Perry Chambers Jr

Sweng 465

Semester Project

Table of Contents

Project Proposal3

Problem Description3

Solution3

Scope3

Requirements3

Project Tools4

Diagrams4

Use Case Diagrams4

Class Diagram5

Sequence Diagram ……………………………………………………………………………………………………………………………………………..7

**Application Programming Interface (API) ……………………………………………………………………………7**

**Running and Deploying System ………………………………………………………………………………………….8**

I would like to preface that I know that the report was only supposed to be 2-3 pages but I didn’t see that until I had most of this already completed. I hope that I do not lose points for it but I understand if I do. The back-end functionality is 100% complete, the front-end is about 50% but still somewhat functional. All of the instructions for running the front-end are located in this document. If there is anything I can answer for you please let me know.

# Project Proposal

**Problem Description**

In a restaurant, it is a pain to have to keep all of the recipes somewhere and to pull them out in order to create a dish, and causes a mess. I am trying to think of a way to simply add items to the menu, with all of the appropriate information about that menu item, such as, price, calories, ingredients, recipe, etc. This will create one singular point of addition for items to the menu, as well as modification and deletion of these items.

**Problem Solution**

For my project, I want to create a Restaurant Management System. In this management system, the main users are customers, chefs and managers. Each of these users will have different access to the system, with the managers having complete access. The base users: customers will only have access to the menu’s items, and the information of the menu item that is important to them, such as price and calories. Once a customer decides on what they want to order, they can send that order to the kitchen. Once at the kitchen, the chefs can see what was ordered, as well as the ingredients needed and the associated recipe all in one place. The order is removed from the queue when finished. If modifications or additions to the menu are desired, a manager will have access to both of these features. Chefs will also have access to modifying the menu items. A manager will have access to all of the functions that chefs have available to them, besides the cook function which will be exclusive for the chefs.

**Scope**

In order to keep the scope of the project small and reasonable, I will only be doing 1 order at a time, instead of a group of orders going into the kitchen at once. Each menu item that is added, will be created via a factory method to ensure proper creation. Since we are using JAVA, I will be attempting to make the system as reusable and extendable as possible. If time permits, will attempt to connect this web service to an AWS RDS Database for data persistence.

**Requirements**

Initial Requirements:

User:

1. A customer is able to view the entire menu.
2. A customer can order a singular item off of the menu.
3. Managers or Chefs can modify a menu item.
4. Managers can create a new menu item.
5. A manager can delete a specific menu item.
6. Chefs can complete orders present in the order queue.

System:

1. An order should take no longer than 10 minutes to process.
2. A user should be able to retrieve the entire menu in less than 10 seconds.

**Project Tools**

The tools that I intend to use for this project are IntelliJ for development IDE, Trello for organizing my tasks, GitHub for version control, and Insomnia for Endpoint Testing. For organization, I will have to create a class diagram involving the Factory design pattern, a sequence diagram, and possibly an ER Diagram if time permits to add a database.

# Diagrams

**Use Case Diagram**

Diagram

Description automatically generated

**Figure 1: Use Case Diagram**

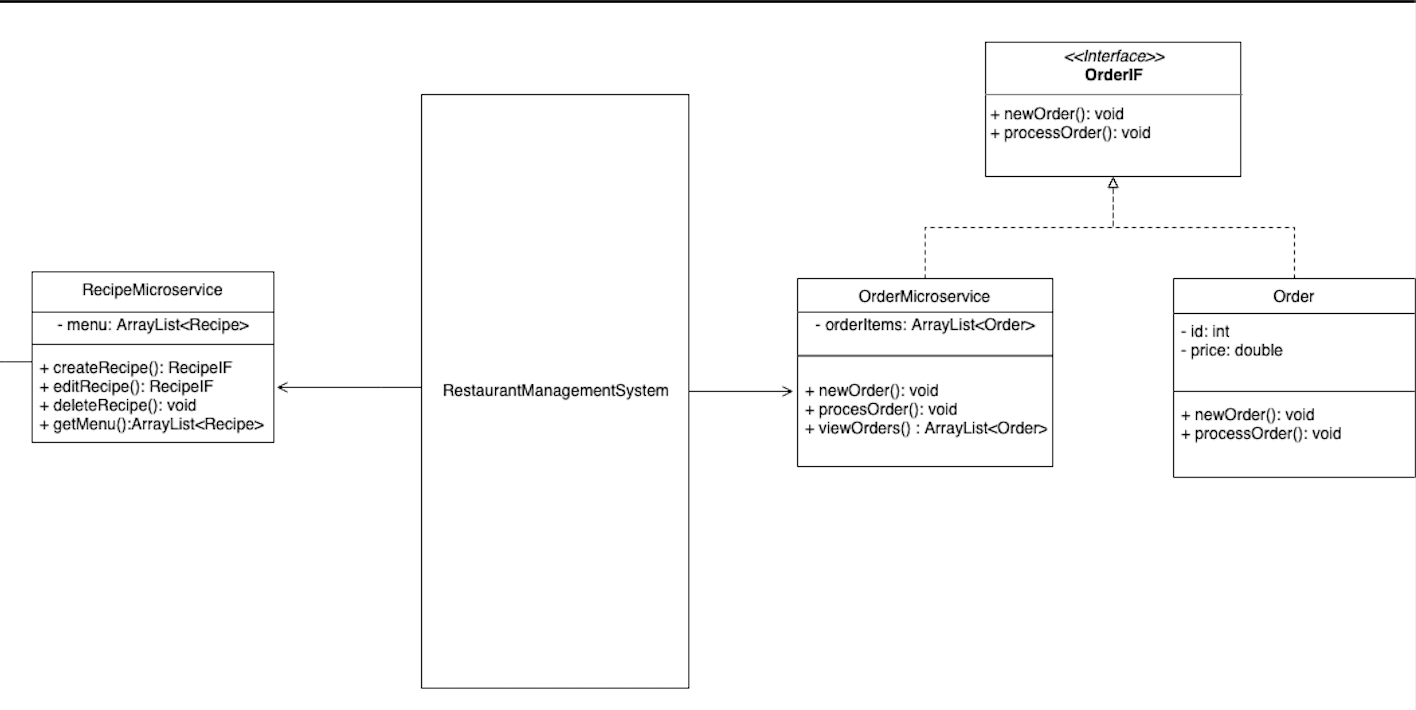
**Class Diagram**

**Diagram

Description automatically generated**

**Figure 2: Factory Design Class Pattern**

Had time permitted, the factory design pattern above would have been implemented and connected to the RecipeMicroservice. Whenever a new recipe would be created, it would be created via the factory.



**Figure 3: Microservice section of the class diagram**

This section of the class diagram is responsible for housing the HTTP communication methods via Spring in their own respective microservices. The recipe microservice will mostly be used by management via the use of the creation of new recipes as well as the modification and deletion of them. When a customer wishes to view the entire menu, they will use the getMenu() method/endpoint inside of the Recipe Microservice. Every method inside of this microservice has a RequestMapping of /recipe/{subsequent method}. Conversely, the Order Microservice has the RequestMapping of /order/{subsequent method}. In this microservice, a customer Is able to create a new order, while the employees working in the kitchen can view all of the incoming orders, as well as process and send out the completed orders (imitated with an HTTP Delete Request). The way that this system is designed allows for further improvement to be made upon it, without having to modify anything inside of the other microservices. An example of this would be if the restaurant owner wished to implement a rewards system. Every time a customer went to pay, they would have an item such as a QR code linking to the back-end endpoint inside of a RewardsMicroservice class called addRewards(@RequestParam(“id”)String id). In this case, only the user’s customer ID would be passed in to then add those points to the customer’s account. If time had permitted, the recipe microservice would have been connected to a database, to allow for permanent storage for the recipes created. For the order microservice, a database is not needed since the orders should only stay in the system for no longer than an hour.

**Sequence Diagram**

Diagram

Description automatically generated

**Figure 4: Sequence Diagram showcasing features**

The sequence diagram above showcases some of the features that will be used most in the system. All of the requests are sent from a client, whether that is a computer in the kitchen or a customer’s phone, they all will be able to use the system. This diagram details the process of an order from creation to being served. First, the client sends a request to the RecipeMicroservice to get the selection of items that can be ordered. In order to modify the menu, a manager is able to create a menu with all of the pertinent information needed to prepare it. Then, once the customer knows what they would like to order, it gets created and sent to the ‘kitchen’ (Order Microservice). This then gets added to a list and once it is done, the chef can serve the order by removing it from the list via the OrderMircroservice’s serveOrder() method. If any other systems such as a reward point system or a checkout system, their lifecycles would occur after these events.

**API**

All of the endpoint URLs are [**http://localhost:8080/**](http://localhost:8080/) **or** [**http://192.168.1.9:8080/**](http://192.168.1.9:8080/)

**Recipe Microservice (/recipe)**

* Get Menu (HTTP GET)
  + /recipe/menu
* Add To Menu (HTTP POST)
  + /recipe/new
  + Takes in a RequestBody of a Recipe Object.
* Update Menu Item (HTTP PUT)
  + /recipe/update?id=x
  + Update recipe with id of x.
* Delete Menu Item (HTTP DELETE)
  + /recipe/delete?id=x
  + Delete’s a recipe in the menu with an id of x.

**Order Microservice (/order)**

* Get orders (HTTP GET)
  + /order/orders
* Place an order (HTTP POST)
  + /order/new?orderItem=”x”
  + Takes in a string representing the name of the menu item to be added.
* Serve an order (HTTP DELETE)
  + /order/serve?id=x
  + Serves (deletes) order with id x from the list of incoming orders.

**Running the System**

Since the front-end client has been developed with Vue.js, a script is needed to run the client. Simply navigate to the client folder (/rms/client). Once the terminal is directed to the correct folder, the user only needs to type “npm run serve”. This command executes the serve function located inside of the package.json file. If this system were to be built for deployment, the command “npm run build” would be ran instead. This command will create a dist package located inside of the root file, which contains all of the built files.

Once npm run serve has finished running, the terminal should similar to below:

Text

Description automatically generated

**Figure 5: Screenshot of terminal for running client**

The addresses listed are for testing purposes, the serve command allows for the client window to automatically refresh whenever the code is saved. Vue utilizes a technology called Vue router which allows for different views to be rendered as a single-page application. These views have an assigned route, i.e., <http://localhost:8080/menu> will display the view that displays the entire menu to a customer. If this system were on a dedicated server and had a domain registered, it would be http://assignedURL/menu.

The Back-End code is created with Spring which is a Java framework that utilizes Maven. Since it is all Java code, the system can be built as a jar, war, or ear file depending on what is needed for deployment. This makes it a portable container that can be ran on any end system, or server. To run the system locally, simply compile and run the java class **RmsApplication**.