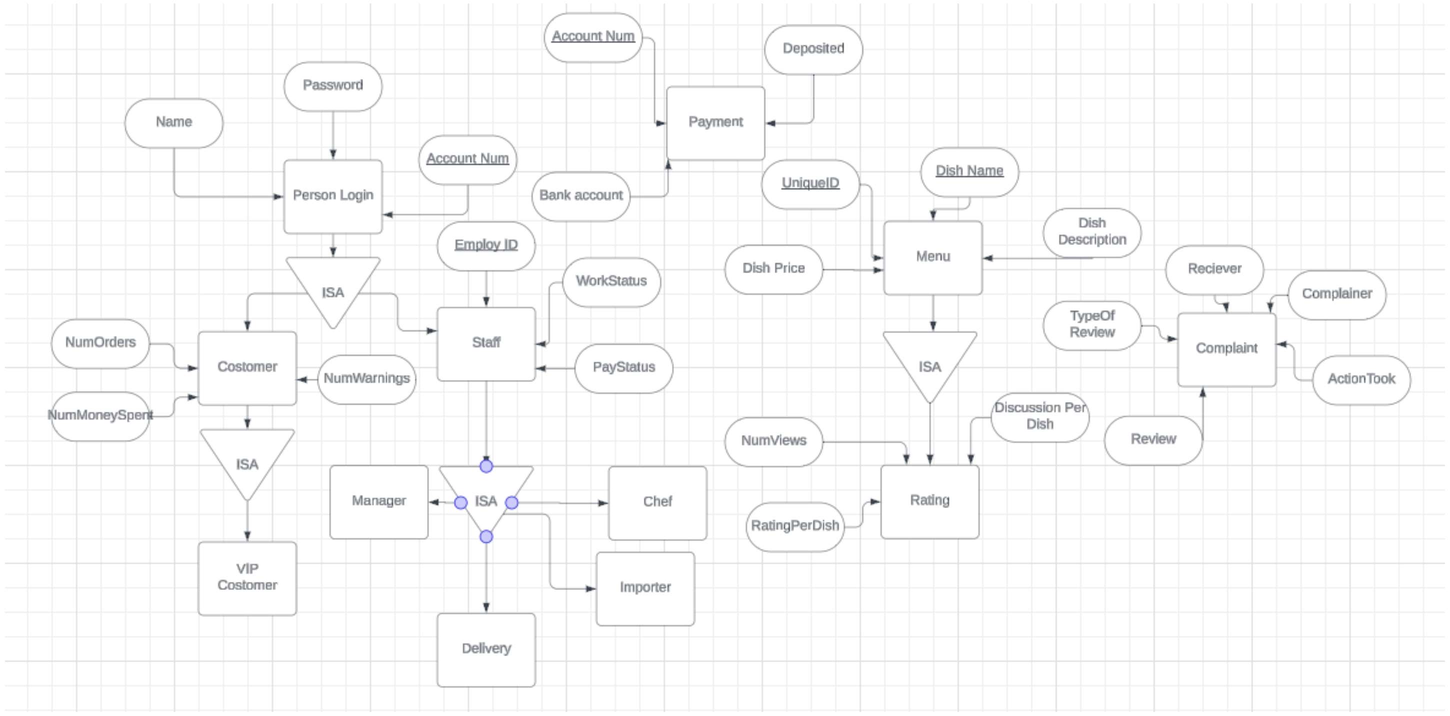
SAVORY SPRINTERS

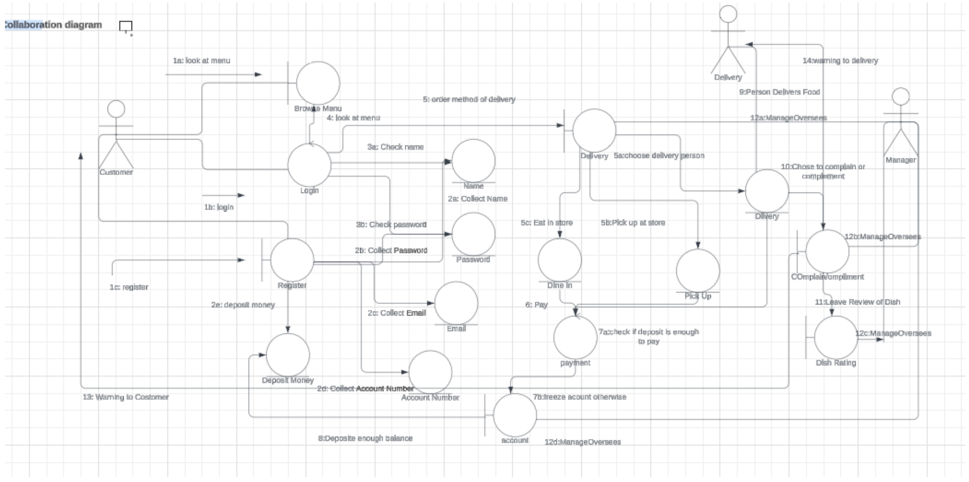
Phase II: Design Report

Group: Sadia H, Sibora B, Khadiza K, Jamiek T

ER Diagram:



Collaboration Diagram:



Detailed Function Design:

All use cases:

- Manage Menus:

Actors: Chef, Manager

Description: Chef creates, updates, or deletes menus. Inputs dish descriptions and keywords for browsing.

- Orders:

Actors: Registered Customer, VIP Customer, Surfer, Manager/Superuser, Delivery Person

Description: Surfers and Customers can browse menus, place orders, and choose delivery or pickup options or dine in.

- Deliveries:

Actors: Delivery Person, Manager/Superuser, Importers

Description: Delivery person delivers food orders, and records that delivery was made. Importers can also check off when delivery was made.

- Payments:

Actors: Manager/Superuser, Registered Customer, VIP Customer

Description: The system checks payment against deposited money, freezes orders if payment exceeds deposited money. Customers can also deposit money.

- Complaints/Compliments:

Actors: Manager/Superuser, Registered Customer, VIP Customer, Surfer, Chef, Delivery person, Importer

Description: Manager resolves complaints and compliments from various users, makes decisions on disputes, and manages warnings. Monitors warnings and manages customer statuses.

- System Login:

Actors: Manager/Superuser, Registered Customer, VIP Customer, Surfer, Chef, Delivery person, Importer

Description: Surfers can become registered Customer. All customers, Manager, and Staff can login. Manager clears deposits and closes accounts for departing customers.

- Monitor Performance:

Actors: Manager/Superuser

Description: Manager monitors chef and delivery person performance, promotes, demotes, or fires based on ratings and complaints.

- Bi-Monthly Competition:

Actors: Registered Customer, VIP Customer, Manager/Superuser

Description: Customers can choose to participate in the competition. The Manager will choose the Customer with the most orders in the month and give them a 25% discount on all orders for the following month.

- Food Reviews and Ratings:

Actors: Registered Customer, VIP Customer, Manager/Superuser

Description: Rating Aggregation: calculate and display average dish ratings based on customer feedback. Service Rating: allow customers to rate the delivery service separately from the food quality.

- **Scenarios**
    - Use Case: Manage Menus
        - Normal Scenario:

          Chef logs into the system, selects the menu management section, adds a new dish by entering details and saving it.
        - Exceptional Scenario:

          Chef attempts to save a new dish without mandatory fields filled, the system displays an error and requests complete information.
    - Use Case: Orders
        - Normal Scenario:

          Customer selects dishes, adds them to the cart, enters delivery details, and completes payment; the system confirms the order and updates the order status as it progresses.
        - Exceptional Scenario:

          Payment fails due to insufficient funds; the system alerts the customer and halts the order process until payment issues are resolved.
    - Use Case: Deliveries
        - Normal Scenario:

          Delivery person receives order details, picks up the order from the kitchen, and delivers it to the customer; the system updates the order status accordingly.
        - Exceptional Scenario:

          Delivery person cannot deliver due to an unexpected incident (e.g., vehicle breakdown); the system notifies the manager and the customer, and an alternative arrangement is made.
    - Use Case: Payments
        - Normal Scenario:

Registered Customers is asked to input card information: name on card, card number, expiration date, CVV, and zip code associated with card. A fixed amount of payment deposit options is offered: $25, $50, $75, $100. Request to bank is sent to pull money. Money is added to wallet. Request to the wallet is made for payment for orders.

    - Exceptional Scenario:

      Wrong card information; Not all fields were entered for card information; Bank kicked back money request; Request for payment of order exceeds money in wallet, order is frozen and request to add money to wallet is presented.

- Use Case: complaints/ compliments
    - Normal Scenario:

      Collect compliments and complaints from registered customers, VIP customers, delivery people, chief and importers. Checks if complaints are valid or fraudulent. Gives warning if complaint is valid. Customer can review about chief and delivery people. Delivery people can review customer. The chef can review importers. Importers can review the chef.

    - Exceptional Scenario:

      Review is deemed invalid.

- Use Case: System Login
    - Normal Scenario:

      Allows a screen to see what chiefs are on the app. At that phase the user is a "surfer" of the app. If the user wants to order on the app, then they can register to become a registered user. Login is separated from customer and workers. Registered customers and VIP customers login from the customer login. Managers, chef, delivery people and importer login from the Staff login. Managers can remove users from the ability to login.

    - Exceptional Scenario:

Email not found in user, so they are offered to create an account. Password used doesn't match the password associated with user, so they are offered to reset password or try again. If the account being closed by the manager is a customer, it refunds their fund in wallet.

- Use Case: Monitor Performance
  - Normal Scenario:

Manage warnings. Removes user if warnings are exceeded.

- Use Case: Bi-Monthly Competition
  - Normal Scenario:

Customer has option to be added to the customer pool. Manager chooses the customer that ordered the most.

  - Exceptional Scenario:

Customer tie.

- Use Case: Food Review and Ratings
  - Normal Scenario:

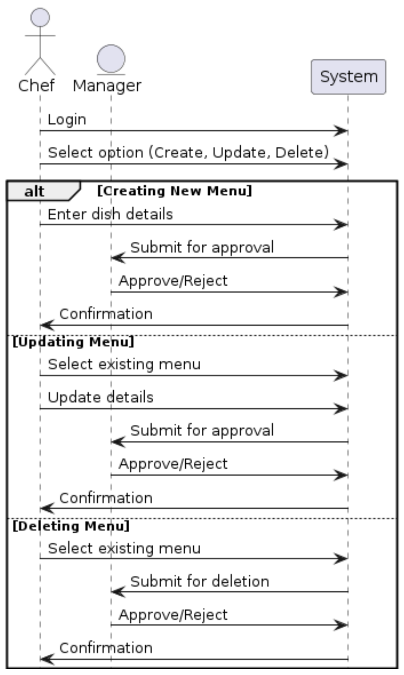
Adds food reviews and ratings

  - Exceptional Scenario:

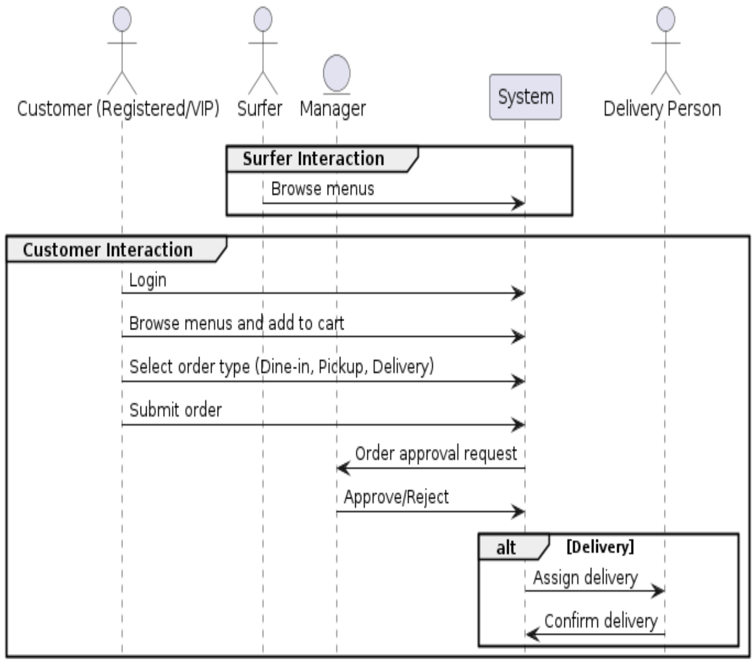
Bad reviews that need to delete

Sequence Diagrams

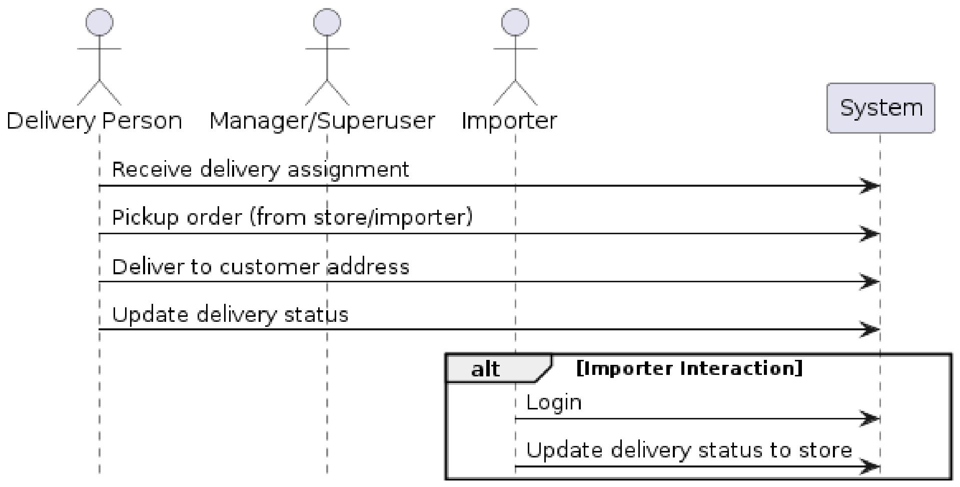
<u>Manage Menus</u>



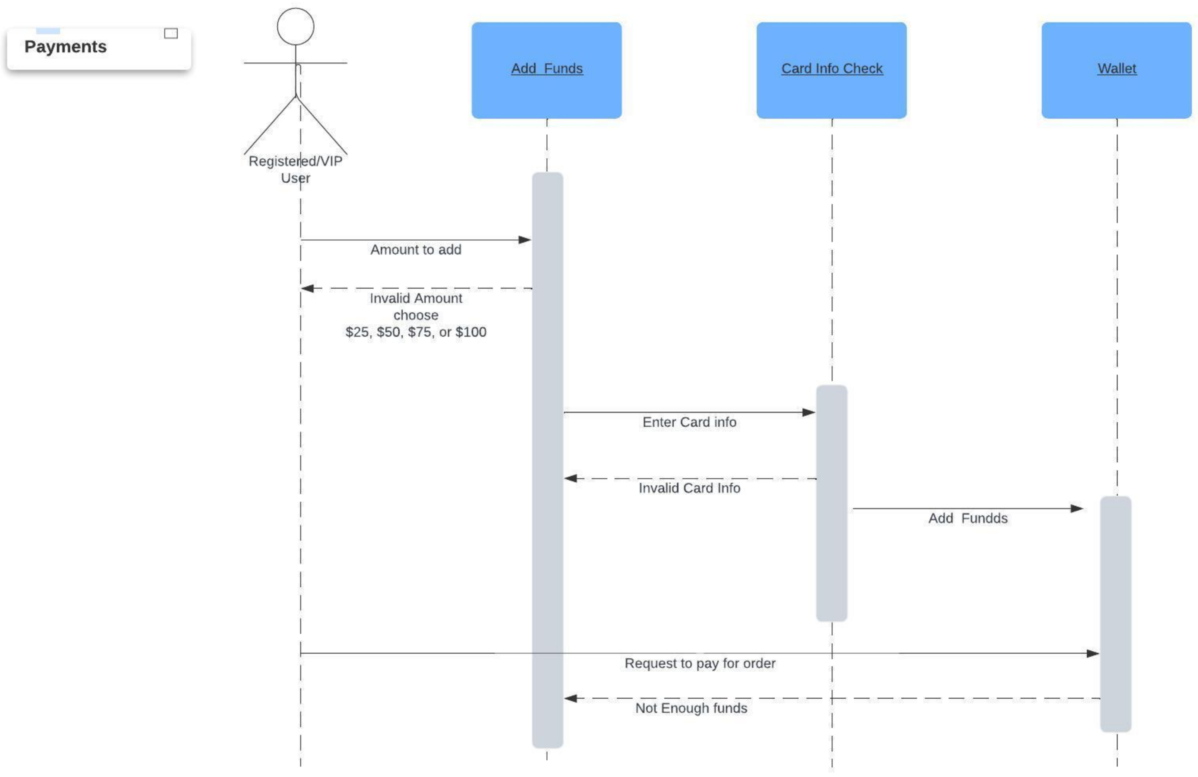
Orders



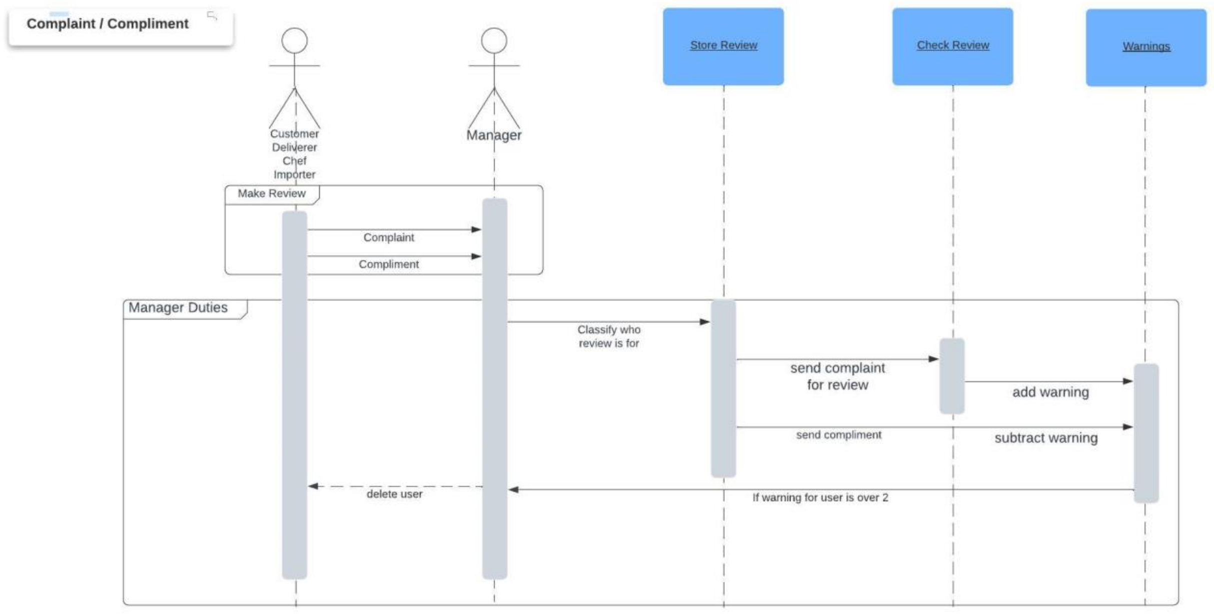
Deliveries


Delivery Person
Manager/Superuser
Importer
System
Receive delivery assignment
Pickup order (from store/importer)
Deliver to customer address
Update delivery status
alt
[Importer Interaction]
Login
Update delivery status to store

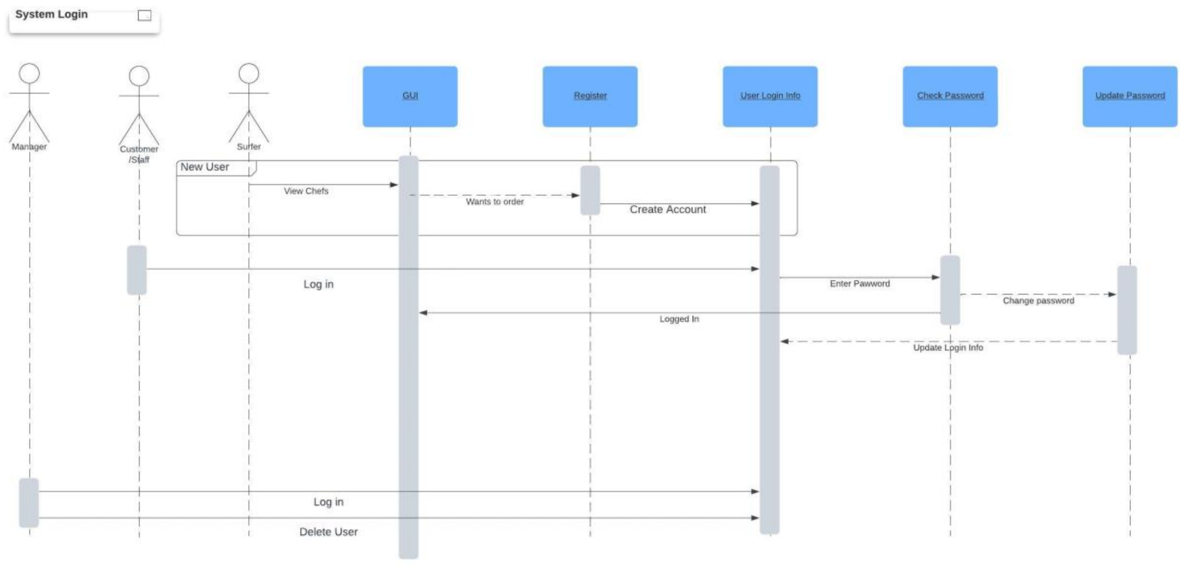
Payment



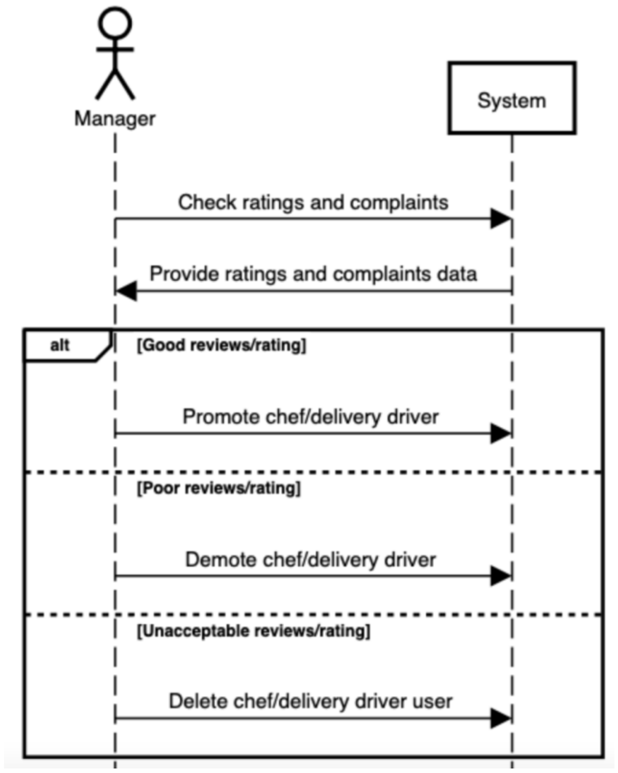
Complaints / Compliments



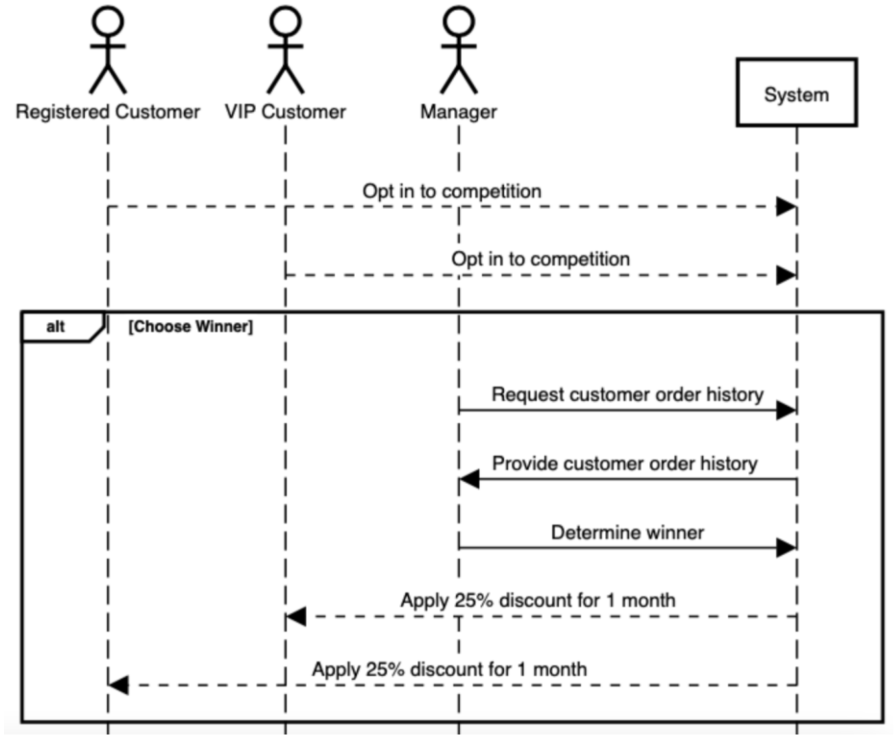
System Login



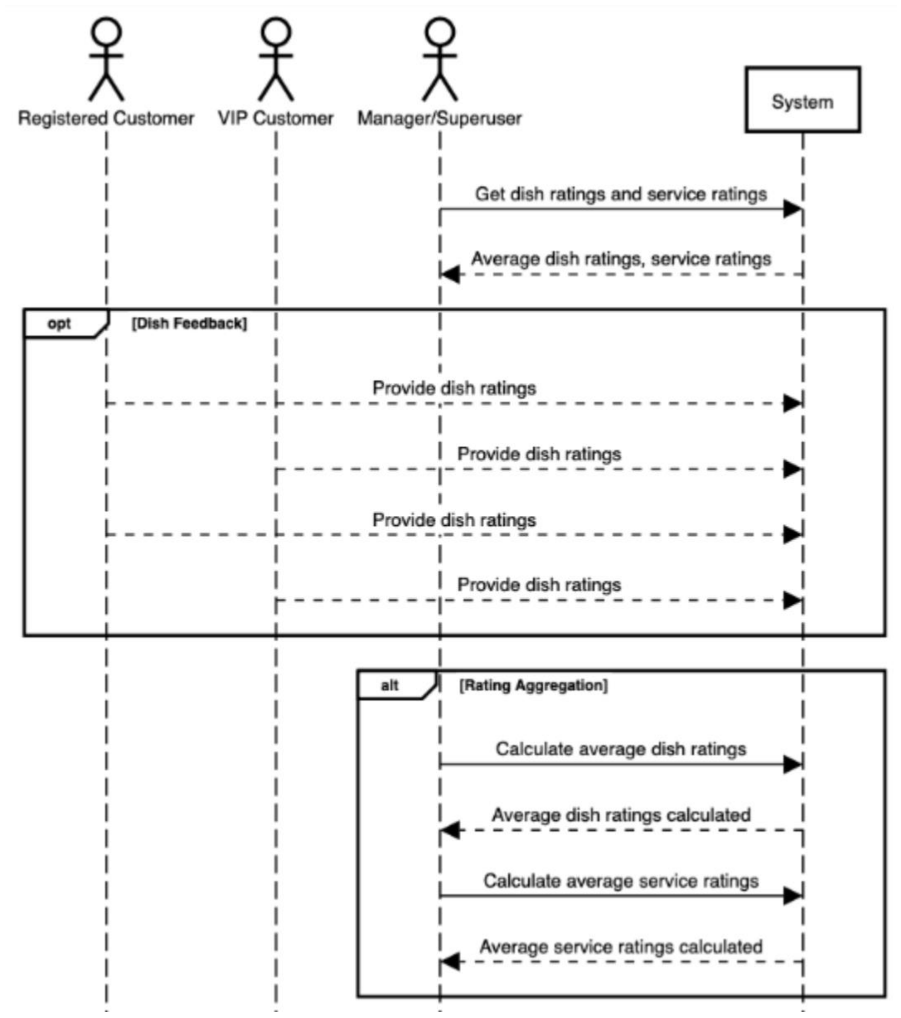
Monitor Performance


Manager
System
Check ratings and complaints
Provide ratings and complaints data
alt
[Good reviews/rating]
Promote chef/delivery driver
[Poor reviews/rating]
Demote chef/delivery driver
[Unacceptable reviews/rating]
Delete chef/delivery driver user

Bi-Monthly Competition

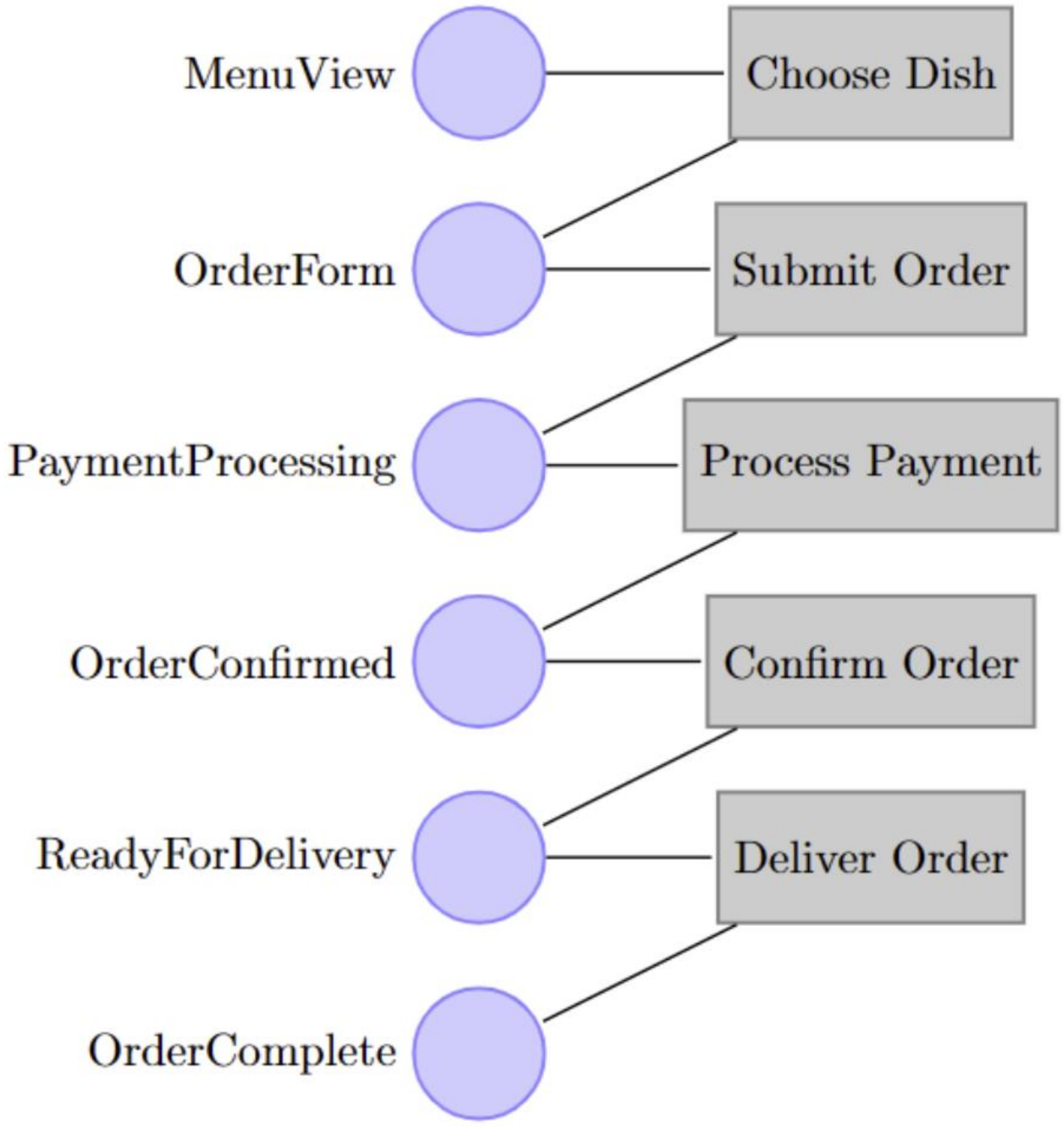

Registered Customer
VIP Customer
Manager
System
Opt in to competition
Opt in to competition
alt
[Choose Winner]
Request customer order history
Provide customer order history
Determine winner
Apply 25% discount for 1 month
Apply 25% discount for 1 month

Reviews and Ratings

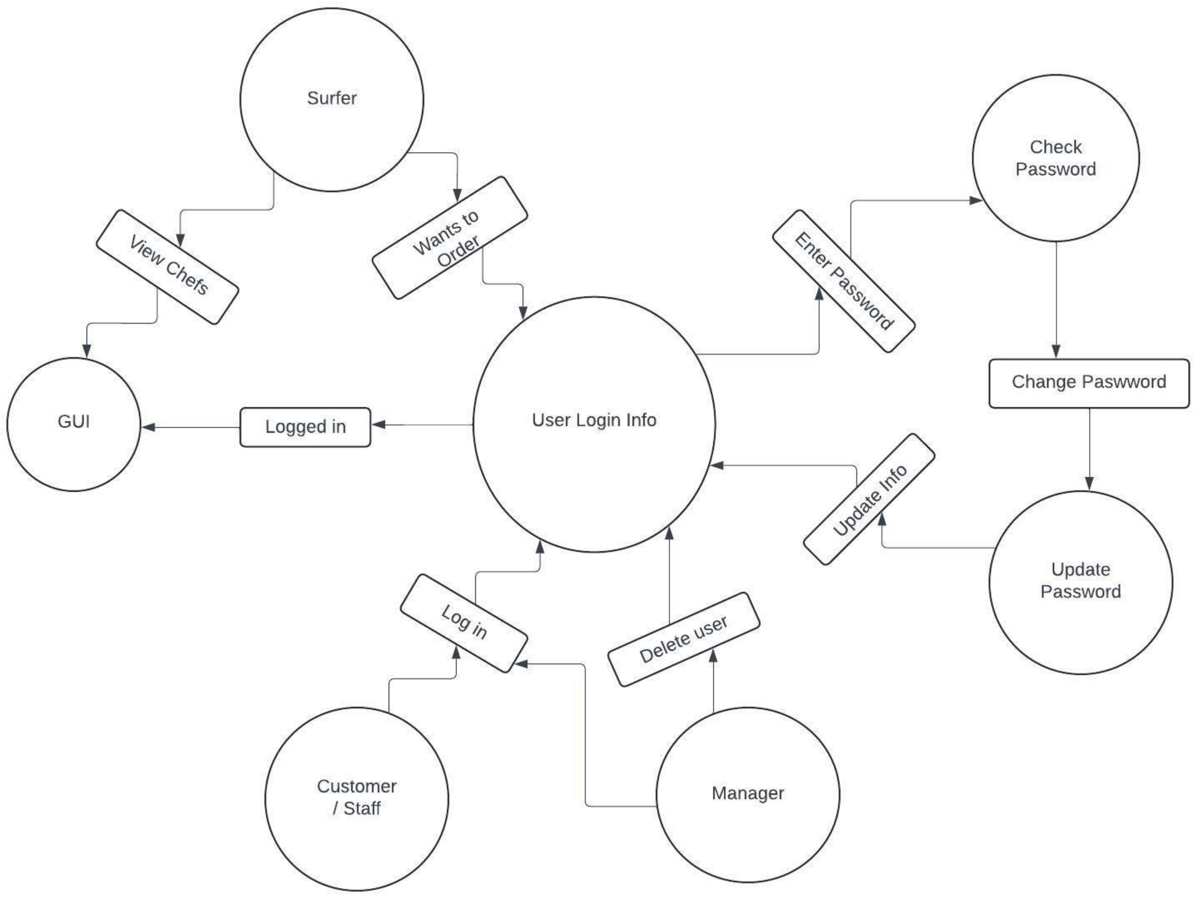

Registered Customer
VIP Customer
Manager/Superuser
System
Get dish ratings and service ratings
Average dish ratings, service ratings
opt
[Dish Feedback]
Provide dish ratings
Provide dish ratings
Provide dish ratings
Provide dish ratings
alt
[Rating Aggregation]
Calculate average dish ratings
Average dish ratings calculated
Calculate average service ratings
Average service ratings calculated

- **Petri-nets**

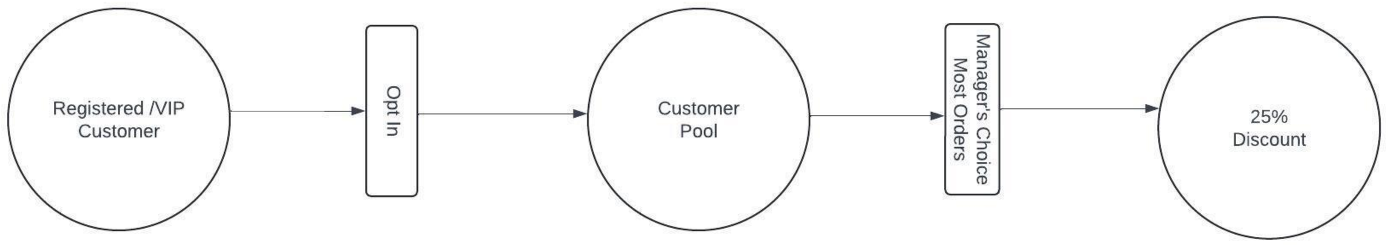
<u>Orders</u>


MenuView
Choose Dish
OrderForm
Submit Order
PaymentProcessing
Process Payment
OrderConfirmed
Confirm Order
ReadyForDelivery
Deliver Order
OrderComplete

System Loggin



Bi-Monthly Competition



- **Pseudocode**
  - Manage Menus

```
// Main Function: manageMenu
    FUNCTION manageMenu(menuAction, menuDetails)
```

```
            IF menuAction IS "Create"
              RETURN createMenu(menuDetails)
            ELSE IF menuAction IS "Update"
              RETURN updateMenu(menuDetails)
            ELSE IF menuAction IS "Delete"
              RETURN deleteMenu(menuDetails.menuId)
            ENDIF
          END FUNCTION

// Supporting Method: createMenu
          FUNCTION createMenu(menuDetails)
            INSERT menuDetails INTO MenuDatabase
            SEND menuDetails TO Manager FOR Approval
            WAIT FOR Manager's Decision
            IF Manager APPROVES
              RETURN True
            ELSE
              RETURN False
            ENDIF
          END FUNCTION

// Supporting Method: updateMenu
          FUNCTION updateMenu(menuDetails)
            UPDATE MenuDatabase SET dishDetails = menuDetails WHERE menuId =
          menuDetails.menuId
            SEND updated menuDetails TO Manager FOR Approval
            WAIT FOR Manager's Decision
            IF Manager APPROVES
              RETURN True
            ELSE
              RETURN False
            ENDIF
          END FUNCTION

// Supporting Method: deleteMenu
          FUNCTION deleteMenu(menuId)
            DELETE FROM MenuDatabase WHERE menuId = menuId
            SEND deletion request TO Manager FOR Approval
            WAIT FOR Manager's Decision
            IF Manager APPROVES
              RETURN True
            ELSE
              RETURN False
            ENDIF
          END FUNCTION
```

- Orders

```
// Main Function: processOrder
          FUNCTION processOrder(customerId, orderDetails)
          isValid = validateCustomerStatus(customerId)
          IF NOT isValid
            RETURN False
          ENDIF

          totalPrice = calculateTotalPrice(orderDetails)
          sufficientFunds = checkCustomerBalance(customerId, totalPrice)

          IF NOT sufficientFunds
            RETURN False
          ENDIF

          isPlaced = placeOrderInDatabase(orderDetails)
          IF NOT isPlaced
            RETURN False
          ENDIF

          isApproved = sendOrderForApproval(orderDetails)
          IF isApproved
            IF orderDetails.requiresDelivery
              ASSIGN DeliveryPerson
            ENDIF
            RETURN True
          ELSE
            RETURN False
          ENDIF
        END FUNCTION

// Supporting Method: validateCustomerStatus
        FUNCTION validateCustomerStatus(customerId)
          RETRIEVE customerStatus FROM CustomerDatabase WHERE customerId = customerId
          IF customerStatus IS "Active"
            RETURN True
          ELSE
            RETURN False
          ENDIF
        END FUNCTION

// Supporting Method: calculateTotalPrice
        FUNCTION calculateTotalPrice(orderDetails)
          SET totalPrice = 0
          FOR EACH item IN orderDetails.dishes
            SET dishPrice = RETRIEVE price FROM MenuDatabase WHERE dishId = item.dishId
            totalPrice += dishPrice * item.quantity
          END FOR
          RETURN totalPrice
        END FUNCTION

// Supporting Method: checkCustomerBalance
        FUNCTION checkCustomerBalance(customerId, totalPrice)
          SET customerBalance = RETRIEVE balance FROM CustomerAccount WHERE
        customerId = customerId
          IF customerBalance >= totalPrice
            RETURN True
          ELSE
            RETURN False
          ENDIF
        END FUNCTION

// Supporting Method: placeOrderInDatabase
        FUNCTION placeOrderInDatabase(orderDetails)
          INSERT orderDetails INTO OrderDatabase
          IF INSERT successful
            RETURN True
          ELSE
            RETURN False
          ENDIF
        END FUNCTION

// Supporting Method: sendOrderForApproval
        FUNCTION sendOrderForApproval(orderDetails)
          SEND orderDetails TO Manager FOR Approval
          WAIT FOR Manager's Decision
          IF Manager APPROVES
            RETURN True
          ELSE
            RETURN False
          ENDIF
        END FUNCTION

// Supporting Method: assignDeliveryPerson
        FUNCTION assignDeliveryPerson(orderDetails)
          ASSIGN DeliveryPerson BASED ON availability AND location
          NOTIFY DeliveryPerson WITH orderDetails
          RETURN DeliveryPerson's ID
        END FUNCTION
```

- Deliveries

```
// Main Function: executeDelivery
      FUNCTION executeDelivery(deliveryId, deliveryDetails)
        deliveryStatus = retrieveOrderDetails(deliveryId)
        IF deliveryStatus IS "Assigned"
          pickupResult = pickupOrder(deliveryDetails)
          IF NOT pickupResult
            RETURN False
          ENDIF

          deliveryResult = deliverOrderToCustomer(deliveryDetails)
          IF NOT deliveryResult
            RETURN False
          ENDIF

          updateResult = updateDeliveryStatus(deliveryId, "Delivered")
          RETURN updateResult
        ELSE
```

```
            RETURN False
          ENDIF
        END FUNCTION

// Supporting Method: retrieveOrderDetails
        FUNCTION retrieveOrderDetails(deliveryId)
          RETRIEVE status FROM DeliveryDatabase WHERE deliveryId = deliveryId
          RETURN status
        END FUNCTION

// Supporting Method: pickupOrder
        FUNCTION pickupOrder(deliveryDetails)
          PERFORM pickup operation WITH deliveryDetails
          IF pickup successful
            RETURN True
          ELSE
            RETURN False
          ENDIF
        END FUNCTION

// Supporting Method: deliverOrderToCustomer
        FUNCTION deliverOrderToCustomer(deliveryDetails)
          PERFORM delivery TO customerAddress FROM deliveryDetails
          IF delivery successful
            RETURN True
          ELSE
            RETURN False
          ENDIF
        END FUNCTION

// Supporting Method: updateDeliveryStatus
        FUNCTION updateDeliveryStatus(deliveryId, status)
          UPDATE DeliveryDatabase SET deliveryStatus = status WHERE deliveryId = deliveryId
          IF UPDATE successful
            RETURN True
          ELSE
            RETURN False
          ENDIF
        END FUNCTION
```

- <u>Payment</u>

```
class Wallet:
  balance = 0

  def add_funds(amount, card_number, expiry_date, cvv):
    try:
      validate_card_info(card_number, expiry_date, cvv)
```

```
            if amount in [25, 50, 75, 100]:
                balance += amount
                return "Funds added successfully."
            else:
                return "Invalid amount. Please choose from $25, $50, $75, or $100."
        except InvalidCardInfoException as e:
            return str(e)

    def validate_card_info(card_number, expiry_date, cvv):
        # Validate card information, such as card number, expiry date, and CVV
        if not card_number or not expiry_date or not cvv:
            raise InvalidCardInfoException("Please enter all required card information.")
        # Additional validation logic can be implemented here

    def make_payment(amount):
        if balance >= amount:
            balance -= amount
            return "Payment successful."
        else:
            return "Insufficient funds. Please add more funds to your wallet."

class InvalidCardInfoException(Exception):
    pass
```

- <u>Complaints / Compliments</u>

```
class UserManager:
    users = {}

    def add_user(user):
        # Add user to the user manager
        users[user.id] = user

    def remove_user(user_id):
        # Remove user from the user manager
        del users[user_id]

    def get_user(user_id):
```

```
    # Get user from the user manager
    return users.get(user_id)

class User:
  def __init__(self, id, role):
    self.id = id
    self.role = role
    self.warnings = 0

  def add_warning():
    # Add a warning to the user
    self.warnings += 1

  def remove_warning():
    # Remove a warning from the user
    if self.warnings > 0:
      self.warnings -= 1

class RegisteredCustomer(User):
  def make_complaint(complaint):
    # Make a complaint
    # Check if complaint is valid or fraudulent
    # Give warning if complaint is valid
    if complaint.valid:
      add_warning()
    else:
      complaint.accused.add_warning()

  def make_compliment(compliment):
    # Make a compliment
    # Subtract a warning or bad review
    if compliment.compliment_type == "warning":
      remove_warning()
    else:
      compliment.accused.remove_warning()

class VIPCustomer(RegisteredCustomer):
  pass
```

```
class DeliveryPerson(User):
  def make_complaint(complaint):
    # Make a complaint
    # Check if complaint is valid or fraudulent
    # Give warning if complaint is valid
    if complaint.valid:
      add_warning()
    else:
      complaint.accused.add_warning()

  def make_review(review):
    # Make a review
    # Check if review is valid or fraudulent
    # Give warning if review is valid
    if review.valid:
      add_warning()
    else:
      review.accused.add_warning()

class Chef(User):
  def make_complaint(complaint):
    # Make a complaint
    # Check if complaint is valid or fraudulent
    # Give warning if complaint is valid
    if complaint.valid:
      add_warning()
    else:
      complaint.accused.add_warning()

  def make_review(review):
    # Make a review
    # Check if review is valid or fraudulent
    # Give warning if review is valid
    if review.valid:
      add_warning()
    else:
      review.accused.add_warning()
```

```
class Importer(User):
  def make_complaint(complaint):
    # Make a complaint
    # Check if complaint is valid or fraudulent
    # Give warning if complaint is valid
    if complaint.valid:
      add_warning()
    else:
      complaint.accused.add_warning()

  def make_review(review):
    # Make a review
    # Check if review is valid or fraudulent
    # Give warning if review is valid
    if review.valid:
      add_warning()
    else:
      review.accused.add_warning()

class Complaint:
  def __init__(self, complainant, accused, valid):
    self.complainant = complainant
    self.accused = accused
    self.valid = valid

class Compliment:
  def __init__(self, complimenter, accused, compliment_type):
    self.complimenter = complimenter
    self.accused = accused
    self.compliment_type = compliment_type
```

- System login

```
class App:
  def start():
    show_chefs_screen()
```

```
def show_chefs_screen():
  # Display chefs available on the app
  pass

def customer_login(email, password):
  # Customer login process
  customer = UserManager.get_customer_by_email(email)
  if not customer:
    offer_to_create_account()
  elif customer.password != password:
    offer_password_reset()
  else:
    # Successful login
    customer_menu(customer)

def staff_login(email, password):
  # Staff login process
  staff = UserManager.get_staff_by_email(email)
  if not staff:
    offer_to_create_account()
  elif staff.password != password:
    offer_password_reset()
  else:
    # Successful login
    staff_menu(staff)

def offer_to_create_account():
  # Offer user to create an account
  pass

def offer_password_reset():
  # Offer user to reset password
  pass

def customer_menu(customer):
  # Display menu for registered and VIP customers
  pass
```

```
    def staff_menu(staff):
        # Display menu for managers, chefs, delivery people, and importers
        pass

class UserManager:
    customers = {}
    staff = {}

    def add_customer(customer):
        # Add customer to the user manager
        customers[customer.email] = customer

    def add_staff(staff_member):
        # Add staff member to the user manager
        staff[staff_member.email] = staff_member

    def get_customer_by_email(email):
        # Get customer by email
        return customers.get(email)

    def get_staff_by_email(email):
        # Get staff member by email
        return staff.get(email)

class User:
    def __init__(self, email, password):
        self.email = email
        self.password = password

class Customer(User):
    def __init__(self, email, password):
        super().__init__(email, password)
        self.wallet_balance = 0

class Staff(User):
    pass

class Manager(Staff):
    def remove_user(self, user):
        # Remove user from the ability to login
        pass

# Usage
app = App()
app.start()
```

- Monitor performance

```
while (true) {
    // Get performance data for chefs and delivery persons
    performanceData = getPerformanceData()

    for each employee in performanceData {
        if (employee.rating < minimumRatingThreshold) {
            // If employee's rating is below minimum threshold, take action
            if (employee.complaints >= complaintThreshold) {
                // If complaints exceed threshold, fire the employee
                fireEmployee(employee)
            } else {
                // If no complaints, demote the employee
                demoteEmployee(employee)
            }
        } else if (employee.rating > promotionRatingThreshold) {
            // If employee's rating is above promotion threshold, promote the employee
            promoteEmployee(employee)
        }
    }
}
```

  - Bi-monthly competition

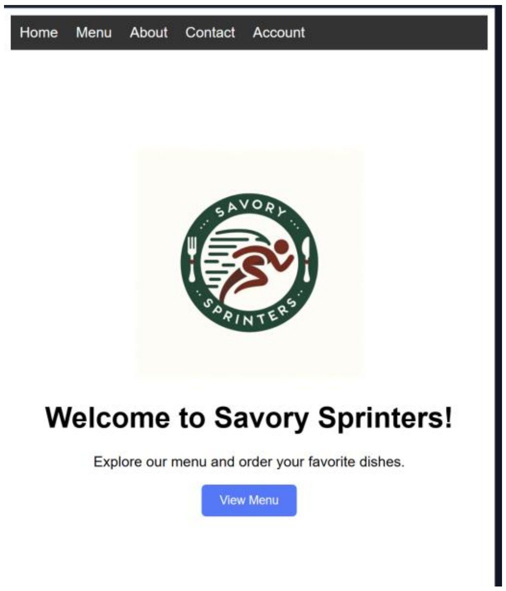
```
// Initialize variables
maxOrders = 0
winner = null

// Get list of participating customers
participants = getParticipatingCustomers()

for each customer in participants {
        // Check if customer has more orders than current maxOrders
        if (customer.orders > maxOrders) {
            maxOrders = customer.orders
            winner = customer
        }
    }

    if (winner != null) {
        // Give the winner a 25% discount on all orders for the following month
        giveDiscount(winner, 25)
    }
```

- Review/Rate

```
// Rating Aggregation: calculate and display average dish ratings based on customer feedback
function calculateAverageDishRating(feedbackList) {
    totalRating = 0
    for each feedback in feedbackList {
        totalRating += feedback.dishRating
    }
    averageRating = totalRating / feedbackList.length
    return averageRating
}

function displayAverageDishRating(averageRating) {
    display("Average dish rating: " + averageRating)
}

// Service Rating: allow customers to rate the delivery service separately from the food quality
function rateService(deliveryRating) {
    // Store the delivery service rating in the system
    storeDeliveryRating(deliveryRating)
}

function displayServiceRating(serviceRating) {
    display("Service rating: " + serviceRating)
}
```

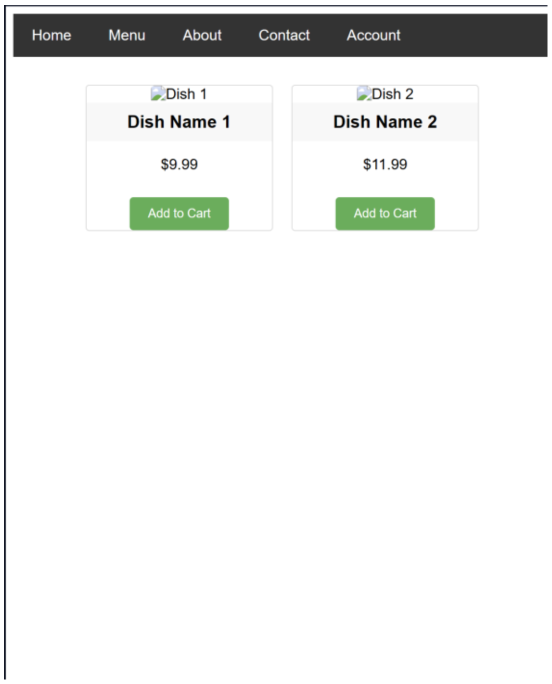
```
// Example usage:
feedbackList = getCustomerFeedback()
averageDishRating = calculateAverageDishRating(feedbackList)
displayAverageDishRating(averageDishRating)

deliveryRating = getCustomerDeliveryRating()
rateService(deliveryRating)
displayServiceRating(deliveryRating)
```

Major GUIs (prototype)



Welcome to Savory Sprinters!

Explore our menu and order your favorite dishes.

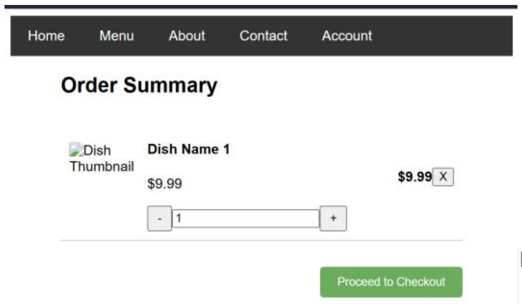


Dish Name 1

$9.99

Dish Name 2

$11.99



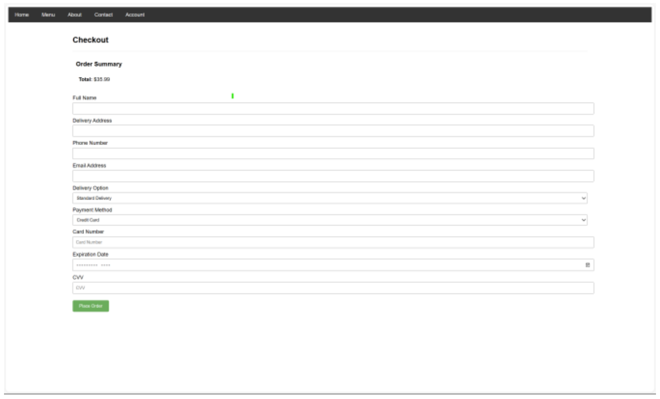
Order Summary

Dish Name 1

$9.99

1

$9.99



Checkout

Order Summary

Total: $35.99

Full Name

Delivery Address

Phone Number

Email Address

Delivery Option

Standard Delivery

Payment Method

Credit Card

Card Number

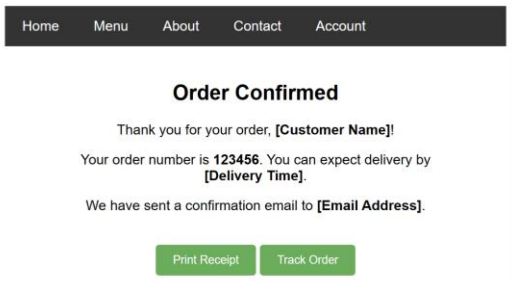
Card Number

Expiration Date

--------- ----

CVV

CVV

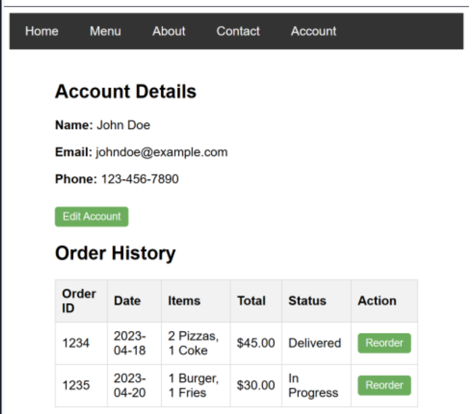


Order Confirmed

Thank you for your order, **[Customer Name]**!

Your order number is **123456**. You can expect delivery by **[Delivery Time]**.

We have sent a confirmation email to **[Email Address]**.



Account Details

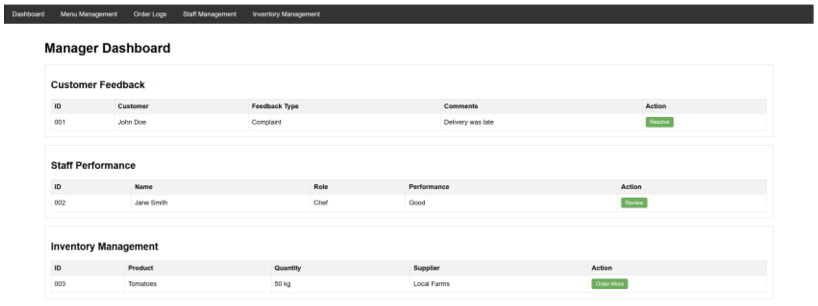
Name: John Doe

Email: johndoe@example.com

Phone: 123-456-7890

Order History

Order ID	Date	Items	Total	Status	Action
1234	2023-04-18	2 Pizzas, 1 Coke	$45.00	Delivered	Reorder
1235	2023-04-20	1 Burger, 1 Fries	$30.00	In Progress	Reorder



Manager Dashboard

Customer Feedback

ID	Customer	Feedback Type	Comments	Action
001	John Doe	Complaint	Delivery was late	Resolve

Staff Performance

ID	Name	Role	Performance	Action
002	Jane Smith	Chef	Good	Review

Inventory Management

ID	Product	Quantity	Supplier	Action
003	Tomatoes	50 kg	Local Farms	Order More

Meeting Memo:
3/20/24: Report 1

- Discussed tools and languages that will be used
  - Python, HTML, CSS, SQL, React, MongoDB
- All members present in person
- Delegated tasks
  - Jamiek: 1.1, 1.2, 1.3
  - Khadiza: Section 1.1, 2.2, 1.3, Use case diagram
  - Sadia: 3.1, 3.2
  - Sibora: 4, Edited 1.1, 1.2, 1.3,

4/16/24: Report 2

- Jamiek and Khadiza present in person, Sadia and Sibora filled in online
- Delegated tasks for report 2 and further discussed use cases and how to correct UML diagrams
  - Jamiek: Petri net diagrams, sequence diagrams, scenarios for payments, complaints, system login

    - Khadiza: Use Cases, Collaboration diagram, ER diagram
    - Sadia: diagrams, scenarios, and pseudocode for Manage menus, Orders, and Deliveries, major GUI screens
    - Sibora: Sequence diagrams, pseudocode for Bi-Monthly Competition, Review/Rate food and service, and Monitor Performance

Github Repo link:
https://github.com/Miek00/CSC322