Design Report

for

Kitchen Zealot

Written by:
Abtahi Chowdhury
Abusaleh Masud
Safwan Shahid
Arman Uddin
Farhan Zaman

Version 2.0

November 23, 2019

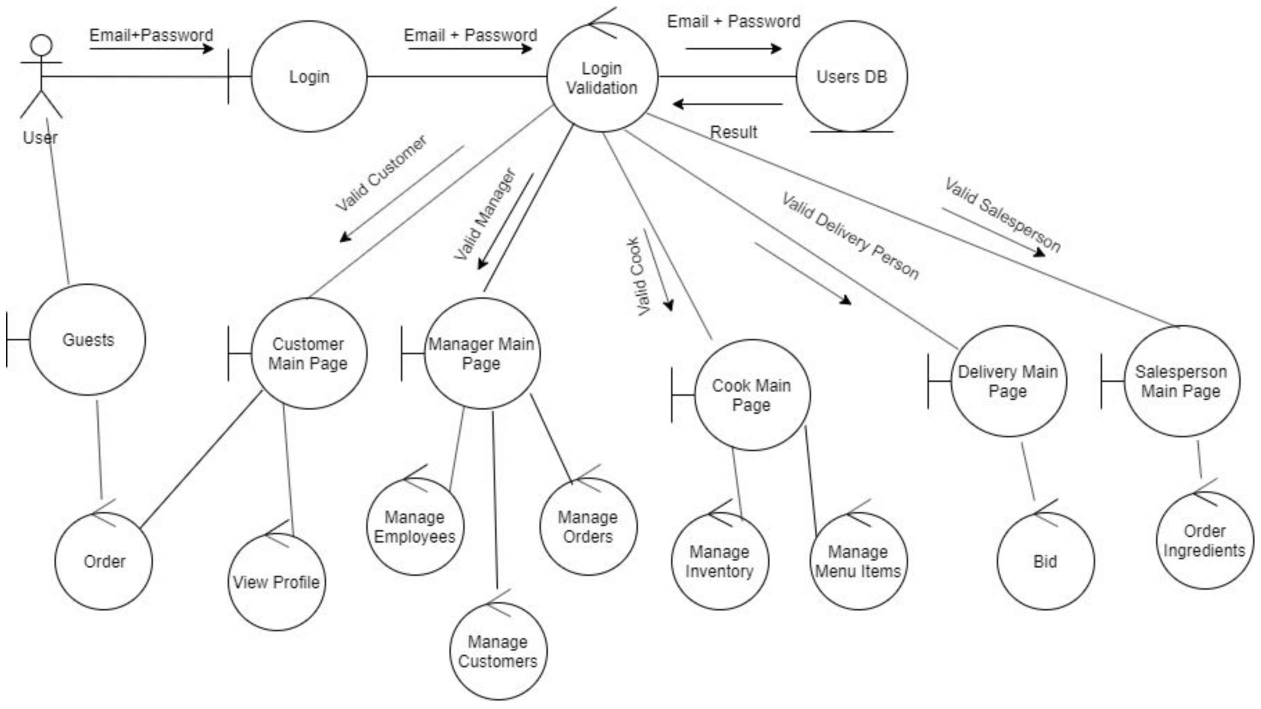
Date	Version	Description	Author
11/2/2019	1.0.0	Initial version of the online restaurant system	Abtahi Chowdhury Abusaleh Masud Safwan Shahid Arman Uddin Farhan Zaman
11/23/2019	2.0.0	This report is meant to provide the data structure and logic to carry out the functionalities dictated by the specification.	Abtahi Chowdhury Abusaleh Masud Safwan Shahid Arman Uddin Farhan Zaman

Table of Contents

1. Introduction

Below you will encounter many diagrams, charts, and pseudocode, as they will assist in breaking down the design of our product. As you continue reading the paper, any questions or confusions that arise will be answered and clarified.
Not only does it allow customers (registered users) and guests (unregistered users) to order and receive food, but also gives managers, salespeople, cooks, and delivery people access to their own page to handle services in the company. Delivery people have access to see all the different orders from customers and guests, and bid on them. Salespeople are given comments from cooks, to know and order ingredients that are needed. Cooks are allowed to request more supplies from salespeople, rate salespeople, and change menu items. Managers can approve guest to customers, view order history, view all ratings, start delivery bidding process per order, pay employees, hire/fire employees, and remove warnings.

The collaboration class diagram below gives an overview on the entire Kitchen Zealot system.



DB: Database

2. Design

2.1 Use Case Scenarios

View Menu
Normal Scenario 1. Guests, or customers click “Menu” on the homepage to see menu. Customers or Guests who are logged in to the site, can click “Order” to view the menu to order.

Login
Normal Scenario 1. User clicks “Login” button in the navigation bar 2. User is asked to enter their email and password 3. User inputs appropriate information 4. Information is checked against database for authenticity 5. User is logged in
Exceptional Scenario 1. User clicks login and enters credentials 2. Credentials are not valid 3. Error message is shown 4. User is asked to try again

Continue as Guest
Normal Scenario 1. User clicks on “Login” and then “Continue as Guest” 2. User inputs their phone number and address when ordering
Exceptional Scenario 1. User clicks on “Login” and then “Continue as Guest” 2. User inputs invalid phone number and/or address 3. Input boxes will highlight red to indicate an error

Register

Normal Scenario

1. Guest clicks login
2. Guest clicks “Sign Up”
3. Guest is asked to enter “Name”, “Email”, “Password”, “Phone Number”
4. Guest clicks “Register”
5. A request is sent to the manager to approve
6. Manager approves or disapproves new
7. If approved, then guest info is sent to the database to be added

Exceptional Scenario

1. Guest clicks login
2. Guest clicks “Sign Up”
3. Guest is asked to enter “Name”, “Email”, “Password”, “Phone Number”
4. Guest enters an email already in use
5. Guest clicks “Register”
6. An error message pops up stating the email is already used

Add/Remove Products to Cart

Normal Scenario

1. Customer clicks on order
2. Customer adds an item to the cart with the quantity they want

Order Food

Normal Scenario

Scenario 1: Customer

1. Customer clicks “Order”
2. Customer views menu and clicks “Add to Cart”
3. Customer adjusts quantity
4. Customer click “Check Out”

Scenario 2: Guest

1. Guest clicks “Menu”
2. Guest adds an item to the cart with the quantity they want
3. Guest clicks “Check Out” to place order
4. Guest is prompted to enter their details: Name, Email, Phone Number, Payment Details

View All Placed Orders

Normal Scenario

1. Customer clicks on “My Orders” to view all placed orders

Rate Food/Delivery

Normal Scenario

1. Customers can rate the food(cooks) and the delivery people individually
2. Customer Views order
3. Rate Food
4. Rate Delivery

Bid on Deliveries

Normal Scenario

1. Delivery person can access the pending bid page via a button on the post-login page
2. Delivery person chooses from a list of pending bids
3. Delivery person places bids on that selected order
4. After 15 minutes time runs out, bid is over

Exceptional Scenario

1. Two users input the same bid at the same time

Give Customers Rating

Normal Scenario

1. Delivery person can access the past delivered ordered in the “Delivered Orders” tab
2. Delivery person can view orders by clicking “View” next to the respective order
3. Delivery person can give customers ratings from 1 to 5 and an optional comment

Exceptional Scenario

1. Can’t give a rating if they view the rating they received from said customer

View Delivered Orders
Normal Scenario 1. Delivery person can access the past delivered ordered in the “Delivered Orders” tab

Modify Menu Items
Normal Scenario 1. Cooks can access the “Modify Menu Page” by clicking a button in the navbar 2. Cook choose a menu item they wish to change or add a new menu item
Exceptional Scenario 1. Two cooks may try to delete the same item at the same time

Request Supplies
Normal Scenario 1. Cooks can click a “Request Supplies” button on the navigation bar 2. On the Request Supplies Page, the cook can order the desired supplies by clicking “Order”

Rate Supplies
Normal Scenario 1. Cooks can click “Rate Past Orders’ 2. Cooks can click “View” on a past order 3. Cooks can rate from 1 to 5, and write an optional comment

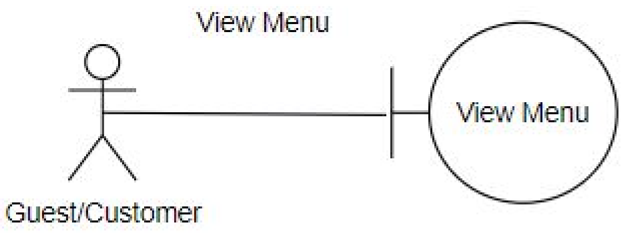
Purchase Supplies
Normal Scenario 1. Salesperson click on “Purchase Supplies” 2. Salesperson can open a list of items and select which items and in what quantity to purchase

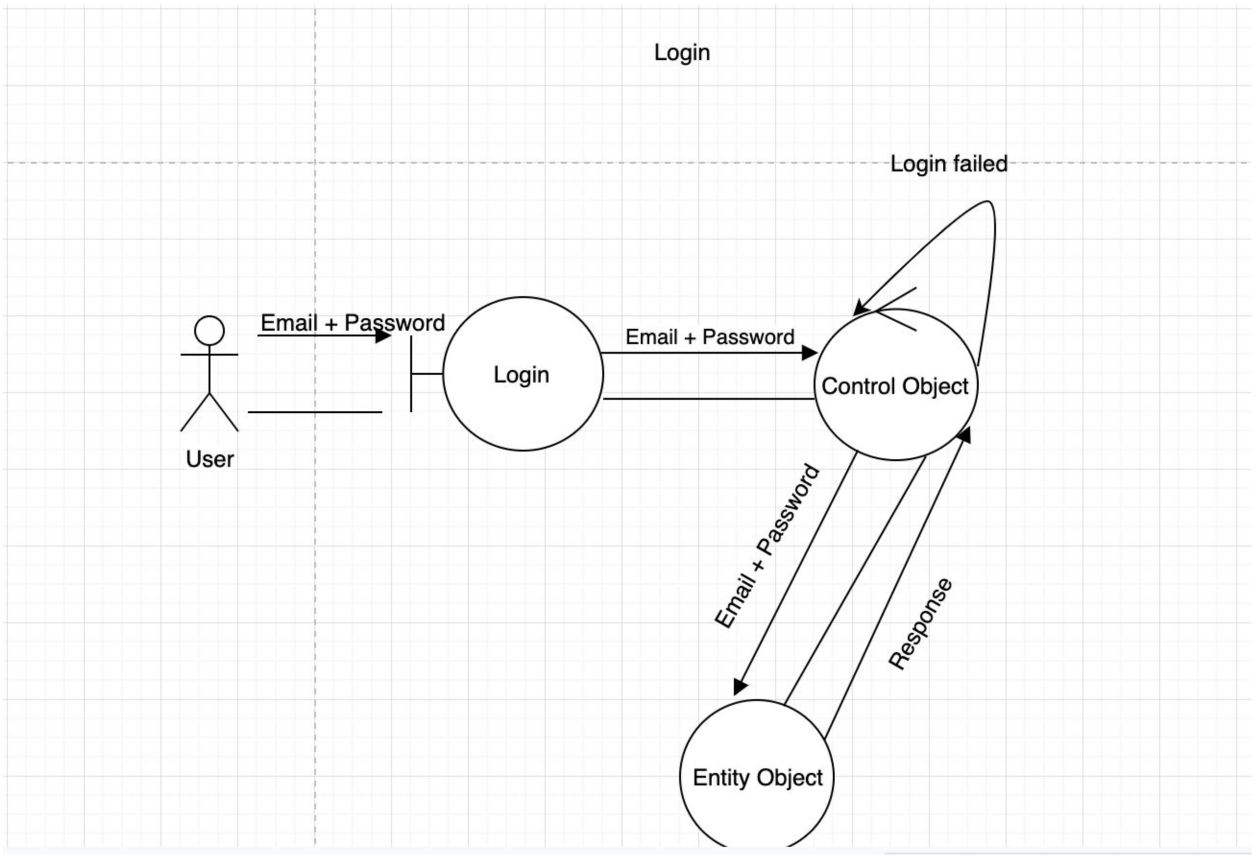
Start Bids
Normal Scenario 1. Managers can click “Manage Orders” to view all orders 2. Managers can then click an order to view order details and start bids if appropriate
Exceptional Scenario 1. No current orders

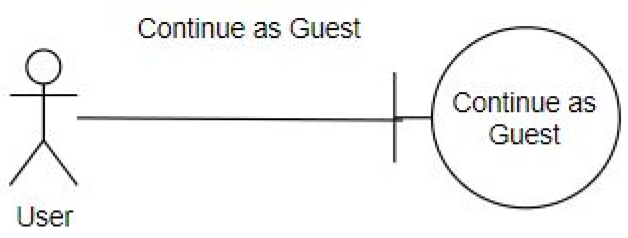
Manage Employees
Normal Scenario 1. Managers can click the “Employees” tab 2. Managers can click “View” next to any Employee to view employee details 3. Managers can hire new employees by manually adding the required information
Exceptional Scenario 1. May go below minimum number of employees

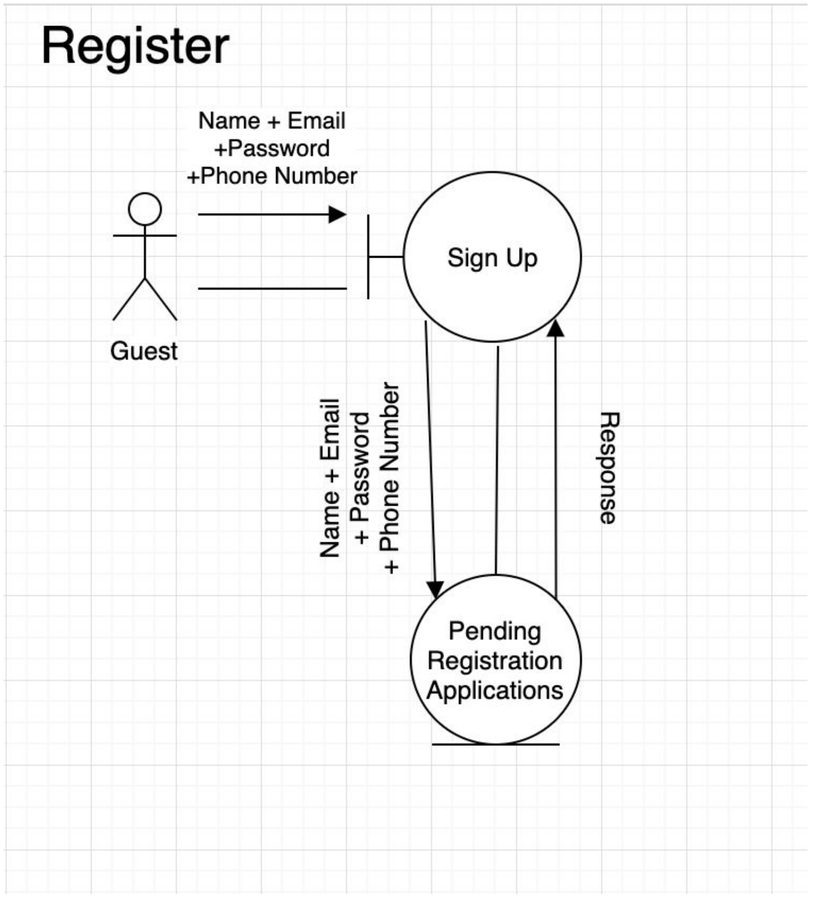
Approve Registration
Normal Scenario 1. Manager clicks “Pending Registrations” 2. Manager can click “View” next to applicants name 3. Manager clicks “Yes” / “No” next to each new guest
Exceptional Scenario 1. Guest may not have order history

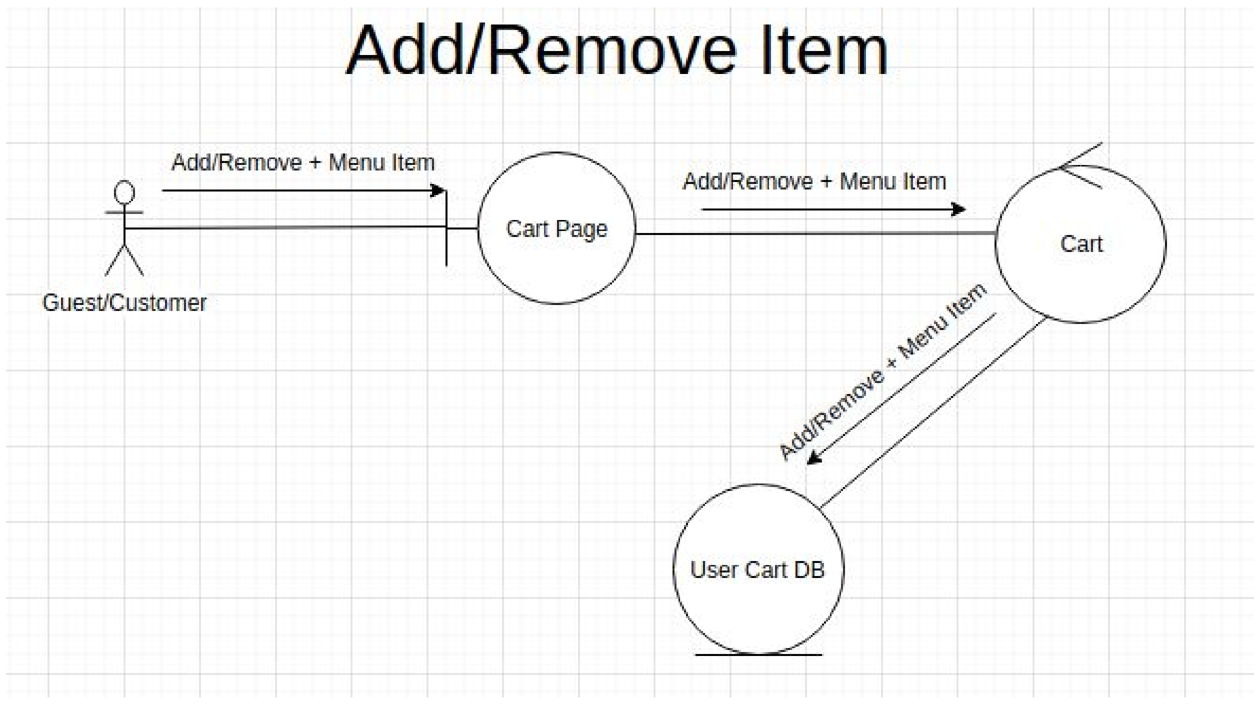
2.2 Collaboration Diagrams



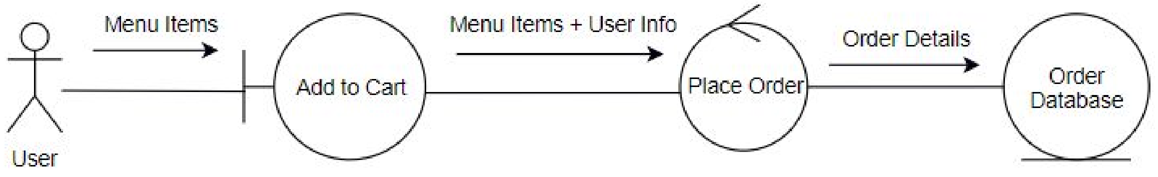



Continue as Guest
User
Continue as Guest

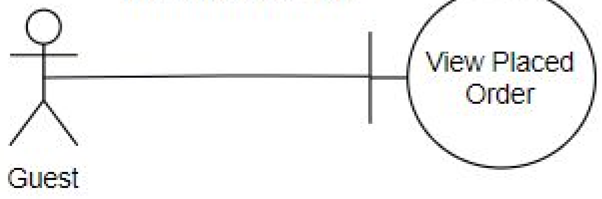

Register
Name + Email +Password +Phone Number
Guest
Sign Up
Name + Email + Password + Phone Number
Response
Pending Registration Applications

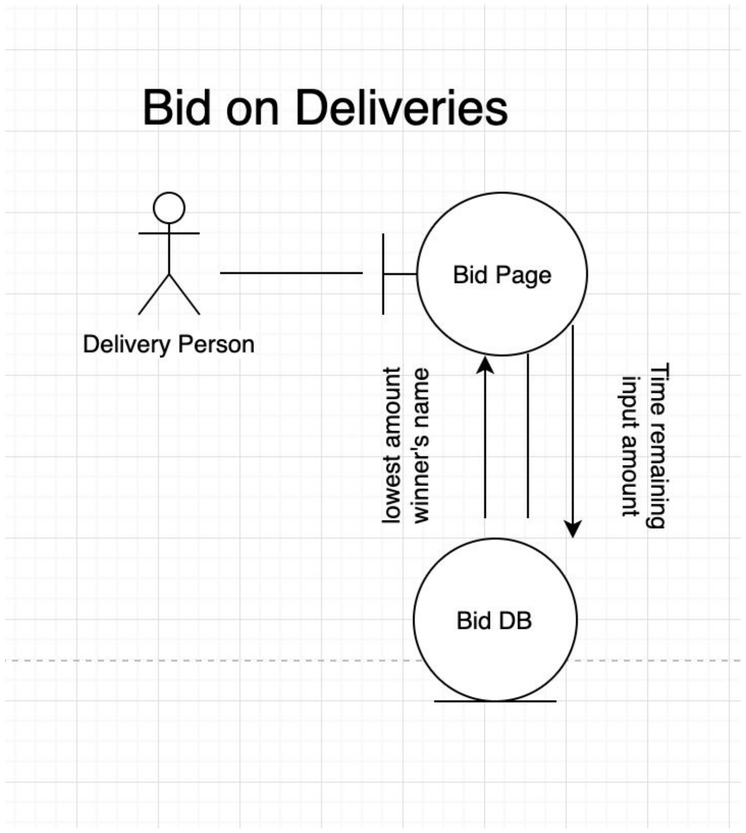


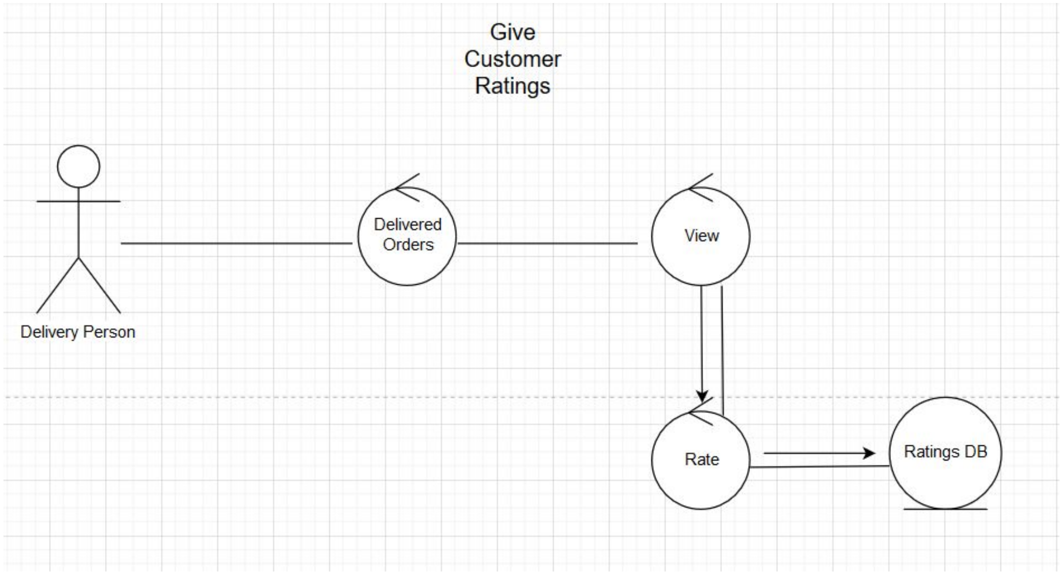
ORDER

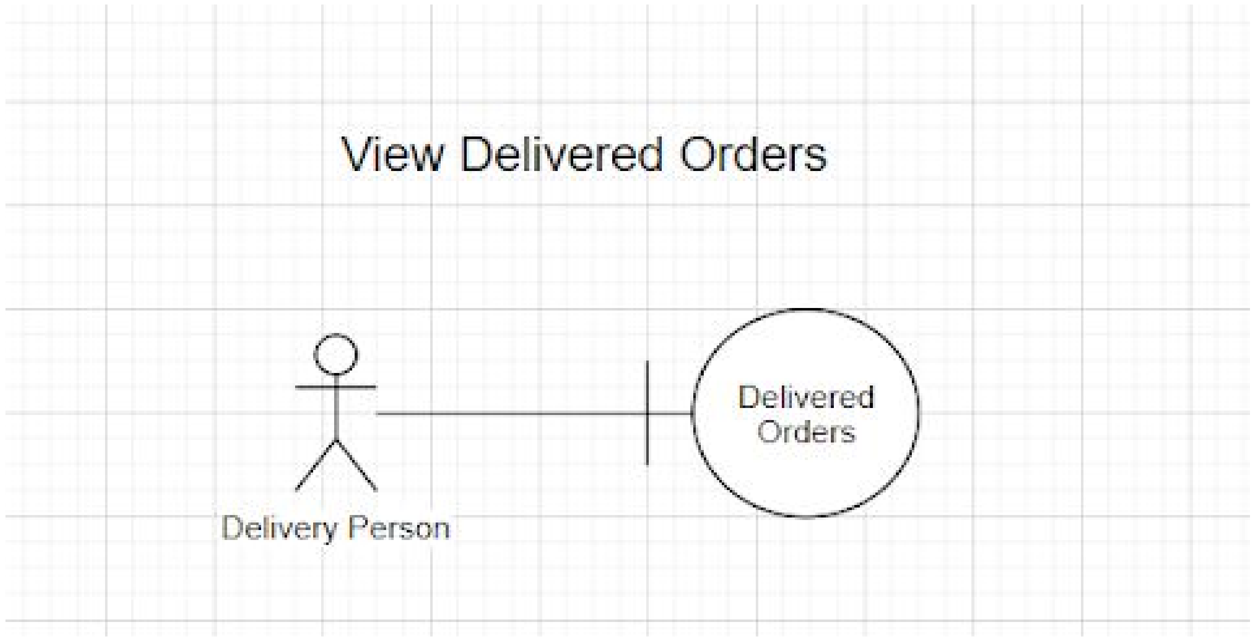


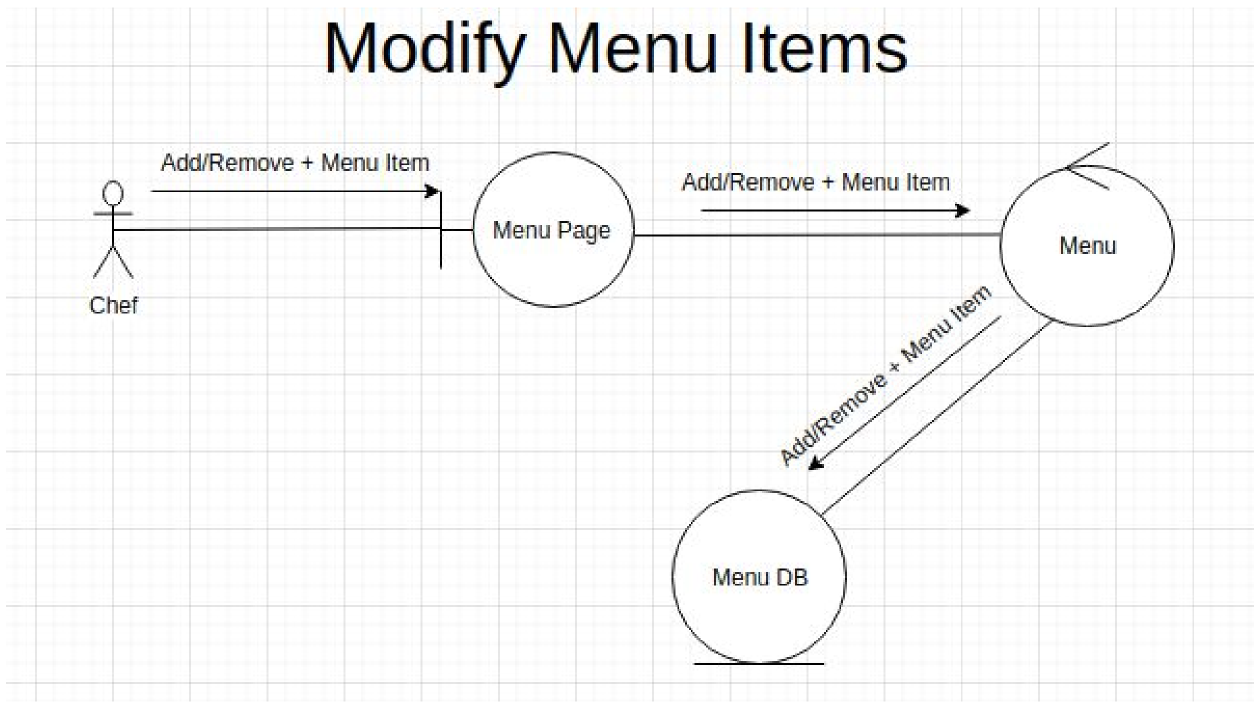
View Placed Orders

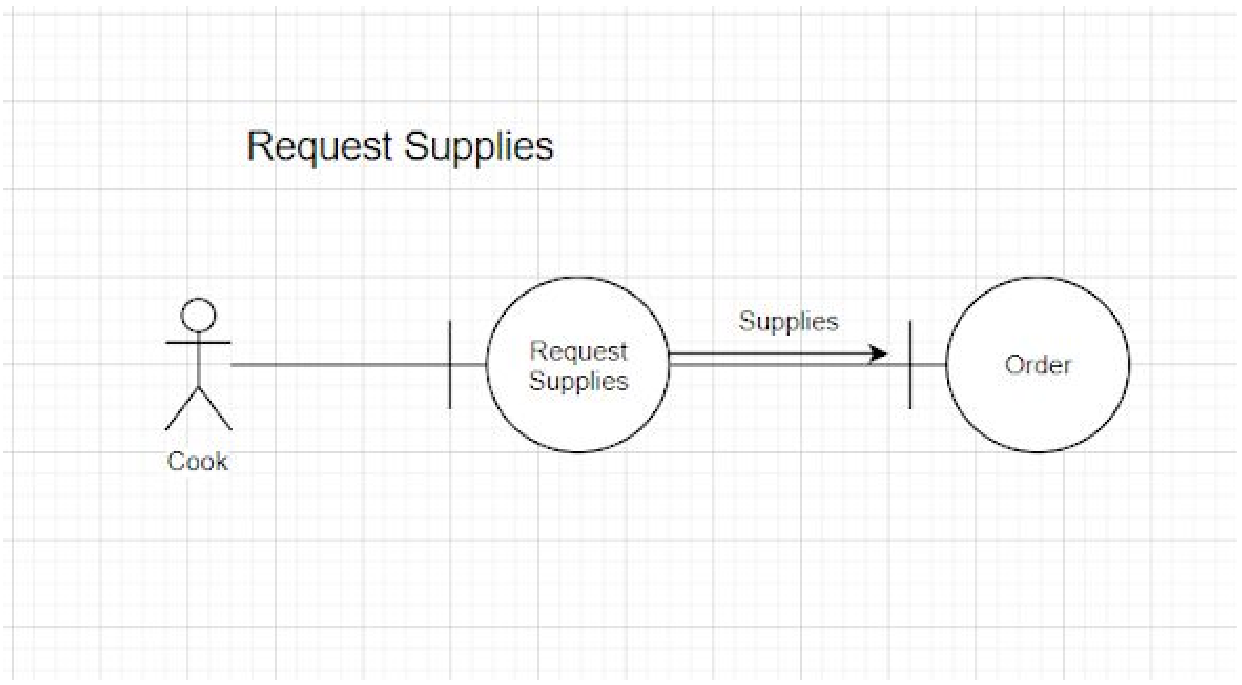


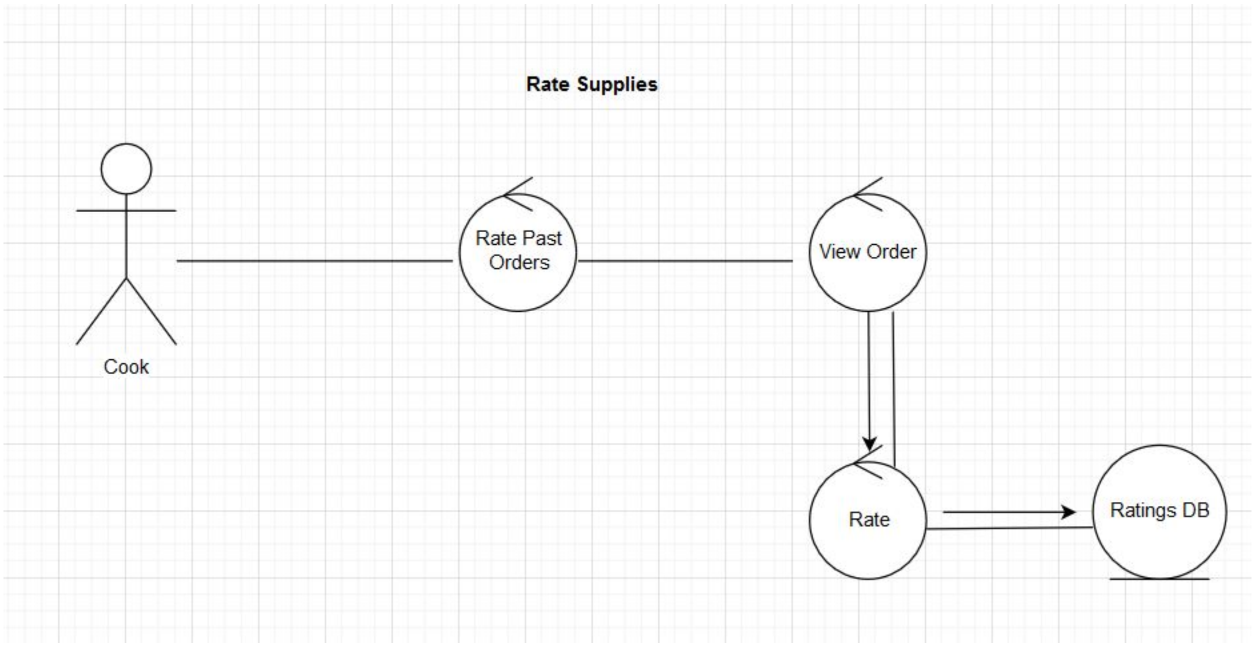

Bid on Deliveries
Delivery Person
Bid Page
lowest amount
winner's name
Time remaining
input amount
Bid DB

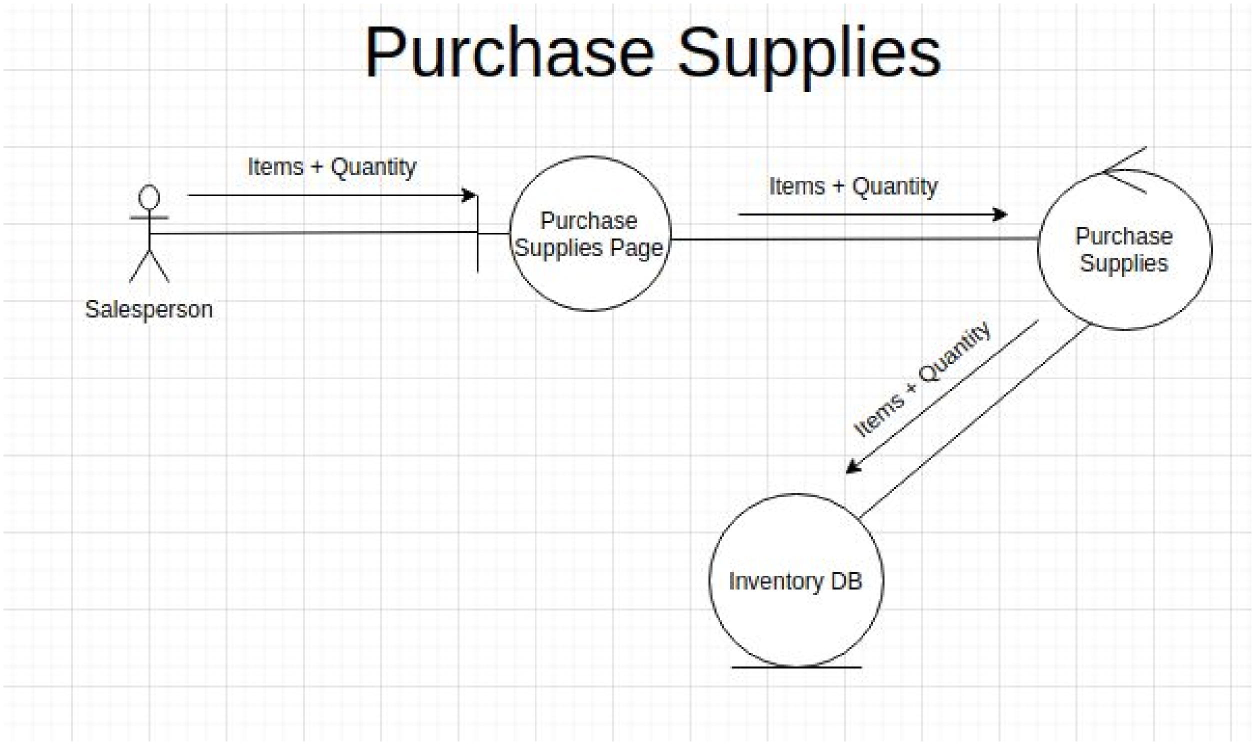

Give Customer Ratings
Delivery Person
Delivered Orders
View
Rate
Ratings DB

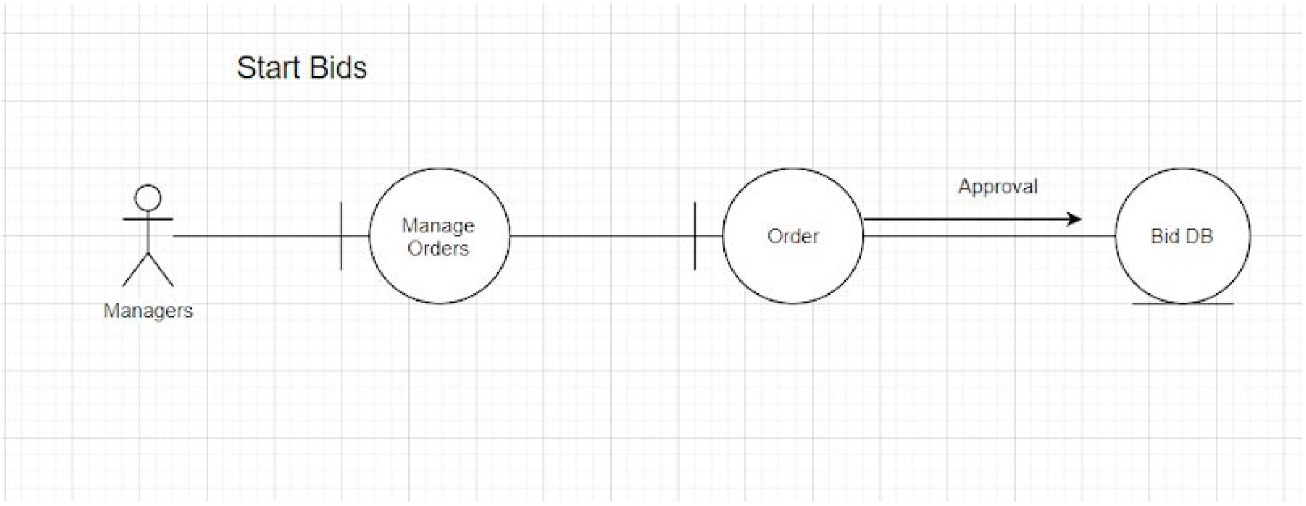

View Delivered Orders
Delivered Orders
Delivery Person

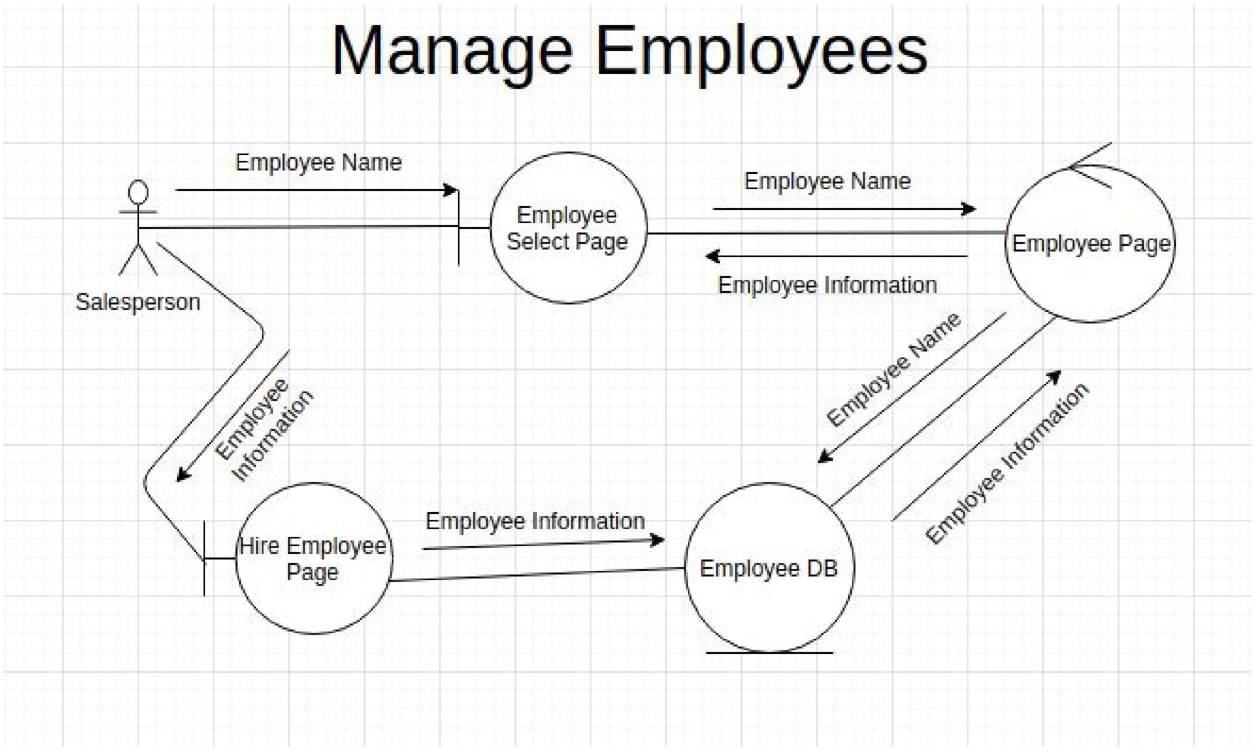

Modify Menu Items
Add/Remove + Menu Item
Menu Page
Add/Remove + Menu Item
Menu
Chef
Add/Remove + Menu Item
Menu DB

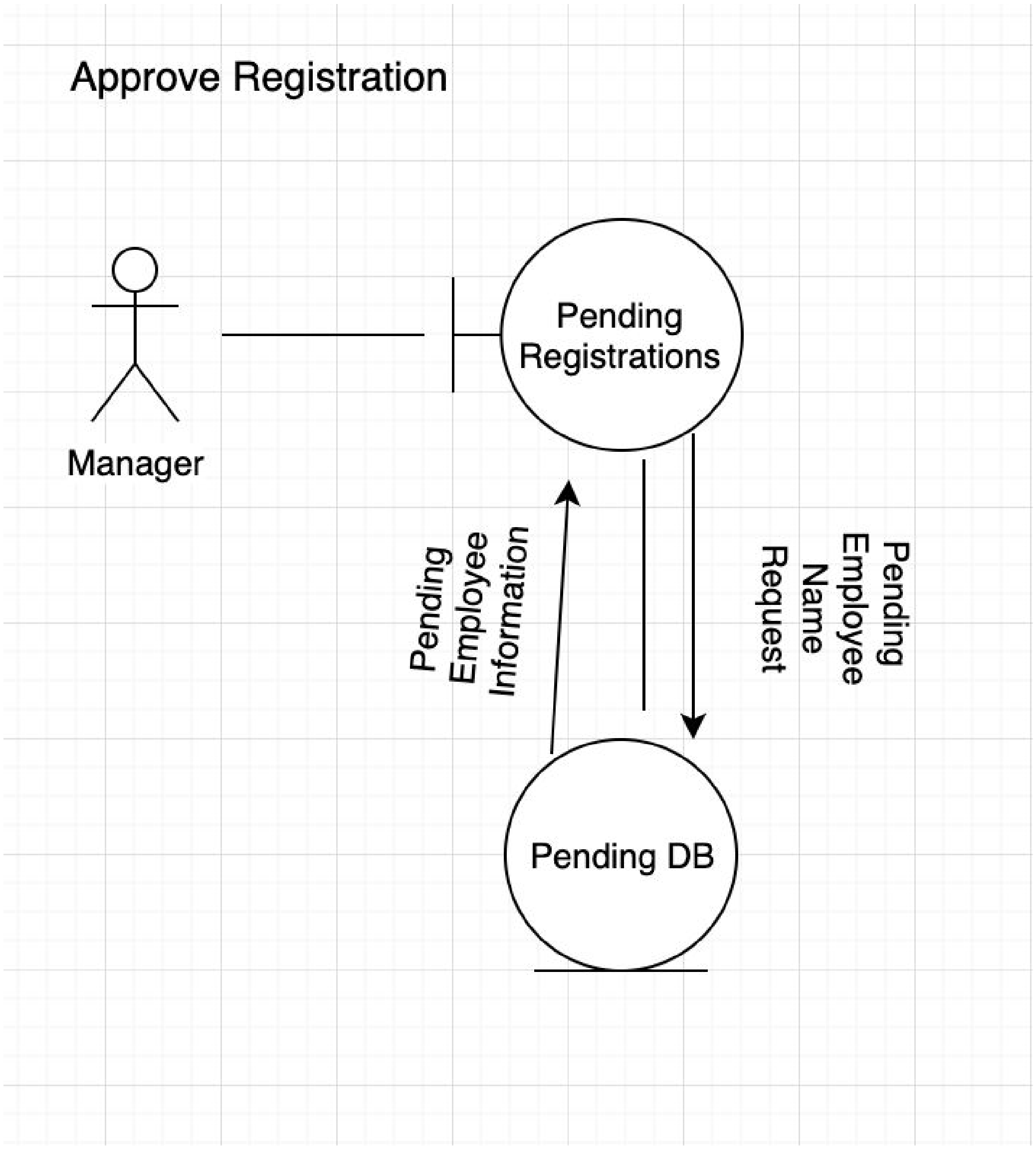

Request Supplies
Cook
Request Supplies
Supplies
Order


Rate Supplies
Cook
Rate Past Orders
View Order
Rate
Ratings DB

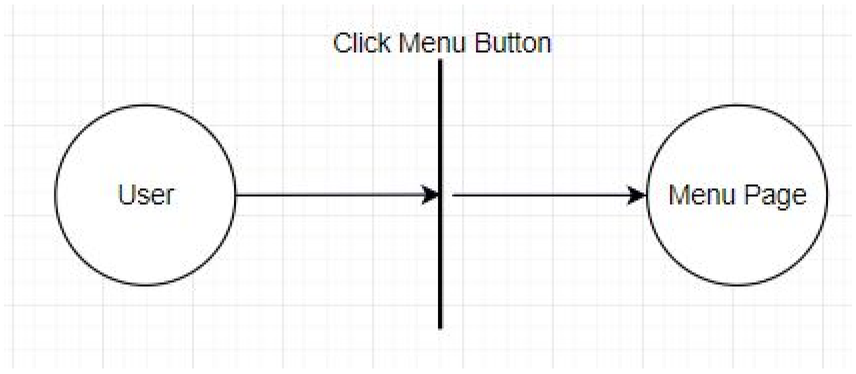

Purchase Supplies
Items + Quantity
Items + Quantity
Salesperson
Purchase Supplies Page
Purchase Supplies
Items + Quantity
Inventory DB

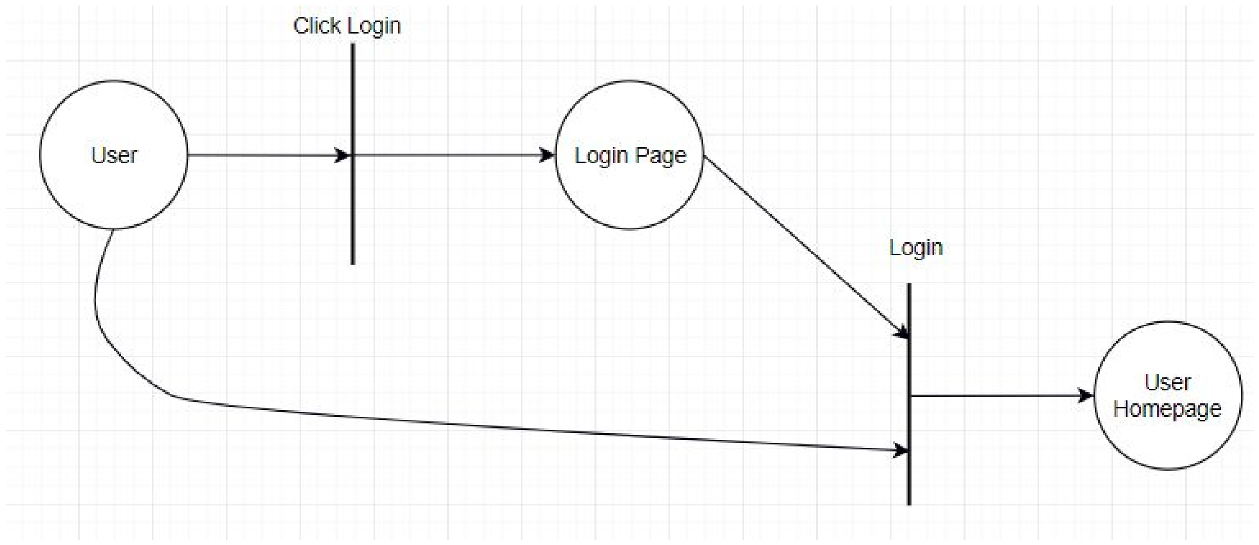

Start Bids
Approval
Manage Orders
Order
Bid DB
Managers

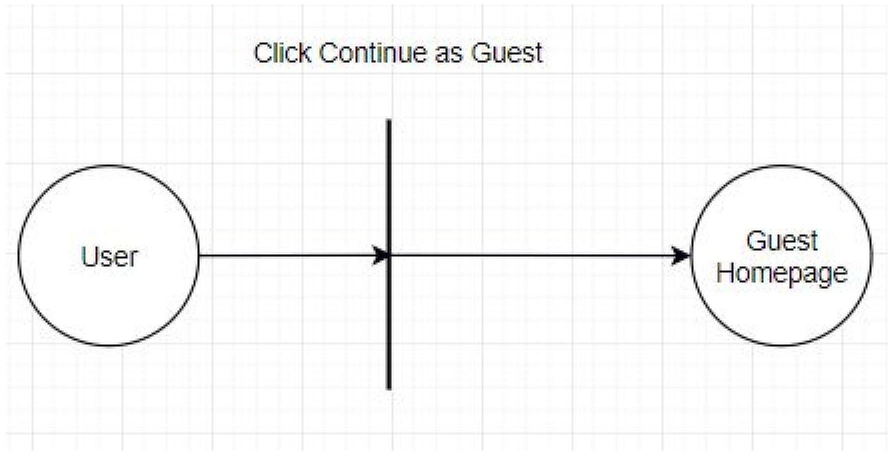

Manage Employees
Employee Name
Salesperson
Employee Select Page
Employee Name
Employee Page
Employee Information
Employee Name
Employee Information
Employee Information
Hire Employee Page
Employee Information
Employee DB
Employee Information

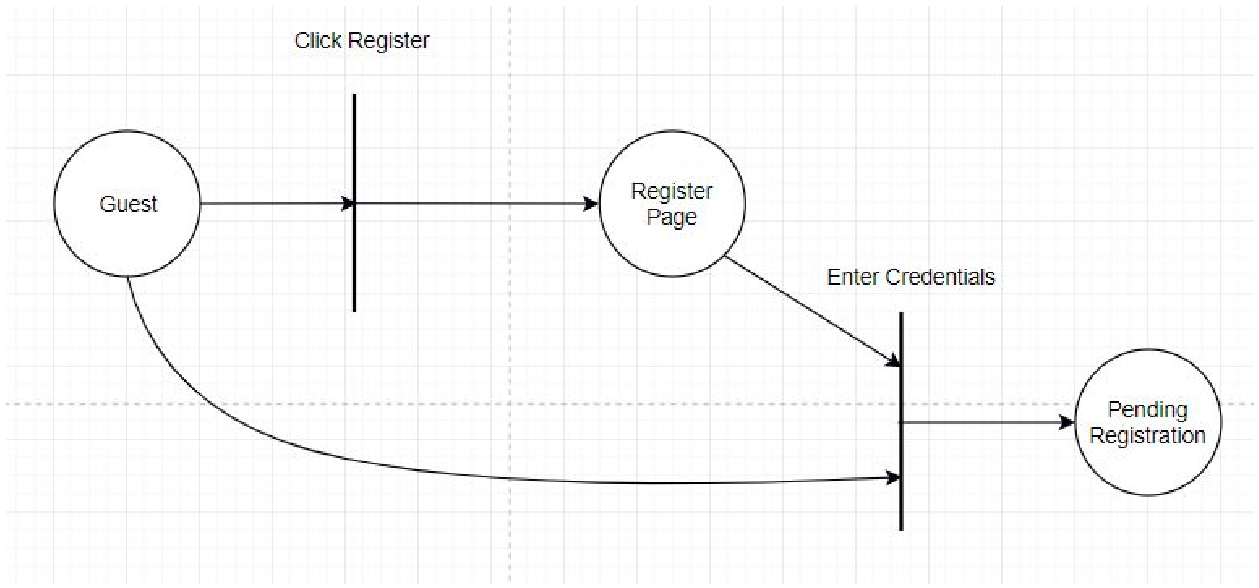

Approve Registration
Manager
Pending Registrations
Pending Employee Information
Pending Employee Name Request
Pending DB

2.3 State Diagrams/ Petri-net

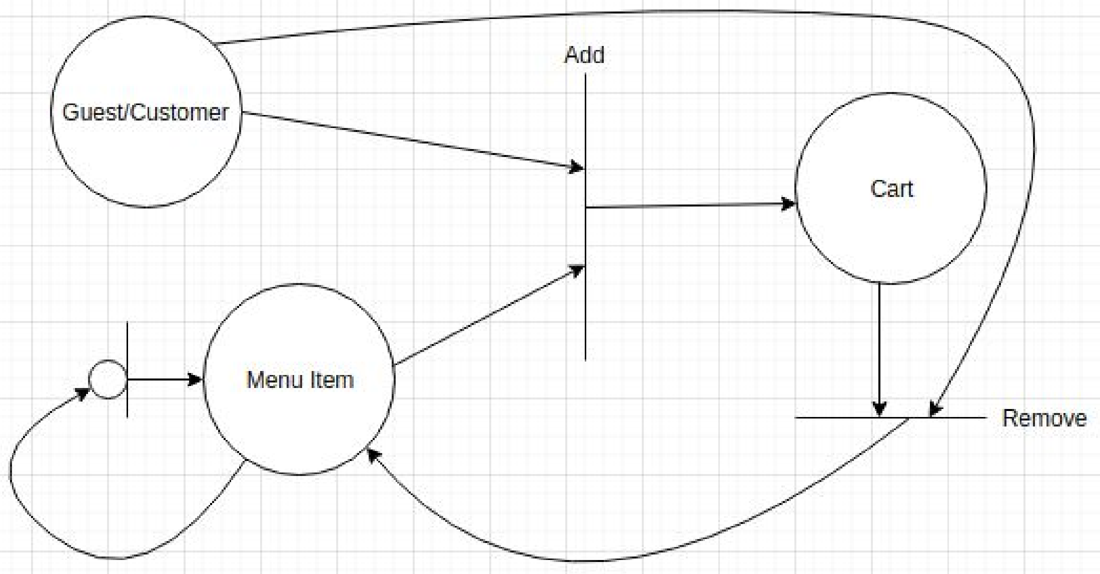


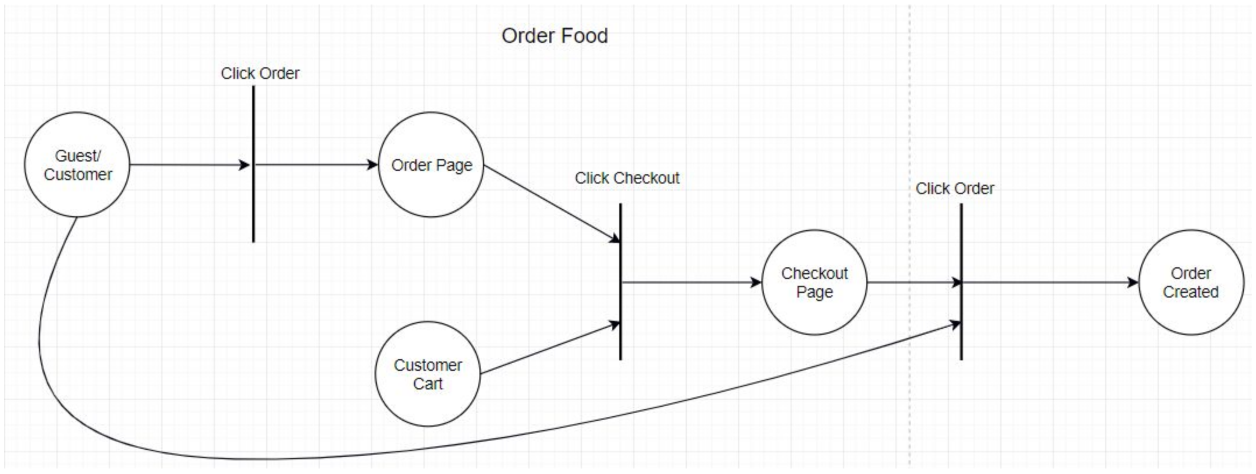


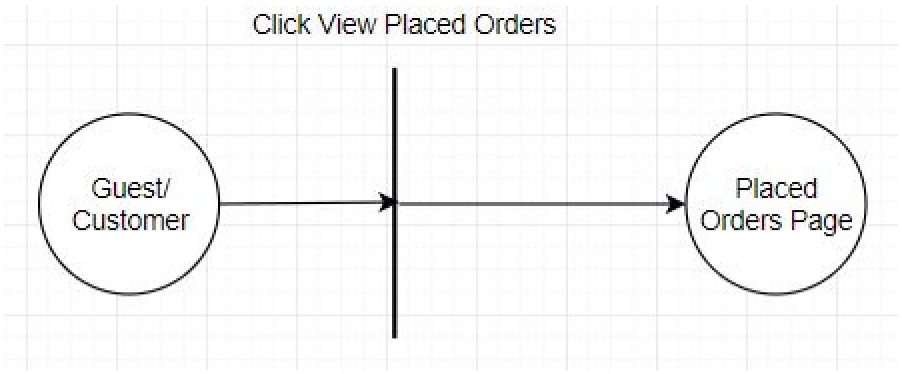


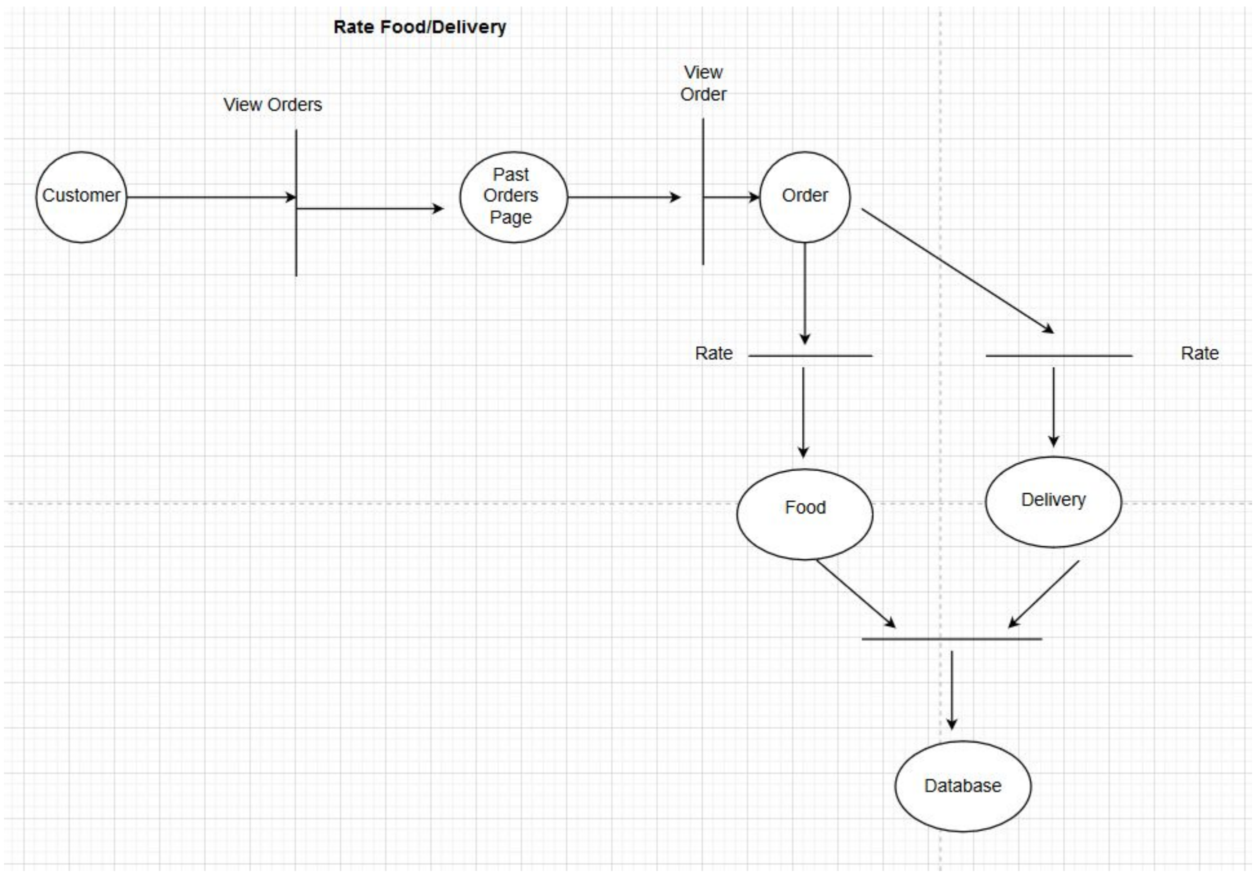


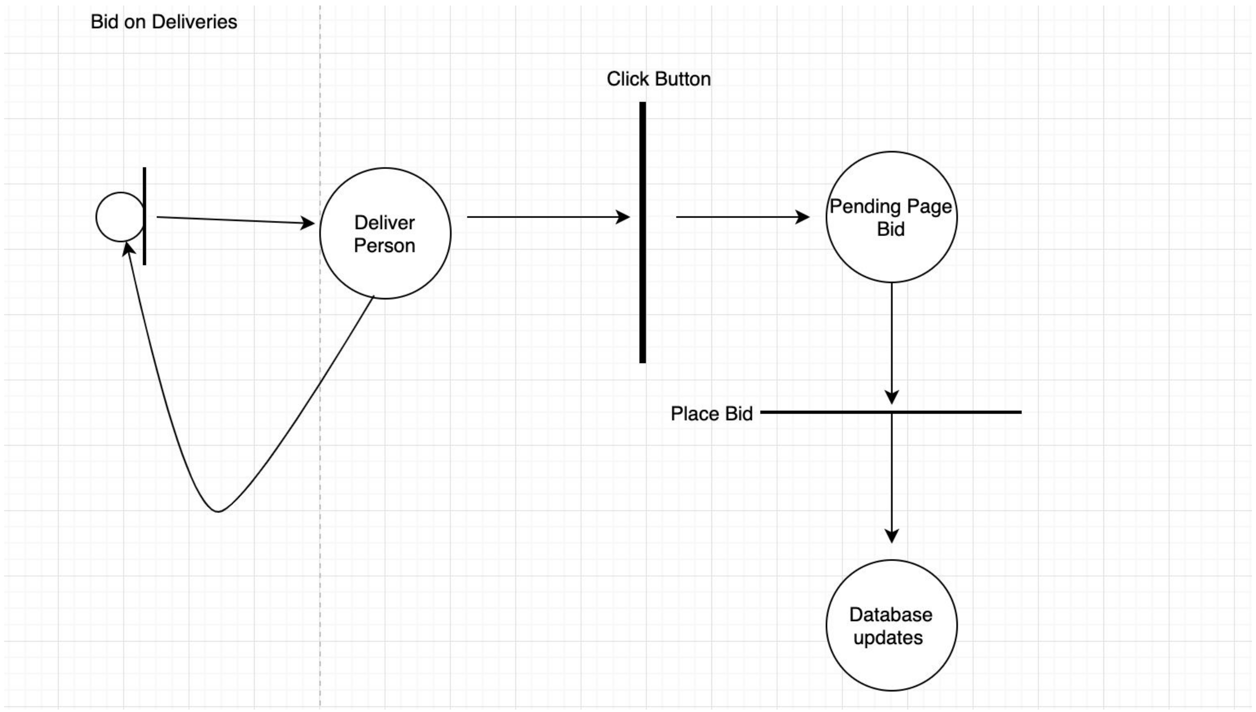
Add/Remove Products to Cart

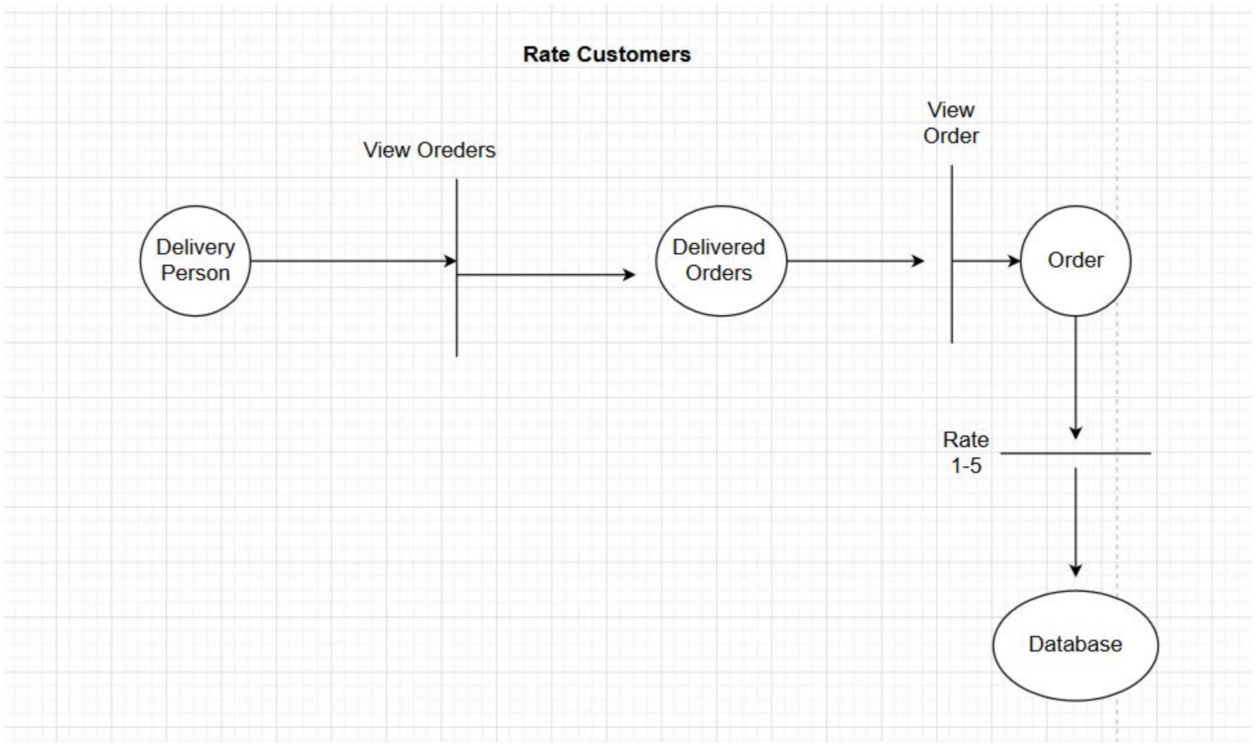


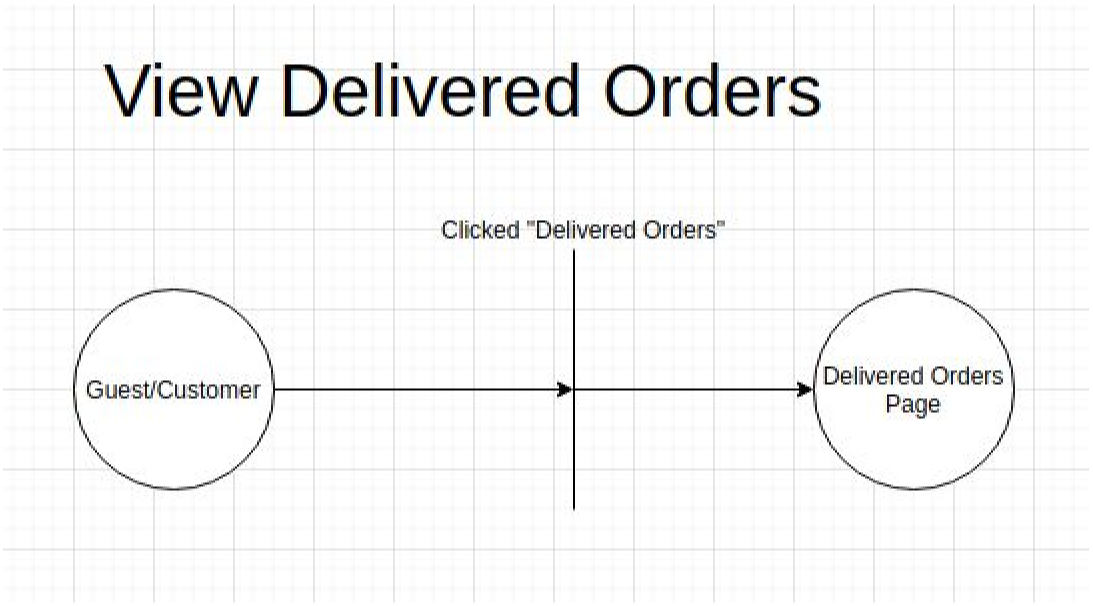

Order Food
Click Order
Guest/ Customer
Order Page
Click Checkout
Customer Cart
Checkout Page
Click Order
Order Created

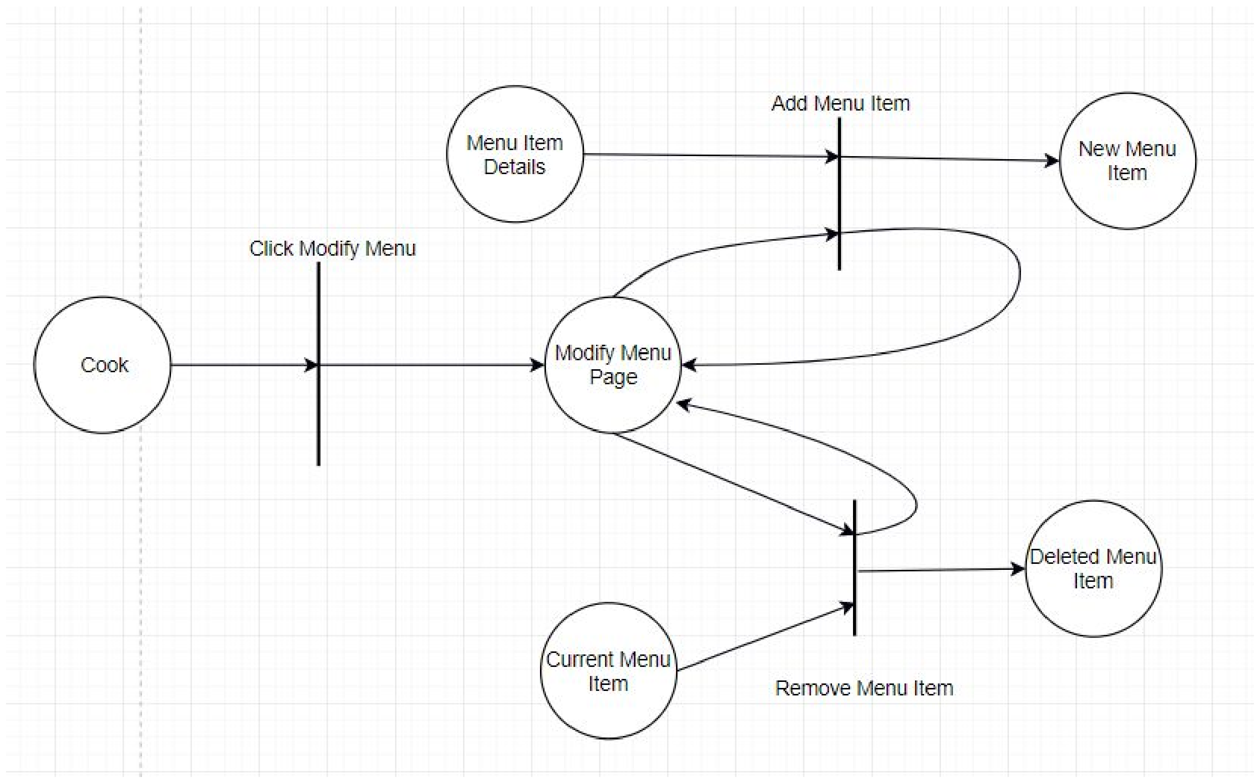

Click View Placed Orders
Guest/ Customer
Placed Orders Page

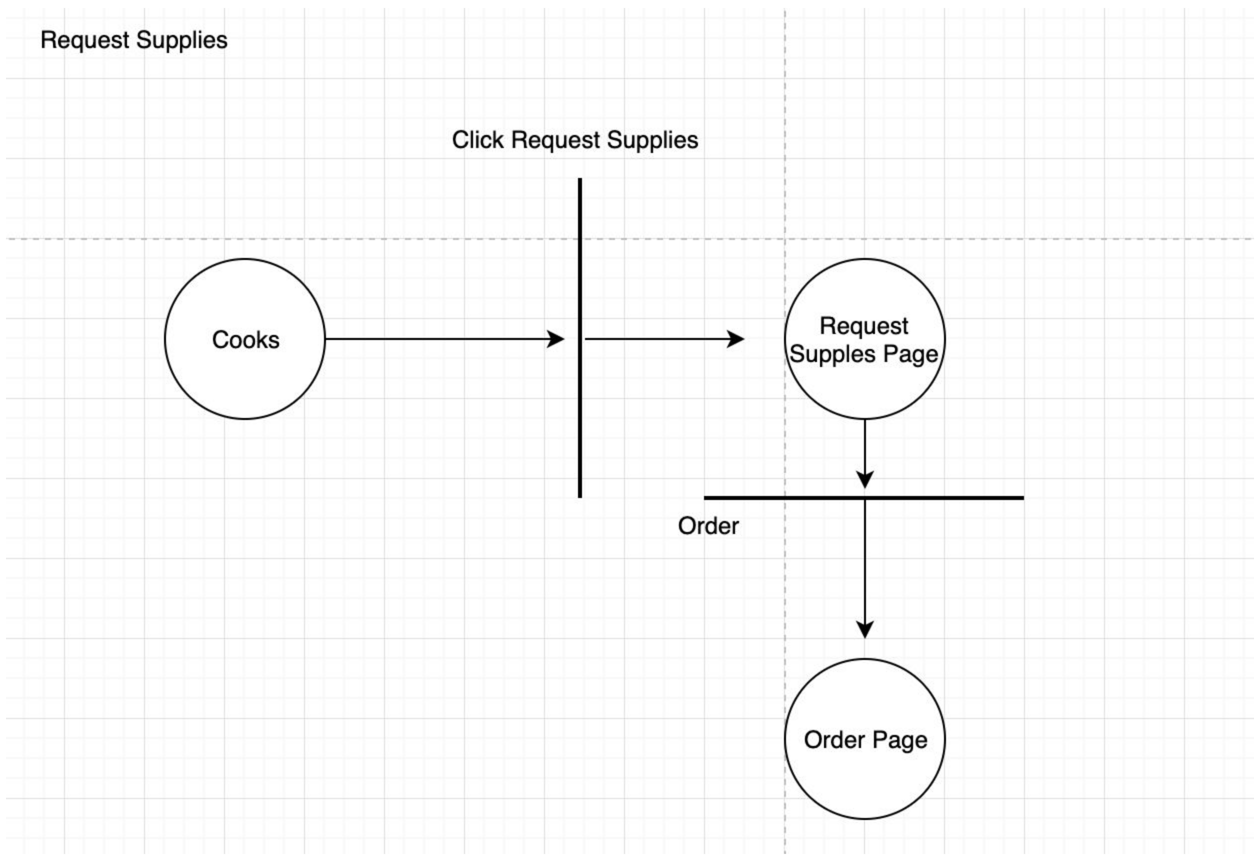

Rate Food/Delivery
View Orders
View Order
Customer
Past Orders Page
Order
Rate
Rate
Food
Delivery
Database

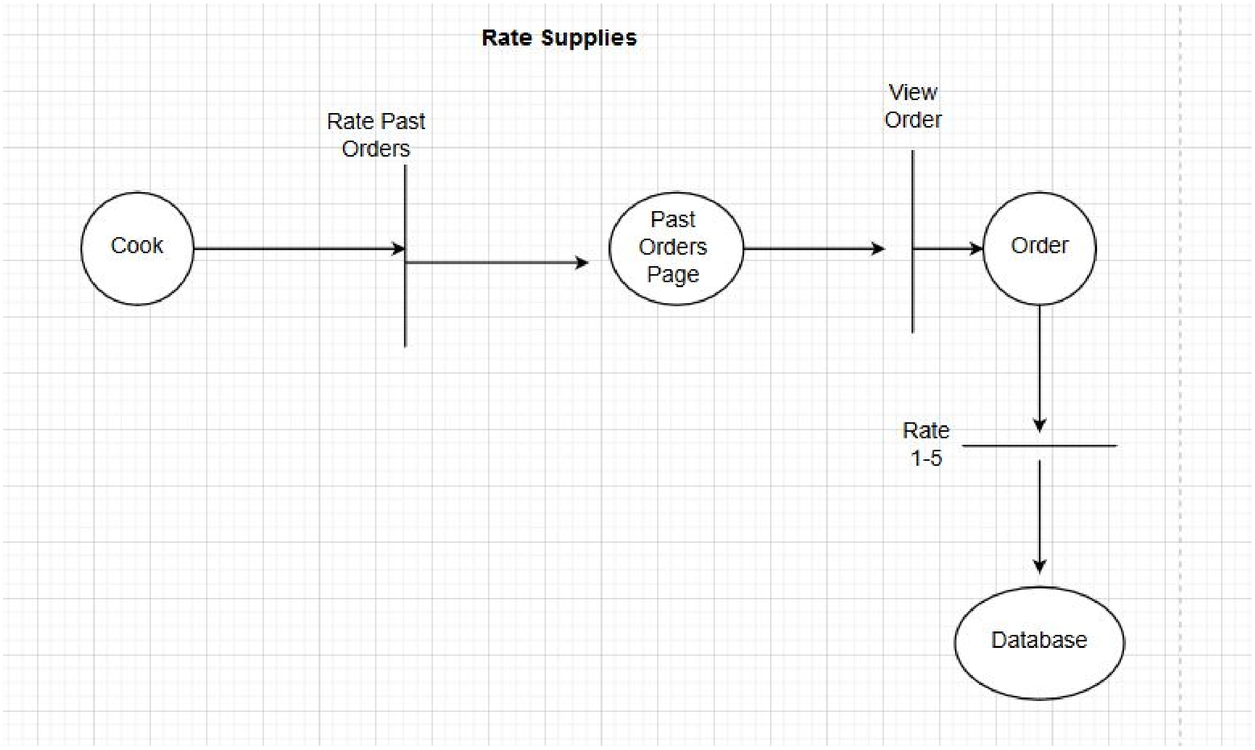

Bid on Deliveries
Click Button
Deliver Person
Pending Page Bid
Place Bid
Database updates

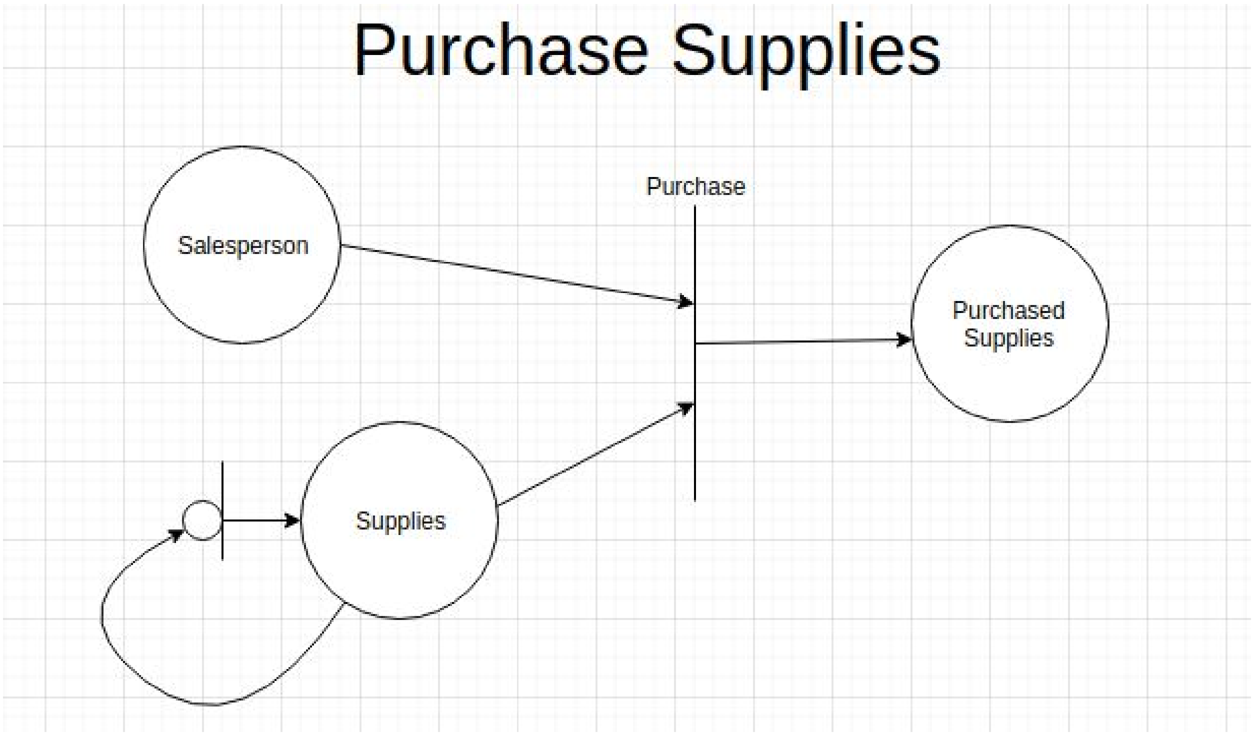

Rate Customers
View Oreders
View Order
Delivery Person
Delivered Orders
Order
Rate 1-5
Database

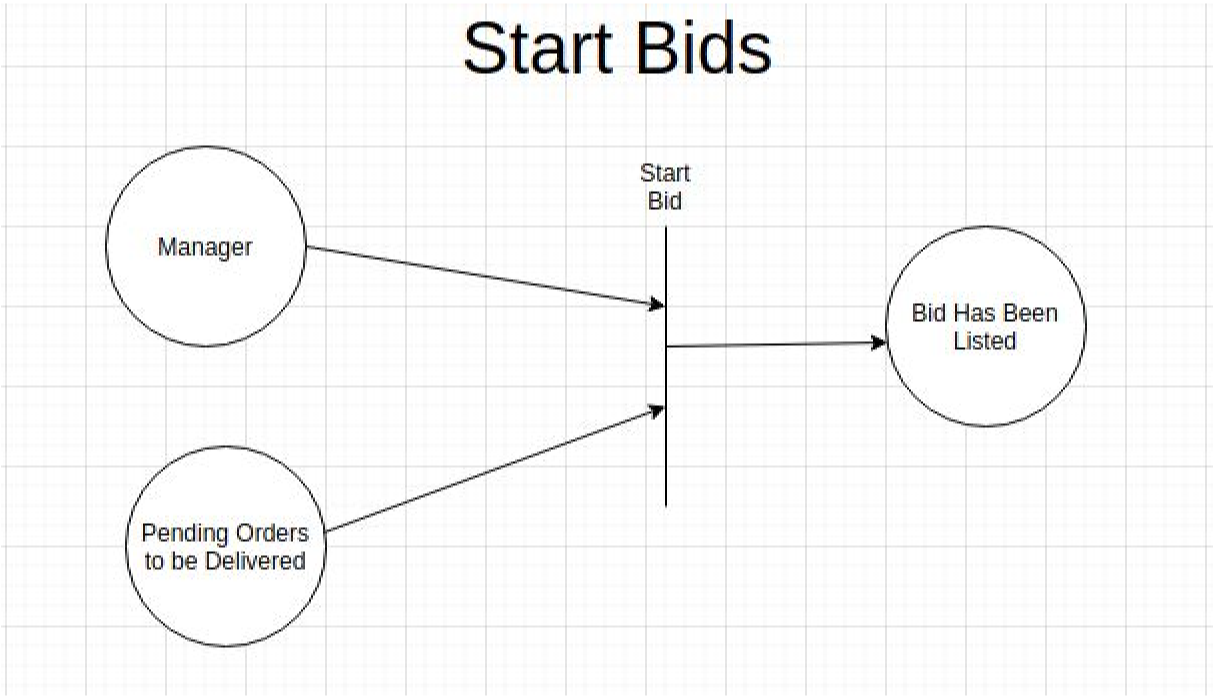

View Delivered Orders
Clicked "Delivered Orders"
Guest/Customer
Delivered Orders Page

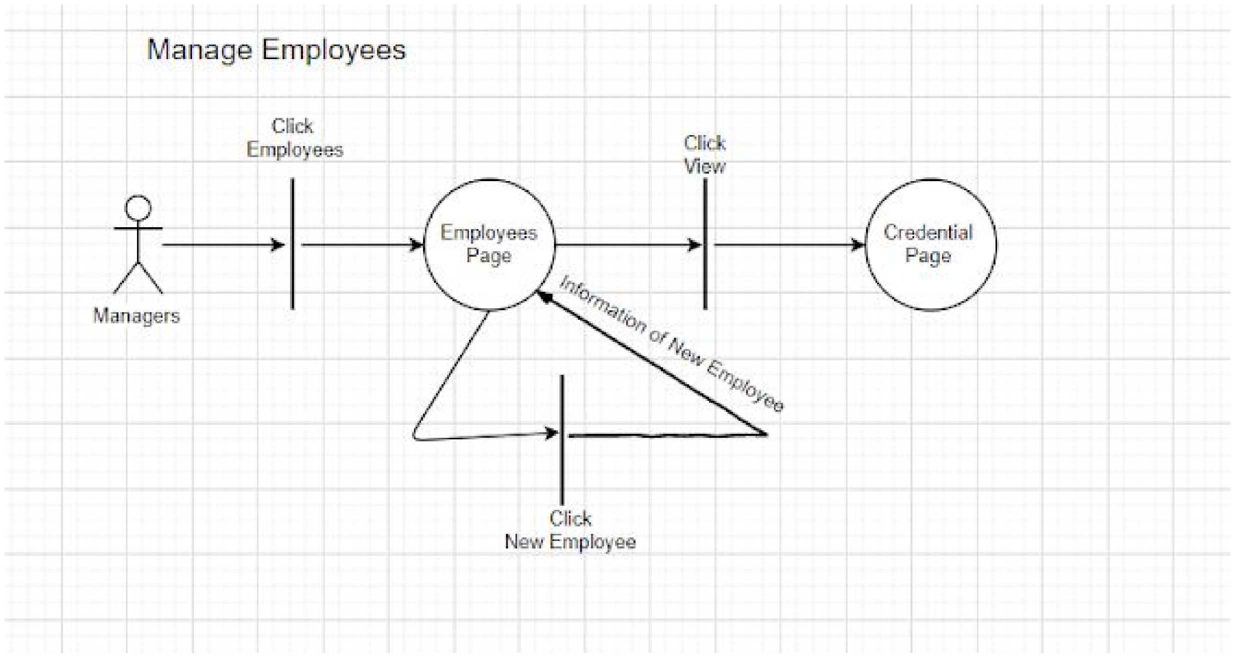

Add Menu Item
Menu Item Details
New Menu Item
Click Modify Menu
Cook
Modify Menu Page
Deleted Menu Item
Current Menu Item
Remove Menu Item

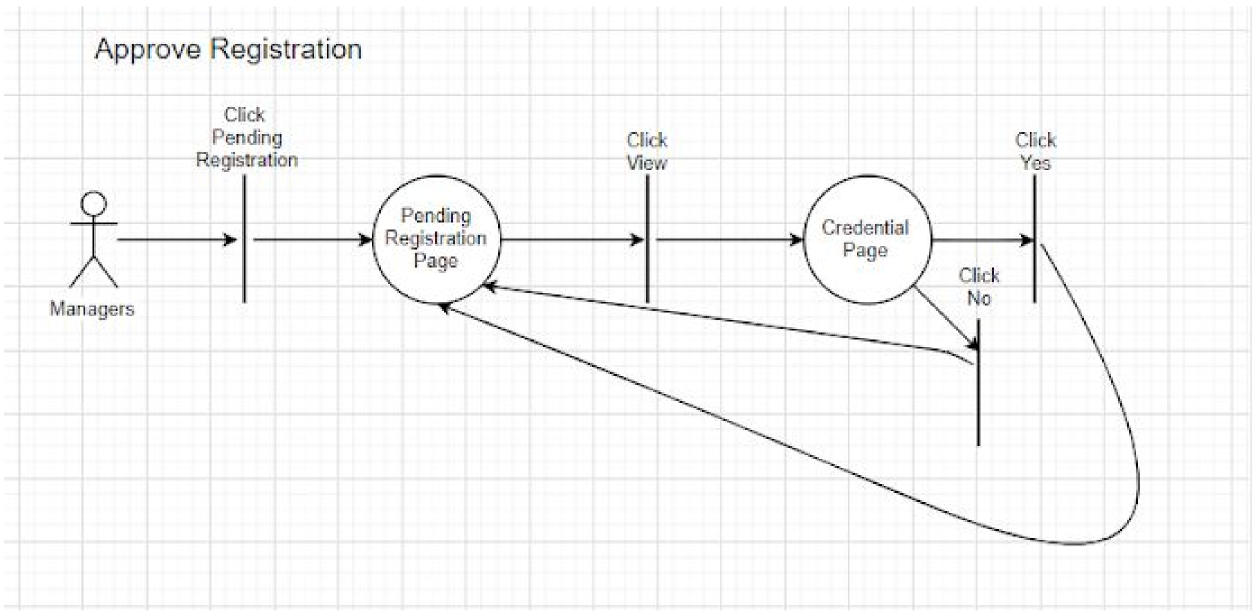

Request Supplies
Click Request Supplies
Cooks
Request Supples Page
Order
Order Page


Rate Supplies
Rate Past Orders
View Order
Cook
Past Orders Page
Order
Rate 1-5
Database

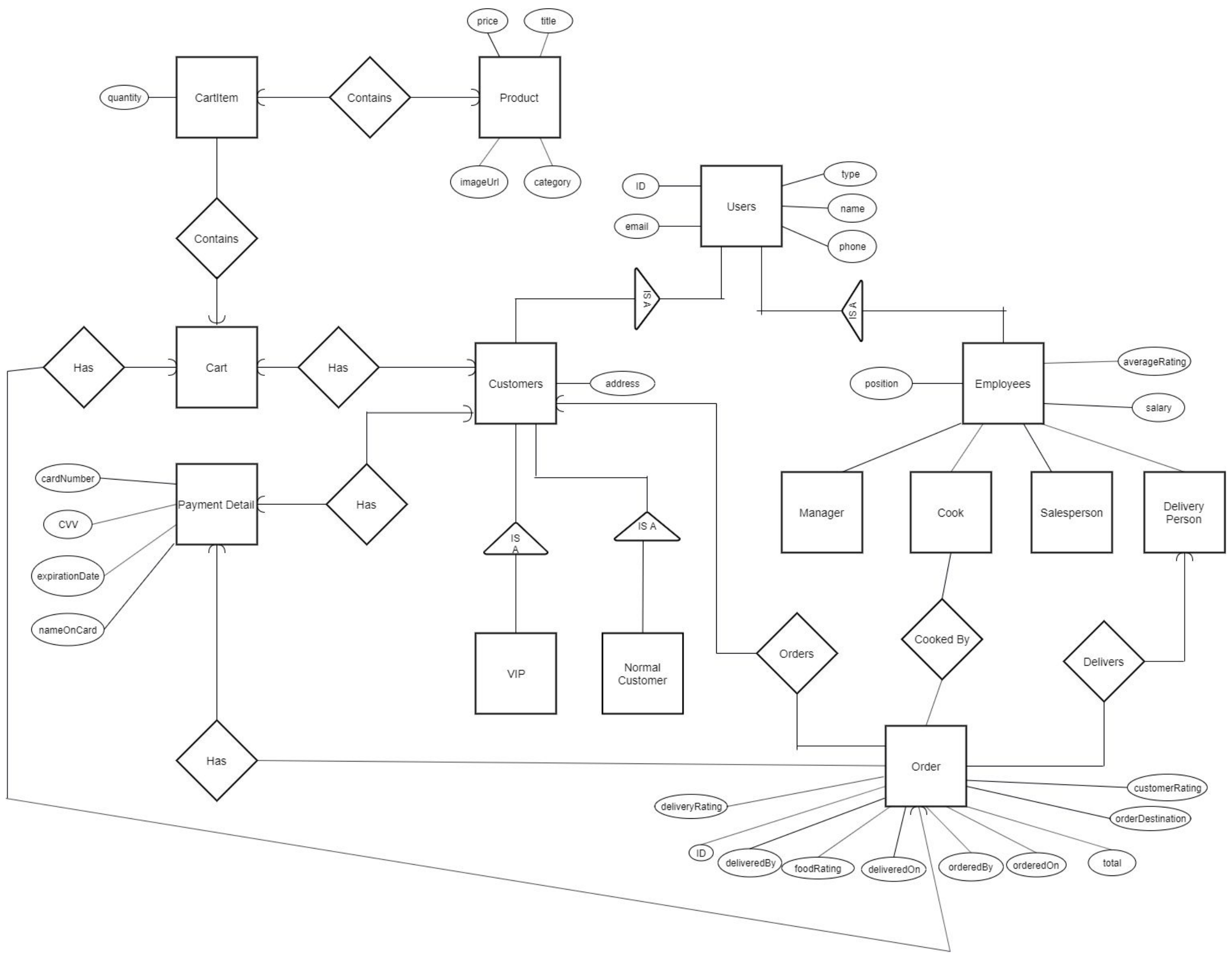

Purchase Supplies
Purchase
Salesperson
Purchased Supplies
Supplies


Start Bids
Manager
Start Bid
Bid Has Been Listed
Pending Orders to be Delivered


Manage Employees
Click Employees
Managers
Employees Page
Click View
Credential Page
Information of New Employee
Click New Employee


Approve Registration
Click Pending Registration
Click View
Click Yes
Managers
Pending Registration Page
Credential Page
Click No

3. E/R Diagram



4. Pseudocode

1. addNewEmployee(formValue)

Input: New Employee Information
Output: None
Description: Add new employee's information to the database

```
addNewEmployee(formValue)
 {
   let secondaryApp =
firebase.initializeApp(environment.firebase,"Secondary");
secondaryApp.auth().createUserWithEmailAndPassword(formValue.e
mail,"123456").then(userCredentials=>{
     let tempUser:User =
{uid:userCredentials.user.uid,email:userCredentials.user.email
,phone:formValue.phone,name:formValue.name,type:"employee"};
     this.userServe.addUser(tempUser);
     let tempEmployee:Employee =
{uid:userCredentials.user.uid,position:formValue.position,sala
ry:formValue.salary};
     this.emplServe.addEmployee(tempEmployee);
     secondaryApp.auth().signOut();
   });
 }
```

2. login(email:string, password:string)

Input: Email, Password
Output: redirected to appropriate URL
Description: Allows a person to login, and DB returns proper type and displays appropriate page

```
 login(email:string, password:string)
 {
   this.afAuth.auth.signInWithEmailAndPassword(email,
password).then(userCredentials=>{
     this.user$.pipe(take(1)).subscribe(user=>{
       this.userServe.getUser(user.uid).pipe(take(1)).subscribe(user=>{
         if(user.type=="customer")
         {
           this.router.navigateByUrl("/" + user.type);
         }
         else
         {

this.emplServe.getEmployee(user.uid).pipe(take(1)).subscribe(employee=>{
             this.router.navigateByUrl("/" + employee.position);
           })
         }
       })
     })
   }).catch(error=>{this.error = error});
 }
```

3. register(email:string, password:string)

Input: email, password
Output: None
Description: Creates a customer object with the email and password

```
register(email:string, password:string)
{
this.afAuth.auth.createUserWithEmailAndPassword(email,password
);
 }
```

4. logout()

Input: none
Output: none
Description: Logs current user out

```
logout()
{
  this.afAuth.auth.signOut();
  this.router.navigateByUrl("");
}
```

5. guestLogin()

Input: none
Output: none
Description: Redirects to guest homepage

```
  guestLogin()
  {
    this.afAuth.auth.signInAnonymously().then(guestCredentials=>{
      this.router.navigateByUrl("/guest");
    });
  }
```

6. addToCart(product:Product)

Input: Product
Output: none
Description: Added product to cart

```
addToCart(product:Product)
 {
   this.authServe.user$.pipe(take(1)).subscribe(user=>{

this.custServe.getCustomer(user.uid).pipe(take(1)).subscribe(customer=>{
       for(let cartItem of customer.shoppingCart)
       {
         if(cartItem.product.title == product.title)
         {
           cartItem.quantity += 1;
           this.custServe.updateCustomer(user.uid,customer);
           return;
         }
       }
       customer.shoppingCart.push({product:product,quantity:1});
       this.custServe.updateCustomer(user.uid,customer);
     });
   });
 }
```

7. removeFromCart(product:Product)

Input: product
Output: none
Description: Remove product from cart

```
removeFromCart(product:Product)
 {
   this.authServe.user$.pipe(take(1)).subscribe(user=>{

this.custServe.getCustomer(user.uid).pipe(take(1)).subscribe(customer=>{
       for(let cartIndex =
0;cartIndex<customer.shoppingCart.length;cartIndex++)
       {
         if(customer.shoppingCart[cartIndex].product.title ==
product.title)
         {
           customer.shoppingCart[cartIndex].quantity -= 1;
           if(customer.shoppingCart[cartIndex].quantity == 0)
           {
             customer.shoppingCart.splice(cartIndex,1);
           }
           this.custServe.updateCustomer(user.uid,customer);
           return;
         }
       }
     });
   });
 }
```

8. getCustomer()

Input: none
Output: customer object
Description: Returns customer object

```
getCustomer()
 {
   return this.custServe.getCurrentCustomer();
 }
```

9. getUser()

Input: none
Output: user object
Description: Returns user object

```
getUser()
 {
   return this.custServe.getCurrentUser();
 }
```

10. updateCart(customer:Customer)

Input: Customer

Output: updated cart
Description: Customer object updates its cart member

```
updateCart(customer:Customer)
 {
   this.authServe.user$.pipe(take(1)).subscribe(user=>{
     this.custServe.updateCustomer(user.uid,customer);
   })
 }
```

11. addToGuestCart(product:Product)

Input: product
Output: none
Description: Adds the product to the guest user's card

```
addToGuestCart(product:Product)
 {
   this.cart = JSON.parse(localStorage.getItem("cart"));
   for(let cartItem of this.cart)
   {
     if(cartItem.product.title == product.title)
     {
       cartItem.quantity += 1;
       localStorage.setItem("cart",JSON.stringify(this.cart));
       this.cart$.next(this.cart);
       return;
     }
   }
   this.cart.push({product:product,quantity:1});
   localStorage.setItem("cart",JSON.stringify(this.cart));
   this.cart$.next(this.cart);
   return;
 }
```

12. removeFromGuestCart(product:Product)

Input: product
Output: none
Description: Remove the product from the guest's cart

```
removeFromGuestCart(product:Product)
 {
   this.cart = JSON.parse(localStorage.getItem("cart"));
   for(let cartIndex = 0;cartIndex <
this.cart.length;cartIndex++)
   {
     if(this.cart[cartIndex].product.title == product.title)
     {
       this.cart[cartIndex].quantity -= 1;
       if(this.cart[cartIndex].quantity == 0)
       {
         this.cart.splice(cartIndex,1);
       }
       localStorage.setItem("cart",JSON.stringify(this.cart));
       this.cart$.next(this.cart);
       return;
     }
   }
 }
```

13. addCustomer(customer:Customer)

Input: customer
Output: none
Description: Adds customer

```
addCustomer(customer:Customer)
 {
   this.customerCollection.doc(customer.uid).set(customer);
 }
```

14. removeCustomer(customer:Customer)

Input: customer
Output: None
Description: Removes Customer

```
removeCustomer(customer:Customer)
 {
   this.afs.doc("customers/" + customer.uid).delete();
 }
```

15. updateCustomer(uid:string, customer:Customer)

Input: uid, customer
Output: none
Description: Update customer object

```
updateCustomer(uid:string,customer:Customer)
 {
   this.afs.doc("customers/" + uid).update(customer);
 }
```

16. getCustomer(uid:string) : Observable<Customer>

Input: uid
Output: observable<customer>
Description: Returns an object to observe changes in the customer

```
getCustomer(uid:string):Observable<Customer>
 {
   return this.afs.doc("customers/" + uid).valueChanges() as
Observable<Customer>;
 }
```

17. getUser(uid:string) : Observable<User>

Input: uid
Output: observable<user>
Description: Returns an object to observe changes in the user

```
getUser(uid:string):Observable<User>
 {
   return this.userServe.getUser(uid);
 }
```

18. getCurrentUser()

Input: none
Output: user object
Description: Get user object from the database

```
 getCurrentUser()
 {
   return this.authServe.user$.pipe(switchMap(user=>{
     return this.getUser(user.uid);
   }));
 }
```

19. getCurrentCustomer()

Input: none
Output: customer object
Description: Get customer object from the database

```
getCurrentCustomer()
{
  return this.authServe.user$.pipe(switchMap(user=>{
    return this.getCustomer(user.uid)
  }));
}
```

20. addEmployee(employee:Employee)

Input: employee
Output: none
Description: Adds employee to the database

```
addEmployee(employee:Employee)
 {
  this.employeeCollection.doc(employee.uid).set(employee);
 }
```

21. removeEmployee(employee:Employee)

Input: employee
Output: none
Description: Remove employee from the database

```
removeEmployee(employee:Employee)
 {
  this.afs.doc("employees/" + employee.uid).delete();
 }
```

22. getEmployee(uid:string): Observable<Employee>

Input: String of an employee

Output: Returns the name of the employee
Description: Finds the name of the employee in the input and returns it

```
getEmployee(uid:string):Observable<Employee>
 {
   return this.afs.doc("employees/" + uid).valueChanges() as
Observable<Employee>;
 }
```

23. getUser(uid:string) : Observable<User>

Input: A string of a user
Output: Returns the name of user
Description: Finds the name of the user in the input and outputs it

```
getUser(uid:string):Observable<User>
 {
   return this.userServe.getUser(uid);
 }
```

24. addGuest(guest:Guest)

Input: guest
Output: None
Description: Adds guest to guest database

```
addGuest(guest:Guest)
 {
   this.guestCollection.add(guest);
 }
```

25. removeGuest(guest:Guest)

Input: guest
Output: None
Description: Guest deleted from database

```
removeGuest(guest:Guest)
 {
   this.afs.doc("guests/" + guest.uid).delete();
 }
```

26. getGuest(uid:string) : Observable<Guest>

Input: uid
Output: Returns guest object
Description: Calls the guest object with parameter uid, and returns it.

```
getGuest(uid:string):Observable<Guest>
 {
   return this.afs.doc("guests/" + uid).valueChanges() as
Observable<Guest>;
 }
```

27. addOrder(order:Order)

Input: Customer’s order
Output: Nothing
Description: Creates an id for the order and adds it into the “orderCollection” database

```
addOrder(order:Order)
 {
   let id = this.afs.createId()
   this.orderCollection.doc(id).set(order);
   return id;
 }
```

28. removeOrder(order:Order)

Input: Order object
Output:none
Description: removes an order object

```
removeOrder(order:Order)
 {
   this.afs.doc("orders/" + order.uid).delete();
 }
```

29. updateOrder(uid:string, order:Order)

Input: takes in string
Output:none
Description: takes in string and updates order

```
updateOrder(uid:string,order:Order)
{
  this.afs.doc("orders/" + uid).update(order);
}
```

30. getOrder(uid:string): Observable<Order>

Input: takes in a string
Output: returns order
Description: Takes a string and returns an order

```
getOrder(uid:string):Observable<Order>
{
  return this.afs.doc("orders/" + uid).valueChanges() as
Observable<Order>;
}
```

31. create(product:Product)

Input: product object
Output: none
Description: A Product object is created

```
create(product:Product)
{
  this.productsCollection.add(product);
}
```

32. lookUp(uid:string): Observable<Product>

Input: Any string
Output: The string as the desired product
Description: Takes in a string and a product and returns a string as that product

```
lookup(uid:string):Observable<Product>
  {
    return this.afs.doc('products/'+uid).valueChanges() as
Observable<Product>;
  }
```

33. update(uid:string, product:Product)

Input: Any string and any product
Output: Nothing
Description: Changes the specified and replaces it with the product

```
update(uid:string, product:Product)
  {
    return this.afs.doc('products/'+uid).update(product);
  }
```

34. delete(uid:string)

Input: Any string
Output: The string
Description: Finds and deletes a specified string and returns it

```
delete(uid:string)
  {
    return this.afs.doc('products/'+uid).delete();
  }
```

35. addUser(user:User)

Input: Name of any user of type user
Output: Nothing
Description: Adds the specified user

```
  addUser(user:User)
  {
    this.userCollection.doc(user.uid).set(user);
  }
```

36. removeUser(user:User)

Input: Name of any user of type user
Output: Nothing
Description: Removes the specified user

```
  removeUser(user:User)
  {
    this.afs.doc("users/" + user.uid).delete();
  }
```

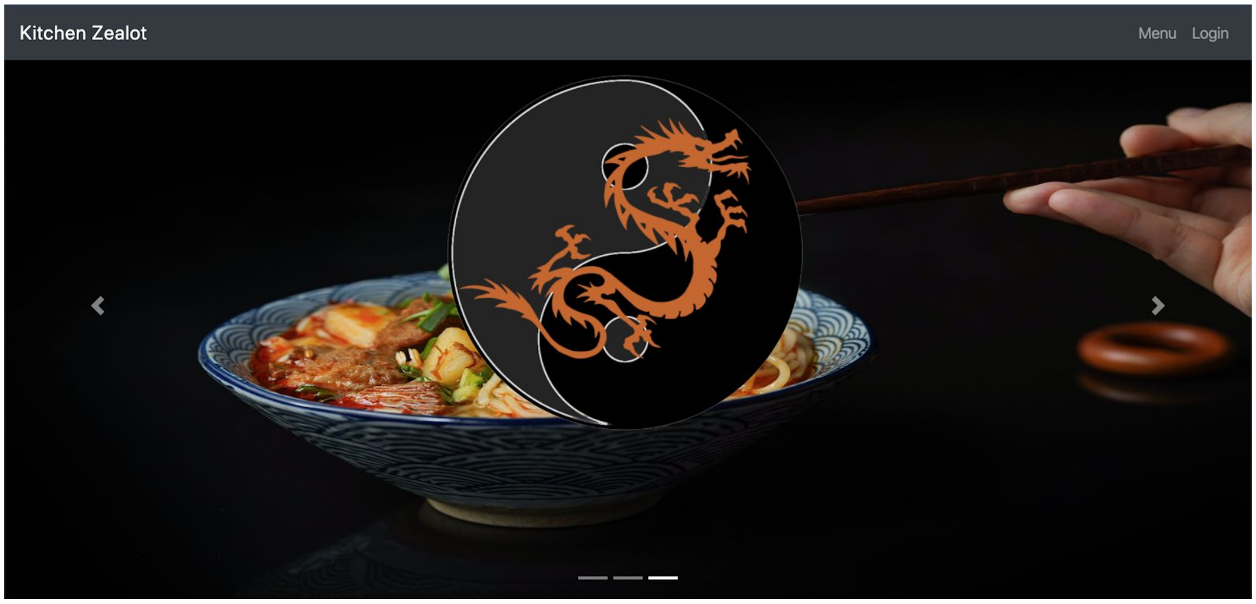
37. getUser(uid;string): Observable<User>

Input: String of the user you want to find
Output: String of the user
Description: Returns the name of the input string, if it exists

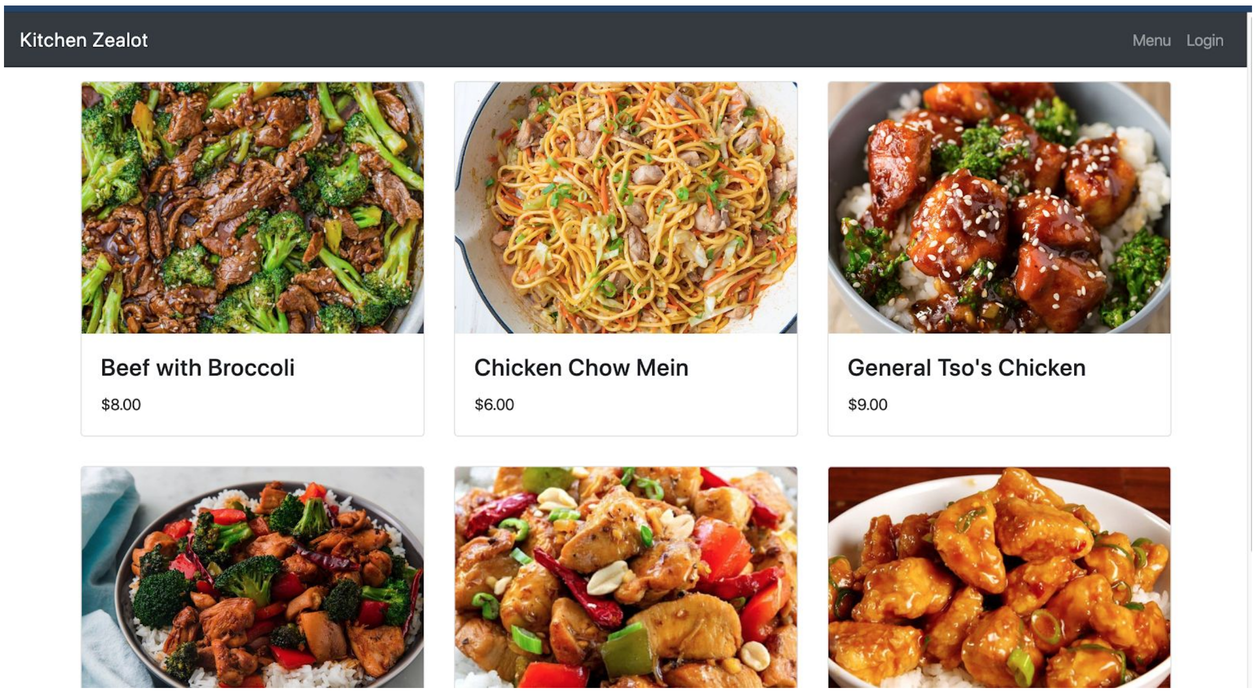
```
  getUser(uid:string):Observable<User>
  {
    return this.afs.doc("users/" + uid).valueChanges() as
Observable<User>;
  }
```

5. System Screen

Main Page



Order Page



Beef with Broccoli

$8.00

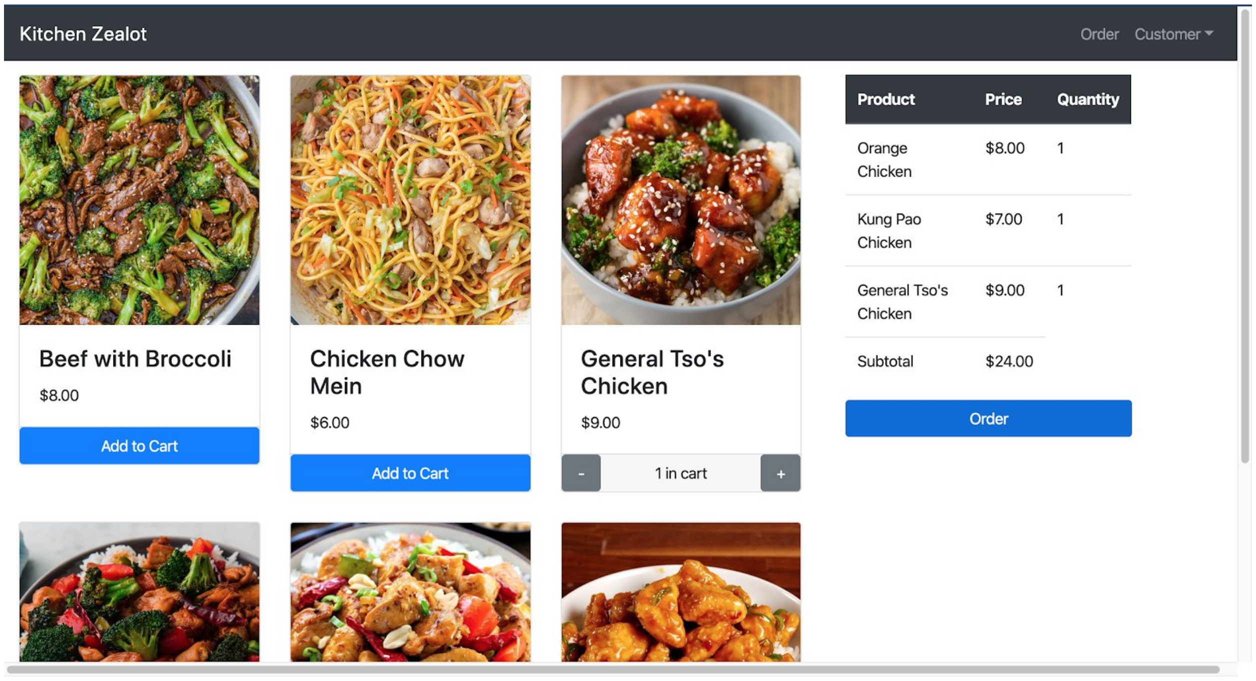
Chicken Chow Mein

$6.00

General Tso's Chicken

$9.00

Cart



Beef with Broccoli

$8.00

Chicken Chow Mein

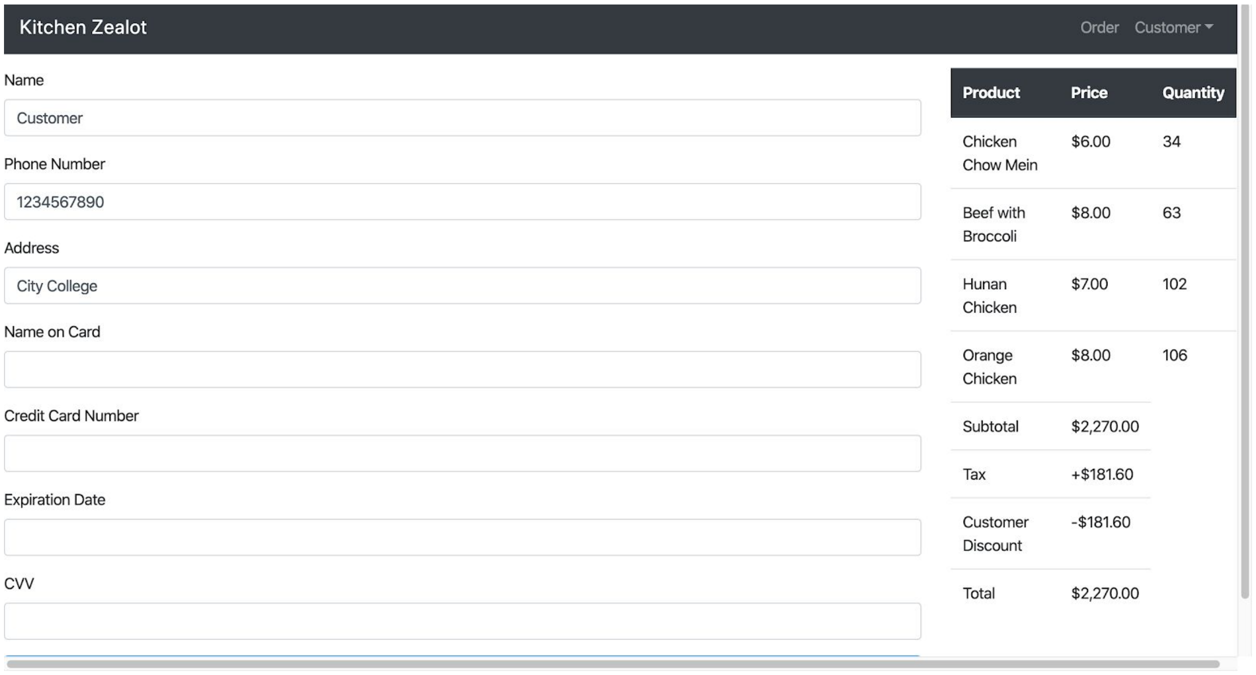
$6.00

General Tso's Chicken

$9.00

Product	Price	Quantity
Orange Chicken	$8.00	1
Kung Pao Chicken	$7.00	1
General Tso's Chicken	$9.00	1
Subtotal	$24.00	

<u>CheckOut Page</u>



Name

Customer

Phone Number

1234567890

Address

City College

Name on Card

Credit Card Number

Expiration Date

CVV

Product	Price	Quantity
Chicken Chow Mein	$6.00	34
Beef with Broccoli	$8.00	63
Hunan Chicken	$7.00	102
Orange Chicken	$8.00	106
Subtotal	$2,270.00	
Tax	+$181.60	
Customer Discount	-$181.60	
Total	$2,270.00	