
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	6
2. Design	7
2.1 Use Case Scenarios	7
2.2 Collaboration Diagrams	12
View Menu	12
Login	12
Continue as Guest	13
Register	13
Add/Remove Products to Cart	14
Order Food	14
View All Placed Orders	14
Rate Food/Delivery	15
Bid on Deliveries	16
Give Customer Rating	16
View Delivered Orders	17
Modify Menu Items	17
Request Supplies	18
Rate Supplies	18
Purchase Supplies	19
Start Bids	19
Manage Employees	20
Approve Registration	21
2.2 State Diagrams/ Petri-net	21
View Menu	21
Login	21
Continue as Guest	22
Register	23
Add/Remove Products to Cart	23
Order Food	23
View All Placed Orders	24

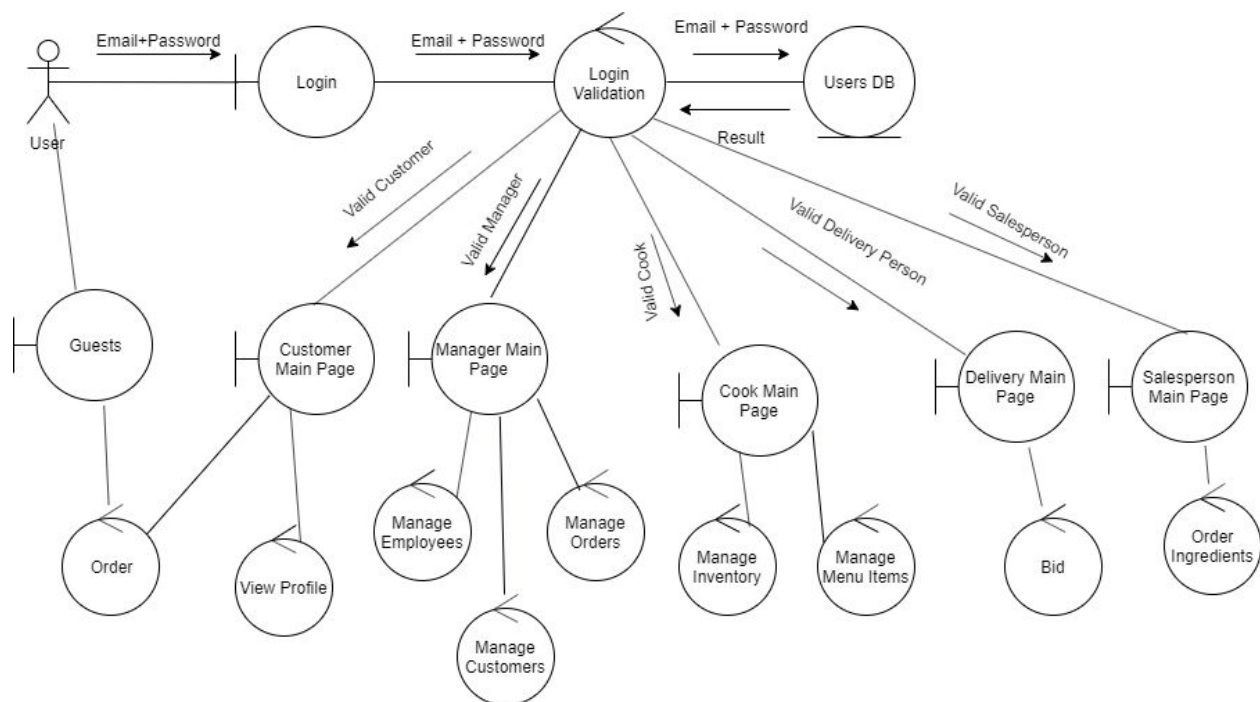
Rate Food/Delivery	25
Bid on Deliveries	26
Give Customer Rating	26
View Delivered Orders	27
Modify Menu Items	28
Request Supplies	29
Rate Supplies	30
Purchase Supplies	31
Start Bids	31
Manage Employees	32
Approve Registration	32
3. E/R Diagram	33
4. Detailed Design	34
addNewEmployee(formValue)	34
login(email:string, password:string)	35
register(email:string, password:string)	35
logout()	35
guestLogin()	36
addToCart(product:Product)	37
removeFromCart(product:Product)	38
getCustomer()	38
getUser()	38
updateCart(customer:Customer)	39
addToGuestCart(product:Product)	40
removeFromGuestCart(product:Product)	41
addCustomer(customer:Customer)	41
removeCustomer(customer:Customer)	42
updateCustomer(uid:string, customer:Customer)	42
getCustomer(uid:string) : Observable<Customer>	42
getUser(uid:string) : Observable<User>	43
getCurrentUser()	43
getCurrentCustomer()	43

addEmployee(employee:Employee)	44
removeEmployee(employee:Employee)	44
getEmployee(uid:string): Observable<Employee>	44
getUser(uid:string) : Observable<User>	44
addGuest(guest:Guest)	44
removeGuest(guest:Guest)	45
getGuest(uid:string) : Observable<Guest>	45
addOrder(order:Order)	45
removeOrder(order:Order)	46
updateOrder(uid:string, order:Order)	46
getOrder(uid:string): Observable<Order>	46
create(product:Product)	46
lookUp(uid:string): Observable<Product>	47
update(uid:string, product:Product)	47
delete(uid:string)	47
addUser(user:User)	47
removeUser(user:User)	47
getUser(uid;string): Observable<User>	48
5. System Screens	49
Main Page	49
Order Page	50
Cart	50
CheckOut Page	51
6. Minutes	51
7. Git Repository	51

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
<p>Normal Scenario</p> <ol style="list-style-type: none">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
<p>Normal Scenario</p> <ol style="list-style-type: none">1. User clicks “Login” button in the navigation bar2. User is asked to enter their email and password3. User inputs appropriate information4. Information is checked against database for validation5. User is logged in
<p>Exceptional Scenario</p> <ol style="list-style-type: none">1. User clicks login and enters credentials2. Credentials are not valid3. Error message is shown4. User is asked to try again

Continue as Guest
<p>Normal Scenario</p> <ol style="list-style-type: none">1. User clicks on “Login” and then “Continue as Guest”2. User inputs their phone number and address when ordering
<p>Exceptional Scenario</p> <ol style="list-style-type: none">1. User clicks on “Login” and then “Continue as Guest”2. User inputs invalid phone number and/or address3. 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

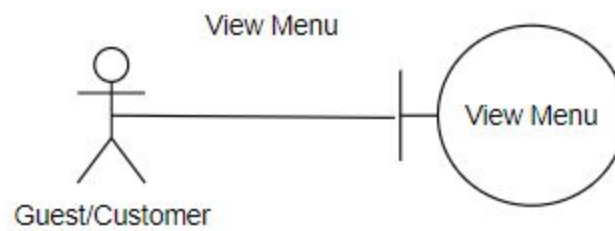
1. Manager clicks “Pending Registrations”
2. Manager can click “View” next to applicants name
3. Manager clicks “Yes” / “No” next to each new guest

Exceptional Scenario

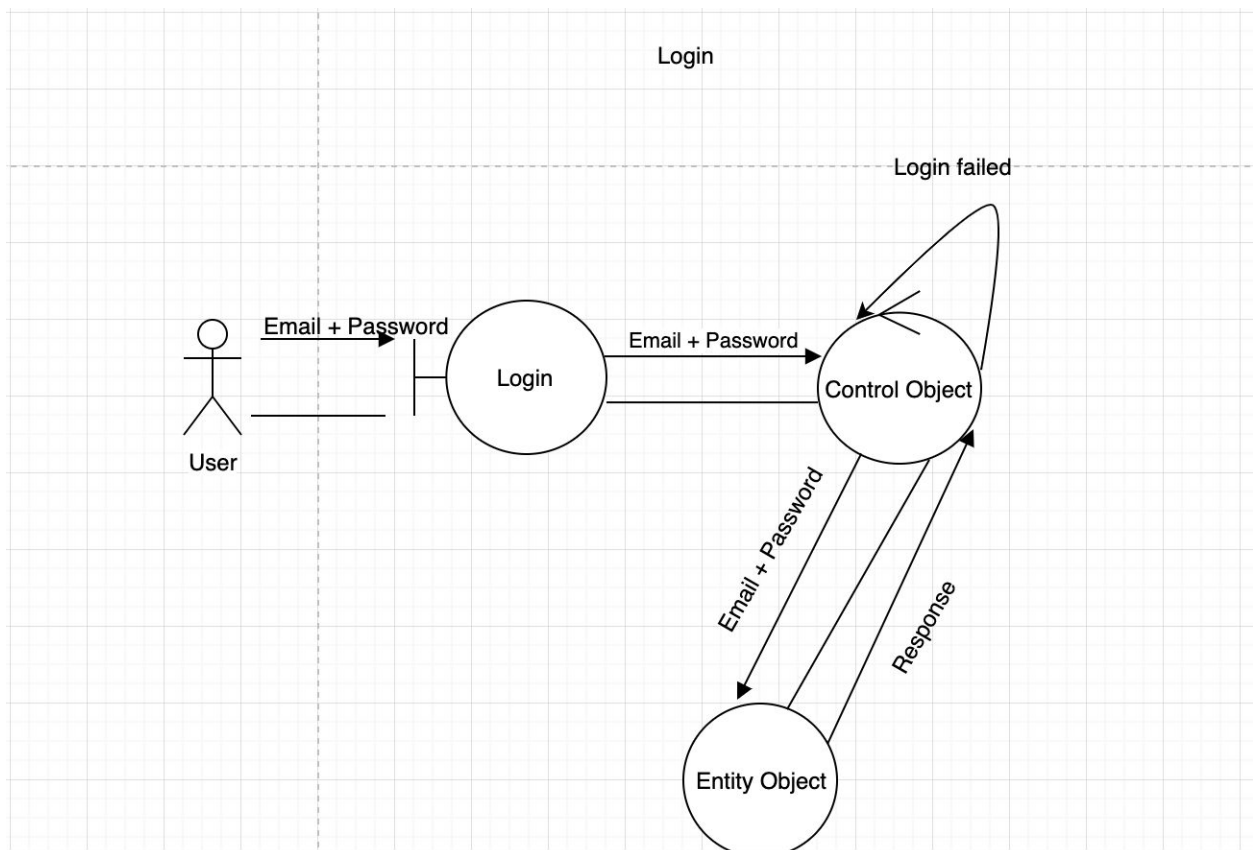
1. Guest may not have order history

2.2 Collaboration Diagrams

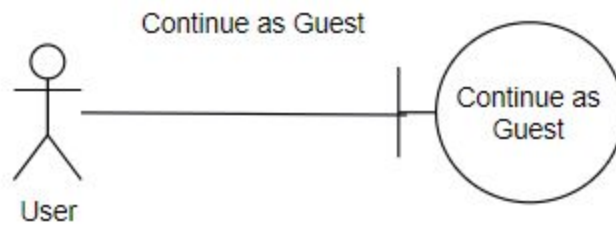
View Menu



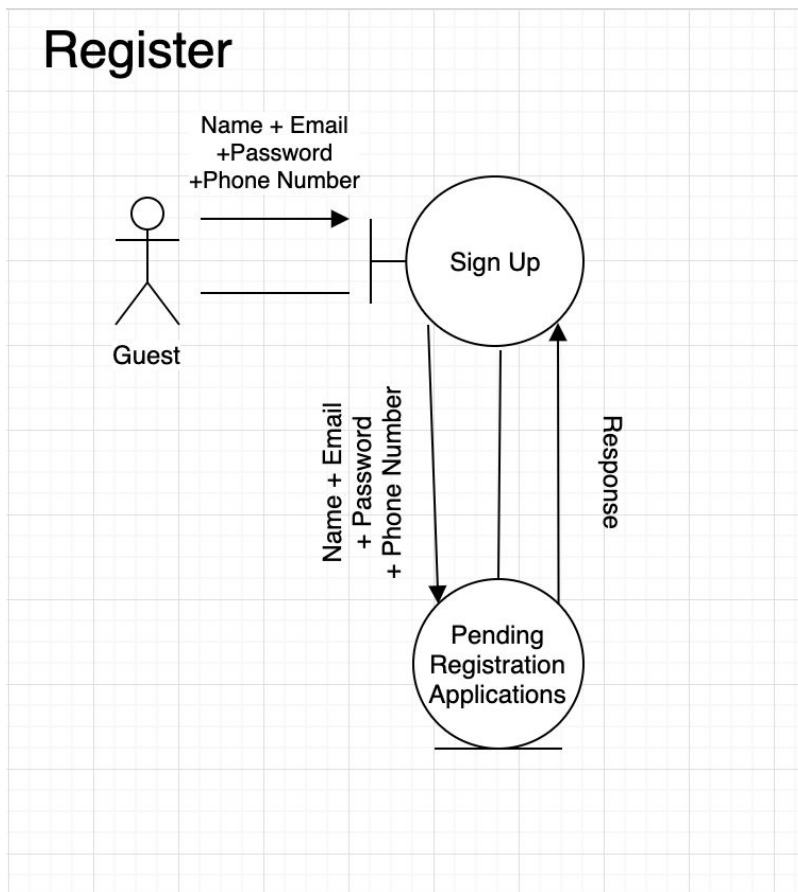
Login



Continue as Guest

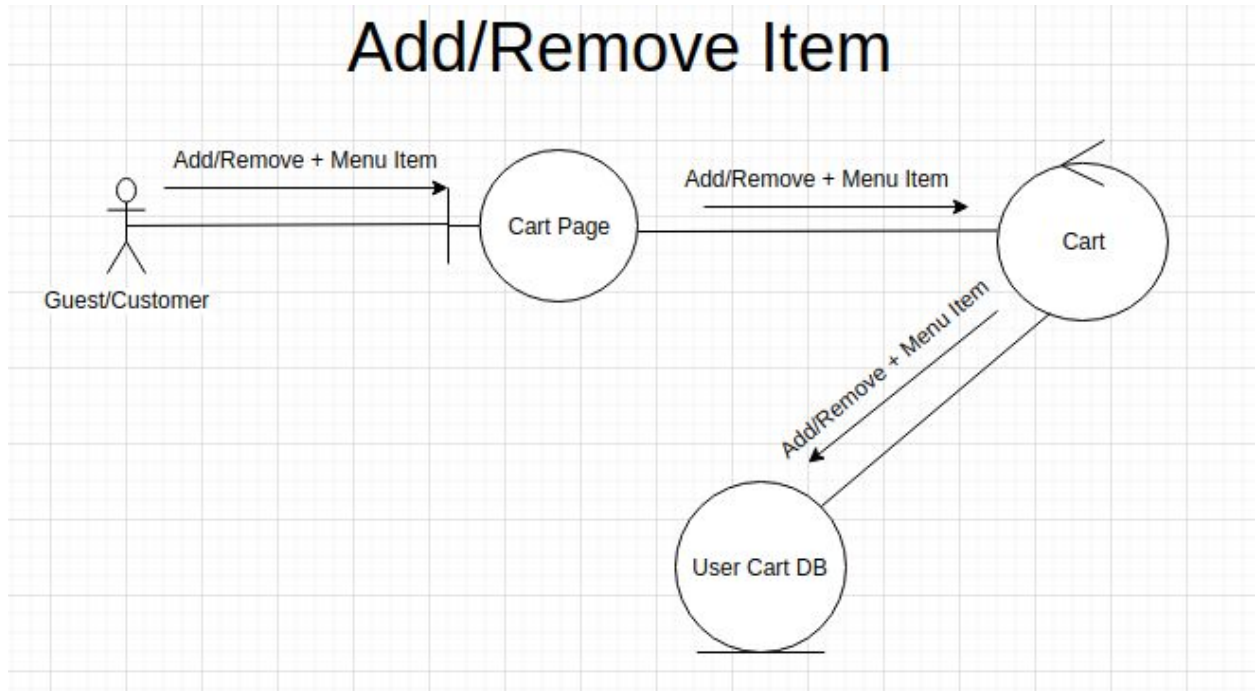


Register

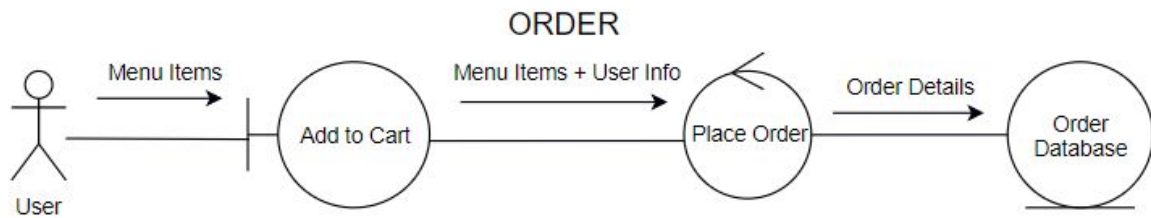


Add/Remove Products to Cart

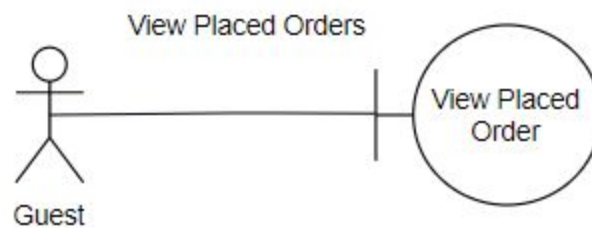
Add/Remove Item



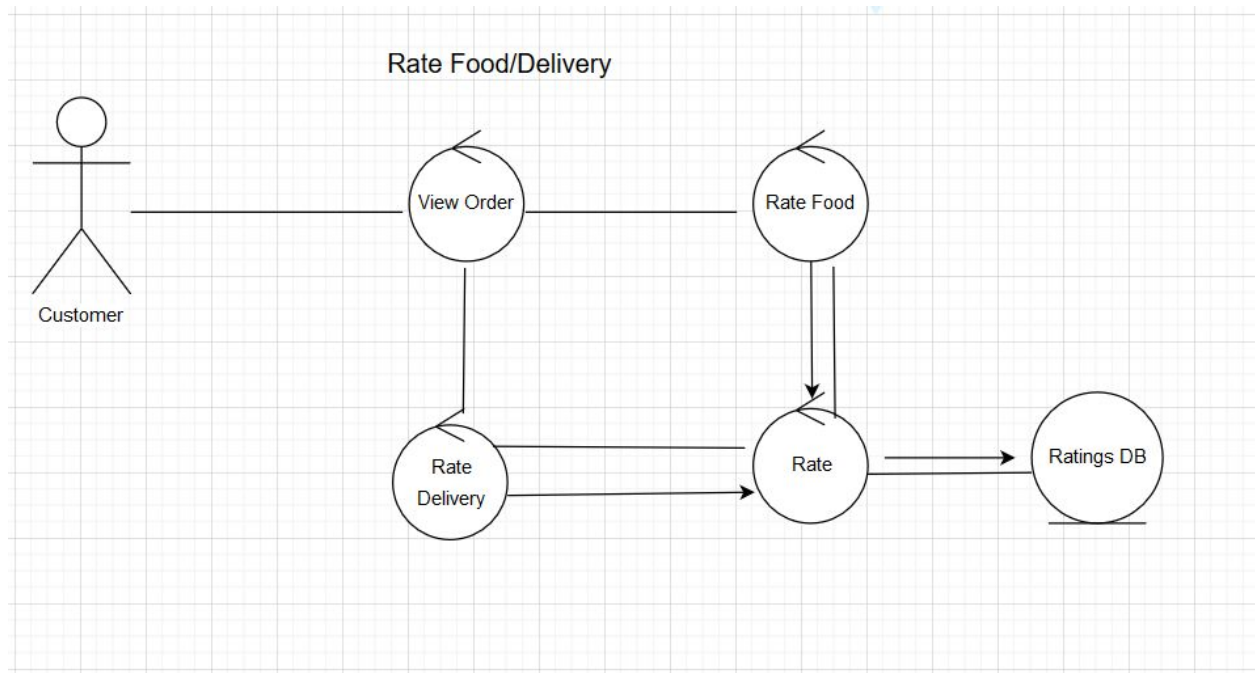
Order Food



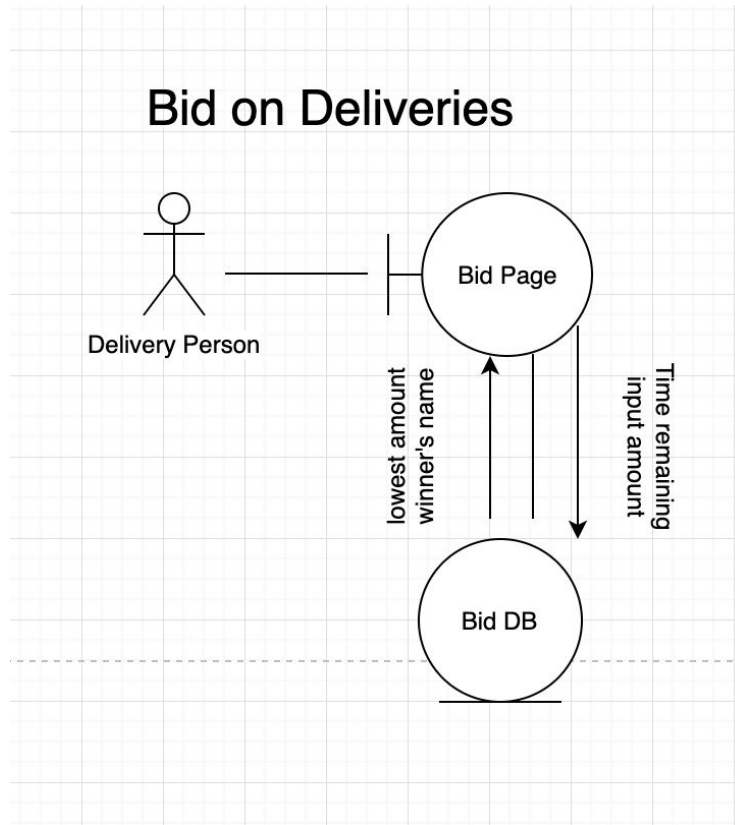
View All Placed Orders



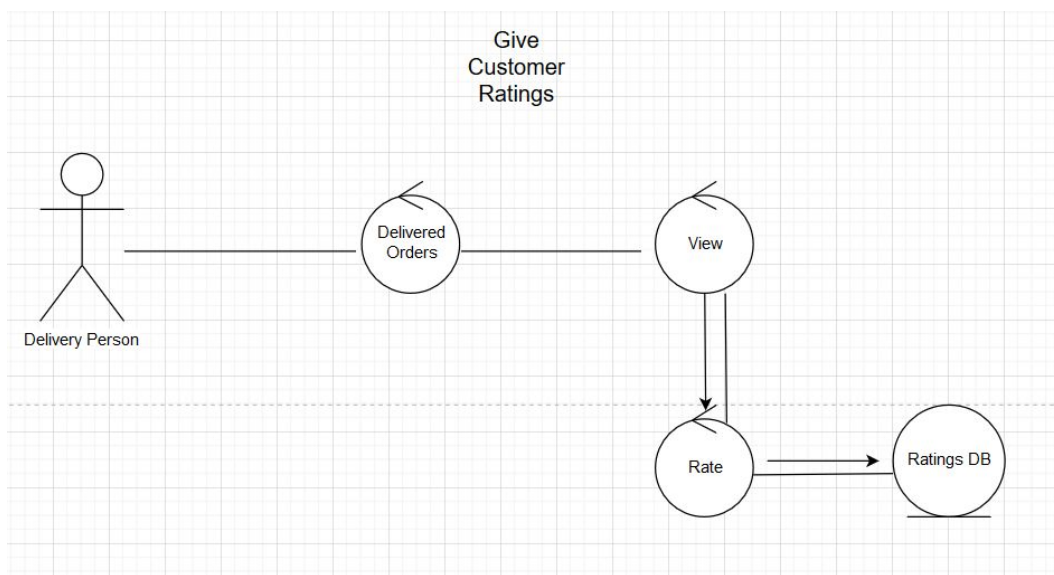
Rate Food/Delivery



Bid on Deliveries



Give Customer Rating



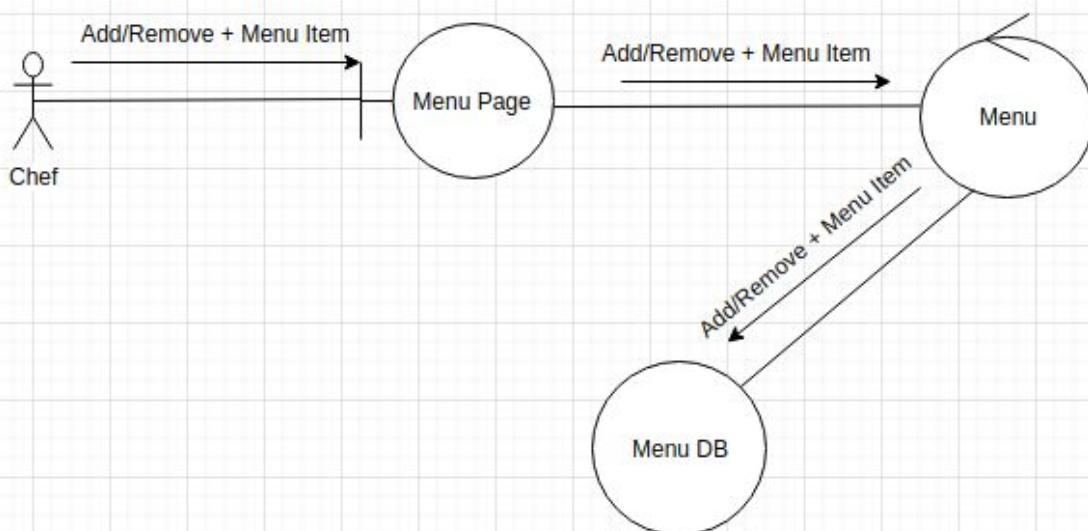
View Delivered Orders

View Delivered Orders

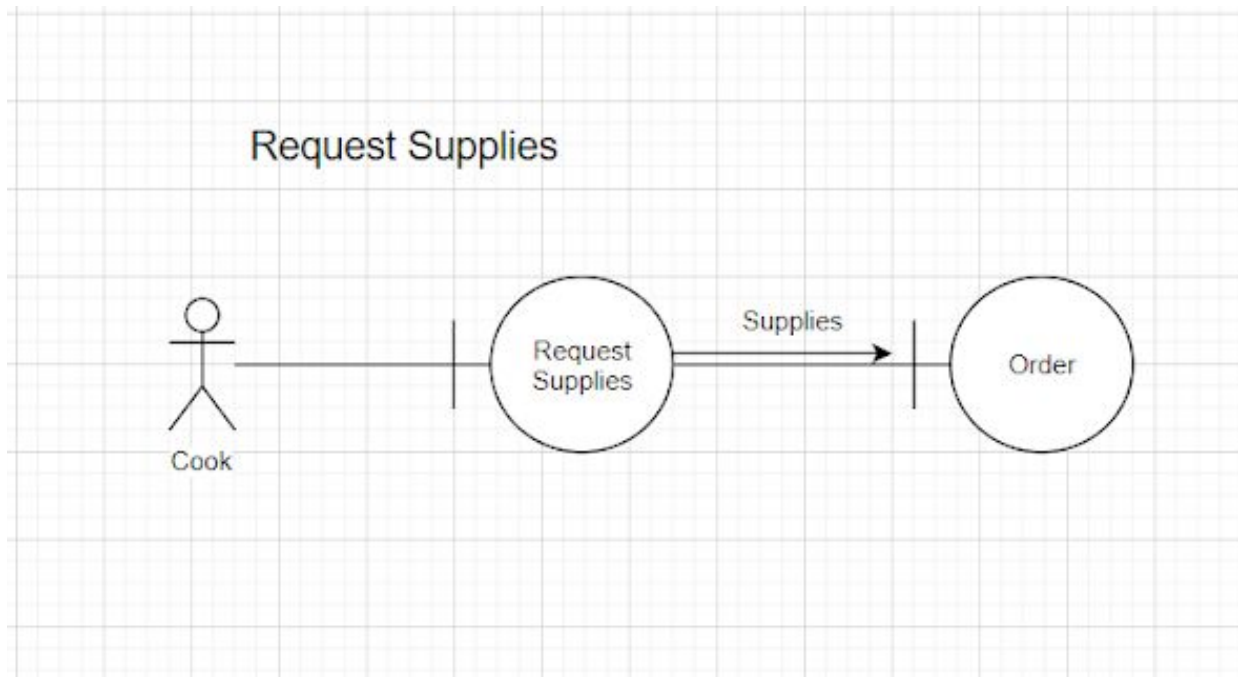


Modify Menu Items

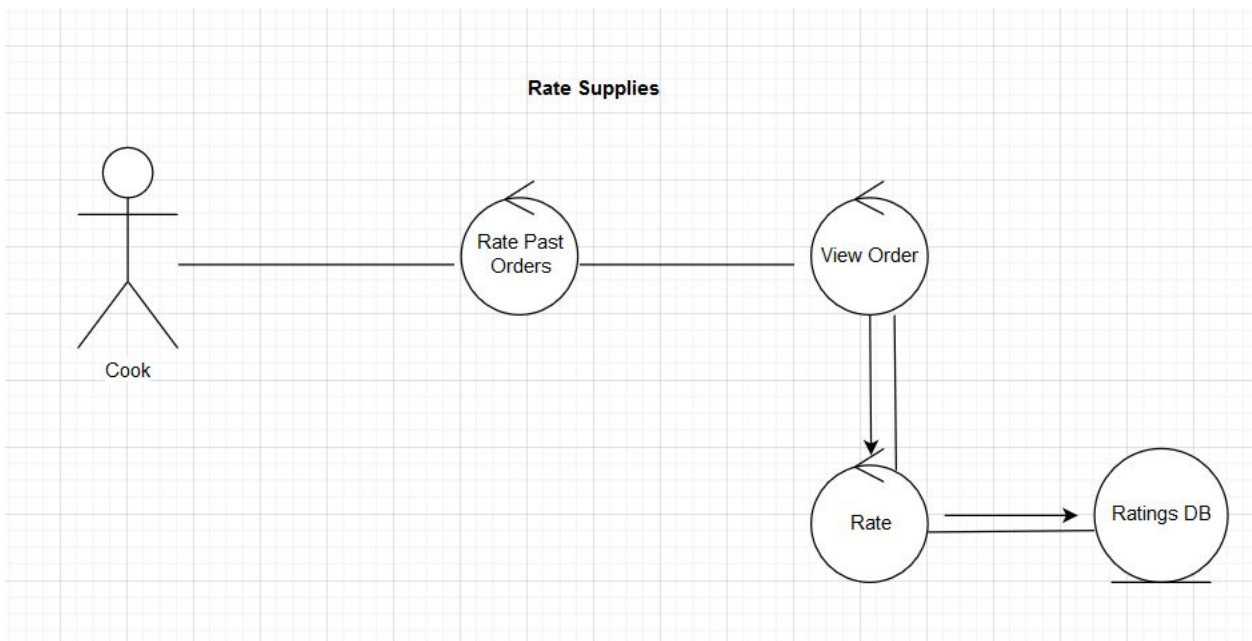
Modify Menu Items



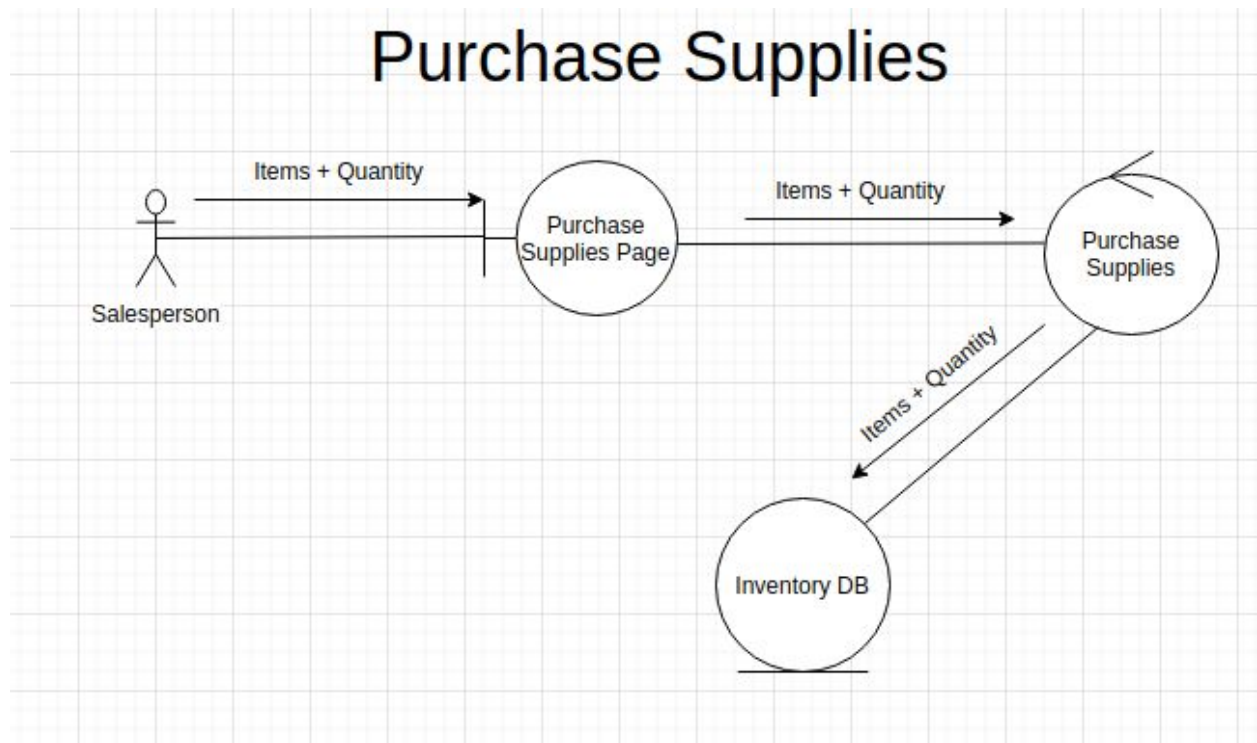
Request Supplies



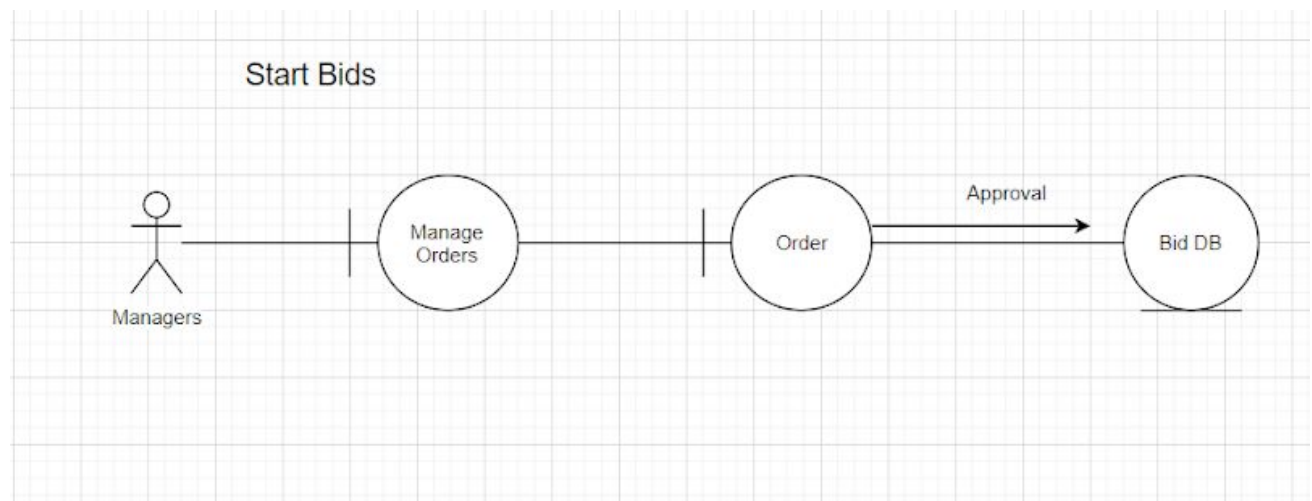
Rate Supplies



Purchase Supplies

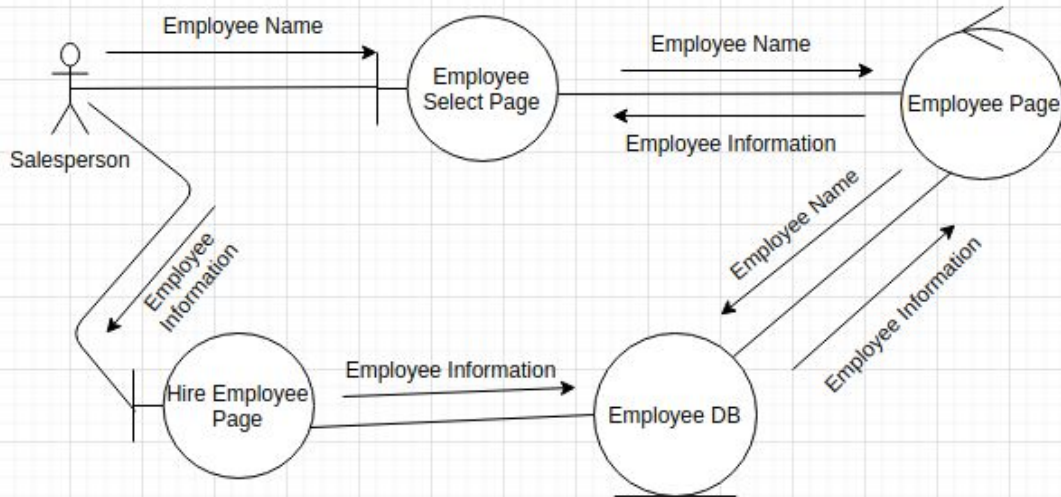


Start Bids

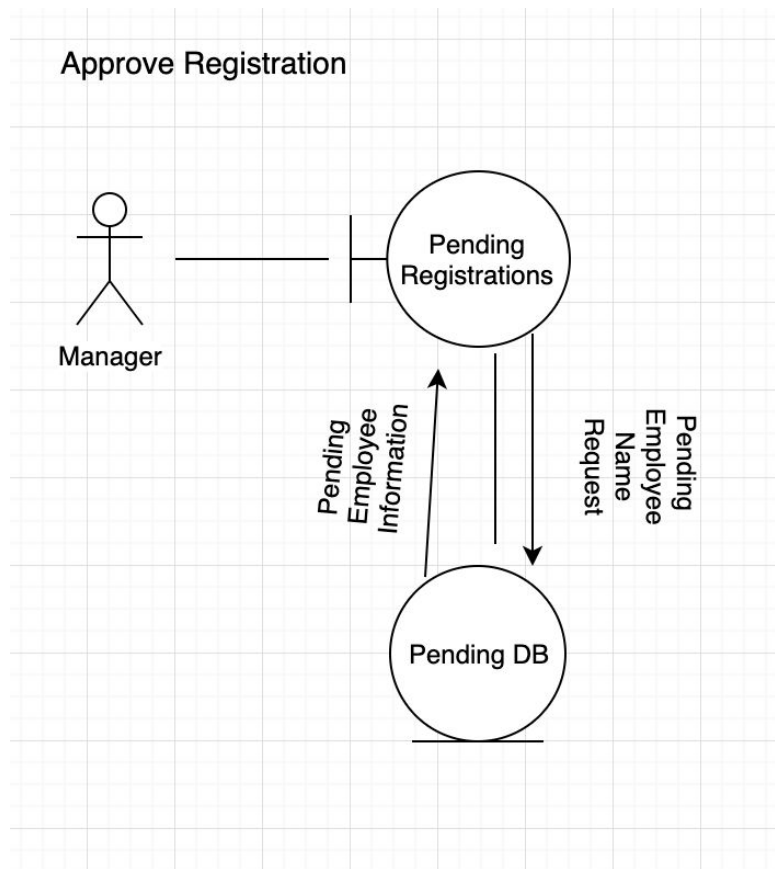


Manage Employees

Manage Employees

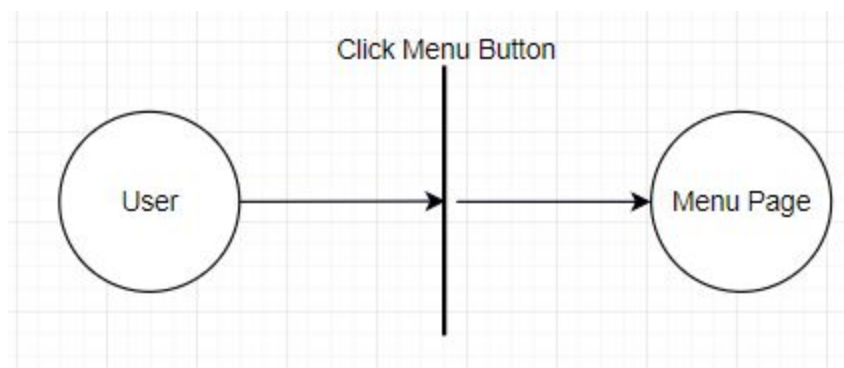


Approve Registration

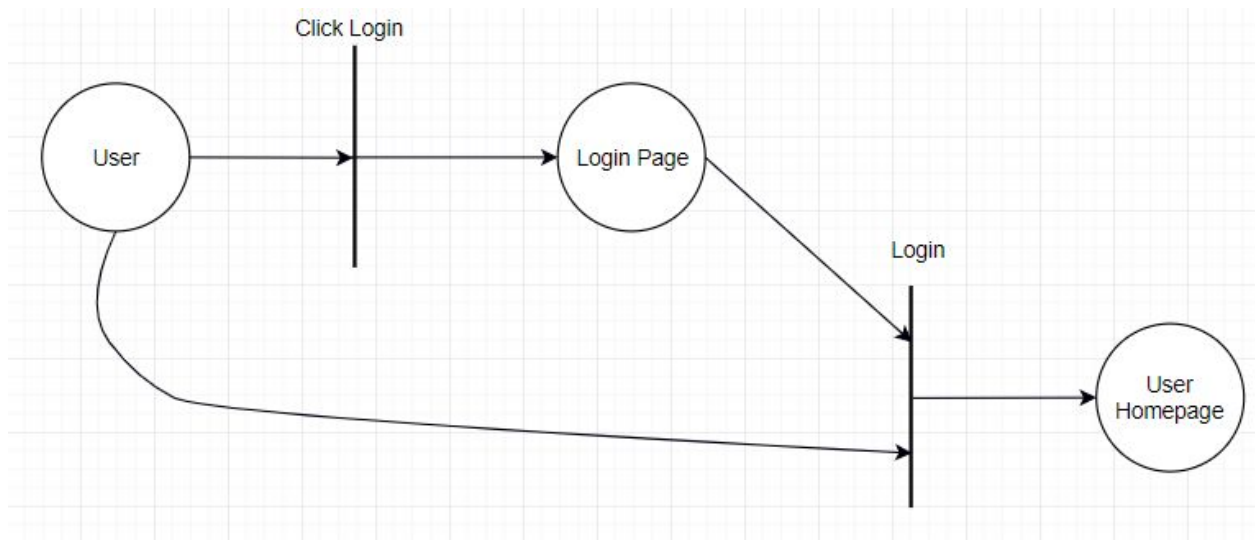


2.2 State Diagrams/ Petri-net

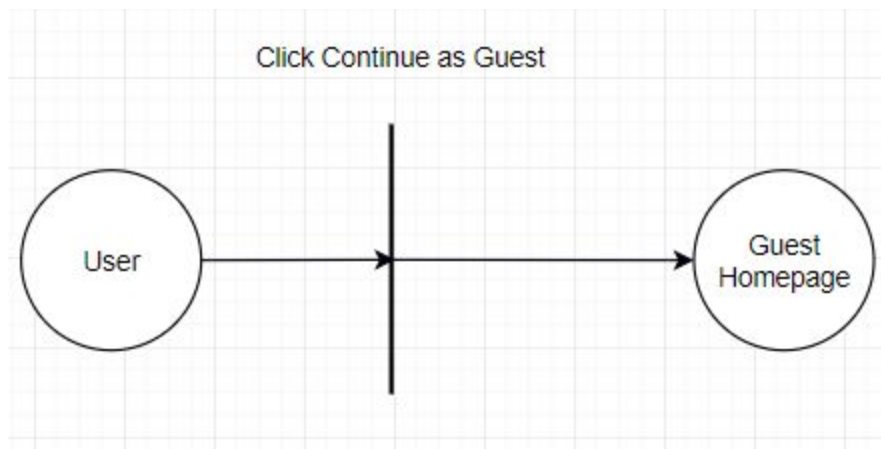
View Menu



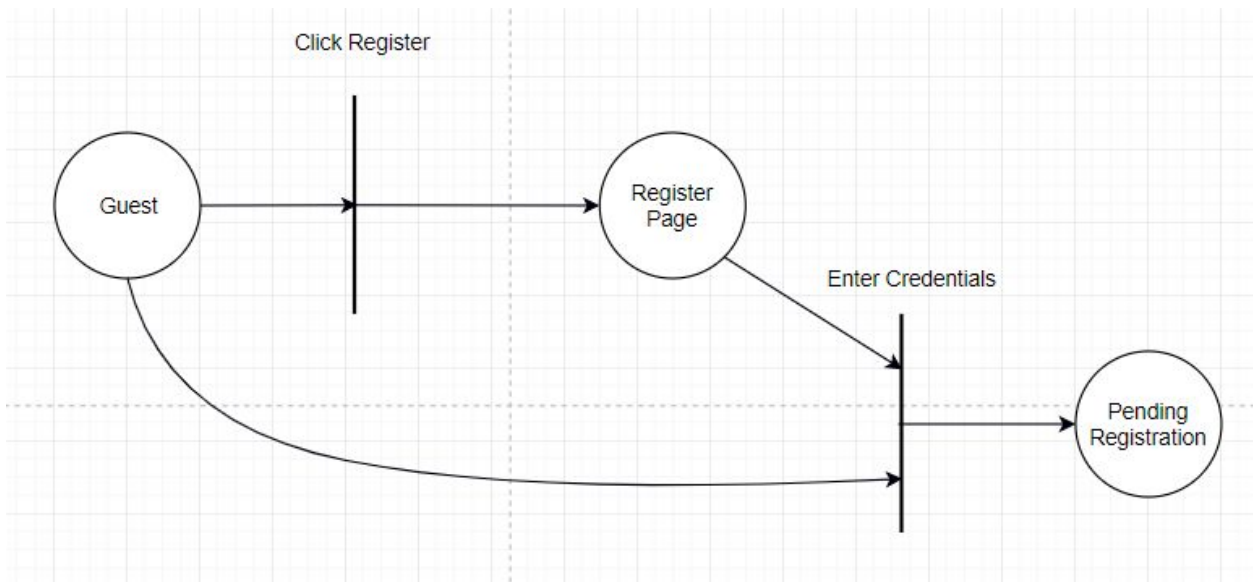
Login



Continue as Guest

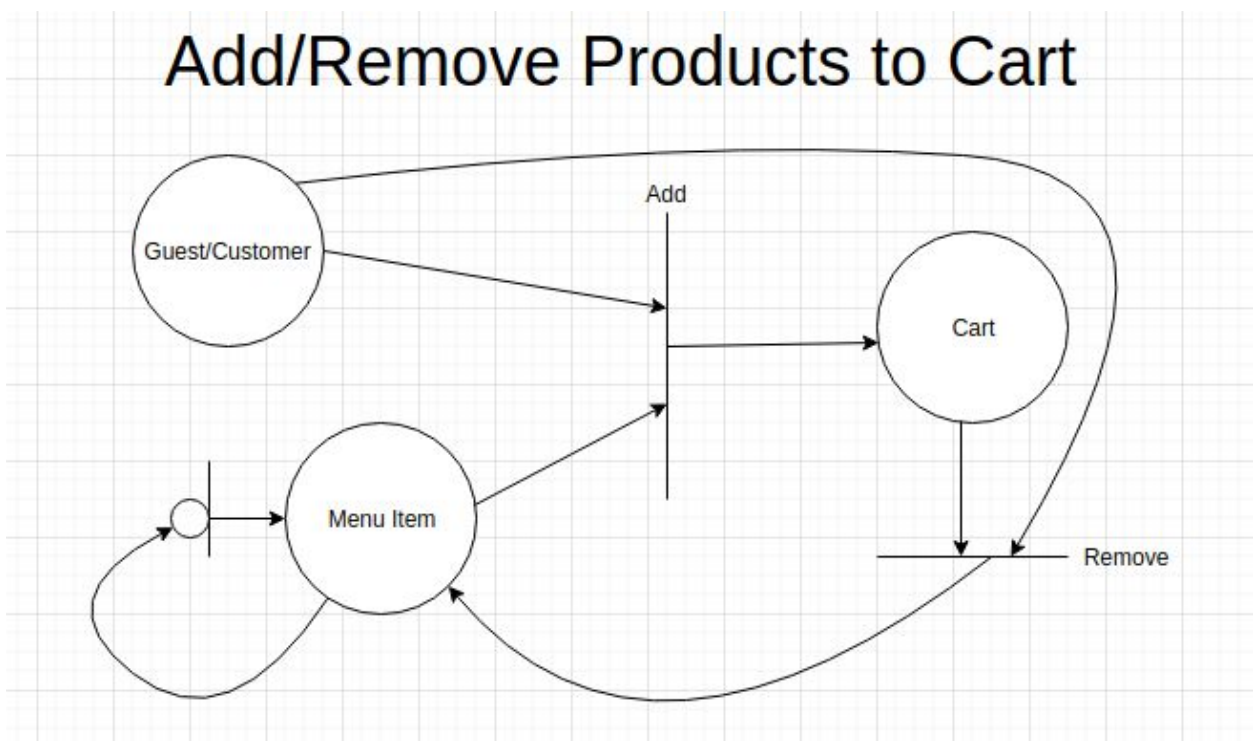


Register

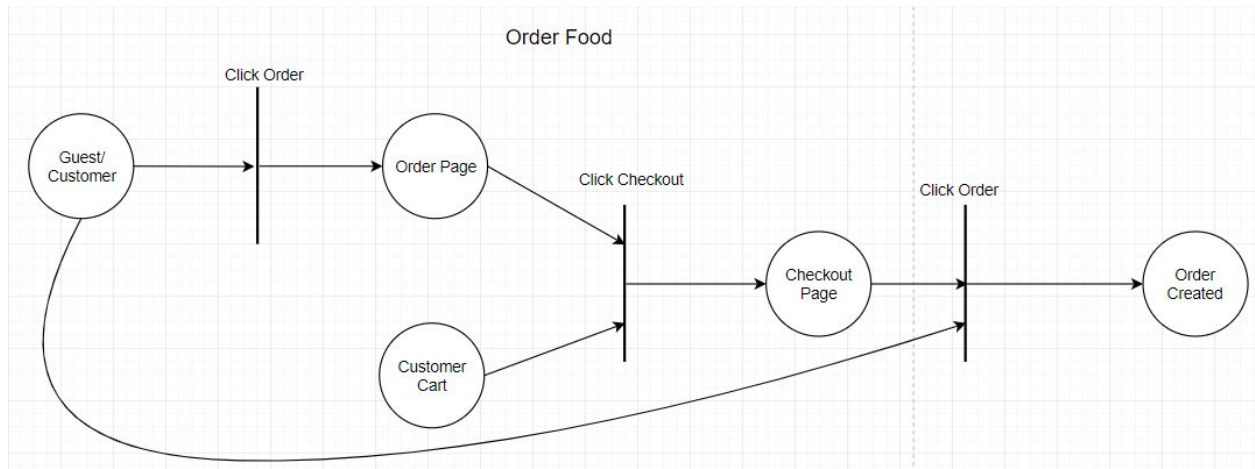


Add/Remove Products to Cart

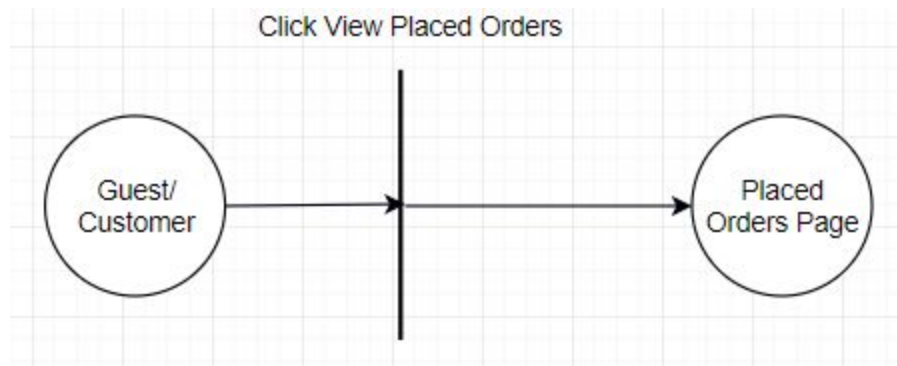
Add/Remove Products to Cart



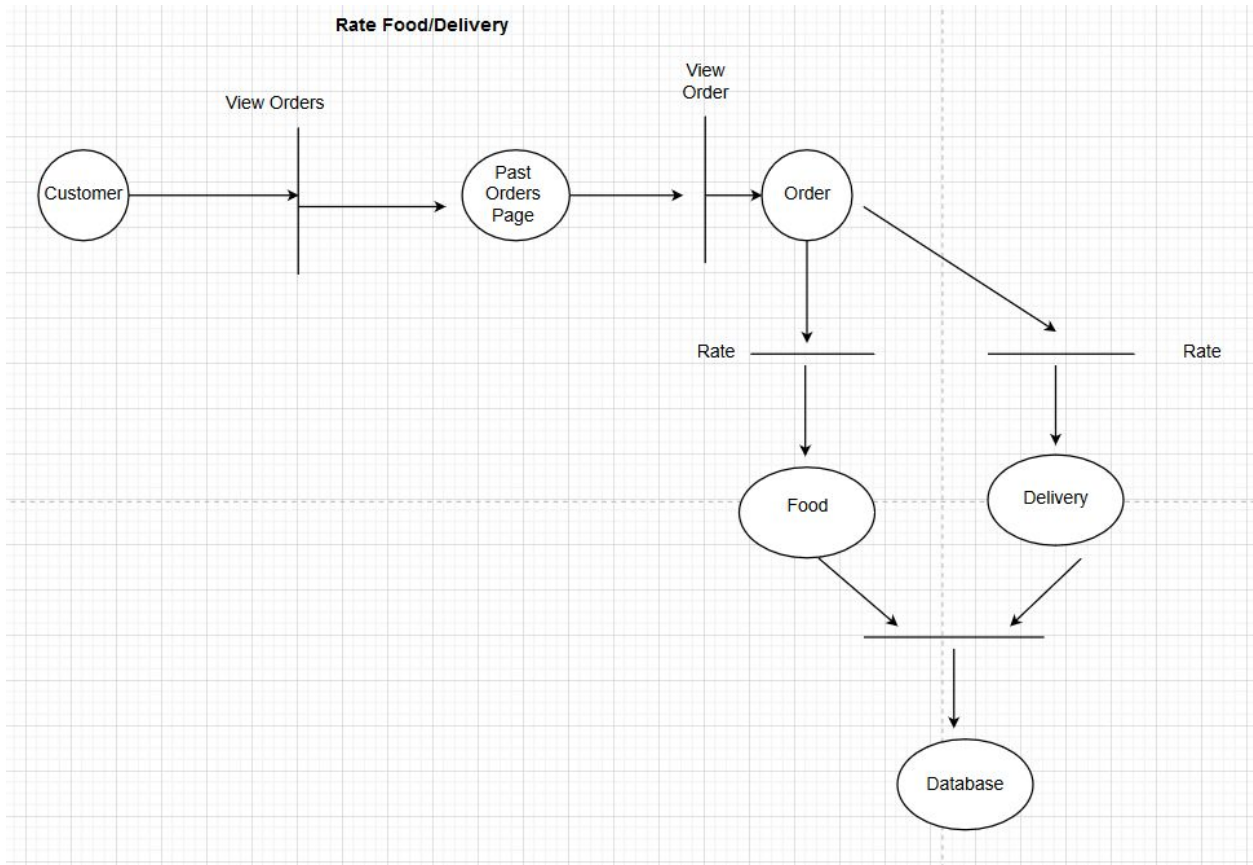
Order Food



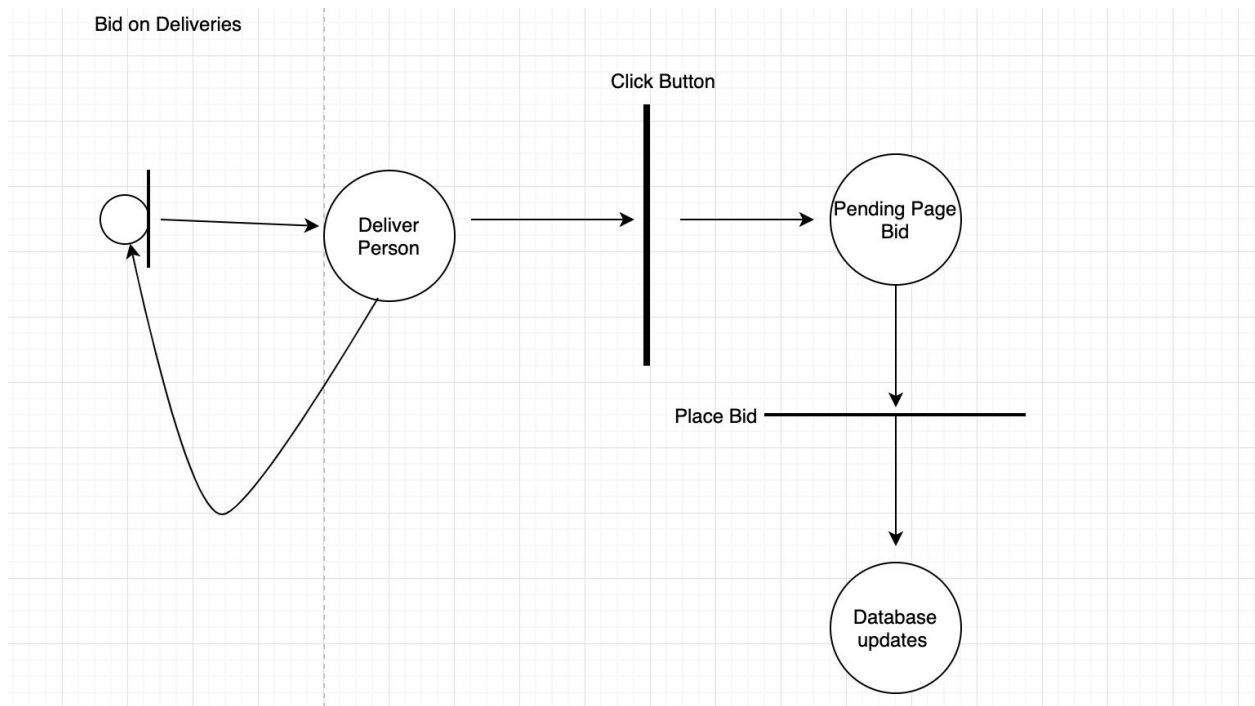
View All Placed Orders



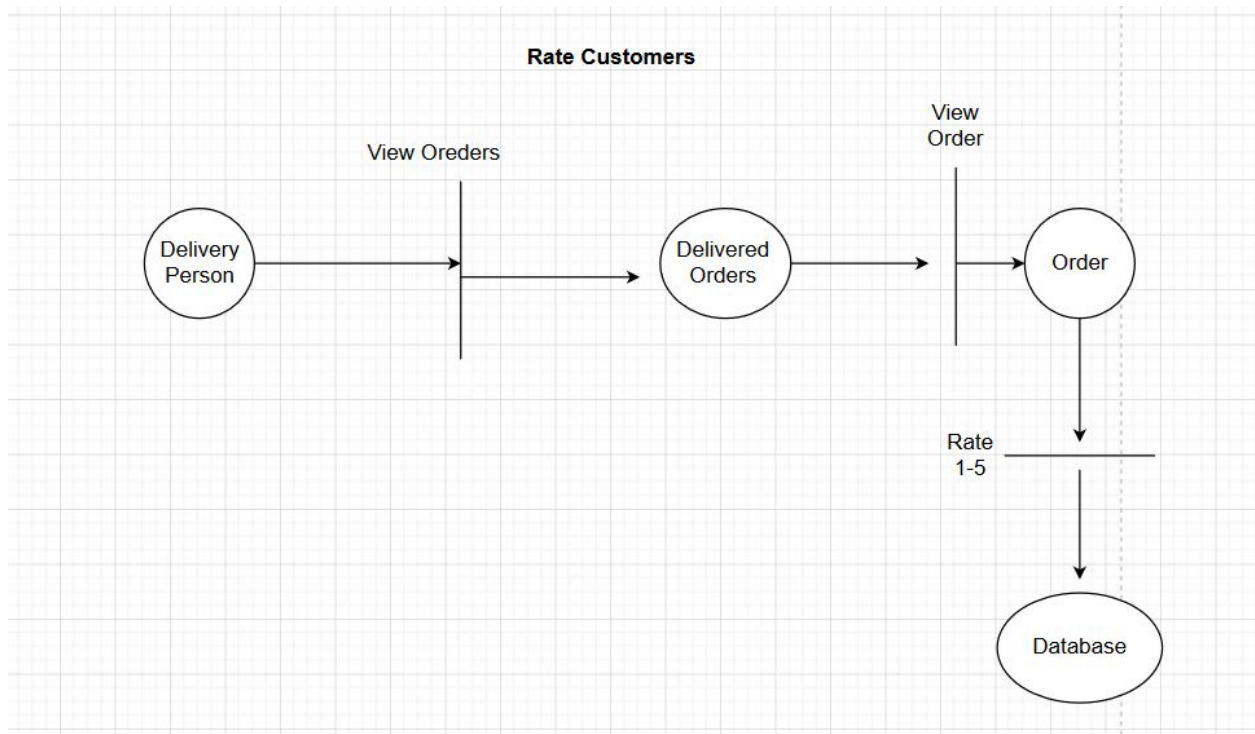
Rate Food/Delivery



Bid on Deliveries

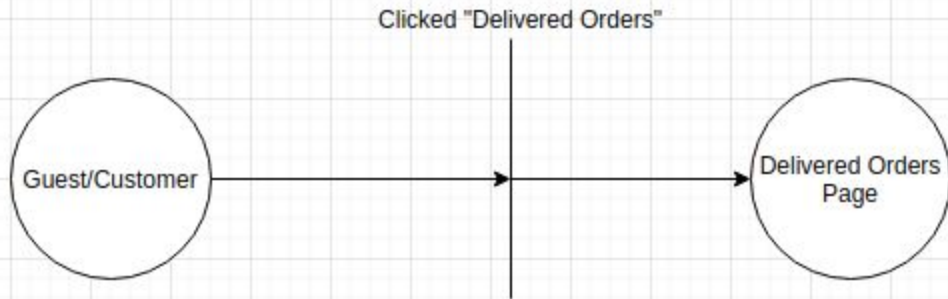


Give Customer Rating

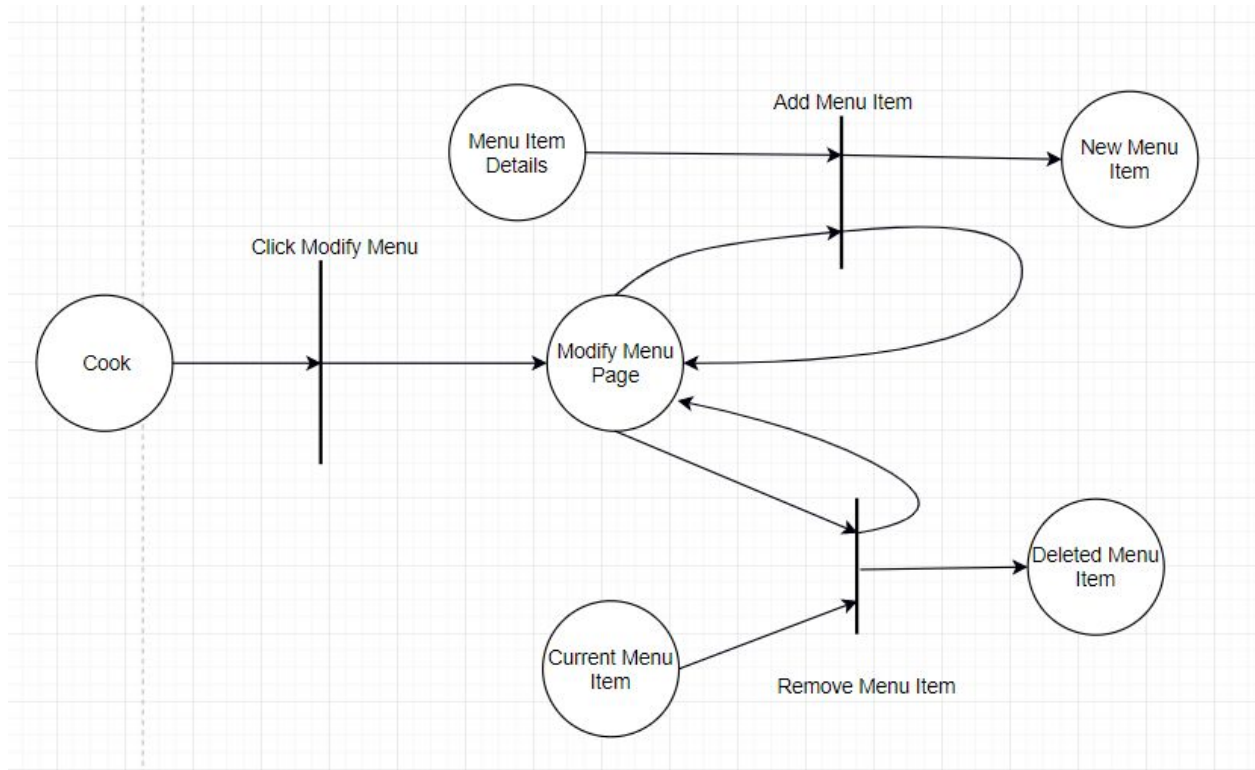


View Delivered Orders

View Delivered Orders

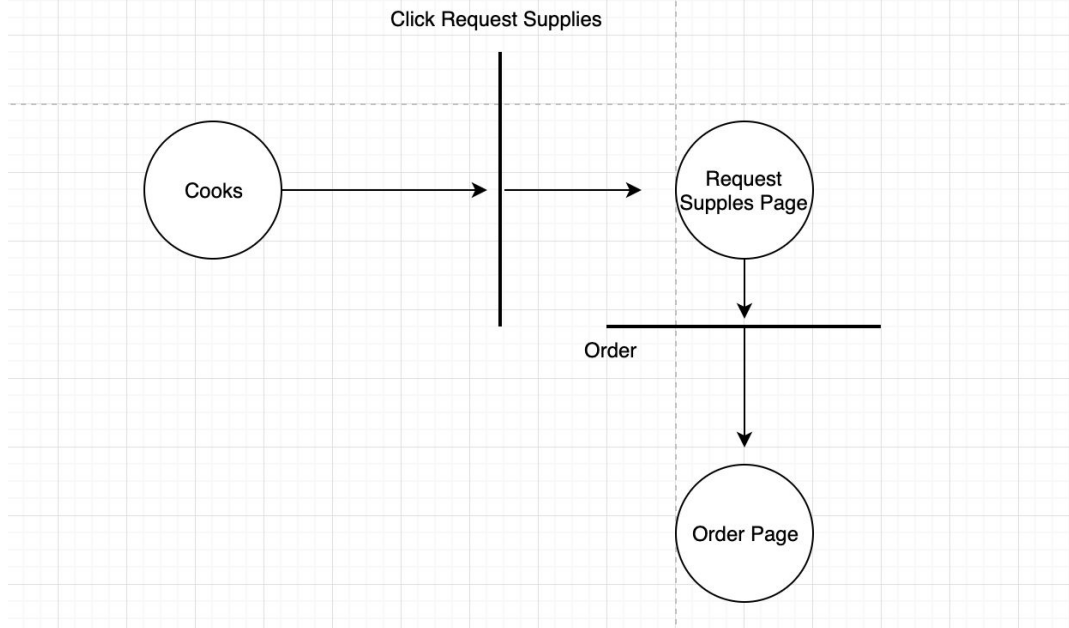


Modify Menu Items

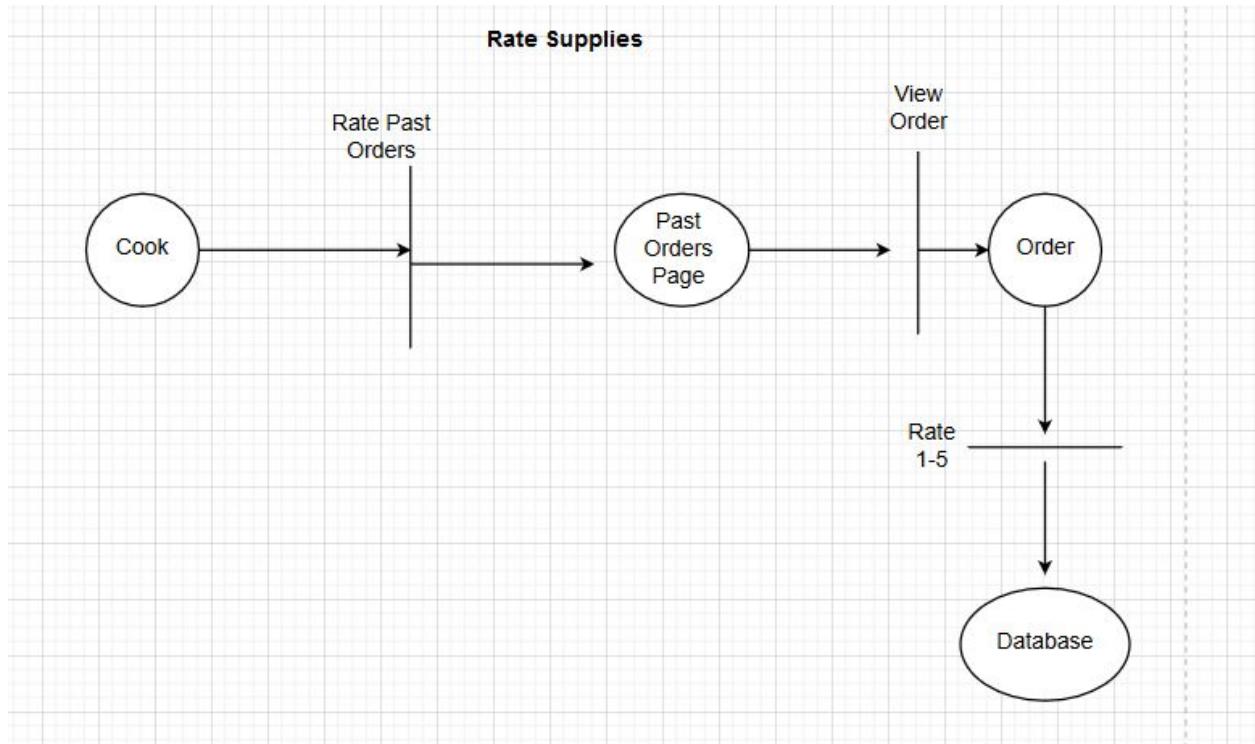


Request Supplies

Request Supplies

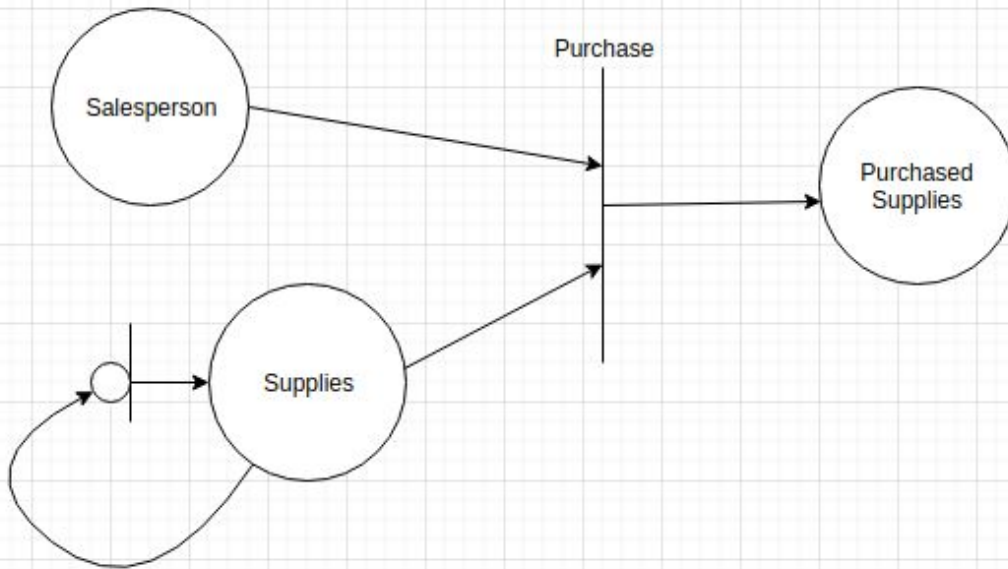


Rate Supplies



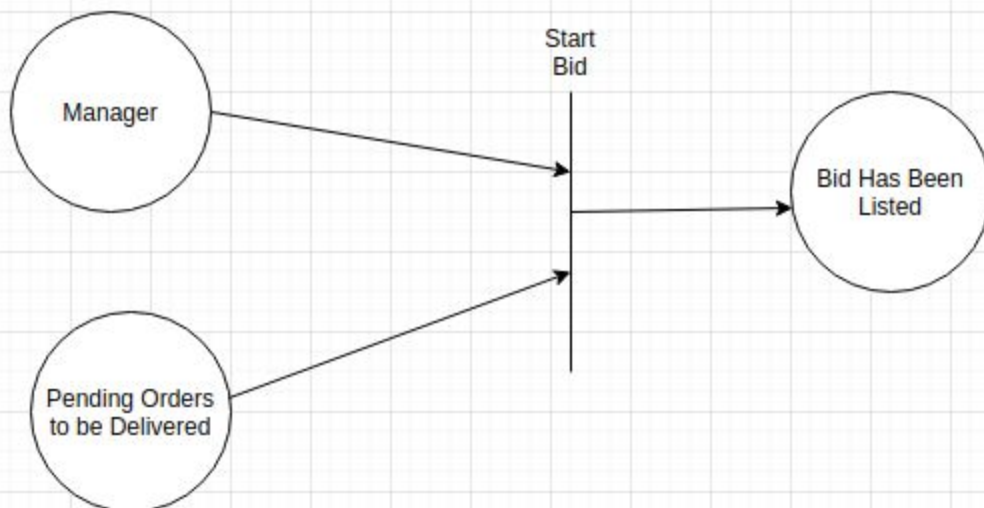
Purchase Supplies

Purchase Supplies

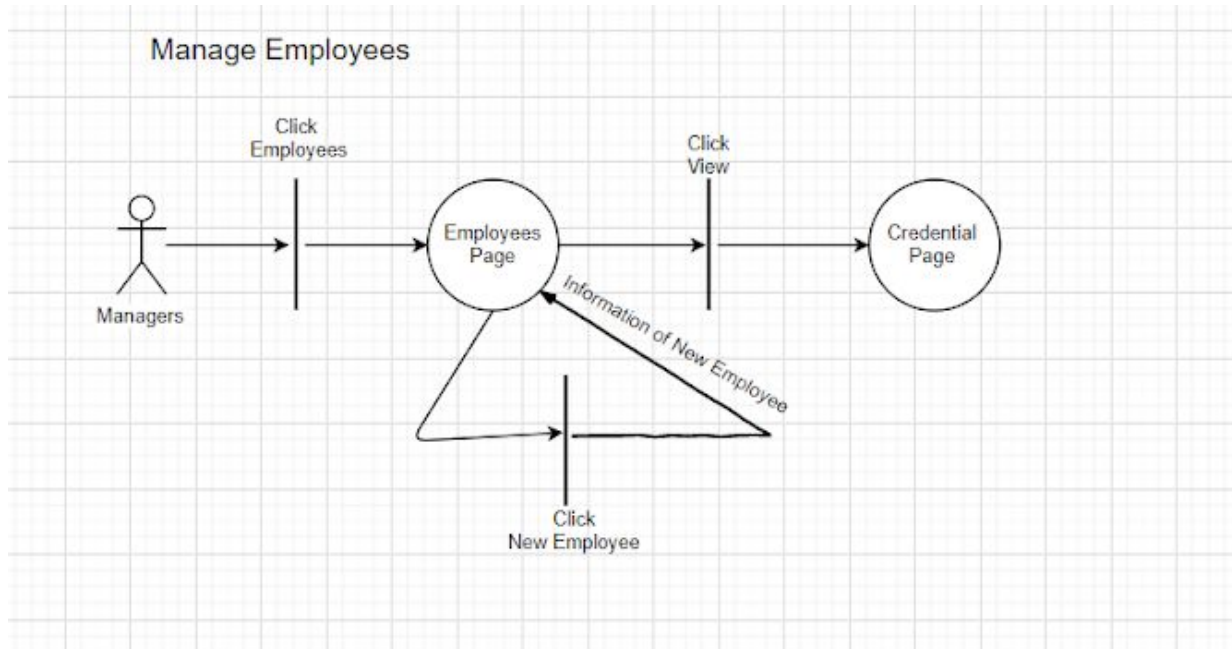


Start Bids

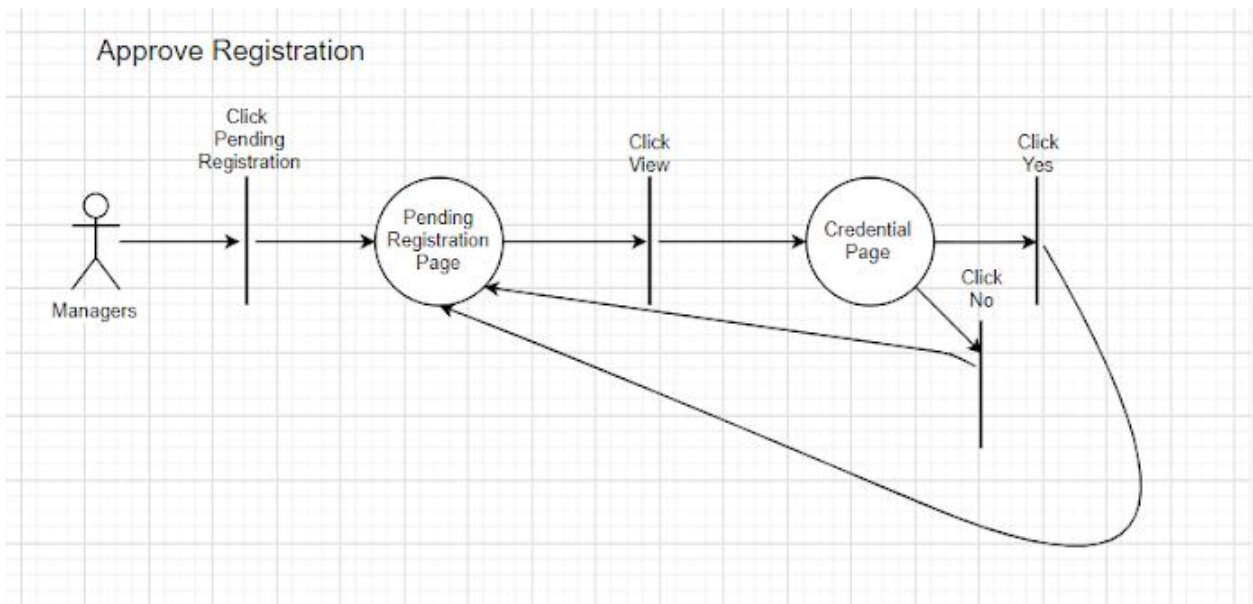
Start Bids



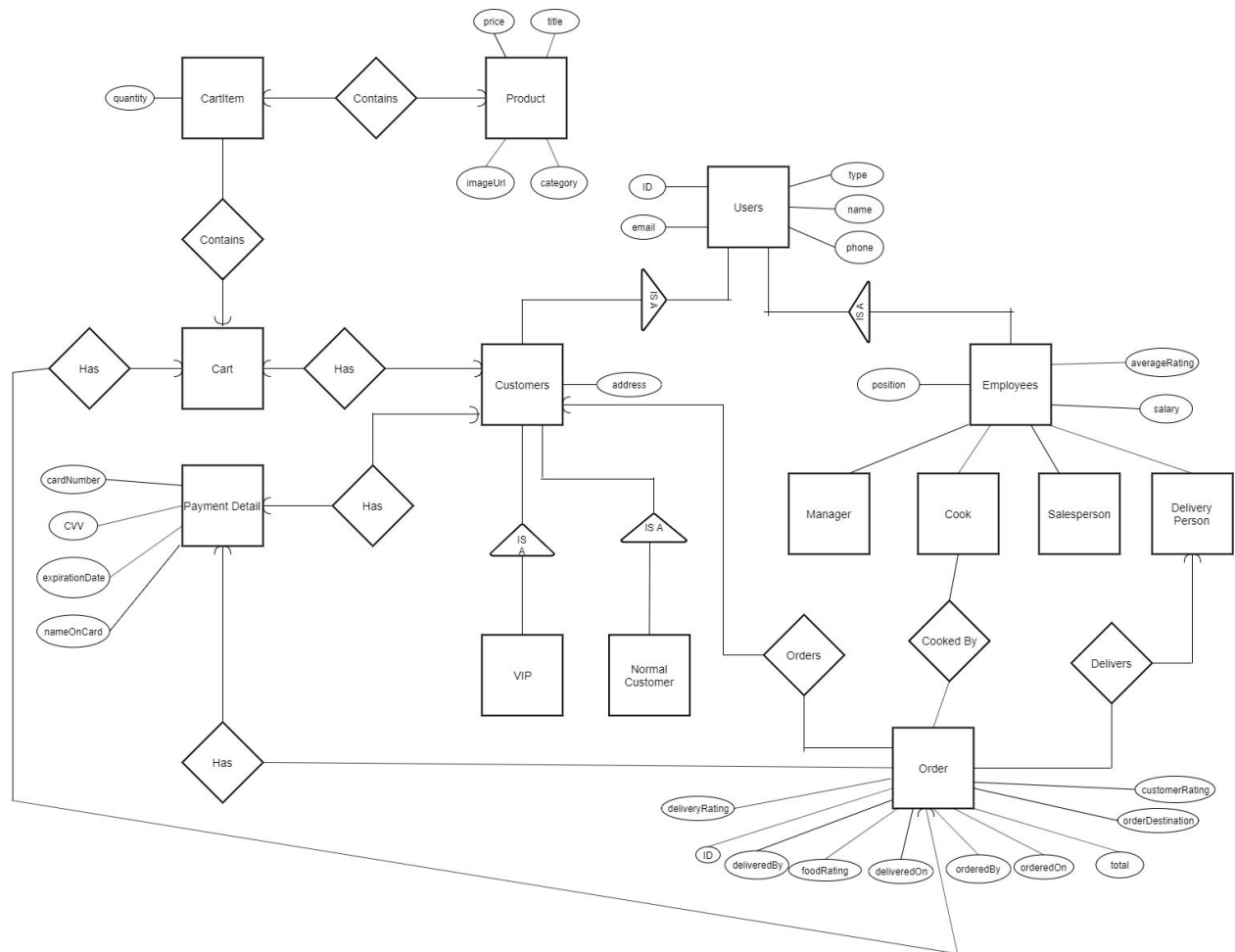
Manage Employees



Approve Registration



3. E/R Diagram



4. Detailed Design

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.email, "123456
").then(userCredentials=>{
    let tempUser:User =
{uid:userCredentials.user.uid,email:userCredentials.user.email,phone:formV
alue.phone,name:formValue.name,type:"employee"};
    this.userService.addUser(tempUser);
    let tempEmployee:Employee =
{uid:userCredentials.user.uid,position:formValue.position,salary: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.userService.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.userService.getUser(uid);
}
```

18. getCurrentUser()

Input: none

Output: user object

Description: Get user object from the database

```
getCurrentUser()
{
    return this.authService.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.authService.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 Screens


Main Page




Order Page

Kitchen Zealot


MenuLogin




Beef with Broccoli
\$8.00




Chicken Chow Mein
\$6.00




General Tso's Chicken
\$9.00



Kung Pao Chicken
\$7.00



Orange Chicken
\$8.00




Beef with Broccoli
\$8.00


Cart

Kitchen Zealot


OrderCustomer




Beef with Broccoli
\$8.00
Add to Cart




Chicken Chow Mein
\$6.00
Add to Cart




General Tso's Chicken
\$9.00
1 in cart



Kung Pao Chicken
\$7.00



Orange Chicken
\$8.00



Beef with Broccoli
\$8.00

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

Order

CheckOut Page

Kitchen Zealot

Order Customer ▾

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	

6. Minutes

Meeting # 1 (3 hours) : We discussed who are team members are, their skill levels, basic planning of the system, what languages, technologies to learn and use, when we would have future checkpoints, and what parts and roles each of us would fill in

Meeting # 2 (6 hours): We discussed and worked on the Phase 1 template report, and equally distributed the work while bouncing ideas off of each other and improving on it.

Meeting # 3 (6 hours): We discussed and worked on the Phase 2 template report, and equally distributed the work. We each made a handful of diagrams for the use-case and worked together for the general collaboration class diagram.

7. Git Repository

<https://github.com/AbtahiChowdhury/csc322>