Team 6 Phase Report #2

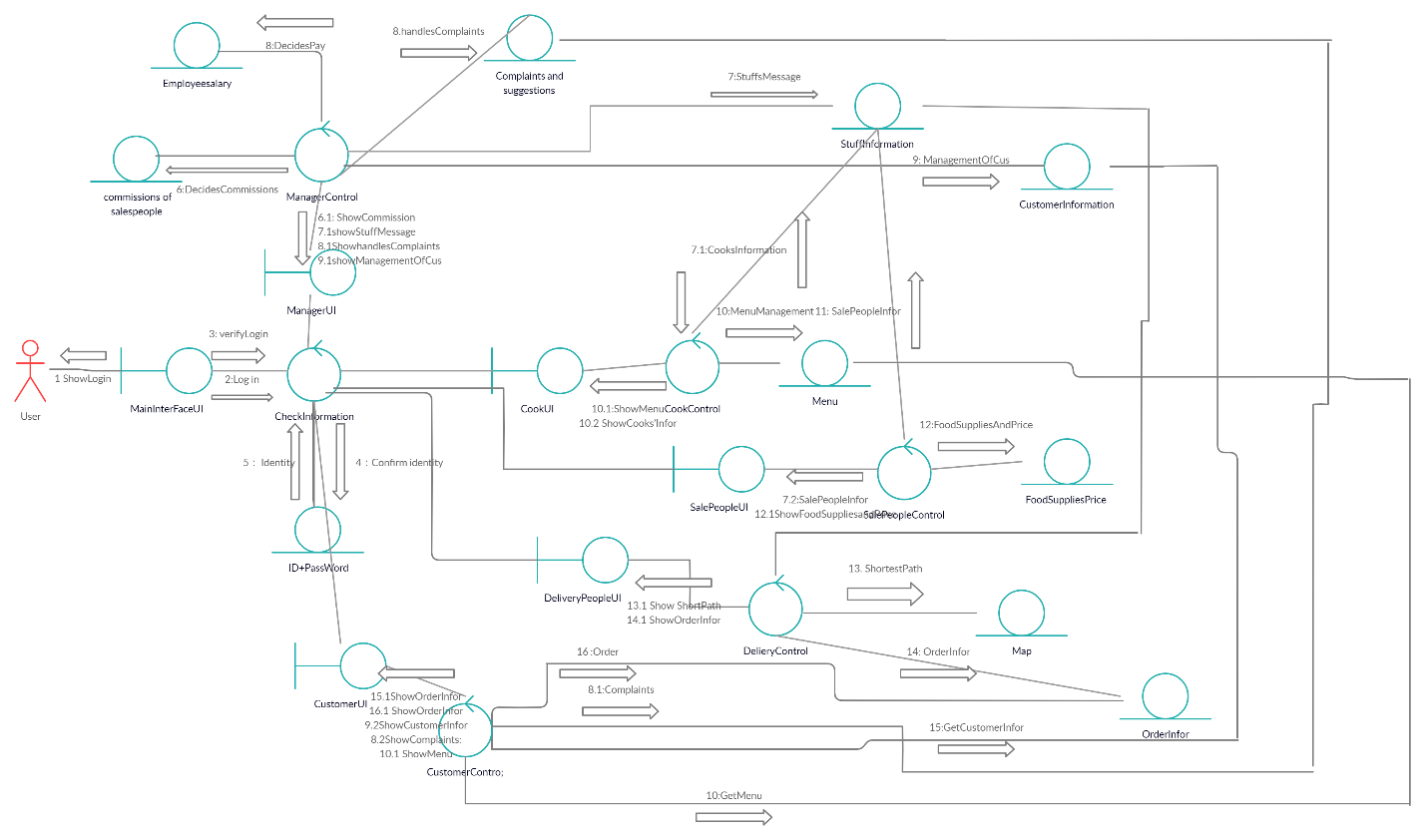
By

Chue Zhang, Eric Mai, Jia Ming Ma, Jian De Li, Wilson Song

**1. Collaboration class Diagram 1**

*In this section, we will be illustrating our general collaboration class diagram and what databases are available in our software and how everything each thing is connected to one another*

**1.1 General collaboration class diagram for our software**

****

**2.0 Use-Cases**

*In this section we will be talking about the Use-cases in our software, being specific about user case scenarios such as exceptional and normal scenarios. Furthermore, there is illustrations on each use-case that we have using the Petri-Net diagram and the Collaboration Class Diagram*

**2.1 Use-cases, both Exceptional and Normal**

Manager use cases

**Normal Scenario**

1. A manager hires a crew of salespeople
2. cooks and delivery men and sets their salary
3. A worker does his/her jobs
4. If a worker receives too many complaints, they will get a warning then will be laid off
5. If a worker receives good ratings and no complaints, they will receive a 10% salary increase
6. A customer has decided to use the system
7. If the customer is not rated well by delivery men, they are put in the blacklist
8. If the customer is rated well by the delivery men, they are put in the VIP list
9. A menu is created
10. If menu does not fit the Managers needs, the manager will edit the menu

**Exceptional Scenario**

1. A manager is angry
2. Lays off a lot of workers and hires a new set of workers
3. Nothing in the Menu fits the need of manager, everything Is edited
4. Manager steals money
5. No one gets paid, might not have enough money for restaurant and to pay for salary, people quit

Customer Use Cases

**Normal Scenario**

1. A user starts the software
2. User can Login if they already have a username and password
3. User can signup if they have not yet created a username and password
4. User can enter as guest
5. The user Enter the Menu Page
6. User adds whatever they want to the cart and confirms their purchases
7. The user Enters Confirmation Page
8. User confirms what they have purchase and finalize purchase and given an arrival time of food
9. The user receives the food
10. The user can rate the person that delivered the food and the chef as well
11. If user is an active user and is rated well
12. They are promoted to VIP and receive discount in the menu

**Exceptional Scenario**

1. User Closes software
2. End goal of customer buying something is not achieved, no one gains anything
3. User enters wrong address
4. User does not acquire the food, files complaints and a gets refund

Salesperson Use case

**Normal Scenario**

1. Receives a Budget
2. Buys item to keep inventory in stock based on Cook input
3. Views suppliers available
4. Has 3 suppliers to choose from, whichever has best price and quality, he/she will buy from
5. Checkout
6. Confirms purchases from suppliers and has items shipped immediately

**Exceptional Scenario**

1. Salesperson quits
2. One less person to do the job or no one to do the job. End goal is either harder to reach or is not reached at all
3. Salesperson only gets bad quality food
4. Has a talk with the manager, food can barely be used and may stop food from being cooked for customer

Cooks Use Case

**Normal Scenario**

1. Food order is received
2. Set timer for food preparation and given to delivery when done
3. Change in Menu
4. Removes certain dishes from menu due to lack of ingredients
5. Talk to salesperson about inventory
6. Takes inventory
7. Takes a look at what is in stock and tells salesperson what is needed

**Exceptional Scenario**

1. The Cook forgets to cook the food
2. He is fired or given a warning, the chef will be re-assigned, and complaints will be deal with. Food may be refunded due to lateness
3. The Cook quits
4. One less cook to do the job or no cook to do the job. End goal is harder to reach or is not reached at all
5. The Cook doesn’t do inventory
6. There are no ingredients to cook the food. Either there will be no food in the menu or customer will order food to find out that they have wasted their time

Delivery Man Use Case

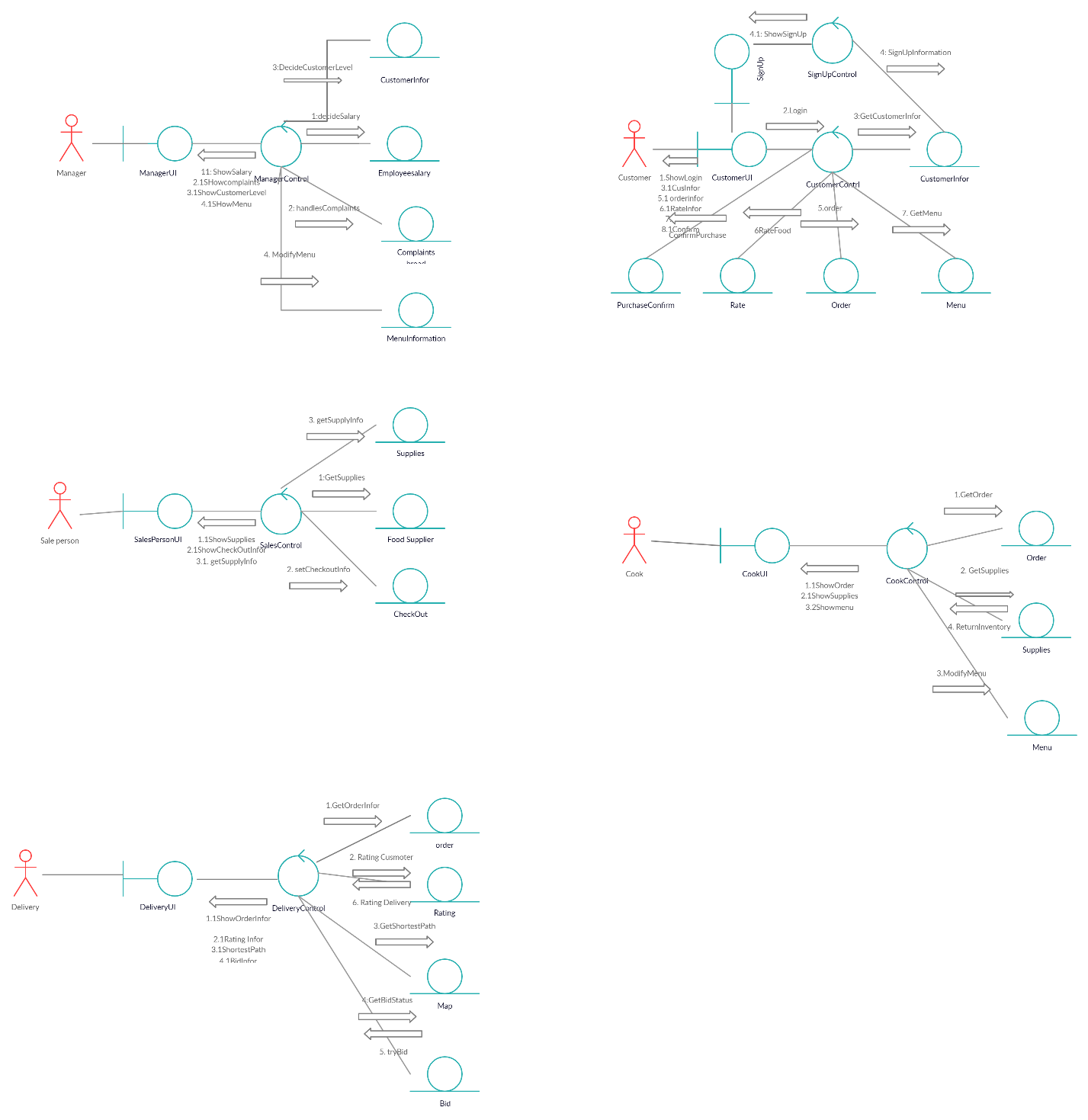
**Normal Scenario**

1. Receives Order Confirmation
2. Starts a bid with other delivery men on who gets to deal with the delivery
3. Wins Delivery
4. Delivery man gets address to deliver to after picking up food
5. Delivers Food
6. After Delivering food, rate the customer based on how customer

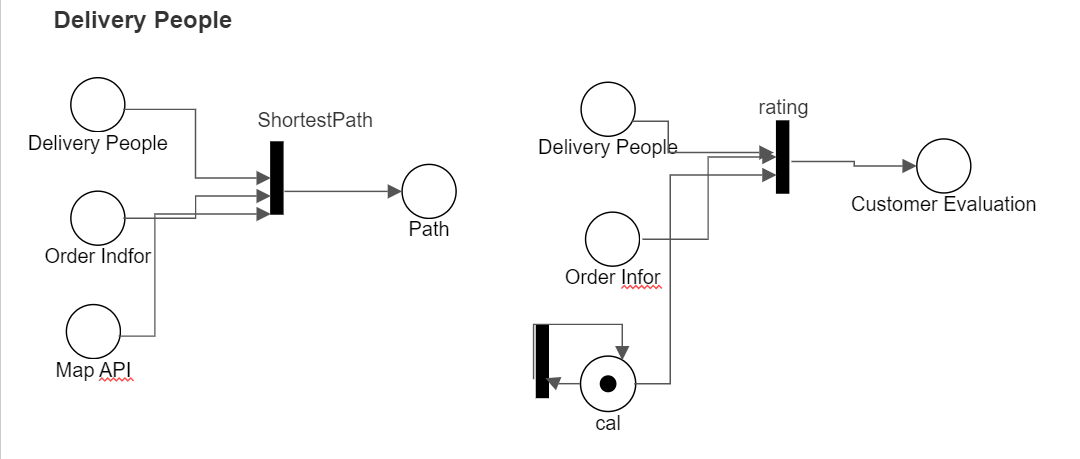
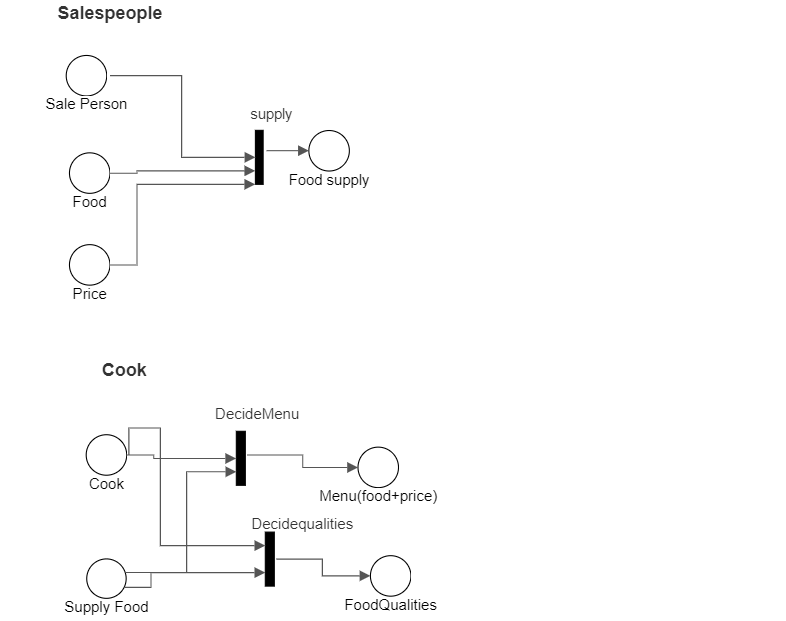
**Exceptional Scenario**

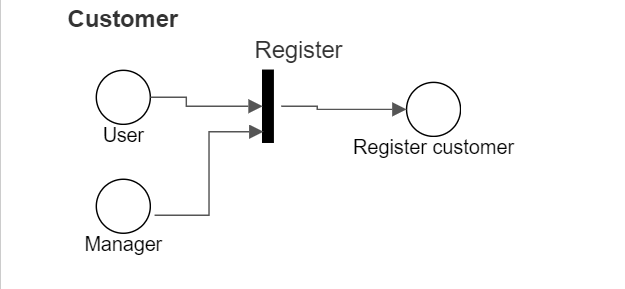
1. Delivery Man decides to take the food home for himself
2. He is fired immediately, and no one achieves the end goal
3. Delivery Man Quits
4. One less person to do Delivery or none to do delivery. Alternatively, there are other workers to deliver and bid on delivery however, depending on situation, end goal may not be reached

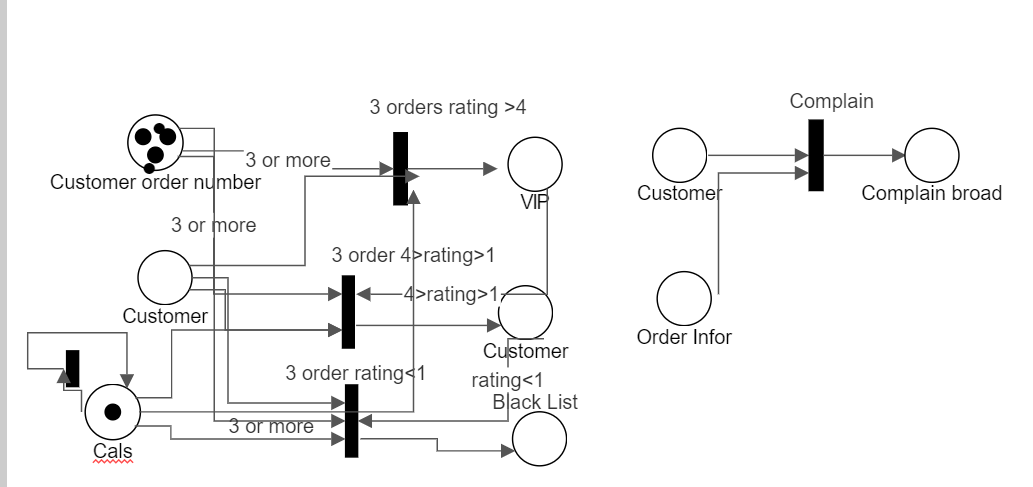
**2.2 Collaboration Class Diagram for our Use cases**

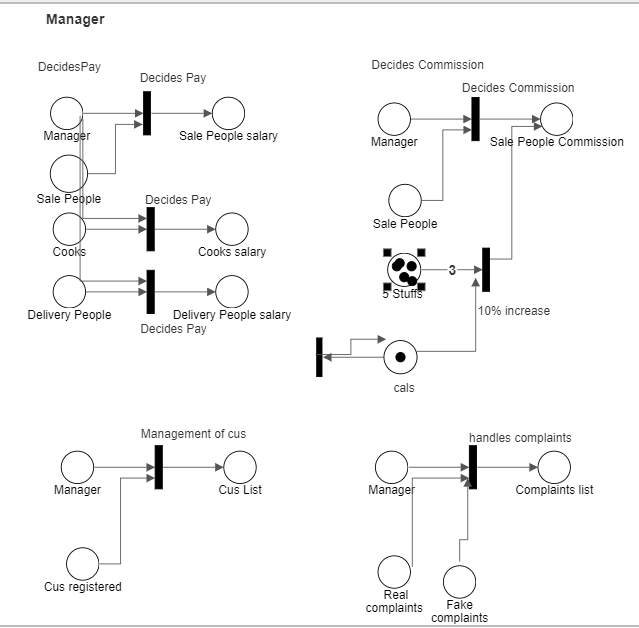
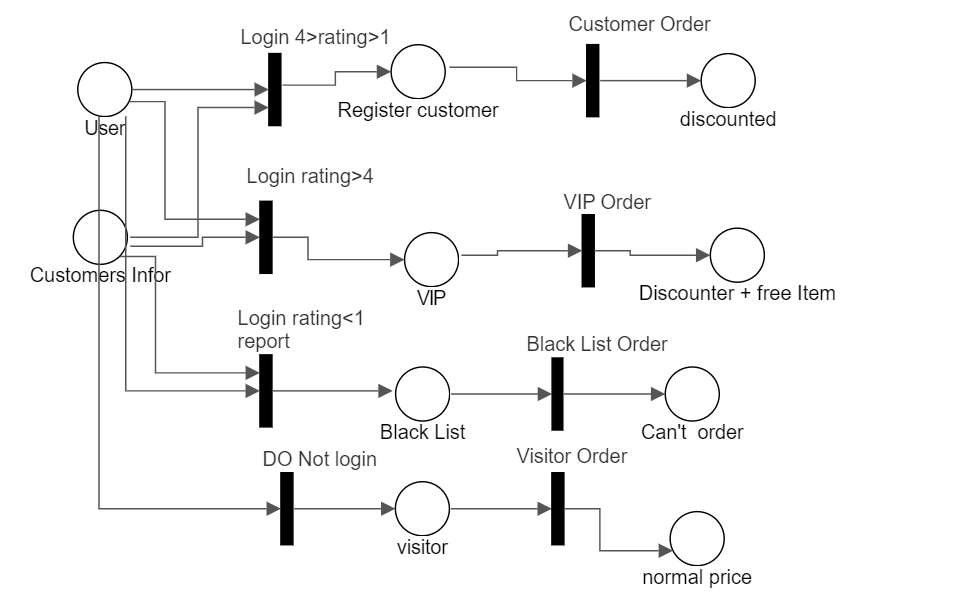


**2.3 Petri-Net Diagram for our Use cases**





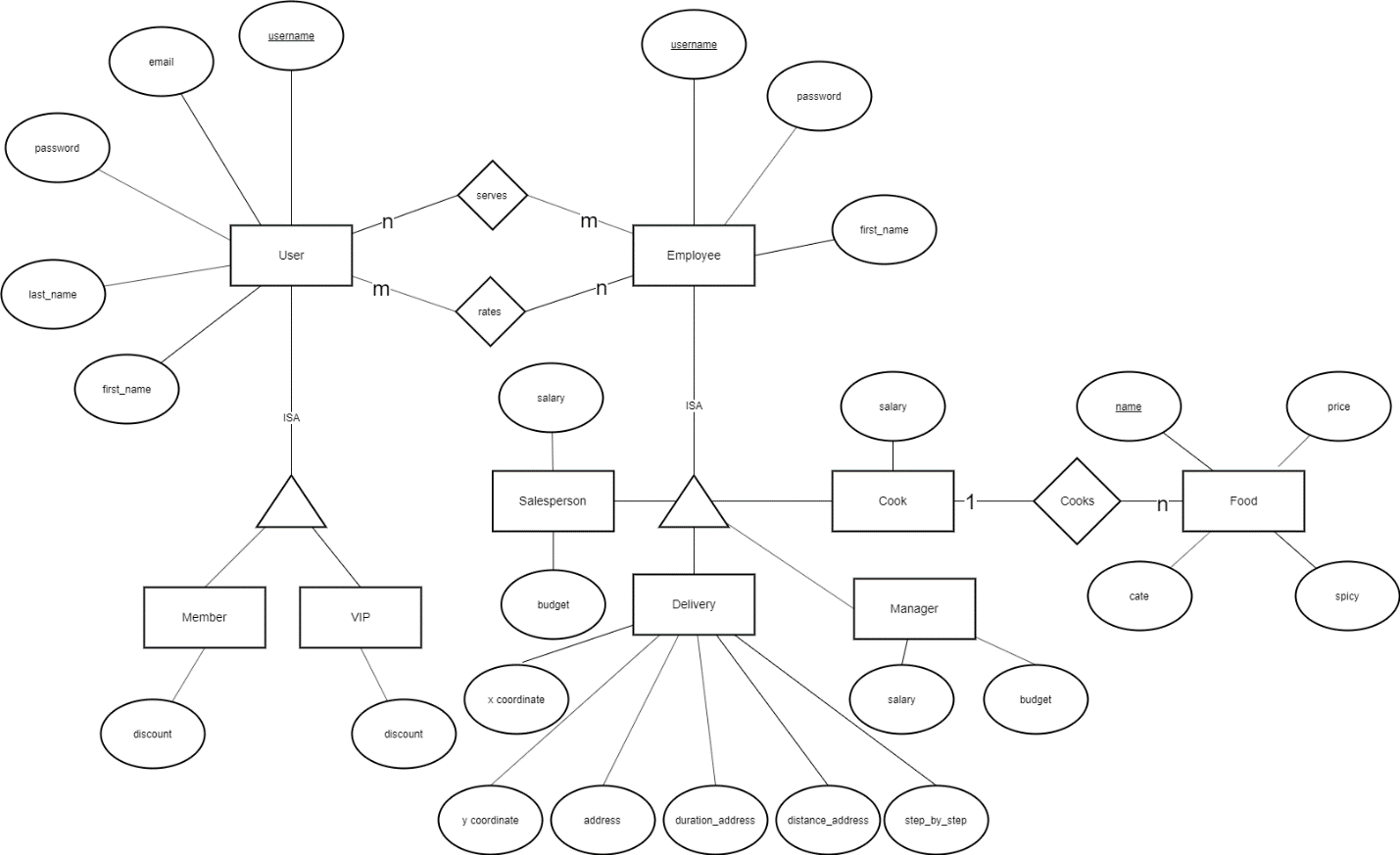




**3. E-R diagram**

*In this section, there will be an entity-relationship diagram showing the attributes our actors have and what schemas are currently available.*

**3.1**



**4. Pseudo-Code Delineating functions**

*This section details what goes on in the back-end of our software. We will be showcasing our methods that we use to help our functions run properly and there will be descriptions on what each of our class files do*

**4.1 Food.py**

*The Food class is used to instantiate the different Foods that we may have on our restaurant menu. We have things such as the category it will belong to, the quality of the ingredients going into the food as well as the pricing and name*

**Class Food():**

1. **\_\_init\_\_** takes in self, name, price, cate, spicy, quantity and quality as inputs:

Instantiate new Food object with

self.name = name

self.price = price

self.cate = cate

self.spicy = spicy

self.quantity = quantity

self.quality = quality

1. **getName** takes in self as input:

Returns the name of this Food object instance

1. **getPrice** takes in self as input:

Returns the price of this Food object instance

1. **getCate** takes in self as input:

Returns the price of this Food object instance

1. **getSpice** takes in self as input:

Returns a boolean determining whether the Food is spicy or not

1. **getQuantity** takes in self as input:

Returns the quantity that this Food object instance has

1. **getQuality** takes in self as input:

Returns the quality of this Food object instance

1. **setQuality** takes in self and quality as inputs:

Sets self.quality to quality

1. **setQuantity** takes in self and quality as inputs:

Sets self.quantity to quantity

**4.2 Employee.py**

*The Employee python file has 5 classes with the parent class being Employee and the 4 sub classes being Cooks, Delivery, Salesperson, and Manager which all inherit Employee because they are Employees.*

**Class Employee():**

1. **\_\_init\_\_** takes in self, first\_name, username, and password as input:

Instantiate an Employee object with

self.first\_name = first\_name.title() which capitalizes the first char

self.username = username

self.password = password

1. **getFirst** takes in self as input:

Returns the first name of this Employee object instance.

1. **getUser** takes in self as input:

Returns the username of this Employee object instance.

1. **getSalary** takes in self as input:

Returns the salary of how much this Employee object instance makes.

1. **getBudget** takes in self as input:

Returns the budget of how much this Employee object instance can spend.

1. **increaseSalary** takes in self as input:

Sets this Employee object instance salary to current salary \* 1.1 so 10% increase.

Then it returns the new salary of this Employee.

**Class Delivery(Employee):**

1. **\_\_init\_\_** takes in self, first\_name, username, password, and address as inputs:

1. Calls the super \_\_init\_\_ function from Employee for first\_name, username and password.
2. We create a new list called step\_by\_step and decoded, the first one contains the delivery instructions for this Delivery person and the 2nd one includes the polylines which is needed for me to draw the lines on the map later.
3. Set the variable gmaps to retrieve information from the Google API by using an API key.
4. Set geocode\_result to retrieve geocode information regarding this delivery person’s current address.
5. Set x to the latitude of this person’s current address from Google API.
6. Set y to the longitude of this person’s current address from Google API.

2. **create\_direction\_result** takes in self and a destination address as input:

1. Set direction\_results to the directions retrieved from Google Direction API with this delivery person’s address as start address and the destination address from input.
2. Set duration\_to\_address to the duration it takes to get to the destination address by calling GoogleMapParser.get\_duration.
3. Set distance\_to\_address to distance to destination address by calling GoogleMapParser.get\_distance.
4. Set step\_by\_step to a List of steps the delivery man should take to get to the destination by calling GoogleMapParser.get\_step\_by\_step\_directions.
5. Set polylines to the List of polylines retrieved from Google Directions API by calling get\_polyline from GoogleMapParser.
6. Set decoded to the decoded polylines by calling GoogleMapParser.decode\_polyline.
7. Returns nothing, but sets all the necessary information for this Delivery person instance to use to reach the customer.

3. **get\_duration\_to\_address** takes in self as input:

Returns the duration it takes this Delivery person object to reach the customer.

4. **get\_distance\_to\_address** takes in self as input:

Returns the distance it takes this Delivery person object to reach the customer.

5. **getX** takes in self as input:

Returns the latitude coordinate of this Delivery object.

6. **getY** takes in self as input:

Returns the longitude coordinate of this Delivery object.

**Class Cook(Employee):**

1. **\_\_init\_\_** takes in self, first\_name, username, and password as inputs:

1. Calls the super \_\_init\_\_ function from Employee to instantiate new Cook with first\_name, username and password from inputs.
2. Sets salary to 10.

2. **addFood** takes in a menuList and a Food object as inputs:

Appends the Food object into the menuList.

3. **removeFood** takes in a menuList and name as inputs:

Removes the Food object in menuList which has name as it’s name.

4. **changeFoodQuantity** takes in menuList, name and quantity as inputs:

1. Iterate through the menuList for the Food with the same name as input.
2. Once found, the function will call setQuantity(quantity) from the Food instance to

set the new quantity.

5. **changeFoodQuality** takes in menuList, name and quality as inputs:

1. Iterate through the menuList for Food with the same name as input.
2. Once found, the function will call setQuality(quality) from the Food instance to set

the new quality.

**Class Salesperson(Employee): More work needs to be done**

1. **\_\_init\_\_** takes in self, first\_name, username, and password as inputs:

1. Calls the super \_\_init\_\_ function from Employee to instantiate new Salesperson

with first\_name, username and password from inputs.

1. Set salary to 20.
2. Set budget to 600.

These are starting budget and salary.

**Class Manager(Employee): More work needs to be done**

1. **\_\_init\_\_** takes in self, first\_name, username, and password as inputs:

1. Calls the super \_\_init\_\_ function from Employee to instantiate new Manager with

first\_name, username and password from inputs.

1. Set salary to 250.
2. Set budget to 1000.

These are starting budget and salary.

**4.3 GoogleMapParser.py**

*Jia Ming created the GoogleMapParser file for simplicity and reusability when trying to parse information from the Google Geocode/Direction API. These functions here can retrieve certain important information from the big List/Dictionary from Google.*

1. **get\_formatted\_address** takes in a List/Dictionary named gecode as input:

Returns and print geocode[0][‘formatted\_address’] which contains a better formatted address than the one that the User inputted when signing up.

2. **get\_lat** takes in a List/Dictionary named gecode as input:

Returns and print geocode[0]['geometry']['location']['lat'] which contains the latitude location of the address.

3. **get\_long** takes in a List/Dictionary named as geocode as input:

Returns and print geocode[0]['geometry']['location']['lng'] which contains the longitude location of the address.

4. **get\_duration** takes in a List/Dictionary named direction as input:

Returns and print direction[0]['legs'][0]['duration']['text'] which contains the duration it takes from start location (delivery man) to get to the end location(customer) in minutes.

5. **get\_distance** takes in a List/Dictionary named direction as input:

Returns and print direction[0]['legs'][0]['distance']['text'] which contains the distance from start location to an end location in miles.

6. **get\_step\_by\_step** takes in a List/Dictionary named direction as input:

1. I have a for loop iterating through direction[0]['legs'][0]['steps'].
2. During each iteration i, we have direction[0]['legs'][0]['steps'][i]['html\_instructions'].
3. Use the Regex module to remove all HTML tags by re.compile(r'<[^>]+>').
4. Then we append each of those instructions into a List in which we return.
5. Returns the List of step by step instructions such as where to go or what street to

go for the person (delivery person).

7. **get\_polyline** takes in a List/Dictionary named direction as input:

1. I have a for loop iterating through direction[0]['legs'][0]['steps'].
2. During each iteration i, we append

direction[0]['legs'][0]['steps'][i]['polyline']['points'] to a List in which we return.

1. Return the list containing all the polylines, which I would need to draw the lines on the map.

8. **decode\_polyline** takes in a List of encoded polylines from the get\_polyline function:

1. Google API returns the polylines encoded.
2. We use a module called Polyline to decode these Google polylines into sets of

coordinates (latitude and longitude).

1. We do this by using a for loop and calling polyline.decode(polylines[i]).
2. Append these results to a new list and return it.

**4.4 Main.py**

*The purpose of the Main class file is to compile all of the other class files into one for organization. We instantiate the arrays and the classes that we need to help make things function properly in the main file and to call functions in other files more easily.*

1. **addDelivery** takes in a Delivery object called ‘employee’ as input:

1. Append the Delivery object to the ‘DeliveryPeople’ list
2. Print “Added (employee’s first name) to DeliveryPeople”

2. **addCook** takes in a Cook object called ‘employee’ as input:

1. Append the Cook object to the ‘Cooks’ list
2. Print “Added (employee’s first name) to Cooks”

3. **addSalesperson** takes in a Salesperson object called ‘employee’ as input:

1. Append the Salesperson object to the ‘Sales’ list
2. Print “Added (employee’s first name) to Sales”

4. **findCook** takes in a name string as input:

1. For each element in the ‘Cooks’ list:
2. If the current element’s (Cook’s first name) = ‘name’:
3. Print “Cook (Cook’s first name) has been found”
4. Return the ‘Cook’ object at the current element

5. **printCooks** takes no input:

a. Prints a list of all Cook’s first name from the List Cooks

b. If the list is empty, print that it is empty

6. **addMenuItem** takes in a Food object as input:

a. Append the Food object into the Menu List

b. Print “Added (food’s name) to Foods”

7. **printMenu** takes in no input:

a. Print “Printing list of Foods…”

b. If Menu List is empty, print that it is empty

c. Else, print the name of all Food in the Menu List

8. **getMemberSize** takes no input:

a. Return the size of the Members List

9. **getVIPsize** takes no input:

a. Return the size of the VIPMembers List

10. **addMember** takes in a Member object as input:

a. Append the Member object into the Members List

b. Print “Added (member’s first name) to Members”

11. **removeMember** takes in a Member object as input:

a. Remove the Member object from the Members List

b. Print “Removed (member’s first name) from Members”

12. **addVIP** takes in a VIPMember object as input:

a. Append the VIPMember object to the VIPMembers List

b. Print “Added (VIP’s first name) to VIPMembers”

13. **findMember** takes in a name string as input:

a. Iterate through the Members List

b. Print the Member’s first name if it has been found

c. Return the Member object that was found

14. **findVIP** takes in a name string as input:

a. Iterate through the VIPMembers List

b. Print the VIPMember’s first name if it has been found

c. Return the VIPMember object that was found

15. **promoteVIP** takes in a name string as input:

a. Iterate through the VIPMembers List

b. Print the VIPMember’s first name if it has been found

c. Call removeMember to remove that Member from the Members List

d. Call addVIP to add a new VIPMember to VIPMembers list with the same person

16. **printAllMembers** takes in no input:

a. Print “Printing list of Members and VIPMembers…”

b. If Members List has no elements, print that it is empty

c. If VIPMembers List has no elements, print that it is empty

d. Iterate through the Members List and print the first name of all Members

e. Iterate through the VIPMembers List and print the first name of all VIPMembers

**4.5 User.py**

*The Purpose of having a User class file is for the creation of the VIP member, the Guest and the Member. It holds the methods that we need in order to get specific information such as the users type or the discount that will be applied to their Menu*

**class User():**

**Function** ‘**\_\_init\_\_**’ takes in the parameters **‘self’, ‘first\_name’, ‘last\_name’, ‘username’,** ‘**password’, and ‘email’**

         Instantiate and **User** object with

**self.first\_name** = **first\_name.title()** which capitalizes the first character

**self.last\_name** = **last\_name.title()** which capitalizes the first character

**self.username** = username

**self.email** = email

**self.password** = password

1. ‘**getDiscount**’ takes in the parameter **‘self’**

                     Return the **‘discount’** of this **User** object instance

2. ‘**getFirst**’ takes in the parameter **‘self’**

                     Return the **‘first\_name’** of this **User** object instance

3. ‘**getLast**’ takes in the parameter **‘self’**

                     Return the **‘last\_name’** of this **User** object instance

4. ‘**getUser**’ takes in the parameter **‘self’**

                     Return the **‘username’** of this **User** object instance

5. ‘**getPass**’ takes in the parameter **‘self’**

                     Return the **‘password’** of this **User** object instance

6. ‘**getEmail**’ takes in the parameter **‘self’**

                     Return the **‘email’** of this **User** object instance

7. ‘**getType**’ takes in the parameter **‘self’**

                     Return the ‘**user\_type’** of this **User** object instance

**class Member(User):**

**Function ‘\_\_init\_\_’** that takes in the parameters **‘self’, ‘first\_name’, ‘last\_name’, ‘username’, ‘password’, and ‘email’**

Calls the super **\_\_init\_\_** function from **User** for **‘first\_name’, ‘last\_name’, ‘username’, ‘password’, and ‘email’**

 Set **‘discount’** = 0.85

 Set **‘user\_type’** = 1

**class VIP(User):**

**Function ‘\_\_init\_\_’** that takes in the parameters **‘self’, ‘first\_name’, ‘last\_name’, ‘username’, ‘password’, and ‘email’**

Calls the super **\_\_init\_\_** function from **User** for **‘first\_name’, ‘last\_name’, ‘username’, ‘password’, and ‘email’**

Set ‘discount’ = 0.75

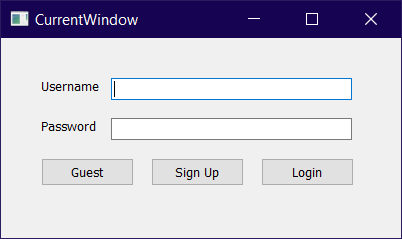
Set ‘user\_type’ = 2

**5. GUI’s**

*This section showcases all of our major GUI’s as well as a prototype of one of our functionalities. The prototype demonstrates our food ordering menu and how It will work.*

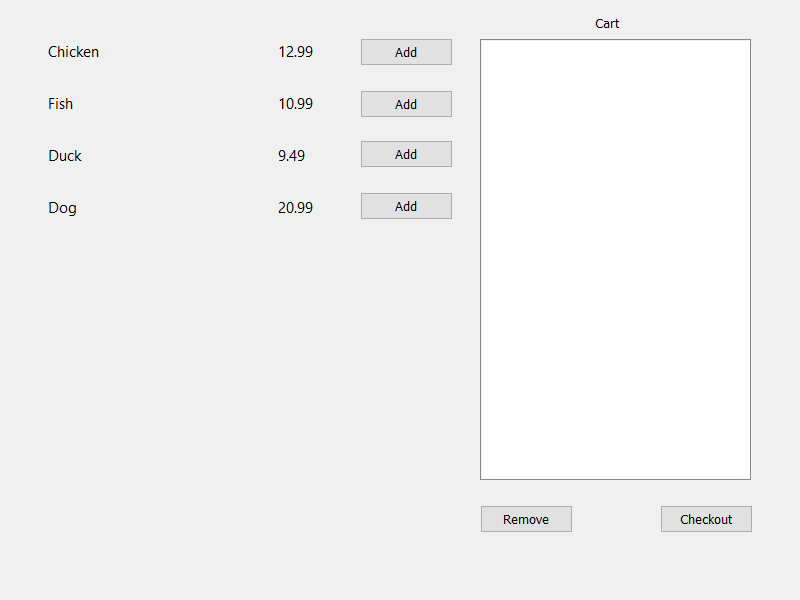
**5.1 All major GUI’s**

**Login S**

****

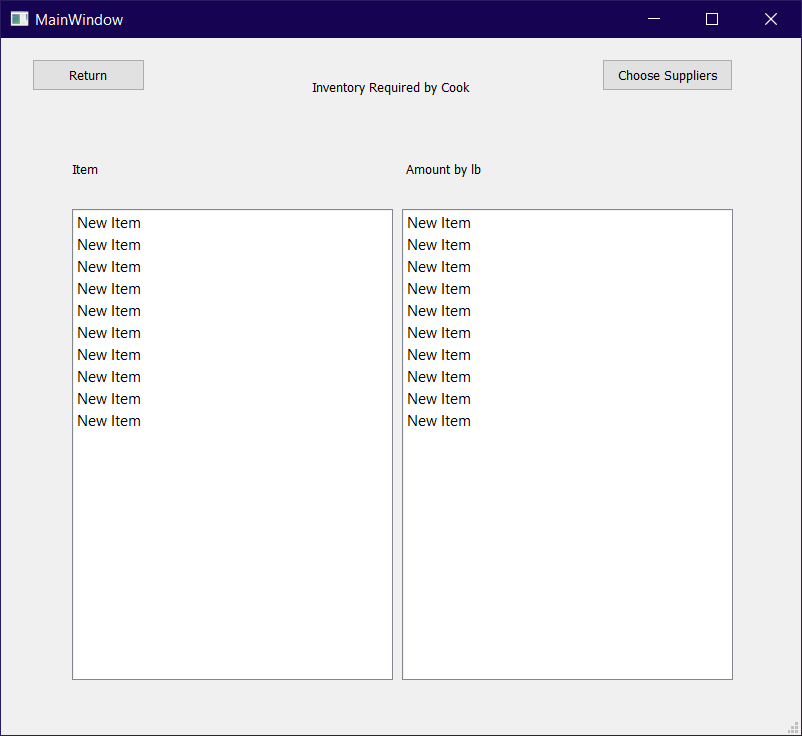
**Login and Menu screen, The login will show different GUI based on the input of the users, username and password. Each different type of user has a user type that will determine which GUI will be shown**

**Menu Screen**

****

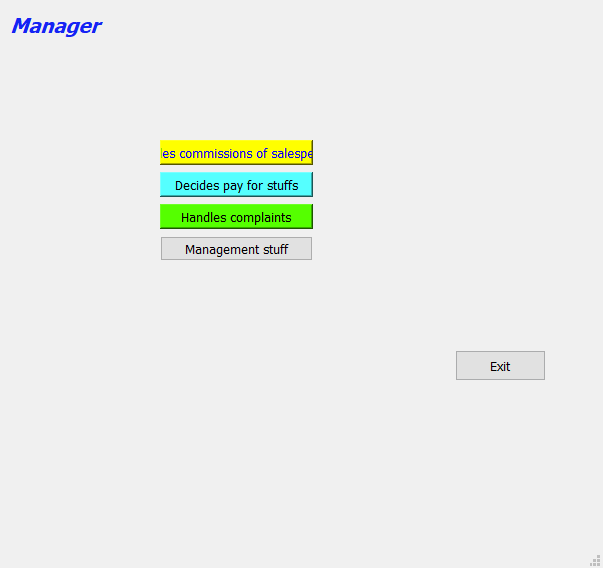
**This screen demonstrates how the menu screen will look like. There will be a Checkout and cart removal button**

**Salesperson Screen**

****

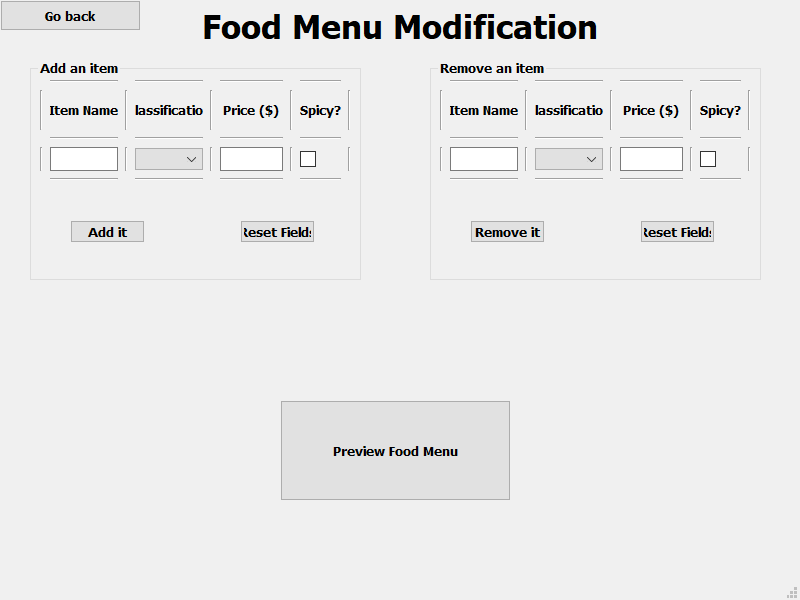
**The Salesperson GUI will show the inventory of the items as well as a button for suppliers that they will purchase goods from depending on the quality**

**Manager Screen**

****

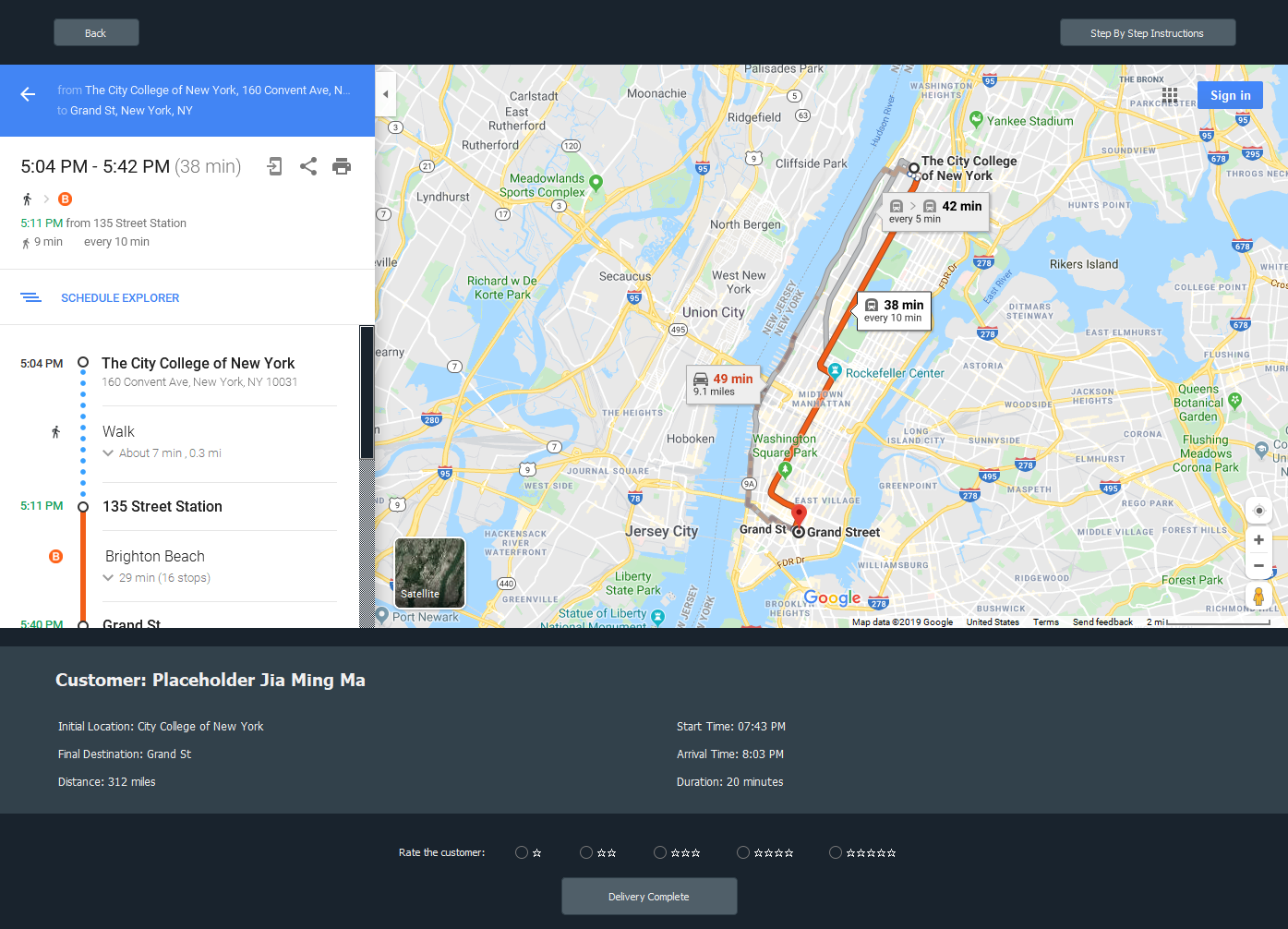
**This will be the base Manager screen, This is not finalized and many changes are due for the style of the Managers GUI as it does not fit with everyone elses style.**

**Chefs Menu Modification**

****

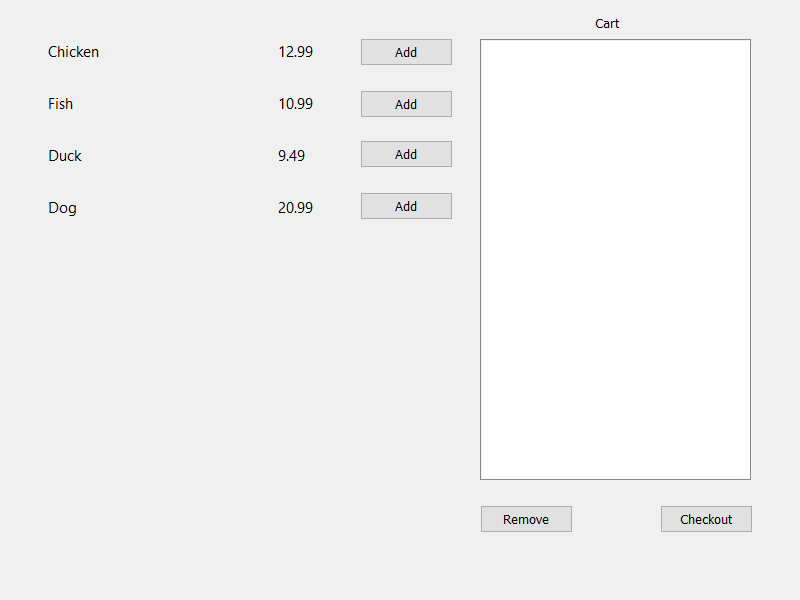
**Basic demonstration of the Chefs GUI, This will be how the chef will modify items in the menu and may be imported to the Managers tab. This is not final and changes may happen due to GUI style**

**DeliveryPerson Routing and Idea for ETA**

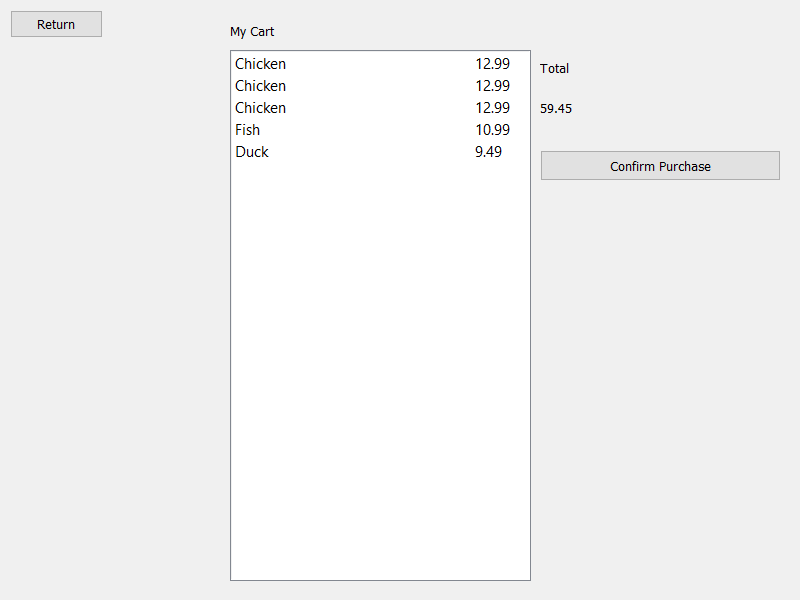
****

**This will be the Delivery Window for the Delivery person to look after they receive an order after bidding. It uses Google Map API to grab information, providing them with all the information they need. Once the item is delivered, they can rate the customer and click finish delivery.**

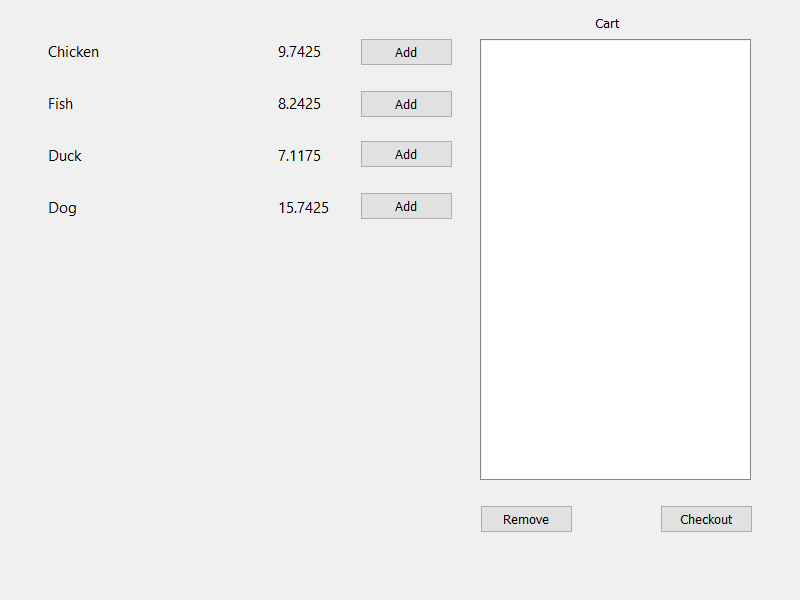
**5.2 Ordering Prototype**

****

**You can choose to remove or add items into the cart, on selection you can remove the item using the given remove button. After your cart is completed, You may checkout**

****

**If the items you have chosen are not to your liking or not all that you want, you can press the return button to go back and remove or add more items into your cart. Finally, after confirming purchase, Order will be sent to Cooks and Delivery person.**

****

**This GUI is a demonstration of the applied discount on a VIP member. The GUI shown above demonstrates a guest users discount, which is none and this GUI demonstrates a 25% discount on items for the VIP members.**

**What is still required to be done is the implementation of most popular item on the menu as well as who created the menu.**

**6. Minutes of group meetings and concerns on teamwork**

11/7

* Discussion plans on how to break down and partition work, what the requirements of the software is and getting GitHub setup between everyone

11/12

* Discussion plans on GUI designs so far and what we can do to improve on what we already have and if we would possibly need to add anything onto it. Creating general design and deciding to design all GUI like each other’s when completed

11/17

* meeting on how we want to store data, discussion plans on how to approach functionalities, final decision came to decide that back-end logic would be the most convenient

11/18

* Meeting on the Administrators and their purposes, what they do and how we should approach it given the uniqueness of the firing system and discussions food quality and how to deal with inventory

11/19

* Discussions on how to implement the bidding system and the routing system for delivery people. Thoughts on using google API to get a live map of our neighborhood for finding best route

11/23

* Meeting on the method of shortest path using google maps as well as organization of our code.

Successfully organized code to make things more reader friendly and accessible.

11/25

* Meeting on some classic layouts and style we could be using for our project as well as well as how the bidding system will work.

**Summary:** Overall concern with the organization in the team and how to successfully partition work equally. Had a hard time planning out what was needed for the project however, Group has great communication and gets problems solved quickly

**7. Github Link**

<https://github.com/Chuezhang2278/Foodies_322_Final_Project>