foodBuff

Online Restaurant System

Design Report

For Web Application

Version 1.1

Revision History

Date	Version	Description	Author
11/04/19	1.0	Create software requirement specification	Kartikeya Sharma, Varun Chennamadhava, Kazi Siam, Lifu Tao
11/27/19	1.1	Create design report	Kartikeya Sharma, Varun Chennamadhava, Kazi Siam, Lifu Tao

Table of Contents

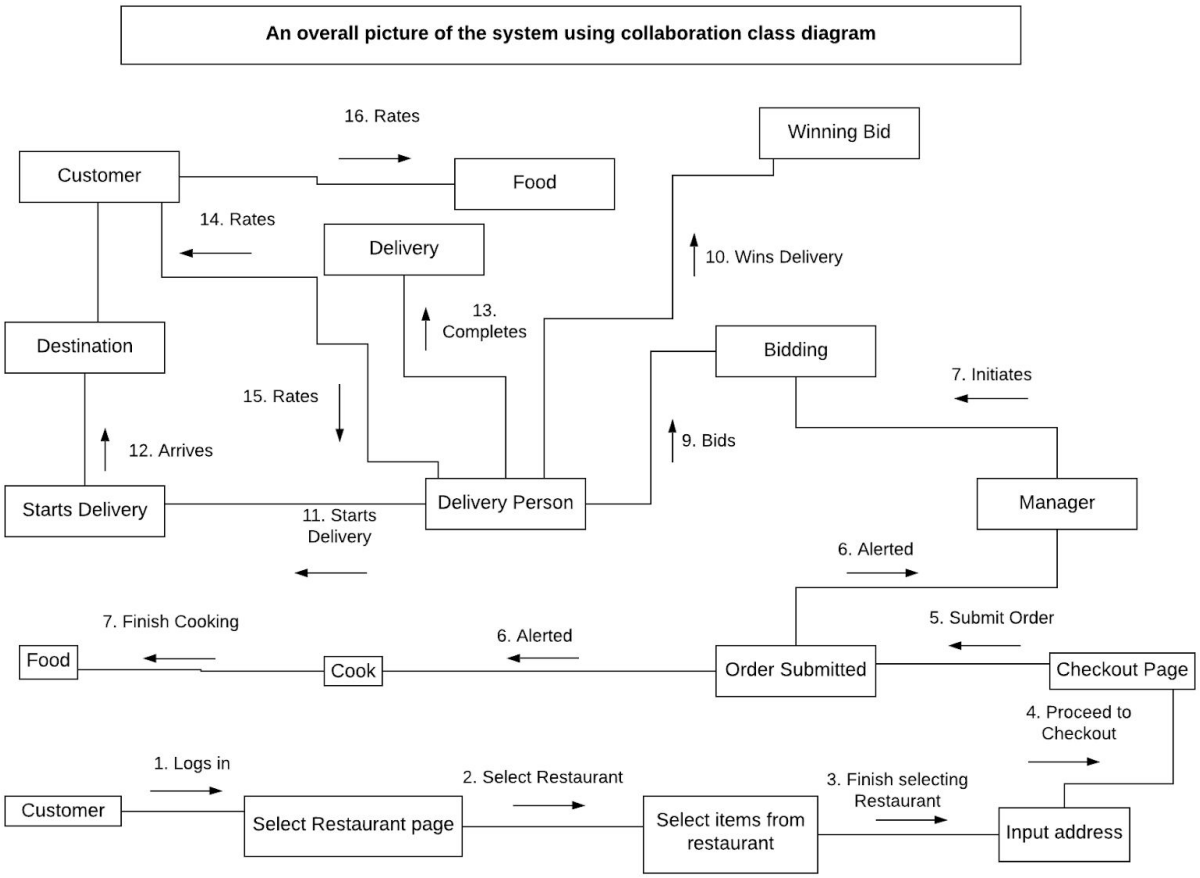
1. Introduction

foodBuff is a food ordering platform where users can order food to their locations with items from multiple restaurants. A rigorous review system ranging from the chef, customer, to the delivery driver, all working together to create the best experience for all parties involved.

1.1 Purpose

The purpose of foodBuff is to provide an easy way for customers to order food from their favorite restaurants and have it delivered to their home. Consumer voices are also heard through a rigorous review system affecting all parts revolved in a truly consumer-centric product.

1.2 Collaboration Class Diagram



2. Use Case Analysis

Customer Scenarios overview:

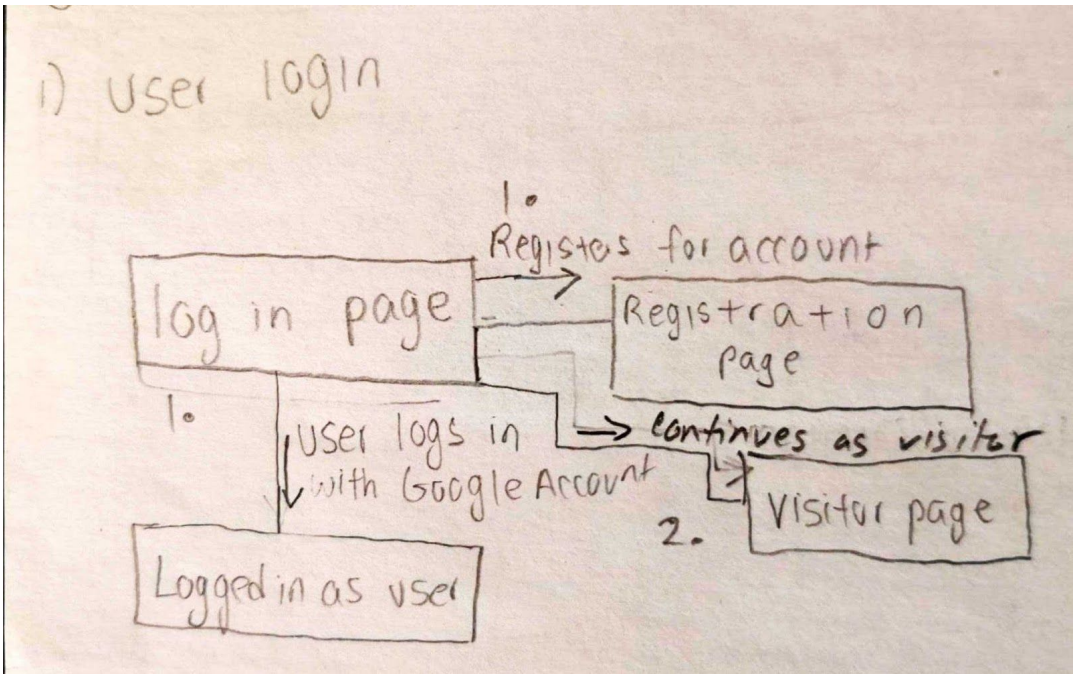
1) User Login
2) Pick a restaurant
3) Selects items from that restaurant
4) Address input
5) Checkout page
6) Customer receives order and rates food and delivery person

1) User login

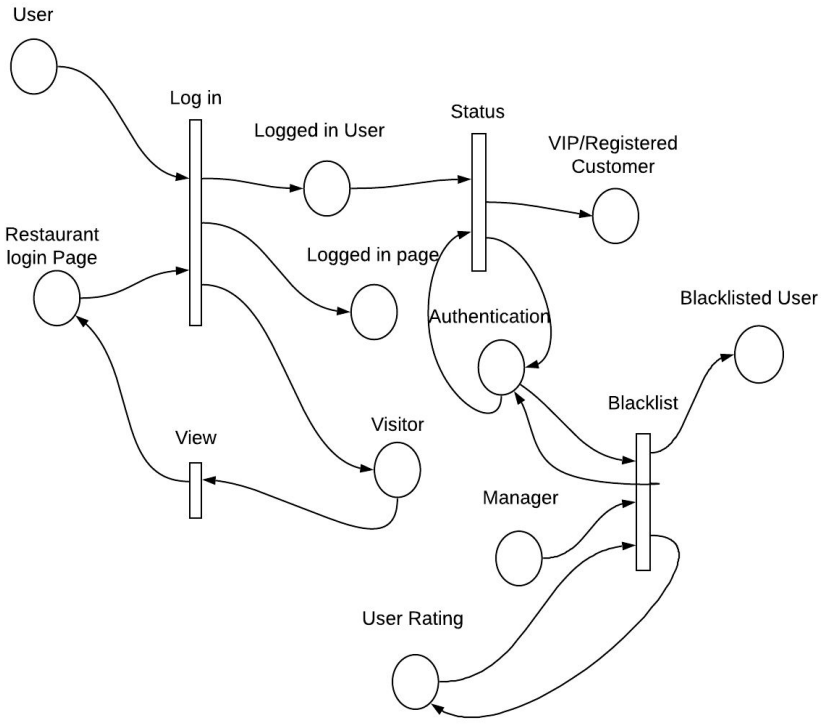
a) Normal Scenario
   i) Users such as registered customers can sign up or already registered and VIP customers can log in with a Google Account. Users such as guest don't have to login or sign up. Once registered and VIP customers login, they can access the account. They can see their rating and can also order food from restaurants. Guest users can order food from restaurants as well without logging in.

b) Exceptional Scenario
   i) If user has a bad rating, manager has an option to blacklist the user.

Collaboration Class Diagram



Petri-net - User login

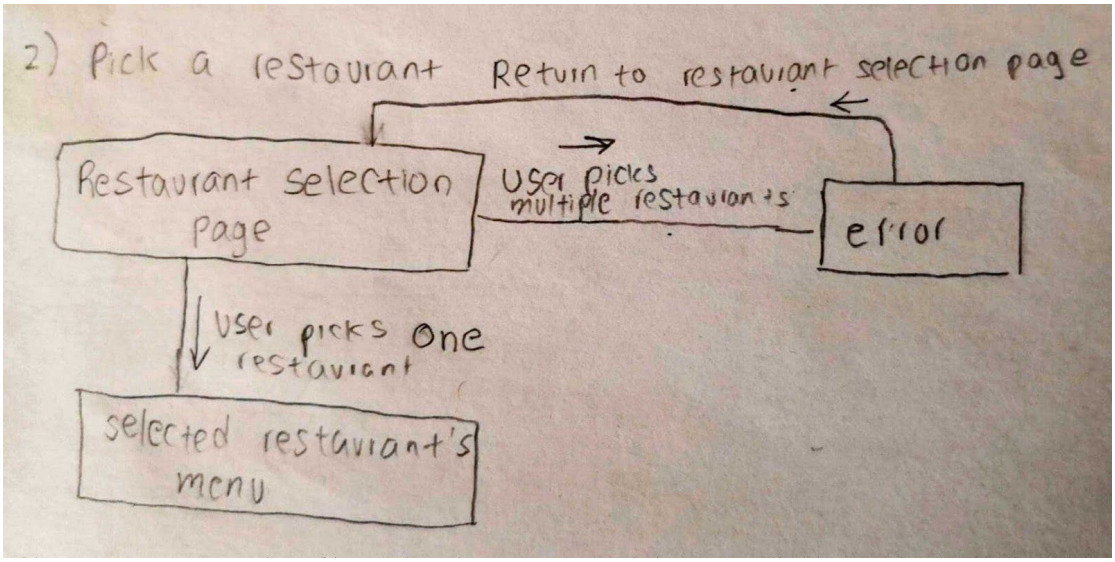


2) Pick a restaurant

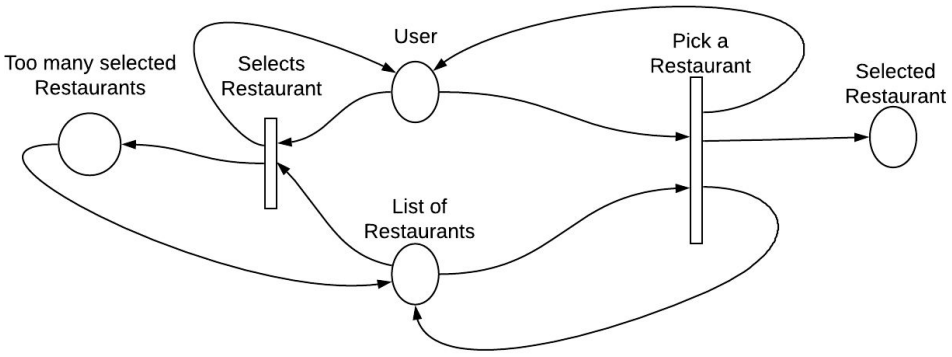
a) Normal Scenario
   i) User picks a restaurant

b) Exceptional Scenario
   i) User picks food items from 2 different restaurants and is alerted that they cannot do that (Can only order items from one restaurant per order)

Collaboration Class Diagram



Petri-net - Pick a restaurant



3) Selects items from that restaurant

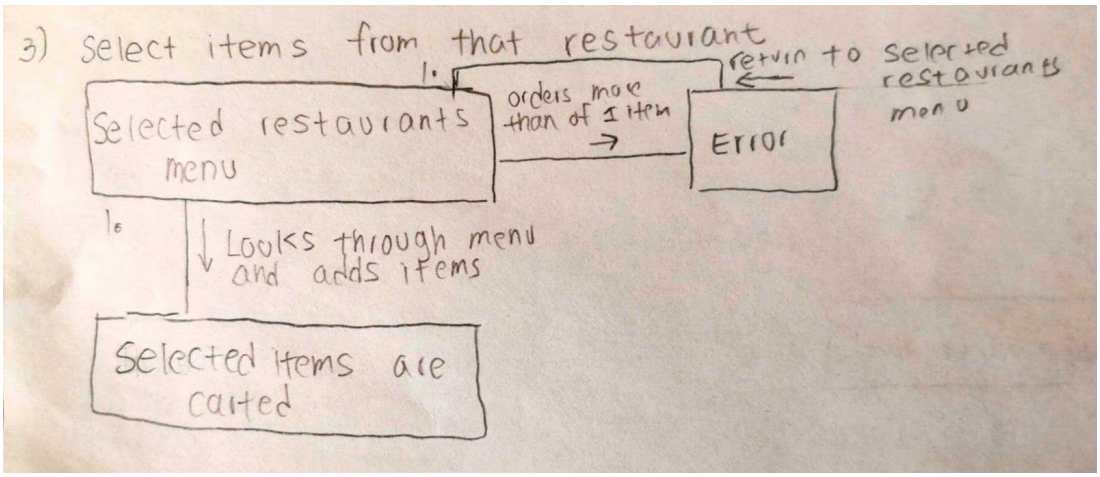
a) Normal Scenario

i) Customer look through menu and adds to cart the items they want

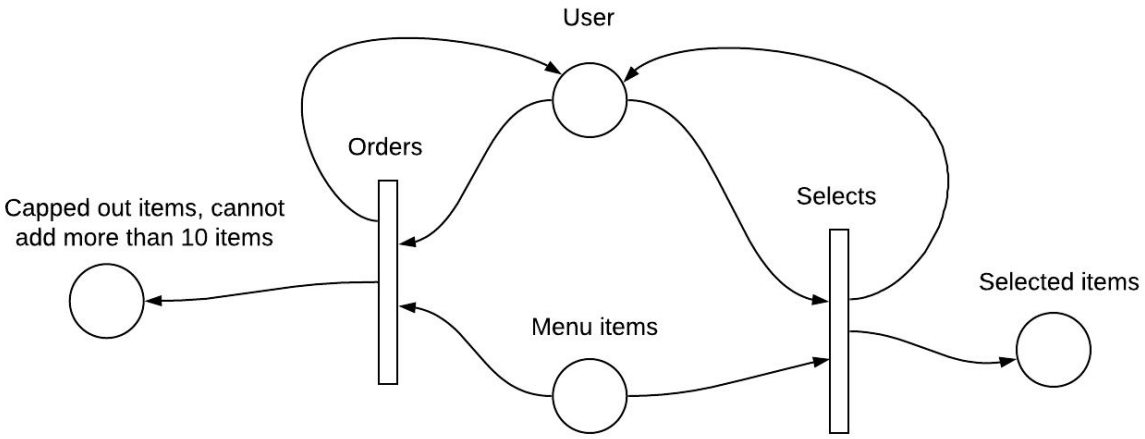
b) Exceptional Scenario

i) Customer cannot order the same item more than 10 times

Collaboration Class Diagram



Petri-net - Select Items from Restaurant

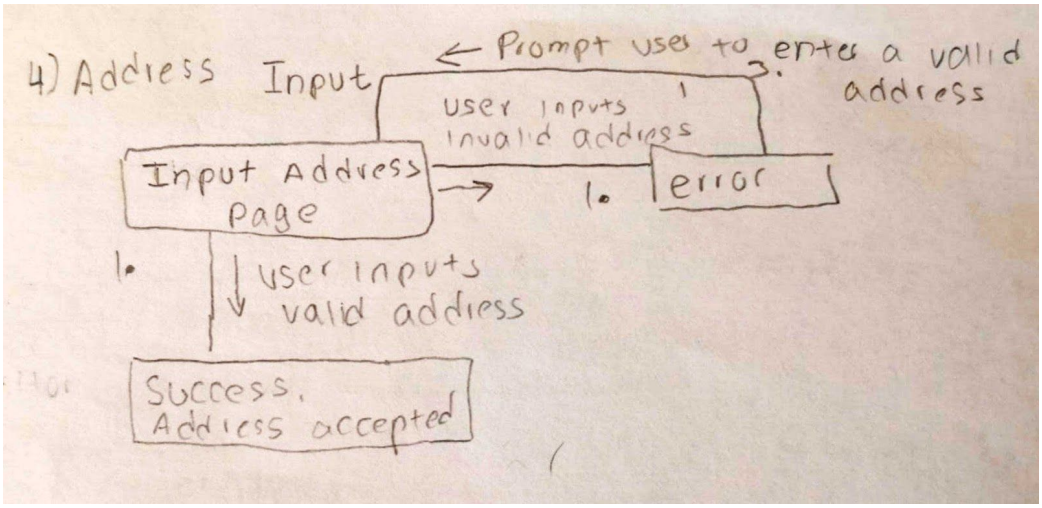


4) Address Input

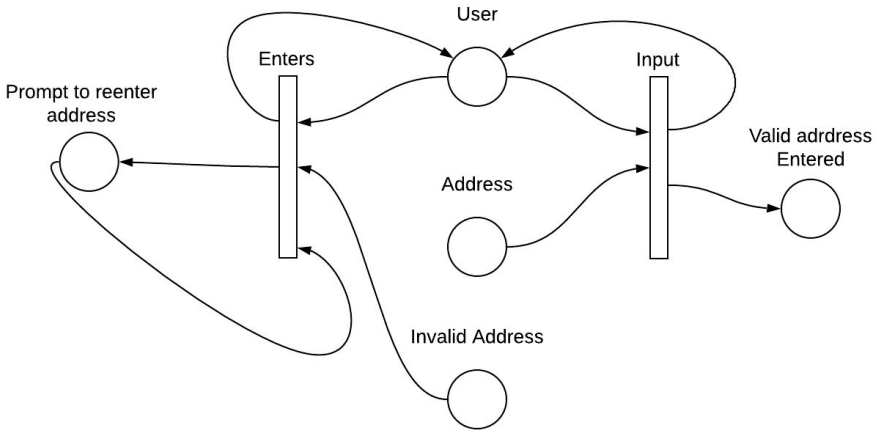
a) Normal Scenario
   i) User manually enters valid address

b) Exceptional Scenario
   i) Enters invalid address. System prompts customers to enter a valid address.

Collaboration Class Diagram



Petri-net - Address Input

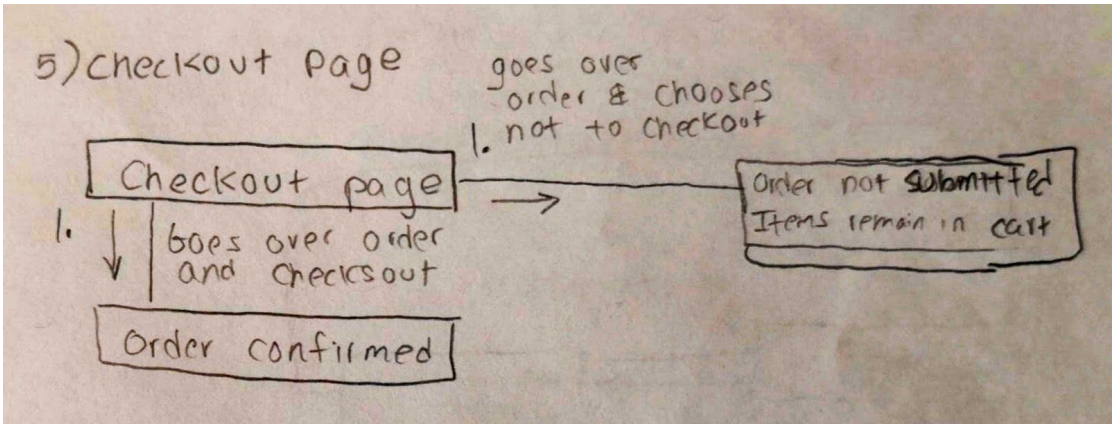


5) Checkout Page

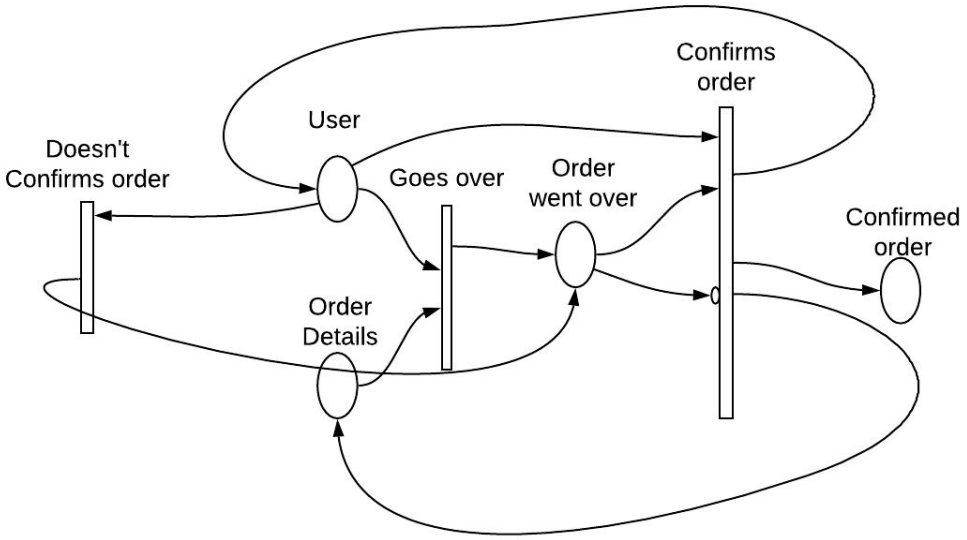
a) Normal Scenario
   i) Customer go over their order prices for each item and total price and checkout

b) Exceptional Scenario
   i) Customer goes over their order and chooses not to checkout
   ii) The items stay in the cart. Even after logging out.

Collaboration Class Diagram



Petri-net - Checkout Page

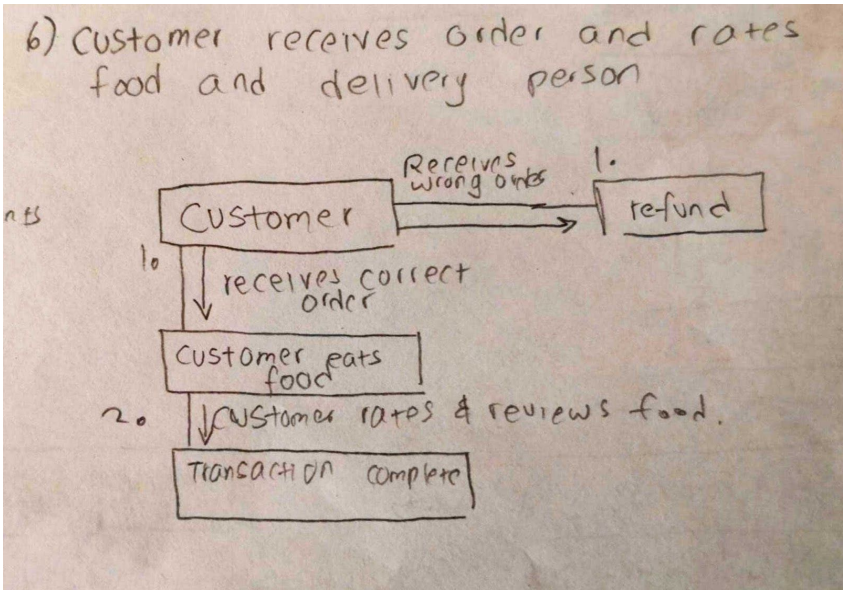


6) Customer receives order and rates food and delivery person

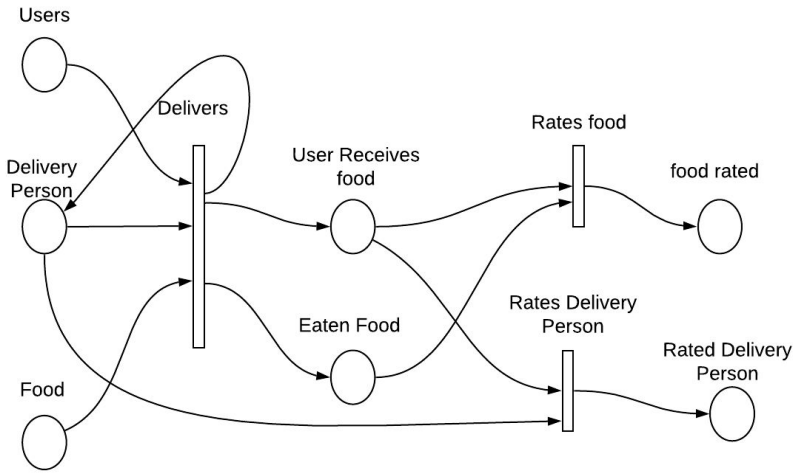
a) Normal Scenario
   i) Customer reviews delivery person
   ii) Customer reviews food

b) Exceptional Scenario
   i) Customer receives the wrong order

Collaboration Class Diagram



Petri-net - Customer receives order and rates food and delivery person



Chef Scenarios Overview:

1. Determine supply quantities
2. Creates menu and sets prices
3. Customer order comes in
4. Customer receives food and gives rating

1) Determine supply quantities

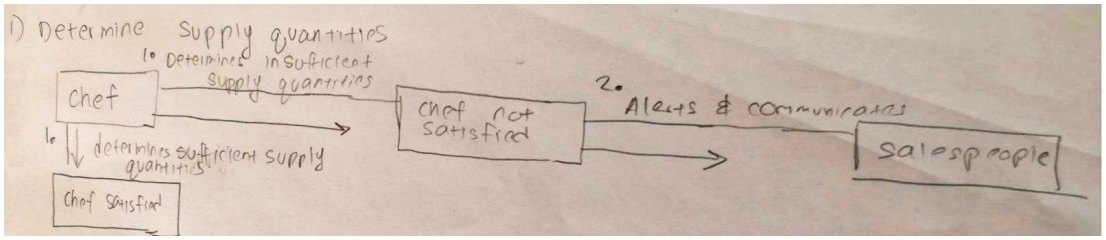
a) Normal Scenario

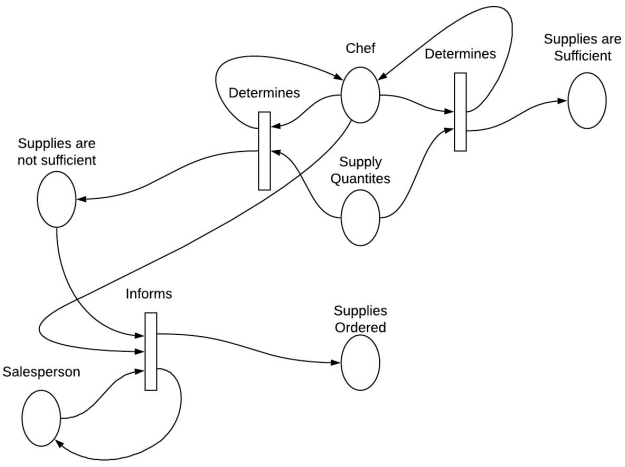
i) Chef determines supply quantities to be sufficient

b) Exceptional Scenario

i) Chef determines supply quantities to be insufficient and alerts the salespeople

Collaboration Class Diagram





2) Creates menu and sets prices

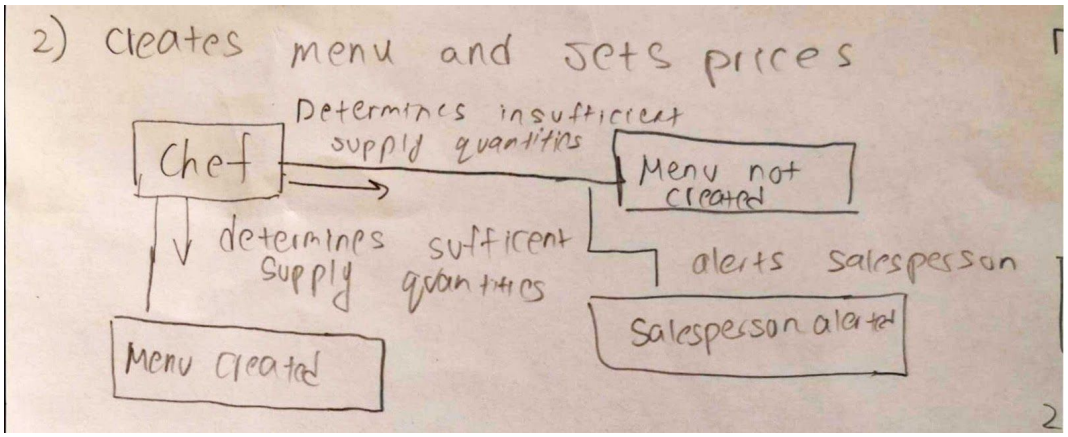
a) Normal Scenario

i) Determines the day's menu and sets prices

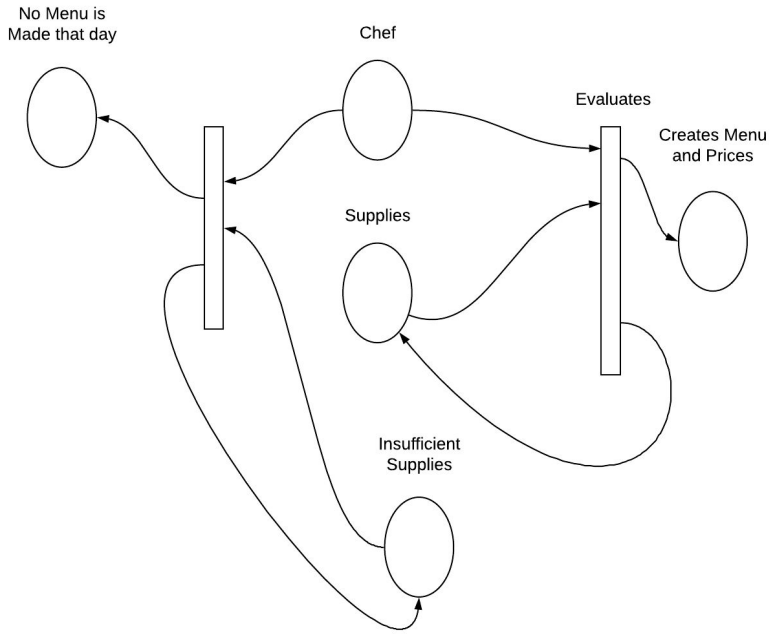
b) Exceptional Scenario

i) No supplies, cannot set menu

Collaboration Class Diagram



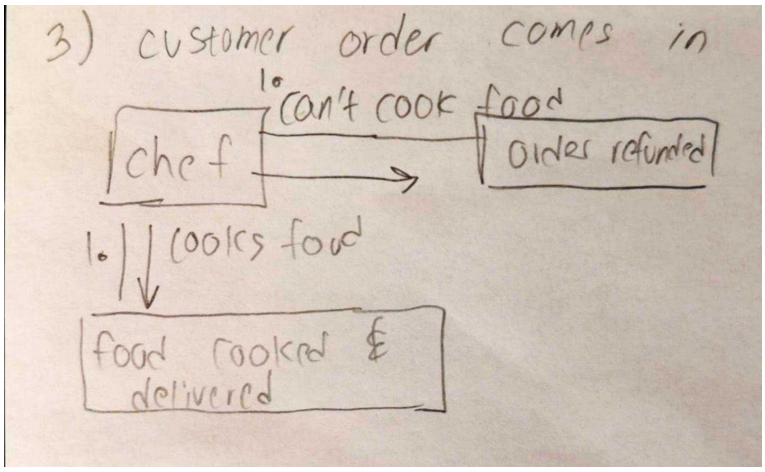
Petri-net - Creates Menu and Set Prices



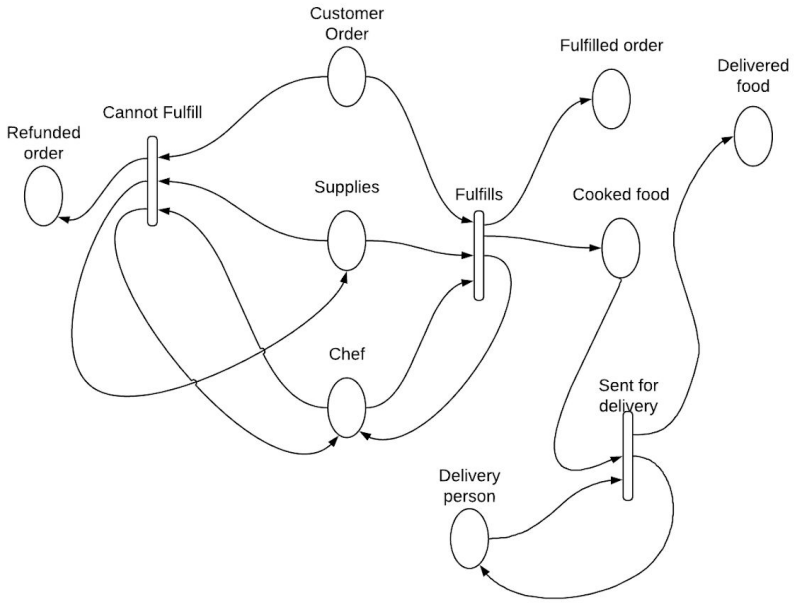
3) Customer order comes in

a) Normal Scenario
   i) Chef cooks food and is delivered to customer
b) Exceptional Scenario
   i) Chef is unable to cook food

Collaboration Class Diagram



Petri-net - Customer Order Comes in

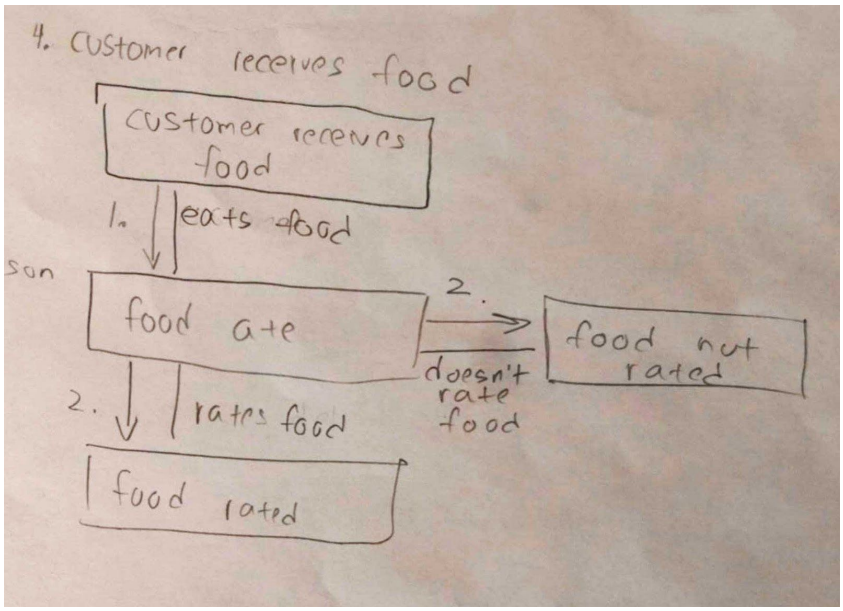


4) Customer receives food

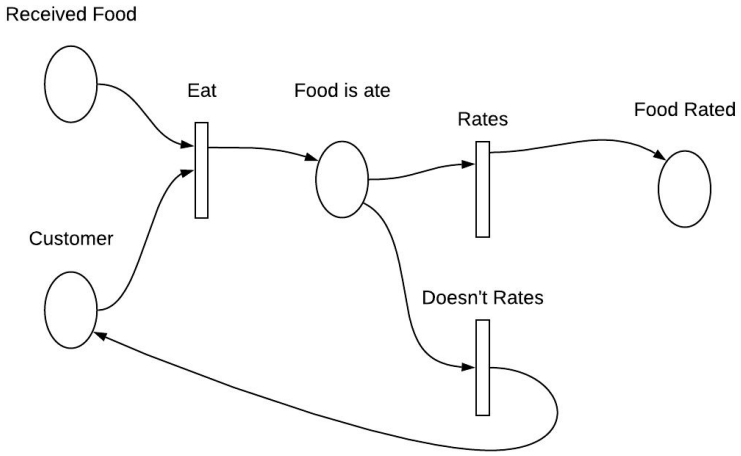
a) Normal Scenario
   i) Customer eats then rates food

b) Exceptional scenario
   i) Customer doesn't rate food

Collaboration Class Diagram



Petri-net - Customer Receives Food



Delivery People Scenarios Overview:

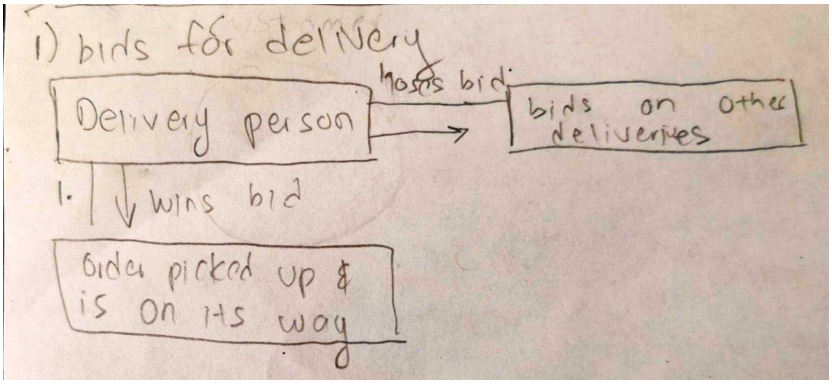
1. Bids for delivery (2 minute period)
2. decides on optimal route
3. arrives at destination and gives food to customers in exchange for money

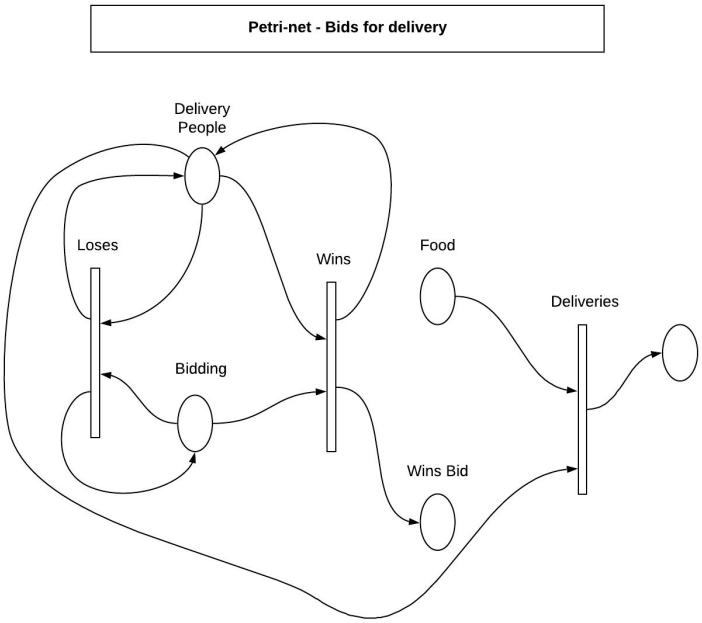
1) Bids for delivery

a) Normal Scenario
   i) Wins bid. Arrives at restaurant, picks up order, and starts delivery

b) Exceptional Scenario
   i) Decides not to go low enough for winning the delivery

Collaboration Class Diagram





2) Decides on optimal route

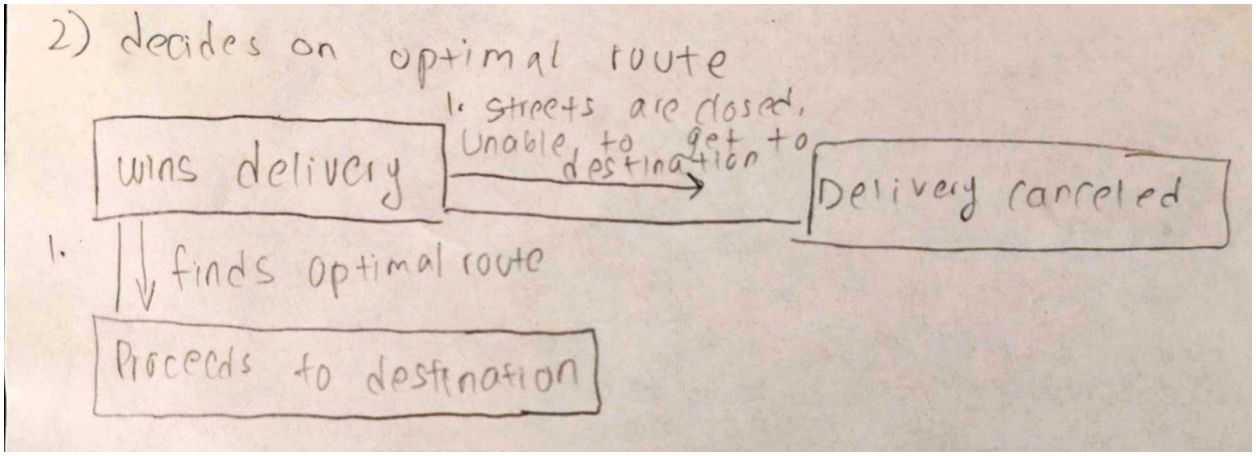
a) Normal Scenario

i) finds optimal route. Starts proceeding to destination

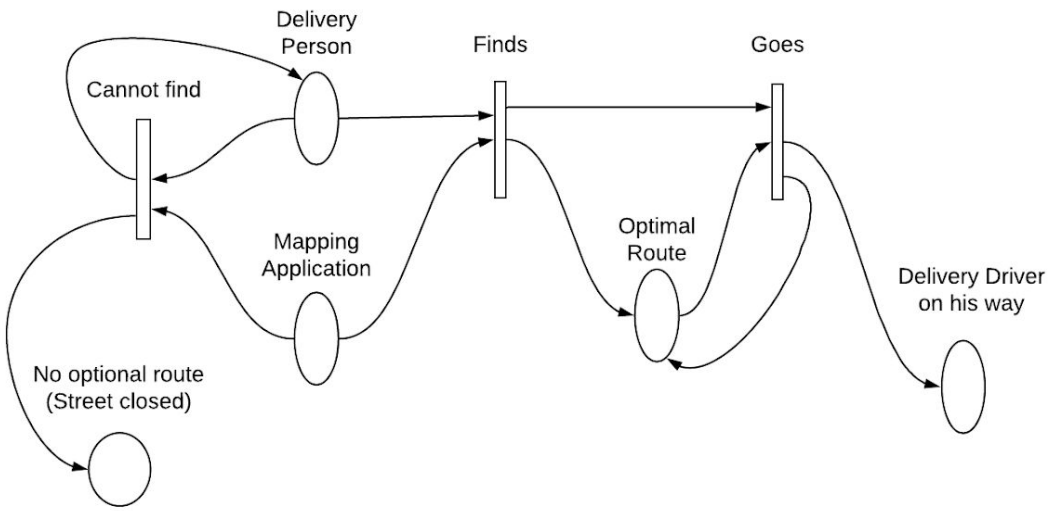
b) Exceptional Scenario

i) streets in the route are closed. Unable to get to destination. Must cancel order

Collaboration Class Diagram

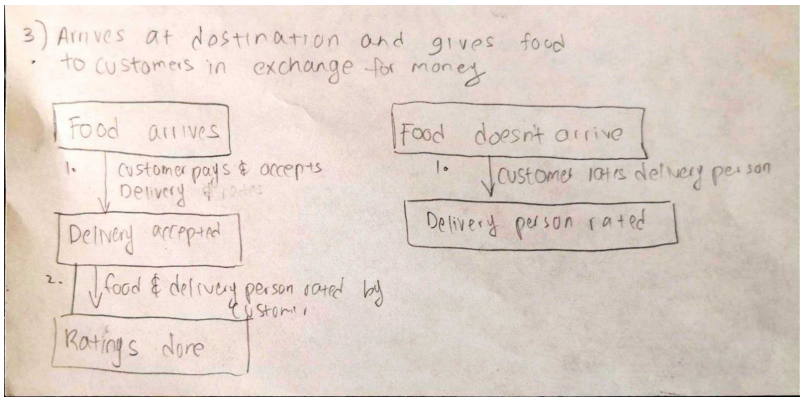


Petri-net - Decides on optimal route

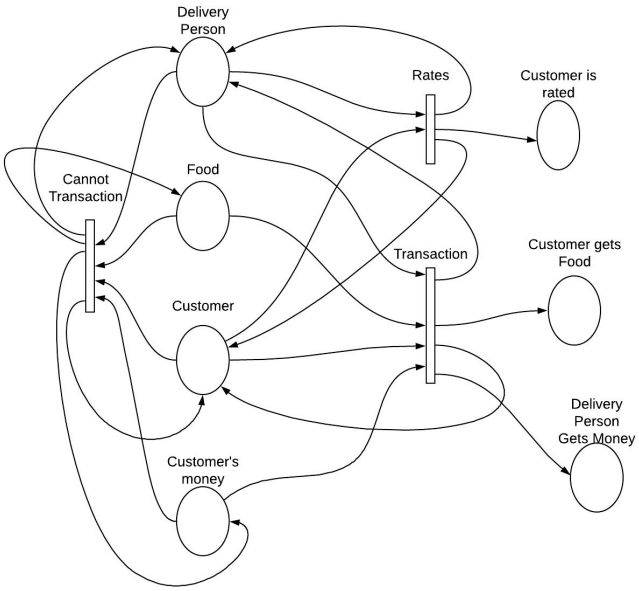


3) Arrives at destination and gives food to customers in exchange for money
    a) Normal Scenario
        i) Customer accepts food delivery and gives cash payment to delivery driver. Delivery person rates customer
    b) Exceptional Scenario
        i) customer cannot complete food delivery. Delivery person rates customer

Collaboration Class Diagram



Petri-net - Arrives at destination and gives food to customers in exchange for money



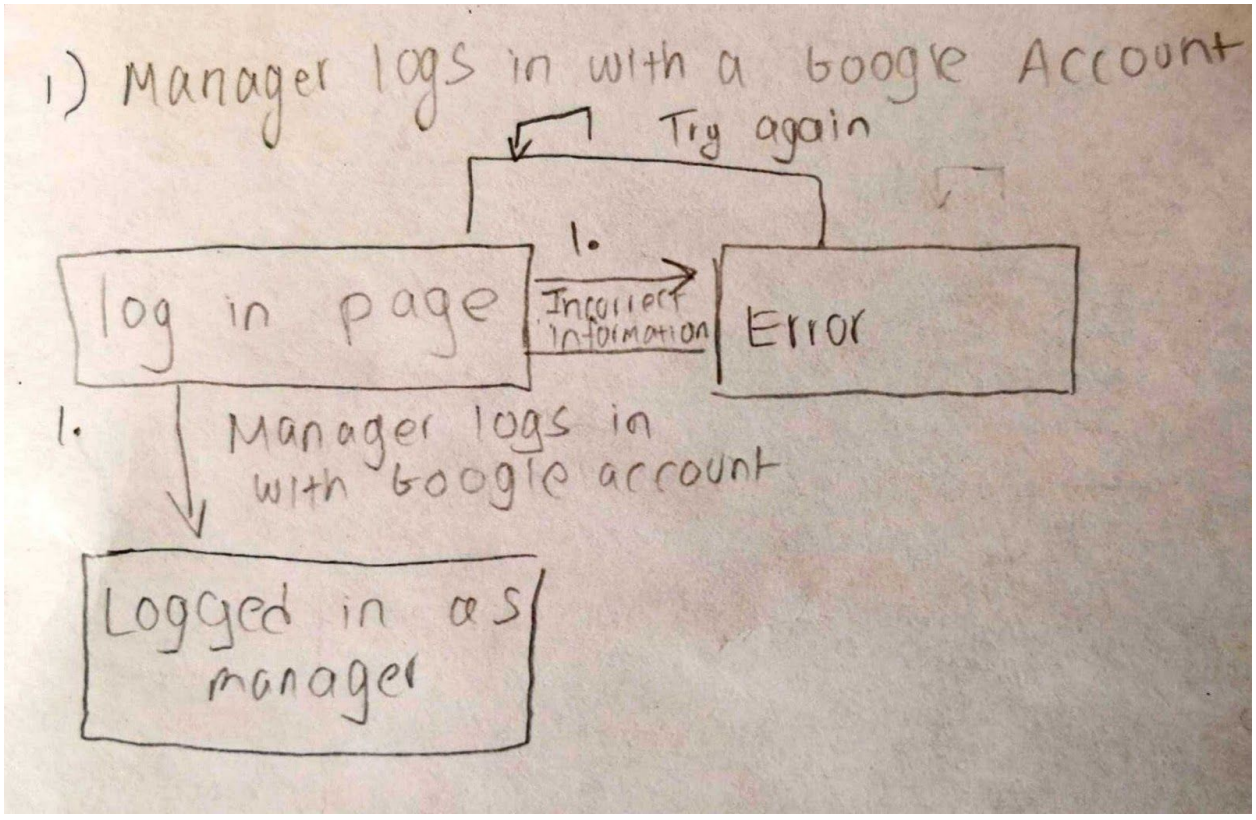
Manager Scenarios Overview:

1. Manager logs in with a Google Account
2. Receives order from customer
3. Initiates bidding for delivery for 2 minutes, awards delivery to lowest bid
4. Delivery person arrives at restaurant and picks up order

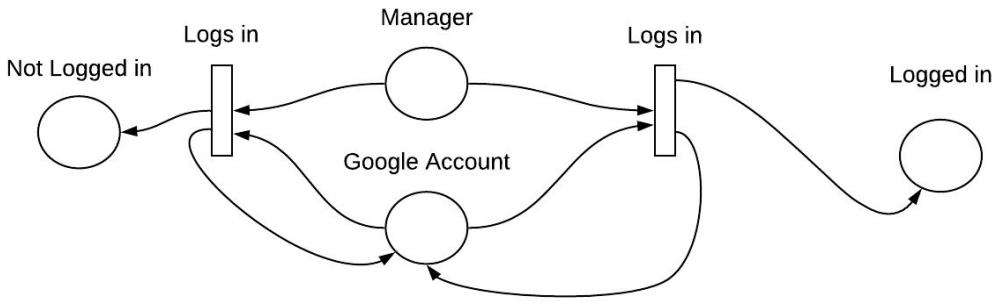
1) Manager logs in with a Google Account

1) Normal Scenario
    a) Manager logs in with a Google Account
2) Exceptional Scenario
    a) Manager enters wrong information. Prompted to enter correct information

Collaboration Class Diagram



Petri-net - Manager logs in with a Google Account

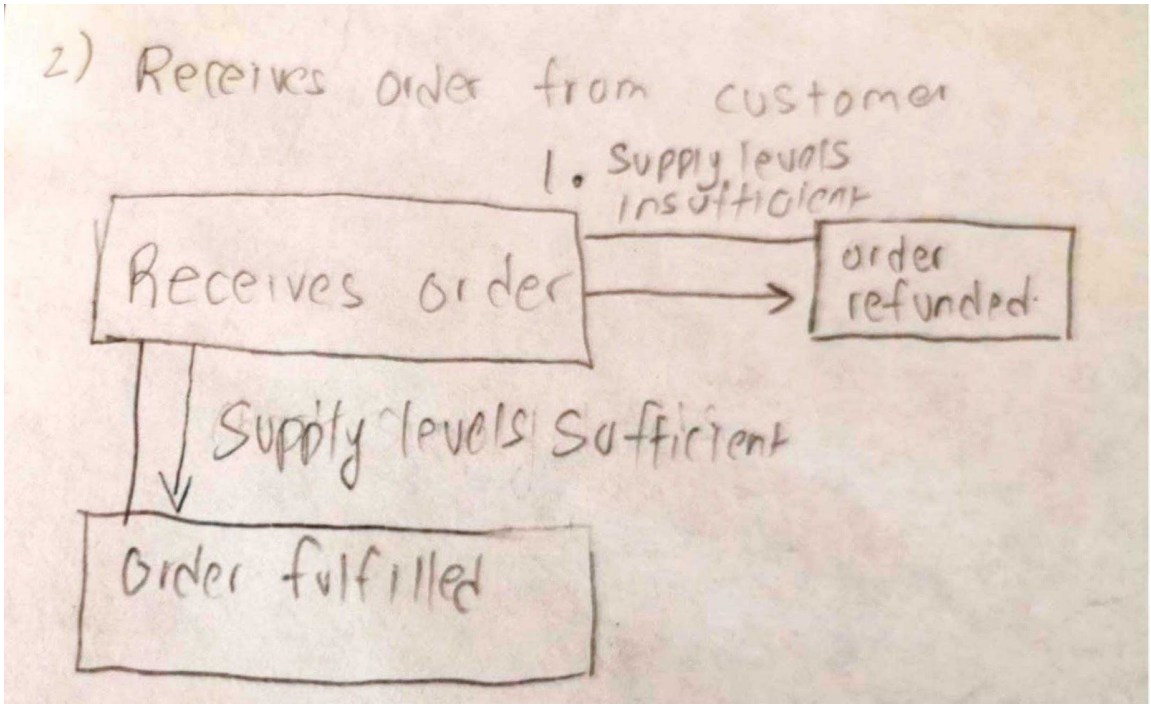


3) Receives order from customer

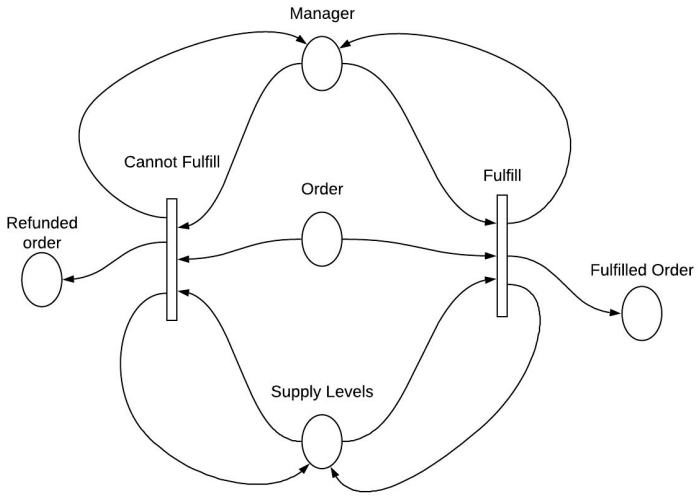
a) Normal Scenario
   i) There is enough supplies to fulfill the order

b) Exceptional Scenario
   i) There's not enough supplies to fulfill the order

Collaboration Class Diagram



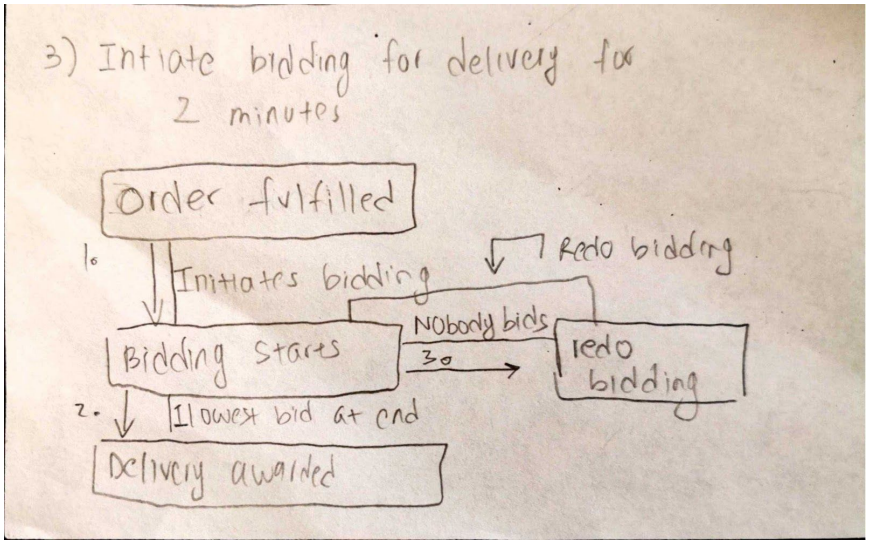
Petri-net - Received order from customer



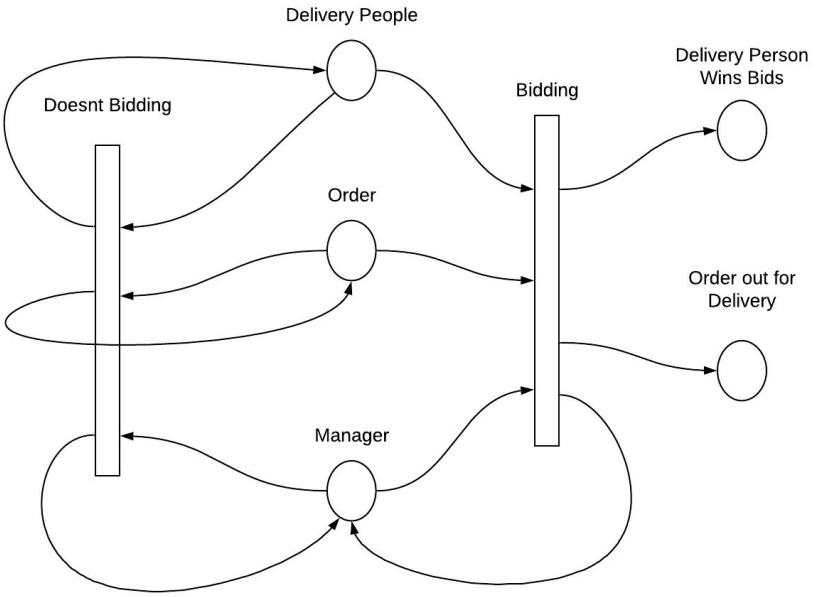
4) Initiate bidding for delivery for 2 minutes

a) Normal Scenario
   i) Multiple delivery people bid and the lowest bid emerges
b) Exceptional Scenario
   i) No delivery people bid

Collaboration Class Diagram



Petri-net - Initiates bidding for delivery for 2 minutes, awards delivery to lowest bid

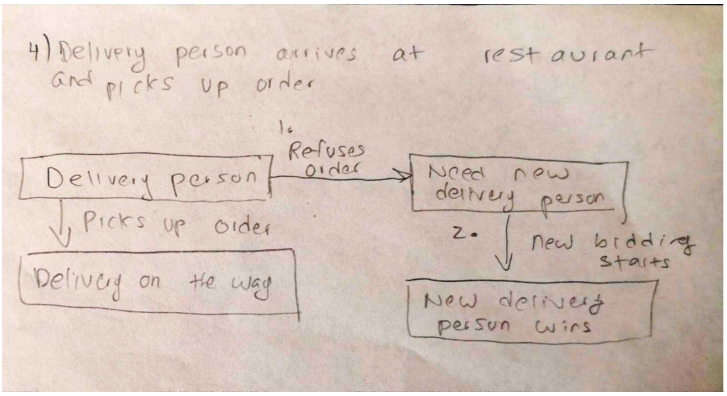


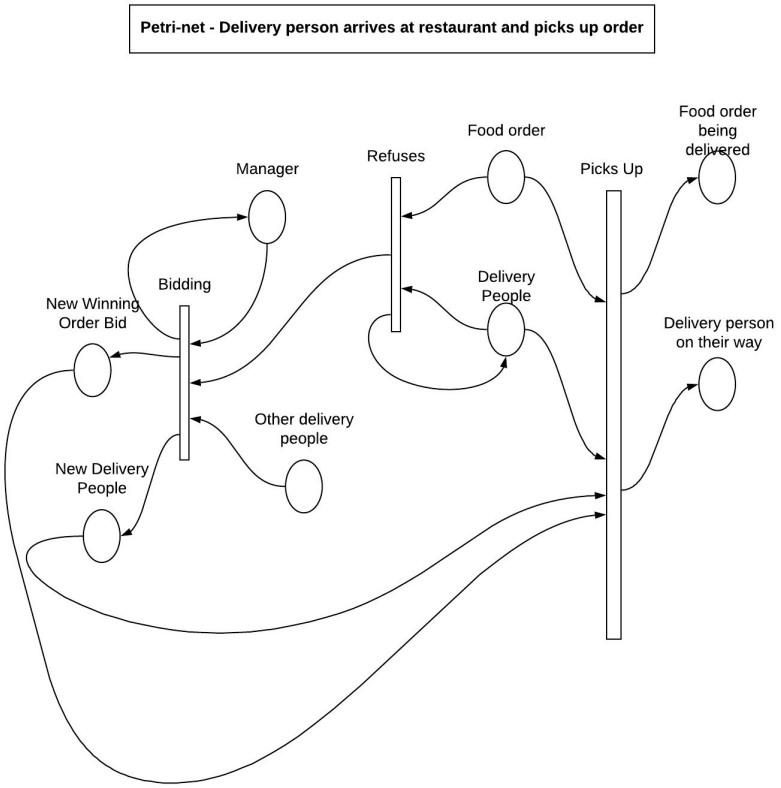
5) Delivery person arrives at restaurant and picks up order

a) normal scenario
   i) Delivery person picks up order and is on his way

b) exceptional scenario
   i) delivery person refuses to delivery to that address
      (1) must initiate bidding process again for a new delivery person

Collaboration Class Diagram





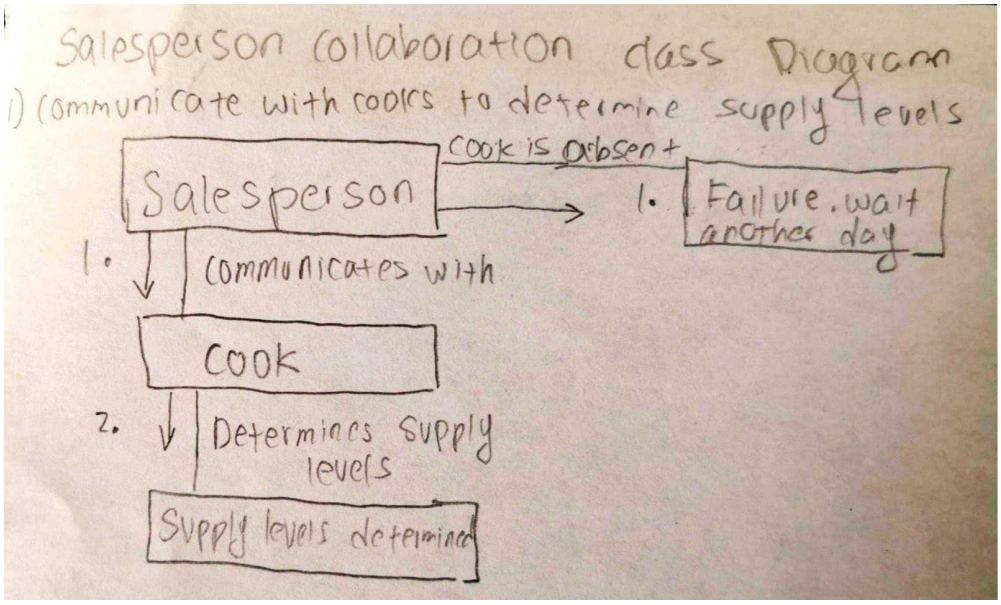
Salesperson Scenarios Overview:

1. Communicate with cooks to determine supply levels
2. Negotiates with suppliers for the best price and product
3. delivery for supplies arrive
4. update supply levels with cooks

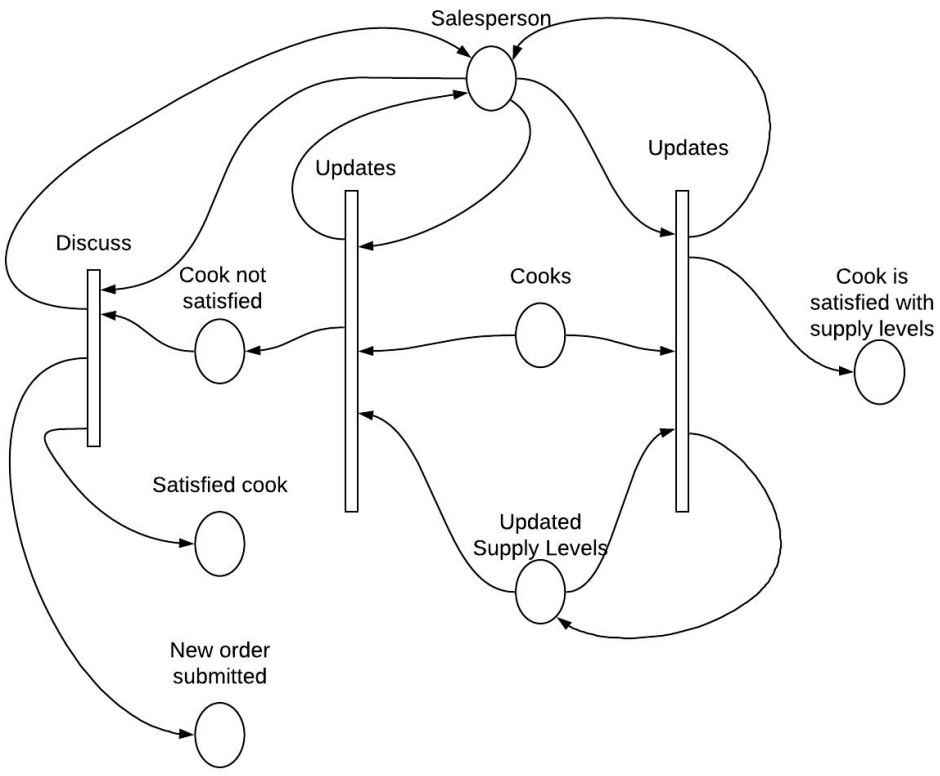
1) Communicate with cooks to determine supply levels

a) Normal Scenario
   i) Supply levels are determined
b) Exceptional Scenario
   i) Cook is not there, has to wait for next day

Collaboration Class Diagram



Petri-net - Update supply levels with cooks



2) Negotiates with suppliers for the best price and product

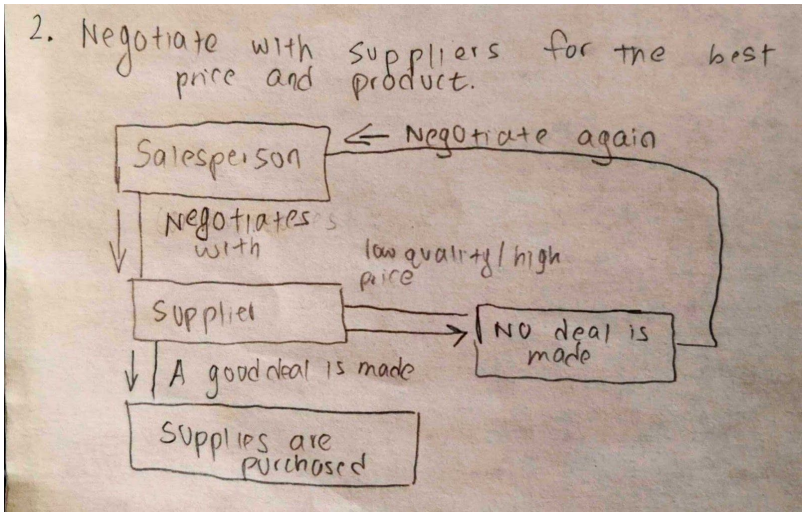
a) Normal Scenario

i) Suppliers give the salesperson a good deal on a good product. Supplies are purchased

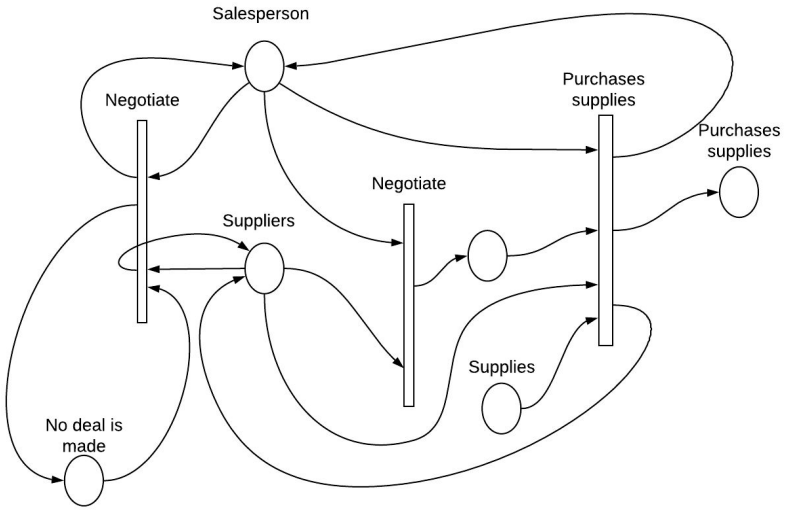
b) Exceptional Scenario

i) Suppliers have a low quality product or too high of a price (no deal made) and negotiations continue

Collaboration Class Diagram



Petri-net - Negotiate with supplies for the best price and product

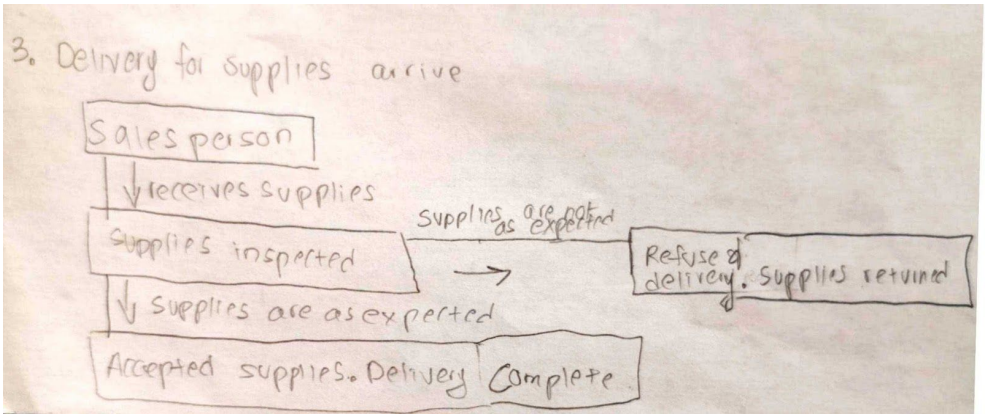


3) Delivery for supplies arrive

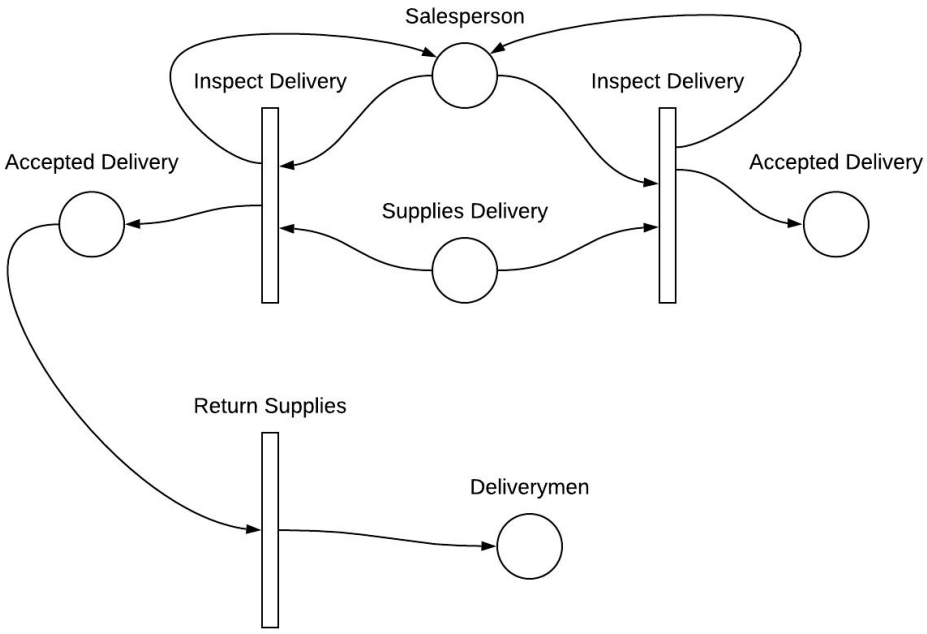
a) Normal Scenario
   i) Supplies are as expected and delivery is accepted

b) Exceptional Scenario
   i) Part of order is missing, damaged, or spoiled. Restaurant refuses delivery. Requests redelivery

Collaboration Class Diagram



Petri-net - Delivery for supplies arrive

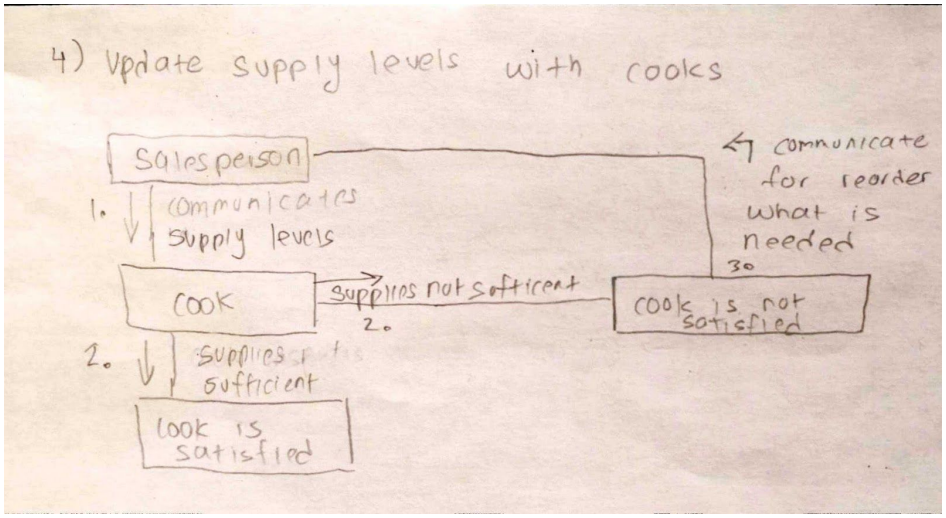


4) Update supply levels with cooks

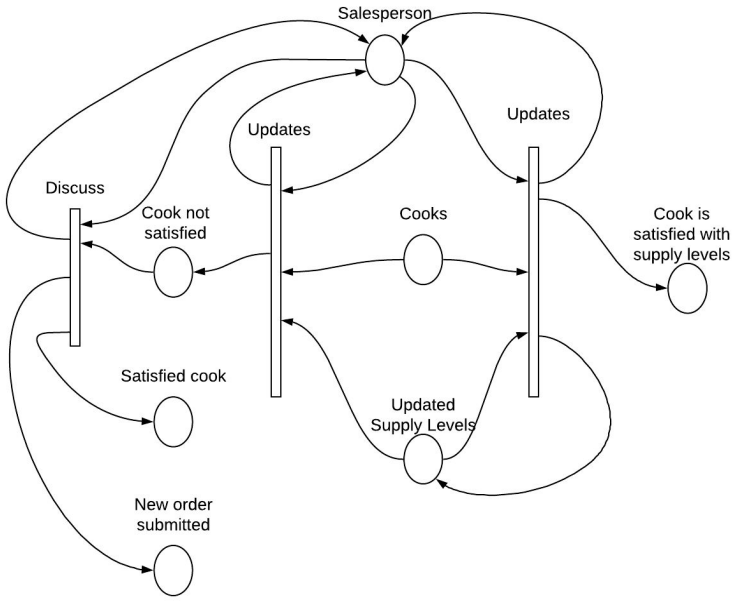
a) Normal Scenario
   i) Cooks are satisfied with supply levels

b) Exceptional Scenario
   i) Cooks indicate that something they ordered is missing or is the wrong product. Must order whatever is missing

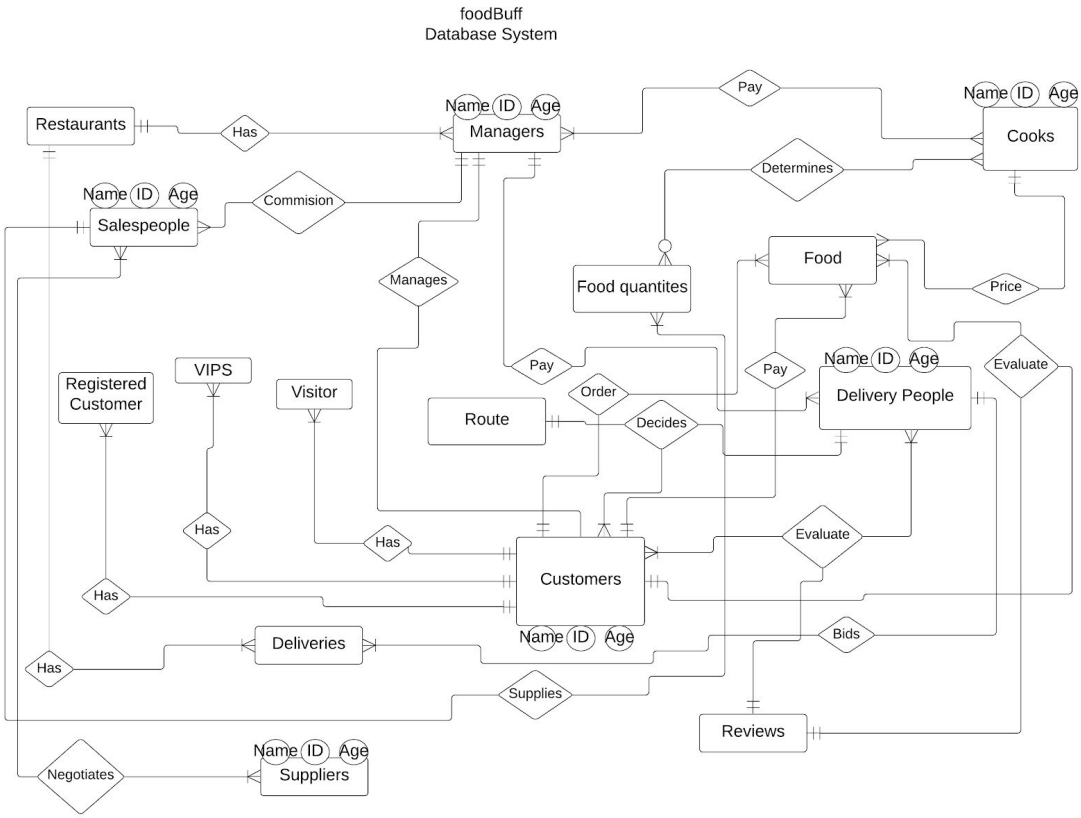
Collaboration Class Diagram



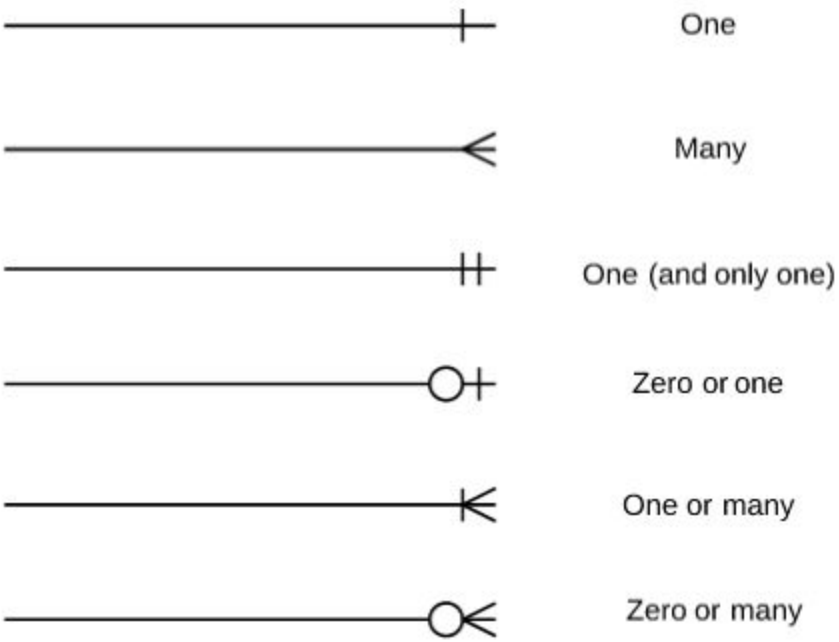
Petri-net - Update supply levels with cooks



3.Entity relation diagram



Key:



4. Detailed Design (Pseudo-Code)

Firebase.js

const provider()

This function uses firebaseConfig and creates a new firebase authentication provider (by using firebase.auth.GoogleAuthProvider()).

It is used to sign a user with their Google account.

const auth()

This function initializes Firebase authentication (by using .auth()) so a user's authentication data can be stored in Firebase.

const database()

This function initializes Firebase database (by using .database()) so we can read and write data to the firebase database in JSON format

Admin.js

function timeConverter(UNIX_timestamp)

We receive a UNIX timestamp as input and this function converts it to a readable date format

We do this by separating the month, year, day, hour, minute, second from the UNIX timestamp and then we combine it and return it like so:

day + ' ' + month + ' ' + year + ' ' + hour + ':' + min + ':' + sec

function getUserData()

This function reads all the order data. It fetches all the orders that have been placed by customers.

We do this by going into the "orders/" branch in Firebase database and storing the whole JSON object in a variable.

We then convert it to an array (since we can't iterate through JSON data directly in React).

Once it's converted into an array, we seperate the orderID and the orderID data and store it into a local variable so we can display it properly to the Admin page.

function handleChange(cook)

This function is responsible for keeping track of all the cooks that have been selected by the admin for the specific order.

Once the admin selects a cook, we append it to an array and store it into the local "cook" variable which will be pushed to the database in the updateOrder() function.

function updateOrder()

This function takes in all the local data that the admin has changed and pushes/updates the database accordingly for that order.

We do this by putting all the local data in a JSON object and pushing it to the path: `orders/${this.state.customerOrderId}`

This will push all the data changed by the admin into the "order/" branch of the database and specifically the branch which has the same OrderID (to ensure we don't change another order)

GoogleAuthentication.js

function GoogleAuthentication()

This function helps the user login with Google and also logout.

For logout, we use "auth.signOut()" where auth is the function we used in "firebase.js"

For login, we use "auth.signInWithPopup(provider)" where provider is the function we defined in "firebase.js"

OrderDialog.js

function OrderDialog()

This function opens the order confirmation dialog box once the user clicks on "Order". It displays: "Your order is on the way!" "Thank you for choosing foodBuff".

Order.js

const sendOrder(total, orders, { email, displayName })

This function takes in the order total as "total", the order itself as "orders", the user email as "email" and the username as "displayName" and pushes it to the "orders/" branch of firebase database.

We do this by grouping multiple variables as a JSON object and then pushing all the local data to firebase database so we can view it later.

The data we push includes:

customerOrder,
customerEmail,
customerName,
orderDate,
orderTotal,
orderStatus,
deliveryStatus,
cook

const Order({orders})

This function calculates the subtotal, tax, the total amount and also renders all the order details which the customer selected (from the "orders" parameter)

Navbar.js

const Navbar({login, logout, loggedIn})

This function takes in login, logout, and loggedIn as parameters.

If the user is loggedIn, it will render: `Welcome back ${(loggedIn.displayName)}`

If the user clicks on the login button, the "login" function is called which logs in the user (this function is located in GoogleAuthentication.js)

If the user clicks on the logout button, the "logout" function is called which logs out the user" (this function is located in GoogleAuthentication.js)

Menu.js

const Menu({setPopup, loggedIn})

This function takes in the setPopup data which is used to display the food items to the user. It also takes in the user credentials as a "loggedIn" parameter to determine if they are a Visitor, Registered User, or VIP.

Depending on the user's status, this function will render different food items.

SetQuantity.js

const SetQuantity({quantity})

This function is used to set the quantity of the food item (how much quantity the customer wants to order).

We do this by having increment (quantity.setValue(quantity.value + 1)) and decrement (quantity.setValue(quantity.value - 1)) logic.

FoodDialog.js

const getPrice(order)

This function takes in the order details and returns: quantity (of the order) * price (for the food item)

Choices.js

function Choices({ openPopup, choiceRadio })

This function takes in "openPopup" which contains all the food data for what the user clicked. "Choiceradio" is used to record the chosen option and set it as the user choice (so we can push it to the database later)

We map through all the JSON data and display it as radio buttons so the user can choose one option.

The format of choices is: "choices: ["Coke", "Sprite", "Root Beer"]"

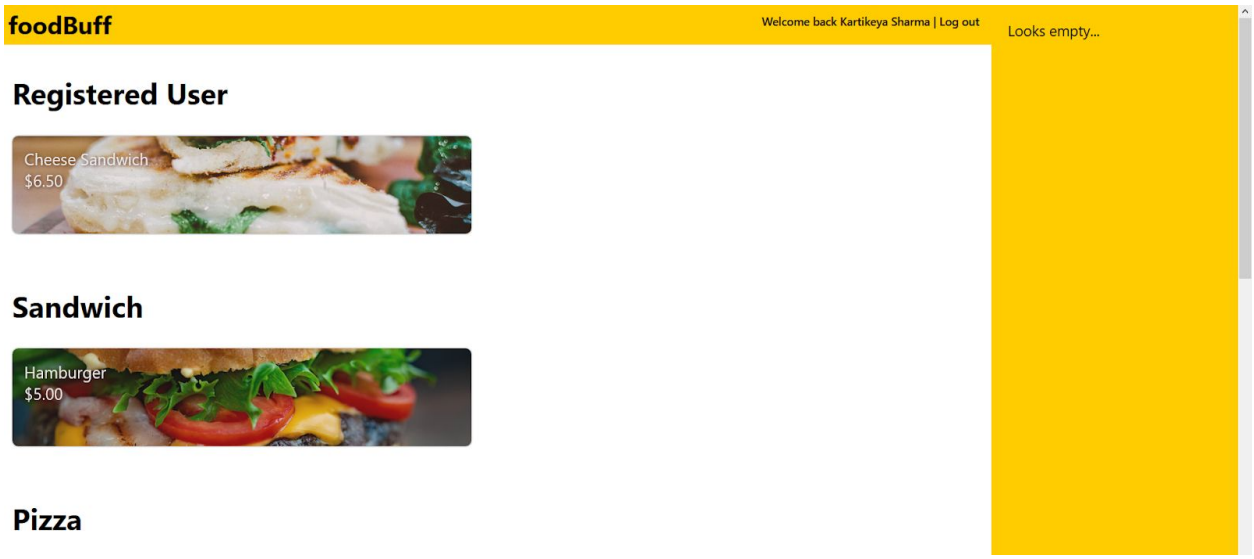
intToUSD.js

function intToUSD(price)

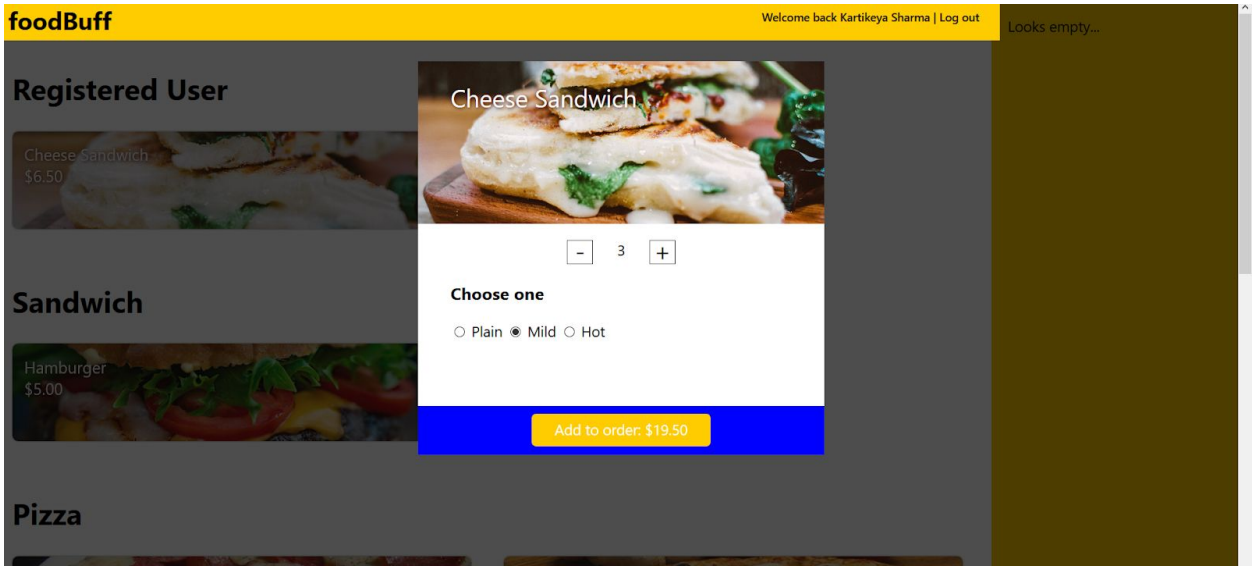
This function takes the food price as its parameter and returns it in dollar format by using ".toLocaleString({ style: "currency", currency: "USD" })"

5.System Screens

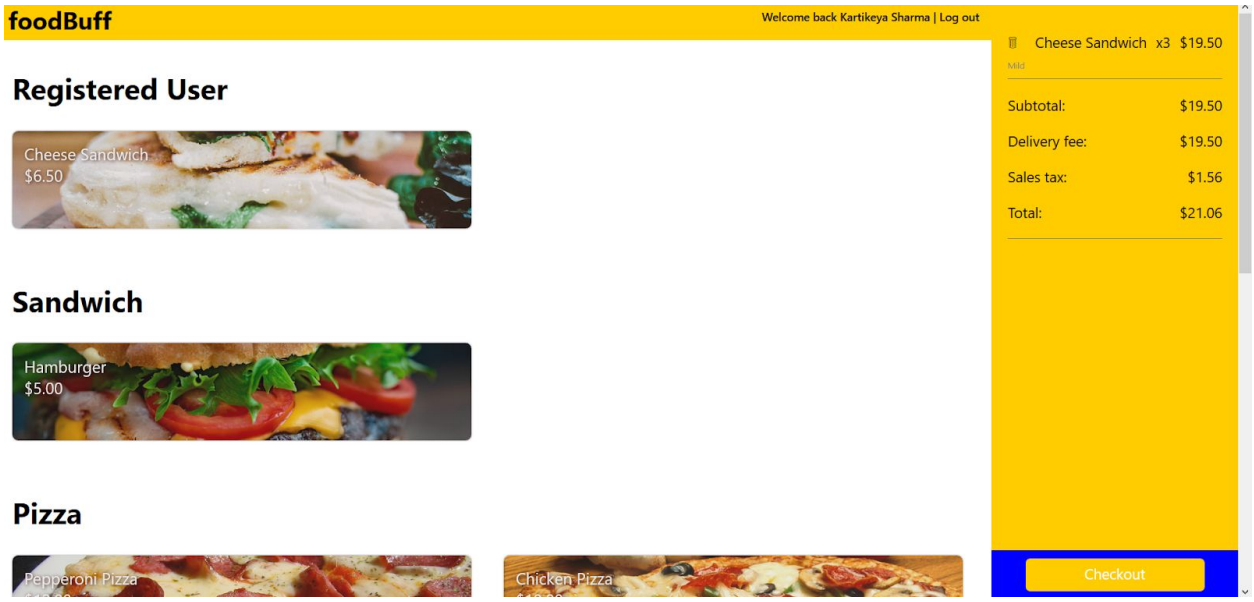
Customer menu screen:



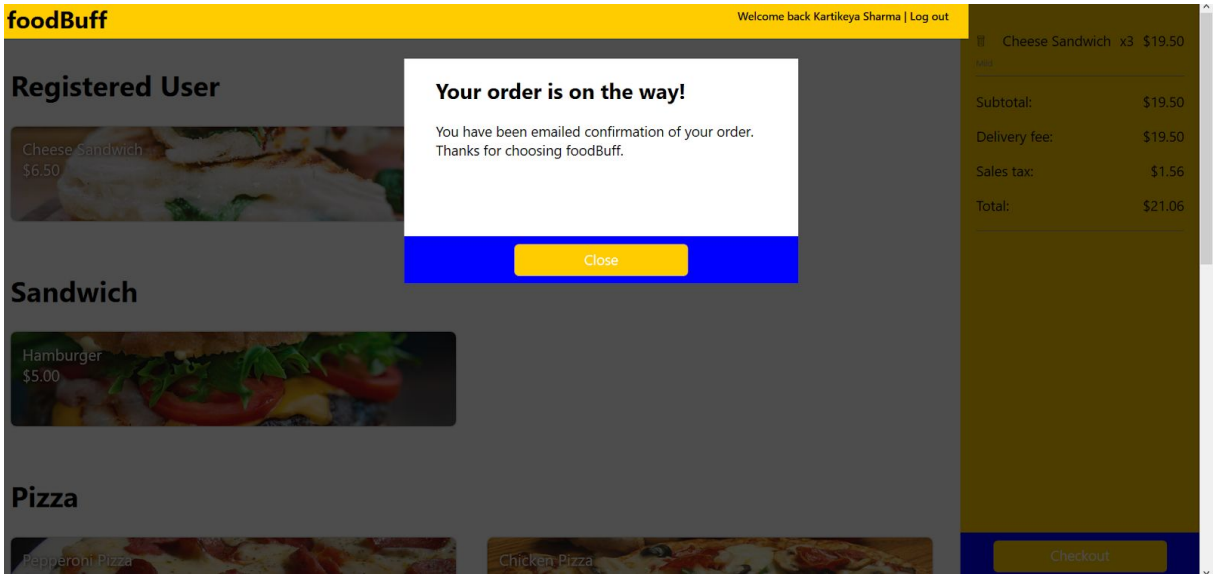
Food option screen:



Cart Screen:



Order Confirmation page:



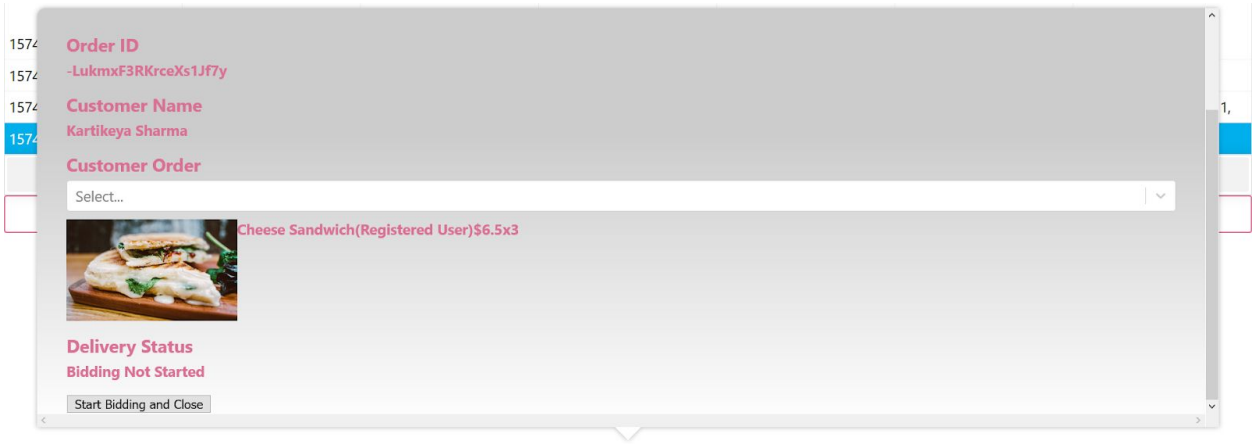
Admin page:

Order Date	Customer Email	Customer Name	Order Total	Order Status	Delivery	Cook(s)
1574584550958	kartikeyasharma04@gm...	Kartikeya Sharma	33.48	Pending	Bidding In Progress	Cook 4, Cook 5,
1574621424368	kartikeyasharma04@gm...	Kartikeya Sharma	9.72	Pending	Bidding In Progress	Cook 3, Cook 5,
1574709535582	kartikeya.sharma08@g...	Kartikeya Sharma	42.66	Pending	Bidding In Progress	Cook 3, Cook 4, Cook 1,
1574924177644	kartikeya.sharma08@g...	Kartikeya Sharma	21.06	Pending	Bidding Not Started	Not Assigned

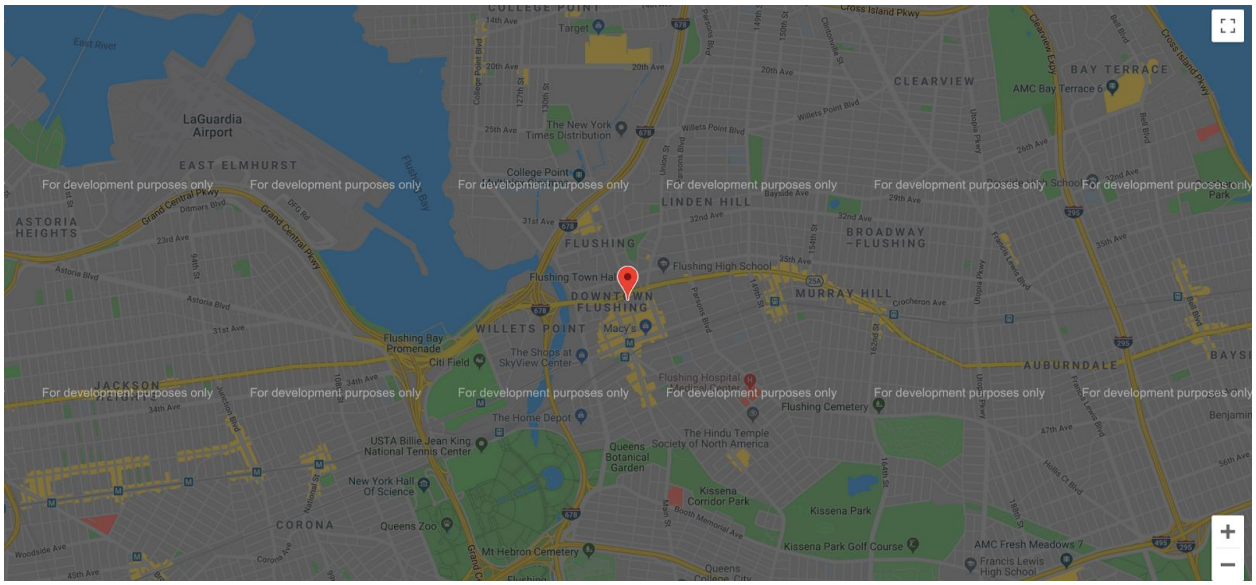
Previous Page 1 of 1 5 rows Next

MODAL

Admin Edit page:



Delivery page:



6. Minutes

Date	Reason	Duration	Communication
11.6.19	Assign work and decide on technologies	4.25hrs	In Person
11.11.19	Work on screen mockups, flow, and	4.25hrs	In Person

	current progress		
11.13.19	Work on Phase II Report	4.25hrs	In Person

11.18.19	Work on Phase II Report	3.75hrs	In Person
11.20.19	Work on Phase II Report	2.5hrs	In Person
11.25.19	Work on Phase II Report	3hrs	In Person
11.27.19	Finishing touches of Phase II Report	3hrs	Google Hangouts

7. Github Repo

You can find our project source code at this link: https://github.com/kartikeya01/foodBuff