Norm's Pizza

Norm's

PIZZA

Electronic Restaurant Order and Delivery System Design Report For Online Restaurant System

Version 2.0

Revision History

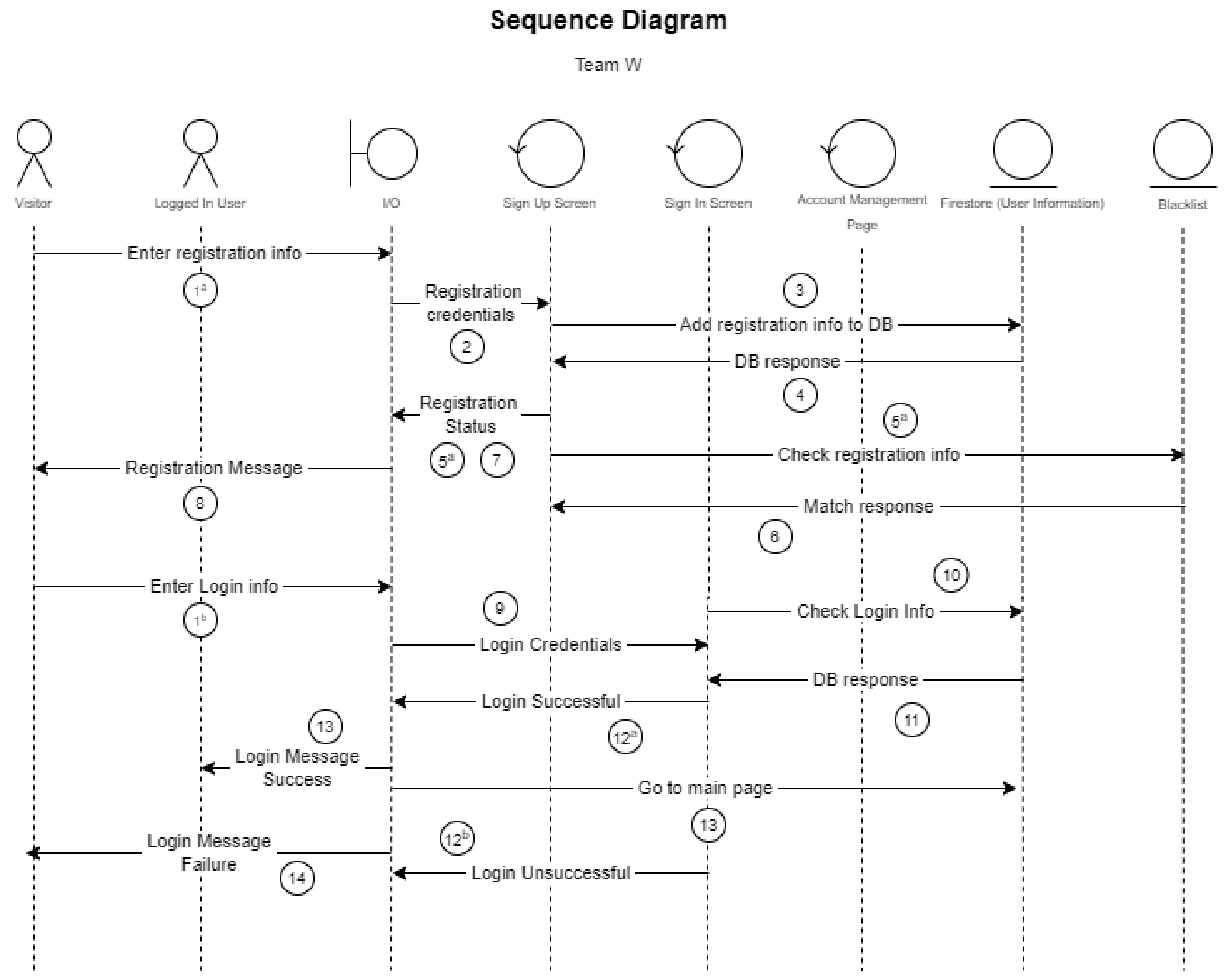
Date	Version	Description	Author
28 / Apr / 22	2.0	Sequence class diagram, Detailed Design	Azwad Shameem
28 / Apr / 22	2.0	Memos of group meetings, Use Cases	Dawa Sherpa
28 / Apr / 22	2.0	Use Cases	Jeevan Bastola
28 / Apr / 22	2.0	ER Diagram, System Screens	Ali Syed
28 / Apr / 22	2.0	Sequence class diagram, Detailed Design	Nishanth Prajith

Table of Contents

Design Report

1. Introduction

1.1 Sequence class diagram



2. Use Cases

[Visitors Use Case]

Use-Case : Apply to be registered customers

Descriptions : Visitors can apply to be registered customers, they will have to be approved by the Manager.

Normal Case : The manager approves the request and the person now becomes a registered user.

Exceptional Case : The manager rejects the user due to some possible reason and then they would have to register again.

Use-Case : Browse the menu

Descriptions : Visitors can browse the menu and look at the dishes. There are no exceptional cases for this

[Registered Customers and VIP Customers Use Cases]

Use-Case : Browse the menu

Descriptions : Registered customers can browse the menu and add to their cart.

Normal Case : Register customers can see all the dishes with the exception of the special dishes.

Exceptional Case : If they are VIP customers they can also see special dishes (chef specials)

Use-Case : Rating orders and Posting complaints/compliments

Description : Registered Customers can rate and post complaints/compliments on the food they ordered to the chef of that food and if the order was delivered they are also allowed to rate and post complaints/compliments on the delivery personnel about their delivery. Rating will be from 1 star to 5 stars.

Normal Case : Registered customer complaints are taken into account and for the manager to review them and make a decision in regards to them.

Exceptional Case : VIP customers' complaints/compliments are counted twice as important as ordinary ones.

Use-Case : Discuss about chefs/dishes/delivery personnel.

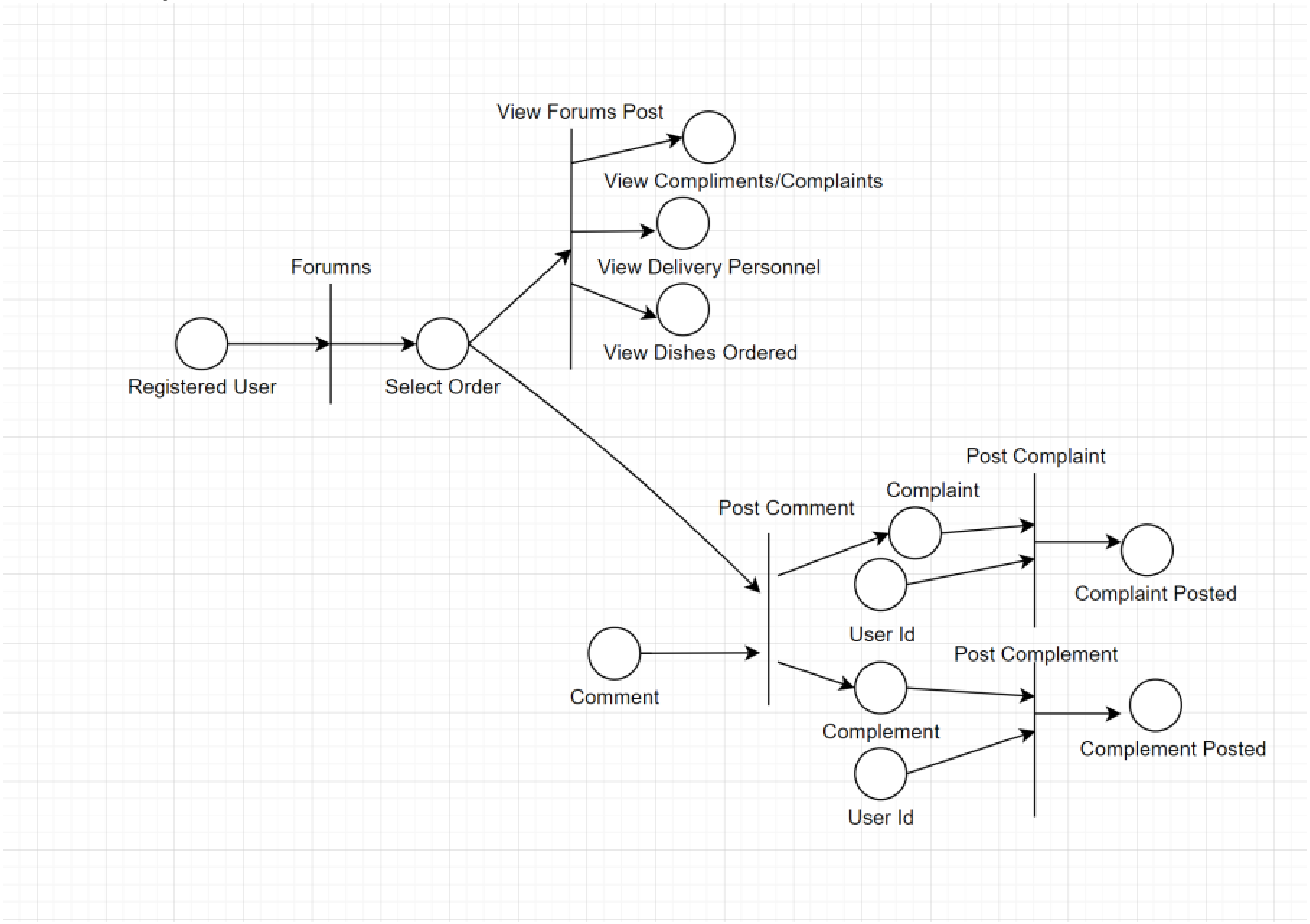
Description : Registered Customers can start/participate on a discussion topic on chefs/dishes/delivery personnel. There are no exceptional cases.

Use-Case : Dispute Complaints

Description : If a complaint is made about a registered customer or VIP customer, they have the right to dispute the complaint.

Normal Case : A dispute is made to a complaint and the manager will take into account the dispute before making their decision.

Exceptional Case : The dispute is ignored and the user gets a warning for making a false argument.

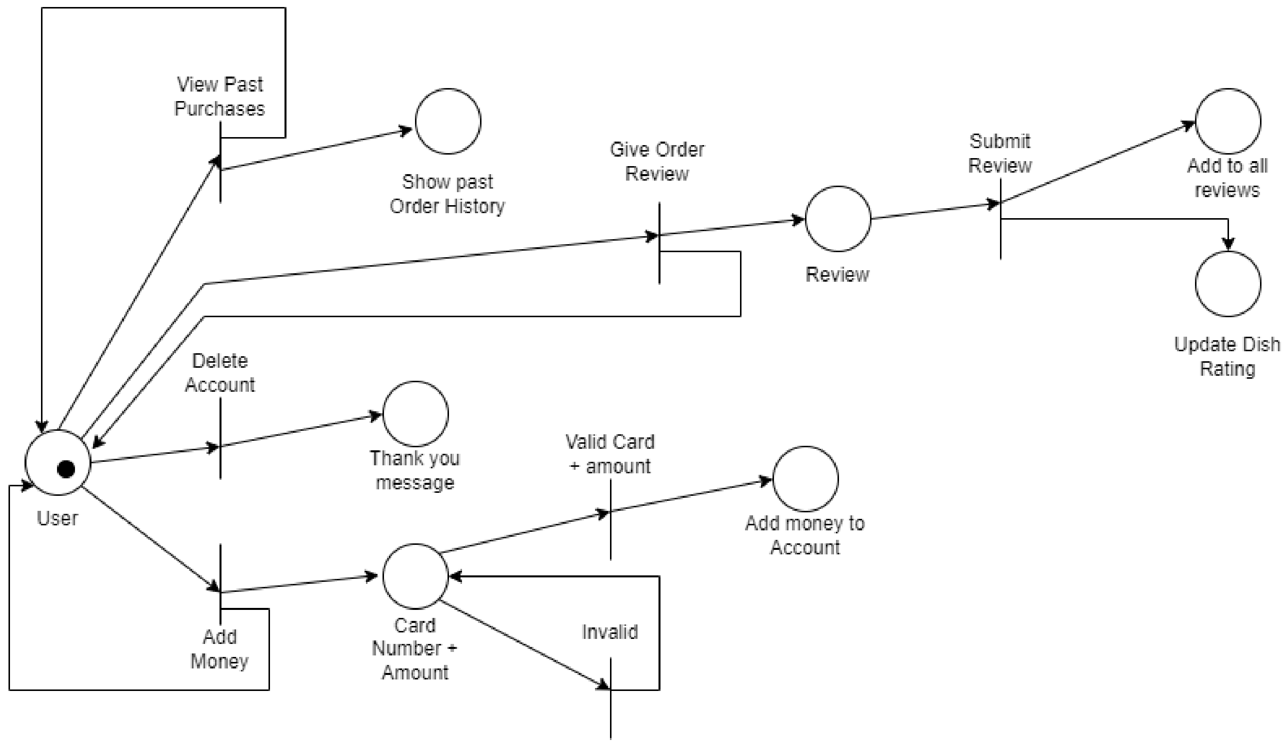


Use-Case : Deposit money

Description : Registered customers are allowed to deposit money to the system. They do this by going to their profile and adding to their wallet. Then they will have to provide the card number and the amount of money they would like to deposit to their amount (max amount is $100). There are no exceptional cases here.

Use-Case : VIew Past Purchase History

Description : A registered custom or VIP customer can view their past purchase history. Including the total cost of the order, as well as see the review they left for that order. There are no exceptional cases here.



Use-Case : Order food

Description / Normal Case : Registered Customers can order any of the food options available either for pickup or delivery.

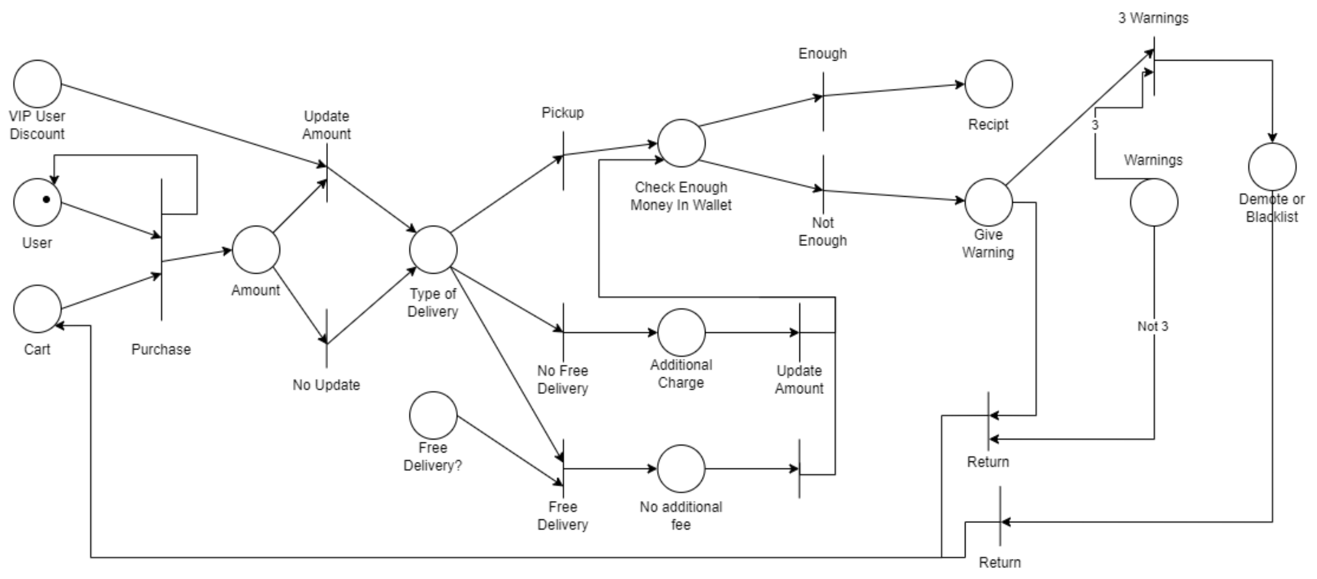
Exceptional Case : If they are VIP customers they will receive 5% discount of their ordinary orders and 1 free delivery for every 3 orders.

Use-Case : Make a purchase

Description : A registered or VIP customer can make a purchase by adding some items to their cart.

Normal Case : During the checkout process the customer chooses the type of delivery and if they have enough money in their wallet the order will get placed.

Exceptional Case : The user does not have enough money in their wallet, in which case they are given a warning and asked to add money before proceeding.



[Employees Use Cases (Manager, Chefs, and Delivery personnel)]

Use-Case : Create dish

Description : Chefs are allowed to independently create dishes and decide on the menus. There are no exceptional cases here.

Use-Case : Post complaints/compliments about Customers

Description: Delivery personnel and chefs are allowed to post complaints/compliments about customers to whom they have delivered food. There are no exceptional cases.

Use-Case : Assign orders to delivery personnel

Description : Managers will assign orders to delivery personnel depending on the bidding results. In general, the one with the lowest delivery price is picked; if the one with the higher asking price is chosen, the manager should justify his decision with a memo in the system.

Use-Case : Remove Customers

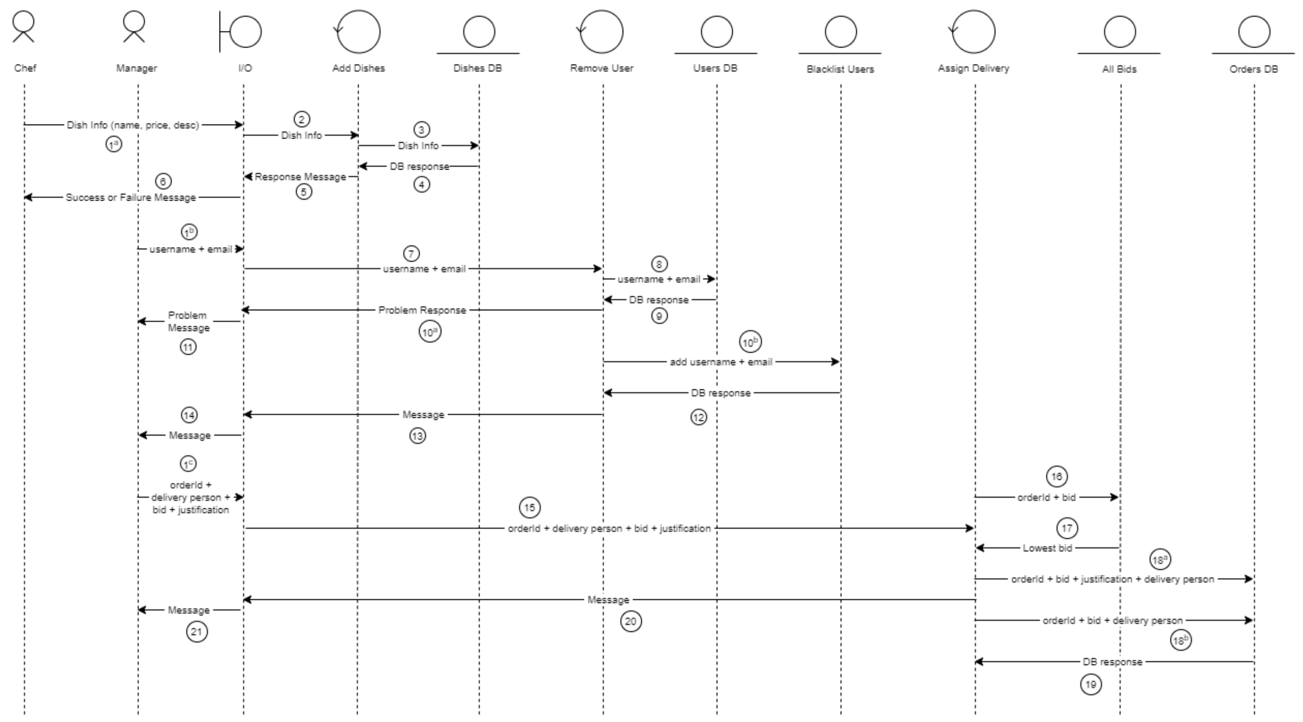
Description : Managers are allowed to kick customers out of the system.

Normal Case : Customers choose to quit the system and the manager will clear the deposit and close the account.

Exceptional Case : Manager kicks out the customer and then they will be added to the blacklist of the restaurant and won’t be able to register anymore.

Use-Case : Handle Customer Comments and Delivery Personnel Comments

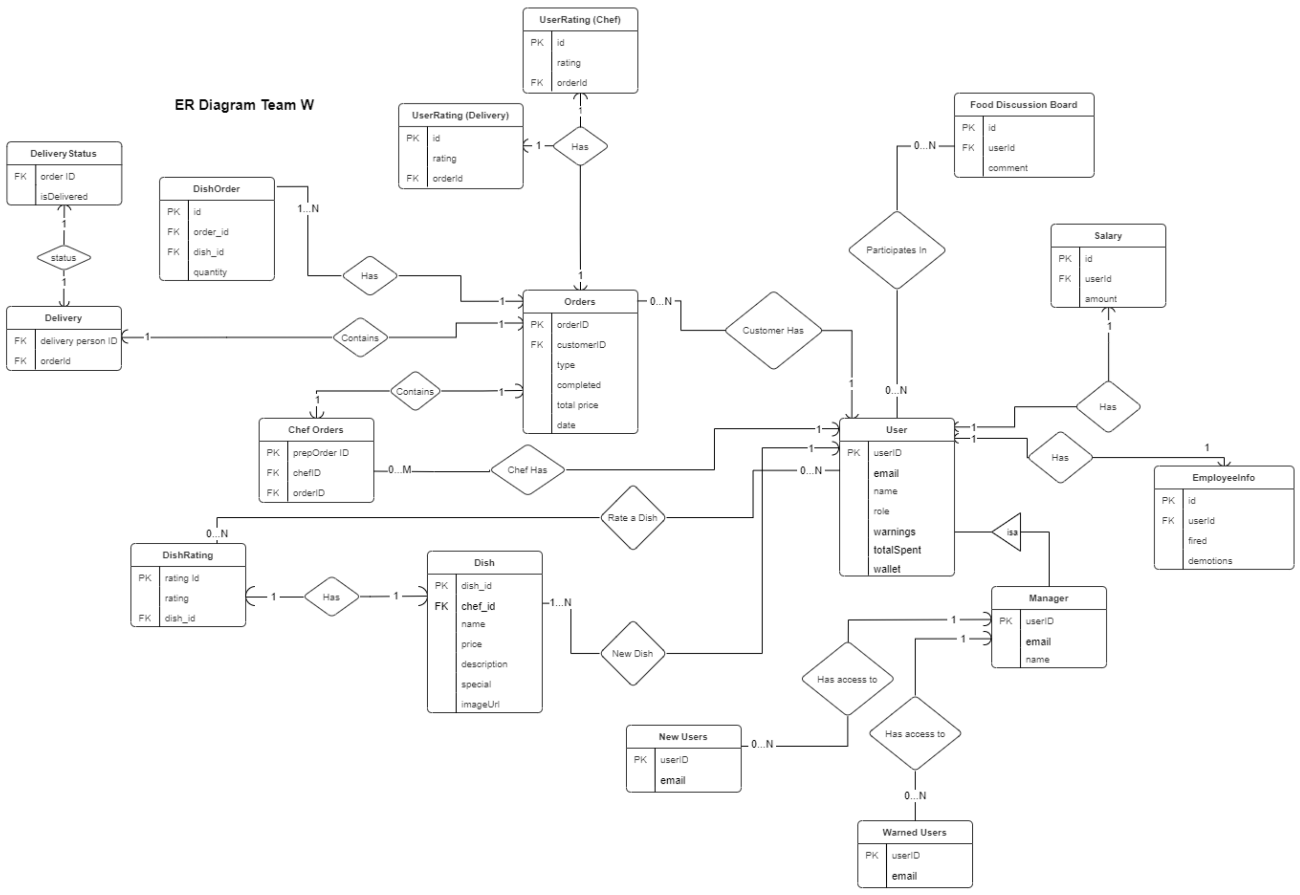
Description : Managers look at customer comments and delivery personnel comments then decide if any action should be taken. Managers make the final call to dismiss the complaints or give warnings and inform the impacted parties.



https://drive.google.com/file/d/1XSJpYOCB17XbFRcb-a67xOla1vTtBlGD/view?usp=sharing

3. E-R diagram

Link to E-R diagram : https://drive.google.com/file/d/1tWfH4ZN0XsiGsMLbyCJgnYkgXtntMVtK/view?usp=sharing



4. Detailed design

Home Page:

AddToCart: This is a function for the user to add a particular item to the cart.

```
def addtoCart(dishId):
      dishes[dishId].currentQuantity += 1
      updateCart(dishes)
```

SubtractFromCart: This is a function for the user to subtract a particular item from the cart.

```
def subtractfromCart(dishId):
      a = dishes[dishId].currentQuantity - 1
      dishes[dishId].currentQuantity = max(0, a)
      updateCart(dishes)
```

PreviousOrders: This is a function automatically called when the registered user logs in. It returns the past 3 dishes based on the user's previous orders. Note that this function only returns something if there are more than 3 dishes to be returned else it returns an empty array.

```
def previousOrders(userId):
      try :
            previousDishes = []
            orders = getUserOrders(userId)
            for order in orders:
                  for dish in order:
                        previousDishes (dish)
                        if len(previousDishes) == 3:
                              return previousDishes
            return []
      catch (error) :
            return error.message
```

View Listings : This is a function accessible to all users Registered, VIP and Visitor. The ordinary case is that the listings are available for anyone to see. The exceptional case is if the user is a VIP they get to see additional special listings that visitors and regular registered customers will not be able to see.

```
def viewListings(user):
      listings = []
      if (user.info != "VIP"):
            q = "SELECT * FROM Dishes where special = False"
            listings = executeQuery(q)
      else:
           q = "SELECT * FROM Dishes"
           listings = executeQuery(q)
      return listings
```

Browse Listings : This is a function accessible to all users to search for a particular item. The ordinary case is that they will be able to search for all the dishes available. The exceptional case is that visitors and registered customers who are not VIP will not be able to search for chef’s special dishes.

```
def browseListings(user, searchKeyword):
      possibleListings = viewListing(user)
      filteredListings = []
      for dish in possibleListings:
            if (dish.name.__contains__(searchKeyword)):
                  filteredListings.append(dish)
      return filteredListings
```

Login Page:

Register: This is a function which is accessible to a visitor to apply to be a registered customer. The normal case is that the visitor does not have an account and they enter their information. Exceptional cases include the visitor already being a registered user or the visitor being on the avoid list, in both cases they will receive a denial message.

```
def Register(name, email, password) {
        if (name === "" || email === "" || password === ""):
            return "empty field"
        else :
            try :
               # DB function to register
```

```
                status = signup(name, email, password)
                return status.message
            catch (error) :
                 return error.message
```

Login: This a function which is accessible to all users Ordinary and Privileged to Login to their respective accounts. They will gain access to their respective functions. The normal case is the user provides a valid username and password. Exceptional cases are when the user provides a wrong username and/or password.

```
def Login(email, password):
       if (email === "" || password === ""):
            return "empty Field"
      else :
            try :
                  # Verify username and password from DB
                  userInfo, status = signIn(email, password)
                  updateLocalInfo(userInfo)
                  return status.message
            catch (error):
                  return error.message
```

Account Management:

Add money to wallet: This is a function accessible to registered users to add money to their wallet. The ordinary case is that the user enters valid information for the card number and money field. The exceptional case is that the user enters an invalid or unacceptable input for any one or multiple inputs.

```
def addToWallet(cardNumber, amount, user):
      # Note: Input field only takes integers
      # Users cannot input letters
      if (cardNumber === Null):
            return "Invalid Card Number"
      elif (amount < 0):
            return "Amount should be greater than 0"
      else :
            try :
                # Verify card
                status, amountAvaliable = verifyCard(cardNumber)
                if (amount > amountAvaliable):
                     return "Insufficient Funds"
                executeTransaction(cardNumber, amount)
                addMoneyToUserAccount(user, amount)
            catch (error):
                return error.message
```

Accept or reject new registrations : This is a function accessible to the manager, where they have the opportunity to either accept or reject a new registration. The ordinary case is that the manager can put the choice value of ‘1’ which means accept and ‘0’ means reject. There are no exceptional cases.

```
def AcceptOrReject(email, choice):
      if (choice == 1):
            try :
              # Accept the user and add change their role in DB
                  accept(email)
                  return "Success"
            catch (error):
                  return error
                  if (choice == 0):
            try :
                  # reject the user and remove their info to DB
                  deny(email)
                  return "Success"
            catch (error):
                  return error
```

View Past Purchase History : This is a function accessible for the registered user to view their past purchases. The ordinary use case is that the registered user’s purchase history becomes available. The exceptional case is when the purchase history is empty, in which it would show nothing. The other exceptional case is where there is a database error in which it returns the error message.

```
def viewPurchases(userID):
      try :
            # Ask DB to give the past purchases information
            orders = getUserOrders(userID)
            orderData = []
            for order in orders :
                  orderData.push(order.info)
            return orderData
      catch (error) :
            return error.message
```

Issue Account Warning : This is a function accessible to the manager where they have the power to issue a warning. The ordinary case is that the manager issues the warning depending on the complaints to the user and the user can view the warning. The exceptional case is that the current warning is the third warning, which will lead to the user being automatically removed from the system and added to the avoid list. The exceptional case for the chef is that if the current warning is the third warning, the chef will be demoted, if the chef is demoted twice they are fired.

```
def issueAccountWarning(userId):
      try :
         if (userId.info.role == "Chef"):
            if (userId.info.numWarnings == 2):
                  if (userId.info.numDemotions == 1):
                        fireChef(userId)
                        return "Chef Fired"
                  else:
                        demoteChef(userId)
                        return "Chef Demoted"
             else:
                  issueWarning(userId)
         else:
            issueWarning(userId)
            return "Issued Warning"
      catch (error) :
            return error.message
```

Suspend account : This is a function accessible to the manager to restrict a registered user/chef/delivery. The ordinary case is that the manager suspends a registered customer. The exceptional case is that the manager suspends a chef or delivery personnel.

```
def suspendAccount(userId):
      try:
            # Delete account info from DB and add them to blacklist
            deleteAccount(userId)
            addToBlackList(userId.info.email)
            return "success"
```

```
        catch (error) :
                return error.message
```

Handle account deletion: This is a function accessible to the manager to handle registered users who have decided to quit from the system. The ordinary case is that the manager deletes the registered customer's account and refunds the money in their account. There are no exceptional cases.

```
def deleteQuitUserAccount(userId):
        # The user has decided to quit so we delete their account
        q = "DELETE FROM userTable WHERE userId =" + str(userId)
        execute.query(q)
```

Purchases:

Submit Purchase : This is a function accessible to registered users to submit their purchase order once they have made their desired selections along with the delivery choice (pickup or delivery). The usual case is that the registered customer's order is submitted without issue. The exceptional case is that the registered customer's payment method does not have sufficient funds at which point the user gets a warning for negligence.

```
def submitPurchase(user, cart, choice):
        if (user.info.wallet  < cart.total):
                print("Insufficient Funds")
                giveWarning(user)
        else:
                # Removes money from wallet
                payFromWallet(user, cart.total)
                # Add order to orders table
                addOrders(user, cart, choice)
                # Clear the cart
                cart.clear()
```

Bid on Delivery : This is a function accessible to a delivery personnel to place a bid on delivering a purchase to the customer. The ordinary case is that the delivery personnel bids and waits for the manager to notify if they received the order. There is no exceptional case.

```
def deliveryBid(user, orderId,  bid):
      # Delivery personnel bids to deliver the order
      if (bid <= 0):
            alert("Bid cannot be negative or zero")
            return
      else:
            # saves bid in the DB
            placeBid(userId, orderId, bid)
            # Pings the manager
            notifyManager()
```

Accept Purchase Delivery : This is a function accessible to managers to accept a purchase delivery option. The ordinary case is that the manager picks the cheaper delivery bid or choses another option with sufficient justification. The exceptional case is that the manager does not pick the cheapest delivery bid and does not give justification, at which point the manager gets notified of the problem and the function returns without accepting the bid.

```
def chooseDelivery(orderId, selectedBid, justification):
      # Shows the manager the bids
      bids = loadBids(orderId)
      if (min(bids) == selectedBid):
            AcceptBid(orderId, selectedBid, "")
            notifyDelivery(selectedBid.user)
            return
      else:
            if (justification = "" OR justification = NULL):
                  alert("Justification must be made")
                  return
            AcceptBid(orderId, selectedBid, justification)
            notifyDelivery(selectedBid.user)
            return
```

Complaints Page :

File Complaints : This is a function accessible to a registered user to file a complaint about a dish, delivery person, or chef. The ordinary case is that the registered user makes a complaint and the user it is made against is informed. The exceptional case is that the user that the complaint was made against provides sufficient counter information and the manager clears the warning on their account.

```
def fileComplaint(orderId, personId, complaint, userId):
   try :
      addComplaintToOrder(complaint, userId)
      GiveComplaintToUser(personId, userId, orderId, complaint)
      print("Complaint filed successfully")
   catch (error):
      print(error.message)
```

View Complaint : This is a function accessible to all users except the manager to view the complaint made against them. There is only an ordinary case which is that the user the complaint was made against and the manager user can view the complaint.

```
def viewComplaint(user, userId):
   try :
      var complaints = getAllComplaints(userId, user.orders)
      var filtered = []
      for each complaint in complaints:
            if (complaint against userId):
                  filtered.append(complaint)
      return filtered
   catch (error) :
      return error.message
```

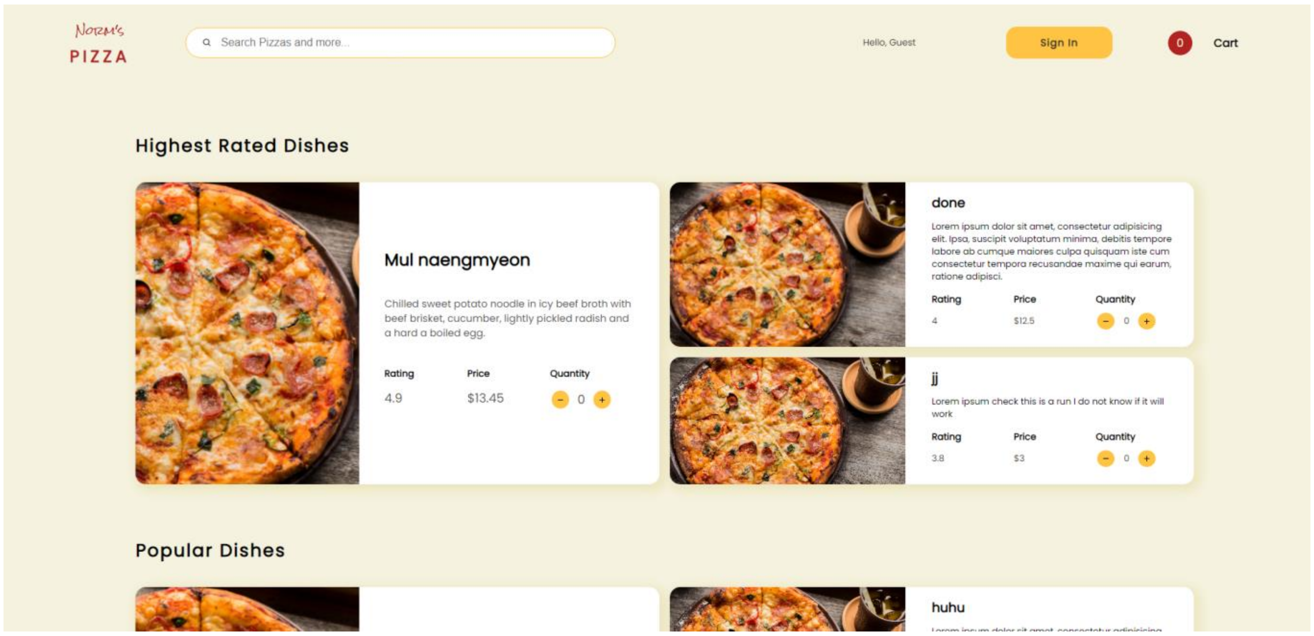
Manage Complaint : This is a function accessible to the manager users to remove complaints when justification has sufficient information. The ordinary case is that the justification does not have sufficient information to be removed and is kept in place or it does have sufficient information and is removed. The exceptional case is that the complaint is the third warning, and the account is either suspended and then reinstated, or indefinitely suspended until the issues are resolved.

```
def manageComplaint(userId, justification):
      # Get number of complaints
      var numComplaints = getNumComplaints(userId)
      # If justification is sufficient remove complaint
      if justification == 1:
            updateUserNumComplaints(userId, numComplaints - 1)
            removeComplaint(userId, complaint)
      # Justification is not sufficient + 3rd complaint
      if userId.info.numComplaints == 2:
            suspend(userId)
```

5. System screens

5.1 Major GUI Screens

The Landing/Home Page of the Norm’s Pizza Website



Highest Rated Dishes

Mul naengmyeon

Chilled sweet potato noodle in icy beef broth with beef brisket, cucumber, lightly pickled radish and a hard a boiled egg.

Rating	Price	Quantity
4.9	$13.45	0

done

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Ipsa, suscipit voluptatum minima, debitis tempore labore ab cumque maiores culpa quisquam iste cum consectetur tempora recusandae maxime qui earum, ratione adipisci.

Rating	Price	Quantity
4	$12.5	0

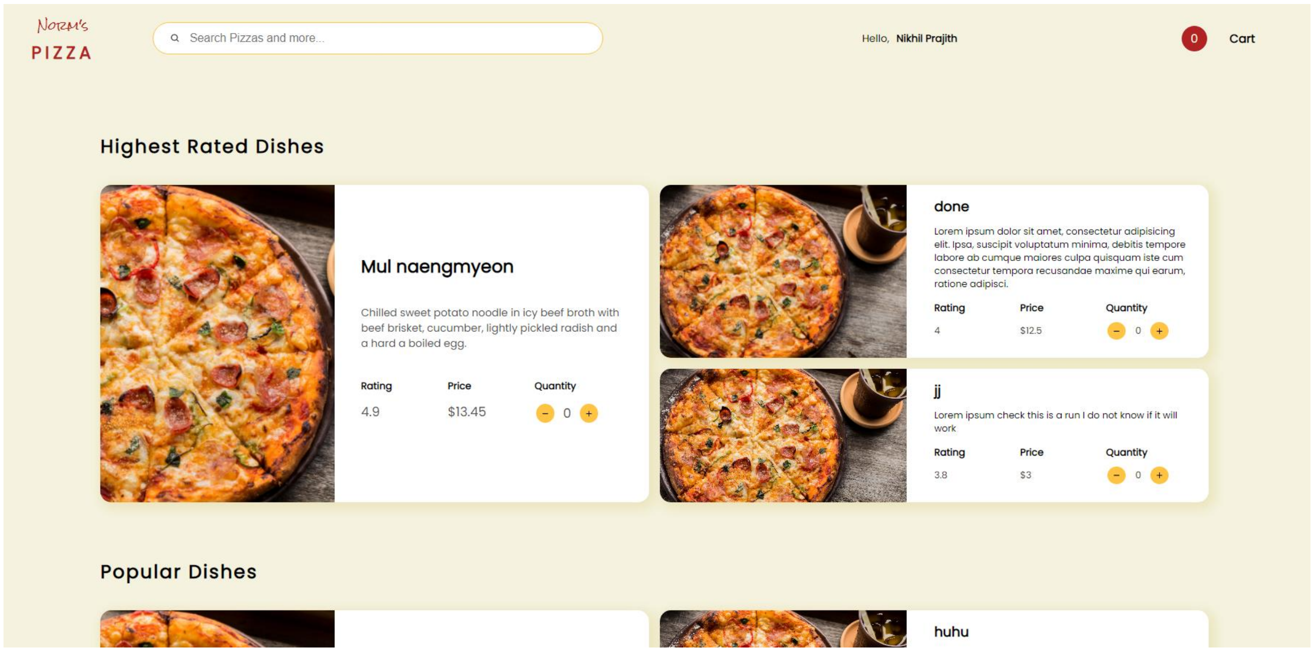
jj

Lorem ipsum check this is a run I do not know if it will work

Rating	Price	Quantity
3.8	$3	0

Popular Dishes

huhu



Highest Rated Dishes

Mul naengmyeon

Chilled sweet potato noodle in icy beef broth with beef brisket, cucumber, lightly pickled radish and a hard a boiled egg.

Rating	Price	Quantity
4.9	$13.45	0

done

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Ipsa, suscipit voluptatum minima, debitis tempore labore ab cumque maiores culpa quisquam iste cum consectetur tempora recusandae maxime qui earum, ratione adipisci.

Rating	Price	Quantity
4	$12.5	0

jj

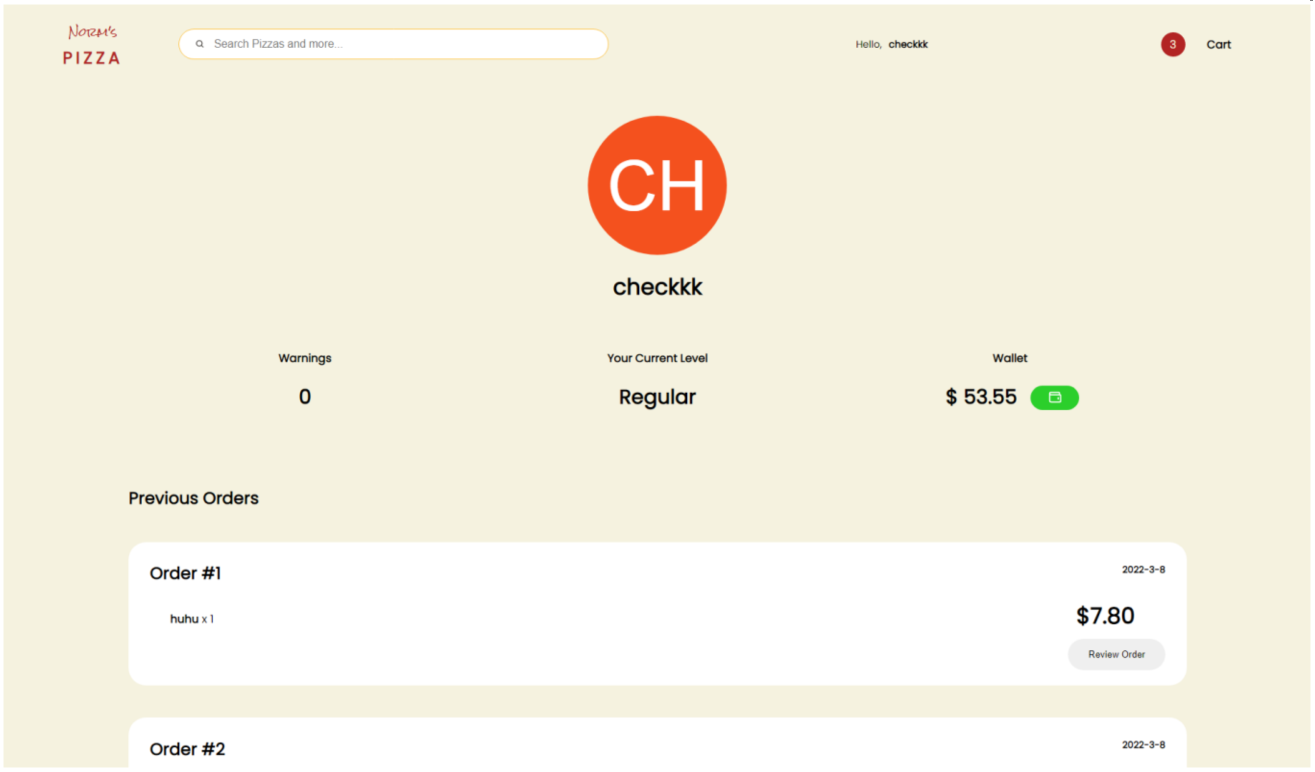
Lorem ipsum check this is a run I do not know if it will work

Rating	Price	Quantity
3.8	$3	0

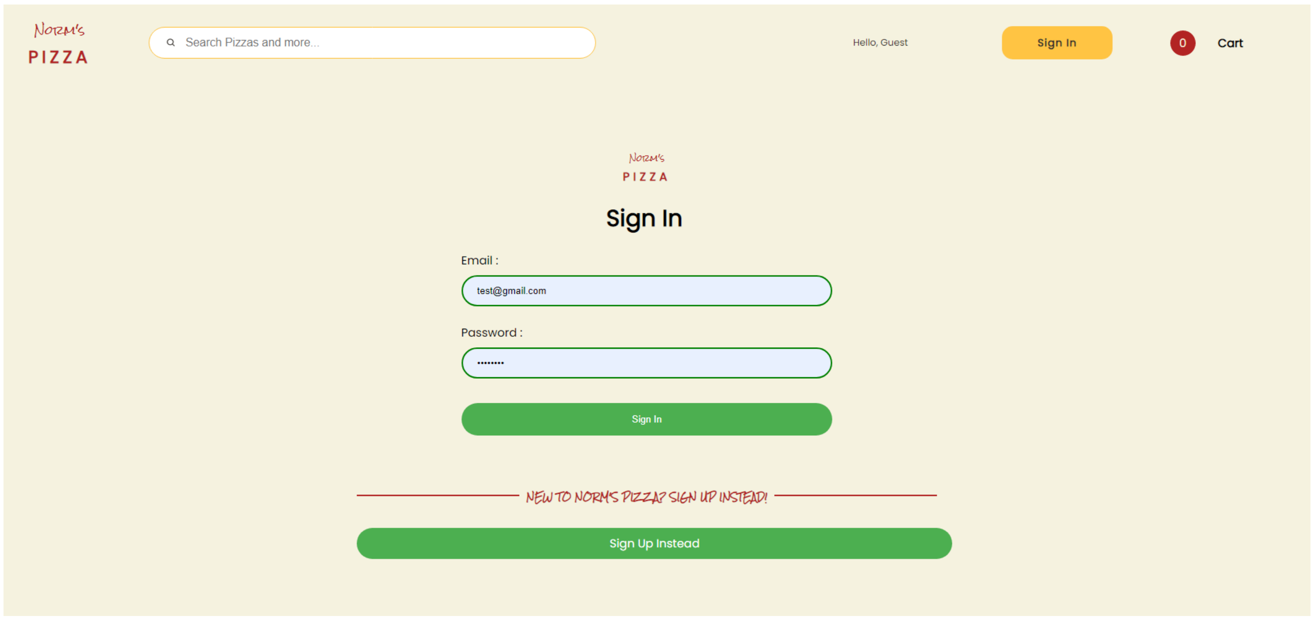
Popular Dishes

huhu

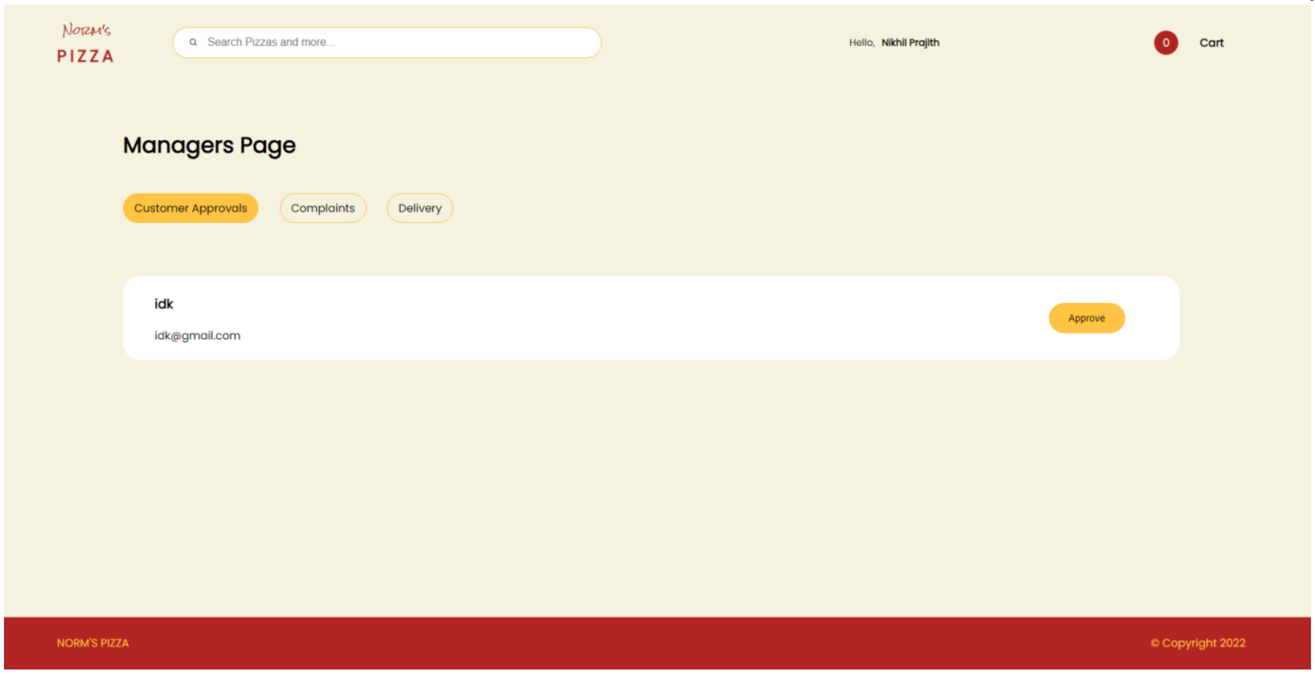
The User Profile Page



The Login and Sign Up Page



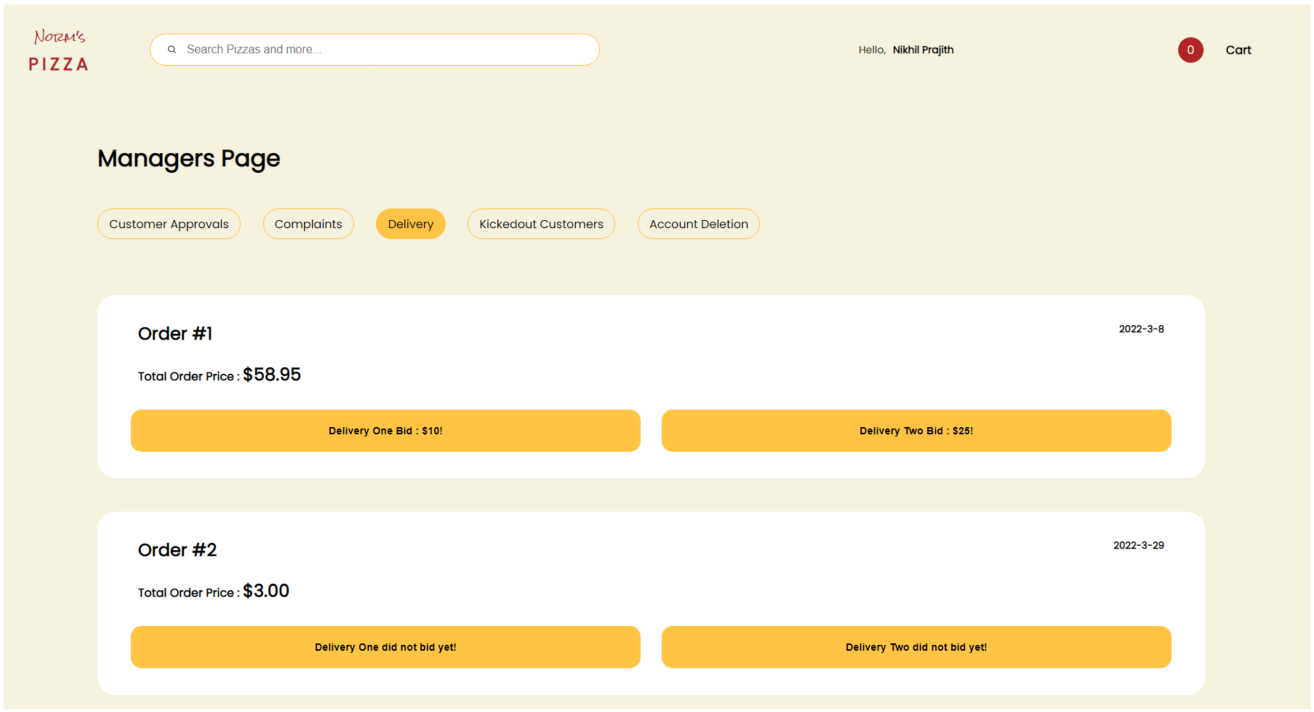
The Manager Profile Page



Managers Page

idk
idk@gmail.com

The Manage Profile Page

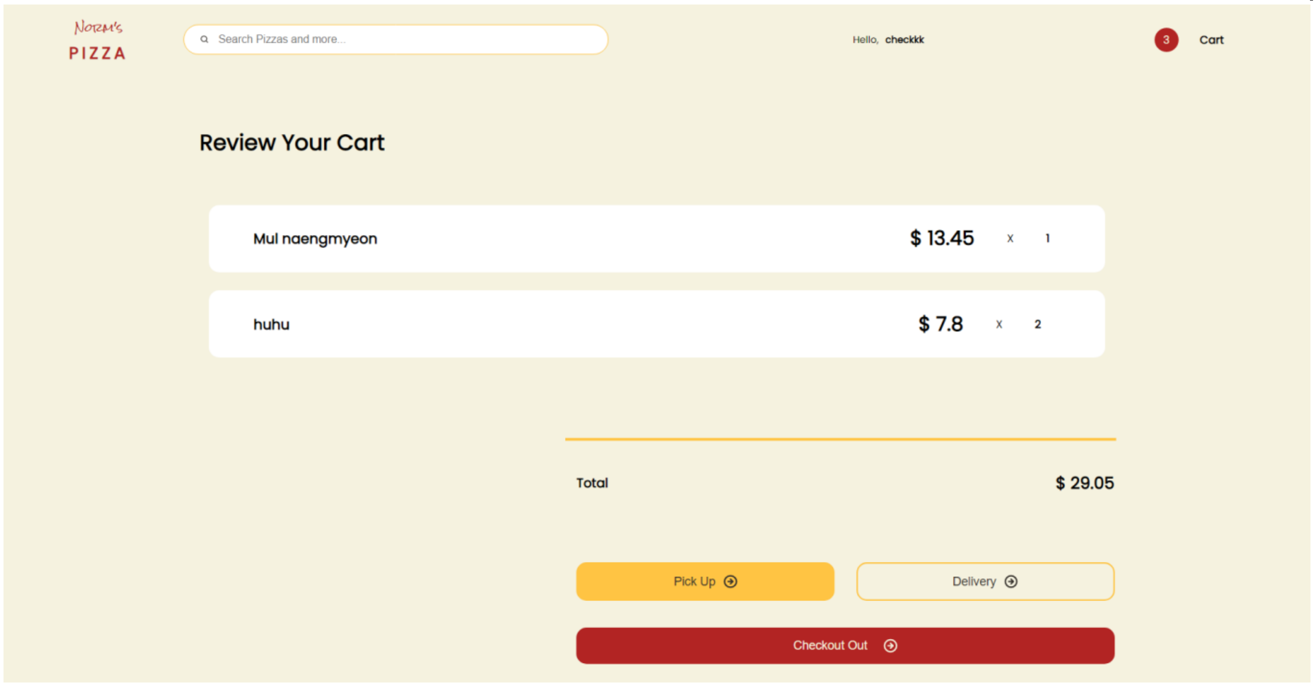


Managers Page

Order #1
2022-3-8
Total Order Price : $58.95
Delivery One Bid : $10!
Delivery Two Bid : $25!

Order #2
2022-3-29
Total Order Price : $3.00
Delivery One did not bid yet!
Delivery Two did not bid yet!

The Checkout Page

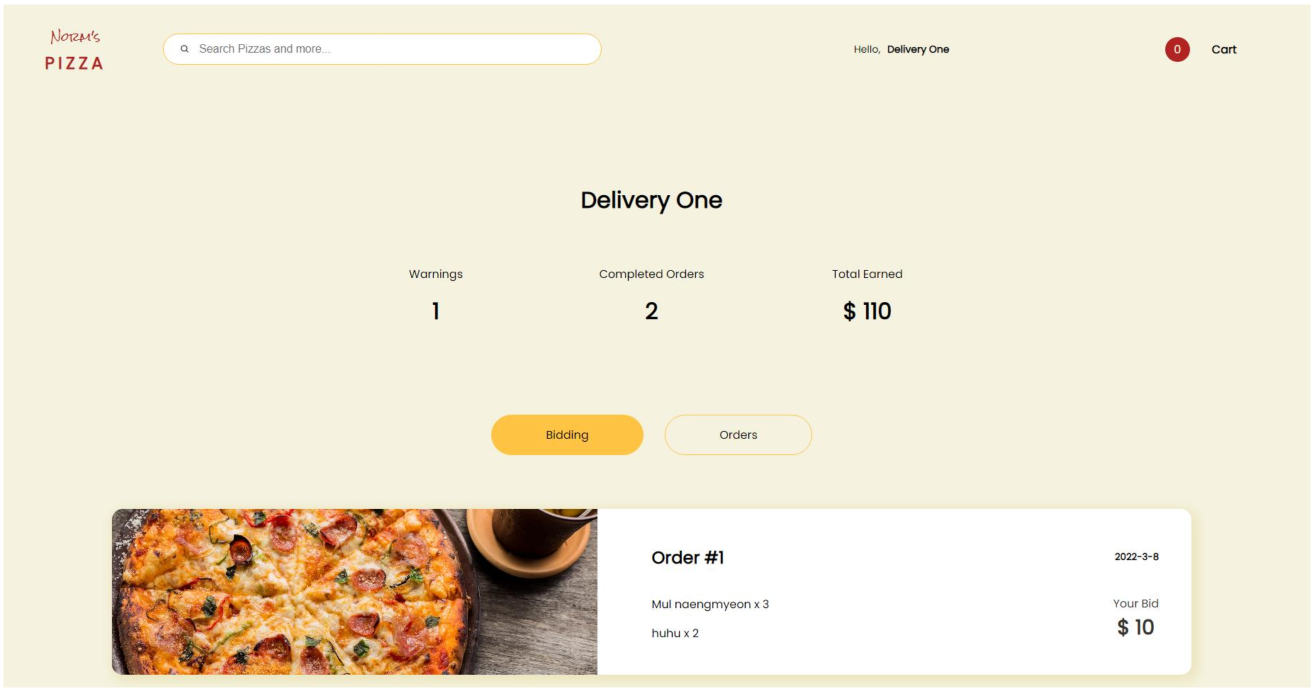


Review Your Cart

Mul naengmyeon	$ 13.45	x	1
huhu	$ 7.8	x	2

Total $ 29.05

The Delivery profile Page



Delivery One

Warnings	Completed Orders	Total Earned
1	2	$ 110

Order #1 2022-3-8

Mul naengmyeon x 3

huhu x 2

Your Bid $ 10

5.2 Sample prototype

We recorded a video of the checkout process of the app. The link to the video is provided below.
Sample_Prototype.mp4

6. Memos of group meetings

The first meeting we had was in the very beginning when the project was finalized by the professor. We met to get to know each other and figure out what language and platforms we wanted to use for the project. We decided on using Reac.js for the frontend and then Firebase Firestore for the database. Then the second meeting we had was to divide the phase 1 report to see what each person would do. We decided on the following:

Azwad Shameem : Use-Case Model Survey, & Supporting Information
Dawa Sherpa : Overview & Use-Case Reports
Jeevan Bastola : Supplementary Requirements & Assumptions and Dependencies
Ali Syed : Scope & References
Nishanth Prajith : Purpose, Definitions, Acronyms, and Abbreviations & Supporting information.

Then we had our last meeting about the phase 2 report and what each person was going to do. We decided on the following:

Azwad Shameem : Sequence class diagram, Detailed Design
Dawa Sherpa : Memos of group meetings, Use Cases
Jeevan Bastola : Use Cases
Ali Syed : ER Diagram, System Screens
Nishanth Prajith : Sequence class diagram, Detailed Design, Memos of group meetings

Throughout the whole process, we were in constant communication on our discord server, and if anybody needed anything, one or two people would try and help. If we cannot solve the problem through the chat, we will hop on a quick zoom call. We did not have any problems. Whenever we made some changes to the code, we informed everyone about the changes and even set up an automatic email system in GitHub to pull the updated code. When working with backend code, we only allowed one person to work on it simultaneously, as mistakes could be hard to fix. Overall, we have not run into any issues as of yet.

7. Github Repository Address

Address to our GitHub repository : NishanthPrajith/CSC322_Final_Project (github.com)