**What’s Cooking**

**Meal Spot**

**Software Requirements Specification**

**For Online Restaurant System**

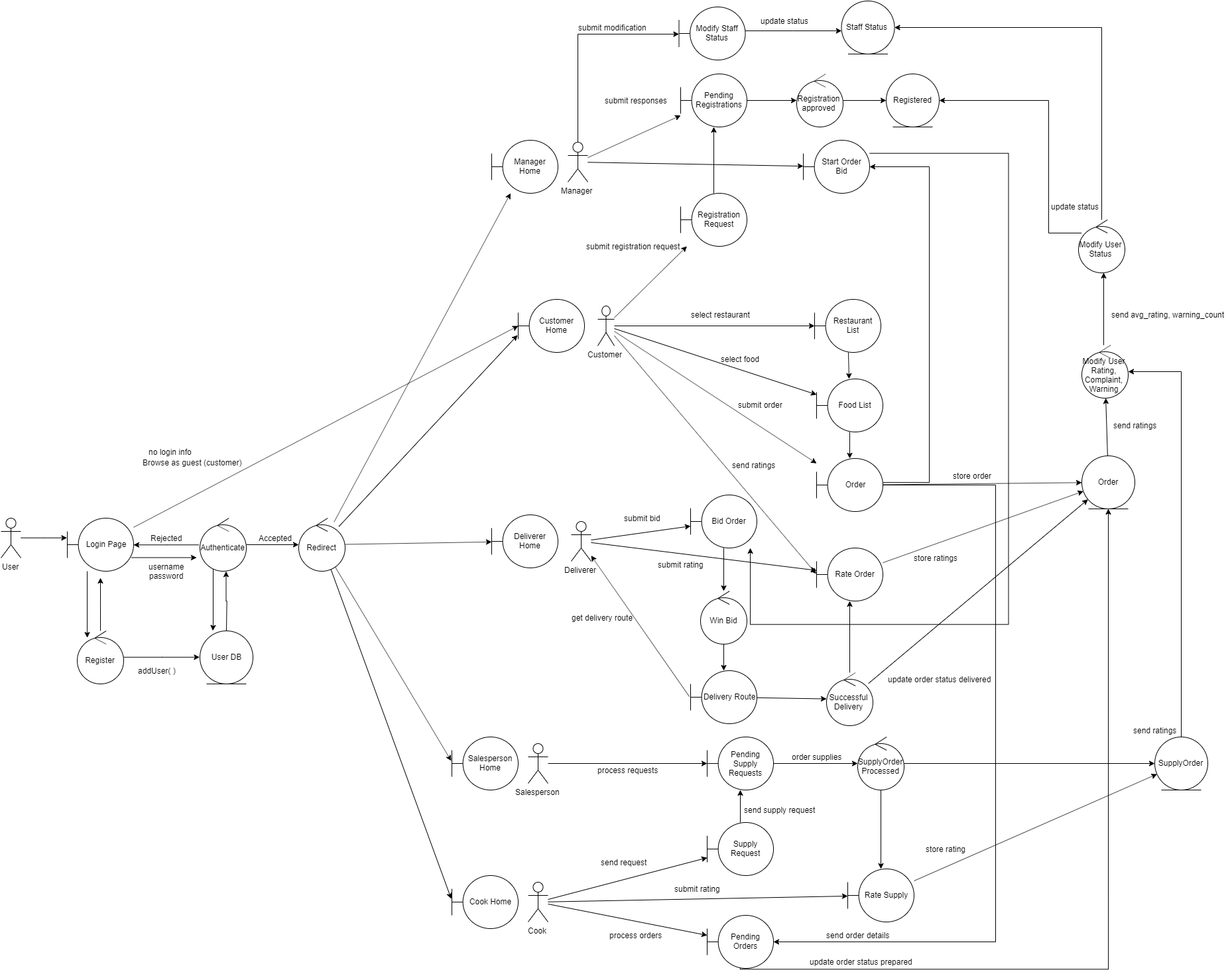
**Version 2.0**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 11/04/19 | 1.0 | Starting design for our application. Use-Case and ER diagrams | Oren Ben-Meir, Maui Arcuri, Florence Fong, Warin Wohab |
| 11/27/19 | 2.0 | Finalizing detailed design for our site.  Use-Case Scenarios with diagrams. Methods with pseudo-code. System screens with pictorial displays. | Oren Ben-Meir, Maui Arcuri, Florence Fong, Warin Wohab |

**Introduction**

Collaboration Class Diagram of Overall System



**Use Cases**

1. **User Login / Register**
   1. Scenarios
      1. Success:
         1. User enters email and password
         2. User successfully logs in
         3. Based on what user type they are, they are redirected to their appropriate page
      2. Alternative:

1a. User does not put in login credentials (leaves blank), becomes Guest Customer

1b. User is asked if want to register for account

* + - * + i. if yes, send to signup page

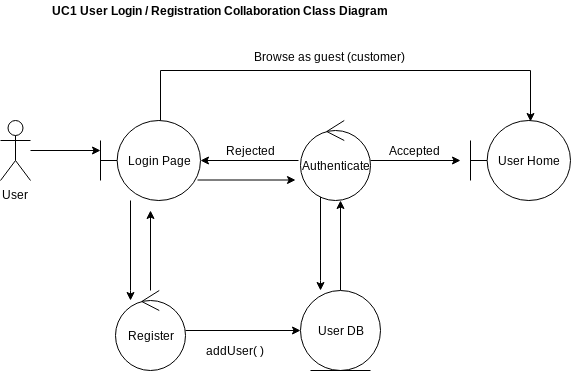
User signs up for an account (firstName, lastName, email, password, userType)

* + - * + ii. if no, send to List of Restaurants page
    1. Exception:

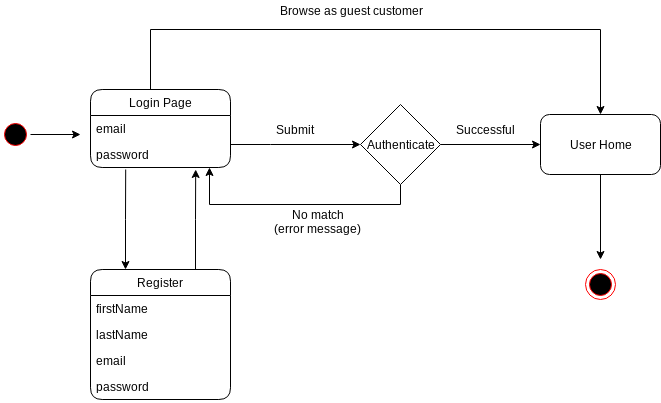
1a. User puts invalid (wrong) login credentials.

1b. error msg of invalid credentials

* 1. Collaboration Class Diagram



* 1. State diagram or Petri-net

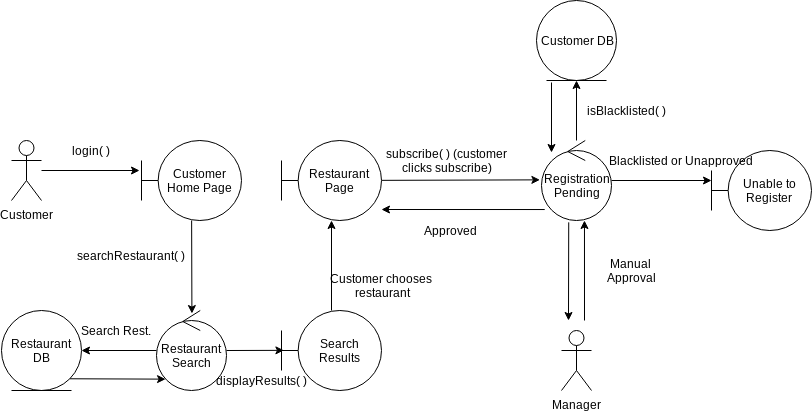


1. **Restaurant Registration**
   1. Scenarios
      1. Success:
         1. Customer logs in
         2. Customer searches for restaurant
         3. Customer clicks to register
         4. Manager approves of registration
      2. Exception:

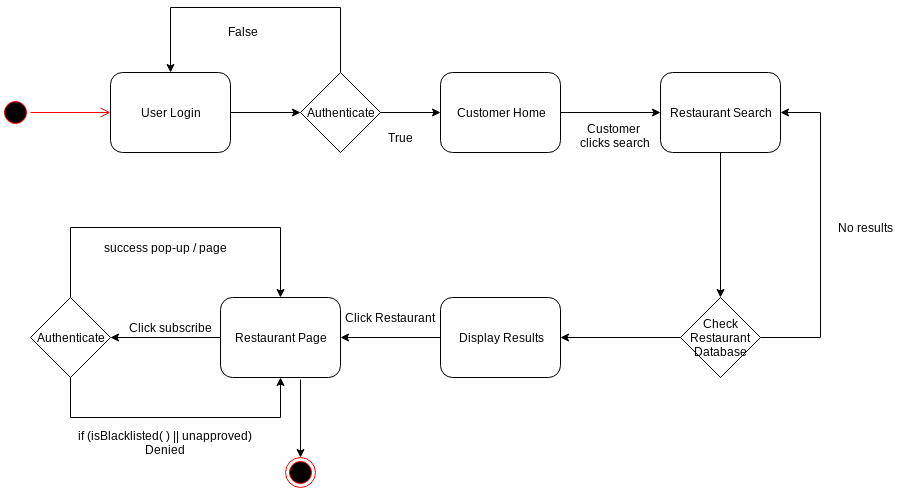
4a. Customer registration is not approved

4b. Customer is blacklisted and registration is automatically rejected

* 1. Collaboration Class Diagram



* 1. State Diagram

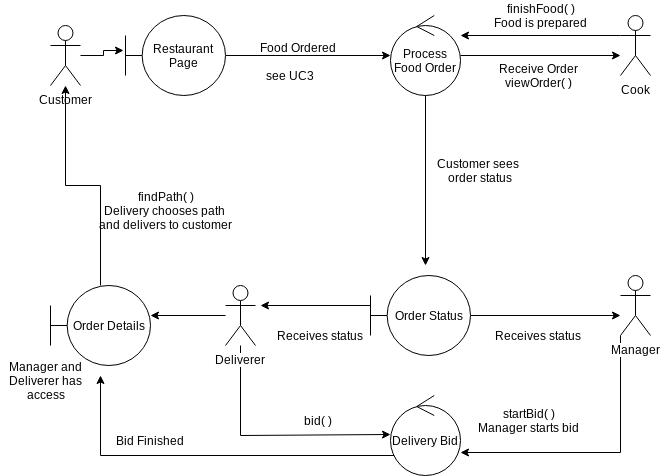


1. **Process Food Order**
   1. Scenarios
      1. Success:
         1. Customer orders from restaurant (payment is successful). Order status == requested/pending
         2. Cook sees Order request. Cook prepares foods and submits once done (order status == prepared)
         3. Manager starts a bid for the Deliverer
         4. Deliverer bids the order
         5. Manager chooses the Deliverer, lowest bid
         6. Deliverer picks up order (order status == in transit), chooses the best route with findPath(), and delivers order successfully.
         7. Order status == delivered/complete
      2. Exception:

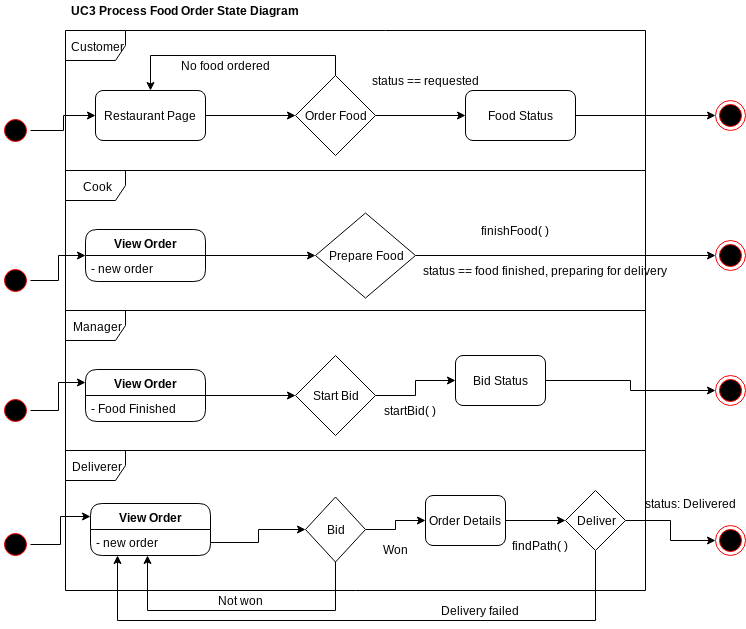
4a. No deliverer bids

6a. Delivery is unsuccessful

* + - * + can’t get in contact with Customer.
        + delivery to wrong address/person
  1. Collaboration Class Diagram



* 1. State diagram or Petri-net



1. **Customer Sends Order Request to Restaurant (Details and Processing)**
   1. Scenarios
      1. Success:
         1. Customer views menu of restaurant,
         2. Customer selects food and reads reviews, adds food to order (cart)
         3. Customer clicks to view Checkout and Complete/submit order
         4. Customer chooses payment method
         5. Customer fills in payment form (with option to save)
         6. Customer completes order, order is sent to system/restaurant
      2. Alternative:

1a. i. Customer status = Registered

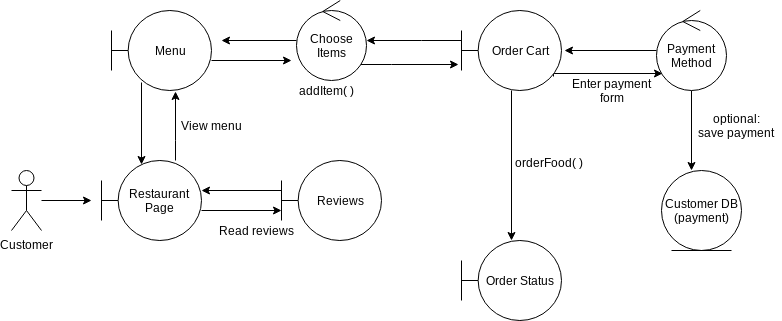
* + - * + Gets discounted price.
        + View menu differently by personal order history. Top 3 food items previously ordered are at top

ii. Customer = VIP

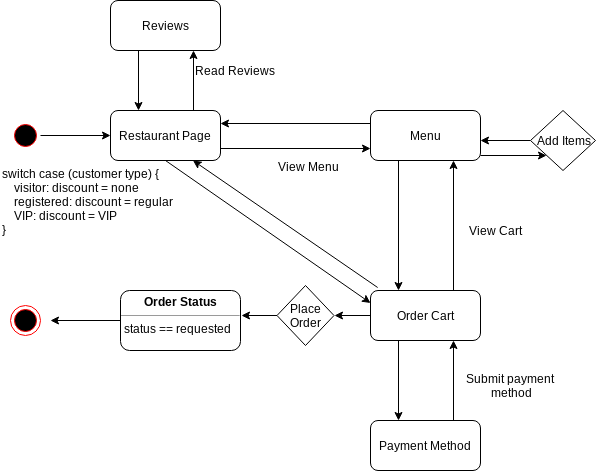
* + - * + Gets higher discount and free food item

iii. Customer = Visitor

* + - * + Views menu by top 3 most ordered
  1. Collaboration Class Diagram



* 1. State diagram or Petri-net



1. **Customer Rate Order**
   1. Scenarios
      1. Success:
         1. Customer rates food (Cook) 1-5, delivery (Deliverer) 1-5.
         2. Customer Ratings saved into Order
         3. Rating updates Food rating history. Rating updates Deliverer rating history.
      2. Alternative:

1a. For each, if rating <=2, must write a complaint sentence

3a. FoodRating == 1. Food has 2 previous rating == 1. Food is dropped. Cook foodDropCount++.

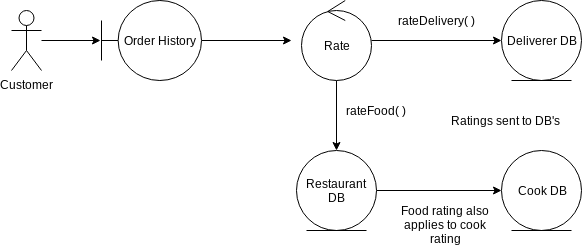
* + - * + if foodDropCount%2 == 0, warningCount++
        + if warningCount == 4, Cook fired

3b. DeliveryRating == 1. Deliverer has 2 previous rating == 1. Deliverer warningCount++.

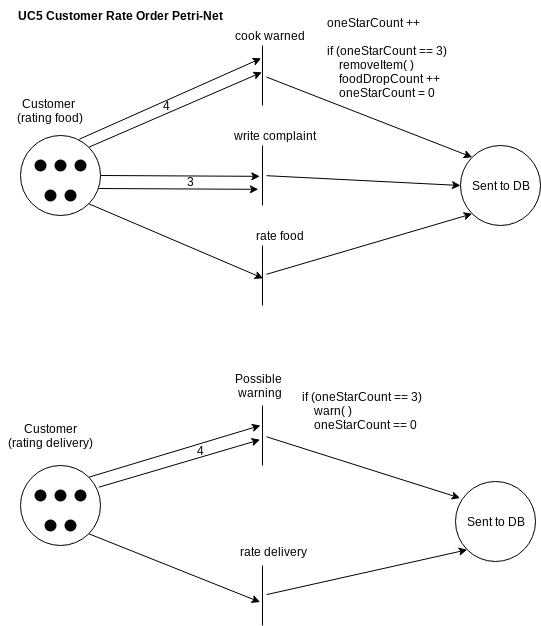
* + - * + if warningCount == 4, Deliverer fired
    1. Exception:

3a. Food item deleted before Rating is saved into Food rating history

* 1. Collaboration Class Diagram



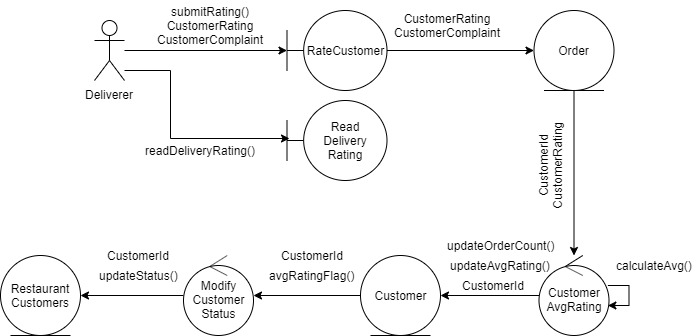
* 1. Petri-Net Diagram



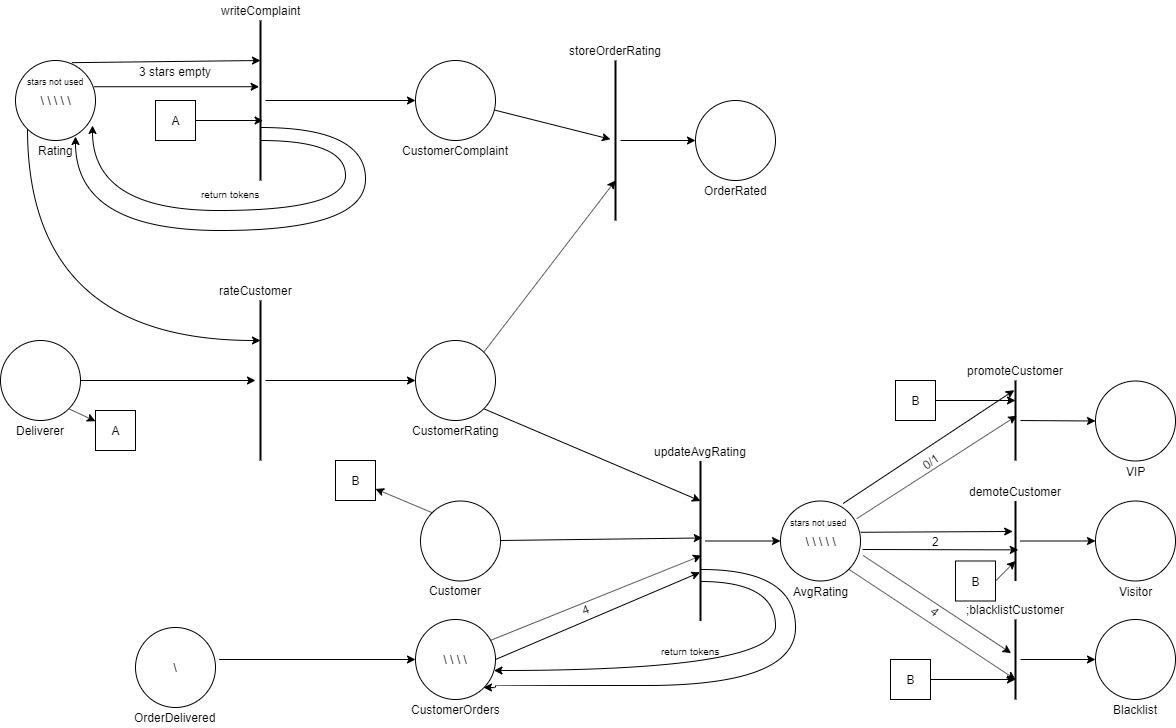
1. **Deliverer Rate Customer**
   1. Scenarios
      1. Success:
         1. Deliverer rates Customer 1-5
         2. Deliverer Rating saved into Order
         3. Rating updates Customer rating history and avgRating. May affect Customer status.
      2. Alternative:

3a. Customer OrderCount >= 4. AvgRating = x:

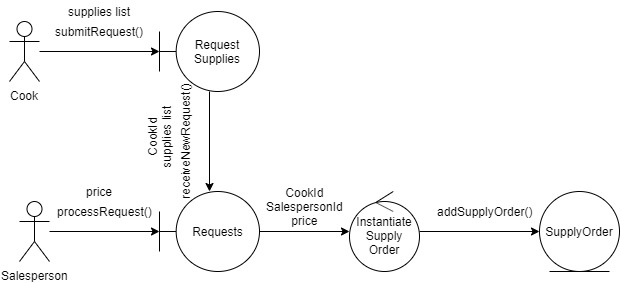
* + - * + x > 4 : Customer is VIP
        + 1 < x < 2 : Customer loses registration, is Visitor
        + x == 1 : Customer is Blacklist
  1. Collaboration Class Diagram



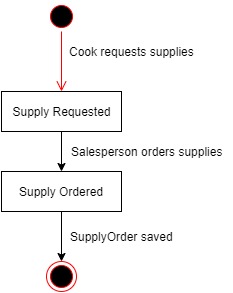
* 1. Petri-net



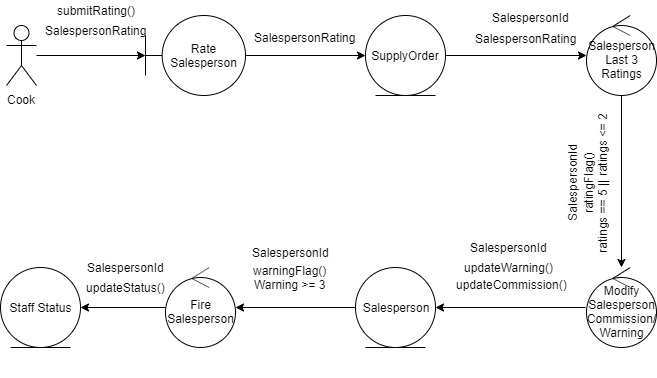
1. **Cook Order Supplies**
   1. Scenarios
      1. Success:
         1. Cook requests supplier to order supplies
         2. Salesperson orders supplies
         3. Supply Order saved
   2. Collaboration Class Diagram



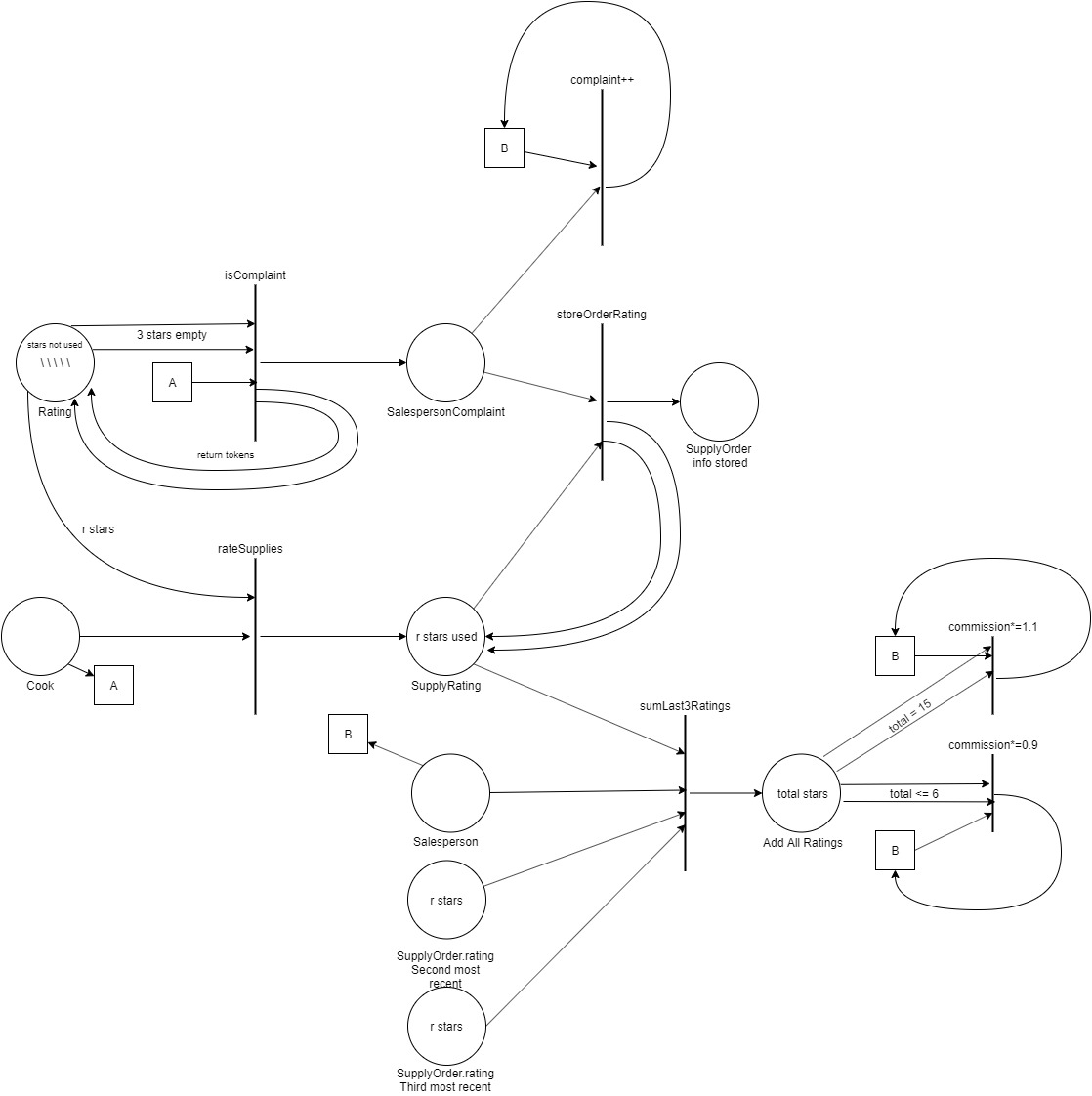
* 1. State diagram



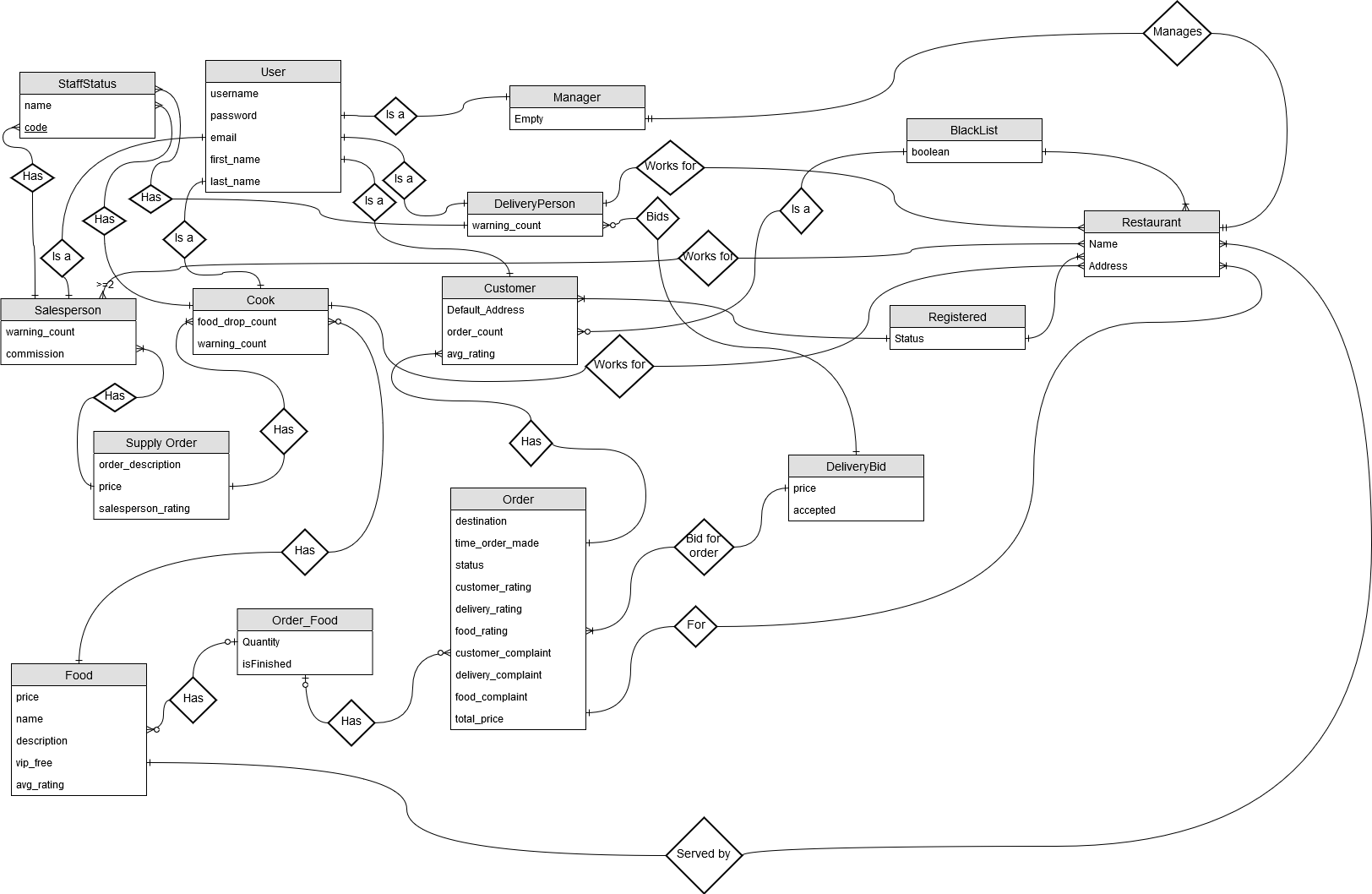
1. **Cook Rate Supplies**
   1. Scenarios
      1. Success:
         1. Cook rates supplies 1-5
         2. SupplyRating saved into SupplyOrder
         3. Salesperson rating history updated. May affect Salesperson commission/staff status.
   2. Collaboration Class Diagram



* 1. Petri-net



**E-R Diagram (Entire System)**



**Detailed Design**

Methods with pseudo-code

void searchRestaurant(string s){

output t;

t = select name from restaurants where name = s;

displayResults(t);

}

void subscribe(customer c) {

if(isBlackListed(c)) {

c.customer\_status = denied;

}

else {

c.customer\_status = pending;

}

}

bool isBlackListed(customer c, restaurant r) {

return (select c from blacklist where restaurant is r).length > 0

}

void finishFood(order o) {

o.status = prepared; //ready for delivery

}

void startBid(order o) {

o.bid\_status = true;

}

void bid(float bid) {

if(bid > o.bid\_price){

o.bid\_price = bid;

}

}

findPath(address restaurant, address customer,

method deliverer.transportation\_type)

{

//use map api to find path between the two restaurants

//using transportation type

}

//o.items = {item: quantity}

void addItem(Item i, Order o) {

if(contains(i, o.items)){

//increment item quantity

}

else o.items.add(i:1);

}

void removeItem(Item i, order o) {

if(contains(i, o.items) and quantity > 1) {

quantity --;

}

else o.items.remove(i);

}

void orderFood(order o) {

o.status = pending; //ready for cook

}

void rateDelivery(int rating, order o) {

o.deliverer.ratings.add(rating);

if(oneStarCount(o.deliverer) == 3){

o.deliverer.warnings ++;

}

if(o.deliverer.warnings == 4) {

fire(o.deliverer);

}

}

int oneStarCount(user u) {

int count = 0;

for (int i = 0; i < u.ratings; i ++) {

if(u.ratings[i] == 0) {

count ++;

}

}

return count;

}

void rateFood(int rating, item i, order o) {

i.ratings.add(rating);

o.cook.ratings.add(rating);

if (oneStarCount(i) == 3){

remove(item);

o.cook.dropCount ++;

}

if (o.cook.dropCount == 2) {

warn(o.cook);

}

if (o.cook.warningCount == 4){

fire(o.cook);

}

}

void warn(user u) {

//send warning to user

u.warningCount ++;

}

void fire(user u) {

delete u;

}

void rateSupplies(int rating, SupplyOrder o) {

o.salesperson.ratings.add(rating);

if(rating <= 2){

o.salesperson.complaintCount++;

if(complaintCount(o.salesperson)%3 == 0){

o.deliverer.warnings ++;

}

}

if(ratings[n], ratings[n-1], ratings[n-2] <= 2){

o.salesperson.commission \*= 0.9;

}

elseif(rating[n], rating[n-1], rating[n-2] == 5){

o.salesperson.commission \*= 1.1;

}

if(o.salesperson.warnings == 3) {

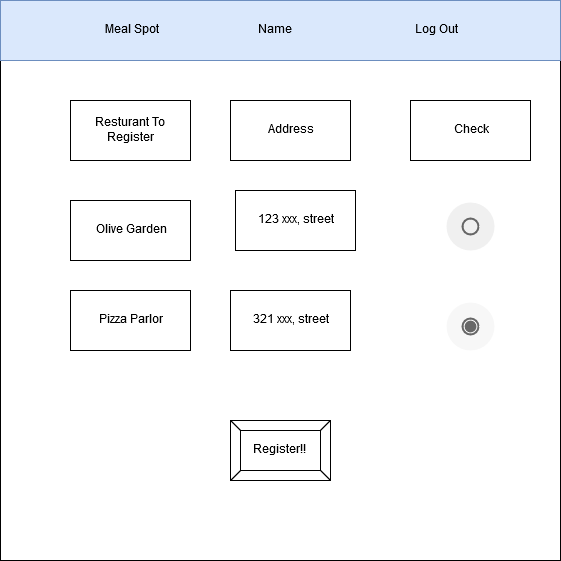
fire(o.salesperson);

}

}

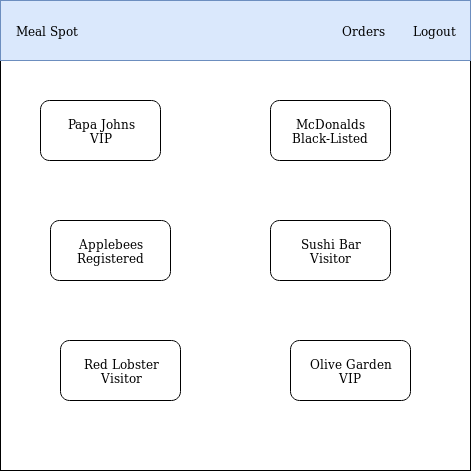
**System Screens**

1. Staff Registration

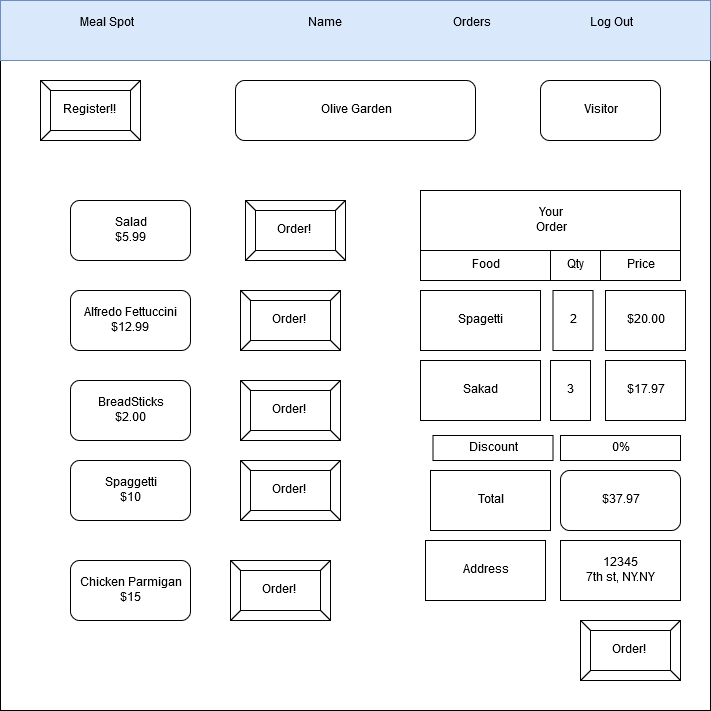


*Staff Restaurant Registration Page*

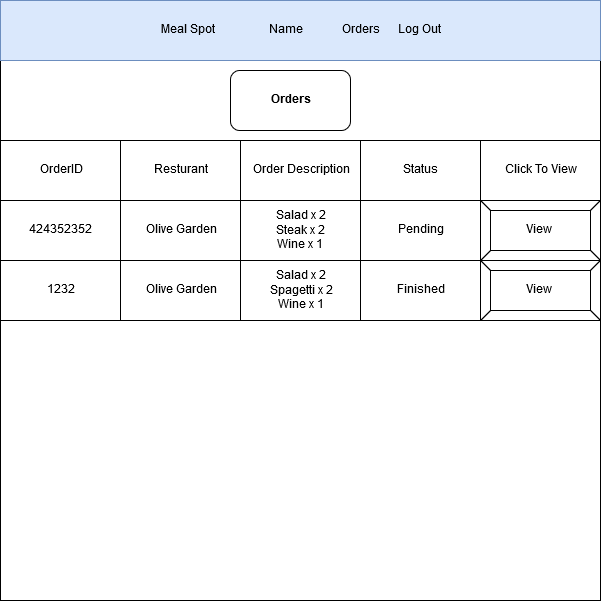
1. Customer



*Customer Home Page*

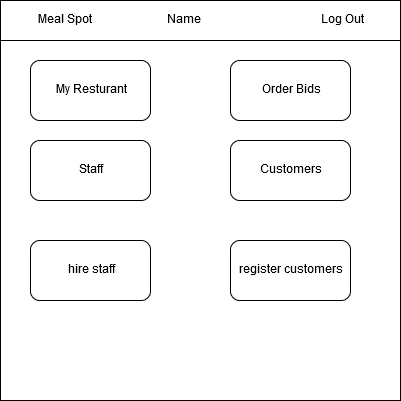


*Customer Restaurant Page*

**

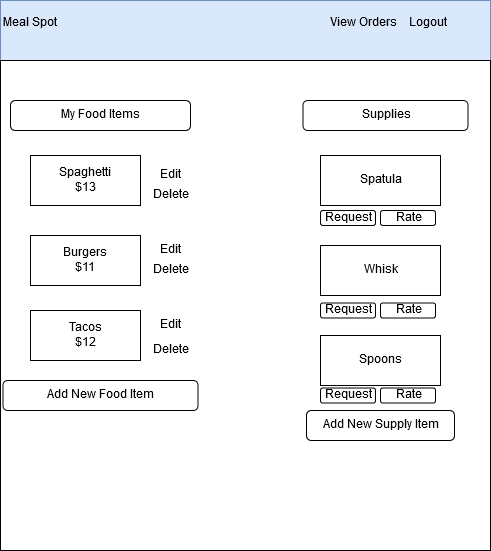
*Customer OrdersPage*

1. Manager

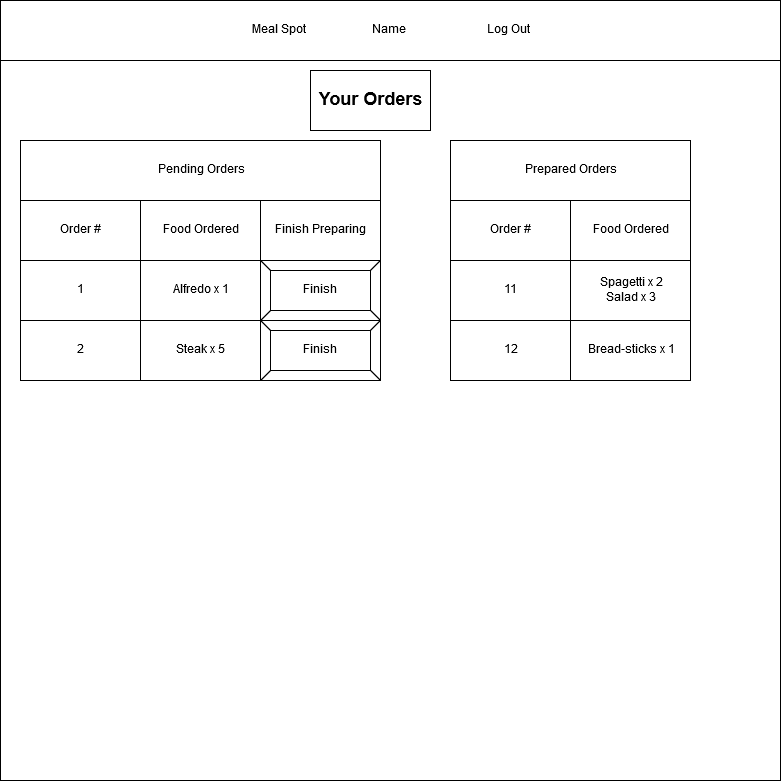


*Manager Home Page*

1. Cook

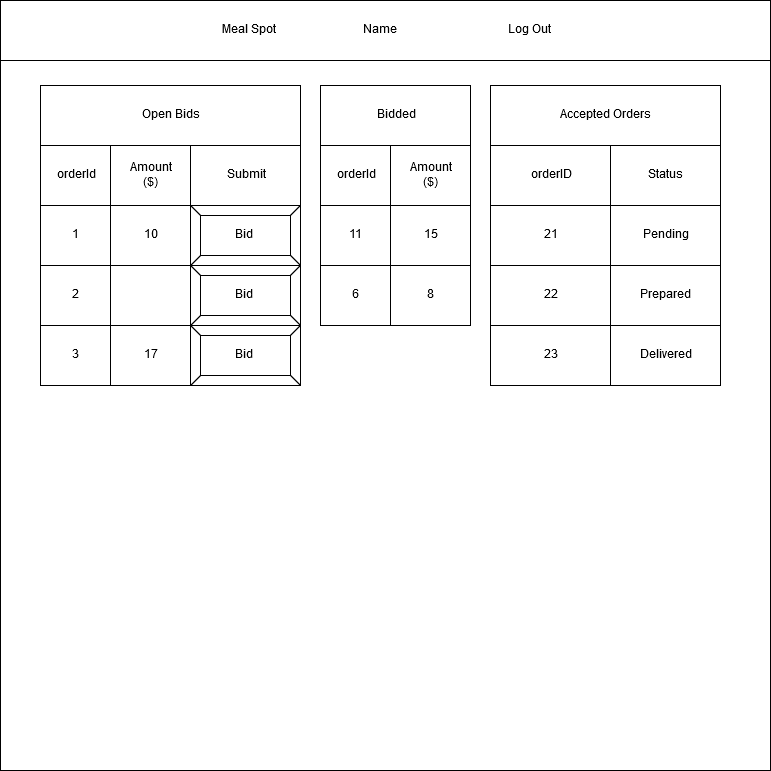


*CookHome Page*

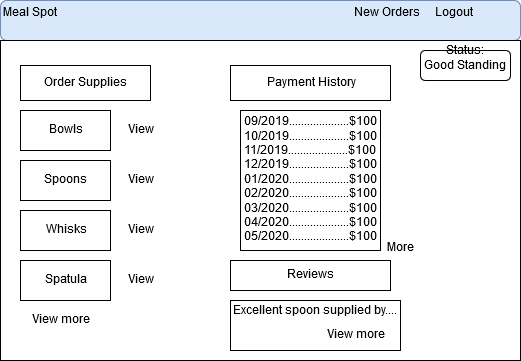
**

*Cook Orders*

1. Deliverer

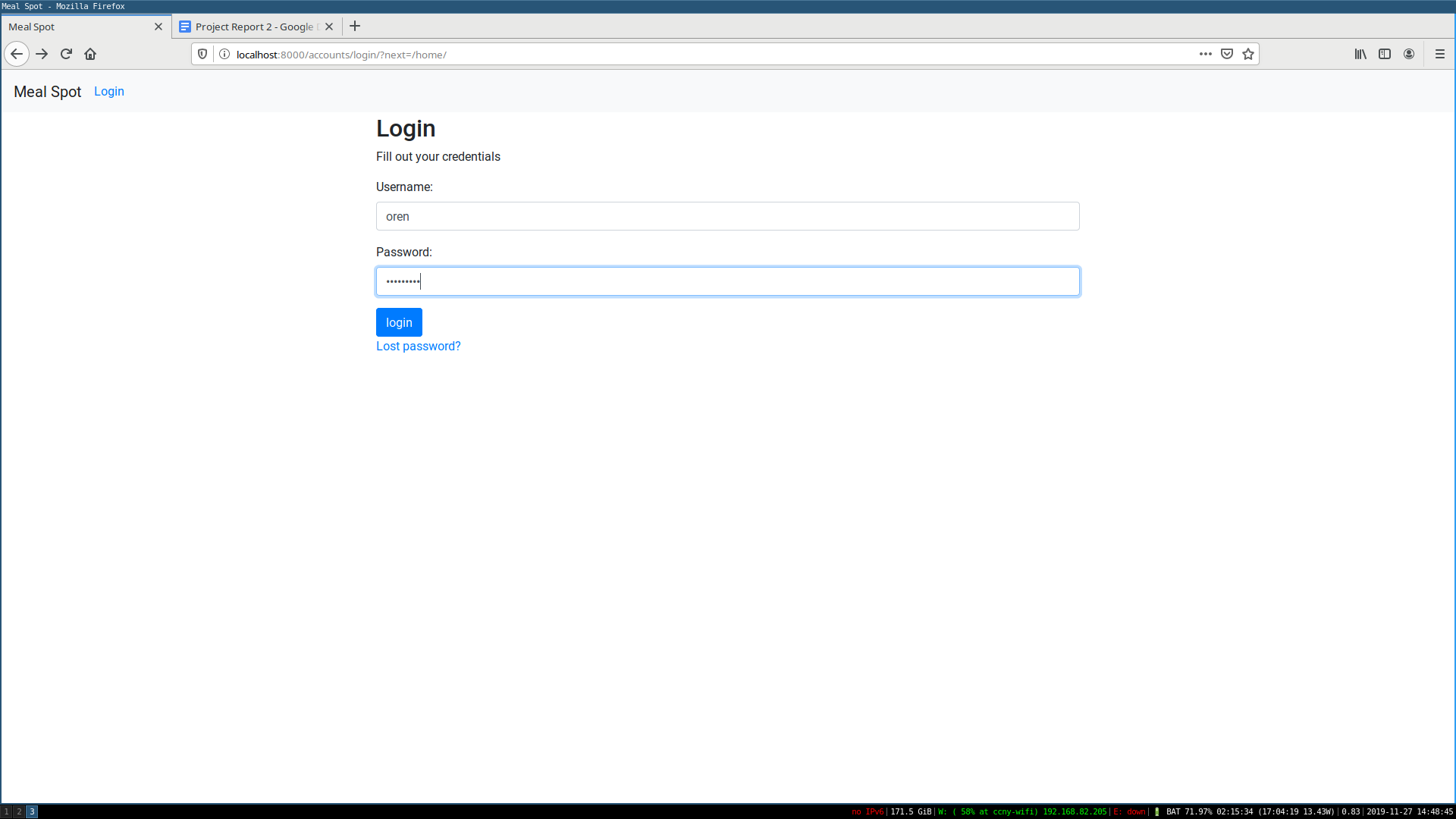


1. Salesperson



*SalesPersonPage*

1. Prototype of Login



*Login Page (Prototype)*

**Team Meeting**

1. Time Track

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Date | Meetup Type  (Hangouts,  In Person) | Members | Clock In | Clock Out | Total  (minutes) |
| 11/3 | Hangouts | O.B., W.W., F.F., M.A. | 14:00 | 16:00 | 120 |
| 11/24 | Hangouts | O.B., F.F., M.A. | 19:00 | 21:00 | 120 |
| 11/25 | In Person | O.B., W.W., F.F., M.A. | 15:15 | 19:00 | 165 |
| 11/27 | In Person | O.B., F.F., M.A. | 15:15 | 19:30 | 255 |

1. Concerns

* Not many meetings until nearing deadline.
* Need to have more scheduled weekly meetings, especially in person. Meetings need to be less spontaneous and have designated days/hours.
* Difficulty with syncing with all team members’ schedules. Not all members present full-time of meeting.

**GitHub Link**

<https://github.com/OrenBen-Meir/Meal-Spot>