Los Tres Locos

Online Restaurant and Delivery Service App Design Report

Version 2.0

Revision History

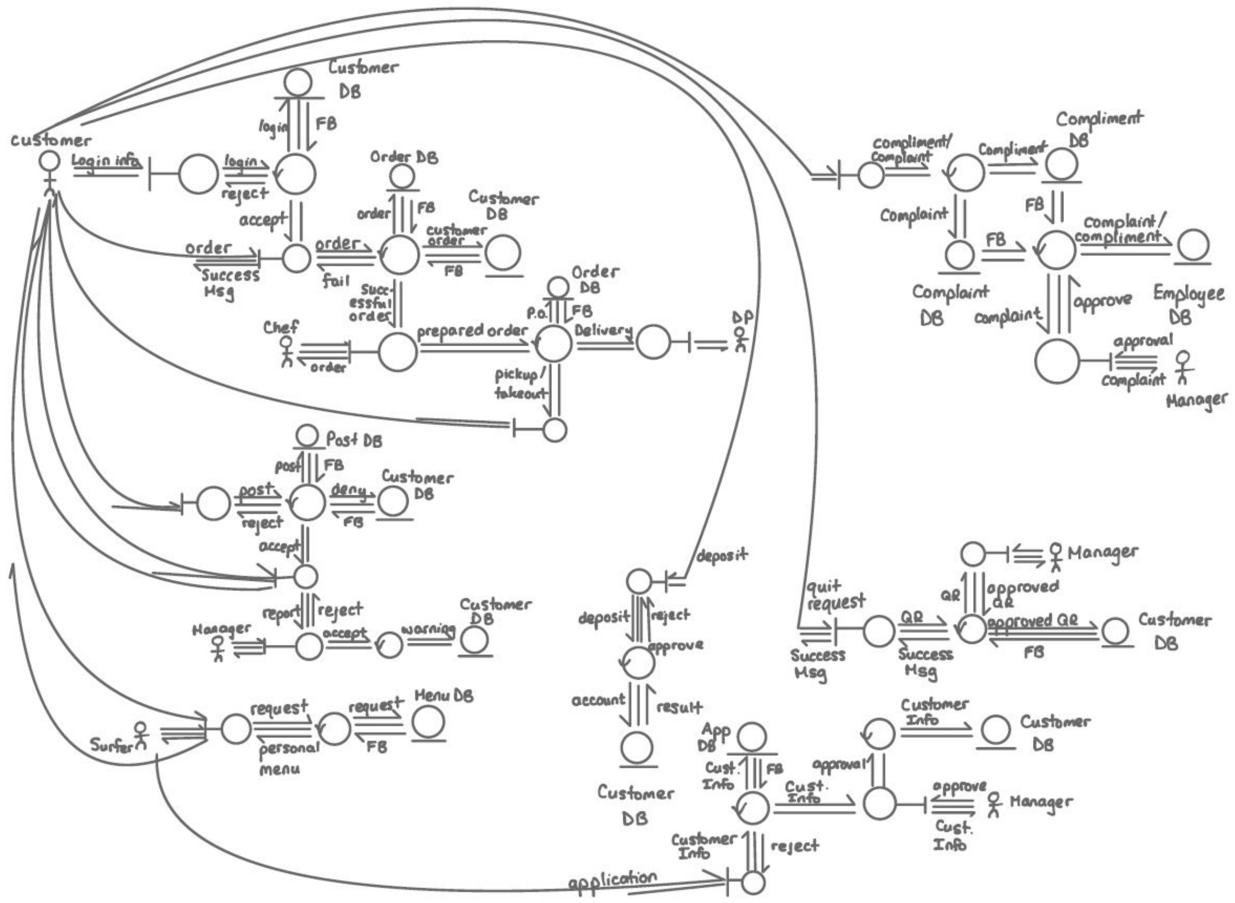
Date	Version	Description	Author
11/14/2020	2.0	Phase 2 Report	Nicholas De La Cruz, Kareem Ibrahim, Sufian Ilyas
10/18/2020	1.0	Phase 1 Report	Nicholas De La Cruz, Kareem Ibrahim, Sufian Ilyas

Table of Contents

Design Report

1. Introduction

1.1 Overall Collaboration Class Diagram



Any registered customer has the option to log into the application, view the menu, place an order, make a post to the discussion board, file a report, make a deposit, compliment or complain about a delivery person or chef, or make a request to quit.

When a customer logs in, the login information is checked against the customer database and is either accepted or denied based on whether the information is correct or not. After the customer is logged in, they have full access to the application.

A customer can place an order which is sent to the orders and customer databases. A chef can then see this order and start to prepare it. Once the order is prepared, the order database is updated, and the dish either goes to a delivery person for delivery or to the customer directly for pickup.

A customer can choose to post to the discussion board. The post is sent to a posts database and is also parsed for taboo words. If the post is denied for having too many taboo words, the customer receives a warning through the customer database and is notified. If the post is accepted, the user is sent a success message.

A customer can report another customer for misbehaving on the discussion board by reporting a specific post. The manager is notified of this report and can choose to accept or reject the report. If the report is accepted, the reported customer is given a warning through the customer database.

Both customers and surfers can view the menu. By choosing to see the menu, the user creates a request which queries the menu database. The user then can view their personalized menu.

A customer can make a deposit that is either accepted or rejected. If the deposit is rejected, the customer is notified immediately. If the deposit is accepted, the customer's balance is updated in the customer database, and then they are notified of the deposit's success.

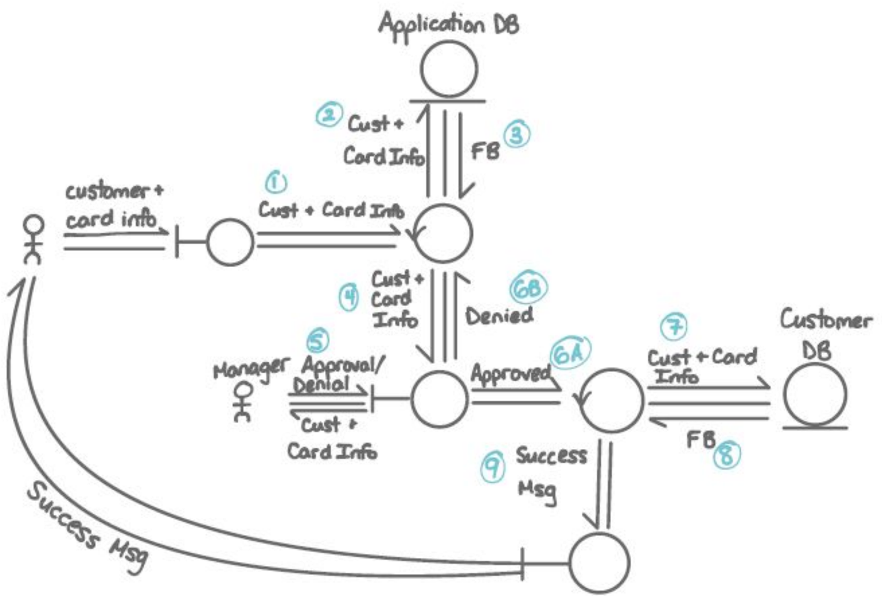
A customer can file a complaint or compliment which gets sent to a respective database for each event. The manager is notified of each complaint and compliment and can choose to approve or reject them. If either is approved, the compliments and complaints counter in the employee database is updated for the given employee.

A customer can request to quit. This request is sent to the manager. When the request is approved by the manager, the customer's account balance is cleared and they are removed from the customer database. An email is then sent to the customer to notify them that they have been removed from the database and are no longer a registered customer.

A surfer can apply to become a registered customer by inputting their personal and card information. This information goes to an application database. The data is then sent to the manager who can approve the application. Once approved, the surfer's information is added to the customer database, and the surfer is now a registered customer.

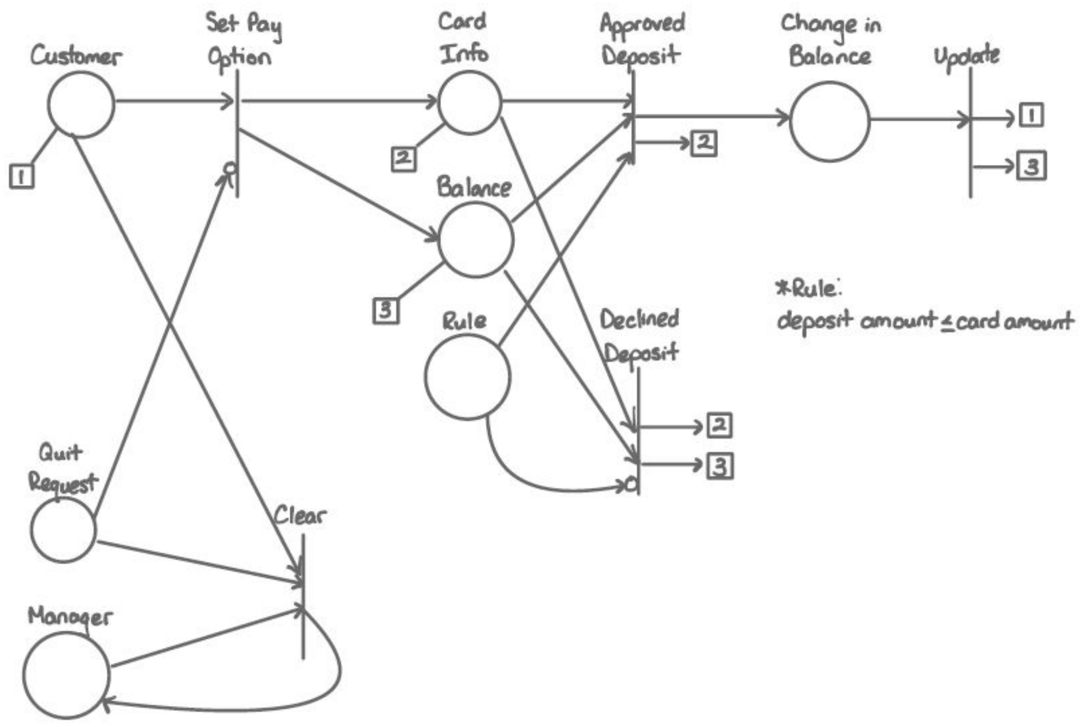
2. Use Cases

2.1 Registration Collaboration Class Diagram



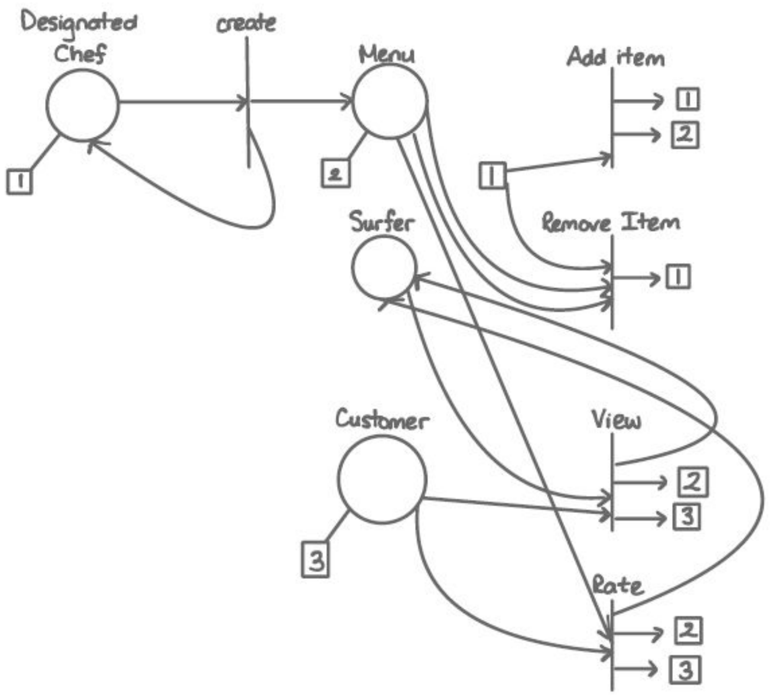
When a surfer applies to become a registered customer, they input their personal and card information. This information is sent to an application database, and the data is then given to the manager. The manager has the option to approve or deny the surfer's application. If the application is denied, the surfer is notified via email. If the application is approved, the surfer's information is added to the customer database, and a success message is then emailed to the new customer.

2.2 Deposit Petri Net



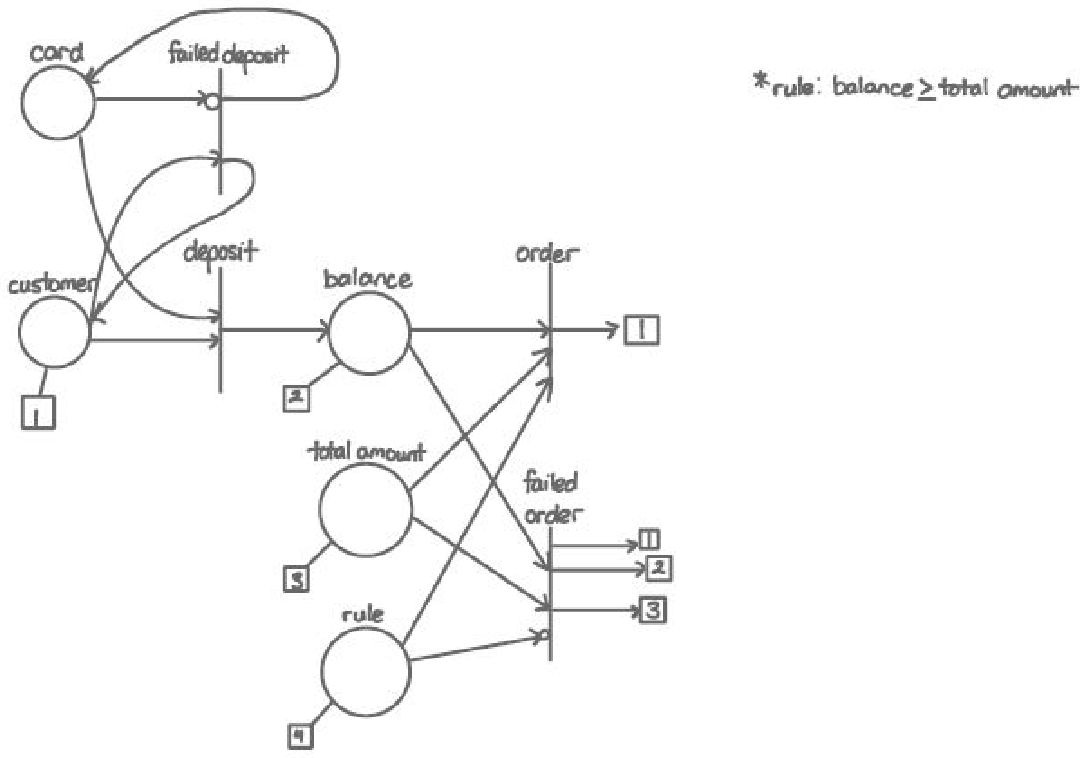
A customer has the option to either set their payment option or clear their account when they make a quit request. If the customer makes a quit request, the manager clears the account and the customer is deregistered. Otherwise, the customer chooses their pay option (credit card, debit card, or cryptocurrency). After setting the payment option, the deposit is either approved or denied based on whether the deposit amount is less than or equal to the card amount or not. If it is not, the deposit is declined. If it is, then the deposit is approved, and the change in balance updates the customer's balance.

2.3 Menu Petri Net



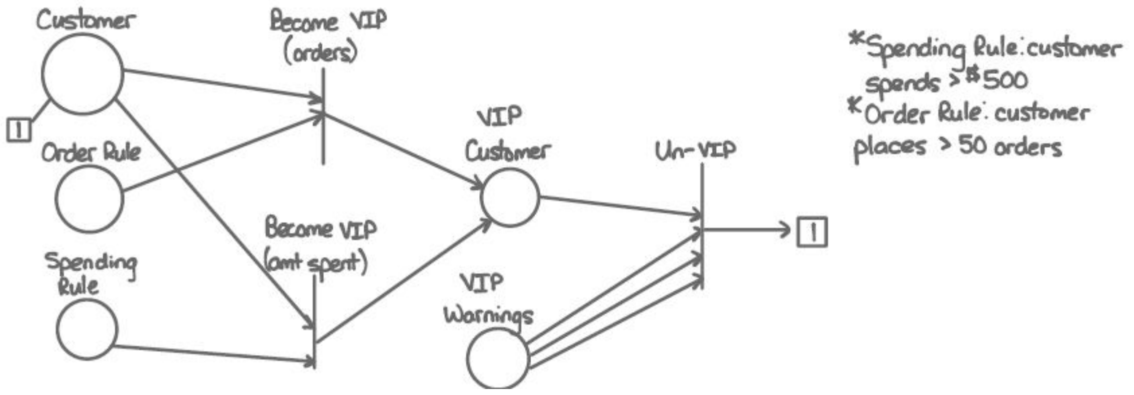
A designated chef first creates the menu. After the menu is created, one of the designated chefs can add or remove a dish from the menu. If they add an item, a token is added to the menu. If they remove an item, a token is removed from the menu. Both surfers and customers can view the menu; however, only customers can rate the menu's dishes.

2.4 Order Petri Net



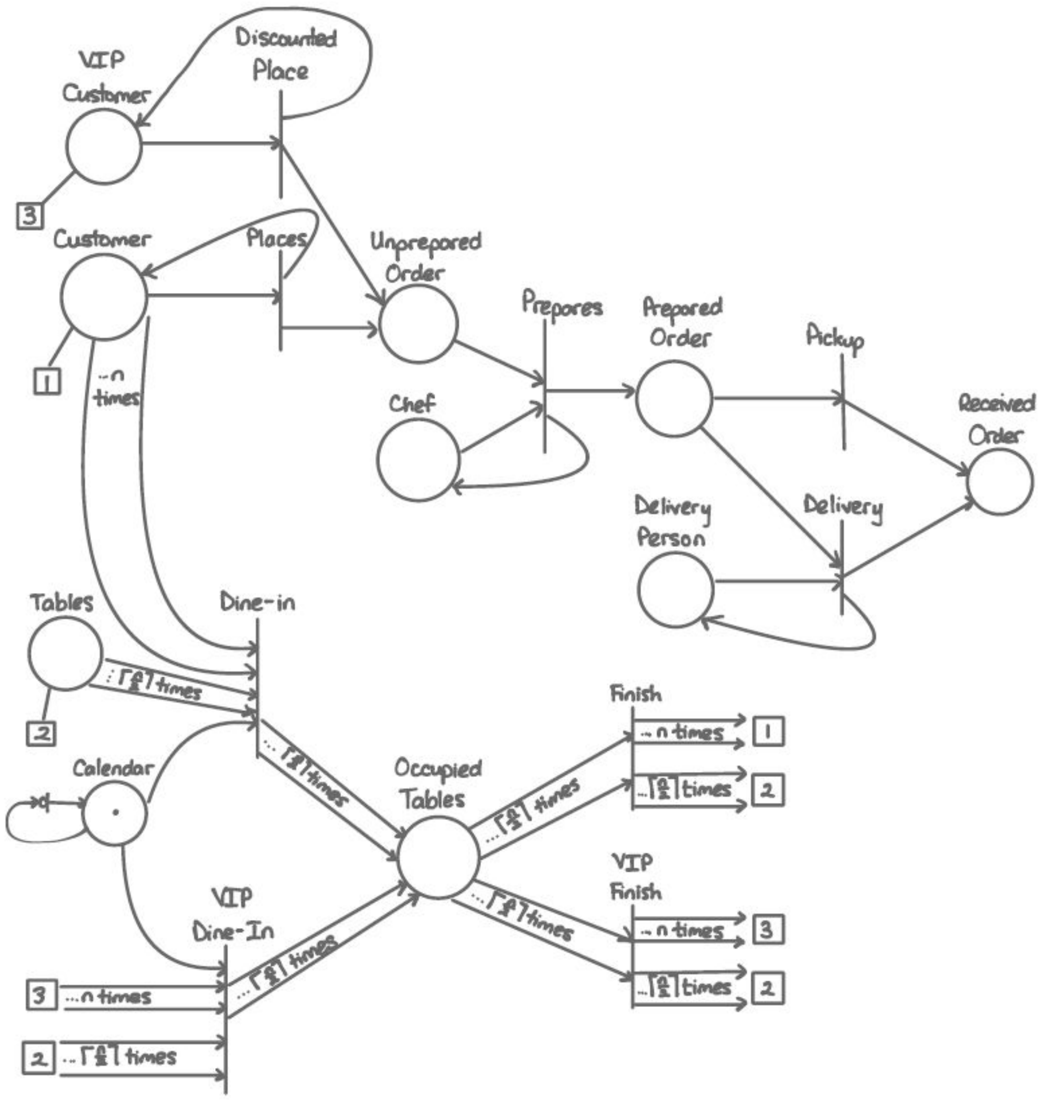
In order to place an order, a customer (regular or VIP) must have made a deposit. If the card is rejected, the deposit fails, and the customer is able to try and make another deposit. Otherwise, the deposit is successful and their balance increases. If the order's amount is greater than the customer's balance, then the order fails. The customer can then make another deposit to increase their balance. If the customer's balance is greater than or equal to the amount of the order, the order is a success and they are able to order again.

2.5 VIP Petri Net



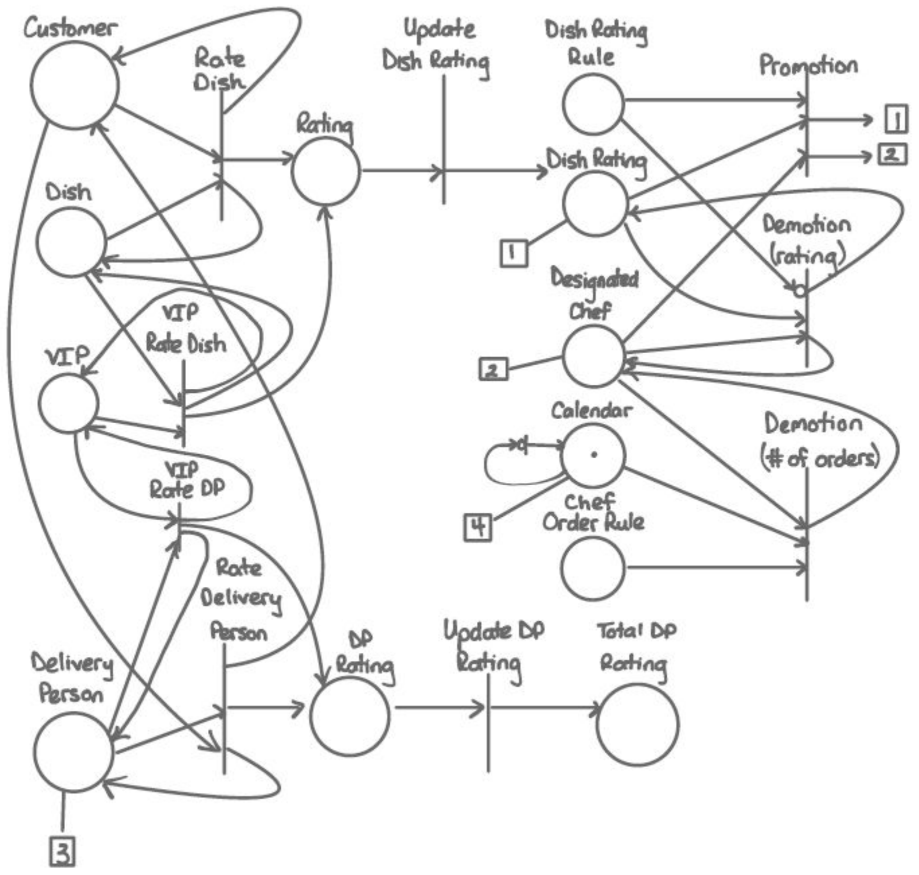
A customer can become a VIP customer by either placing 50 orders or by spending more than $500 at the restaurant. If a VIP customer receives three warnings, then their status is returned to a regular customer.

2.6 Service Petri Net



Customers can choose to place an order or to dine in through the application. VIP customers can place a discounted order or choose to dine in at the restaurant. When a regular or VIP customer places an order, an unprepared order is created. Chefs then prepare the order to either be picked up by a customer or delivered to a customer by a delivery person. If the customer chooses to dine in, they have a party of n customers. Since the restaurant is designed to only have tables with two seats each, n customers occupy ceil(n/2) tables. When the customers (either VIP or regular) finish, the ceil(n/2) occupied tables are freed up.

2.7 Rating Petri Net

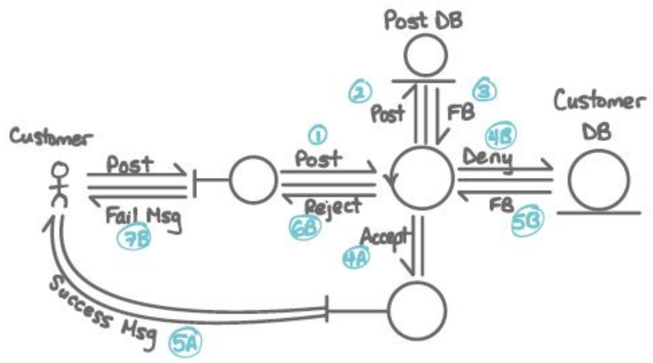


*Dish rating rule: Consistently high ratings ⇒ promotion for designated chef

*Chef order rule: A dish which isn't ordered for 3 days ⇒ demotion for designated chef

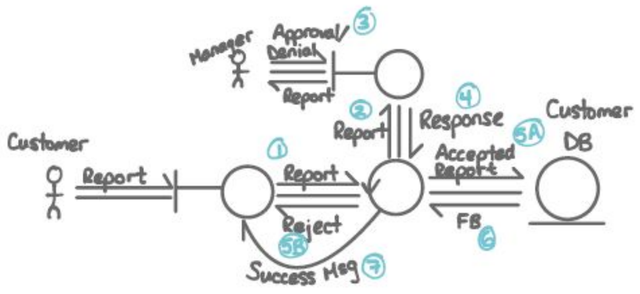
A regular customer can choose to rate a dish or a delivery person. This adds to the rating for a given dish or delivery person respectively. VIP customers can also rate a dish or delivery person, but their rating counts twice. After the rating is given to the dish or delivery person, the respective rating is updated. If a dish's overall rating is consistently high, then the designated chef who designed the dish is promoted. However, if a dish's overall rating is consistently low, then the designated chef who designed the dish is demoted. If a dish has not been ordered for three days, the designated chef who designed the dish is demoted.

2.8 Discussion Board Post Collaboration Class Diagram



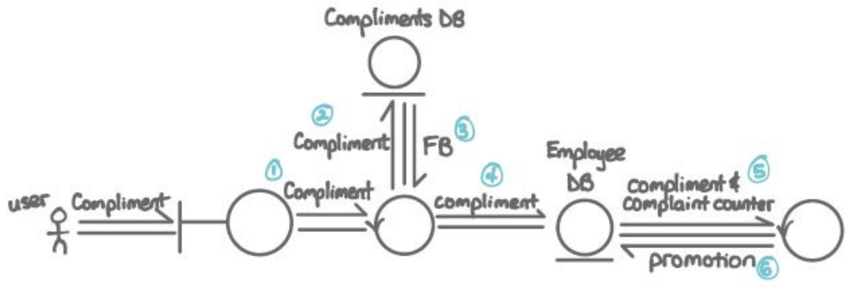
A customer can make a post to the discussion board. The post is sent to the posts database and is checked for the number of taboo words inside it. If the post contains more than three taboo words, the post is denied and the customer is given a warning in the customer database. They are then notified that their post has been denied. On the other hand, if the post is accepted, the customer is notified that their post has been posted to the discussion board.

2.9 Reporting Collaboration Class Diagram



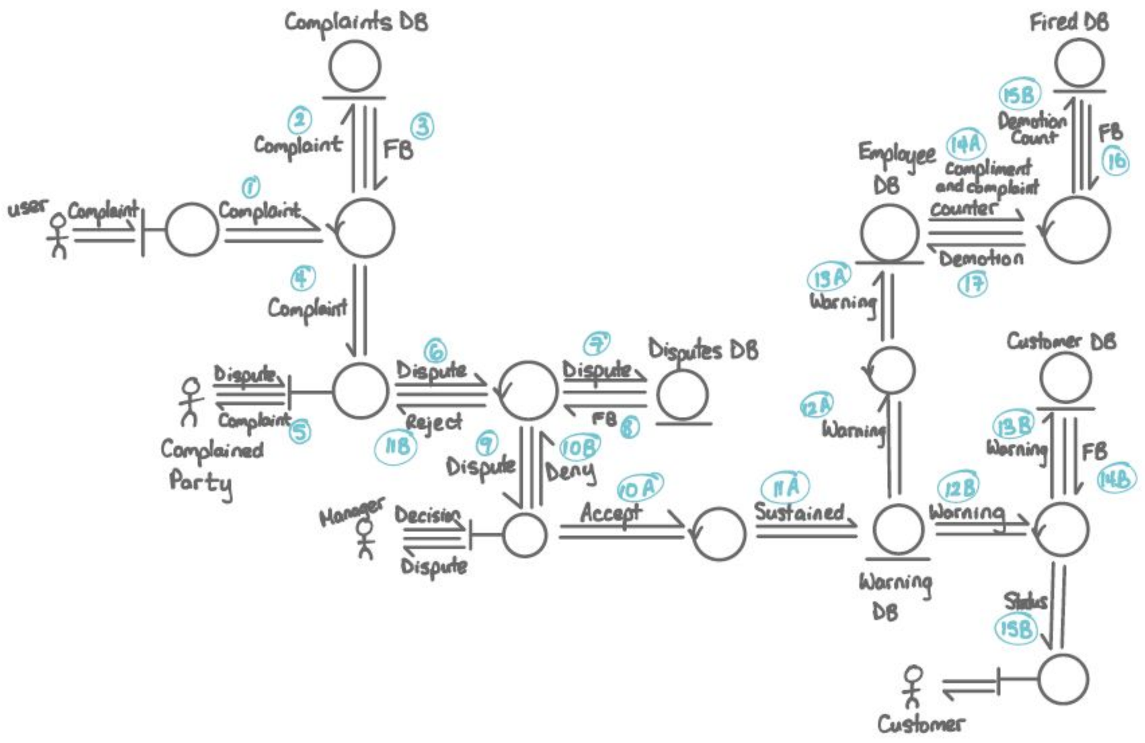
A customer can file a report against another customer for misbehaving on the discussion board. The report is then sent to the manager who can accept or deny the report. If the report is rejected, the customer is notified. If the report is accepted, the reported party is given a warning in the customer database. A success message is then sent to the reporter.

2.10 Compliment Collaboration Class Diagram



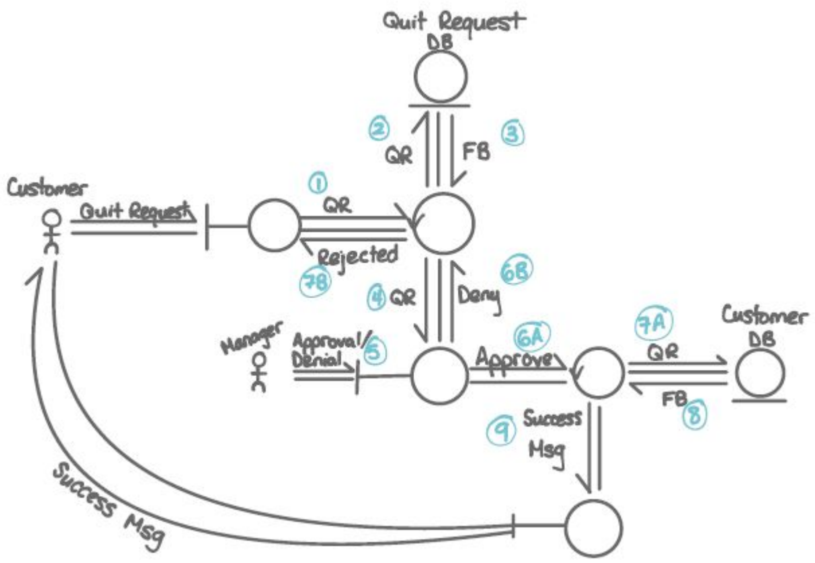
A user (registered customer or delivery person) can file a compliment which gets sent to a compliments database. The compliment then goes to the employee database and adds a compliment to the complaint and compliment counter. If the counter reaches three compliments, the employee is promoted and their salary is updated in the employee database.

2.11 Complaint Collaboration Class Diagram



A user (registered customer or delivery person) can file a complaint which gets sent to a complaints database. A notification is then sent to the complained party who then has the option to dispute the complaint. If the complained party (registered customer, delivery person, or chef) chooses to dispute the complaint, the dispute is sent to a disputes database, and the manager is then notified of the dispute. After being notified, the manager makes a decision for the dispute. If the dispute is rejected, the complained party is notified. On the other hand, if the dispute is accepted, the complaint then becomes a warning for the complainer. Depending on whether the complainer was a customer or employee, the warning is sent to the customer database or the employee database respectively. If the warning is for a customer, a warning is added to a customer's warning counter. The customer is then notified that their complaint was disputed. If the warning is for an employee, a warning is added to that employee's compliment and complaint counter. The system then checks if the employee has three complaints. If this is the case, they are demoted. The system then checks if the employee has been demoted twice. If this is the case, they are added to a fired database and are subsequently removed from the employee database.

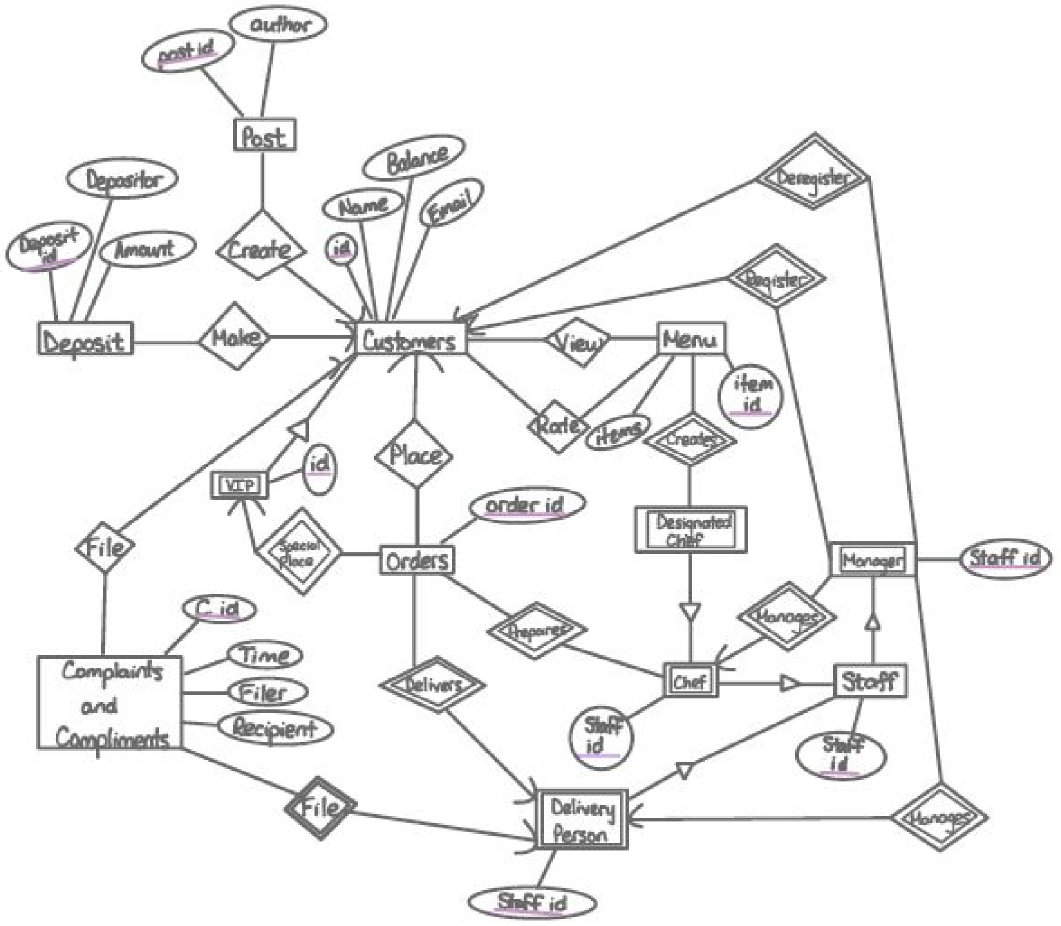
2.12 Quitting Collaboration Class Diagram



A customer is able to submit a quit request in the web app. The quit request is then sent to a database containing quit requests and is then sent to the manager who can approve or deny the request. If the request is denied, the user is notified that their request has been rejected. If the request is approved, the customer is removed from the customer database, and then a success message is sent to the customer.

3. Overall E-R Diagram

3.1 E-R Diagram



Customers are an entity with an ID, name, balance, and email as their attributes with their ID as their primary key. Customers are able to make a post to the discussion board or make a deposit to update their balance. Each post and deposit belongs to one and only one customer. Customers can also view the menu, rate the menu, place orders, and file complaints or compliments. Customers place many orders but an order belongs to one customer. Similarly, they can file many complaints or compliments but each complaint or compliment belongs to one customer. A VIP is a special type of customer who still has their ID as its primary key. VIPs are able to place a special order.

Staff is an entity with a primary key of a Staff ID. The manager, chefs, and delivery people are all staff, and their primary keys are their Staff ID. Since their primary keys are foreign keys, the manager, chef, and delivery people are weak entities. Consequently, any relation connecting them to another entity is a weak relation.

There is only one manager who manages chefs and delivery people. The manager also registers and deregisters customers. Delivery people can deliver many orders, but each order is delivered by one and only one delivery person. Additionally, delivery people can file many complaints or compliments, and each complaint and compliment belongs to one and only one delivery person. A chef prepares many orders, and an order can also be prepared by many chefs. This is because order can consist of multiple dishes, and each dish can be prepared by a different chef. A designated chef is a chef who specifically creates the menu by adding or removing dishes.

4. Detailed Design

4.1 Pseudocode

```
''' Registration '''
# pre cond: application object sent in, must be have all fields filled out
# post cond: applicant awaits approval
def RegisterApplicant(application):
   appDB = GetAppDB()
   appDB.add(application)

# pre cond: manager must have the proper credentials of a manger
# post cond: send back all pending applications
def getApplications(manager):
   appDB = GetAppDB()
   managerCredentials = manager.getCredentials()
   return appDB.retrieveApplications(managerCredentials)

# pre cond: manager must have proper credentials, application must be a valid
application
# post cond: add to customer DB, remove application from DB, notify applicant
def approveApplication(manager, application):
   appDB = GetAppDB()
   customerDB = GetCustomerDB()

   if deposit(application.payment_method):
       appDB.remove(application, managerCredentials)
       customerDB.add(application)
       email(application.email, 'You got accepted')
   else:
       rejectApplication(manager, application)
```

```
# pre cond: manager must have proper credentials, application must be a valid
application
# post cond: remove application from DB, notify applicant
def rejectApplication(manager, application):
   appDB = GetAppDB()
   managerCredentials = manager.getCredentials()
   appDB.remove(application, managerCredentials)
   email(application.email, 'You got rejected')

''' Quitting '''
# pre cond: customer must be a customer
# post cond: customer is added to quit DB to be approved by manager
def quitRequest(customer):
   quitDB = GetQuitDB()
   quitDB.add(customer)

# pre cond: manager is the manager
# post cond: all customer quit requests are returned
def getQuitRequests(manager):
   quitDB = GetQuitDB()
   managerCredentials = manager.getCredentials()
   return quitDB.retrieveQuitRequests(managerCredentials)

# pre cond: manager is a manager, customer is a customer with a quit request
# post cond: the customer is deregistered and removed from the DB
def acceptQuitRequest(manager, customer):
   quitDB = GetQuitDB()
   customerDB = GetCustomerDB()

   managerCredentials = manager.getCredentials()
   clearDeposit(customer, managerCredentials)
   customerDB.remove(customer)
   quitDB.remove(customer)

   email(customer.email, 'Bye :(')

# pre cond: manager is a manager, customer is a customer with a quit request
# post cond: customer is still a customer, their quit request is removed
def denyQuitRequest(manager, customer):
   quitDB - GetQuitDB()
   managerCredentials = manager.getCredentials()

   quitDB.remove(customer)
   email(customer.email, 'Nah')

''' Menu '''
# pre cond: desigChef is a designated chef, itemInfo has the name of the menu item,
description, and picture
# post cond: itemInfo is added to the menu DB
def addMenuItem(desigChef, itemInfo):
   menuItem = createMenuItem(desigChef.id,
                             itemInfo.name,
                             itemInfo.desc,
                             itemInfo.picture)
   GetMenuDB.add(menuItem)

# pre cond: desigChef is a designated chef, itemID is a valid menu item
# post cond: itemID is removed from menu DB
def removeMenuItem(desigChef, itemID):
   GetMenuDB.remove(desigChef, itemID)

# Surfers
# pre cond: customer is not logged in i.e. surfer, menu exists
# post cond: returns a menu list with default featured items
def getMenu():
   menuDB = GetMenuDB()
   # Sets the default menu items to personalizedMenu
   personalizedMenu = Menu(menuDB)

   topThreeRatedDishes = menuDB.getTopRatedDishes()
   topThreeOrderedDishes = menuDB.getTopOrdereddDishes()
   personalizedMenu.setFeaturedDishes(topThreeRatedDishes,
                                      topThreeOrderedDishes)

   return personalizedMenu
```

```
# Customers
# pre cond: customer is logged in
# post cond: return a menu that is personalized to the customers orders
def getMenu(customer):
   ordersDB = GetOrdersDB()
   allCustomerOrders = ordersDB.getAllOrders(customer)
   # only allows unique dishes
   distinctOrders = set(allCustomerOrders.allCustomerDishes)

   if distinctOrders.length < 3:
       return getMenu()

   menuDB = GetMenuDB()
   personalizedMenu = Menu(menuDB)

   topThreeDishes = set(allCustomerOrders.sortByRatings().allCustomerDishes())
   personalizedMenu.setFeaturedDishes(topThreeDishes)

   if customer.isVIP:
       personalizedMenu.setSpecials(menuDB.getSpecials())

   return personalizedMenu

''' Discussion Board '''
# pre cond: customer is a customer, postInfo has the subject and body of the post
# post cond: if no more than 3 taboo words, add the post to the discussion baord
def post(customer, postInfo):
   # Return a list of all taboo words
   tabooWords = scanTabooWords(postInfo)
   if tabooWords is not empty:
       customer.incrementWarnings()
       GetCustomerDB.update(customer)
       punishCustomer(customer)
       if tabooWords.length <= 3:
           replaceAllTabooWords(postInfo, '***')
       else:
           # Do not add post if too many bad words
           return

   postItem = createPostItem(customer,
                             postInfo.subject,
                             postInfo.body)
   GetPostDB().add(postItem)

# pre cond: customer is a customer, commentInfo has a body
# post cond: if no more than 3 taboo words, comment is posted to the post
def comment(customer, post, commentInfo):
   tabooWords = scanTabooWords(commentInfo)
   if tabooWords is not empty:
       customer.incrementWarnings()
       GetCustomerDB.update(customer)
       punishCustomer(customer)
       if tabooWords.length <= 3:
           replaceAllTabooWords(commentInfo, '***')
       else:
           # Do not add post if too many bad words
           return

   commentItem = createCommentItem(customer,
                             commentInfo.body)
   post.addComment(commentItem)
   GetPostDB().update(post)

# pre cond: customer is a customer with a warning
# post cond: if customer is VIP with >= 2 warnings, deregister to regular customer
#            if customer has >= 3 warnings, deregister customer to a surfer
def punishCustomer(customer):
   customerDB = GetCustomerDB()
   totalWarnings = customer.getWarnings()
   if customer.isVIP and totalWarnings >= 2:
       customer.unregisterVIP()
       customerDB.update(customer)
   else if totalWarnings >= 3:
       clearDeposit(customer)
       GetCustomerDB().remove(customer)
       customerDB.update(customer)



''' Reporting '''
# pre cond: customer is a customer, post is a valid post or comment on discussion board
# post cond: add report to reports database
def reportPost(customer, post):
   post.setSnitch(customer)
   GetReportsDB.add(post)

# pre cond: manager is a manager
# post cond: return all active reports to the manager
def getReports(manager):
   managerCredentials = manager.getCredentials()
   return GetReportsDB().getAllReports(managerCredentials)

# pre cond: manager is a manager, report is an active report, accepted is a boolean if
the complainer's report is valid
# post cond: if accepted, complainee gets a warning, if snitch is VIP, then it is
counted twice
#             else, snitch gets a warning
def managerAdjudicateReport(manager, report, accepted):
   reportsDB = GetReportsDB()
   customerDB = GetCustomerDB
   managerCredentials = manager.getCredentials()

   if accepted:
       if report.snitch.isVIP():
           report.poster.incrementWarnings()
           report.poster.incrementWarnings()
       else:
           report.poster.incrementWarnings()

       customerDB.update(report.poster)
       punishCustomer(report.poster)
   else:
       report.snitch.incrementWarnings()
       customerDB.update(report.snitch)
       punishCustomer(report.snitch)

   reportsDB.remove(report)

''' Complaints '''
# dp = Delivery Person
# pre cond: complaint must have a complainee, snitch must be a customer or dp
#           if snitch is a customer, they must have had either dp or chef as the
complainee
#           if snitch is a dp, they must have had the customer as the complainee
# post cond: a complaint for the complainee is added to the complaints DB
def complain(snitch, complaint):
   complaintItem = createComplaint(snitch,
                                   complaint.complainee,
                                   complaint.reason)
   GetComplaintsDB.add(complaintItem)

# pre cond: manager is a manager
# post cond: return all active complaints to the manager
def getAllComplaints(manager):
   managerCredentials = manager.getCredentials()
   return GetComplaintsDB.getComplaints()

# pre cond: compalinee must have complaint as active against them, reason must be a
reason to dispute
# post cond: a dispute is added to complaint
def disputeComplaint(complainee, complaint, reason):
   disputeItem = createDispute(complainee,
                               reason)
   GetComplaintsDB().addDispute(complaint, disputeItem)

# pre cond: manager is a manager, complaint must be an active complaint, accepted is a
boolean if the complainer's complaint is valid
# post cond: if accepted, complainee gets a warning, if snitch is VIP, then it is
counted twice
#            else, snitch gets a warning
#            person who gets the complaint gets punished
def managerAdjudicateComplaint(manager, complaint, accepted):
```

Online Restaurant and Delivery Service Application	Version: 2.0
Design Report	Date: 11/14/2020
Phase 2 Report	

```
   complaintsDB = GetComplaintsDB()
   managerCredentials = manager.getCredentials()

   complaintsDB.remove(complaint)
   if accepted:
       if complaint.snitch.TYPE is Customer and complaint.snitch.isVIP():
           complaint.complainee.incrementWarnings()
           complaint.complainee.incrementWarnings()
       else:
           complaint.complainee.incrementWarnings()
       # getTypeDB() gets the complainee's specific database i.e. chef, delivery
person, or customer
       getTypeDB(complaint.complainee.TYPE).update(complaint.complainee)
       if getTypeDB(complaint.complainee.TYPE) is Staff:
           punishStaff(complaint.complainee)
       else:
           punishCustomer(complaint.complainee)
   else:
       complaint.snitch.incrementWarnings()
       getTypeDB(complaint.snitch.TYPE).update(complaint.snitch)
       if getTypeDB(complaint.snitch.TYPE) is Staff:
           punishStaff(complaint.snitch)
       else:
           punishCustomer(complaint.snitch)

# pre cond: staffMember must be a dp or chef
# post cond: if staffMember had >= 3 warnings, they get demoted, decreased salary,
warnings set to 0
#             if staffMember had >= 2 demotions, they are fired and removed from the DB
def punishStaff(staffMember):
   staffDB = getTypeDB(staffMember.TYPE)
   totalWarnings = staffMember.getWarnings()

   if totalWarnings >= 3:
       staffMember.incrementDemotions()
       staffMember.decreaseSalary()
       # If over 3 complaints, set back to 0 complaints for another demotion
       staffMember.setWarningsZero()
```

```
        staffDB.update(staffMember)
    else if staffMember.getDemotions() >= 2:
        # This fires them
        GetFiredDB().add(staffMember)
        staffDB.remove(staffMember)

''' Rating '''
# pre cond: customer is a customer, menuItem is in the menu DB, rating is between 1-5
# post cond: rating is added to menuItem, if customer is VIP it's counted twice
#            check if the chef who added the menu item needs to be promoted/demoted
def rate(customer, menuItem, rating):
    if customer.isVIP():
        menuItem.incrementTotalRatings()
        menuItem.incrementTotalRatings()
        menuItem.addRating(2 * rating)
    else:
        menuItem.incrementTotalRatings()
        menuItem.addRating(rating)

    GetMenuDB().update(menuItem)
    checkChefMenuRatings(menuItem)

# pre cond: menuItem is in the menu DB
# post cond: if the chef has consistently high ratings (an average rating of >= 4 with
10 or more ratings), give promotion
#            if the chef has consistently low ratings (an average rating of <= 2 with
10 or more ratings), give demotion and punish
def checkChefMenuRatings(menuItem):
    chefDB = GetChefDB()
    desigChef = menuItem.assignedBy()
    totalRatings = menuItem.getTotalRatings()
    averageRating = menuItem.averageRating()
    if totalRatings >= 10 and averageRating >= 4:
        desigChef.incrementPromotions()
        desigChef.increaseSalary()
        chefDB.update(desigChef)
    else if totalRatings >= 10 and averageRating <= 2:
        desigChef.incrementDemotions()
```

```
       desigChef.decreaseSalary()
       chefDB.update(desigChef)
       punishStaff(desigChef)

''' Compliment '''
# pre cond: complimenter must be a customer or dp, compliment has a reason and a
complimentee who is a dp or customer
#           complimenter and complimentee cannot be both dp and dp or both customer and
customer
# post cond: if complimentee has warnings, remove them first before adding an official
compliment
#            if complimentee is Staff with no warnings, give them an official
compliment and reward
#            if the complimenter is a VIP, their compliment is counted twice
def giveCompliment(complimenter, compliment):
   complimentee = compliment.complimentee
   complimentItem = createCompliment(complimenter,
                                     complimentee
                                     compliment.reason)
   personDB = getTypeDB(complimentee.TYPE)

   if complimenter.TYPE is Customer and complimenter.isVIP():
       if complimentee.getWarnings() >= 2:
           complimentee.decrementWarnings()
           complimentee.decrementWarnings()
       else if complimentee.getWarnings() == 1:
           complimentee.decrementWarnings()
           complimentee.incrementCompliments()
           reward(complimentee)
       else:
           complimentee.incrementCompliments()
           complimentee.incrementCompliments()
           reward(complimentee)
   else:
       if complimentee.getWarnings() > 0:
           complimentee.decrementWarnings()
       else if complimentee is Staff:
           complimentee.incrementCompliments()
```

```
            reward(complimentee)

# pre cond: staffMember is a dp or chef with a newly added compliment
# post cond: if staffMember has >= 3 official compliments, give promotion, increase
their salary, and reset their compliments to 0
def reward(staffMember):
    staffDB = getTypeDB(staffMember.TYPE)

    if staffMember.getCompliments() >= 3:
        staffMember.incrementPromotions()
        staffMember.increaseSalary()
        staffMember.setComplimentsZero()
        staffDB.update(staffMember)

''' Deposit '''
# pre cond: customer is a customer, payment type is either money or cryptocurrency,
depositAmount is the total deposit in USD
# post cond: if paymentType is money, just check if their card has the funds to deposit
and return whether or not it does
#            if paymentType is crypto, check if they have enough crypto (when converted
to USD) to pay and return whether or not it does
def deposit(customer, paymentType, depositAmount):
    customerDB = GetCustomerDB()
    if paymentType == Money:
        if totalAmount(customer.card) >= depositAmount:
            withdraw(customer.card, depositAmount)
            customer.addMoney(depositAmount)
            customerDB.update(customer)
            return 'Deposit approved.'
        else:
            return 'Deposit denied.'
    else:
        # Crypto Payment
        if convertToUSD(totalCryptoAmount(customer.card)) >= depositAmount:
            # Crypto network is Ethereum rinkeby
            withdrawFromCryptoNetwork(customer.card, depositAmount)
            customer.addMoney(depositAmount)
            customerDB.update(customer)
            return 'Deposit approved.'
        else:
            return 'Deposit denied.'

''' Orders '''
# pre cond: customer is a customer with a deposit, menuItems is a list of of items from
the menu DB
# post cond: remove money from customers account, if VIP then apply a 10% discount
#             return True if the user has the proper deposit amount and false otherwise
def pay(customer, menuItems):
    # Get total price of all the orders
    totalPrice = menuItems.reduce(total, item => total + item.price)
    if customer.money() >= totalPrice:
        if customer.isVIP:
            customer.removeMoney(totalPrice * 0.90)
        else:
            customer.removeMoney(totalPrice)
        return True
    else:
        return False

# pre cond: time is a time to reserve tables, partyMembers are the total people in the
reservation
# post cond: each table in the restaurant has two seats, if there exist enough tables
for each member in the party at time, return True
#             otherwise there are not enough tables at time for the party, return False
def dineInTimeConflict(time, partyMembers):
    tables = GetTablesDB()
    tablesNeeded = ceil(totalSeats / 2)
    tablesOccupied = tables.tablesOccupiedAt(time)
    totalTablesInRestaurant = tables.getTotalTables()
    if tablesNeeded < totalTablesInRestaurant - tablesOccupied:
        tables.occupyMoreTables(time, tablesNeeded)
        return False
    else:
        return True
```

```
# pre cond: customer is a customer with a deposit, menuItems is a list of of items from
the menu DB, time is a time to reserve tables, partyMembers are the total people in the
reservation
# post cond: a success condition reserves the time for the party and a chef is
assigned, checks if to make a customer a VIP
#             a fail condition tells the customer why it failed
def orderDineIn(customer, menuItems, time, partyMembers):
   if dineInTimeConflict(time, partyMembers):
       return "Time conflicts. Please choose another."
   else if pay(customer, menuItems):
       reserveTime(time, partyMembers)
       order = createOrderDineIn(customer,
                                 menuItems,
                                 time,
                                 partyMembers,
                                 randomChef())
       GetOrdersDB().add(order)
       if not customer.isVIP():
           checkIfNowVIP(customer)
       return "Order successful"
   else:
       return "Deposit more money please"

# pre cond: customer is a customer with a deposit, menuItems is a list of of items from
the menu DB
# post cond: if the customer has enough deposit the order is placed and a chef is
assigned, checks if to make a customer a VIP
def orderTakeout(customer, menuItems):
   if pay(customer, menuItems):
       order = createOrderTakeout(customer,
                                  menuItems,
                                  randomChef())
       GetOrdersDB().add(order)
       if not customer.isVIP():
           checkIfNowVIP(customer)
       return "Order successful."
   else:
       return "Order failed."
```

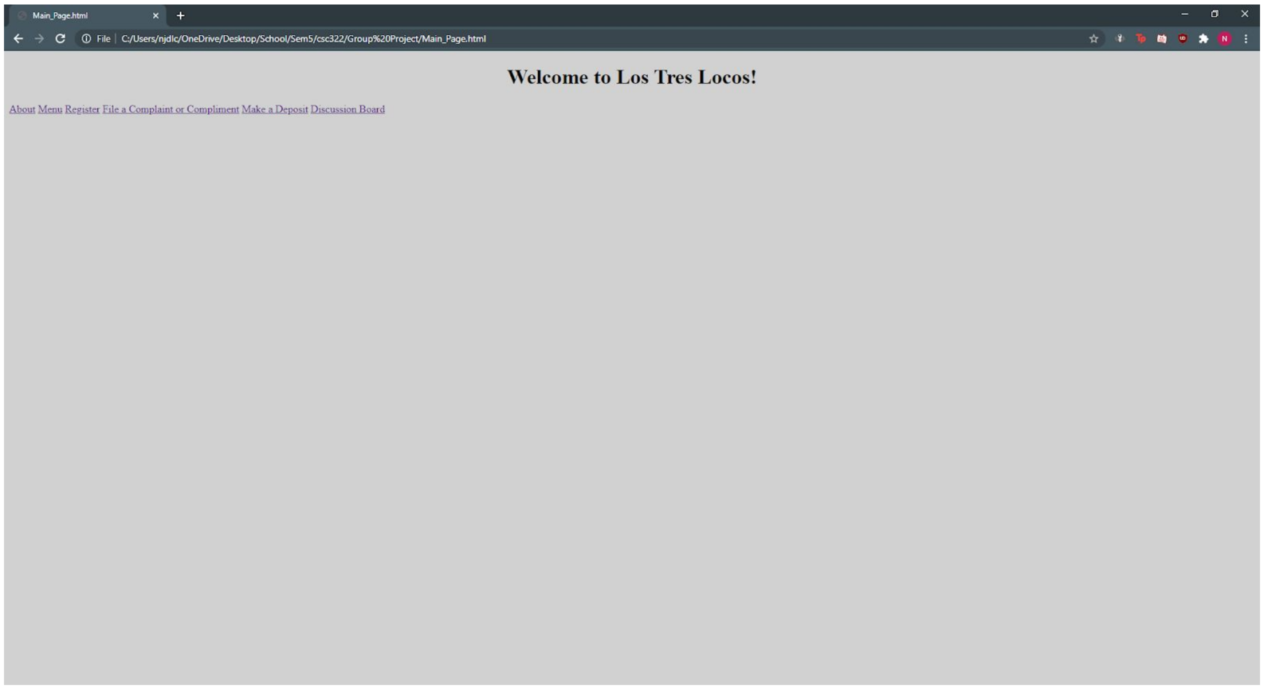
```
# pre cond: customer is a customer with a deposit, menuItems is a list of of items from
the menu DB
# post cond: if the customer has enough deposit the order is placed and a dp is
assigned, checks if to make a customer a VIP
def orderDelivery(customer, menuItems):
   if pay(customer, menuItems):
       order = createOrderDelivery(customer,
                                   menuItems,
                                   randomChef(),
                                   randomDP())
       GetOrdersDB().add(order)
       if not customer.isVIP():
           checkIfNowVIP(customer)
       return "Order successful."
   else:
       return "Order failed."

# pre cond: customer is a non VIP customer
# post cond: if the customer has >= $500 of orders placed or more than 50 orders placed
then turn them into a VIP
def checkIfNowVIP(customer):
   customerOrders = GetOrdersDB().getCustomerOrders(customer)
   totalNumberOfOrders = customerOrders.length
   totalPriceOfAllOrders = customerOrders.reduce(total, order => total + order.price)

   if totalNumberOfOrders >= 50 or totalPriceOfAllOrders >= 500:
       customer.setVIPStatus()
       GetCustomerDB().update(customer)
```

Note: Since the first phase report, we have changed our creative feature from birthday coupons to giving the customer the option to pay with cryptocurrency on the Ethereum network. All input added to the DB is expected to be formatted properly prior to being added. For example, all of the input fields of each variable added to the database is expected to have the components needed in the columns of its specific DB.

5. Systems Screens

5.1 Major GUI Screens

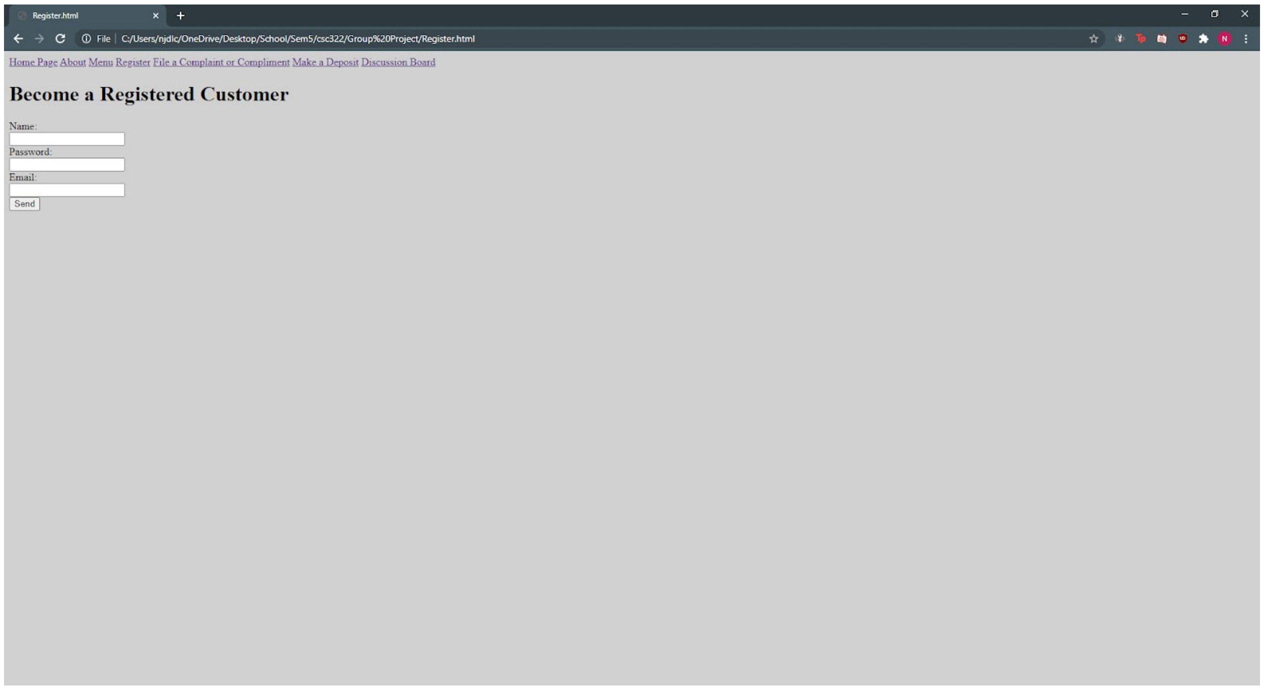
5.1.1 Main Page


Main_Page.html
File
C:/Users/njdlc/OneDrive/Desktop/School/Sem5/csc322/Group%20Project/Main_Page.html

Welcome to Los Tres Locos!

About Menu Register File a Complaint or Compliment Make a Deposit Discussion Board

5.1.2 Registration



Register.html
C:/Users/njdlc/OneDrive/Desktop/School/Sem5/csc322/Group%20Project/Register.html

Home Page About Menu Register File a Complaint or Compliment Make a Deposit Discussion Board

Become a Registered Customer

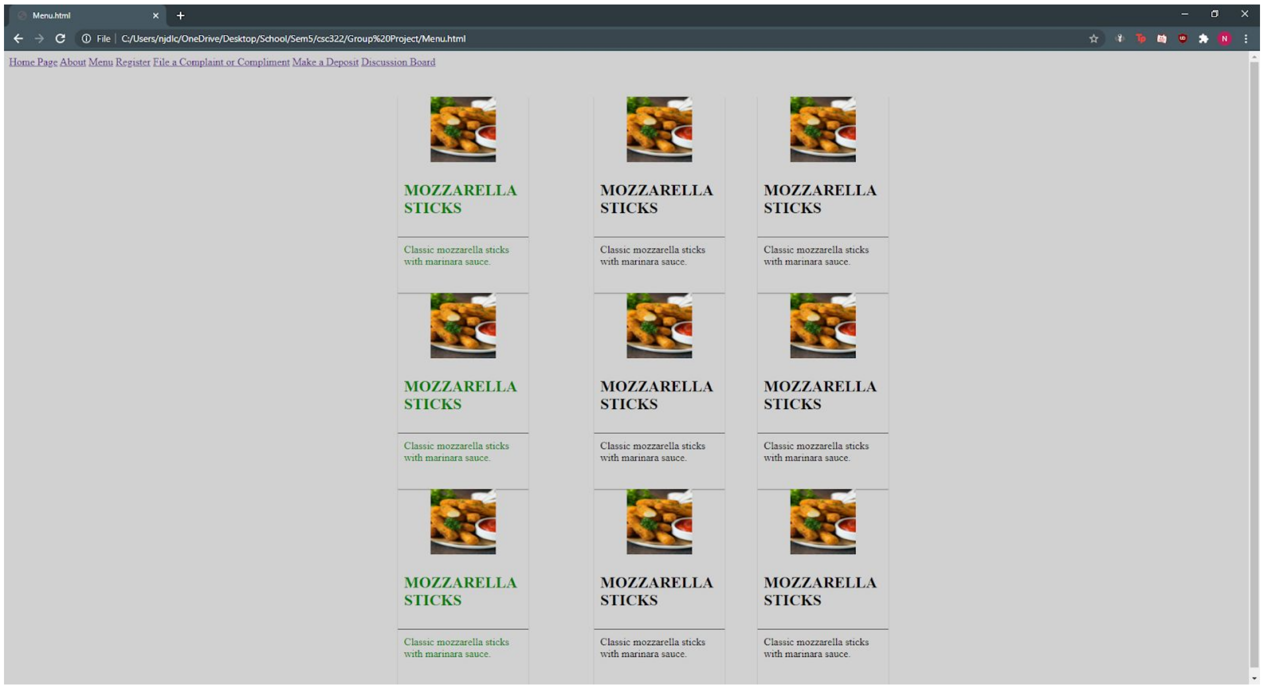
Name:

Password:

Email:

Send

5.1.3 Menu

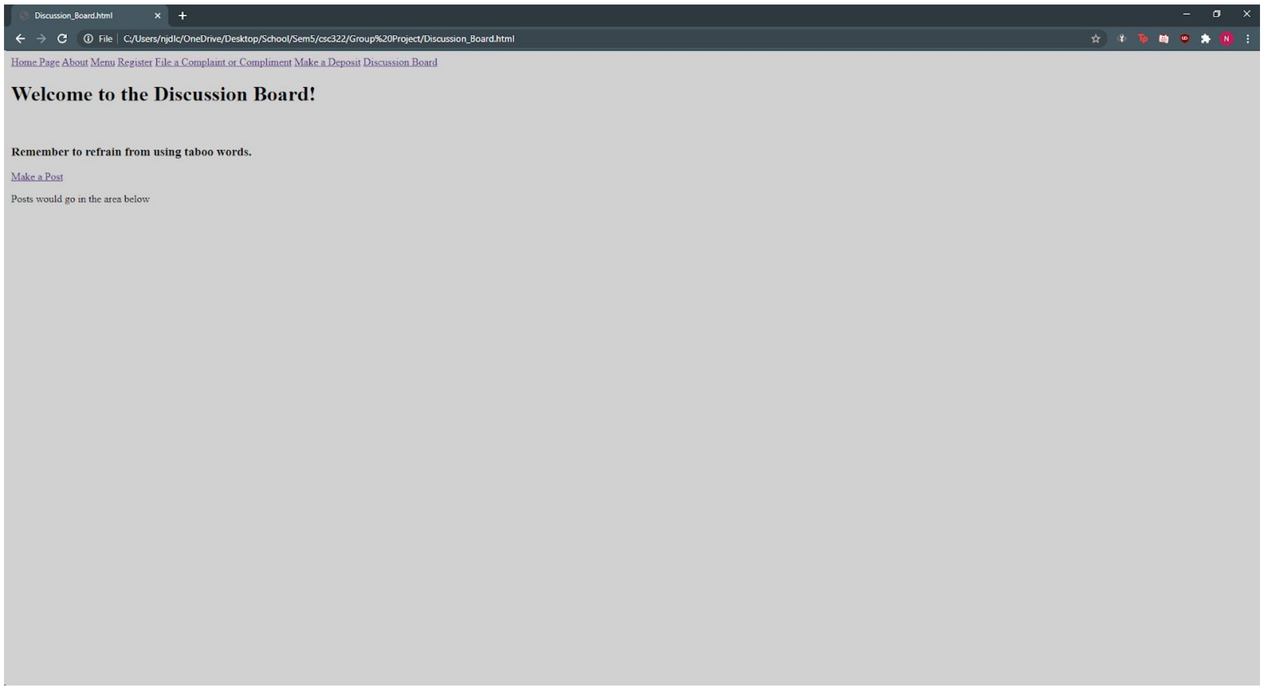


Menu.html
C:/Users/njdlc/OneDrive/Desktop/School/Sem5/csc322/Group%20Project/Menu.html

Home Page About Menu Register File a Complaint or Compliment Make a Deposit Discussion Board

MOZZARELLA STICKS Classic mozzarella sticks with marinara sauce.	MOZZARELLA STICKS Classic mozzarella sticks with marinara sauce.	MOZZARELLA STICKS Classic mozzarella sticks with marinara sauce.
MOZZARELLA STICKS Classic mozzarella sticks with marinara sauce.	MOZZARELLA STICKS Classic mozzarella sticks with marinara sauce.	MOZZARELLA STICKS Classic mozzarella sticks with marinara sauce.
MOZZARELLA STICKS Classic mozzarella sticks with marinara sauce.	MOZZARELLA STICKS Classic mozzarella sticks with marinara sauce.	MOZZARELLA STICKS Classic mozzarella sticks with marinara sauce.

5.1.4 Discussion Board


Discussion_Board.html

C:/Users/njdlc/OneDrive/Desktop/School/Sem5/csc322/Group%20Project/Discussion_Board.html

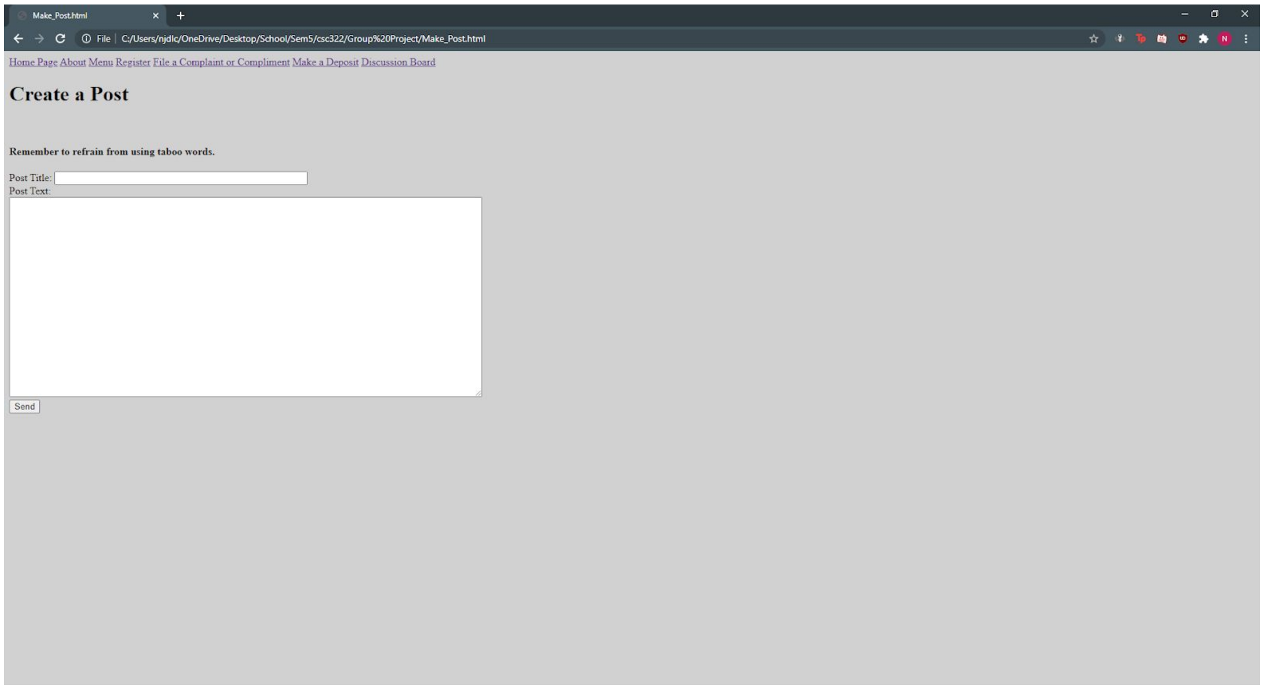
Home Page About Menu Register File a Complaint or Compliment Make a Deposit Discussion Board

Welcome to the Discussion Board!

Remember to refrain from using taboo words.

Make a Post

Posts would go in the area below


Make_Post.html

C:/Users/njdlc/OneDrive/Desktop/School/Sem5/csc322/Group%20Project/Make_Post.html

Home Page About Menu Register File a Complaint or Compliment Make a Deposit Discussion Board

Create a Post

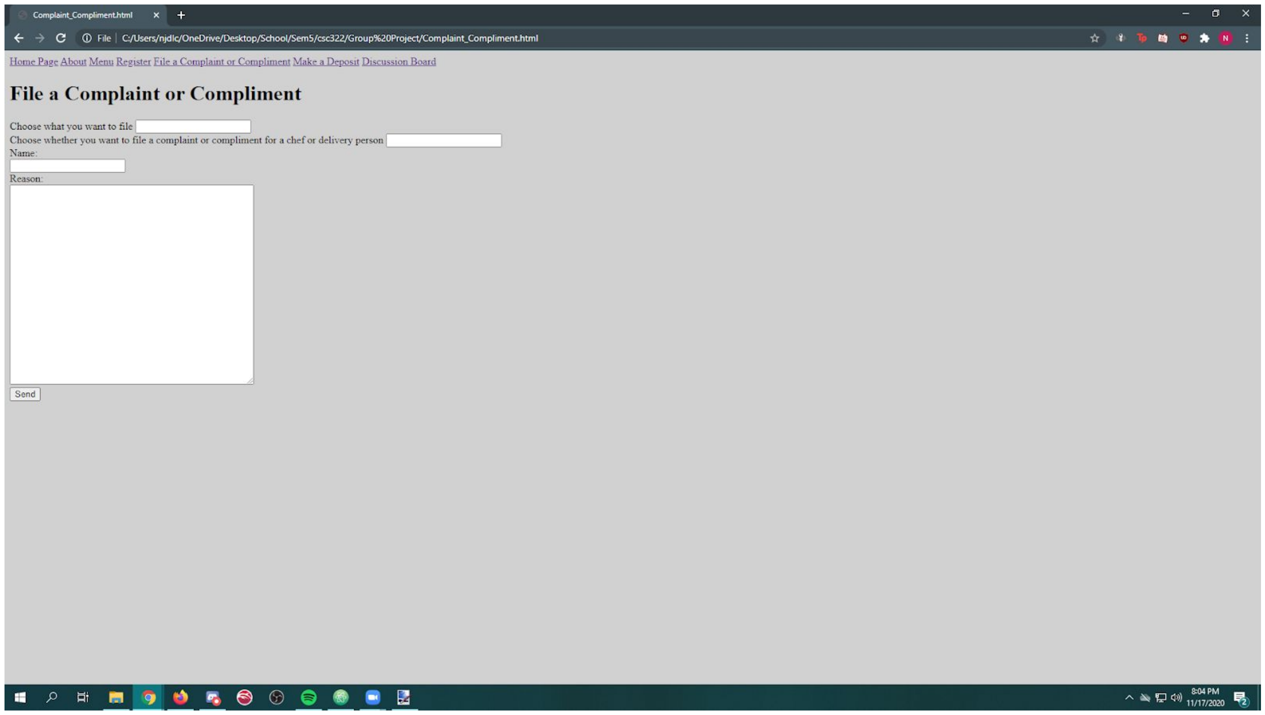
Remember to refrain from using taboo words.

Post Title:

Post Text:

Send

5.1.5 Filing a Complaint/Compliment


Complaint_Compliment.html

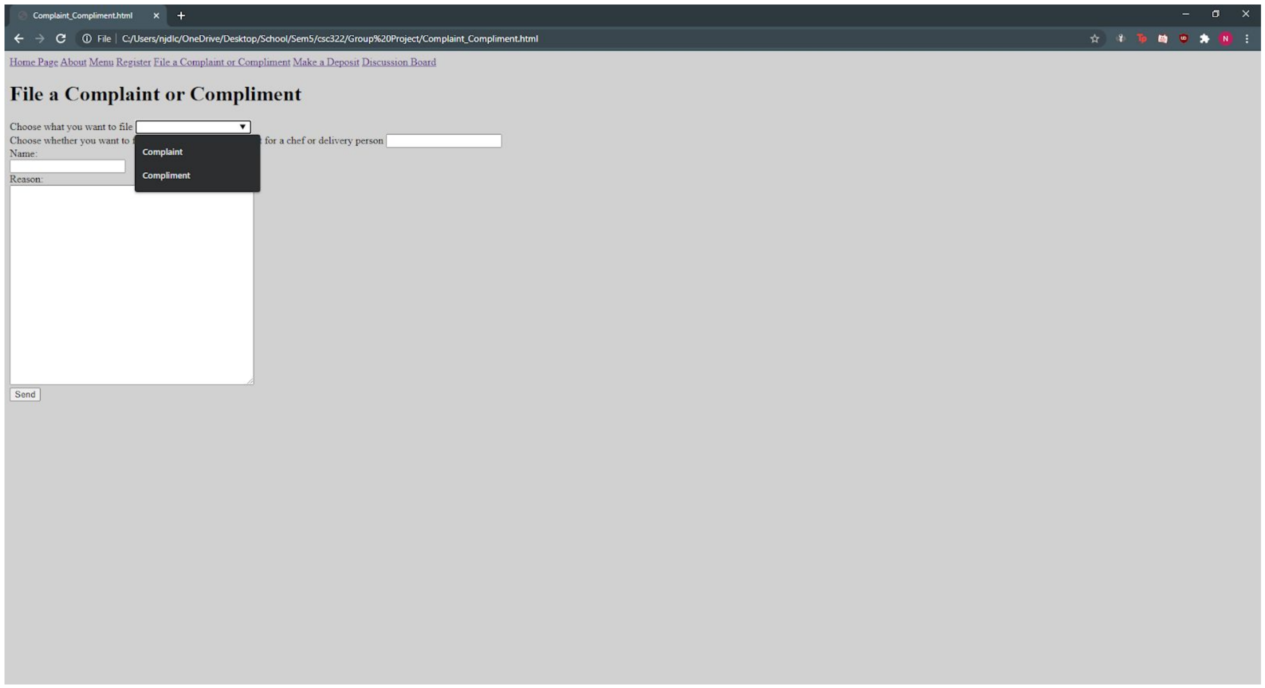
File | C:/Users/njdlc/OneDrive/Desktop/School/Sem5/csc322/Group%20Project/Complaint_Compliment.html

Home Page About Menu Register File a Complaint or Compliment Make a Deposit Discussion Board

File a Complaint or Compliment

Choose what you want to file
Choose whether you want to file a complaint or compliment for a chef or delivery person
Name:
Reason:

Send

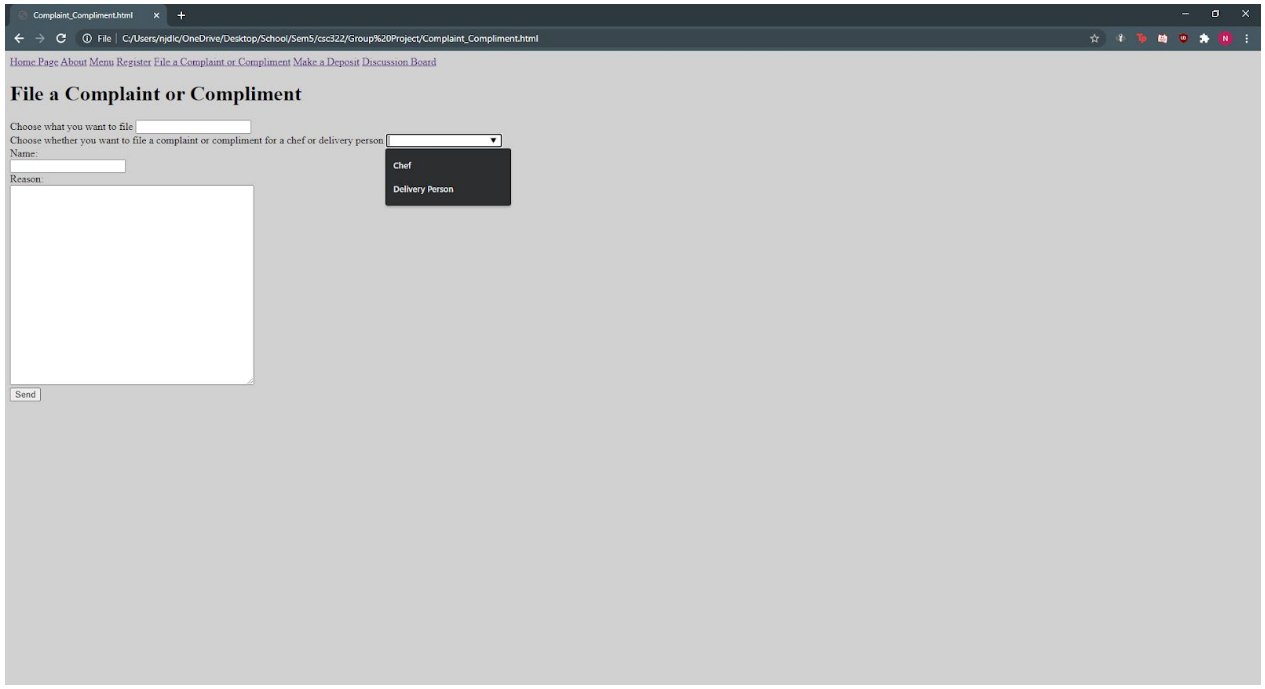

Complaint_Compliment.html

File | C:/Users/njdlc/OneDrive/Desktop/School/Sem5/csc322/Group%20Project/Complaint_Compliment.html

Home Page About Menu Register File a Complaint or Compliment Make a Deposit Discussion Board

File a Complaint or Compliment

Choose what you want to file
Choose whether you want to file a complaint or compliment for a chef or delivery person
Name:
Reason:


Complaint_Compliment.html
File | C:/Users/njdlc/OneDrive/Desktop/School/Sem5/csc322/Group%20Project/Complaint_Compliment.html

Home Page About Menu Register File a Complaint or Compliment Make a Deposit Discussion Board

File a Complaint or Compliment

Choose what you want to file

Choose whether you want to file a complaint or compliment for a chef or delivery person

Chef

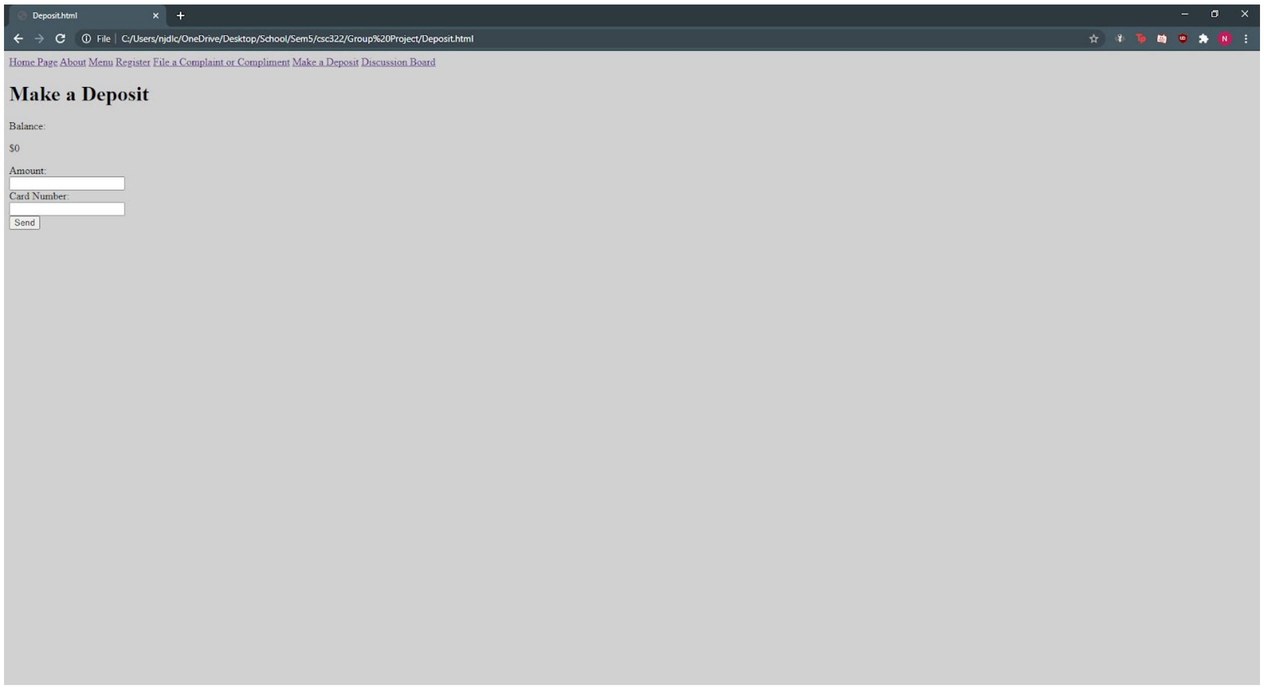
Delivery Person

Name:

Reason:

Send

5.1.6 Deposit


Deposit.html
File | C:/Users/njdlc/OneDrive/Desktop/School/Sem5/csc322/Group%20Project/Deposit.html

Home Page About Menu Register File a Complaint or Compliment Make a Deposit Discussion Board

Make a Deposit

Balance:

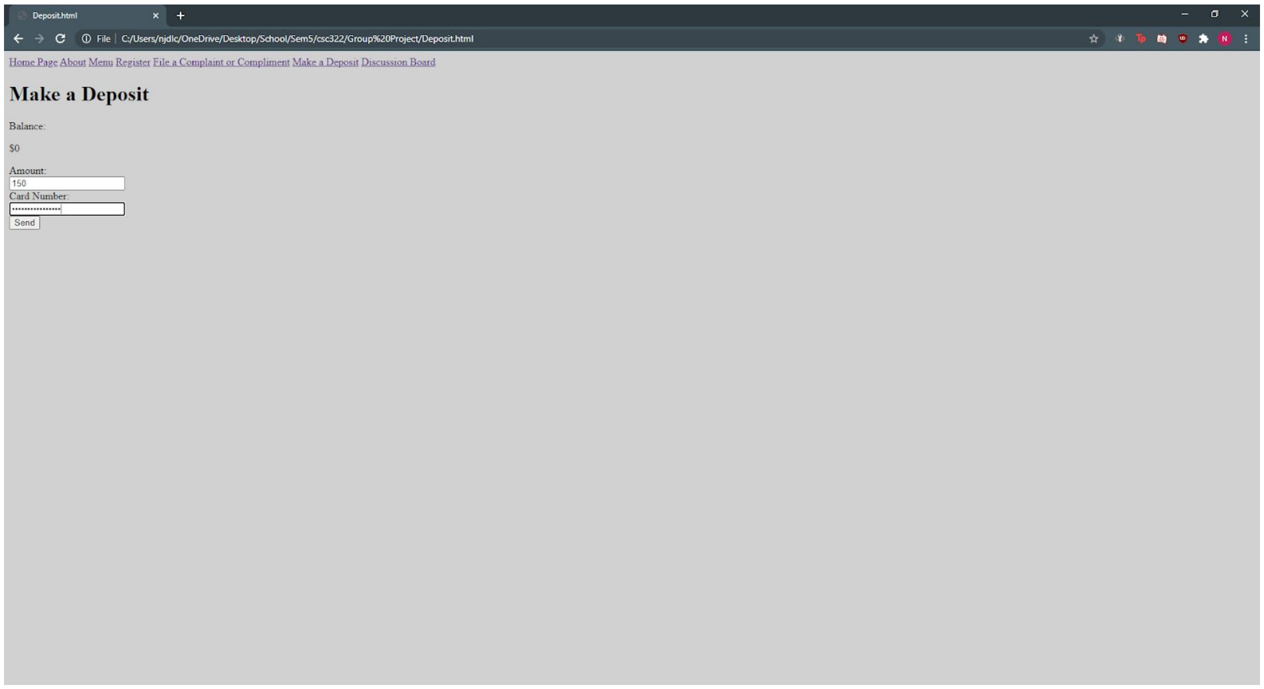
$0

Amount:

Card Number:

Send

5.2 Deposit Prototype



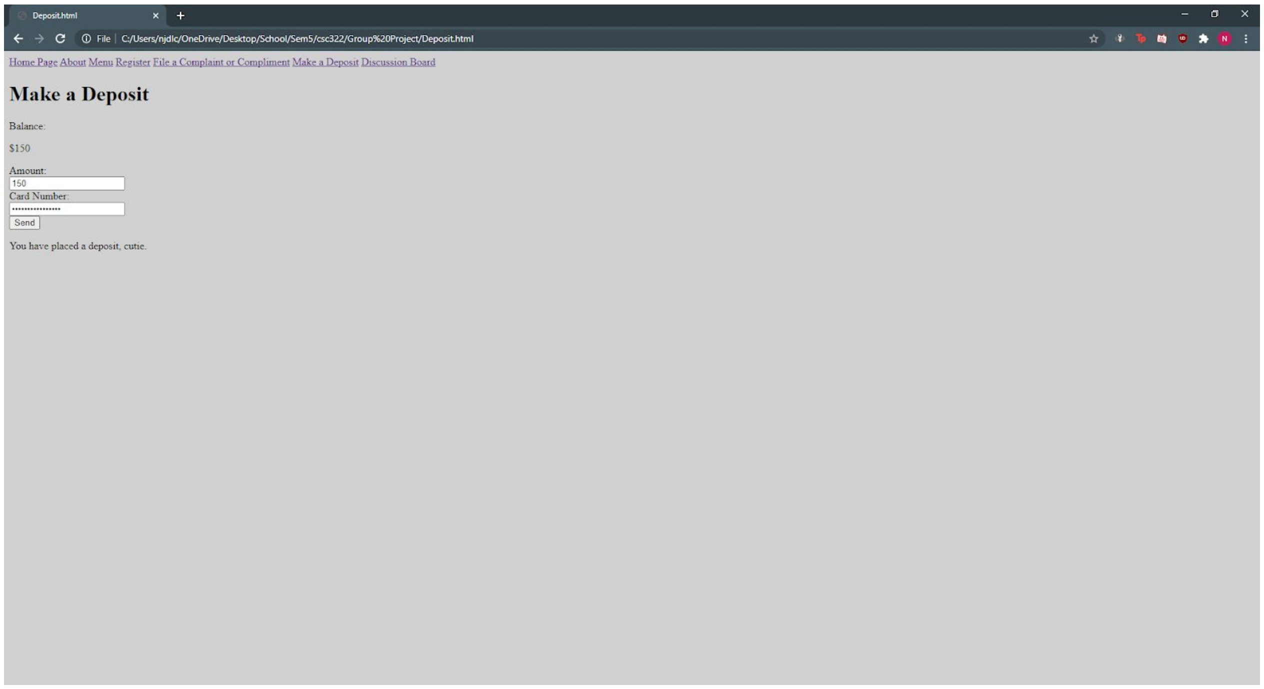
Deposit.html
File | C:/Users/njdlc/OneDrive/Desktop/School/Sem5/csc322/Group%20Project/Deposit.html
Home Page About Menu Register File a Complaint or Compliment Make a Deposit Discussion Board
Send

Make a Deposit

Balance:

$0

Amount:
150
Card Number:
••••••••••••••••



Deposit.html
File | C:/Users/njdlc/OneDrive/Desktop/School/Sem5/csc322/Group%20Project/Deposit.html
Home Page About Menu Register File a Complaint or Compliment Make a Deposit Discussion Board
Send

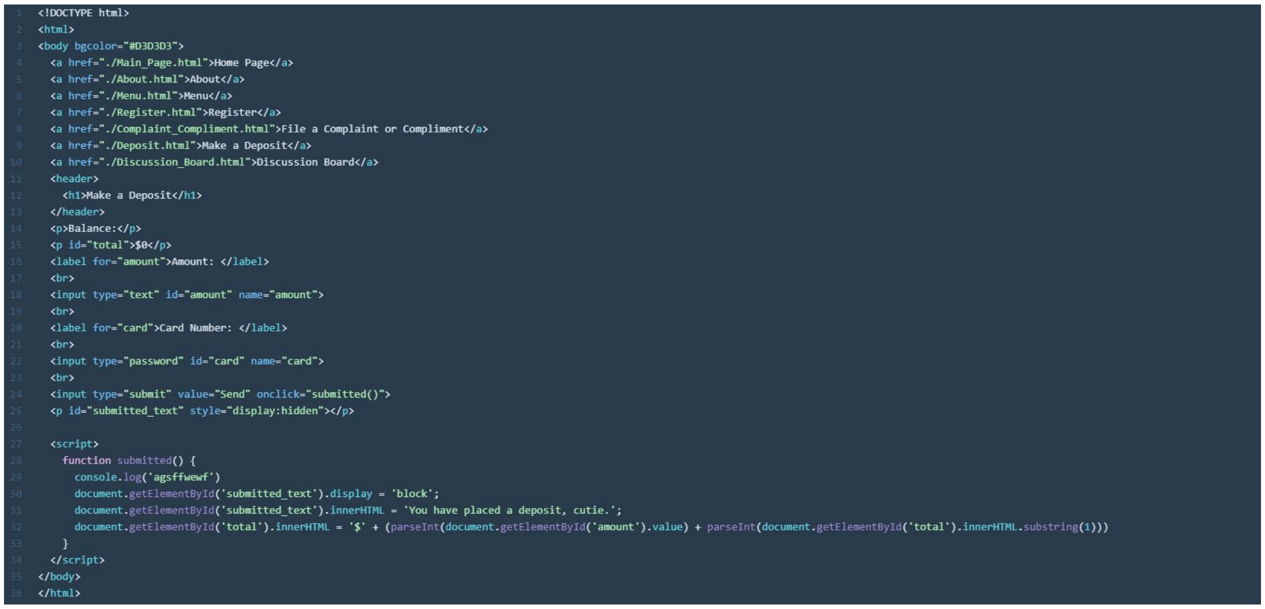
Make a Deposit

Balance:

$150

Amount:
150
Card Number:
••••••••••••••••

You have placed a deposit, cutie.



```
<!DOCTYPE html>
<html>
<body bgcolor="#D3D3D3">
  <a href="./Main_Page.html">Home Page</a>
  <a href="./About.html">About</a>
  <a href="./Menu.html">Menu</a>
  <a href="./Register.html">Register</a>
  <a href="./Complaint_Compliment.html">File a Complaint or Compliment</a>
  <a href="./Deposit.html">Make a Deposit</a>
  <a href="./Discussion_Board.html">Discussion Board</a>
  <header>
    <h1>Make a Deposit</h1>
  </header>
  <p>Balance:</p>
  <p id="total">$0</p>
  <label for="amount">Amount: </label>
  <br>
  <input type="text" id="amount" name="amount">
  <br>
  <label for="card">Card Number: </label>
  <br>
  <input type="password" id="card" name="card">
  <br>
  <input type="submit" value="Send" onclick="submitted()">
  <p id="submitted_text" style="display:hidden"></p>

  <script>
    function submitted() {
      console.log('agsffwewf')
      document.getElementById('submitted_text').display = 'block';
      document.getElementById('submitted_text').innerHTML = 'You have placed a deposit, cutie.';
      document.getElementById('total').innerHTML = '$' + (parseInt(document.getElementById('amount').value) + parseInt(document.getElementById('total').innerHTML.substring(1)))
    }
  </script>
</body>
</html>
```

6. Table of Group Meetings

6.1 Timetable of Group Meetings

Date	Time Spent in Meeting (hours)
10/9/20	0.5
10/15/20	1
10/16/20	1.5
10/18/20	1.5
10/19/20	6
10/20/20	1.5
10/30/20	1
10/31/20	1.5
11/5/20	0.5
11/6/20	1.5
11/7/20	2.5
11/8/20	1.5

11/12/20	1
11/13/20	1.5
11/14/20	2.5
11/15/20	4
11/17/20	3
Sum	32.5

7. GitHub Repository

7.1 Link to GitHub Repository

https://github.com/kareem-ib/CSC322-Restaurant-App