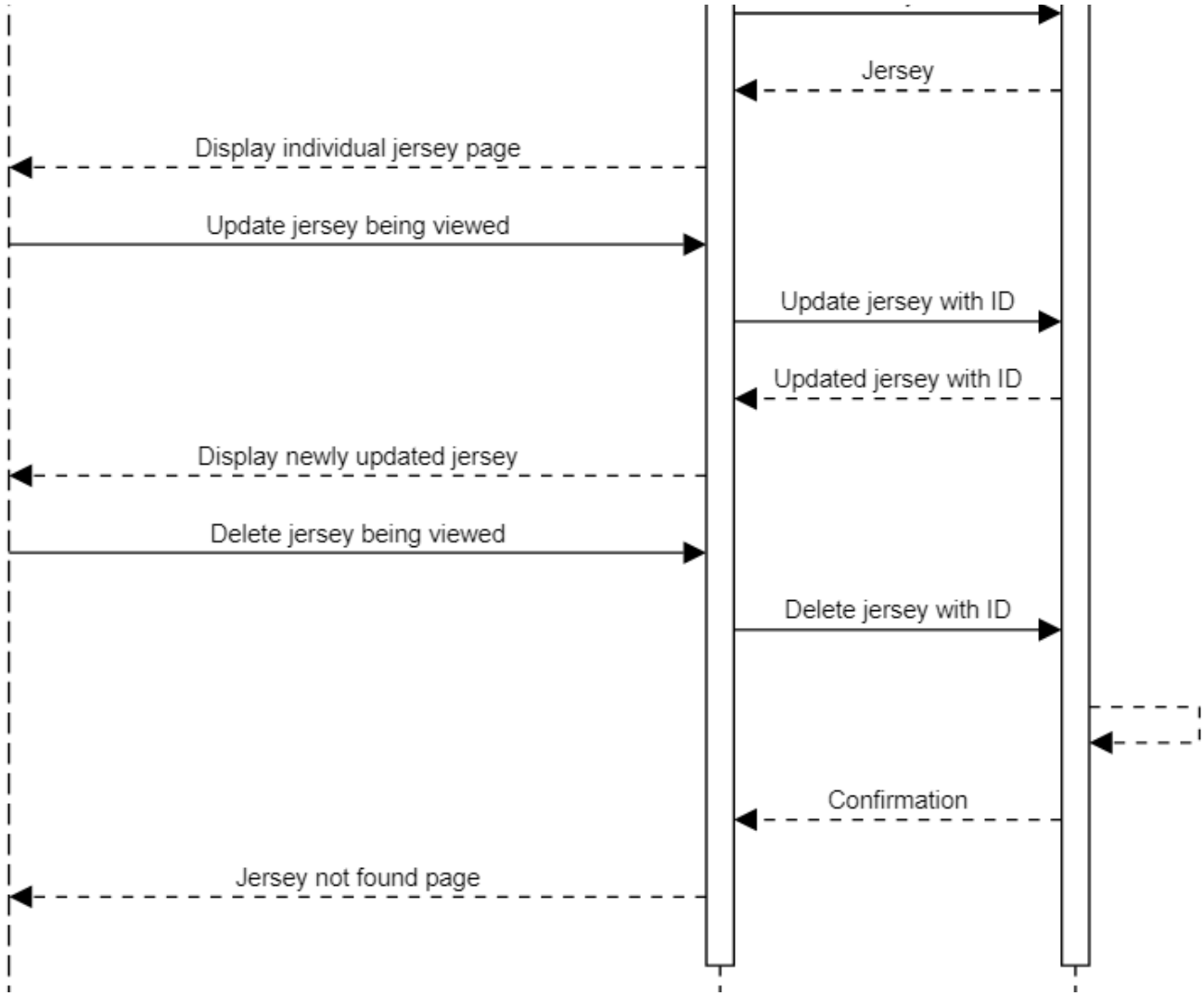
ViewModel Tier

This tier involves the interaction between the client and the server. The user will first interact with the website, which is the client. This results in the client giving a call to the API server, which returns a response entity. Based on the response entity the client will update the website view accordingly.

The client is composed of the angular components that represent each individual html page as well as the user and jersey service. The user and jersey service are what call the API. The API server is composed of the UserController and the JerseyController. The user parts deal with logging in the user and updating the shopping cart of the specified user. The jersey parts deal with updating the inventory. Below are two sequence diagrams that show how the admin and user interact with the server and client. The user sequence diagram does not include logging in, searching jerseys, and viewing jerseys because it is the same as what is shown in the admin sequence diagram.


User Store Interaction
User
Client
Server
Add jersey being viewed to shopping cart
Add jersey to cart
Jersey that was added
Confirmation jersey was added
View shopping cart items button
Get shopping cart
List of jerseys
Display shopping cart items
Press X on a jersey in the cart
Delete jersey with ID
Confirmation
Display updated shopping cart
Click buy button
Clear cart
Confirmation
Display empty shopping cart


Admin Store Interaction
Admin
Client
Server
Sign up
Create new account
New user account
Account created
Login
Authenticate username
User account
Get all jerseys
List of all jerseys
Logged in and display homepage/all jerseys
Interact with search bar
Search jerseys
List of filtered jerseys
Display filtered jerseys
Click add jersey button
Add jersey
Jersey that was added
Display all jersey products including newly created jersey
Click jersey to view
Get Jersey with ID


Jersey
Display individual jersey page
Update jersey being viewed
Update jersey with ID
Updated jersey with ID
Display newly updated jersey
Delete jersey being viewed
Delete jersey with ID
Confirmation
Jersey not found page

Model Tier

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

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

Static Code Analysis/Design Improvements

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

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

Testing

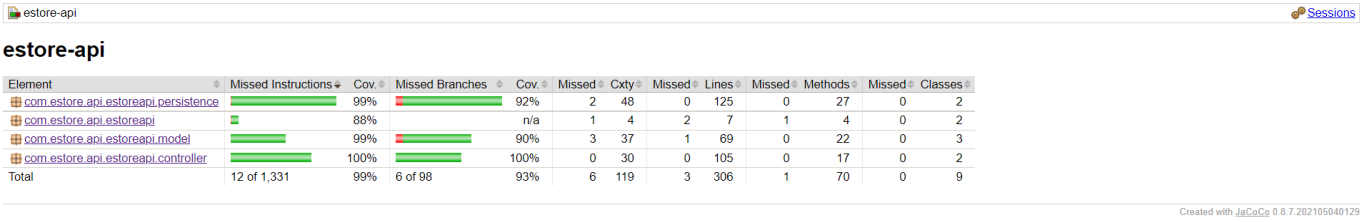
Testing was done using JaCoCo test coverage and JUnit tests. No testing was done on the UI component besides people going through the website.

Acceptance Testing

Acceptance criteria was made in the form GIVEN some precondition WHEN I do some action THEN I expect some result. 15 user stories have passed all their acceptance criteria tests. 3 user stories each have one test failing. These include, search jerseys, edit jerseys, and checkout. Search jerseys requires the user to input a space in the search bar before all jerseys are displayed. Edit jerseys does not have the form autofilled, so the admin could accidentally fill out the form wrong by not filling in the previous data on the form, causing a blank jersey to be created. Checkout does not display the correct total cost of the jerseys. 2 user stories have not been tested because they are still on the sprint backlog.

Unit Testing and Code Coverage

Unit testing was done by creating JUnit tests for each class. Each public method in a class was tested, where different branching routes of the method were tested via individual testing methods. Code coverage was calculated by JaCoCo. We selected a coverage target of 95% because we wanted most of the paths tested. The reason why it isn't 100% is because we felt that some of the paths added that we missed were part of the private methods that we didn't test and other paths tested instanceOf. We were able to meet this code coverage result for persistence, model, and controller. Estore-API code coverage percentage was not met; it had a code coverage of 88%. This is because this was pre-provided for us and we didn't want to touch it.


estore-api

Sessions

estore-api

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.estore.api.estoreapi.persistence		99%		92%	2	48	0	125	0	27	0	2
com.estore.api.estoreapi		88%		n/a	1	4	2	7	1	4	0	2
com.estore.api.estoreapi.model		99%		90%	3	37	1	69	0	22	0	3
com.estore.api.estoreapi.controller		100%		100%	0	30	0	105	0	17	0	2
Total	12 of 1,331	99%	6 of 98	93%	6	119	3	306	1	70	0	9

Created with JaCoCo 0.8.7.202105040129