Happy Eats Inc.

Happy Eats
System Design For Food Order and Delivery System

Version <2.0>

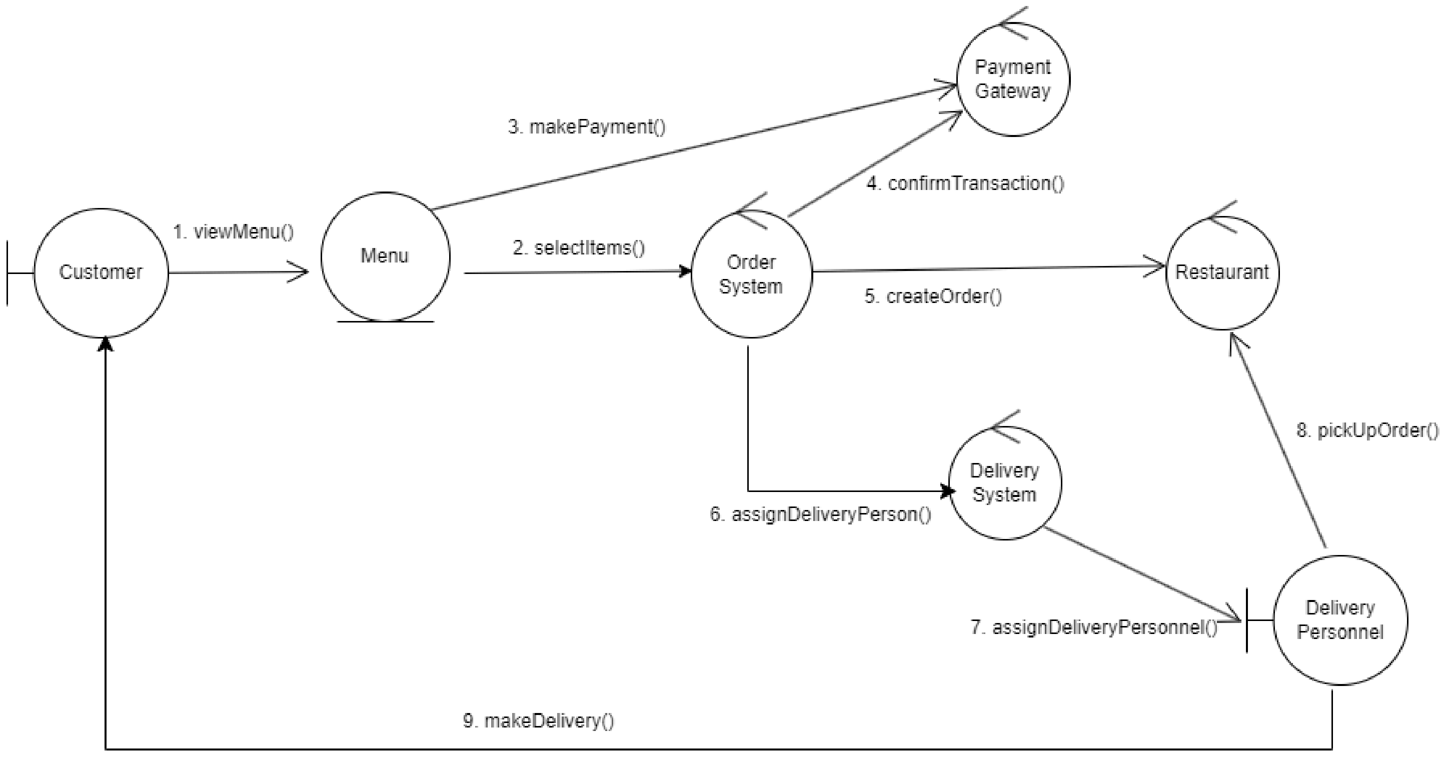
Table of Contents

1. Introduction

This report serves as a comprehensive guide to the design phase of our food delivery application. Within its pages, you will find detailed insights into the system's architecture, including collaboration class diagrams, use case scenarios, E-R diagrams, method designs, system screens, and memos from group meetings.

1.1 - Overall Class Collaboration Diagram

The Collaboration Class diagram below represents an overall overview of the Happy Eats food delivery application. It details the interaction between customers and the process that happens when an order is placed on the application



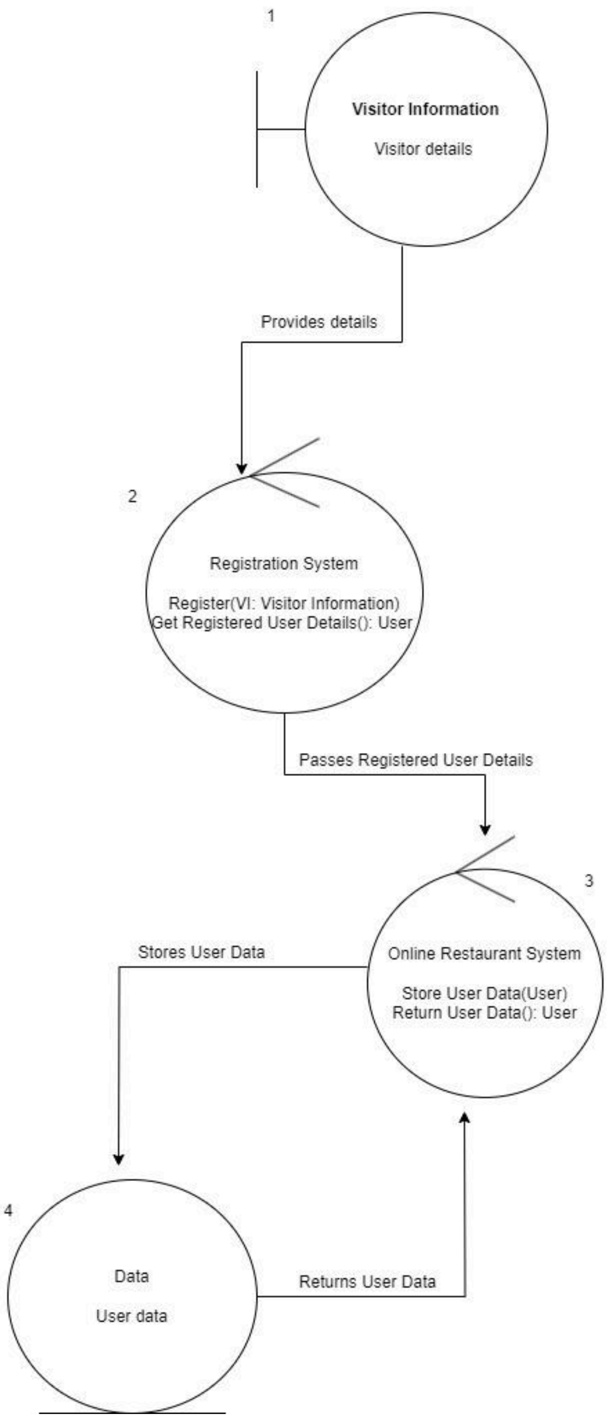
2. Use Case Diagrams

In this part, we present a detailed examination of each use case. For every scenario, collaboration class diagrams and State diagrams are provided to facilitate a clearer comprehension of how the system operates.

2.1 - Collaboration Class Diagrams

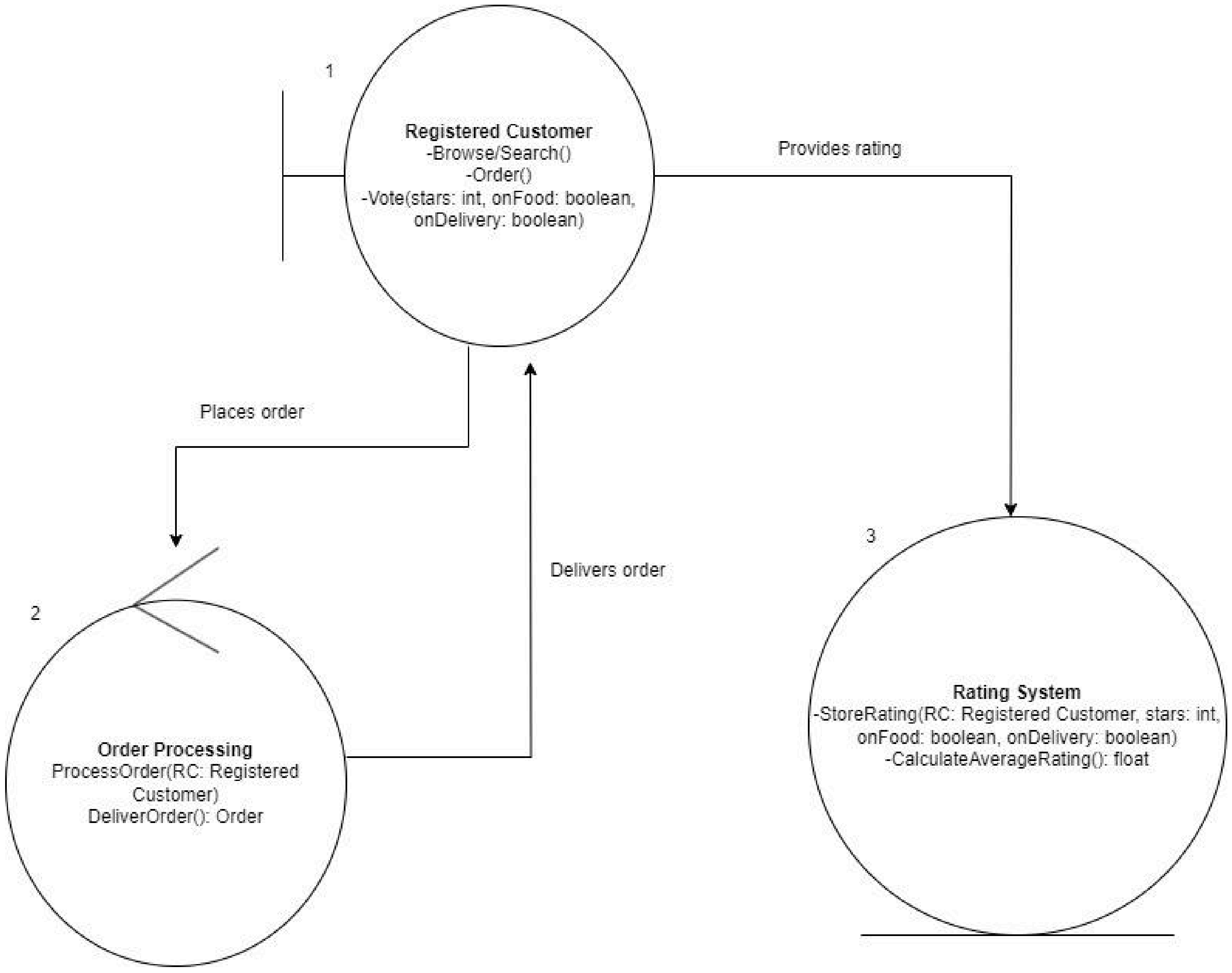
2.1.1 Visitor Information

This outlines the information accessible to visitors, yet the system persistently requires registration. This protocol is implemented in the Online Restaurant System to solicit information from registered users.



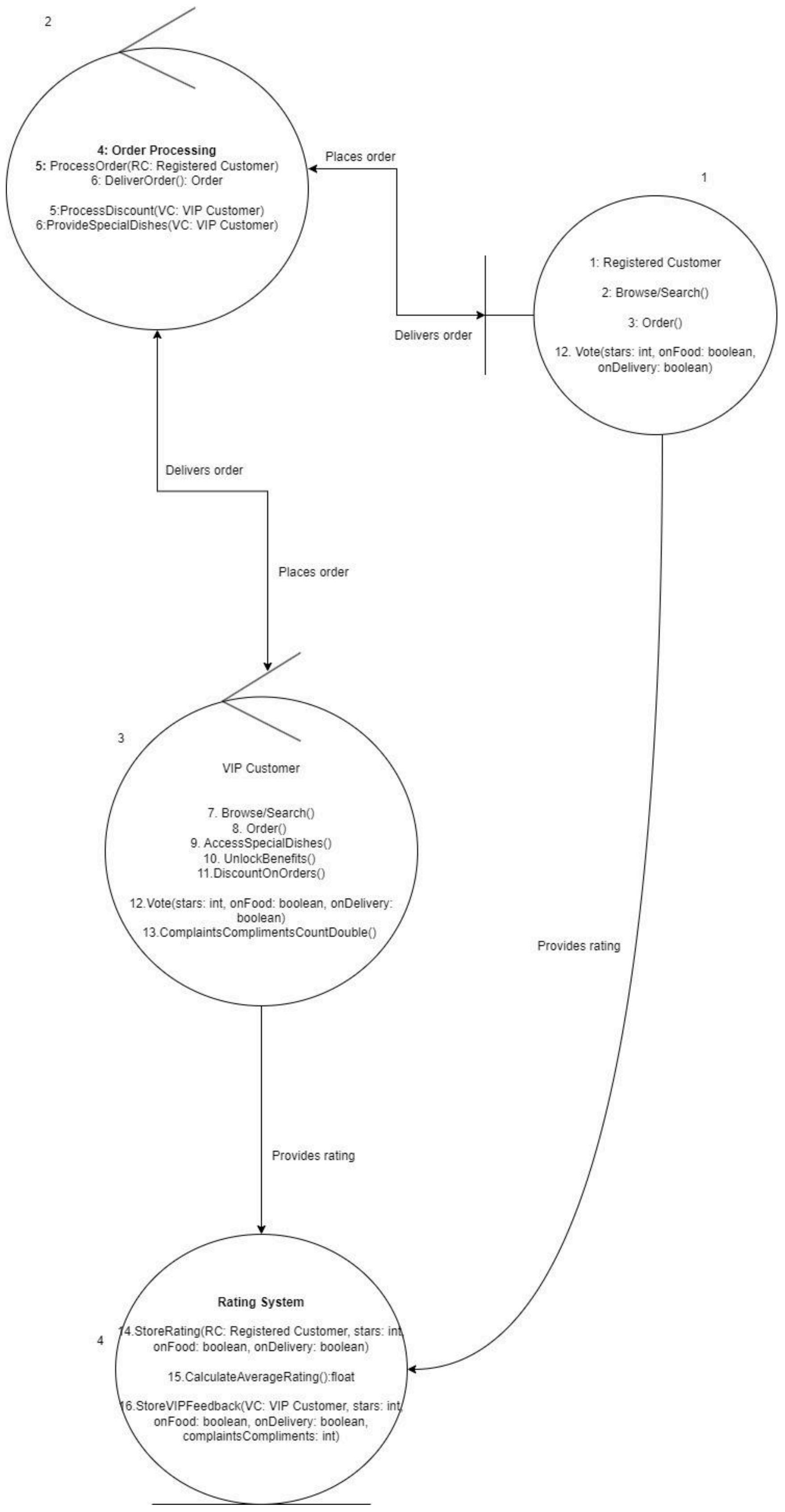
2.1.2 Use Case Analysis And Customer Interactions

In this section, registered customers, the system will enable them to browse and search for food, place orders, and rate their experience from 1 (lowest) to 5 (highest) stars, assessing both the food quality and the delivery service separately


1
Registered Customer
-Browse/Search()
-Order()
-Vote(stars: int, onFood: boolean, onDelivery: boolean)
Provides rating
Places order
Delivers order
2
Order Processing
ProcessOrder(RC: Registered Customer)
DeliverOrder(): Order
3
Rating System
-StoreRating(RC: Registered Customer, stars: int, onFood: boolean, onDelivery: boolean)
-CalculateAverageRating(): float

2.1.3 Customer Account Status(VIP)

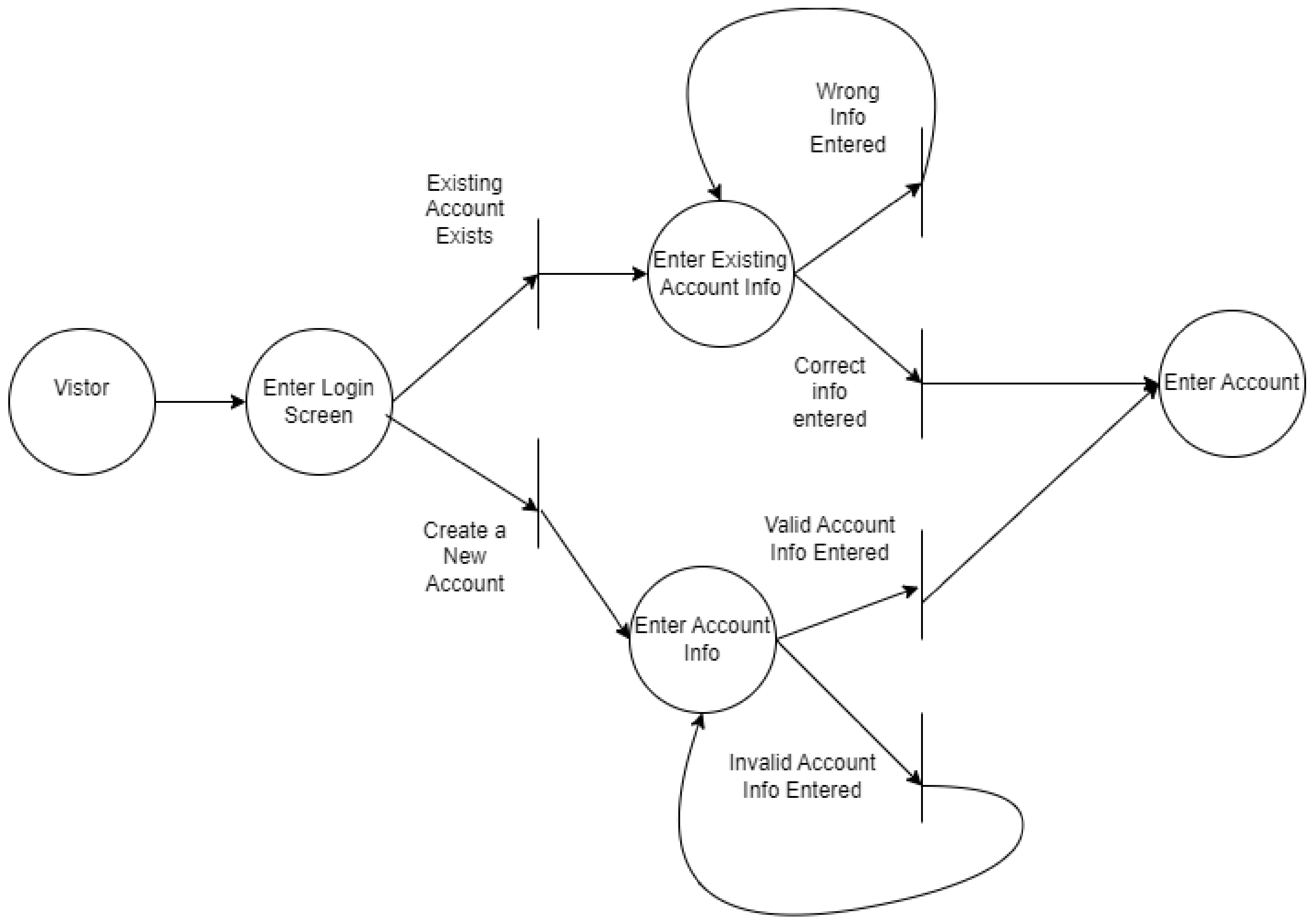
For VIP customers who have either spent over $500 or placed 50 orders as registered customers—whichever milestone is reached first—the system will unlock exclusive benefits and privileges to enhance their shopping experience.


2
4: Order Processing
5: ProcessOrder(RC: Registered Customer)
6: DeliverOrder(): Order
5:ProcessDiscount(VC: VIP Customer)
6:ProvideSpecialDishes(VC: VIP Customer)
Places order
1
1: Registered Customer
2: Browse/Search()
3: Order()
12. Vote(stars: int, onFood: boolean, onDelivery: boolean)
Delivers order
Delivers order
Places order
3
VIP Customer
7. Browse/Search()
8. Order()
9. AccessSpecialDishes()
10. UnlockBenefits()
11.DiscountOnOrders()
12.Vote(stars: int, onFood: boolean, onDelivery: boolean)
13.ComplaintsComplimentsCountDouble()
Provides rating
Provides rating
4
Rating System
14.StoreRating(RC: Registered Customer, stars: int, onFood: boolean, onDelivery: boolean)
15.CalculateAverageRating():float
16.StoreVIPFeedback(VC: VIP Customer, stars: int, onFood: boolean, onDelivery: boolean, complaintsCompliments: int)

2.2 - Petri-net Diagrams

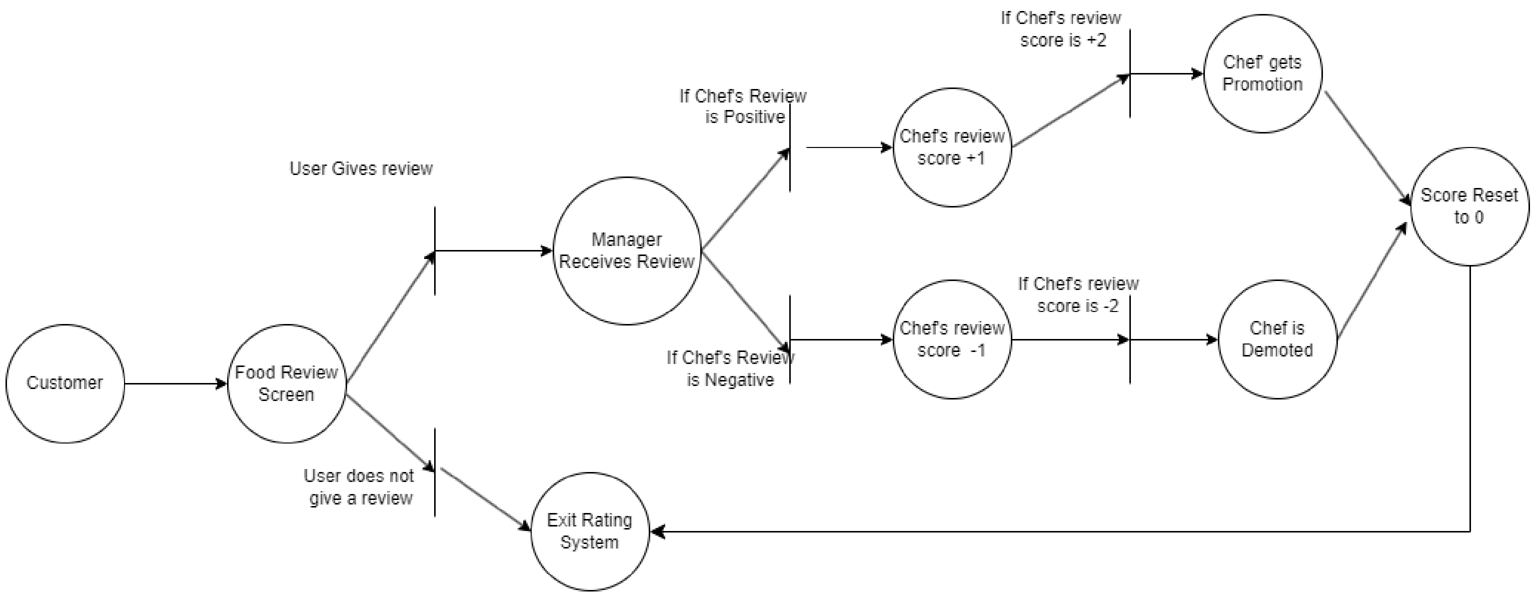
2.2.1 - User Login

This Petri-net diagram outlines the login process for a user accessing the system. This diagram summarizes what will occur in the system whether a returning user is accessing the system or if they are a new user.



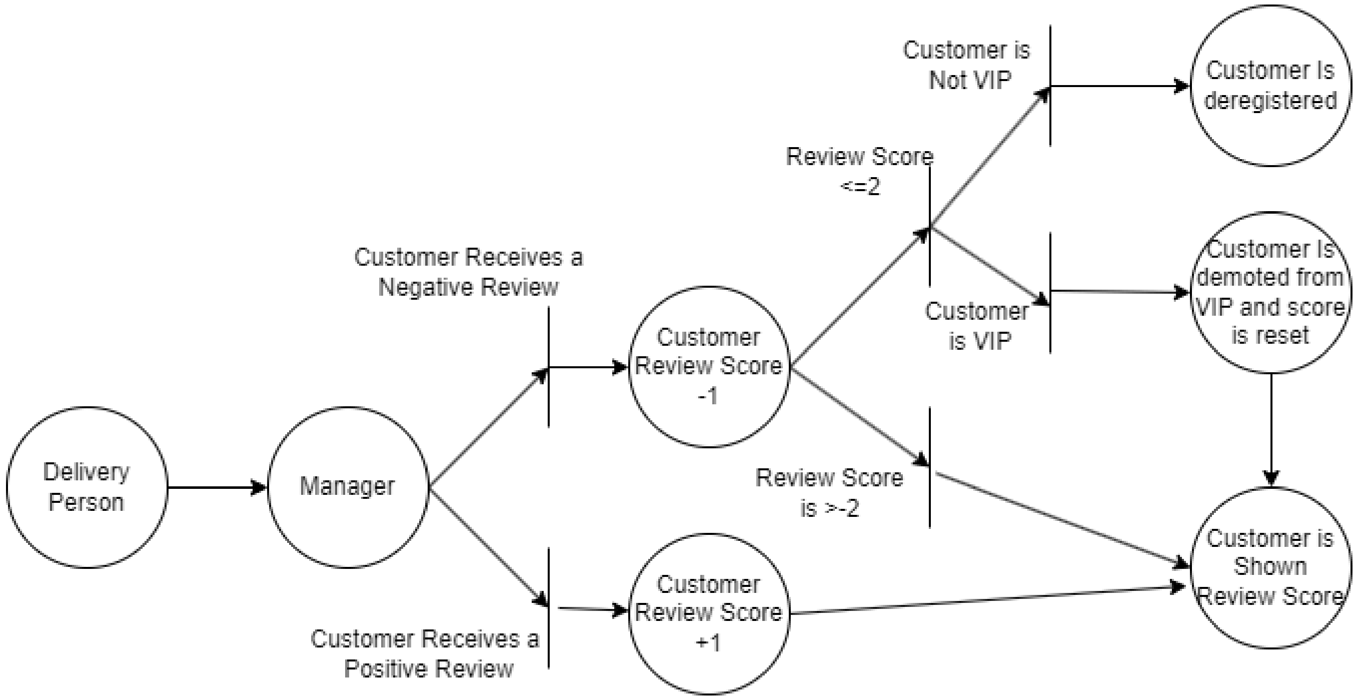
2.2.2 - Restaurant Rating System

This diagram outlines the Customer review system. Restaurant Chef's salaries are adjusted based on their ratings. Each Chef is given a review score which begins at 0. Each time they receive a review, their score will be changed based on if the Review is positive or negative. If their score is >= 2 then their salary is increased. If their score is <= -2 then their salary is decreased.



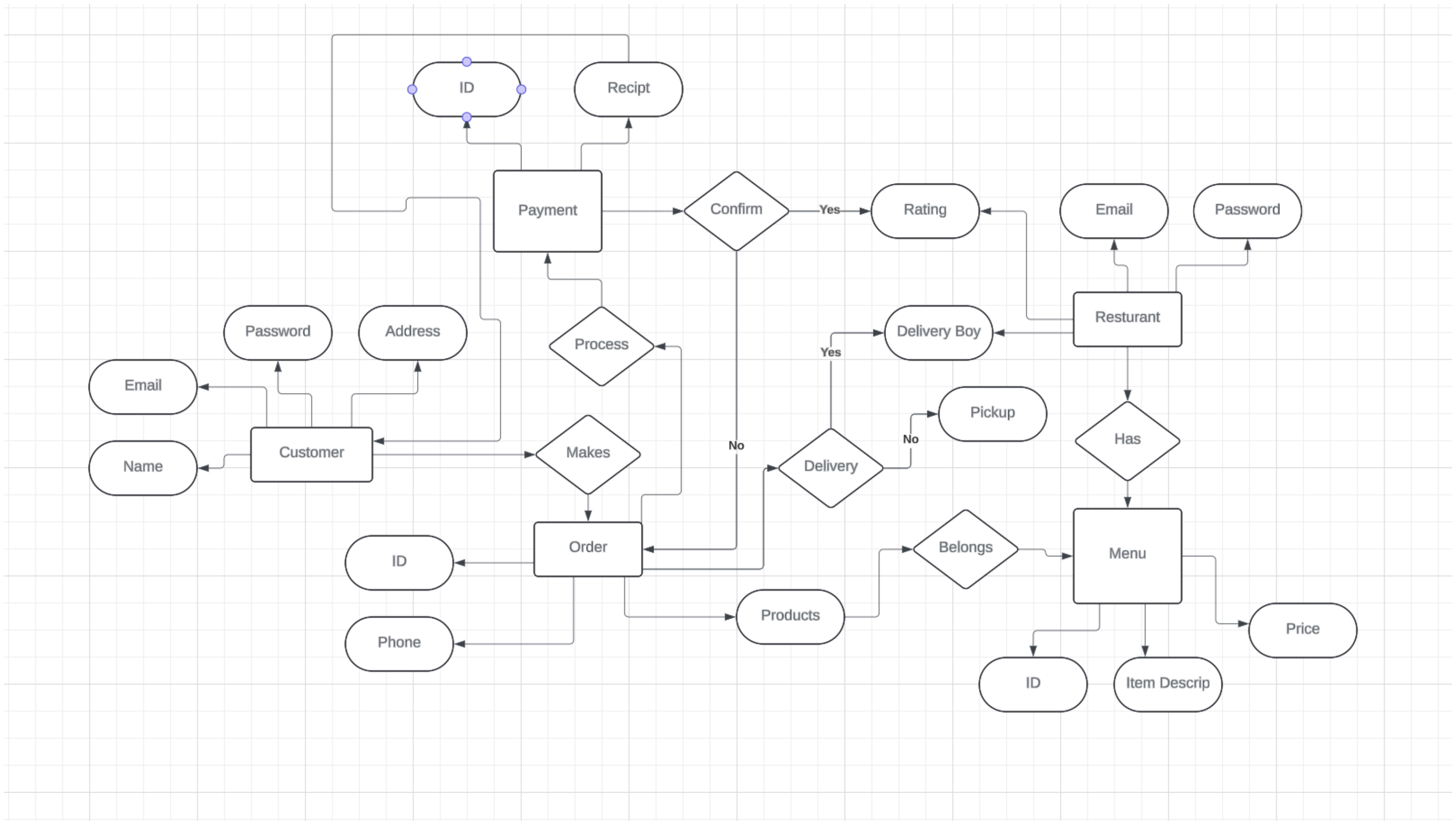
2.3.3 - Customer Rating System

This diagram outlines the system where the delivery people for restaurants may review customers. Similar to the Restaurant Rating System, Customers are given a review score where if their score is <= 2 then their account may demoted/deregistered depending if they are a VIP Customer or not.



3. E-R Diagram

This Entity-Relationship diagram outlines the data structure for the Happy Eats application. The diagrams shows the relationship between different systems within the application and how they interact with eachother. Customers, with detailed profiles, place orders that follow a pathway through processing, payment, and optional delivery. Restaurants manage menus with itemized products, each with unique identifiers and prices, facilitating a customized ordering experience. The diagram serves as a crucial blueprint for the application



4. Detailed Design

4.1 - Pseudo-code for Main Functionalities

GUI with personalization

System Overview/User Groups:
This system organizes the roles and responsibilities of chefs, delivery personnel, food importers, and a manager within a food service platform, facilitating tasks such as menu decisions, order delivery, food item importation, and staff and customer management.

```
# Set up different roles in the food service system

# Chef role
Make Chef:
  Chef has chef_id and name
  Chef can decide_menu with dishes:
        Say "Menu decided by Chef [name]: [dishes]"

# Delivery Person role
Make DeliveryPerson:
  Delivery Person has delivery_id and name
  Delivery Person can deliver_food with order:
        Say "Order [order] delivered by [name]"

# Food Importer role
Make FoodImporter:
  Importer has importer_id and name
  Importer can import_food with food_items:
        Say "Food items imported by [name]: [food_items]"

# Manager role
Make Manager:
  Manager has manager_id and name
  Manager can do these:
        - process_registration with customer:
        Say "Customer [customer] registered by Manager [name]"
        - handle_compliments_complaints with feedback:
        Say "Feedback processed by Manager [name]: [feedback]"
        - manage_staff with action and staff_member:
        Say "[action] action taken by Manager [name] on [staff_member]"

# Example of how everyone works together
# Make users
Chef alice = Make Chef(1, "Alice")
Chef bob = Make Chef(2, "Bob")
DeliveryPerson charlie = Make DeliveryPerson(1, "Charlie")
DeliveryPerson diana = Make DeliveryPerson(2, "Diana")
FoodImporter eve = Make FoodImporter(1, "Eve")
FoodImporter frank = Make FoodImporter(2, "Frank")
Manager grace = Make Manager(1, "Grace")

# Chefs decide on what to cook
Say alice.decide_menu(["Pizza", "Salad"])
Say bob.decide_menu(["Sushi", "Ramen"])

# Delivery persons deliver food
Say charlie.deliver_food("Order 123")
Say diana.deliver_food("Order 456")

# Importers bring in food supplies
Say eve.import_food(["Tomatoes", "Basil"])
Say frank.import_food(["Fish", "Rice"])

# Manager signs up a new customer
Say grace.process_registration("John Doe")

# Manager deals with a compliment
Say grace.handle_compliments_complaints("Great service!")

# Manager deals with staff
Say grace.manage_staff("Fire", "Chef Bob")
```

Customer Interaction System

system for managing customer interactions on a food service platform, distinguishing between regular and VIP customers, where customers can browse menus, order food, rate services, participate in discussions, and VIPs can receive discounts and have their feedback counted more significantly.

```
# Food Ordering System

# Basic Customer
Make Customer:
  Customer has id, name, orders, and money_spent
  Customer can:
        - look_at_menu:
```

```
        Say "Menu looked at by [name]"
        - buy_food with dish and price:
        Add 1 to orders
        Add price to money_spent
        Say "[name] bought [dish] for $[price]"
        - give_stars to food and delivery:
        Say "[name] gives [food stars] stars to food and [delivery stars] stars to delivery"
        - talk_about with topic:
        Say "[name] talks about [topic]"

# Special VIP Customer
Make VIPCustomer from Customer:
  VIP has vip_status as not a VIP
  VIP can:
        - check_if_vip:
        If money_spent over $500 or orders over 50:
        Make vip_status as VIP
        Say "[name] is now a VIP"
        Else:
        Say "[name] is not a VIP"
        - buy_food with dish and price:
        If vip_status is VIP:
        Reduce price by 10%
        Use buy_food from Customer
        - special_feedback with feedback:
        Say "VIP [name] says [feedback] (extra important)"

# Example of using the system
# Make customers
normal_customer = Make Customer(1, "Ivy")
vip_customer = Make VIPCustomer(2, "James")

# Normal customer actions
Say normal_customer.look_at_menu()
Say normal_customer.buy_food("Pasta", 20)
Say normal_customer.give_stars(4, 5)
Say normal_customer.talk_about("Best dinner dishes")

# VIP customer actions
Say vip_customer.check_if_vip()
Say vip_customer.buy_food("Exclusive Sushi", 30)
Say vip_customer.special_feedback("Loved the sushi!")
Say vip_customer.talk_about("Quick deliveries")
```

Surfer Registration and Interaction System
This Outlines a system where surfers (potential customers) can browse food menus, view ratings, and apply for registration through a manager, who decides on their application based on a required deposit.

```
# Surfer Registration System

# Surfer
Make Surfer:
  Surfer has id and name
  Surfer can:
          - look_at_menu:
          Say "Menu looked at by [name]"
          - see_ratings:
          Say "Ratings seen by [name]"
          - try_to_register with deposit and a manager:
          If deposit is $100 or more:
          If manager says yes:
          Say "[name] is now registered"
          Else:
          Say "Manager said no to [name]"
          Else:
          Say "[name] can't register because not enough money"

# Manager
Make Manager:
  Manager can:
          - say_yes_or_no to deposit:
          If deposit is $100 or more:
          Return "yes"
          Return "no"

# Example of what happens
# Make a surfer and a manager
surfer = Make Surfer(1, "Mia")
manager = Make Manager(1, "Grace")

# Surfer looks at menu and sees ratings
Say surfer.look_at_menu()
Say surfer.see_ratings()

# Surfer tries to register
Say surfer.try_to_register(120, manager)  # Has enough money
Say surfer.try_to_register(80, manager)   # Not enough money
```

GUI and Personnel Management for Food Service

GUI class that personalizes menu displays for users, a Personnel Management class that handles employee feedback by promoting or demoting them based on customer feedback, and an enhanced Customer class with options for dining preferences. The system enhances user experience and manages staff performance in a food service environment.

```
# Food Ordering System with Fancy Screens

# Fancy Screen for Showing Stuff
Make FancyScreen:
  FancyScreen knows the user
  FancyScreen can:
          - show_menu:
          Say "Menu for [user name]"
          - show_favorites:
          Say "Top 3 favorites for [user name]"
          - show_best_dishes:
          Say "Best dishes shown on screen"
          - log_in with password:
          Say "[user name] logged in with password"

# Handling Staff
Make StaffHandler:
  StaffHandler keeps track of good and bad things
  StaffHandler can:
          - get_feedback from someone about good or bad:
          If bad:
          Add 1 bad point to someone
          If bad points are 2:
          Say "[someone's name] is in big trouble"
          Say "[someone's name] got a bad mark"
          If good:
          Add 1 good point to someone
          If good points are 2:
          Say "[someone's name] is doing great!"
          Say "[someone's name] got a good mark"

# Chef that can be promoted or in trouble
Make Chef:
  Chef has name
  Chef can:
          - be_in_trouble:
          Say "Chef [name] is in big trouble"
          - do_great:
```

```
          Say "Chef [name] is doing great!"

# Customer Stuff
Make Customer:
  Customer has name
  Customer can:
          - eat_here:
          Say "[name] is eating here"
          - pick_up:
          Say "[name] is picking up food"
          - get_delivery:
          Say "[name] wants food delivered"

# Example of how it works
# Set up the screen and customer
customer = Make Customer("Ivy")
fancy_screen = Make FancyScreen(customer)
say fancy_screen.show_menu()
say fancy_screen.show_favorites()
say fancy_screen.log_in("1234")

# Handle staff stuff
chef = Make Chef("Alice")
staff_handler = Make StaffHandler()
say staff_handler.get_feedback(chef, "bad")
say staff_handler.get_feedback(chef, "good")

# Customer doing things
say customer.eat_here()
say customer.pick_up()
say customer.get_delivery()
```

Enhanced Feedback and Dispute Resolution System for Service Personnel

System that allows the recording of feedback (complaints and compliments) for service personnel and customers, managed by a Manager who makes final decisions on complaints. Additionally, recipients can dispute complaints, enhancing accountability and resolution processes within the service environment.

```
# Feedback and Manager System

# Feedback Stuff
Make FeedbackBox:
  FeedbackBox keeps all the feedback
  FeedbackBox can:
          - get_feedback from sender to recipient about good or bad stuff:
          Keep the feedback
```

```
        Say "[sender] said [content] to [recipient]"
        - fix_problems with manager and the person who got complaints:
        Look at all the bad stuff said about the person
        For each bad thing:
        Manager decides what to do
        Say "Manager fixed the problem said by [who said it]"
        - argue_against_bad_thing by person who got the complaint:
        Say "[person] says the complaint is not true because [reason]"

# Manager
Make Manager:
  Manager can decide_what_to_do about the complaint:
        Decide something based on the complaint
        Say "Manager decided: [decision]"

# Example of how it works
# Make people and feedback box
customer = Make Customer("Ivy")
chef = Make Chef("Alice")
delivery_guy = Make DeliveryPerson("Charlie")
manager = Make Manager("Grace")
feedback_box = Make FeedbackBox()

# People saying good and bad things
say feedback_box.get_feedback(customer, chef, "good", "Yummy food!")
say feedback_box.get_feedback(customer, delivery_guy, "bad", "You were late!")
say feedback_box.get_feedback(delivery_guy, customer, "good", "Nice person!")

# Fixing problems with manager
say feedback_box.fix_problems(manager, delivery_guy)

# Arguing against a complaint
say feedback_box.argue_against_bad_thing(customer, "I was not late!")
```

Customer Management System with Warnings and Consequences
This outlines an expanded system for managing customer behavior in a service setting, where regular and VIP customers can receive warnings for infractions. Regular customers are deregistered after two warnings, while VIP customers are downgraded to regular status and their warnings reset.

```
# Customer Warnings System

# Regular Customer
Make Customer:
  Customer has id, name, warnings, and is registered
  Customer can:
          - get_a_warning:
          Add 1 to warnings
          If warnings are 2:
          Say "[name] is no longer registered because of too many warnings"
          Make is_registered as not registered
          - show_warnings:
          Say "[name] has [warnings] warnings"

# Special VIP Customer
Make VIPCustomer from Customer:
  VIPCustomer can:
          - get_a_warning:
          Use get_a_warning from Customer
          If warnings are 2:
          Say "[name] is now just a regular customer and warnings reset"
          Make warnings 0

# Example of how it works
# Make a customer and a VIP customer
customer = Make Customer(1, "Ivy")
vip_customer = Make VIPCustomer(2, "James")

# Customers get warnings
say customer.get_a_warning()
say customer.get_a_warning()  # Customer gets deregistered

say vip_customer.get_a_warning()
say vip_customer.get_a_warning()  # VIP gets downgraded

# Show how many warnings they have
say customer.show_warnings()
say vip_customer.show_warnings()
```

Customer Financial Management System
The customers can manage their financial transactions by adding funds to their accounts, placing orders within their budget limits, and closing their accounts with managerial assistance, ensuring a streamlined financial interaction within a service platform.

```
# Customer Money and Orders System

# Customer
Make Customer:
  Customer has id, name, money, and is registered
  Customer can:
        - add_money with amount:
        Add amount to money
        Say "[name] added $[amount]. Now has $[money]"
        - buy_stuff with cost:
        If money is enough for cost:
        Reduce money by cost
        Say "[name] bought stuff for $[cost]. Left with $[money]"
        Else:
        Say "[name] can't buy. Not enough money."
        - close_my_account:
        Make money 0
        Make is registered as not registered
        Say "[name]'s account closed. Money gone."

# Manager
Make Manager:
  Manager can:
        - close_customer_account for customer:
        Use close_my_account from Customer
        Say "Manager closed [customer]'s account."

# Example of how it works
# Make a customer and a manager
customer = Make Customer(1, "Ivy")
manager = Make Manager(1, "Grace")

# Customer doing things with money
say customer.add_money(100)
say customer.buy_stuff(120)  # Can't buy, not enough money
say customer.buy_stuff(80)   # Can buy, has enough money

# Manager closes customer account
say manager.close_customer_account(customer)
```

Chef and Dish Management System

Introduces a system where chefs can create dishes with detailed descriptions and manage ratings for each dish, allowing them to receive feedback and calculate average ratings, enhancing the culinary experience based on customer preferences.

```
# Chef and Dish Rating System

# Dish
Make Dish:
  Dish has id, name, chef, description, keywords, and ratings
  Dish can:
         - get_rated with stars:
         Add stars to ratings
         Say "[name] got [stars] stars"
         - show_average_stars:
         If there are ratings:
         Calculate average stars
         Say "Average stars for [name] is [average]"
         Else:
         Say "[name] has no ratings yet"

# Chef
Make Chef:
  Chef has id, name, and dishes he made
  Chef can:
         - make_new_dish with id, name, description, keywords:
         Make a Dish
         Add Dish to dishes
         Say "Chef [name] made a new dish: [name]"

# Example of how it works
# Make a chef
chef = Make Chef(1, "Alice")

# Chef makes dishes
say chef.make_new_dish(101, "Spaghetti Carbonara", "Creamy pasta with bacon and
cheese", ["pasta", "bacon", "cheese"])
say chef.make_new_dish(102, "Mango Cheesecake", "Sweet cheesecake with mango",
["cheesecake", "mango", "sweet"])

# Make dishes and rate them
carbonara = Make Dish(101, "Spaghetti Carbonara", chef, "Creamy pasta with bacon and
cheese", ["pasta", "bacon", "cheese"])
cheesecake = Make Dish(102, "Mango Cheesecake", chef, "Sweet cheesecake with
mango", ["cheesecake", "mango", "sweet"])

# Dishes getting rated
say carbonara.get_rated(5)
say carbonara.get_rated(4)
say carbonara.show_average_stars()

say cheesecake.get_rated(5)
say cheesecake.show_average_stars()
```

Chef-Importer Dispute Management System

sets up a system to manage and resolve complaints between chefs and food importers, utilizing a manager to judge complaints based on their nature and to implement appropriate disciplinary actions or rewards, thereby maintaining quality and accountability in the food supply chain.

```
# System for Chefs and Importers to Complain

# Feedback System
Make FeedbackBox:
  FeedbackBox knows the manager
  FeedbackBox can:
        - tell_manager_about_complaint from one person to another about problem:
        Manager decides what to do about it
        Say "Complaint from [one person] about [another person] because [problem]. What happened: [what manager decided]"

# Manager
Make Manager:
  Manager can:
        - decide_on_complaint about one person from another based on problem:
        If problem is "bad stuff":
        Fire the person who did bad stuff
        Give bonus to the one who complained
        Say "Bad person fired. Good person got a bonus."
        If problem is "not true":
        Tell complainer they are in trouble
        Say "Complainer in trouble for not telling the truth."

# Chef
Make Chef:
  Chef has id and name
  Chef can:
        - get_in_trouble:
        Say "Chef [name] is in trouble."
        - get_a_bonus:
        Say "Chef [name] got a bonus."

# Importer
Make Importer:
  Importer has id and name
  Importer can:
        - get_fired:
          Say "Importer [name] is fired."
          - get_in_trouble:
          Say "Importer [name] is in trouble."

# Example of how it works
# Make people and feedback box
chef = Make Chef(1, "Alice")
importer = Make Importer(1, "Bob")
manager = Make Manager()
feedback_box = Make FeedbackBox(manager)

# Chef complains about importer
say feedback_box.tell_manager_about_complaint(chef, importer, "bad stuff", "Bad
ingredients")

# Importer complains about chef
say feedback_box.tell_manager_about_complaint(importer, chef, "not true", "Chef lied")
```

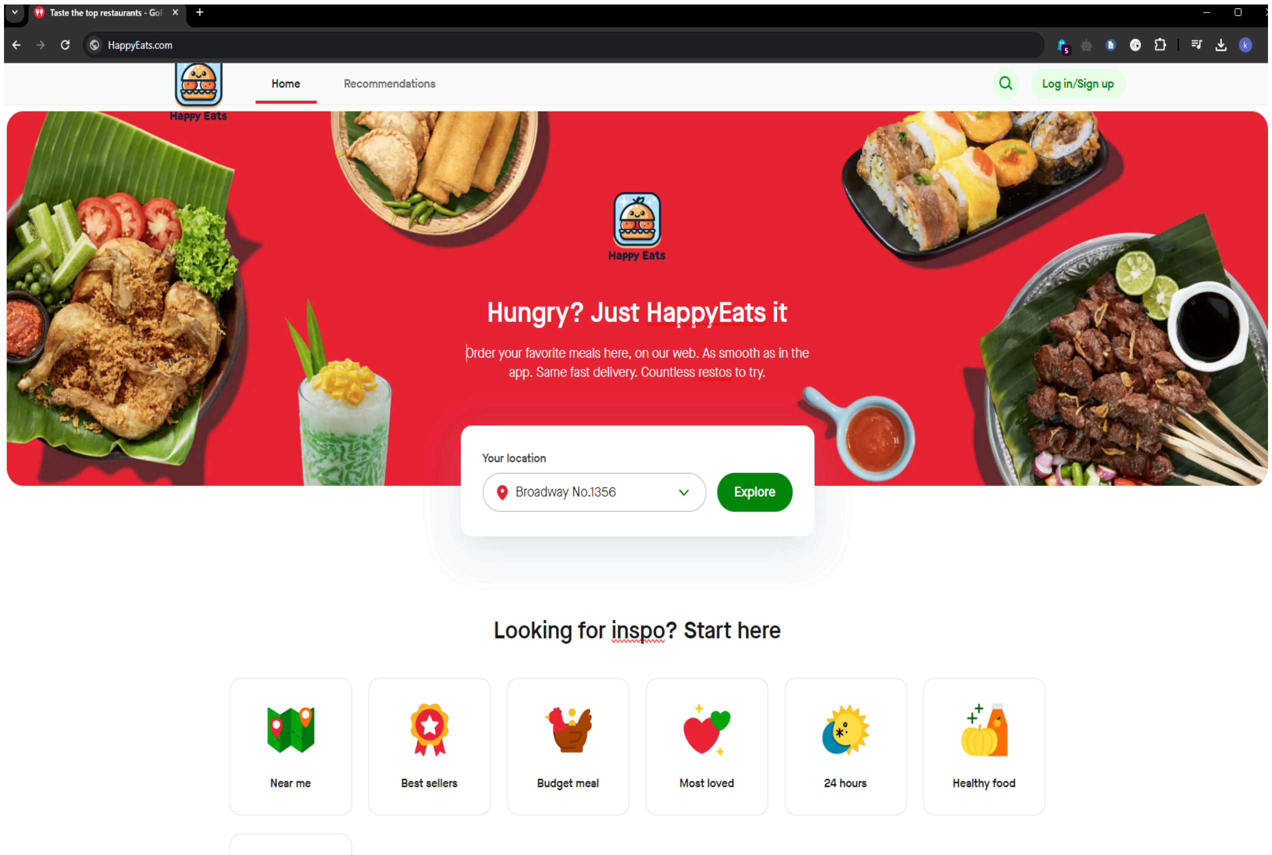
Added Bonus feature
Creative feature:
VIP customers can enjoy exclusive monthly rewards, including 5-10 food coupons for 50% off and a flat delivery rate of only $0.30, no matter the distance. Additionally, they have the option to tick a checkbox for a "Surprise Dish," allowing them to receive a random item from the menu as an extra side dish with their order.

```
          # VIP Customer Special Deals

          # VIP Customer
          Make VIPCustomer:
            VIPCustomer has coupons and can get cheap delivery
            VIPCustomer can:
                    - get_monthly_coupons:
                    Give 5 to 10 coupons for 50% off food
                    Say "You got [number] half-off coupons!"
                    - cheap_delivery:
                    Delivery is always 30 cents
                    Say "Your delivery is just 30 cents!"
                    - pick_a_surprise_dish from menu:
                    Choose a random dish from the menu
                    Say "Surprise! You get [dish name] as a side!"

          # Menu
          Make Menu:
            Menu has lots of dishes
```

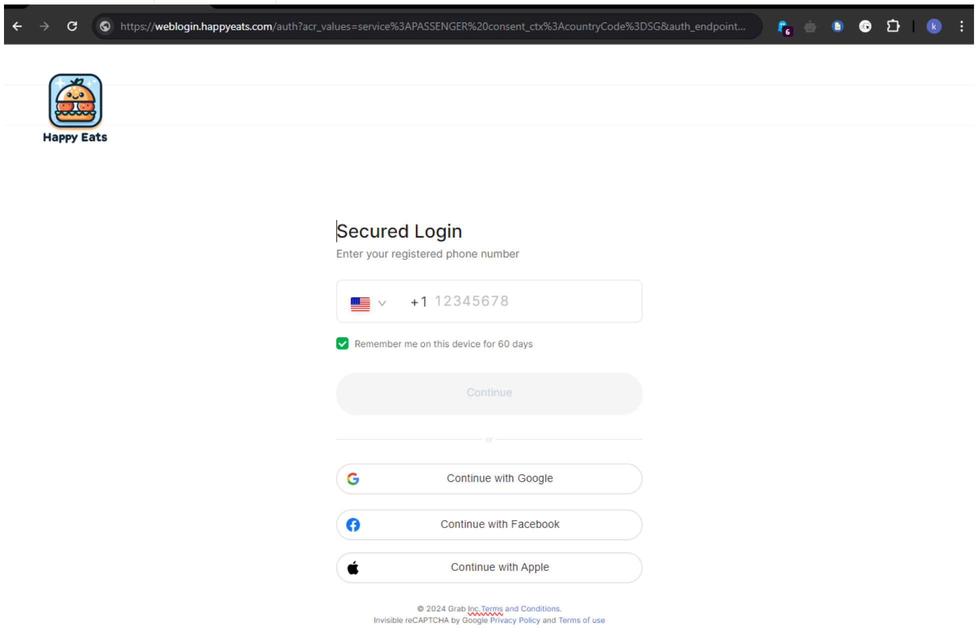
```
 Menu can:
     - choose_random_dish:
     Pick a dish at random
     Return the dish name

# Example of how it works
# Make a VIP customer and a menu
vip_customer = Make VIPCustomer()
menu = Make Menu()

# VIP gets special deals
say vip_customer.get_monthly_coupons()
say vip_customer.cheap_delivery()

# VIP picks a surprise dish
surprise_dish = menu.choose_random_dish()
say vip_customer.pick_a_surprise_dish(surprise_dish)
```

5. System Screens

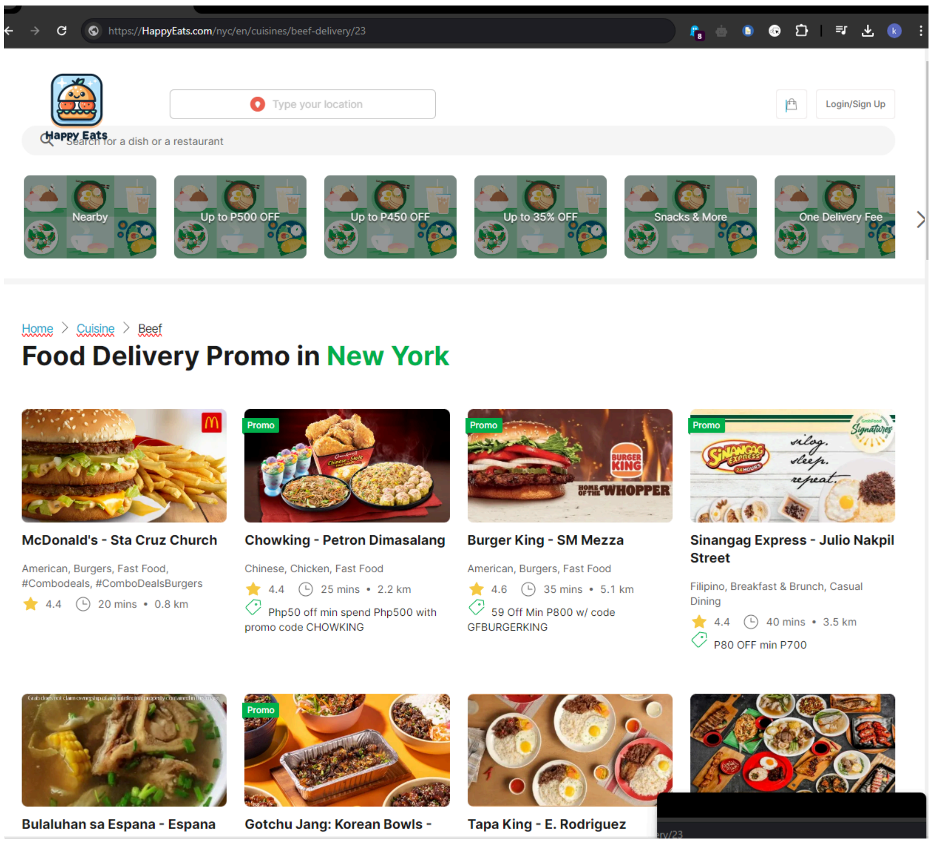
The following images present a detailed layout of the Graphical User Interface (GUI) for the Happy Eats Application, showcasing the user-friendly design and navigation paths users will take to order their meals.



Order your favorite meals here, on our web. As smooth as in the app. Same fast delivery. Countless restos to try.


https://weblogin.happyeats.com/auth?acr_values=service%3APASSENGER%20consent_ctx%3AcountryCode%3DSG&auth_endpoint...
Happy Eats
Secured Login
Enter your registered phone number
+1 12345678
Remember me on this device for 60 days
Continue
or
Continue with Google
Continue with Facebook
Continue with Apple

© 2024 Grab Inc.Terms and Conditions.
Invisible reCAPTCHA by Google Privacy Policy and Terms of use



https://HappyEats.com/nyc/en/cuisines/beef-delivery/23

Happy Eats

Type your location

Login/Sign Up

Search for a dish or a restaurant

Home > Cuisine > Beef

Food Delivery Promo in New York

McDonald's - Sta Cruz Church
American, Burgers, Fast Food, #Combodeals, #ComboDealsBurgers
4.4 · 20 mins • 0.8 km

Chowking - Petron Dimasalang
Chinese, Chicken, Fast Food
4.4 · 25 mins • 2.2 km
Php50 off min spend Php500 with promo code CHOWKING

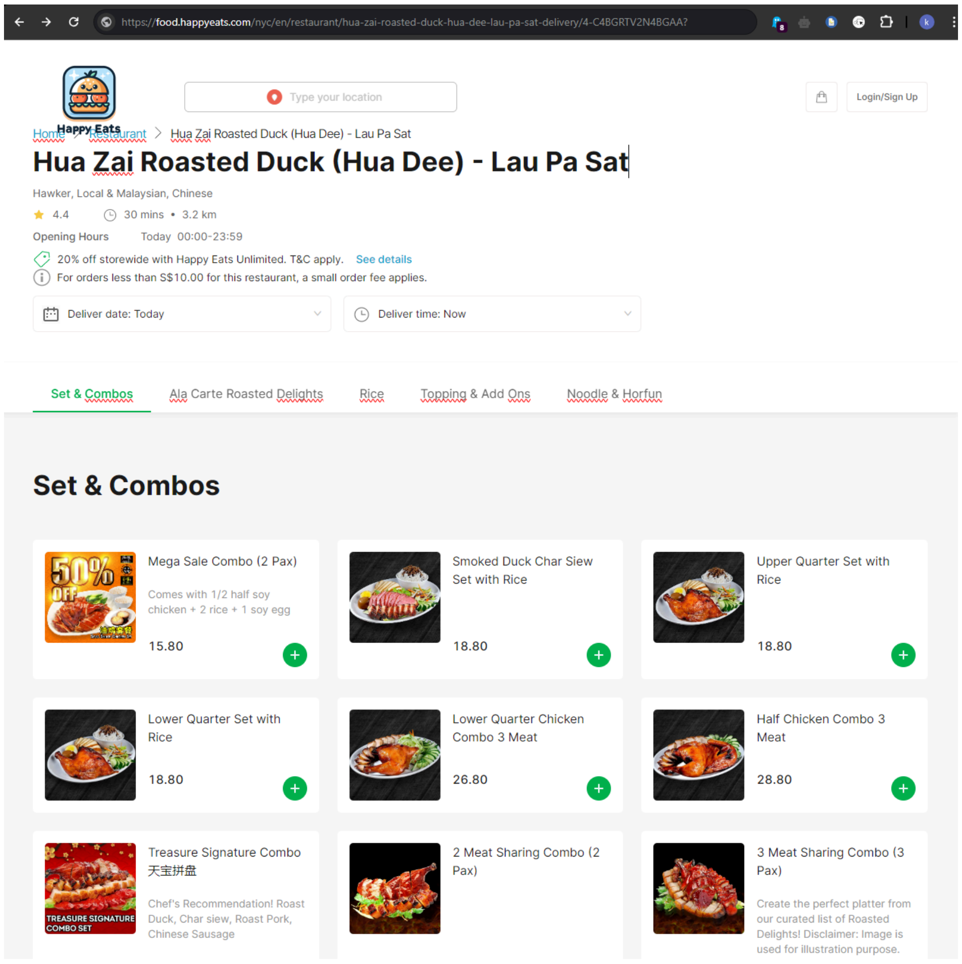
Burger King - SM Mezza
American, Burgers, Fast Food
4.6 · 35 mins • 5.1 km
59 Off Min P800 w/ code GFBURGERKING

Sinangag Express - Julio Nakpil Street
Filipino, Breakfast & Brunch, Casual Dining
4.4 · 40 mins • 3.5 km
P80 OFF min P700

Bulaluhan sa Espana - Espana

Gotchu Jang: Korean Bowls -

Tapa King - E. Rodriguez



https://food.happyeats.com/nyc/en/restaurant/hua-zai-roasted-duck-hua-dee-lau-pa-sat-delivery/4-C4BGRTV2N4BGAA?

Happy Eats

Type your location

Login/Sign Up

Home > Restaurant > Hua Zai Roasted Duck (Hua Dee) - Lau Pa Sat

Hua Zai Roasted Duck (Hua Dee) - Lau Pa Sat

Hawker, Local & Malaysian, Chinese
4.4 · 30 mins • 3.2 km
Opening Hours Today 00:00-23:59
20% off storewide with Happy Eats Unlimited. T&C apply. See details
For orders less than S$10.00 for this restaurant, a small order fee applies.

Deliver date: Today
Deliver time: Now

Set & Combos | Ala Carte Roasted Delights | Rice | Topping & Add Ons | Noodle & Horfun

Set & Combos

Mega Sale Combo (2 Pax)
Comes with 1/2 half soy chicken + 2 rice + 1 soy egg
15.80

Smoked Duck Char Siew Set with Rice
18.80

Upper Quarter Set with Rice
18.80

Lower Quarter Set with Rice
18.80

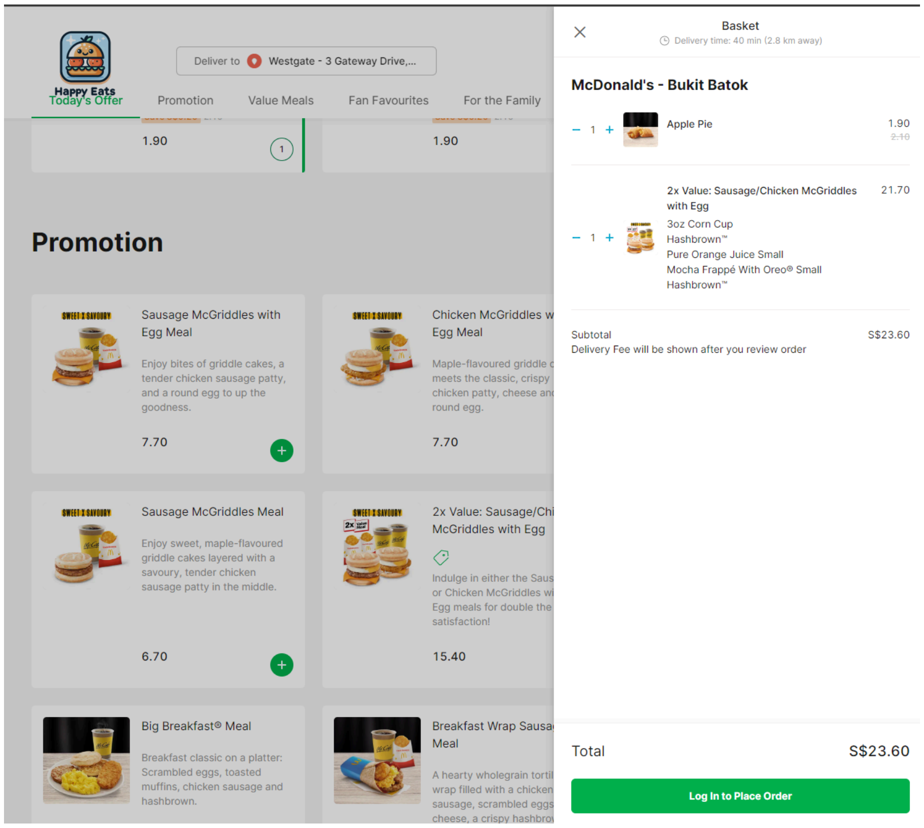
Lower Quarter Chicken Combo 3 Meat
26.80

Half Chicken Combo 3 Meat
28.80

Treasure Signature Combo 天宝拼盘
Chef's Recommendation! Roast Duck, Char siew, Roast Pork, Chinese Sausage

2 Meat Sharing Combo (2 Pax)

3 Meat Sharing Combo (3 Pax)
Create the perfect platter from our curated list of Roasted Delights! Disclaimer: Image is used for illustration purpose.



Happy Eats

Deliver to Westgate - 3 Gateway Drive,...

Today's Offer | Promotion | Value Meals | Fan Favourites | For the Family

1.90

1.90

Promotion

Sausage McGriddles with Egg Meal

Enjoy bites of griddle cakes, a tender chicken sausage patty, and a round egg to up the goodness.

7.70

Chicken McGriddles w Egg Meal

Maple-flavoured griddle c meets the classic, crispy chicken patty, cheese an round egg.

7.70

Sausage McGriddles Meal

Enjoy sweet, maple-flavoured griddle cakes layered with a savoury, tender chicken sausage patty in the middle.

6.70

2x Value: Sausage/Chi McGriddles with Egg

Indulge in either the Saus or Chicken McGriddles wi Egg meals for double the satisfaction!

15.40

Big Breakfast® Meal

Breakfast classic on a platter: Scrambled eggs, toasted muffins, chicken sausage and hashbrown.

Breakfast Wrap Sausa Meal

A hearty wholegrain torti wrap filled with a chicken sausage, scrambled eggs cheese, a crispy hashbro

Basket

Delivery time: 40 min (2.8 km away)

McDonald's - Bukit Batok

− 1 +	Apple Pie	1.90 ~~2.10~~
− 1 +	2x Value: Sausage/Chicken McGriddles with Egg 3oz Corn Cup Hashbrown™ Pure Orange Juice Small Mocha Frappé With Oreo® Small Hashbrown™	21.70

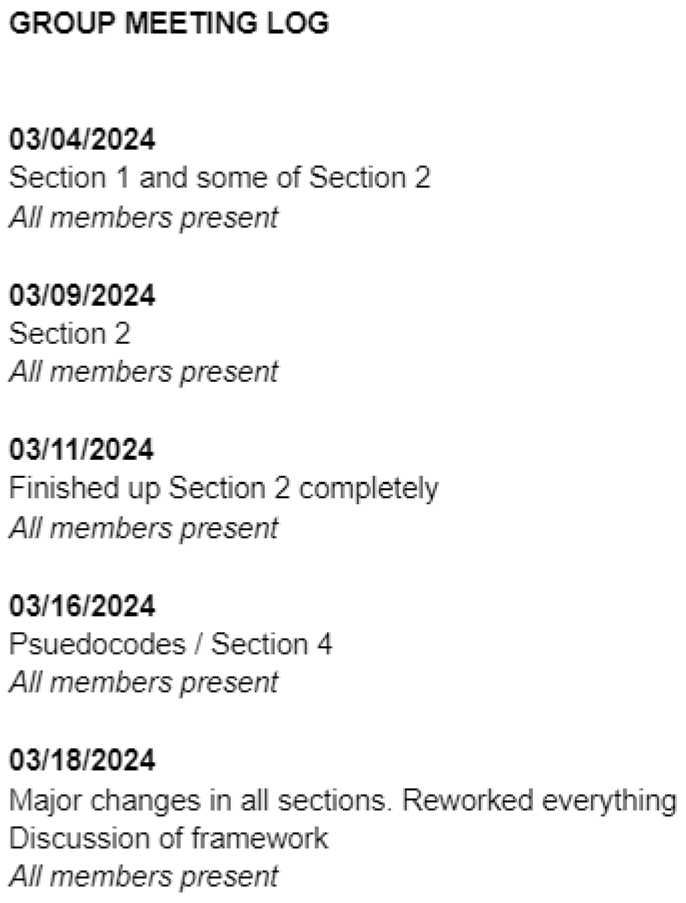
Subtotal S$23.60

Delivery Fee will be shown after you review order

Total S$23.60

Log In to Place Order

6. Group Meeting log



GROUP MEETING LOG

03/04/2024
Section 1 and some of Section 2
All members present

03/09/2024
Section 2
All members present

03/11/2024
Finished up Section 2 completely
All members present

03/16/2024
Psuedocodes / Section 4
All members present

03/18/2024
Major changes in all sections. Reworked everything
Discussion of framework
All members present

7. GitHub Repository

https://github.com/Kelvin205/happyeats-app