Online Restaurant Order and Delivery System

Design Report

Version 1.0

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 16/Nov/20 | 1.0 | Document Initialization | Leji Li, Weiye Kuang, Xian Chen, Bingjing Dong |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

1. Introduction 5

2. All Use Cases 6

3. E-R diagram for the entire system 16

4. Detailed Design 16

4.1 Login 16

4.1.1 Customer Login 17

4.1.2 Staff Login 17

4.2 View Menu 17

4.3 View Special Menu 18

4.4 Search Menu 18

4.5 Voting 18

4.6 Rating Change 19

4.7 Staff Promote Or Demote 19

4.8 Customer Files Complaints or Compliments 19

4.9 Process Customer Complaints or Compliments 20

4.10 Manager Finalizes the Complaint 20

4.11 Process Finalized Complaint 20

4.12 Customer Disputes a Complaint 21

4.13 Customer Place an Order 21

4.14 Charge Payment 22

4.15 Freeze Order 22

4.16 Fund User Balance 22

4.17 Re-open Froze Order 22

4.18 Upgrade Customer 23

4.19 Discussion Thread 23

4.19.1 Create New Discussion 23

4.19.2 Participate in Discussions 23

4.19.3 Discussion Handler 24

4.20 Handle Customer Warning 24

4.21 Downgrade a Customer 25

4.22 Customer Registration Request 25

4.23 Check Registration Code 25

4.24 Create Customer Account 26

4.25 Kitchen Preparing the Order 26

4.26 Order Ready to Delivery 26

4.27 Edit Menu 26

4.28 Edit Keyword 27

4.29 Order Delivery 27

4.30 Order Delivered 27

4.31 Human Resources 27

4.31.1 Hire 27

4.31.2 Fire 27

4.32 Edit Taboo Words 28

4.33 Handle New Registration Request 28

5. System Screens 29

5.1 Login page 29

5.2 Customer screen 30

5.3 Dishes screen 30

5.4 Manager screen 32

5.5 Chef screen 32

5.6 Delivery people screen 34

6. Group Meetings 34

7. Git Repo 35

# Introduction

A picture containing chart

Description automatically generatedThis section provides the over-all collaboration class diagram of the system.

# All Use Cases

The following section represents the scenario and the collaboration diagram, or petri-net of each use case.

Table 2.1.1 presents the scenario of the Customer Registration use case.

|  |  |
| --- | --- |
| **Use Case** | Customer Registration |
| **Primary Actor** | Surfer |
| **Goal In Context** | surfer can apply to be a registered customer |
| **Preconditions** | Not a registered customer |
| **Trigger** | None |
| **Scenario** | 1)The surfer clicks the button “Register”  2)The surfer deposit fixed amount of money  3)The system sent an application to the manager |
| **Exceptions** | None |

**Table 2.1.1**

Class diagram:

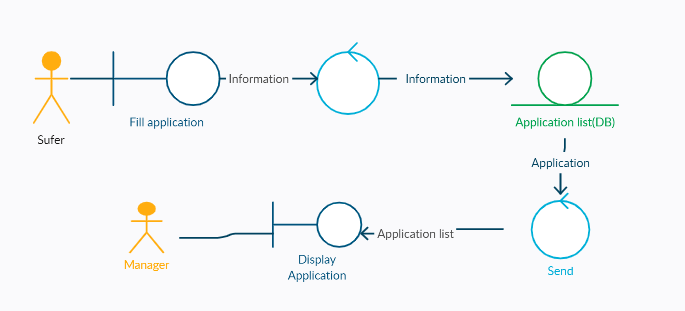


Table 2.1.2 presents the scenario of the Login use case.

|  |  |
| --- | --- |
| **Use Case** | Login |
| **Primary Actor** | All actors |
| **Goal In Context** | Enable the actor access to their subsystems. |
| **Preconditions** | All actors and they are not already logged in |
| **Trigger** | Try to use the system |
| **Scenario** | 1)The actor clicks the “Login” button  2)The actor enters the username and password  3)The actor clicks “Login”  4)The system checks the username and password |
| **Exceptions** | Invalid username or password |

**Table 2.1.2**

Class diagram:

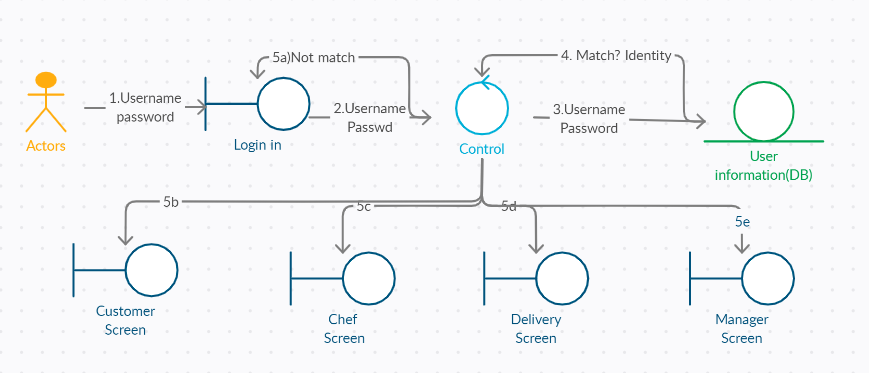


Table 2.1.3 presents the scenario of the Edit Menu use case.

|  |  |
| --- | --- |
| **Use Case** | Edit Menu |
| **Primary Actor** | Chef |
| **Goal In Context** | Modify the menu, add a dish, delete a dish or change the price |
| **Preconditions** | Chef, already logged in |
| **Trigger** | Chef want to change the menu |
| **Scenario** | Add a dish:  1)The chef clicks the button “Add”  2)The chef enters the name of the dishes and description  3)The chef enters the price  4)The chef uploads the image  5)The chef clicks “Submit”  6)The system checks whether the name already existed  7)The system checks the price valid or invalid  8)The system adds the dishes to the menu  Delete a dish:  1)The chef clicks the dishes  2)The chef selects “delete”  3)The chef clicks “Confirm”  4)The system deletes the dishes from the menu  Change price:  1)The chef clicks the dishes  2)The chef selects “Change price”  3)The chef enters the new price  4)Click “Submit” |
| **Exceptions** | Add an dish:  1. Name of the dishes already existed  2. Invalid price  Change price:  1.Invalid price |

**Table 2.1.3**

Petri-net:

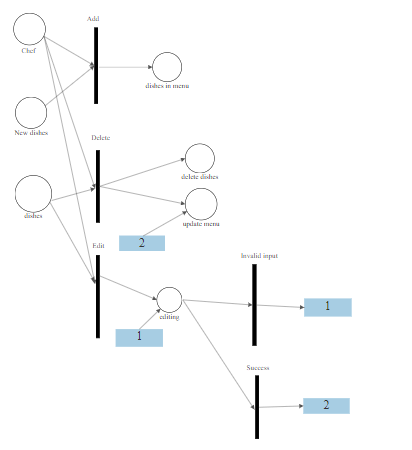


Table 2.1.4 presents the scenario of the Order use case.

|  |  |
| --- | --- |
| **Use Case** | Order |
| **Primary Actor** | Customer, VIP |
| **Goal In Context** | Customers add dishes to cart, delete dishes from cart and check out |
| **Preconditions** | Logged in customer |
| **Trigger** | Customers want to place an order |
| **Scenario** | 1)The customer clicks the dishes he wants  2)The customer selects the quantity  3)The customer adds the dishes to the cart  4)The customer clicks “cart” go to the cart  5)The system shows the dishes in the cart  6)The customer can click “delete” to delete the dishes from the cart  7)The customer can choose eat-in, pick-up, delivery  8)The customer clicks “check out” to pay the bill  9)If VIP, the system will give 10% discount |
| **Exceptions** | Money in the account isn’t enough |

**Table 2.1.4**

Petri-net:

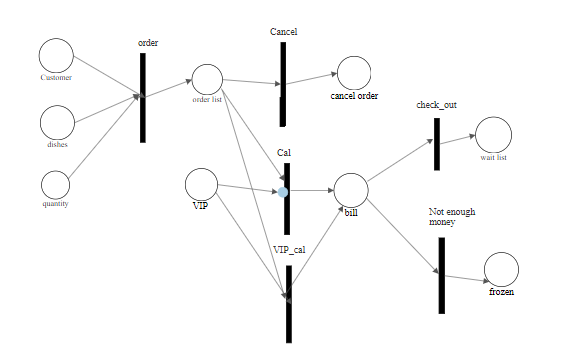


Table 2.1.5 presents the scenario of the hire use case

|  |  |
| --- | --- |
| **Use Case** | hire |
| **Primary Actor** | Manager |
| **Goal In Context** | Managers hire chef or delivery people |
| **Preconditions** | None |
| **Trigger** | None |
| **Scenario** | 1)Manager create a new account for the chef or the delivery people  2)Manager select the identity for the account, chef or delivery people  3)Manager enter the username and password  4)The system checks the username |
| **Exceptions** | Username already existed |

**Table 2.1.5**

Class diagram:

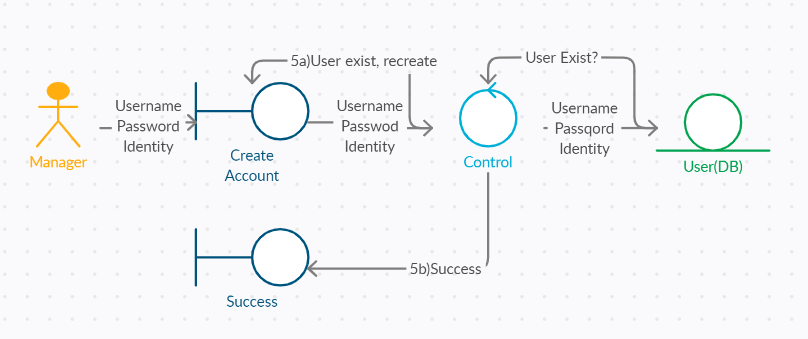


Table 2.1.6 presents the scenario of the Cut Pay use case

|  |  |
| --- | --- |
| **Use Case** | Cut Pay |
| **Primary Actor** | Manager |
| **Goal In Context** | Manager cuts the salary of a chef or a delivery person |
| **Preconditions** | None |
| **Trigger** | A chef or a delivery person is demoted |
| **Scenario** | 1)Manager selects the chef or the delivery people who he wants to cut pay for  2)Manager enters the amount of money he wants to cut |
| **Exceptions** | Invalid input |

**Table 2.1.6**

Class diagram:

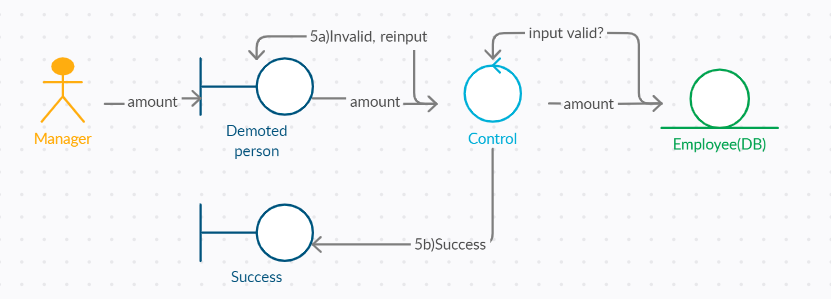


Table 2.1.7 presents the scenario of the Raise Pay use case

|  |  |
| --- | --- |
| **Use Case** | Raise Pay |
| **Primary Actor** | Manager |
| **Goal In Context** | Managers raise the salary of a chef or a delivery person |
| **Preconditions** | None |
| **Trigger** | A chef or a delivery person is promoted |
| **Scenario** | 1)Manager receives a noticement that a chef or a delivery person is promoted  2)Manager selects the chef or the delivery people who he wants to raise pay for  3)Manager enters the amount of money he wants to raise |
| **Exceptions** | Invalid input |

**Table 2.1.7**

Class diagram:

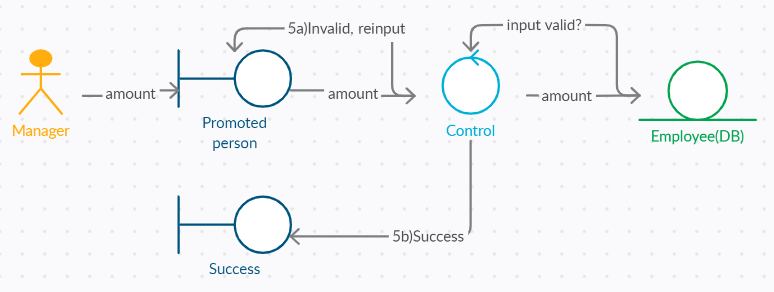


Table 2.1.8 presents the scenario of the Fire use case

|  |  |
| --- | --- |
| **Use Case** | Fire |
| **Primary Actor** | Manager |
| **Goal In Context** | Manager fires a chef or a delivery person |
| **Preconditions** | None |
| **Trigger** | A chef or a delivery person is demoted twice. |
| **Scenario** | 1)Manager received the noticement that a chef or a delivery person is demoted twice  2)Manager decides to fire the chef or the delivery person  3)The system deletes the account of the chef or the delivery people |
| **Exceptions** | None |

**Table 2.1.8**

Class diagram:

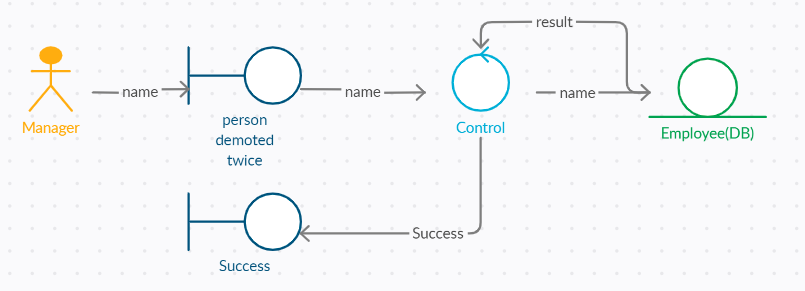


Table 2.1.9 presents the scenario of the Vote use case

|  |  |
| --- | --- |
| **Use Case** | Vote |
| **Primary Actor** | Customer, VIP |
| **Goal In Context** | Customers grade the food. |
| **Preconditions** | The customer purchased food before |
| **Trigger** | The customer receives the food delivered |
| **Scenario** | 1)The voting window display  2)The customer vote |
| **Exceptions** | None |

**Table 2.1.9**

Class diagram:

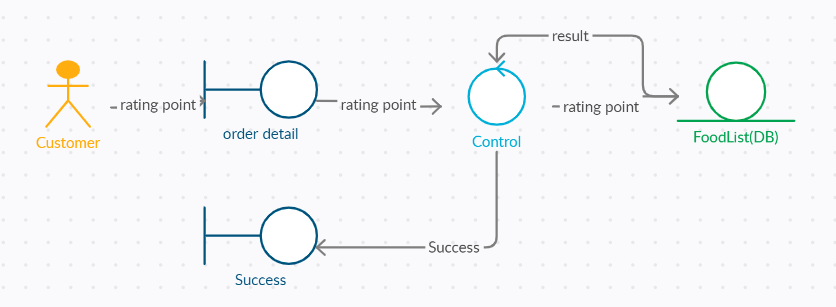


Table 2.1.10 presents the scenario of the Vote use case

|  |  |
| --- | --- |
| **Use Case** | Search |
| **Primary Actor** | Customer |
| **Goal In Context** | The customer can find a dish quickly |
| **Preconditions** | Logged in customer |
| **Trigger** | The customer wants to find a dish |
| **Scenario** | 1)The customer enters the name of dishes  2)The system displays the searching results |
| **Exceptions** | The dishes don’t exist |

**Table 2.1.10**

Class diagram:

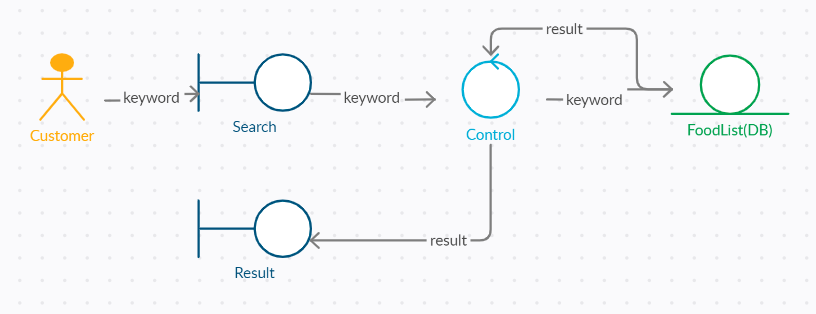


Table 2.1.11 presents the scenario of the Discussion use case

|  |  |
| --- | --- |
| **Use Case** | Discussion |
| **Primary Actor** | Customers, VIPs |
| **Goal In Context** | Customers can share their comments and discuss with other customers |
| **Preconditions** | Logged in customers |
| **Trigger** | None |
| **Scenario** | 1)The customer clicks the “Discussion” button  2)The system goes to the discussion window and lists the hottest discussion topic.  3)The customer can click the topic to view the comments from other comments and join in the discussion  3)The customer can click “Create” to start a new discussion |
| **Exceptions** | Taboo words are replaced by \*\*\*\*  Comments with more than 3 taboo words would be blocked |

**Table 2.1.11**

Petri-net:

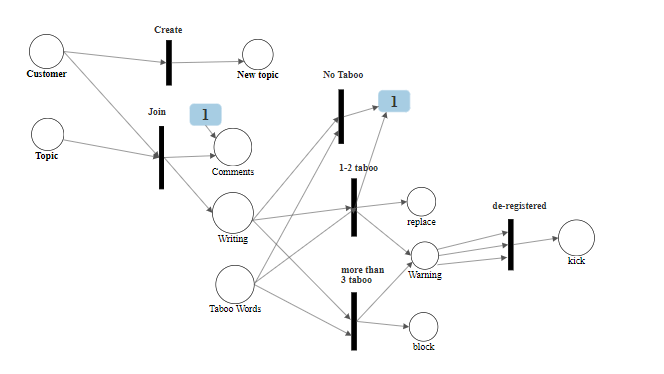


Table 2.1.12 presents the scenario of the Fire Compliments/Complaints use case

|  |  |
| --- | --- |
| **Use Case** | Fire Compliments/Complaints |
| **Primary Actor** | Customer, VIP, deliver people |
| **Goal In Context** | Customers can file complaints or compliments to the chef or the delivery people  Delivery people can file complaints or compliments to the customers he delivered to. |
| **Preconditions** | The customer purchased food before (for eat-in and pick up, the customer can only complaint or compliment to the chef)  The delivery person delivered dishes to the customer before |
| **Trigger** | The customer is dissatisfied or very satisfied with the dishes and delivery.  The customer takes an abominable attitude toward the delivery person |
| **Scenario** | For customers:  1)The customer can find the chef and the delivery person from his order history  2)The customer can go to the description page of the chef or the delivery person by clicking the name  3)The customer can click “Complaint” or “Compliment” and write down the reason  4)The system sends the complaint or compliment and the reason to the manager.  For delivery people:  1)The delivery person can find the customer from his delivered history  2)The delivery person clicks the username of the customer and selects “Complaint” and then writes down the reason.  3)The system sends the complaint and the reason to the manager. |
| **Exceptions** | None |

**Table 2.1.12**

Class diagram:

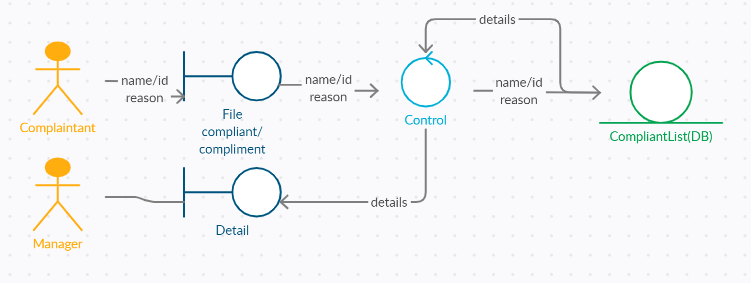


Table 2.1.13 presents the scenario of the Dispute Complaints use case

|  |  |
| --- | --- |
| **Use Case** | Dispute Compliant |
| **Primary Actor** | Customer, VIP, Delivery people, Chef |
| **Goal In Context** | When the customer, the chef or the delivery person disagrees with the complaint, they can dispute the complaint. |
| **Preconditions** | The customer, the chef or the delivery person receives a submitted complaint |
| **Trigger** | They are dissatisfied with the result |
| **Scenario** | 1)The system displays a window to inform that he was complained and the reason  2)If the customer, the chef, or the delivery person disputes the complaint, he can write down the reason and click “Dispute”  3）The system sends this compliant to the manager |
| **Exceptions** | None |

**Table 2.1.13**

Class diagram:

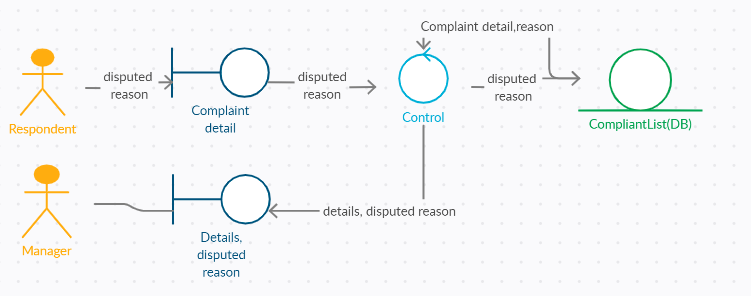


Table 2.1.14 presents the scenario of the Handle Complaints use case

|  |  |
| --- | --- |
| **Use Case** | Handle Compliant |
| **Primary Actor** | Manager |
| **Goal In Context** | Manager handles the complaints |
| **Preconditions** | None |
| **Trigger** | Someone files a complaint to the other or disputes the complaint he received. |
| **Scenario** | 1)The system lists the complaints  2)The manager can click the complaint to see the details  3)The manager handles the complaint  4)If the manager thinks the complaint is without foundation, he can dismiss the complaint, also he could send a warning to the person who files the complaint.  5)If the manager agrees with the complaint, then the system would inform the person to whom the complaint is. |
| **Exceptions** | None |

**Table 2.1.14**

Class diagram:

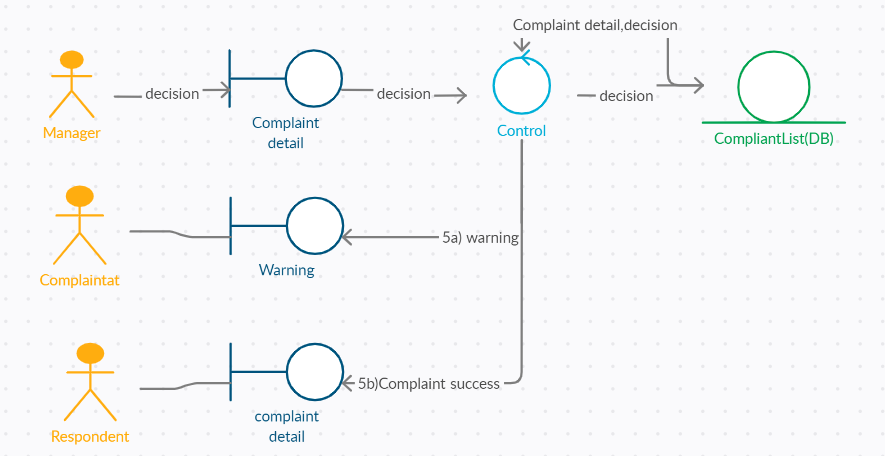
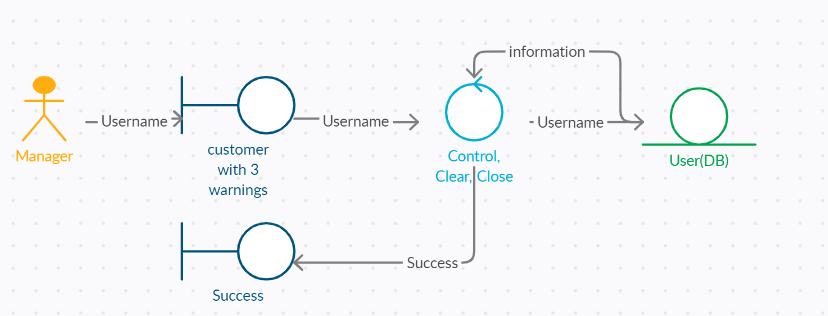


Table 2.1.15 presents the scenario of the Kick use case

|  |  |
| --- | --- |
| **Use Case** | Kick |
| **Primary Actor** | Manager |
| **Goal In Context** | Manager kicks the customer out. |
| **Preconditions** | None |
| **Trigger** | A customer receives 3 warnings and de-registered.  A customer wants to quit the systems |
| **Scenario** | 1)The system displays the customer who is de-registered or wants to quit the system  2)The manager clicks “Confirm”  3)The system clears the deposit and delete the account and file of the customer |
| **Exceptions** | None |

**Table 2.1.15**

Class diagram:



# E-R diagram for the entire system

# Detailed Design

## Login

Login feature includes customer login and staff login

### Customer Login

Customer Login requires customer’s username and password. When the combination of the username and password match a record in the user login database, the application will guide the user to the main page; otherwise, the application will stay in the login page and show the warning message about login failed.

|  |
| --- |
| /\* User Login function \*/  function userLogin(userName, userPassword)  begin      userID = queryUserLoginDB(userName, userPassword);      if userID exists then          startUserMainPage(userID);      else          print("UserName and password didn't match.");      end if  end |

### Staff Login

Staff Login requires staff’s id and password. the combination of the staff id and password match a record in the staff login database, the application will route to staff’s main page, according to the staff type; otherwise, the application will stay in the login page and show the warning message about login failed.

|  |
| --- |
| /\* staff Login function \*/  function staffLogin(staffID, staffPassword)  begin      staffType = queryStaffLoginDB(staffID, staffPassword);      if staffType exists then          if staffType == Chef              startChefMainPage(staffID);          else if staffType == Delivery              startDeliveryMainPage(staffID);          else if staffType == Manager              startManagerMainPage(staffID);          end if      else          print("StaffID and password didn't match.");      end if  end |

## View Menu

This function gets the ids of dishes from the menu database and return the list.

|  |
| --- |
| /\* view Menu \*/  function getMenu()  begin      dishesList = query Menu DB for dish IDs where the dishes are in regular menu;      return dishesList;  end |

## View Special Menu

This function can only be reached by VIP customers. This function will take the customer id as input, and return the special dishes list if and only if the customer is a VIP customer.

|  |
| --- |
| /\* VIP Customers have a special menu \*/  function getSpecialMenu(userID)  begin      userType = queryUserAccount(userID);      if userType == VIP          specialMenu = query Menu DB for dish IDs where the dishes are in special menu;          return specialMenu;      else          return empty;  end |

## Search Menu

This function will take the list of keywords as input and return the list of distinct dishes ids that matching at least one of the keywords.

|  |
| --- |
| /\* search by keyword \*/  function searchByKeyword(keywords)  begin      resultHashSet = create a hashSet using dish id as key;      for keyword in keywords:          dishList = query keyword DB for dish IDs by keyword;          for dish in dishList:              put dish into resultHashSet;          end for      end for      return resultHashSet;  end |

## Voting

Customers can vote for a dish. This function will

1. Insert the rating to database.
2. Notify a function to calculate if the rating of this dish increases or decreases in the recent 3 days.

|  |
| --- |
| /\* vote \*/  function vote(userID, dishID, rating)  begin      insert dish rating to dish-rating DB;      trigger the rating promote/demote function;  end |

## Rating Change

When a customer made the voting, this function should be called in order to decide whether the chef should get promoted or demoted.

|  |
| --- |
| /\* check if the chef should be promoted, demoted or nothing happen \*/  function ratingChange(dishID)  begin      ratingTendence = get latest week ratings of dishID from DB;      chefID = get chefID according to the dishID;      if high rating in the latest week then          promoteOrDemote(chefID, promote);      else if low rating in the latest week then          promoteOrDemote(chefID, demote);      end if  end |

## Staff Promote Or Demote

This function will take the staff id and promote/demote operation code as input. If the staff get promoted, just increase the staff’s bonus; if the staff get demoted, the system will decrease the staff’s bonus as well as check if the staff should be fire.

|  |
| --- |
| /\* promote and demote handler \*/  function promoteOrDemote(staffID, operationCode)  begin      if operationCode equals promote then          increase the staff's bonus;      else if operationCode equals demote then          decrease the staff's bonus;          if this is the third demote of the staff then              send a fire notication to manager;          end if      end if  end |

## Customer Files Complaints or Compliments

When a customer files a complaint or compliment.

|  |
| --- |
| /\* File Complaint or Compliment \*/  function customerFileComplaintOrCompliment(userID, staffID, operationCode, Message)  begin      insert record to complaint-compliment DB;      processComplaintOrCompliment(userID, staffID, operactionCode);  end |

## Process Customer Complaints or Compliments

If the operation is a compliment, then process immediately.

If the operation is a complaint, notify the manager and wait for review.

|  |
| --- |
| /\* Work Complaint or Compliment \*/  funciton processComplaintOrCompliment(userID, staffID, operationCode)  begin      factor = 1;      status = get staff's complaint stat      if user is VIP then          factor = 2;      end if      if operationCode equals Compliment then          status = status + factor;          if status >= 3 then              promotOrDemote(staffID, promote);              reset status;              write the new status to staff's profile;          end if      else          notify the manager to review the complaint;      end if  end |

## Manager Finalizes the Complaint

A manager can either declines or approves a complaint.

|  |
| --- |
| /\* Finalize Customer Complaint \*/  function finalizeComplaint(complaintID, operationCode)  begin      if operationCode equals Approved then          processCustomerComplaint(complaintID);      else          update the complaint status to declined;      end if      notify the staff and user about the decision;  end |

## Process Finalized Complaint

When the Complaint was reviewed by a manager and being approved, this function will be called.

|  |
| --- |
| /\* Process the finalized Complaint \*/  function processCustomerComplaint(complaintID)  begin      update the complaint status to approved;      userID, staffID = get IDs according to complaintID;      factor = 1;      if user is VPI then          factor = 2;      end if      staffStatus = get staff compliment/complaint status by staffID;      staffStatus = staffStatus - factor;      if staffStatus <= -3 then          promoteOrDemote(staffID, demote);          reset staffStatus;          write staffStatus to staff's profile;      end if  end |

## Customer Disputes a Complaint

When a customer receives a complaint, the customer has a chance to dispute.

|  |
| --- |
| /\* Customer Dispute a Complaint \*/  function customerDisputeComplaint(userID, complaintID, message)  begin      insert the dispute message to DB;      notify manager about this message;  end |

## Customer Place an Order

The process of placing an order.

|  |
| --- |
| /\* Make Order \*/  function placeOrder(userID, dishesList)  begin      totalAmount = 0;      for dish in dishesList          totalAmount = totalAmount + amount of dish;      end for      if user is VIP then          totalAmount = totalAmount \* 0.9;      end if      insert order detail to order DB;      set the order record status to "on processing";      orderID = the id of new inserted record;      paymentCharge(orderID);      if payment is charged successfully then          update the order record status to "Paid";          update the dish-order database;          send the order to the kitchen;          upgradeCustomer(userID);      else          freezeOrder(orderID);          notify user about the freeze;      end if  end |

## Charge Payment

Charge payment from user’s balance.

|  |
| --- |
| /\* charge the payment \*/  function paymentCharge(orderID)  begin      balance = user's balance;      amount = order's amount;      userID = get userID by orderID;      if balance >= amount then          newBalance = balance - amount;          update user's balance to newBalance;          update order's status to Charged;          return success;      else          return fail;      end if  end |

## Freeze Order

Freeze the order.

|  |
| --- |
| /\* freeze order \*/  function freezeOrder(orderID)  begin      update order record status to freeze;  end |

## Fund User Balance

Fund more balance to user’s account.

|  |
| --- |
| /\* fund balance \*/  function fundBalance(userID, amount)  begin      balance = get user balance;      newBalance = balance + amount;      update user balance to newBalance;  end |

## Re-open Froze Order

Retry the payment charge process again.

|  |
| --- |
| /\* reopen froze order \*/  function reopenOrder(orderID)  begin      orderStatus = get order status by orderID;      if orderStatus not equals freeze then          exit;      end if      paymentCharge(orderID);      if payment success then          update the dish-order database;          send the order to the kitchen;          upgradeCustomer(userID);      end if  end |

## Upgrade Customer

Upgrade regular customer to VIP.

|  |
| --- |
| /\* upgrade user \*/  function upgradeUser(userID)  begin      userType = get user type by userID;      if user is VIP then          exit;      end if      orderTotalAmount = calculate the accumulative total;      orders = count the orders the user placed;      if orderTotalAmount > $500 or orders >= 50 then          upgrade this customer to VIP;          notify the customer about this change;      end if  end |

## Discussion Thread

### Create New Discussion

Customers can create discussion on cook/dishes/deliver people.

|  |
| --- |
| /\* create new discussion thread \*/  function createDiscussion(userID, subjectID, message)  begin      discussionHandler(0, userID, subjectID, message);  end |

### Participate in Discussions

Customer can participate in an existing discussion.

|  |
| --- |
| /\* Participate discussions \*/  function joinDiscussion(discussionID, userID, message)  begin      discussionHandler(discussionID, userID, 0, message);  end |

### Discussion Handler

Since creating a discussion and joining a discussion are having similar process with minor differences, a handler function can help abstract the process.

|  |
| --- |
| function discussionHandler(discussionID, userID, subjectID, message)  begin      taboos = calculateTaboos(message);      if taboos == 0 then          Status = open;          triggerWarning = false;      else          triggerWarning = true;          if taboos < 3 then              message = replace taboo words with "\*\*\*";              Status = open;          else              Status = blocked;      end if      insert message to the discusstion DB;      set the discusstion record to Status;      if triggerWarning then          warningMessage = discussion record ID;          customerWarning(userID, tabooWarning, warningMessage);      end if  end |

## Handle Customer Warning

Determine the action when receive the warning.

|  |
| --- |
| /\* Customer recieves warnings \*/  function customerWarning(userID, warningType, warningMessage)  begin      count = get the total warning count by userID;      count = count + 1;      insert detail to warning DB;      notify user about the warning;      if user is VIP then          if count >= 2 then              downgradeCustomer(userID);              reset user warning count;          end if      else if count >= 3 then          send a notification to manager about kicking out the user;      end if  end |

## Downgrade a Customer

Downgrade a VIP customer to regular customer.

|  |
| --- |
| /\* downgrade a customer \*/  function downgradeCustomer(userID)  begin      if user is not VIP then          exit;      end if      set the user to regular customer;      notify the user about the change;  end |

## Customer Registration Request

When a surfer has the interest to register to become a customer, the surfer can send out an request.

|  |
| --- |
| /\* Customer Registration Request \*/  function customerRegistrationRequest(email)  begin      if email exists in the system then          notify surfer that he/she has an account;          exit;      end if      Send a customer account request to manager;  end |

## Check Registration Code

When the surfer gets a registration code from manager, the surfer should provide the code and the email to the system for validation.

|  |
| --- |
| /\* Check the registration code \*/  function customerRegValidor(registrationCode, email)  begin      if registrationCode is not valided then          notify the surfer this issue;      else          customerAccountCreation(email);      end if  end |

## Create Customer Account

Let user provide the username, password and the amount that the user will fund.

|  |
| --- |
| /\* account creation \*/  function customerAccountCreate(userName, password, fundAmount)  begin      systemThreshold = get the system balance requirement of customer registration;      if fundAmount >= systemThreshold then          while userName is token              let user try another user name;          end while          create new customer account;      else          ask user fund the minimum requirement of balance;      end if  end |

## Kitchen Preparing the Order

When the kitchen is cooking the order, update the order status.

|  |
| --- |
| /\* take order - kitchen \*/  function kitchenTookOrder(orderID)  begin      if order is in "Paid" status then          update order status to "cooking";          notify the user about the change;      end if  end |

## Order Ready to Delivery

Once the kitchen done with the order, it is ready for delivery.

|  |
| --- |
| /\* Ready for delivery \*/  function kitchenOrderDone(orderID)  begin      if order is in "cooking" status then          update order status to "Ready";          notify the user about this change;          send notification to Delivery;      end if  end |

## Edit Menu

The chef can edit menu.

|  |
| --- |
| /\* Edit menu \*/  function editMenu(dishesList, menuID)  begin      bound the menuID with the dishes IDs list;      update the database;  end |

## Edit Keyword

|  |
| --- |
| /\* edit keyword \*/  function editKeyword(dishID, keywords)  begin      bound the keywords to dish;      update the database;  end |

## Order Delivery

|  |
| --- |
| /\* take order - Delivery \*/  function deliveryTookOrder(orderID, staffID)  begin      update the order status to "Out of Delivery";      notify the user about the change;  end |

## Order Delivered

|  |
| --- |
| function orderDelivered(orderID, staffID)  begin      update the order status to "Delivered";  end |

## Human Resources

### Hire

|  |
| --- |
| function hire(managerID, position)  begin      create new account of the given position;      return staffID;  end |

### Fire

|  |
| --- |
| function fire(managerID, staffID)  begin      set the staff status to "fired";  end |

## Edit Taboo Words

The manager can add or edit the existing taboo words list

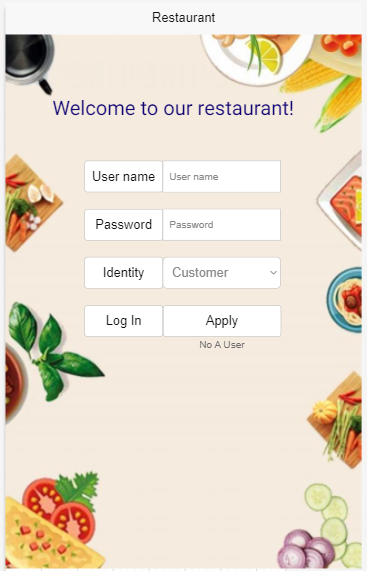
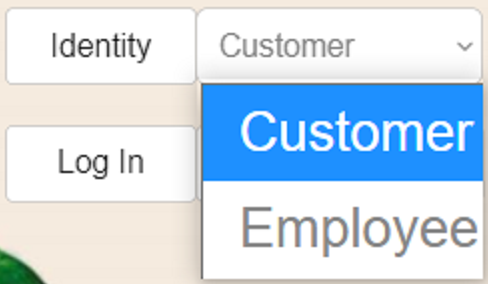
|  |
| --- |
| /\* Taboo list \*/  function editTaboo(tabooWords)  begin      for word in tabooWords          add the word to database;      end for  end |

## Handle New Registration Request

|  |
| --- |
| /\* handle new registration request \*/  function handleRegistration(email)  begin      create a new registration code for the email;      insert a record to the registration request database;      send out the code to the email provided;  end |

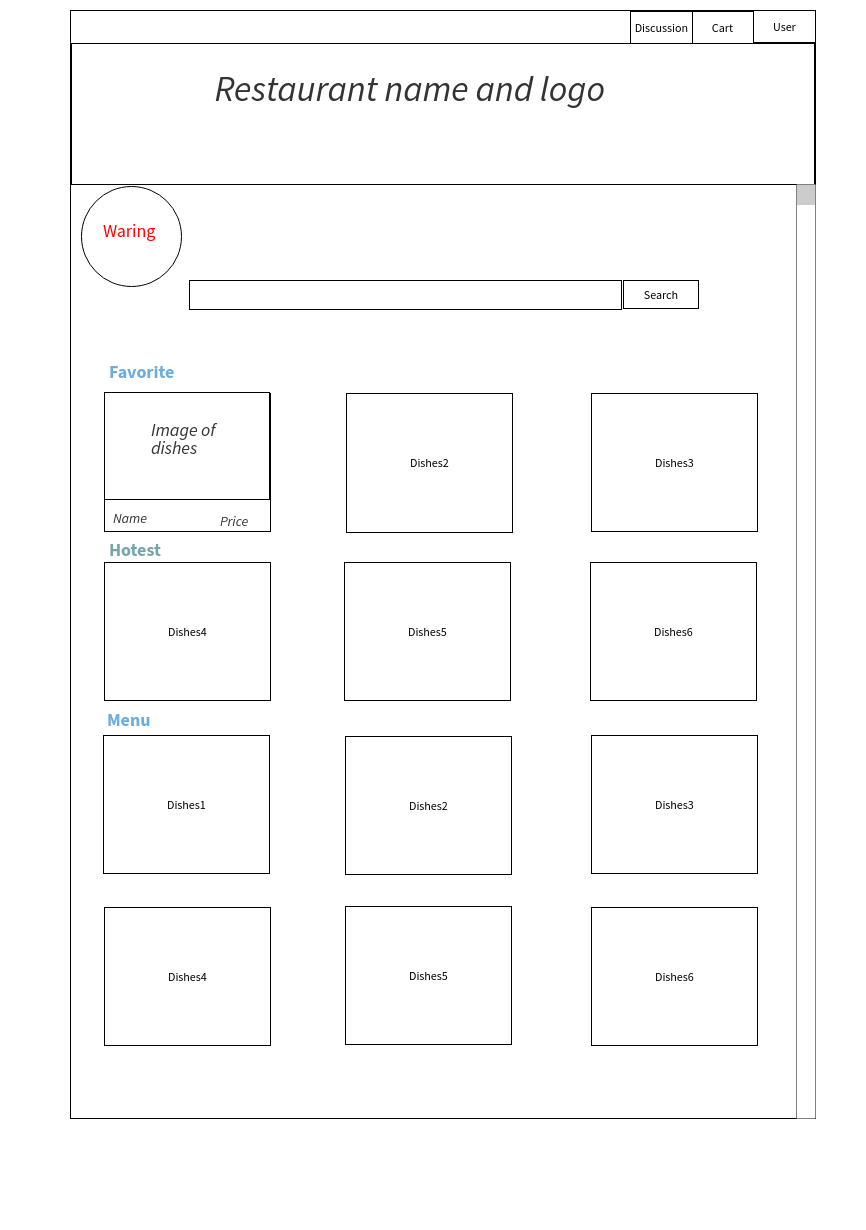
# System Screens

## Login page



Login page design as the user enters username and password, then clicks the ‘Identity’ button to choose what he/she is following and then login by clicking the ‘Log In’ button. If the user is new to this app, he/she can click ‘Apply’ to register.

## Customer screen



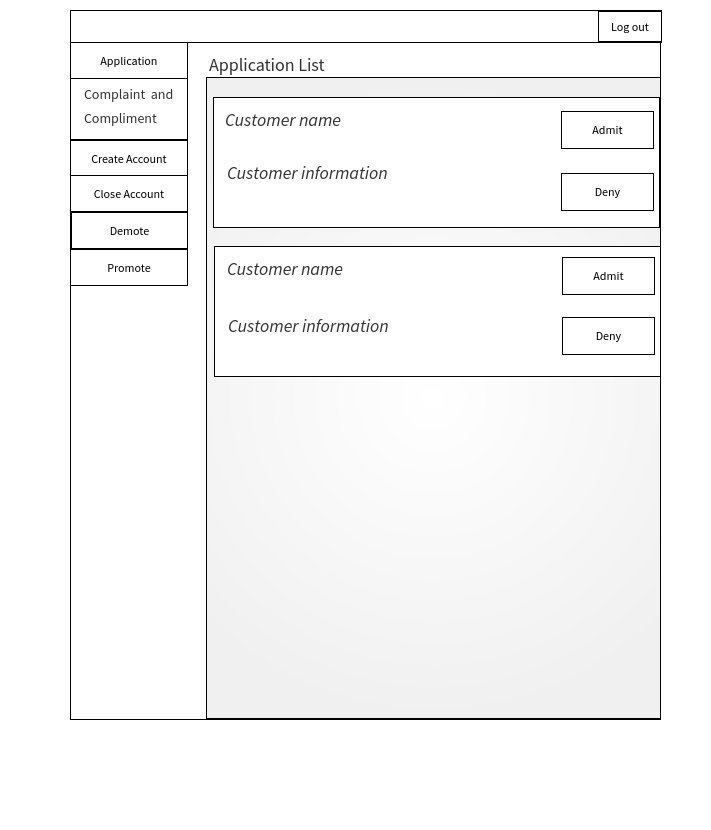
The ‘Favorite’ part displays the most 3 dishes the customer purchased. The ‘Popular’ part displays the most 3 popular dishes. The ‘Menu’ part displays all dishes, and the ‘Waring’ part shows the warnings the customer had received. The customers can click ‘User’ to view and edit their individual information. The customers can click the dishes to view the description and add the dishes to the order cart. By clicking ‘Discussion’, the customers can go to the discussion screen to join or start the discussion.

## Dishes screen

When the customer clicks the dishes, the dishes screen would be displayed.

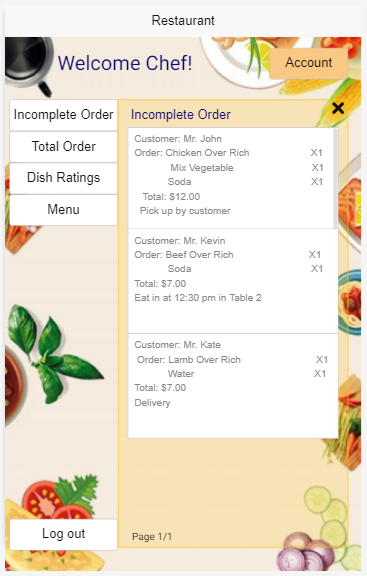


## Manager screen



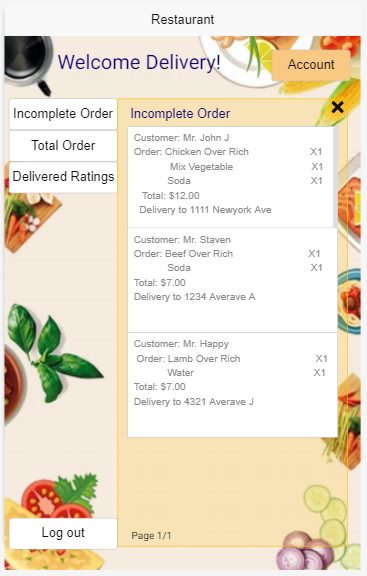
When the manager logins, the application list would be displayed by default. Manager can view the information of the applicant and decide to admit or deny the application. The manager can click the ‘Complaints and Compliments’ to view the details of the complaints and the compliments and handle them. In the ‘Create Account’ part, the manager can create a new account for the new chef or the new delivery person. The manager can close the account of the employees and the customers who are fired or kicked out in the ‘Close Account’.

## Chef screen



This is the chef page, ‘Incomplete Order’ set as default, and the chef can click on the buttons to check different parts. If the chef wants to check the complaints/ compliments, he/she can click ‘Account’ bottom (on the up-right corner) to explore.

## Delivery people screen



This is the delivery people page, ‘Incomplete Order’ set as default, and delivery people can click on the buttons to check different parts. If delivery people want to check the complaints/ compliments, he/she can click the ‘Account’ button (on the up-right corner) to explore.

# Group Meetings

Project: Online restaurant order and delivery system

Meeting Date: 10/31/2020

Meeting Time: 25 minutes

Recorder: Bingjing Dong

Present: Leji Li, Weiye Kuang, Xian Chen, Bingjing Dong

1st group meeting: we separated each part and talked about who we wanted to do, we chose what we preferred first and see if there is anything left over. Leji will do introduction and detailed design. Weiye will do all use cases. Xian will do an E-R diagram for the entire system. Bingjing & Weiye will do a system screen and create a raw git repo to add our group member in.

Project: Online restaurant order and delivery system

Meeting Date: 11/07/2020

Meeting Time: 20 minutes

Recorder: Bingjing Dong

Present: Leji Li, Weiye Kuang, Xian Chen, Bingjing Dong

2nd group meeting: Since last week we have our midterm, we didn’t force too much on the project. We simply talk about what we looked at last week and what we think about how to do. We will continue to work on our part this week and come back for any problem we meet next week.

Project: Online restaurant order and delivery system

Meeting Date: 11/14/2020

Meeting Time: 30 minutes

Recorder: Bingjing Dong

Present: Leji Li, Weiye Kuang, Xian Chen, Bingjing Dong

3rd group meeting: we share what we have up to today. Also ask other group members for any suggestions. We also talk about what we meet questions in doing our part, one of those is that if we design too much, it is hard for us to do the next step. Also, we will meet in two days to finalize and put all parts in one file.

# Git Repo

https://github.com/CSC322TeamL