Team Activity: Express-O Coffee Shop

Training Overview

The purpose of this activity is to get team members acclimated to working in teams and experiencing what development is like on a team. This will demonstrate the value of Git and expose apprentices to a larger code base than we have previously worked with in training. This experience will give us context and foundations to build on in the longer group project later in training when we build more professional processes into the function of the team. The manner in which this project is structured also lays the groundwork for layered architecture and concepts coming in the web services module.

Previous exercises had requirements laid out in a very clear cut and somewhat linear pattern which in part was meant to structure how you approach solving problems. This project's requirements are much broader in scope and part of the challenge and learning here will be working with others to figure out the way forward. As always, raise any clarifying questions to your trainers at any time.

Business Overview

Express-O is a proof of concept application that keeps track of coffee drink ingredients, drink recipes, baked goods, customers, and purchases for a local coffee shop. This coffee shop purchases ingredients and makes their own coffee drinks. They do not have a baker in-house, so baked goods are purchased from vendors for resale at a markup. Customers are tracked for a loyalty program. Purchases are tracked for an understanding of sales and revenue vs. costs.

The core features of the program are to manage the following domain objects:

- ingredients
- drinks (recipes)
- baked goods
- customers
- purchases

Grading and Evaluation

- Associates will not receive a grade for this activity, however 360 degree feedback (see the course for details) will be collected for each team member from each of their team members.
- Talking about code and showing off requirements is an important aspect of team based software development. Each team will provide a demonstration where each team member will present their features to the rest of the cohort.

Data Models

Ingredient

- name
- purchasing cost
- unit amount (ie: 1)
- unit of measure (ie: "kg")

Baked Good

- name
- purchasing cost
- markup percentage
- vendor name
- list of known allergens (ex. wheat, peanuts, milk)
- sale price (calculated from purchasing cost + markup percentage when created)

Drink

- name
- ingredients list
- cost to produce
- markup percentage
- sale price (calculated from purchasing cost + markup percentage when created)

Customer

- name
- email address
- lifetime spent (updated whenever that customer's email is attached to a purchase that gets created)

Purchase

- date/time
- list of purchased items
- total cost
- customer

Requirements

1. Each team will create a google doc to use for planning and information sharing, with all team members and trainers added as editors
2. All team members will use git to create development branches
3. Developers will commit / push and pull **often**
4. Completed features will be merged into the main branch by the developers
5. Each domain will have a file dedicated to describing the data model (see below). Using a JavaScript class is not necessary, but this file should have a function that works like a "constructor" that can easily create and return new objects and takes every parameter associated with the model. It does not have to BE a constructor, it just operates like one.
6. Each domain has a repository file associated with it. This file has a list of the domain objects as a member as well as CRUD (create, read, update, delete) operation methods. These methods act directly on the list.
7. Each domain has a service file associated with it. This file will make use of the repository file to get the work done. This file also has methods for each CRUD operation. This is where any business logic needed lives. This file should NOT directly operate on the list. It will call the repository methods in order to get any work done on the list.
8. Validation Requirements
    a. Drink names should be unique (e.g. we should not have two "Hazlenut Latte" in the system)
    b. Customer email address should have the following format and min/max character limits:
        i. username@domain.extension
        ii. username: 1 - 30 char
        iii. domain: 1 - 30 char
        iv. extension: 2 - 15 char
    c. Customer email should be unique.
    d. Purchase date/time should include a timestamp in UTC so that it is consistent across time zones, following the format below:

        YYYY-MM-DD HH:MM:SS TZ
        2023-11-13 16:17:00 UTC

    e. All monetary values should have two decimal places (e.g. 10.99)

Key Points

- The Drink Repository file contains the list of drinks and methods that operate directly on the list. This list will serve as our in-memory database.
- The Drink Service contains any business logic and / or validation logic in order to validate logic.
- The Drink Service does NOT operate directly on the list. Methods in the Drink Service call methods in the Drink Repository in order to operate on the list.
- For example, the drink service should have a method that creates a new drink. This method should take a drink object and validate any attributes necessary according to the above requirements. If the object is valid, then and only then does the repository get contacted to add the valid drink object to the drink list.
- Repeat this pattern for every domain.

Example Diagram