Food Connect, Inc.

Project Moonshot
System Design
For Electronic Restaurant Order and Delivery System

Vinuk Ranaweera
Minh Le
Khanh Huang
Yauheni Patapau
Tea Nurcellari

Version 0.94

Revision History

Date	Version	Description	Author
4/20/2022	0.91	initial draft version	Minh Le, Tea Nurcellari, Khanh Huang
4/28/2022	0.94	updated version	Khanh Huang, Yauheni Patapau, Vinuk Ranaweera

Table of Contents

Software Design Specification

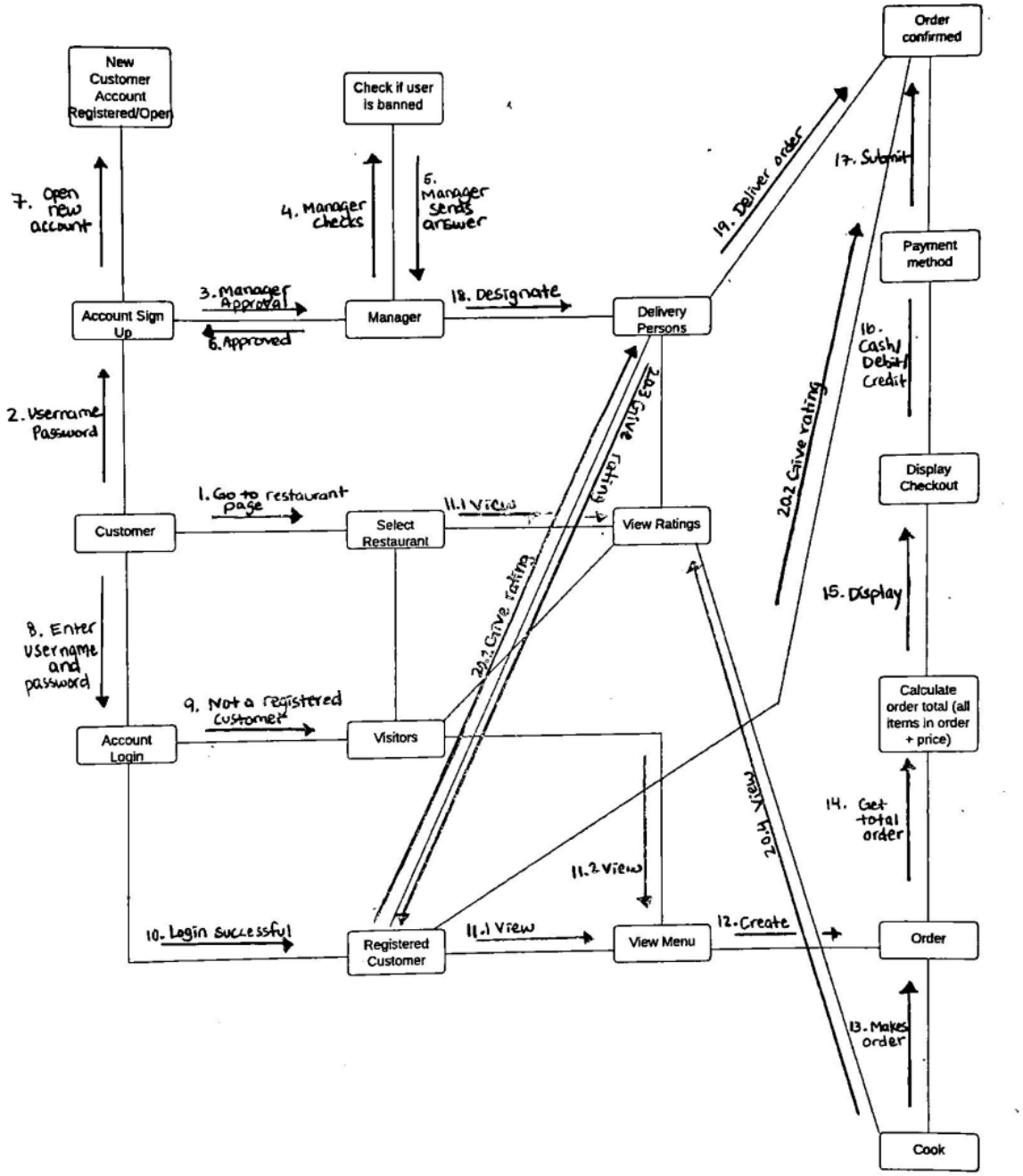
1. Introduction

This report is our phase 2 project report in which we provide the data structure and logic to carry out the functionalities dictated by the specification.

In this report we have included several diagrams which include the collaboration class diagram, sequence class diagram, Petri-nets, and E-R diagram. In the introduction portion we use the collaboration class diagram, also known as a communication diagram, which depicts the links and interactions among software components (UML). These diagrams can be used to depict the dynamic behavior of a certain use case and define each object's purpose. In section 2 of the report (all use cases) we introduce the sequence class diagram which depicts interactions between classes as a series of messages exchanged over time. A sequence diagram is an effective tool for visualizing and validating different runtime scenarios. Petri-nets are also introduced in section 2 of the report and it is a graph model for controlling the behavior of systems that operate concurrently. The last diagram we used in our report is the E-R diagram mentioned in section 3. We have also included pseudocode as well as demonstrated major GUI screens of the system and a sample prototype of one functionality of our own choice. Our pseudocode is a detailed design for every method which delineates the input/output and main functionalities of our food delivery service.

To conclude our report we have included detailed meeting memos from our various group meetings held over zoom as well the address of our github repository.

1.1 - Collaboration Class Diagram (an overall picture of the system):



2. Use Case

1. Overview

This section provides the collaboration diagrams and the petri-nets for the interactions between the system and the users.

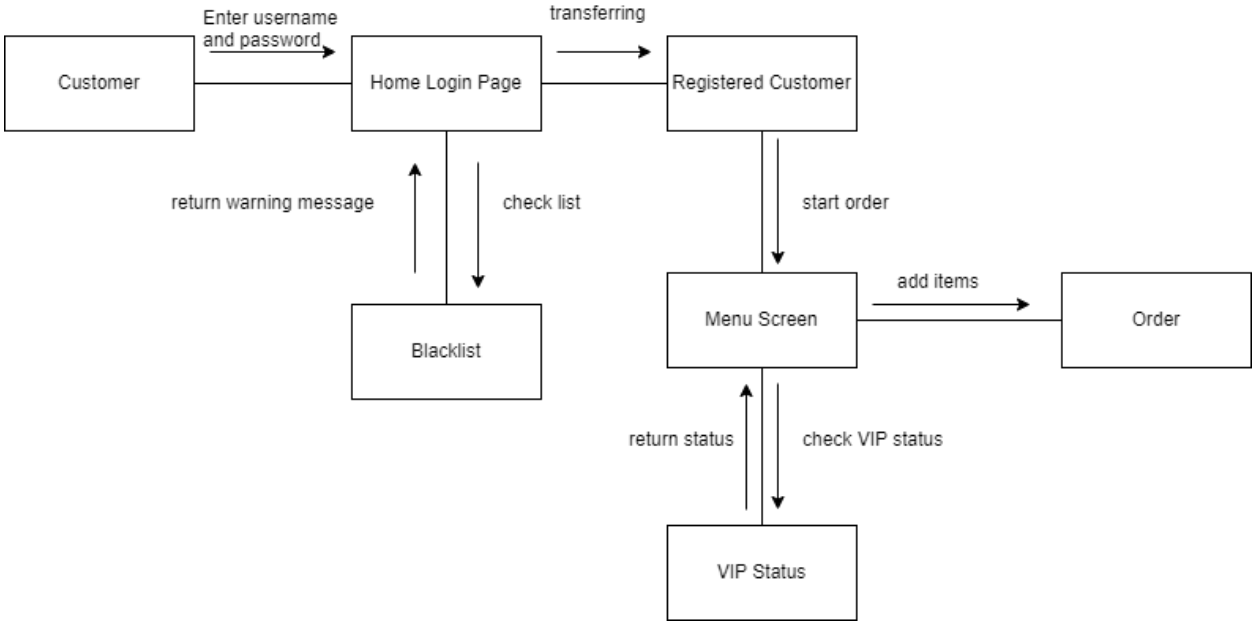
2. All use cases

- Browse/Search Items

Normal Scenario: Registered customers along with visitors are greeted with a screen that allows them to select from a variety of food options along with the quantity of each order.

Exceptional Scenarios: VIPs are able to view special items not accessible to visitors or registered customers.

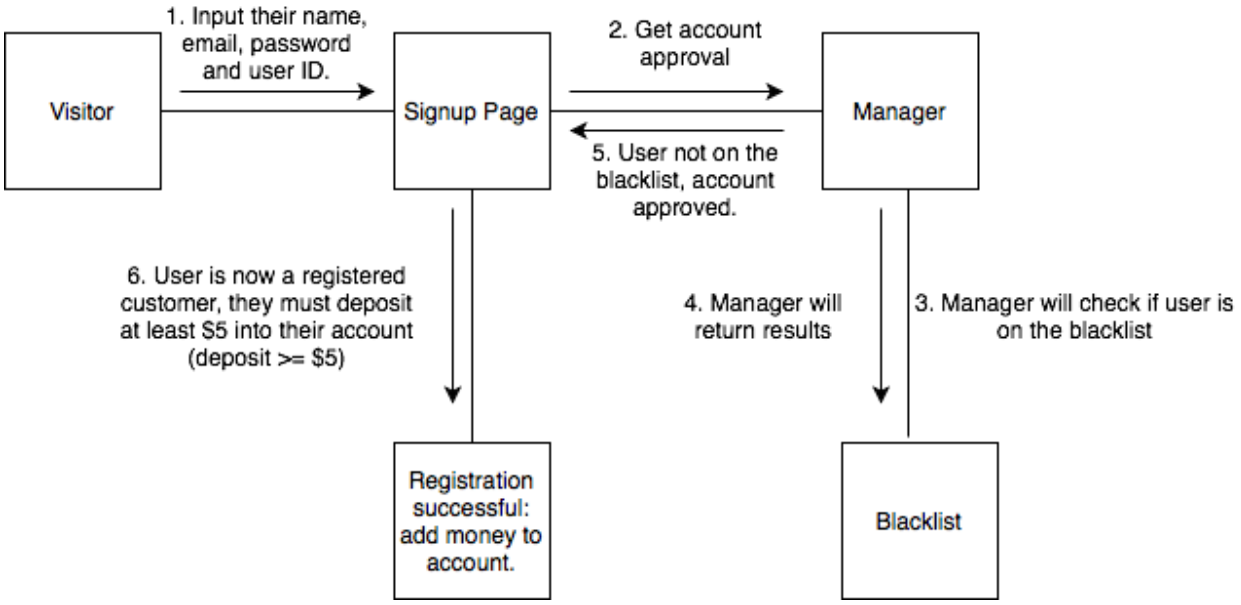
For registered customers/VIPs, they will have a unique top 3 listing dishes based on the history of their prior choices when they login.

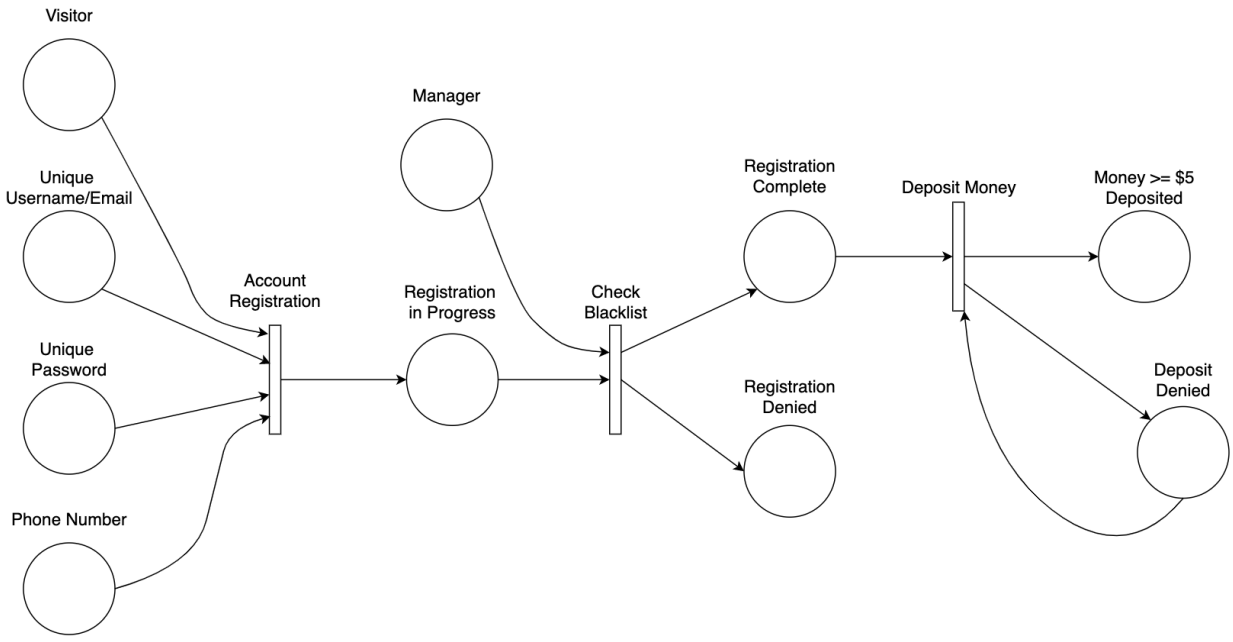


- User Registration

Normal Scenario: Visitors can sign up to be a registered customer with a unique username/email, password, and phone number. Upon registration, users must deposit money into account (at least $5).

Exceptional Scenarios: The customer's background is checked and if the customer is listed in their blacklist, their account registration will be denied.



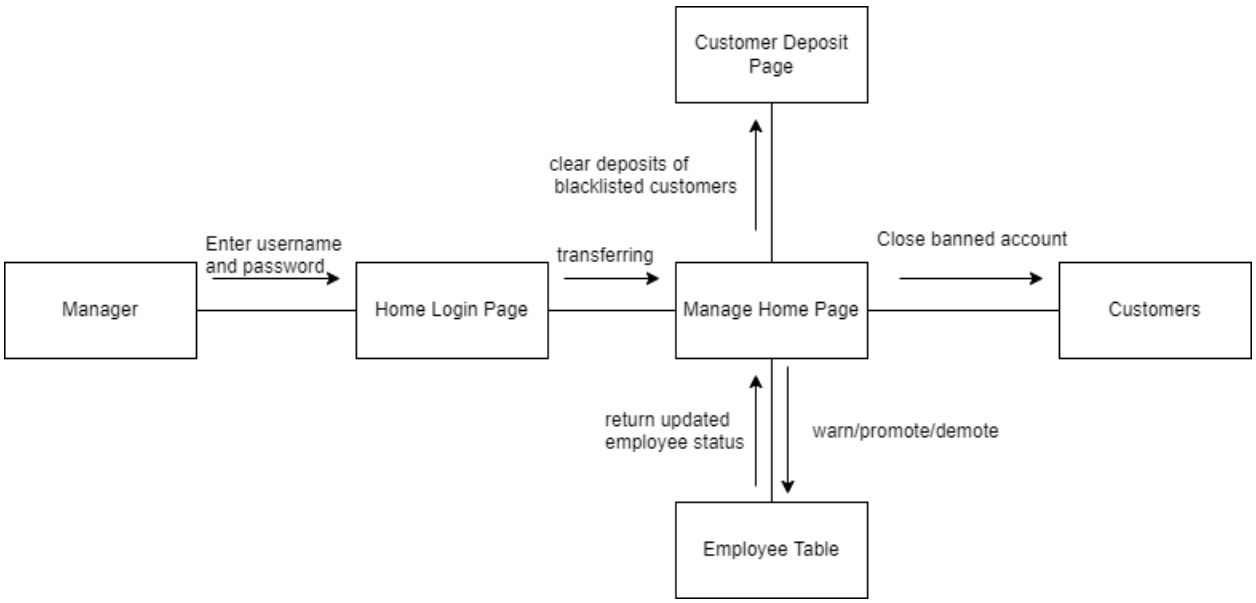


- Manage Accounts

Normal Scenario: The manager handles all system management pertaining to warnings and promotions/demotions.

Exceptional Scenarios: For every registered customer/VIP who is banned from the system or chooses to quit the system, the manager must clear the deposit and close the account.

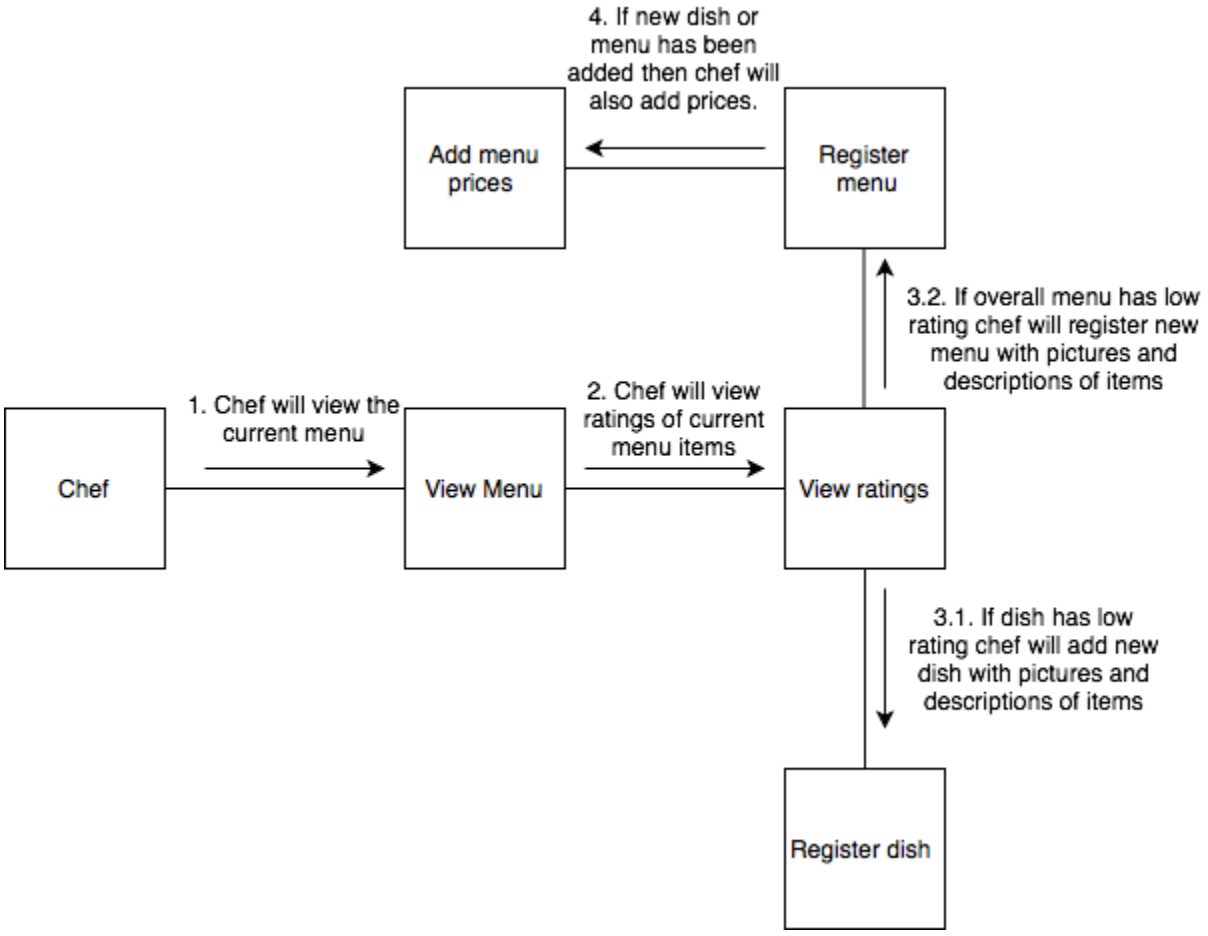
Any banned customer is on the blacklist of the restaurant (not visible to customers) and cannot register any more.



- Item Registration

Normal Scenario: Chefs can register their dish/menu into the system along with applicable pictures and descriptions.

Exceptional Scenarios: The item prices are decided by the chef.

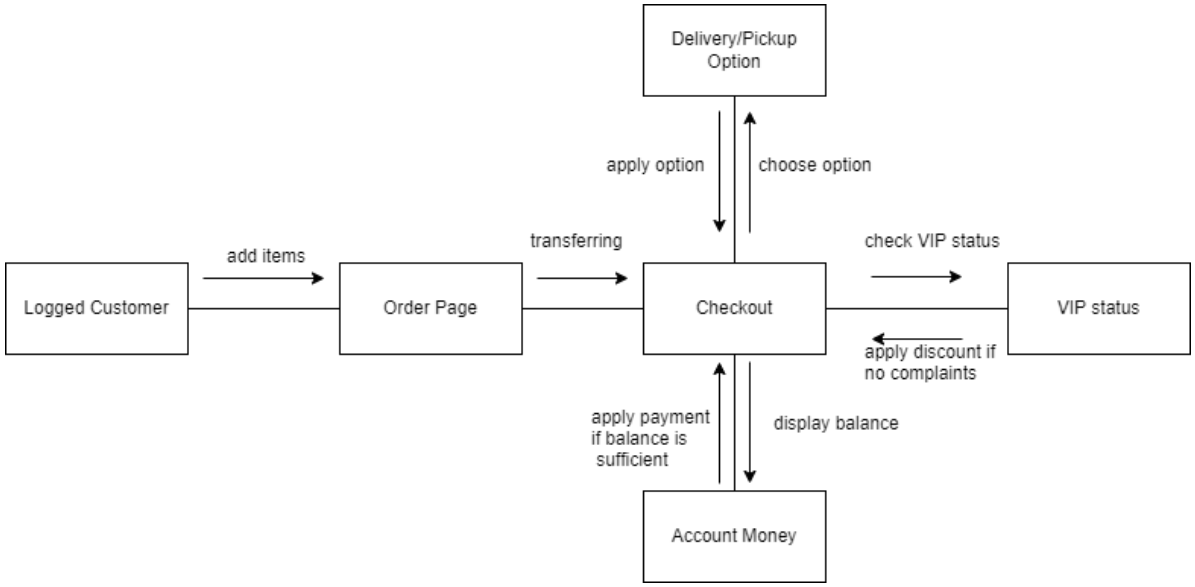


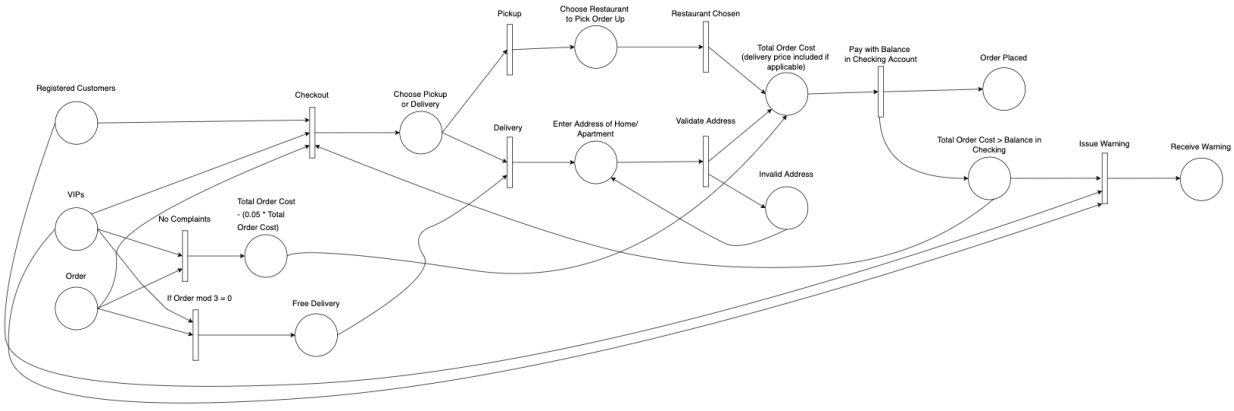
- Purchase Items

Normal Scenario: After the customer is content with their order, they will be able to checkout their order. They then proceed to the checkout page where they will pay with the money in their account. After the user chooses if the items are to be picked up at store or by restaurant delivery, the order will be submitted.

Exceptional Scenarios: If VIP has no complaints, they receive 5% discount on ordinary orders and 1 free delivery for every 3 orders.

For registered customers/VIPs, if the price of the order is greater than deposited money in the account, the order is rejected and the user will get one warning.



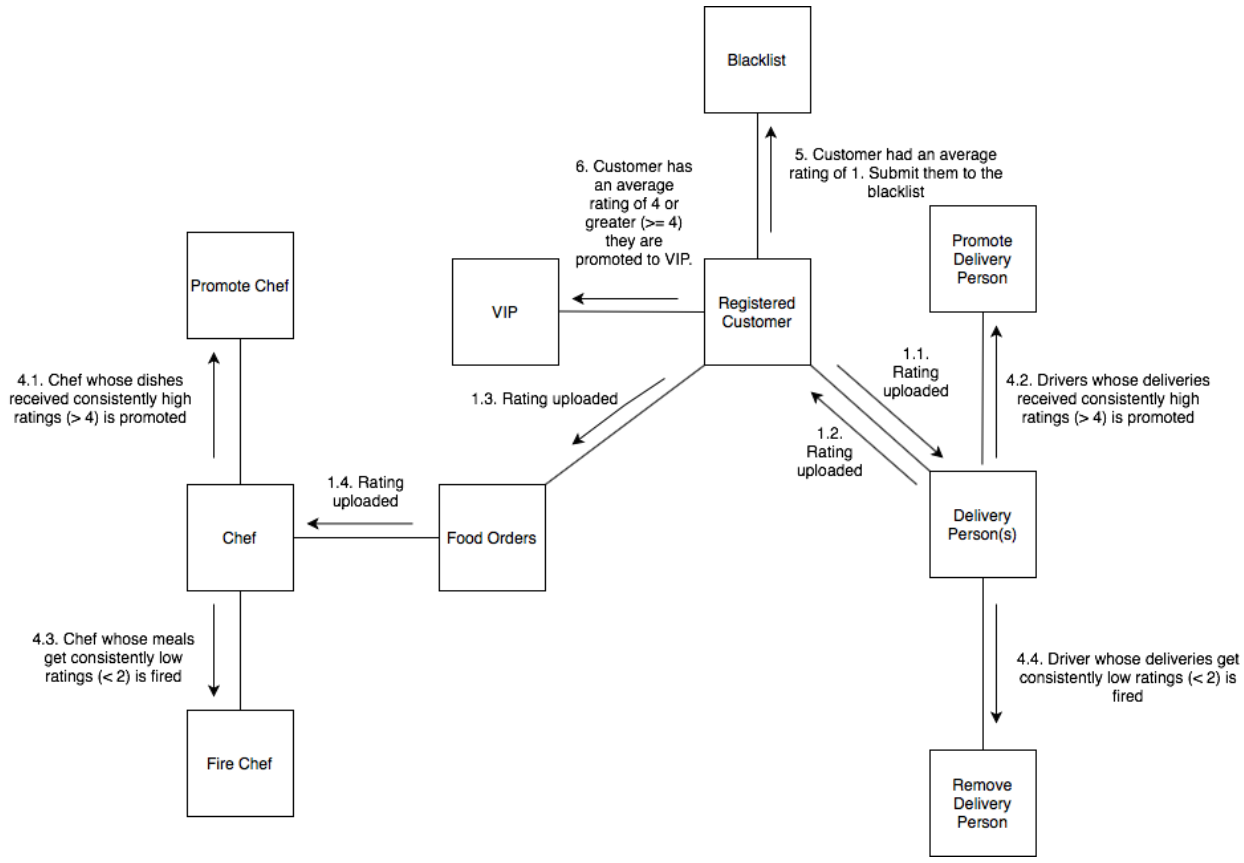


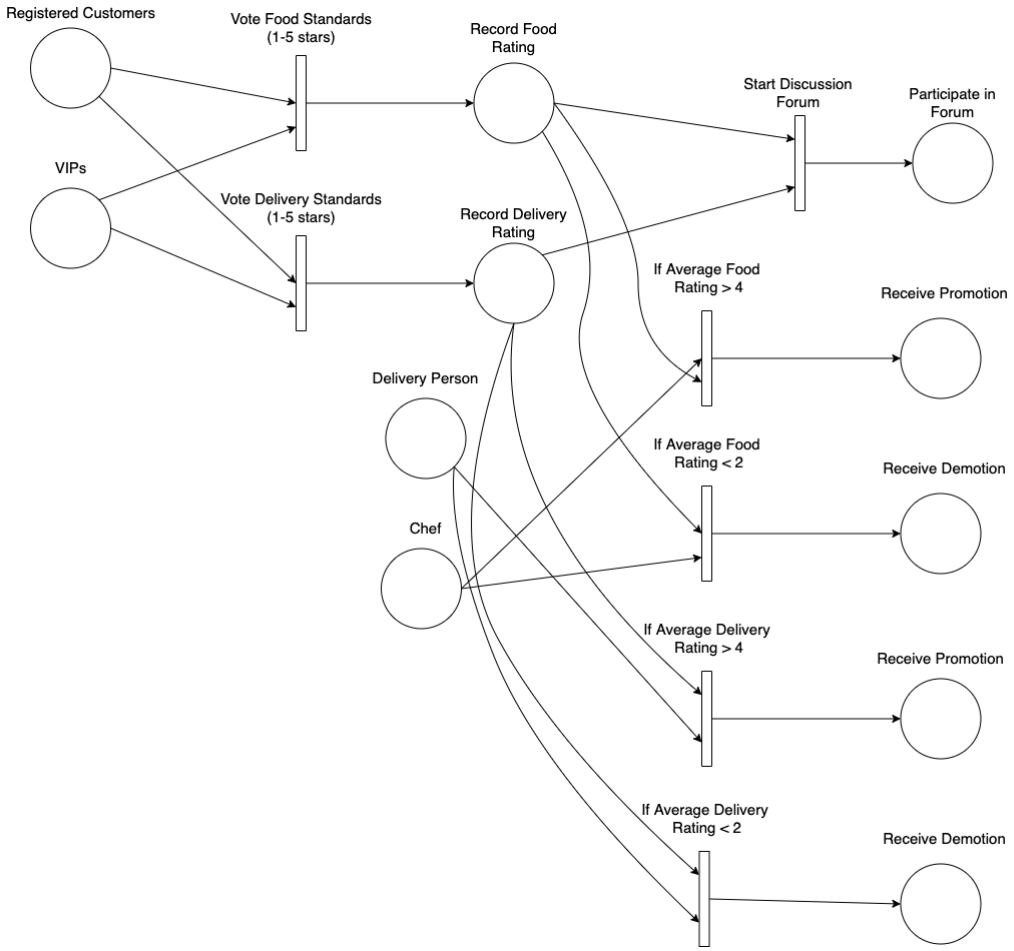
- Rating Users

Normal Scenario: Registered customers/VIPs can vote on food and delivery standards individually (1 star being the worst and 5 stars being the best) and start/participate in discussion forums.

Exceptional Scenarios: A chef whose dishes received consistently low ratings (< 2) or delivery person with consistently low ratings (< 2) will get a demotion.

A chef whose dishes received consistently high ratings (> 4) or delivery person with consistently high ratings (> 4) will get a promotion.




Registered Customers
Vote Food Standards
(1-5 stars)
Record Food
Rating
Start Discussion
Forum
Participate in
Forum
VIPs
Vote Delivery Standards
(1-5 stars)
Record Delivery
Rating
If Average Food
Rating > 4
Receive Promotion
Delivery Person
If Average Food
Rating < 2
Receive Demotion
Chef
If Average Delivery
Rating > 4
Receive Promotion
If Average Delivery
Rating < 2
Receive Demotion

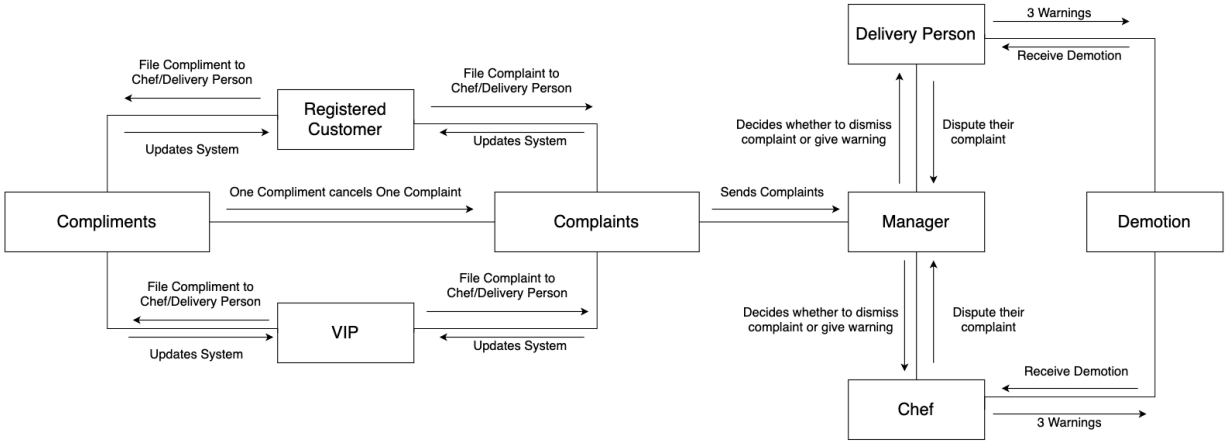
- File Complaints/Compliments

Normal Scenario: Registered customers/VIPs can file compliments/complaints to chefs, delivery people, or other customers that do not behave in discussion forums. All complaints/compliments are to be handled by the manager.

Exceptional Scenarios: Accused chefs and delivery people can dispute their complaint and the manager ultimately decides whether to dismiss the complaint or let the warning stay.

For chefs and delivery people, one compliment can be used to cancel one complaint.

Chefs and delivery people with 3 complaints will receive a demotion.

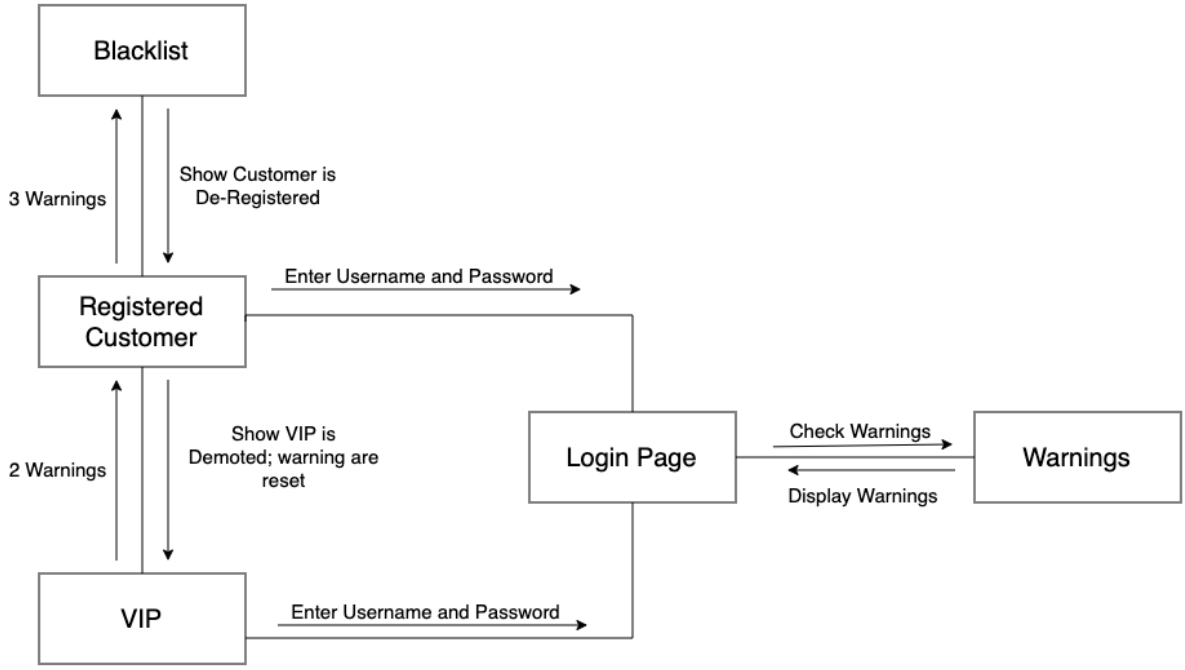


- Issue Warnings

Normal Scenario: Warnings display on the page when the registered customer/VIP logs in.

Exceptional Scenarios: VIPs having 2 warnings are demoted to registered customers, but warnings are then cleared.

Registered customers having 3 warnings are de-registered.

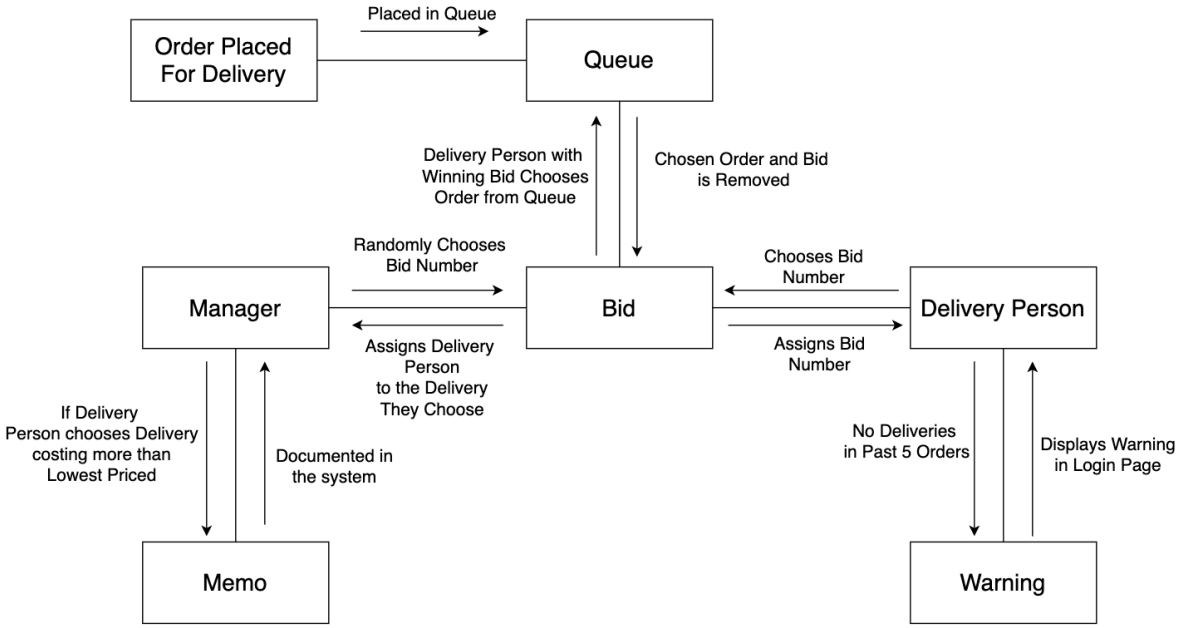


- Bidding System

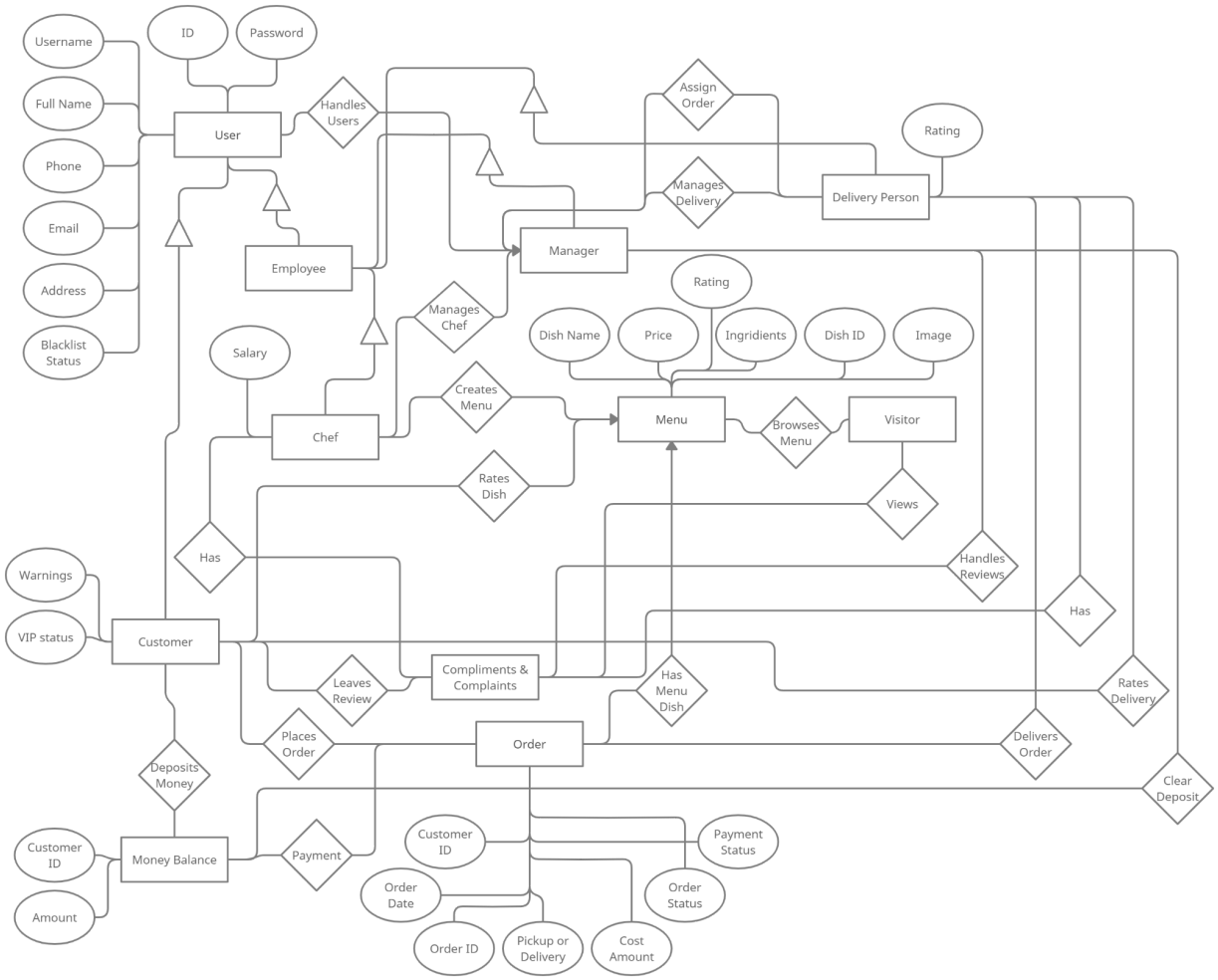
Normal Scenario: When there are multiple deliveries to be made, each delivery person will bid to choose an order to deliver (the first person to win will choose whichever order to deliver and the last person delivers the remaining order). The manager assigns the order from bidding results.

Exceptional Scenarios: Generally, the first person is expected to deliver the lowest priced order; however if he/she chooses to deliver an order higher than the lowest priced, the manager must write a memo in the system.

A delivery person who didn't deliver any in the past 5 orders will automatically receive one warning.



3. Entity-Relationship diagram



4. Design

```
// pseudocode
// data structures

struct Customer {
  int    customer_id;   // unique identifier
  char   name[80];
  float  balance;    // money available to order food
  char   phone[20];
  char   email[200];
  char   street[200];
  char   address2[200];
  char   city[200];
  char   state[2];
  int    zip[5];
  int    status_VIP;  // spending exceeds $100 or more than 5 orders
  int    warnings;    // exceeding 3, account will be terminated
  int    feedback;    // compliments or complaints status (VIP status elevates this field)
  float  total_money;  // total amount from purchase history
  int    number_of_orders;  // total number of orders
  int    discount;          // percent discount on food orders VIP status
  bool   free_delivery;     // toggle this field to indicate free delivery on every 3 orders
  struct Food_Purchase_Orders *history;  // linked-list of previous orders

};

struct Delivery_Person {
  int    delivery_person_id;   // unique identifier
  char   name[80];
  char   phone[20];
  int    salary;
  int    demotions;  // after demoted twice, chef will be fired
  int    ratings;    // food dish ratings total count
  int    compliments;
  int    complaints;

  struct Food_Purchase_Orders *history;  // linked-list of previous orders

};

struct Chef {
  int    chef_id;   // unique identifier
  char   name[80];
  int    salary;
  int    demotions;  // after demoted twice, chef will be fired
  int    terminated; // chef is fired after twice demotions
  int    ratings;    // food dish ratings total count
  int    compliments;
  int    complaints;
  int    bonus;      // high ratings or more compliments

  struct Food_Dish_Record *dish;  // linked-list of previous orders

};
```

```
struct Food_Purchase_Orders {
   int    customer_id;     // unique identifier
   int    transaction_id;  // unique identifier

   float  amount;    // total order cost
   int    quantity;  // items count in this transaction

   int    delivery_pickup;  // 0:  pickup
                            // 1:  home delivery

   float  delivery_price;

   struct Food_Dish_Record  *food_items;  // list of items purchased

   int complete;          // order status (0: in progress / 1: completed)
   Date datetime;         // timestamp of food order
}


struct Food_Dish_Record {
   int    dish_id;     // unique identifier
   int    chef_id;     // chef made this dish

   int    sold_quantity;  // items count in one transaction
   struct Dish_Ratings *ratings;
}


struct Dish_Ratings {
   int    quality;  // 1: very bad
                    // 5: excellent
   int    customer_id; // ratings from this customer

   int    compliments;
   int    complaints;

   char  *feedback;    // comments from customer
}


/* create new account */
/* minimum amount to open is $5
 * input parameter:  pointer to data structure containing customer info
 * return value:  integer representing customer ID
 *                zero indicates failure to add new account to system
*/
int Account_Create(Customer *x, float amount) {
    if (Validate_info(x) == false)
        return 0;  // account is on block list, cannot create

    // minimum funding to create new account is $5
    if (amount < 5)
        return 0;   // cannot proceed

    Account_Add(x);

    return x->customer_id;
}
```

```
/* customer choose to close account */
 * input parameter:  pointer to data structure containing customer info
 * return value:  true/false
                  true indicates account is now closed; otherwise, return false
*/
boolean Account_Close(Customer *x) {
    if (Validate_info(x) == false)
        return false;   // cannot close non-existent account

    authorization_code = Managers_Authorize_Accnt_Closing(x);  // managers provides
authorization code to allow account closing
    Account_Removed(x, authorization_code);

    return true;
}

/* customer account is terminated by manager */
 * input parameter:  pointer to data structure containing customer info
 * return value:  true/false
                  true indicates account is now closed; otherwise, return false
*/
boolean Account_Terminate(Customer *x) {
    if (Validate_info(x) == false)
        return false;  // cannot close non-existent account

    authorization_code = Managers_Authorize_Accnt_Closing(x);  // managers provides
authorization code to block account

    Account_Block_List(x);  // add customer to blacklist (cannot re-register)
    Account_Removed(x);     // remove customer info from system

    return true;
}

int Managers_Authorize_Accnt_Closing(Customer *x) {
    number = Request_Code(x);   // authorization code to close customer account

    Verify_Authorization_Code(number);
    Refund_Customer_Money(x);   // issue refund check to customer mailing address

    x->balance = 0;   // set balance to zero

    return number;
}



/* login prompt   * * *
*
*/
Login_Prompt (username, password) {
  if login = entered username & password correctly then
        Login success
  else
         Login failed (show front page again)

  switch (username):
    case 1     /* employees */
```

```
         Display_Login_Screen(1)

    case 2     /* customer */
         Display_Login_Screen(2)

    case 3     /* visitor */
         Display_Login_Screen(3)

}


Display_Login_Screen(username) {

  switch (username):
    case 1     /* employees */
         Display_Welcome_Page(username)

    case 2     /* customer */
         // populate customer data fields
         Get_Customer_Data(x)
         Display_Customer_Welcome_Page()
}

Display_Customer_Welcome_Page(username) {
  if (x->history == 0)    // no purchase history
      new_customer = true

  if (new_customer) {
      Show_Popular_Dishes(3)       // display top-3 dishes popularity
      Show_High_Rating_Dishes(3)   // display top-3 dishes having high rating
  }
  else {
      Show_High_Rating_Dishes(3)   // display top-3 dishes having high rating
  }
}


/* customer account info */
/* input parameter:  pointer to data structure containing customer info
 * return value:  customer current balance
*/
float Get_Balance(Customer *x) {
    // populate customer data fields
    Get_Customer_Data(x);

    return x->balance;
}

/* increase funds in customer account */
/* input parameters:  pointer to data structure containing customer info
 * amount:  new money funding this account
 * return value:  customer new balance
*/
float Add_Money(Customer *x, float amount) {
    // populate customer data fields
    Get_Customer_Data(x);

    new_balance = amount + x->balance;
    x->balance = new_balance;
```

```
    return x->balance;
}

/* customer account balance update */
 * input parameters:  pointer to data structure containing customer info
 * amount:  this amount represents food purchase from order transaction
 * return value:  customer current balance
*/
float Update_Balance(Customer *x, float amount) {
    // populate customer data fields
    Get_Customer_Data(x);

    new_balance = x->balance - amount;
    x->balance = new_balance;

    return x->balance;
}


/* food order
 * input parameters:  pointer to data structure containing customer info
 * total: this total amount represents food purchase from current order transaction
 * transaction_id: record identification of order transaction
 * return value:  customer new balance after purchase
*/
float Food_Order(Customer *x, float total, int transaction_id) {
    // populate customer data fields
    Get_Customer_Data(x);

    // if order total exceeds account balance, send warning to customer now
    if (total > x->balance) {
        x->warnings += 1     // update warnings count

        Cancel_Order(x, transaction_id);   // cancel food order immediately

        if (x->warnings >= 3)
            Account_Terminate(x)    // close account immediately

        if (x->warnings >= 2)
            x->status_VIP = 0      // remove VIP status

        Generate_Warning(x);
        return;
    }

    // proceed when account is in good standing

    new_balance = Update_Balance(x, total);

    x->number_of_orders += 1;

    // more than $100 purchase history or more than 5 orders
    if (x->total_money > 100) ||
       (x->number_of_orders > 5)) {

        x->status_VIP = 1;  // customer upgrade to VIP status
        x->discount = 5;    // 5% on total order purchase
```

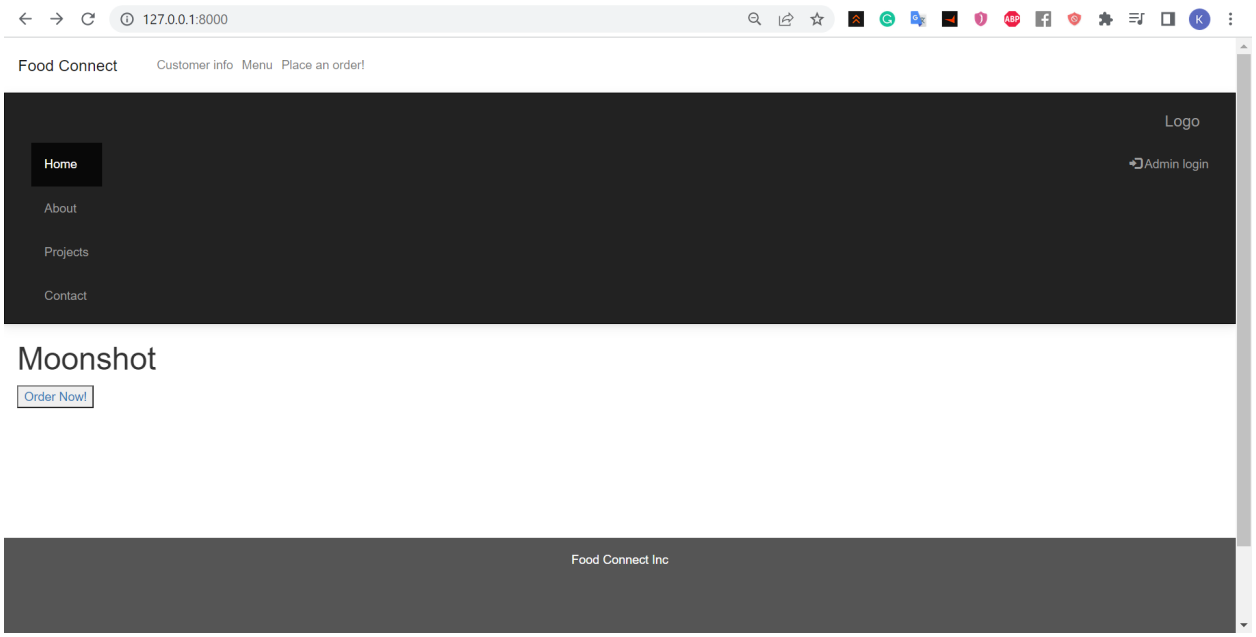
```
        // every 3 orders is free delivery
        // number_of_orders divisible by three
        if ((x->number_of_orders % 3) == 0)
             x->free_delivery = 1;   // free delivery for this order
    }

    /* tabulate new cost total  ***/
    if (x->discount > 0)
        new_total = total - (percent_discount * total);
    if (not free_delivery)
        new_total += 5;   // delivery charge

    // complete order processing
    Food_Order_Checkout(x, new_total, transaction_id);

    // append new purchase transaction to customer history
    Food_Order_History(x->history, transaction_id);

    return new_balance;
}


/* reset customer VIP status
 * input parameter:  pointer to data structure containing customer info
*/
int Generate_Warning(Customer *x) {
    // populate customer data fields
    Get_Customer_Data(x);

    x->warnings += 1;  // update warning data field

    if (x->warnings > 1) {
        x->status_VIP = 0;    // customer lost VIP status
        x->discount = 0;
        x->free_delivery = 0; // free delivery is not allowed
    }

    Display (warning_message);   // insufficient balance to order food
}

/* customer rate dish food items
 * input parameters:  pointer to data structure containing customer info
 * return value:  zero indicates cannot add rating
*/
int Rate_Food_Item(Customer *x, Food_Dish_Record *item) {
    // populate customer data fields
    Get_Customer_Data(x);

    // only registered customers that have purchased this item can rate it
    verify = Verify_Item_Purchased(x->history, item);

    if (verify == 0)
       return 0;    // cannot rate food not purchased

    Add_Ratings(x, item);

    return 1;
}
```

```
/* Delivery_Bidding
 * input parameters:
   bid_amount:  amount bidding from delivery person for the order transaction_id
   return value:  delivery person ID winning the bid (lowest amount)
*/
int Delivery_Bidding(float bid_amount, int delivery_person_id) {

    compare current bid_amount to list of bidding amount from other drivers (array)

    Manager approved winning_bid()

    driver = struct Delivery_Person->delivery_person_id

    return driver;
}


int Dispute_Complaints(Customer *x) {

    Manager review complaints x->feedback

    switch (complaints_category);
       case compliments
            x->compliments += 1
       case complaints
            x->complaints += 1

}

int Chef_Bonus(Chef *ch, int amount) {
    // populate customer data fields
    Get_Chef_Data(ch);

    ch->bonus = amount;   // assign bonus amount to data field
    ch->compliments = 0;  // reset value to zero
    return;
}


void Demote(Chef *ch, int salary) {
  // populate customer data fields
  Get_Chef_Data(ch);

  ch->salary = salary;  // update salary reduction
  ch->demotions += 1;   // update demotions count (more than 2, chef will be fired
  ch->complaints = 0;   // reset value to zero

  if (ch->demotions >= 2)
      Terminate_Chef(ch);

  return;
}

void Terminate_Chef(Chef *ch) {
  // populate customer data fields
  Get_Chef_Data(ch);

  ch->terminated = 1;   // marker indicates this chef is fired
```

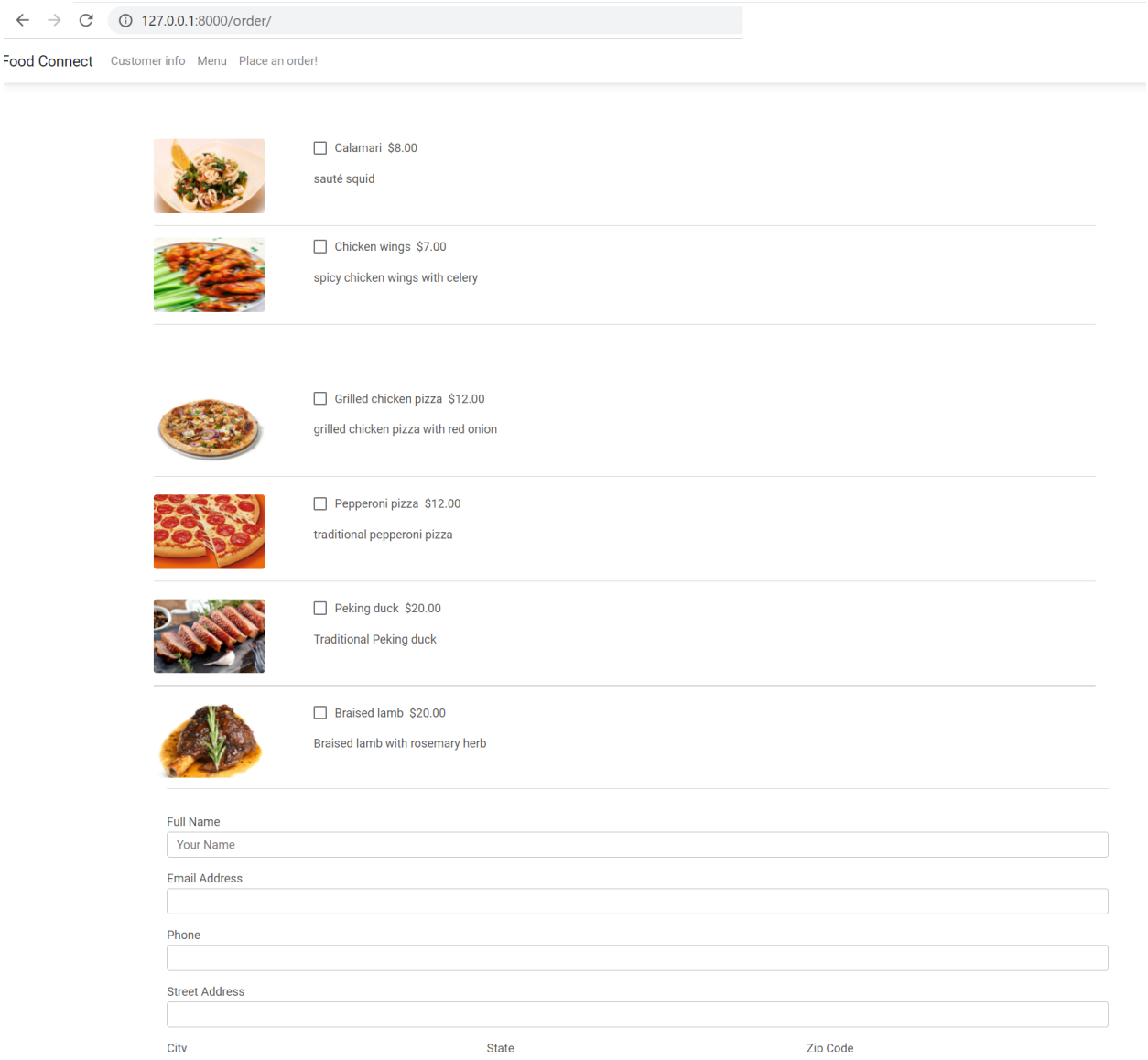
```
  ch->salary = 0;

}


/* helper functions */
/*
   phone number re-format

   (212) 345-6789   => 2123456789
   212-345-6789     => 2123456789
*/
int phone_reformat(int customer_id) {
    string s = get_customer_data(customer_id).phone;

    for (i=0; i < s.length(); i++);
         extract digits from text input representing phone number

    return phone_number;
}
```

5. Displays (GUI)

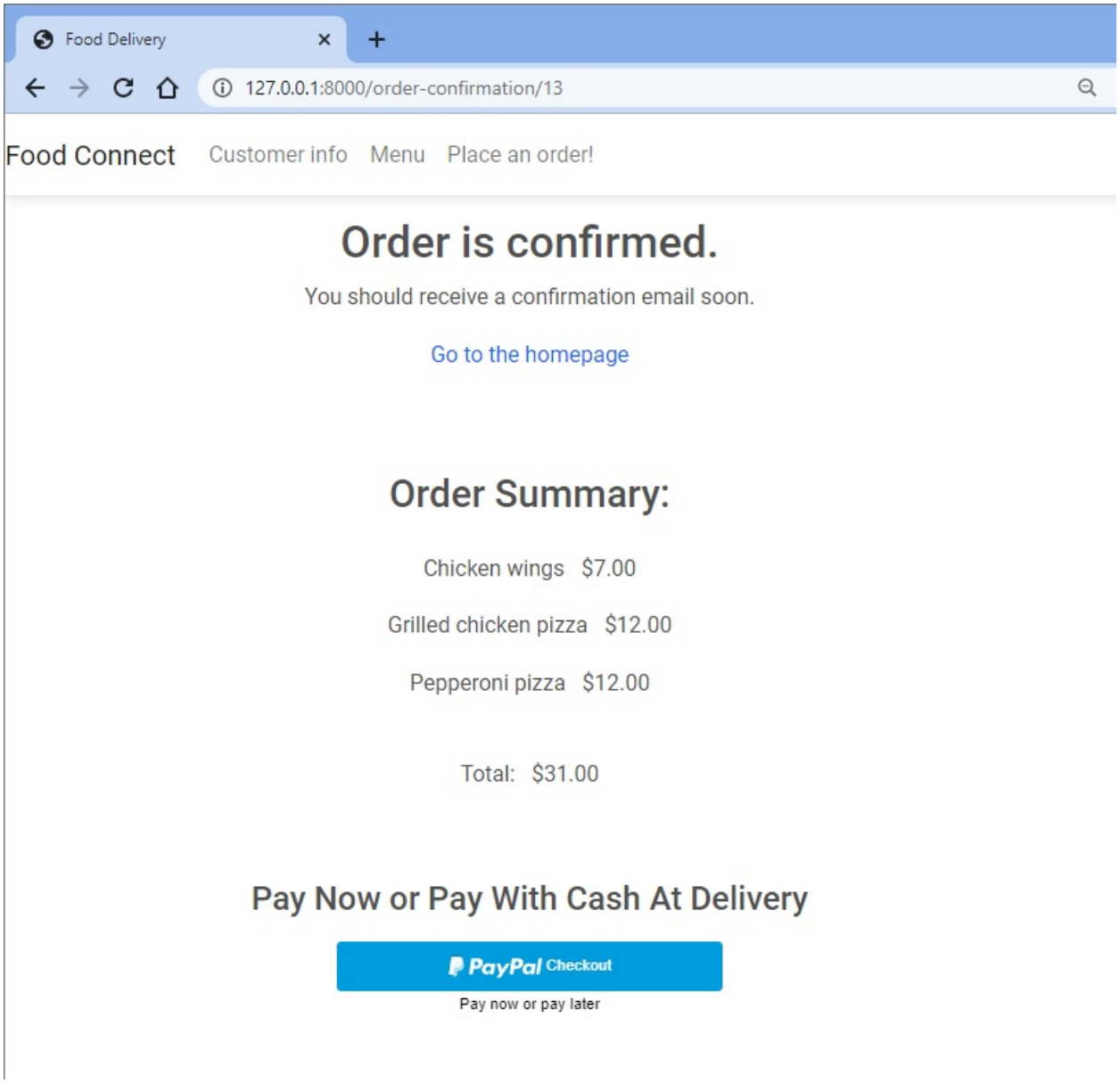
Front page



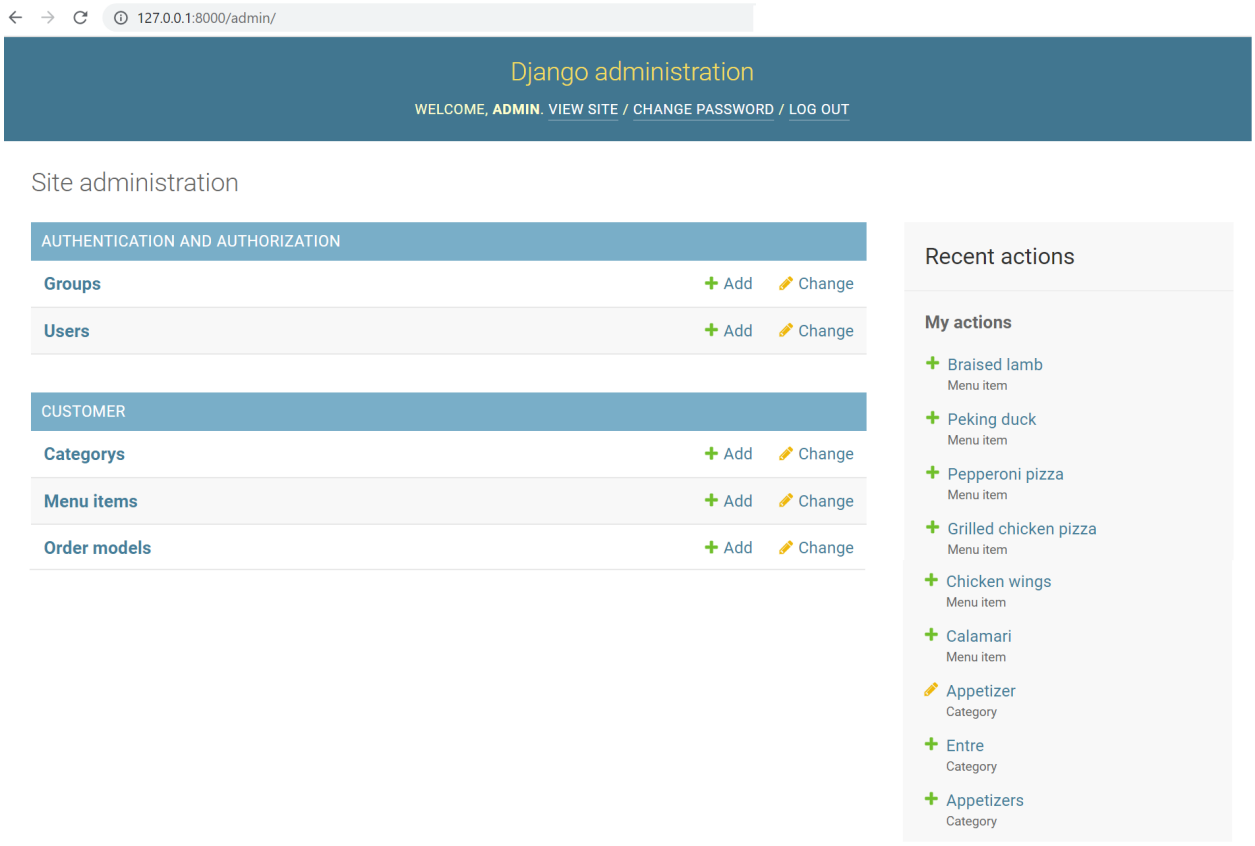
Order page


127.0.0.1:8000/order/
Food Connect
Customer info
Menu
Place an order!
Calamari $8.00
sauté squid
Chicken wings $7.00
spicy chicken wings with celery
Grilled chicken pizza $12.00
grilled chicken pizza with red onion
Pepperoni pizza $12.00
traditional pepperoni pizza
Peking duck $20.00
Traditional Peking duck
Braised lamb $20.00
Braised lamb with rosemary herb
Full Name
Your Name
Email Address
Phone
Street Address
City
State
Zip Code

Order confirmation



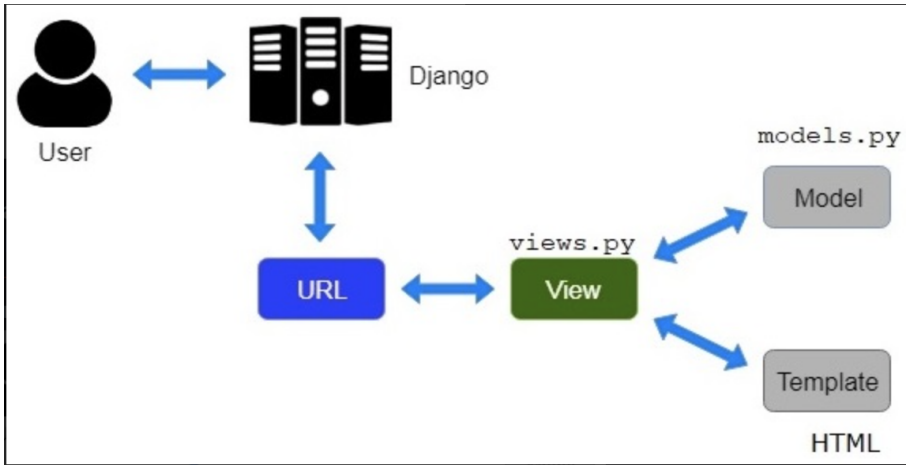
Admin page


127.0.0.1:8000/admin/
Django administration
WELCOME, ADMIN. VIEW SITE / CHANGE PASSWORD / LOG OUT
Site administration
AUTHENTICATION AND AUTHORIZATION
Groups + Add Change
Users + Add Change
CUSTOMER
Categorys + Add Change
Menu items + Add Change
Order models + Add Change
Recent actions
My actions
+ Braised lamb
Menu item
+ Peking duck
Menu item
+ Pepperoni pizza
Menu item
+ Grilled chicken pizza
Menu item
+ Chicken wings
Menu item
+ Calamari
Menu item
Appetizer
Category
+ Entre
Category
+ Appetizers
Category

6. Memorandum

Group meeting memo (April 17th)

- created GitHub repository for Moonshot project
- added team members to GitHub repository
- This memo is included in our github repository.



Group meeting memo (April 26th)

- Confirm django framework run correctly on localhost
- Expand food menu, add more categories
- Add quantity option to food menu item
- order page should have 2 lines for address

Ex: Address 1: 123 Program Street
Address 2: Apt 4

- Right now the webpage looks primitive we can work on fixing it to make it look nicer
- Fix the email issue (email confirmation when you order food)
- Work on software design report (due Thursday April 28th 11:59pm)
- Next meeting set for weekend
- This memo is included in our github repository.

7. GitHub

Project GitHub repository:

https://github.com/mle95/moonshot

Team members GitHub repositories:

Khanh Huang	https://github.com/141newyork
Yauheni Patapau	https://github.com/eugenepotapov2
Tea Nurcellari	https://github.com/Teanur
Vinuk Ranaweera	https://github.com/vinukranaweera
Minh Le	https://github.com/mle95