PROJECT Design Documentation

Team Information

- Team name: StraightFlush
- Team members
  - Ryan Yocum
  - Clay Rankin
  - Reid Taylor
  - Nate Appleby

Executive Summary

This project involves a working E-Store that allows users to buy, search and view cards and related products, standard users are capable of creating accounts to login and purchase products, while also maintaining cart information for their next visit. Admin users are capable of creating/editting/deleting products, along with base user capabilities. Finally with a user interface to facilitate all of these actions.

Purpose

The purpose of this project is to provide an online store with a user interface, important goals were to have a working login system, displaying products, and allowing users to checkout and complete transactions with a cart containing products.

Glossary and Acronyms

Term	Definition
SPA	Single Page Application
API	Application Programming Interface

Requirements

The main features of this application included the user authentication system, including user accounts. Allowing users to store data and access admin powers. There is also everything involving products, which includes all of the searching/viewing/creating/editting of products on the frontend/backend. Another major feature are the carts systems, which allows all normal users to save a cart of products that they want to purchase. The 10% feature will include user reviews and displaying those reviews on products with averaged rating.

Definition of MVP

The Minimum Viable Product displays products and allows users to login to accounts. Normal users must have access to a cart with products for transactions. Admin users must be allowed to edit and create products.

MVP Features

User Authentication Epic, including login, account creation. Product Methods Epic, including creation, deletion, search, editting. Cart Methods Epic, including adding to the cart, clearing the cart and getting the cart. Finally the Transaction Epic, which will include transaction functionality and be linked to user proceeding to transaction with a cart of products.

Roadmap of Enhancements

Our initial plan for enhancements included both custom product images and a product rating and review system. However, we significanly underestimated the difficulty of implementing both features, and decided

that between having only product images and only ratings and reviews, product images provided a superior user experience. As of our latest release, the Store Owner can upload images both when creating a new product, and when editing an existing one. Users can view and scroll through the images uploaded for each product, allowing them to see products before they make a purchase.

Application Domain

Domain Model

First we have the Customer, whom has access to a cart in which they can add, edit and view products in the cart, and this cart defines items to be purchased. The cart contains products, which are viewed/searched for by the customer, these products are able to added, viewed, editted or deleted by the product owner, who is an admin user. After a transaction the product owner receives payment from the customer, who receives confirmation of the transaction. Both users must login using Auth, which grants access to user specific data, or admin priviledges if applicable. The user specific data is saved and accessed from a File Database.

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 Tiers & Layers of the 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

Web Application Interface Statechart

The user interface is comprised of 6 pages On the home page, the user has access to both the Home and View Items buttons on the navbar, and the account icon will open a menu that allows any user to log in, register, or logout dependent on whether the user is logged in at first.

View Tier

Diagram The App renders the Topbar component for user navigation, which allows the user to navigate to different pages such as the View Products page, and individual's product page, as well as the login and register pages using links on the page. As an admin, there are edit links on the products that make a request to the backend ViewModel, as well as login and register send a request to log a user in.

ViewModel Tier

Diagram Each distinct section of the Model tier has its own Controller, which provides API endpoints for all of the relevant persistence functions. These Controllers match up with a corresponding Angular Service running on the client, which makes requests to the provided endpoints.

Model Tier

Diagram The Product model includes an id, name, description, price, and quantity. The UserAccount includes an id, username, and a boolean isAdmin

Static Code Analysis/Design Improvements

Diagram

The main area in which our code could be improved is in unity and parity between code written by different developers. Not only do we have different code styles, but we also have different ideas and levels of experience that result in vast differences in implementation. Assigning responsibilities of team members provides some help in mitigating this, but as the required functionality becomes more complex, more connection between classes implemented by different team members is required. Many of the frontend tests in the static code analysis resulted in some false bugs such as i tags that should be em tags, although this is required by the library. There were many unused imports in both the frontend and backend that should have been removed.

Testing

Testing for the program was performed using both unit testing and human functional testing. The potential conditions were layed out and actions were performed with these conditions to confirm the program was working as intended. So far all components passed their tests except a few bugs found related to the carts.

Acceptance Testing

There were 23 implemented user stories and all of which passed their acceptance criteria tests. This number is not representative of the number of features we implemented, as many of the tasks should have been divided up into multiple user stories

Unit Testing and Code Coverage

Each method in the backend is set to unit tests to confirm they correctly return values depending on parameters. All backend methods directly related to stories such as loginUser/getProduct, have passed all their unit tests. Correctly returning values given each condition.