PROJECT Design Documentation

Team Information

- Team name: Big-Development
- Team members
  - Ryan Current
  - Chris Hamm
  - Michael Ververs
  - Andrew Leon
  - Colin Tondreau

Executive Summary

This website will allow users to buy health tracking status for falcons. It will also allow users to donate.

Purpose

Admins will be able to manage products, users, and inventory and users will be able to purchase inventory and manage their accounts. Customers will be able to browse and search products and birds, sponser birds, view health reports, and recieve notifications. Ornithologists will create health reports. Together, this store will function to connect owners and falcon enthusiests.

Glossary and Acronyms

Term	Definition
SPA	Single Page
API	Application Programming Interface

Requirements

Admins are able to manage all inventory, orders, products, users,notifications, and health reports. Users are able to manage their profile and create orders. Users are able to see health reports. Ornithologists can create health reports.
Everyone is able to see notifications

Definition of MVP

Admins are able to manage all inventory, orders, products, and users. Users are able to manage their profile and create orders.

MVP Features

All

- sign in capability
- view all inventory capability

- search all inventory capability

Admin

- CRUD operations for products.
- Read and update operations for orders.
- CRUD operations for inventory.
- CRUD operations for users.

Users

- adding items to cart
- checking out/creating an order/cancelling an order

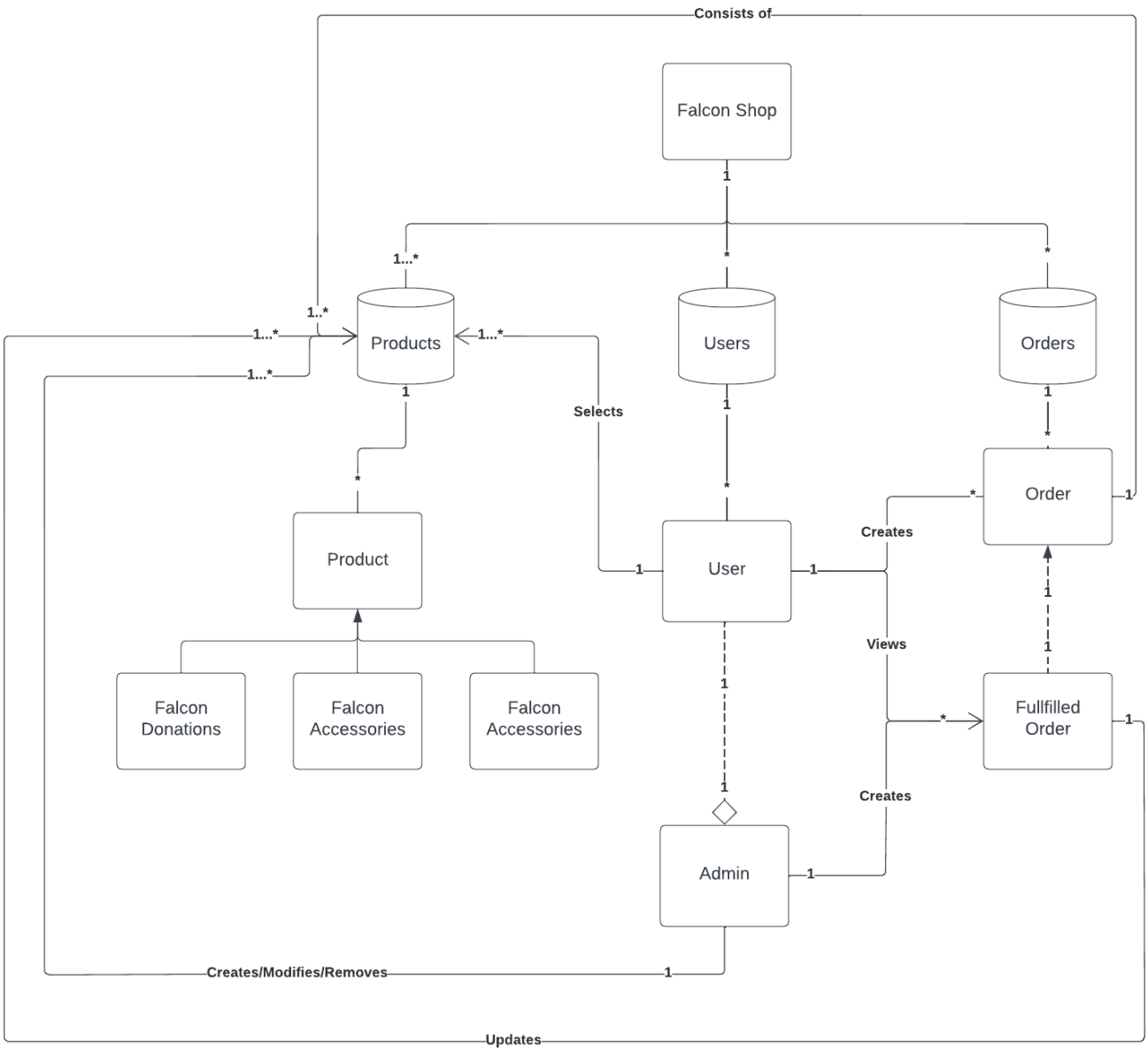
Roadmap of Enhancements

As a team we all picked one data type to work with. From there we started by defining them in the backend, and then moved to the frontend. These data types included:

- Users
    - Address
    - CreditCard
    - Cart
- Products
    - ImageSource
- Orders
    - OrderItem
    - OrderStatus

Application Domain

This section describes the application domain.



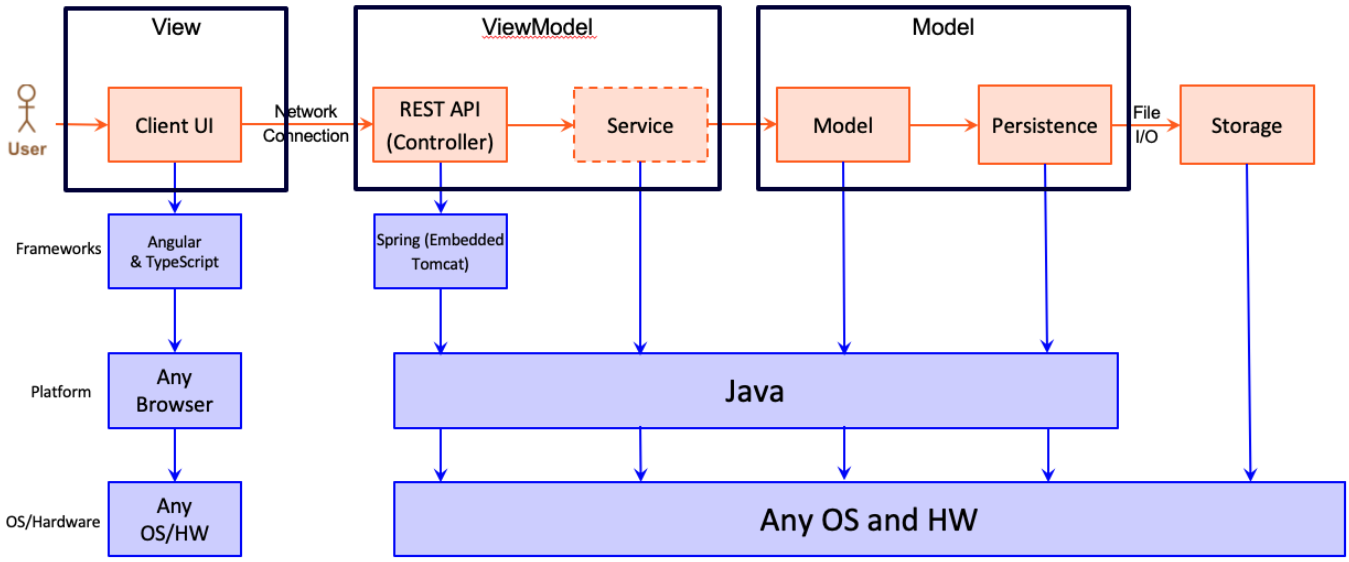
The main data types are Users, Products, and Orders. Users have different access levels to Products and Orders. Overall, Users will create, read, update, and delete Products, Orders, and themselves to make this estore work.

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

Our UI is split into 5 main pages:

- A store page where users can browse and search for products.
- A sign in page for users to sign in and manage their information.
- A cart page (only available to customers) where customers can create orders.
- A product view page where product details and health reports can be viewed
- An administrator/ornithology console, where admins/ornithologists can manage inventory, products, users, and orders.

View Tier

The view tier of Falkoner consists of 5 main pages, starting the user on a non-main page, the landing page describes the goal of the site. From there the user is navigated to the home page through interacting with a button component, that is used consistently throughout the site. The home page allows the user to see all products in the inventory and navigate to see a singular product, via the product detail page. From there, if the user is logged in they can add the product that they are viewing to their cart, if they are not then the user is directed to login before adding said product to their cart. On the login page they are able to login, create an account, update their accounts information and see all previous orders made with Falkoner. Once logged in the user is able to view the products that are apart of their cart and place an order if all of their information is provided. The last main page is only accessible to an administrative user, at the current moment there is only one administrative account that has access to the Admin Dashboard. From here the admin is able to perform CRUD operations on all users, orders, notificatons and products all from the same page.

> *You must also provide 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. Be sure to include an relevant HTTP reuqests from the client-side to the server-side to help illustrate the end-to-end flow.*

ViewModel 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.*

The ViewModel tier handles Aplication Program Interface (API) requests made to our Spring Framework Server. This tier contains the controller which encompasses the business logic which may validate and or manipulate any data before updating our model (Model) tier. This tier also contains the Services that provides mapping to our client (View) to handle get, post, update, and delete requests which are made to the controller. Our estore offers the following controllers for their respective models:

```
Controllers:
  - ProductController
  - OrderController
  - UserController
  - HealthReportController
  - NotificationController
```

Within our Estore UI, we also offer the following services: Services: - Product Service - Order Service - Cart Service - User Service - HealthReport Service - Notification Service Controllers and Service all handle the following operations: - Create (Post) - Read (Get) - Update (Put) - Delete (Delete)

Model Tier

The Model is responsible for saving data and handling the data after the service requests it after an event has occurred. Such examples include the User model, such as when a user logins/creates an account, data must be filled in by the user, and the data is checked and set ,ensuring usability of the data. The Product model likewise sets the data types of a product's attributes, however this model has more responsibilities and functionality when it comes to CRUD operations considering the fact the Product model is an abstract class used to create different type of Product, but with the same attributes.

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

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

Acceptance Testing

There are no remaining user stories that have failed acceptance criteria tests. All user stories pass their acceptance criteria, and all aspects of the store work as intended.

Unit Testing and Code Coverage

Our unit testing goal was to reach at least 95% coverage and all tests passing. If a particular test did not pass we fixed our code to make it pass. We targeted all models and controllers of the api to ensure that we had a solid backend.