CSC 32200 Team M

Foodtopia
Software Requirements Specification
For Restaurant System

Version 2.0

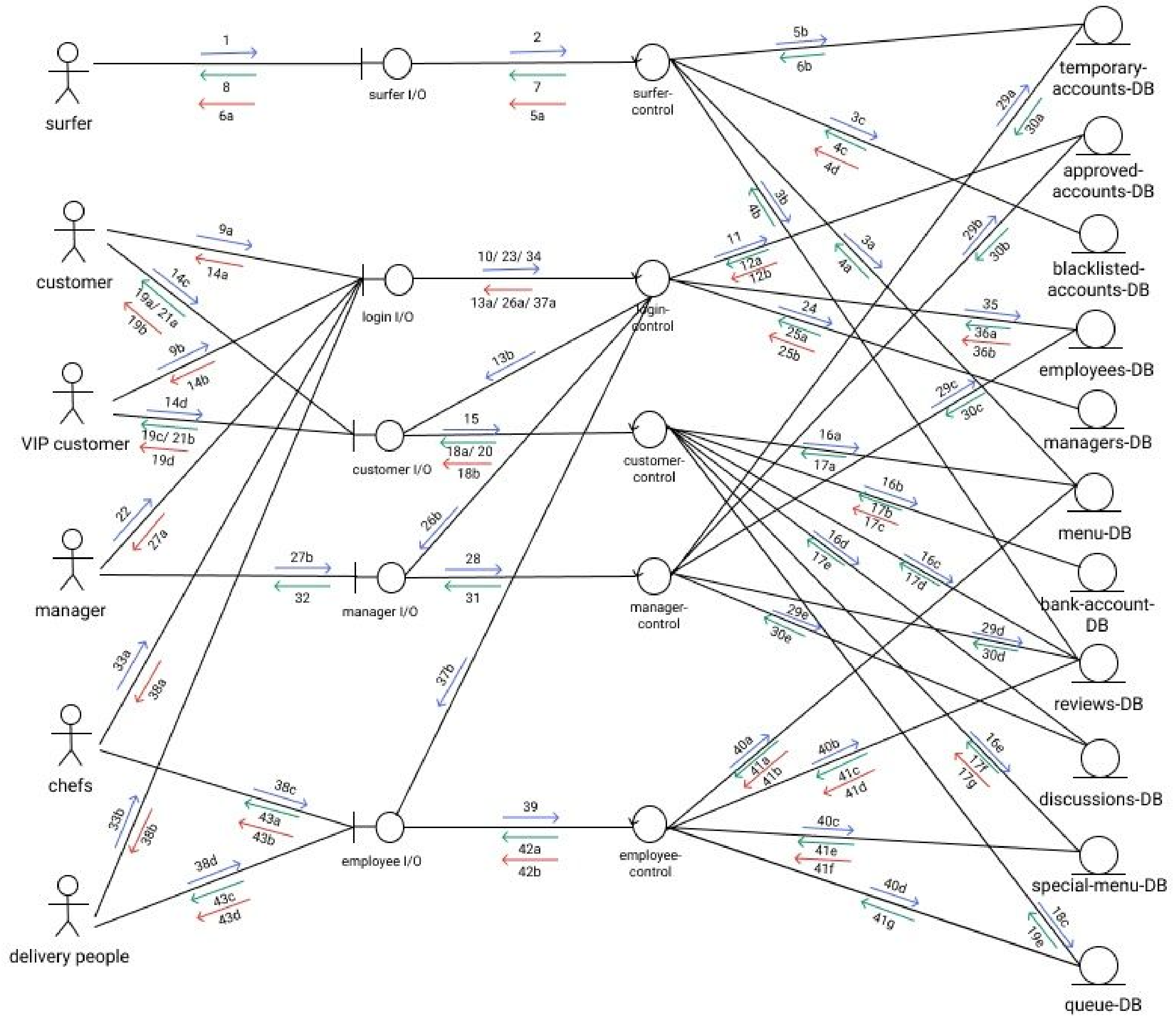
Revision History

Date	Version	Description	Author
10/20/2020	1.0	Version 1 of Software Requirements Specification	Zeal Patel, Bhavesh Shah. Yihui A Wuchen, Greg Kimatov, Xing Yang
11/17/2020	2.0	Version 2 of Software Requirements Specification	Zeal Patel, Bhavesh Shah. Yihui A Wuchen, Greg Kimatov, Xing Yang

Table of Contents

Foodtopia	Version: 2.0
Software Requirements Specification	Date: 11/17/2020
SRS	

1. Collaboration Class Diagram for Entire System



Numbered Actions:

Surfer:

1) Visitor visits website and can choose to browse menu/reviews or register for an account
2) Information of the visitor's choice is passed to the surfer control
3) a) Surfer control sends request to retrieve menu items from menu DB
   b) Surfer control sends request to retrieve reviews from reviews DB
   c) If a user wants to register for an account, surfer control sends a request to blacklisted accounts DB to check if a user is blacklisted.

4) a) Menu DB returns menu items in the response
   b) Reviews DB returns reviews in the response
   c) Blacklisted accounts DB returns success message in the response if surfer is NOT blacklisted
   d) Blacklisted accounts DB returns error message in the response if surfer is blacklisted
5) a) Surfer control sends an error message to the Surfer I/O to be displayed if surfer is blacklisted
   b) If surfer is not blacklisted, surfer control sends a request to create a new entry in temp accounts DB.
6) a) Return an error message back to surfer saying that they cannot sign up
   b) Temporary accounts DB returns success message in the response stating that new entry has been created successfully for surfer
7) Surfer control sends a success message to the Surfer I/O to be displayed since temporary account has been created for the surfer
8) Return a success message to surfer saying that their sign up form has gone through

<u>Customer/ VIP customer:</u>

9) a) Customer is forwarded to login I/O where they can enter their login credentials
   b) VIP customer is forwarded to login I/O where they can enter their login credentials
10) Customer's/ VIP customer's login credentials are passed to the login control
11) Login control sends request to approved accounts DB to check if the login credentials are correct
12) a) Approved accounts DB returns success message in the response if credentials are correct
   b) Approved accounts DB returns error message in the response if credentials are wrong
13) a) Login control sends an error message to the login I/O to be displayed if credentials are incorrect
   b) If credentials are correct, login control redirects to customer I/O
14) a) Return an error message back to customer saying to try again
   b) Return an error message back to VIP customer saying to try again
   c) Prompt customer to choose to browse/ order food, write review, participate in discussion
   d) Prompt VIP customer to choose to browse order food, write review, participate in discussion
15) Information of the customer's/ VIP customer's choice is passed to the customer control
16) a) Customer control sends request to retrieve menu items from menu DB
   b) If a customer/ VIP customer wants to order food, customer control sends a request to bank account DB to check if customer/ VIP customer has sufficient balance
   c) Customer control sends request to post/ get reviews from reviews DB

d) Customer control sends request to post/ get discussions from discussions DB
e) Customer control sends request to retrieve special menu items from special menu DB

17) a) Menu DB returns menu items in the response
b) Bank accounts DB returns success message in the response if there is sufficient balance
c) Bank accounts DB returns error message in the response if there is NOT sufficient balance
d) Reviews DB returns reviews in the response
e) Discussions DB returns discussions in the response
f) Special menu DB returns special menu items in the response if customer is VIP
g) Special menu DB returns error message in the response if customer is NOT VIP
18) a) Customer control sends a success message to the Customer I/O to be displayed if only review/ discussion related operations were performed
b) Customer control sends an error message to the Customer I/O to be displayed if customer was trying to order food but did not have sufficient balance or if customer tried to access special menu without having VIP status
c) If customer/ VIP customer has sufficient balance, customer control sends a request to create a new order in queue DB.
19) a) Return a success message back to customer regarding reviews/ discussion related content
b) Return an error message back to customer saying that they don't have enough money in their account to order food or that they aren't allowed to access special menu
c) Return a success message back to VIP customer regarding reviews/ discussion related content
d) Return an error message back to VIP customer saying that they don't have enough money in their account to order food or that they aren't allowed to access special menu
e) Queue DB returns success message in the response stating that the order has been placed
20) Customer control sends a success message to the Customer I/O to be displayed regarding the order being placed
21) a) Return a success message to customer saying that their order has been placed
b) Return a success message to VIP customer saying that their order has been placed

<u>Manager:</u>

22) Manager is forwarded to login I/O where they can enter their login credentials
23) Manager's login credentials are passed to the login control
24) Login control sends request to managers DB to check if the login credentials are correct
25) a) Managers DB returns success message in the response if credentials are correct
b) Managers DB returns error message in the response if credentials are wrong

26) a) Login control sends an error message to the login I/O to be displayed if credentials are incorrect
    b) If credentials are correct, login control redirects to manager I/O
27) a) Return an error message back to manager saying to try again
    b) Prompt manager to choose to approve surfers in the temporary accounts DB, blacklist users, hire/fire/promote/demote employees, or flag reviews/ discussions as inappropriate
28) Information of the manager's choice is passed to the manager control
29) a) Manager control sends request to approve the registered surfers in temporary accounts DB
    b) Manager control sends request to blacklist users in approved accounts DB
    c) Manager control sends request to hire/fire/promote/demote employees in employees DB
    d) Manager control sends request to flag reviews in reviews DB
    e) Manager control sends request to flag discussion in discussions DB
30) a) Temporary accounts DB returns success message of approved user(s) in the response
    b) Approved accounts DB returns success message of blacklisted user(s) in the response
    c) Employees DB returns success message of updated employee(s) in the response
    d) Reviews DB returns success message of flagged review(s) in the response
    e) Discussions DB returns success message of flagged discussion(s) in the response
31) Manager control sends a success message to the Manager I/O to be displayed regarding the operation performed
32) Return a success message to manager saying that their chosen operation has been executed

<u>Chef/ Delivery Person:</u>

33) a) Chef is forwarded to login I/O where they can enter their login credentials
    b) Delivery person is forwarded to login I/O where they can enter their login credentials
34) Chef's/ Delivery person's login credentials are passed to the login control
35) Login control sends request to employees DB to check if the login credentials are correct
36) a) Employees DB returns success message in the response if credentials are correct
    b) Employees DB returns error message in the response if credentials are wrong
37) a) Login control sends an error message to the login I/O to be displayed if credentials are incorrect
    b) If credentials are correct, login control redirects to employee I/O
38) a) Return an error message back to chef saying to try again
    b) Return an error message back to delivery person saying to try again

c) Prompt chef to choose to update menu, update special menu, or cook the next order in the queue
d) Prompt delivery person to choose to write a review for a customer or to deliver the next order in the queue

39) Information of the chef's/ delivery person's choice is passed to the employee control
40) a) Employee control sends request to add items to menu DB
b) Employee control sends request to post review(s) to reviews DB
c) Employee control sends request to add items to special menu DB
d) Employee control sends request to cook/ delivery next item in queue DB
41) a) Menu DB returns success message that a food dish was added in the response
b) Menu DB returns error message in the response if a delivery person tried to add a food dish instead of a chef
c) Reviews DB returns success message that a review was posted
d) Reviews DB returns error message in the response if a chef tried to post a review instead of a delivery person
e) Special menu DB returns success message that a food dish was added in the response
f) Special menu DB returns error message in the response if a delivery person tried to add a food dish instead of a chef
g) Queue DB returns success message in the response regarding an order being cooked/ delivered
42) a) Employee control sends a success message to the Employee I/O to be displayed if an employee operation was successful
b) Employee control sends an error message to the Employee I/O to be displayed if a delivery person tried to add a dish to the menu/ special menu or if a chef tried to post a review
43) a) Return a success message back to chef regarding regarding the operation they chose
b) Return an error message back to chef saying that they are not allowed to leave reviews
c) Return a success message back to delivery person regarding the operation they chose
d) Return an error message back to delivery person saying that they are not allowed to add dishes to the menu/ special menu

2. All Use-Cases

2.1 Scenarios for each use-case: normal and exceptional scenarios

Types of Users:

- Surfer - Customer - Employee - Manager

Use-Case 1: Surfers and ordinary customers browse menu items & ratings

Normal Scenarios:

- Surfers see the top 3 most popular and top 3 highest rated menu items on the home page.
- Ordinary customers see the top 3 listing menu items, recommended to them based on their previous orders.
- Both customers can search for items via the search bar and also toggle certain conditions (such as vegetarian, kosher, halal, etc.) via checkboxes underneath the search bar. They can click an item for more details.

Use-Case 2: Surfers register for an account

Normal Scenarios:

- Surfers sign up for an account and deposit some money into their account in order to become a "Customer" and start ordering items. If a surfer tries to order an item without having signed up, our application will redirect them to the "Sign Up" page.
- Surfers click on the "Sign Up" button and a new screen will prompt them to input the required information. Required information includes:
    - First name, last name, user type, email, and password

  A manager processes the customer registration and either approves or rejects it.

Use-Case 3: Users log in

Normal Scenarios:

- Attempt to log in is by a user who is not on the blacklist, and the login information is correct.
- Users log in to their account to gain access to functionalities only available to their group of users (e.g. customer/chef/delivery person/manager).

Exceptional Scenarios:

- Attempt to log in by a blacklisted user: login fails.

Use-Case 4: Customers order food and deposit money

Normal Scenarios:

- If the customer has a sufficient balance, they can order food. Customers can add money to their account at any time via the money portal in our system.
- Customers choose three ways to receive the orders: stay in the restaurant, pick up the orders from the restaurant, or have the order delivered to their house.
  - To stay in the restaurant, the customer needs to pick the available time and seat.
  - To have the order delivered to their house, they will need to pick a delivery person.

Exceptional Scenarios:

- If the cost of the order exceeds the funds in the user's account, the account will be frozen until a new deposit is made.

Use-Case 5: Customers chat in discussion forum

Normal Scenarios:

- Customers participate in a discussion forum by chatting with other users.. Discussions could be about the menu items, delivery people, and chefs.

Use-Case 6: Customers leave reviews for menu items/delivery people/chefs & delivery people leave reviews for customers

Normal Scenarios:

- Much like the previous use case, customers leave reviews for food items, delivery people, and chefs.
- Customers can also submit a review of a customer that may have posted something inappropriate in the discussion forum.
- If a customer has earned VIP status, then his/ her review will count as twice as much of that of a regular customer.
- Delivery people can leave a review on the customers they delivered the food to.

Exceptional Scenarios:

- Reviews, in the form of complaints, that hold no merit, as decided by the manager, will receive one warning.

<u>Use-Case 7:</u> Customers upgrade to VIP customers

Normal Scenarios:

- Customers apply for VIP status once they spend over $500 or place more than 50 orders.
- Similar to customers, once a VIP customer is logged in, their accounts possess all of the functionalities of a regular customer, such as the ability to order food, leave reviews, participate in the discussion forum, view their top 3 “For-you dishes”, add money to their account, and edit their account settings.
- When it comes to ordering dishes, VIP customers receive a 10% discount on their order.
- As for leaving reviews, VIP customers’ reviews count as twice as much as that of a regular customer.
- VIP customers can also get access to special, exclusive dishes that regular customers do not have access to.

<u>Use-Case 8:</u> Chefs invent new dishes and cook next order in queue

Normal Scenarios:

- Chefs have the freedom to invent new dishes, where they can upload an image of the dish, list all the ingredients and list dietary restrictions.
- As new orders are made by customers, they are added to the chef’s queue of meals to prepare. This can be viewed from the chef dashboard.

<u>Use-Case 9:</u> Delivery people deliver next order in queue

Normal Scenarios:

- Delivery people view the orders they have to deliver in the order they come in and then process them accordingly.

<u>Use-Case 10:</u> Manager handles reviews/discussion forum

Normal Scenarios:

- The manager handles the reviews from customers and delivery people and decides whether to approve or dismiss the review (which would send a warning to the person that initiated the review) depending on whether the review is genuine and does not contain inappropriate words. If a customer receives 3 warnings, their account would be closed, and their money would be returned.
- If a customer does not follow the terms of use, the manager can take away their VIP status or even deregister them.

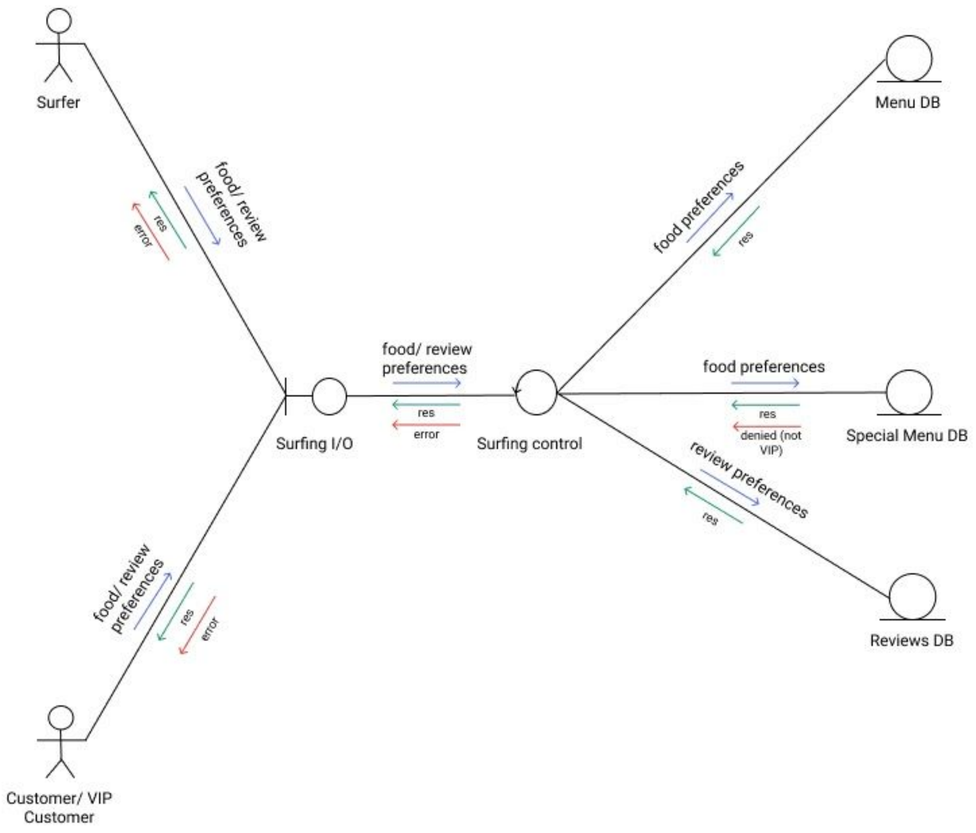
Use-Case 11: Manager hires/fires/cuts pay for an employee

Normal Scenarios:

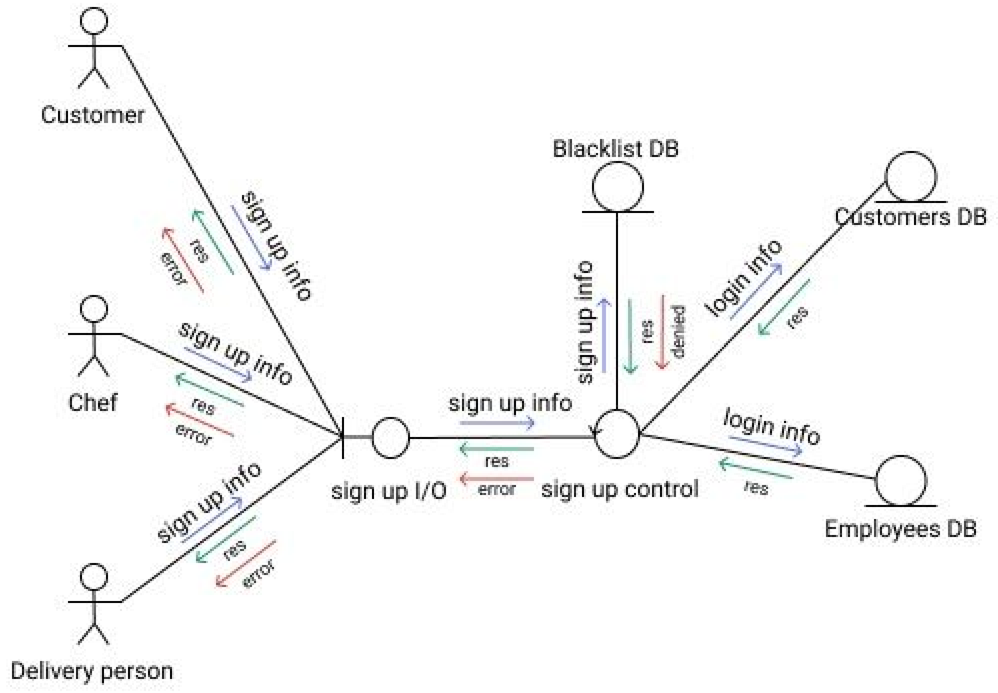
- The manager could choose the employees to hire after employees have gone through registration.
- They could review the profiles of employees (chefs and delivery people) and take action based on their performance.
    - If employees have poor performance, then managers can cut their salary.
        - If their salary is cut twice, managers can fire employees.
    - If employees are performing well, managers have the power to raise their salary.

2.2 Collaboration Class Diagrams or Petri nets for each use-case

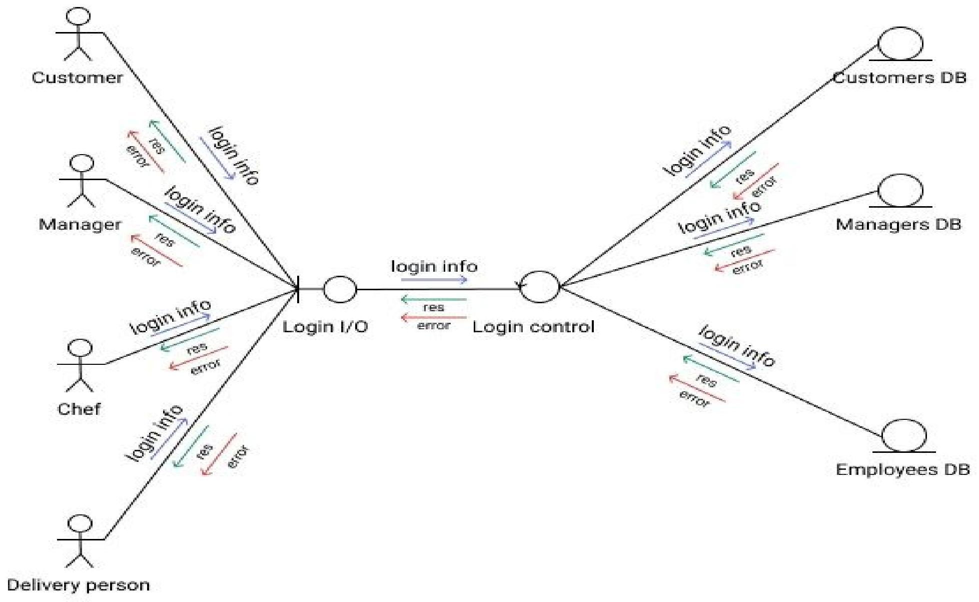
Use-Case 1:



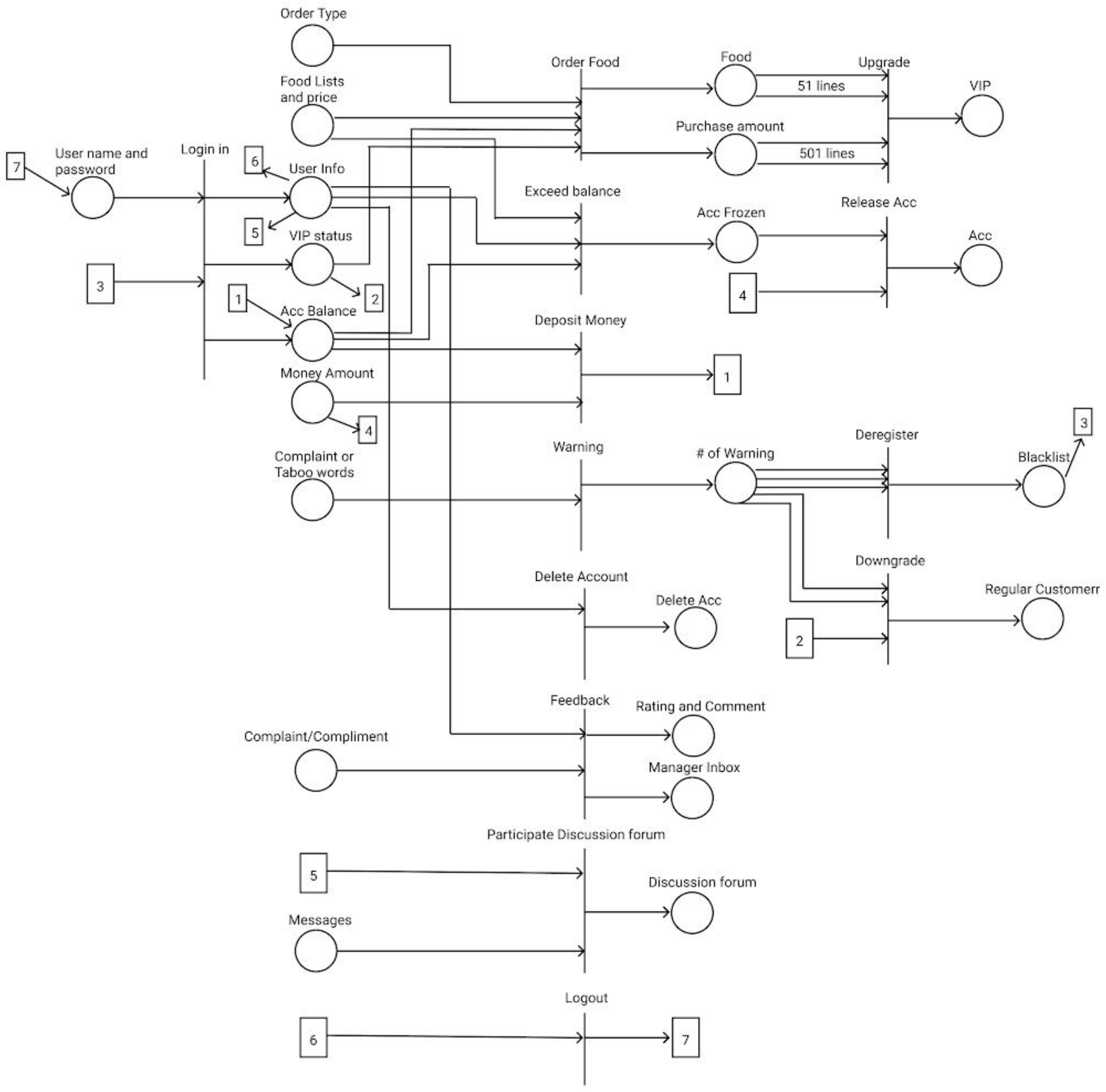
Use-Case 2:



Use-Case 3:

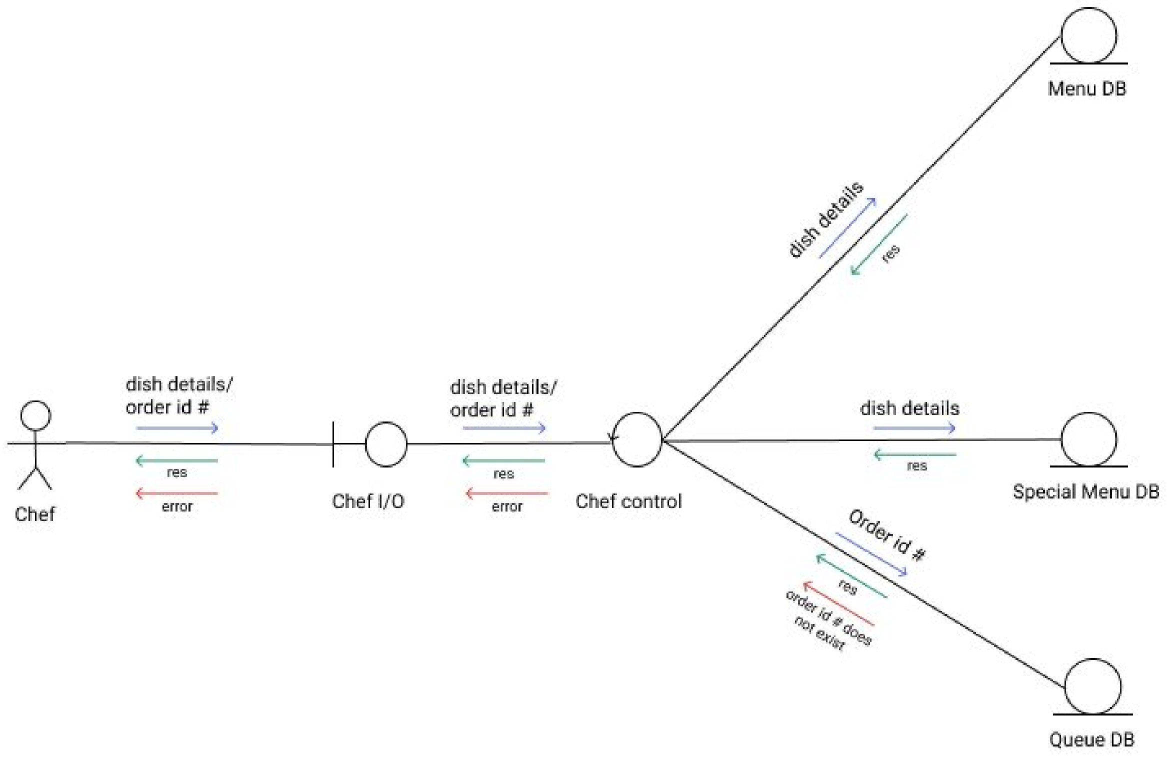


Use-Cases 4-7:

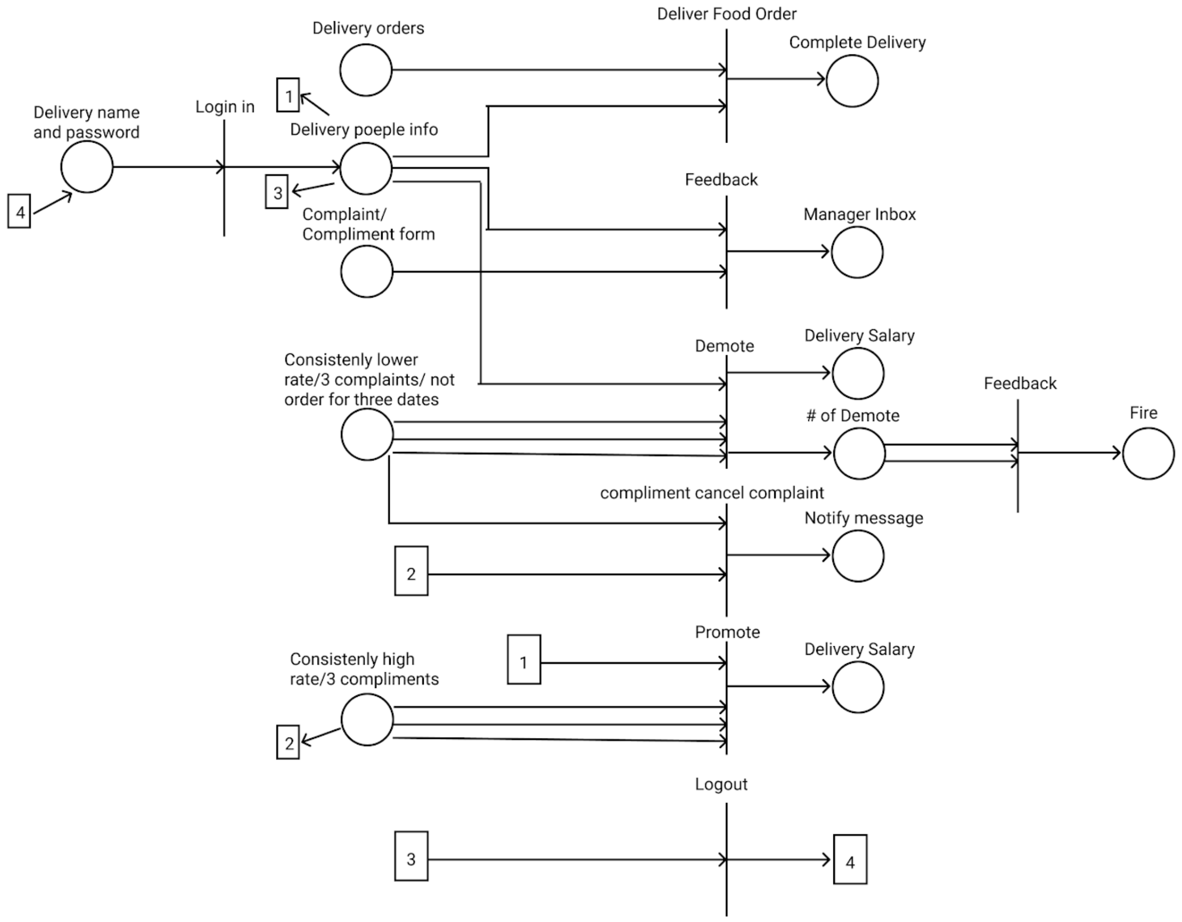


Order Type
Food Lists and price
User name and password
Login in
User Info
VIP status
Acc Balance
Money Amount
Complaint or Taboo words
Order Food
Exceed balance
Deposit Money
Warning
Delete Account
Feedback
Participate Discussion forum
Logout
Food
Purchase amount
51 lines
501 lines
Upgrade
VIP
Acc Frozen
Release Acc
Acc
# of Warning
Deregister
Blacklist
Downgrade
Regular Customerr
Delete Acc
Rating and Comment
Manager Inbox
Complaint/Compliment
Discussion forum
Messages
1
2
3
4
5
6
7

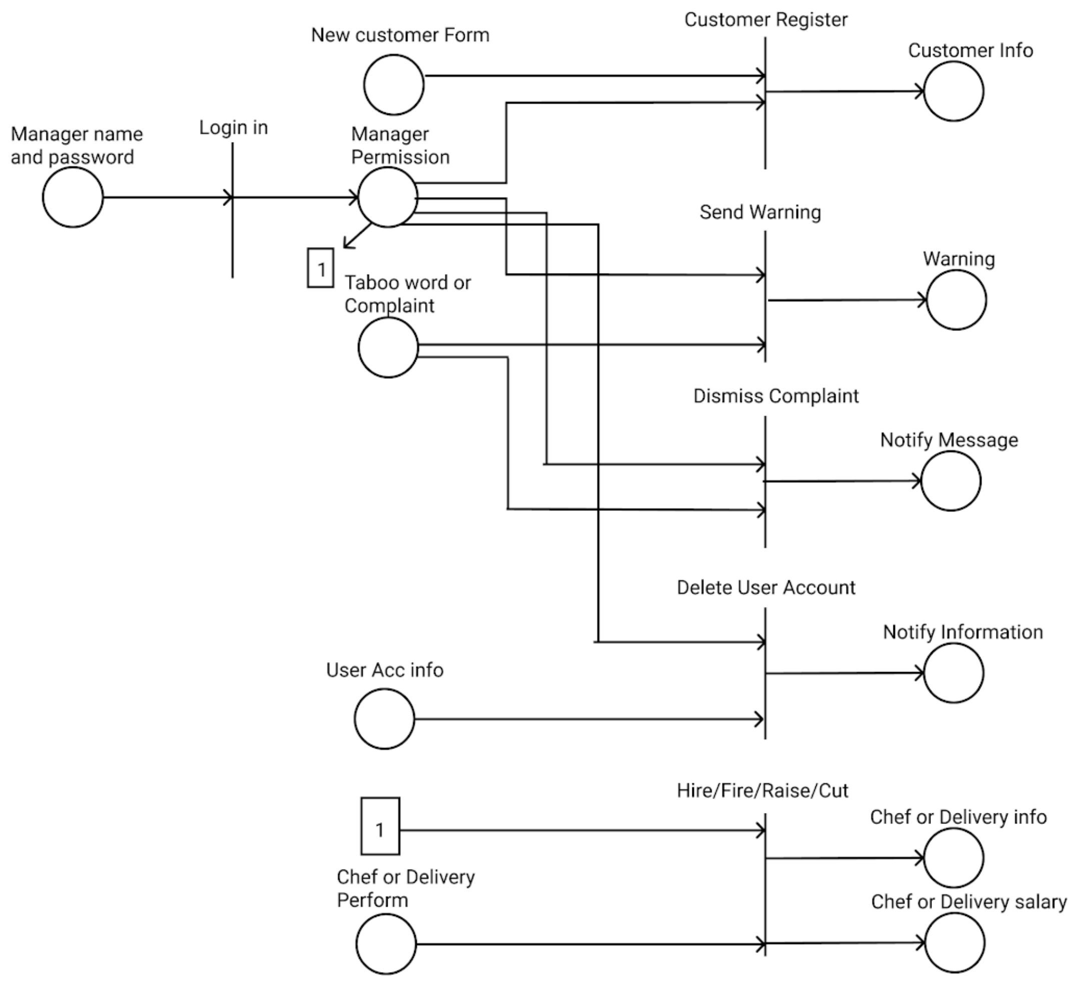
Use-Case 8:


Menu DB
dish details
res
dish details/ order id #
res
error
Chef
Chef I/O
dish details/ order id #
res
error
Chef control
dish details
res
Special Menu DB
Order id #
res
order id # does not exist
Queue DB

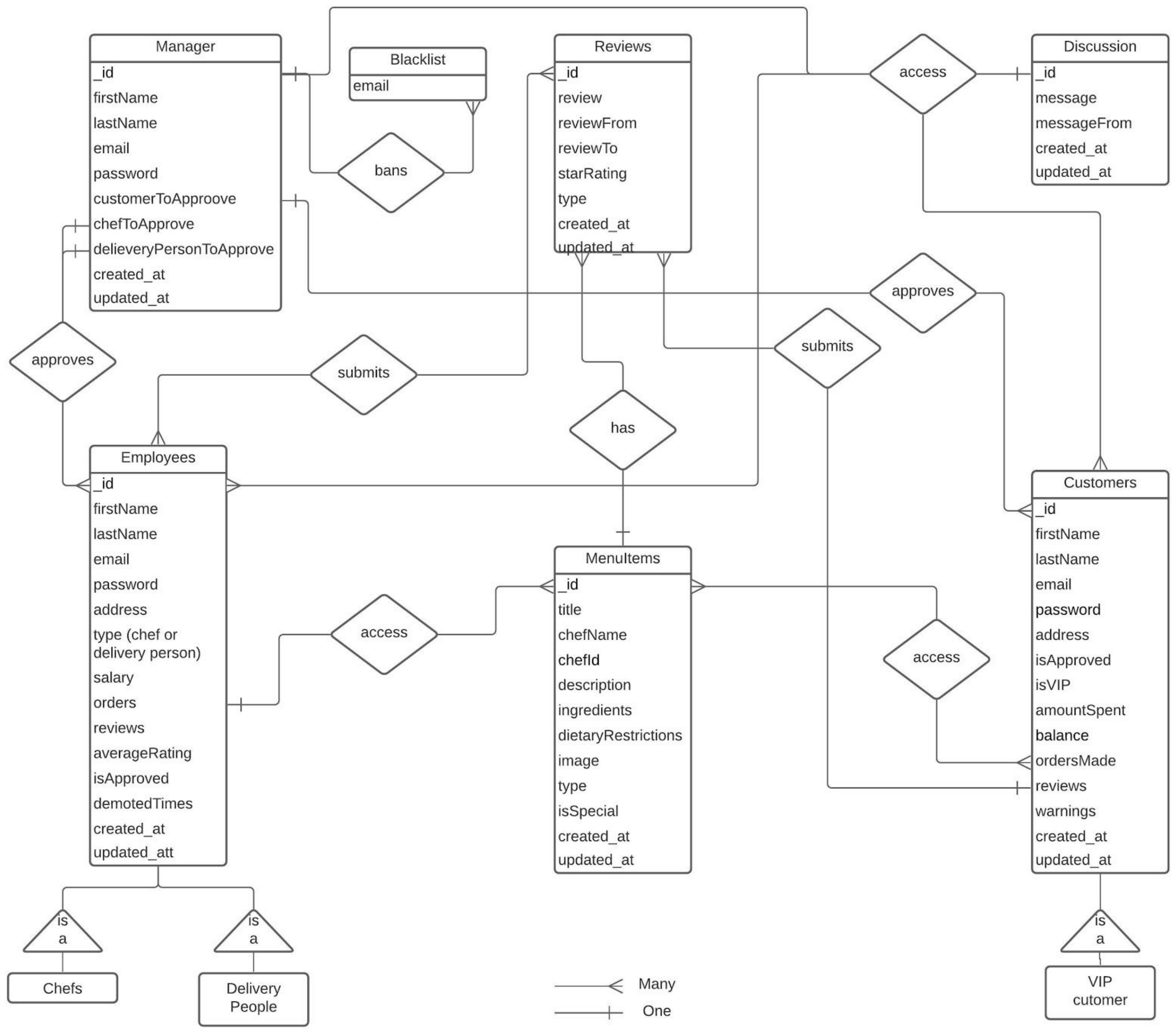
Use-Case 9:


Delivery orders
Deliver Food Order
Complete Delivery
1
Delivery name and password
Login in
Delivery poeple info
3
4
Complaint/ Compliment form
Feedback
Manager Inbox
Consistenly lower rate/3 complaints/ not order for three dates
Demote
Delivery Salary
Feedback
# of Demote
Fire
compliment cancel complaint
Notify message
2
Promote
1
Consistenly high rate/3 compliments
Delivery Salary
2
Logout
3
4

Use-Cases 10-11:



3. E/R diagram for entire system



4. Detailed design using pseudo-code

- **Use-Case 1:** Surfers and ordinary customers browse menu items & ratings
    - Get Menu Item
    - API endpoint: /api/menu-items/:menuItemId [GET]

```
import { Request, Response } from 'express';
import MenuItemsModel, { MenuItems } from '../../../models/MenuItems';

export const getMenuItem = async (req: Request, res: Response) => {
 const menuItems: MenuItems | null = await MenuItemsModel.findById(
   req.params.menuItemId
 );

 if (!menuItems) res.status(404).json({ msg: 'Menu items not found' });

 res.json(menuItems);
};
```

- **Use-Case 2:** Surfers register for an account
    - Sign Up User
    - API endpoint: /api/auth/signUp [POST]

```
import { Request, Response } from 'express';
import { createCustomer } from '../CustomerControllers/createCustomer';

export const signUpUser = async (req: Request, res: Response) => {
  try {
    switch (req.query.role) {
      case 'customer':
        await createCustomer(req, res);
        break;

      default:
        return res
          .status(400)
          .json({ msg: 'Sign up not allowed on this role' });
    }
  } catch (error) {
    console.error(error);
    res.status(500).json({ msg: error.message });
  }
};
```

- **Use-Case 3:** Users log in
  - Sign In User
  - API endpoint: /api/auth/signIn [POST]

```
import { Request, Response } from 'express';
import CustomersModel from '../../../models/Customers';
const bcryptjs = require('bcryptjs');
const jwt = require('jsonwebtoken');

export const signInUser = async (req: Request, res: Response) => {
  try {
    const userRoles: any = {
      customer: CustomersModel,
    };
    const userRole: any = req.query.role;

    const user: any = await userRoles[userRole].findOne({
      email: req.body.email,
    });

    if (!user) res.status(401).json({ msg: 'Invalid Credentials' });

    const isMatch: boolean = await bcryptjs.compare(
      req.body.password,
      user.password
    );

    if (!isMatch) res.status(401).json({ msg: 'Invalid Credentials' });

    const payload: any = {
      currentUser: {
        id: user.id,
        role: req.query.role,
      },
    };
    // "deleting" password from object because we dont want to expose it
    user.password = undefined;

    jwt.sign(
      payload,
      process.env.JWT_SECRET,
      { expiresIn: 3600000 }, // its in seconds, for now have an
arbitrarily big num
      (err: Error, token: string) => {
        if (err) throw err;
        res.json({ token, user });
      }
    );
  } catch (error) {
    console.error(error);
    res.status(500).json({ msg: 'Server Error' });
  }
};
```

- Is User Signed In

```
import { Request, Response, NextFunction } from 'express';
const jwt = require('jsonwebtoken');

export const isUserSignedInMiddleware = (
 req: Request,
 res: Response,
 next: NextFunction
) => {
 const token = req.header('x-auth-token');

 if (!token)
   return res.status(401).json({ msg: 'No token. Authorization denied.' });

 try {
   const decoded: any = jwt.verify(token, process.env.JWT_SECRET);
   req.currentUser = decoded.currentUser;
   next();
 } catch (error) {
   res.status(401).json({ msg: 'Token is not valid' });
 }
};
```

- Is User Allowed

```
import { Request, Response, NextFunction } from 'express';

export const isUserAllowedMiddleware = (roles: string[]) => {
 return (req: Request, res: Response, next: NextFunction) => {
   if (!roles.includes(req.currentUser.role)) {
     return res.status(401).json({ msg: 'Not authorized' });
   }

   next();
 };
};
```

- **Use-Case 4:** Customer orders food and deposits money
    - API endpoint: /api/customers/orders [PUT]

```
def orderMenuItem(menuItemId, chefId, customerId):
 CustomersModel.update({_id: customerId},{orders: orders.push(menuItemId)})

 chef = EmployeesModel.find(chefId)

 if (!chef):
   throw Error('No chef found')

 // notify chef that have a new order to cook
 EmployeesModel.update({_id: chefId}, {orders: orders.push(menuItemId)})
```

- API endpoint: /api/customers/:customerId/depositMoney [PATCH]

```
def depositMoney(customerId, amount):
  customer = CustomersModel.find({_id: customerId})

  if (!customer):
    throw Error('customer not found')
  newBalance = amount + customer.newBalance

  CustomersModel.update({_id: customerId}, {balance: newBalance})
```

- **Use-Case 5:** Customers chat in discussion forum
    - API endpoint: /api/customers/discussions [POST]

```
def postDiscussion(message, messageFromId):
  discussion = new DiscussionsModel({
    message: message,
    messageFrom: messageFromId
  })

  disccusion.save()
```

- API endpoint: /api/customers/discussions [GET]

```
def getDiscussions():
  discussions = DiscussionsModel.find({})

  if (!discussions):
    return 'No discussions posted yet'

  return discussions
```

- **Use-Case 6:** Customers leave reviews for menu items/delivery people/chefs & delivery people leave reviews for customers
    - API endpoint: /api/reviews [POST]

```
def createReview(reviewToId, reviewFromId, reviewMessage, type,
starRating):
  review = new ReviewsModel({
    review: review,
    reviewFrom: reviewFromId,
    reviewTo: reviewToId,
    type: type,
    starRating: starRating
  })

  review.save()
```

    - API endpoint: /api/reviews?reviewTo=””&reviewFrom=”” [GET]

```
def getReviews(reviewToId, reviewFromId):
  reviews = ReviewsModel.find({
    reviewTo: reviewToId,
    reviewFrom: reviewFromId
  })

  if (!reviews || reviews.length === 0):
    return 'No reviews found'

  return reviews
```

- **Use-Case 7:** Customers upgrades to VIP customer [PATCH]
    - API endpoint: /api/customers/:customerId/upgrade

```
def upgradeCustomerToVIP(customerId):
  customer = CustomersModel.find({_id: customerId})

  if (!customer):
    throw Error('customer not found')

  CustomersModel.update({_id: customerId}, {isVIP: true})
```

- Some essential routes that are needed for customer functionality
- Create Customer
    - API endpoint: /api/customers [POST]

```
import { Request, Response } from 'express';
import CustomersModel, { Customers } from '../../../models/Customers';
const bcryptjs = require('bcryptjs');
export const createCustomer = async (req: Request, res: Response) => {
 try {
   let customer: Customers | null = await CustomersModel.findOne({
     email: req.body.email,
   });
```

```
    if (customer) res.status(400).json({ msg: 'User already exists' });

    customer = new CustomersModel({
      password: req.body.password,
      firstName: req.body.firstName,
      lastName: req.body.lastName,
      email: req.body.email,
      address: req.body.address,
    });

    const salt = await bcryptjs.genSalt(10);

    customer.password = await bcryptjs.hash(customer.password, salt);

    await customer.save();

    res.json({ msg: 'Account waiting for approval' });
  } catch (error) {
    console.error(error.message);
    res.status(500).json({ msg: 'Server error' });
  }
};
```

- Get Customers
  - API endpoint: /api/customers [GET]

```
import { Request, Response } from 'express';
import CustomersModel, { Customers } from '../../../models/Customers';
export const getCustomer = async (req: Request, res: Response) => {
 const customer: Customers | null = await CustomersModel.findById(
   req.params.customerId
 ).select('-password');
 if (!customer) res.status(404).json({ msg: 'Customer not found' });
 res.json(customer);
};
```

- Get Customer
  - API endpoint: /api/customers/:customerId [GET]

```
import { Request, Response } from 'express';
import CustomersModel, { Customers } from '../../../models/Customers';

export const getCustomers = async (req: Request, res: Response) => {
 const customers: Customers[] | null = await CustomersModel.find({}).select(
   '-password'
 );

 if (!customers) res.status(404).json({ msg: 'Customer not found' });

 res.json(customers);
};
```

- **<u>Use-Case 8:</u>** Chefs invent new dishes and cook next order in queue
  - API endpoint: /api/menu-items
  - (We are working on uploading image of a menu item to the database)

```
import { Request, Response } from 'express';
import MenuItemsModel, { MenuItems } from '../../../models/MenuItems';

export const createMenuItem = async (req: Request, res: Response) => {
 try {
   const menuItem = new MenuItemsModel({
     title: req.body.title,
     chefName: req.body.chefName,
     description: req.body.description,
     ingredients: req.body.ingredients,
     dietaryRestrictions: req.body.dietaryRestrictions,
     type: req.body.type,
     image: req.file.filename,
   });

   await menuItem.save();

   res.json(menuItem);
 } catch (error) {
   console.error(error);
   res.status(500).json({ msg: 'Server Error' });
 }
};
```

- API endpoint: /api/employees/chefs/cook/:menuItemId [PATCH]

```
def cookOrder(employeeId, menuItemId):
  employee = EmployeesModel.find({_id: employeeId})

  if (!employee):
    throw Error('employee not found')

  employee.orders.pop(menuItemId)

  EmployeesModel.update({_id: employeeId}, {orders: employee.orders})
```

- **<u>Use-Case 9:</u>** Delivery people deliver next order in queue
    - API endpoint: /api/employees/delivery-people/deliver/:menuItemId [PATCH]

```
def cookOrder(employeeId, menuItemId):
  employee = EmployeesModel.find({_id: employeeId})

  if (!employee):
    throw Error('employee not found')

  employee.orders.pop(menuItemId)

  EmployeesModel.update({_id: employeeId}, {orders: employee.orders})
```

- Some essential routes that are needed for employee functionality
- Create Employee
    - API endpoint: /api/employees [POST]

```
import { Request, Response } from 'express';
import EmployeesModel, { Employees } from '../../../models/Employees';
const bcryptjs = require('bcryptjs');

export const createEmployee = async (req: Request, res: Response) => {
 try {
   let employee: Employees | null = await EmployeesModel.findOne({
     email: req.body.email,
   });
```

```
    if (employee) res.status(400).json({ msg: 'User already exists' });

    employee = new EmployeesModel({
      firstName: req.body.firstName,
      lastName: req.body.lastName,
      email: req.body.email,
      password: req.body.password,
      type: req.body.type,
    });

    const salt = await bcryptjs.genSalt(10);

    employee.password = await bcryptjs.hash(employee.password, salt);

    await employee.save();

    res.json({ msg: 'Account waiting for approval' });
  } catch (error) {
    console.error(error);
    res.status(500).json({ msg: 'Server error' });
  }
};
```

- Get Employees
  - API endpoint: /api/employees [GET]

```
import { Request, Response } from 'express';
import EmployeesModel, { Employees } from '../../../models/Employees';

export const getEmployees = async (req: Request, res: Response) => {
  const employees: Employees[] | null = await EmployeesModel.find({}).select(
    '-password'
  );

  if (!employees) res.status(404).json({ msg: 'Employees not found' });

  res.json(employees);
};
```

- Get Employee

- API endpoint: /api/employees/:employeeId

```
import { Request, Response } from 'express';
import EmployeesModel, { Employees } from '../../../models/Employees';

export const getEmployee = async (req: Request, res: Response) => {
 const employee: Employees | null = await EmployeesModel.findOne({
   _id: req.params.employeeId,
 }).select('-password');

 if (!employee) res.status(404).json({ msg: 'Employees not found' });

 res.json(employee);
};
```

- Get Menu Items
  - API endpoint: /api/menu-items [GET]

```
import { Request, Response } from 'express';
import MenuItemsModel, { MenuItems } from '../../../models/MenuItems';

export const getMenuItems = async (req: Request, res: Response) => {
 const menuItems: MenuItems[] | null = await MenuItemsModel.find({});

 if (!menuItems) res.status(404).json({ msg: 'Menu items not found' });

 res.json(menuItems);
};
```

- **Use-Case 10:** Manager handles reviews/discussion forum
  - API endpoint: /api/managers/handleReview/:reviewId [DELETE]

```
def handleReview(reviewId):
  review =  ReviewsModel.find({_id: reviewId})

  if (!review):
    throw Error('review not found')

  ReviewsModel.delete({_id: reviewId})
  return 'Review deleted because of inappropriate behavior'
```

- API endpoint: /api/managers/handleDiscussion/:discussionId [DELETE]

```
def handleDiscussion(discussionId):
  discussion =  DiscussionsModel.find({_id: discussionId})

  if (!discussion):
    throw Error(discussion not found')

  DiscussionsModel.delete({_id: discussionId})

  return 'Discussion deleted because of inappropriate behavior'
```

- **<u>Use-Case 11:</u>** Manager hires/fires/cuts pay for an employee
  - API endpoint: /api/managers/cut-pay/:employeeId [PATCH]

```
def cutPay(employeeId, newPay):
 employee = EmployeesModel.find(employee)

 if (!employee):
   throw Error('No employee found')

 EmployeesModel.update({salary: newPay})
```

  - API endpoint: /api/managers/fire/:employeeId [DELETE]

```
def fireEmployee(employeeId):
 employee = EmployeesModel.find(employee);

 if (!employee):
   throw Error('No employee found');

 EmployeesModel.delete({_id: employeeId})
```

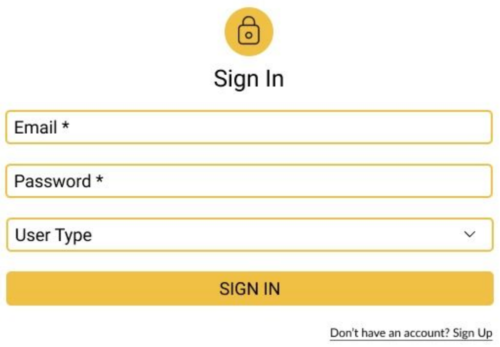
- Some essential routes that are needed for manager functionality
- Approve User
  - API endpoint: /api/managers/approve/:customerId [POST]

```
import { Request, Response } from 'express';
import axios, {AxiosResponse} from 'axios';

const Customer = require("../../../models/Customers");

const sgMail = require('@sendgrid/mail')
sgMail.setApiKey(process.env.SENDGRID_API_KEY)

export const approveCustomer = (req: Request, res: Response) => {
 Customer.findOneAndUpdate({ _id: req.params.customerId }, { isApproved: true
})
   .then(() => {
     var link = 'http://localhost:5000/api/customers/' + req.params.customerId;

     axios.get(link) // retrieve email for customer
       .then((response : AxiosResponse) => {
         var email = response.data.email;
         console.log(email);

         const msg = { // send an email confirmation to customer
           to: email,
           from: 'bhavesh.shah@macaulay.cuny.edu',
           templateId: 'd-296b3c7a87534eedb896f5fc49bdb1d6',
         }
         sgMail
           .send(msg)
           .then(() => {
             console.log('Email sent')
           })
           .catch((error : Error) => {
             console.error(error)
           })
       })
       .catch((error : Error) => {
         console.log(error);
       });

     res.status(202).json("Customer approved");
   })
```

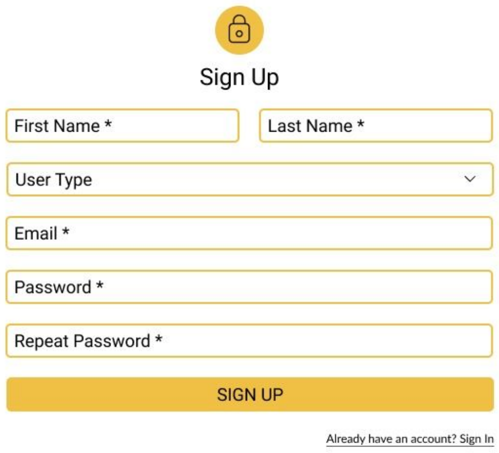
```
    .catch((err : Error) => res.status(500).json(err));
};
```

5. System screens

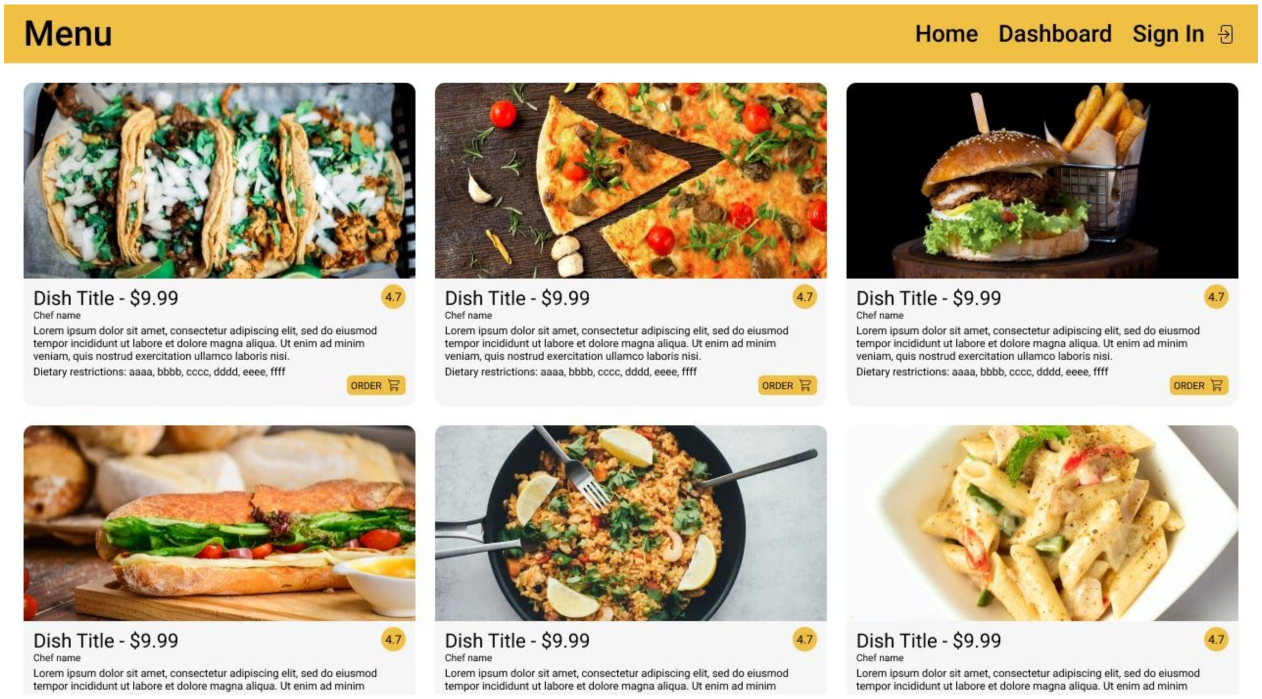
This is how our system would look with all of the required and several additional features fully implemented.



Sign In Page



Sign Up Page



Dish Title - $9.99
4.7
Chef name
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi.
Dietary restrictions: aaaa, bbbb, cccc, dddd, eeee, ffff
ORDER

Dish Title - $9.99
4.7
Chef name
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi.
Dietary restrictions: aaaa, bbbb, cccc, dddd, eeee, ffff
ORDER

Dish Title - $9.99
4.7
Chef name
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi.
Dietary restrictions: aaaa, bbbb, cccc, dddd, eeee, ffff
ORDER

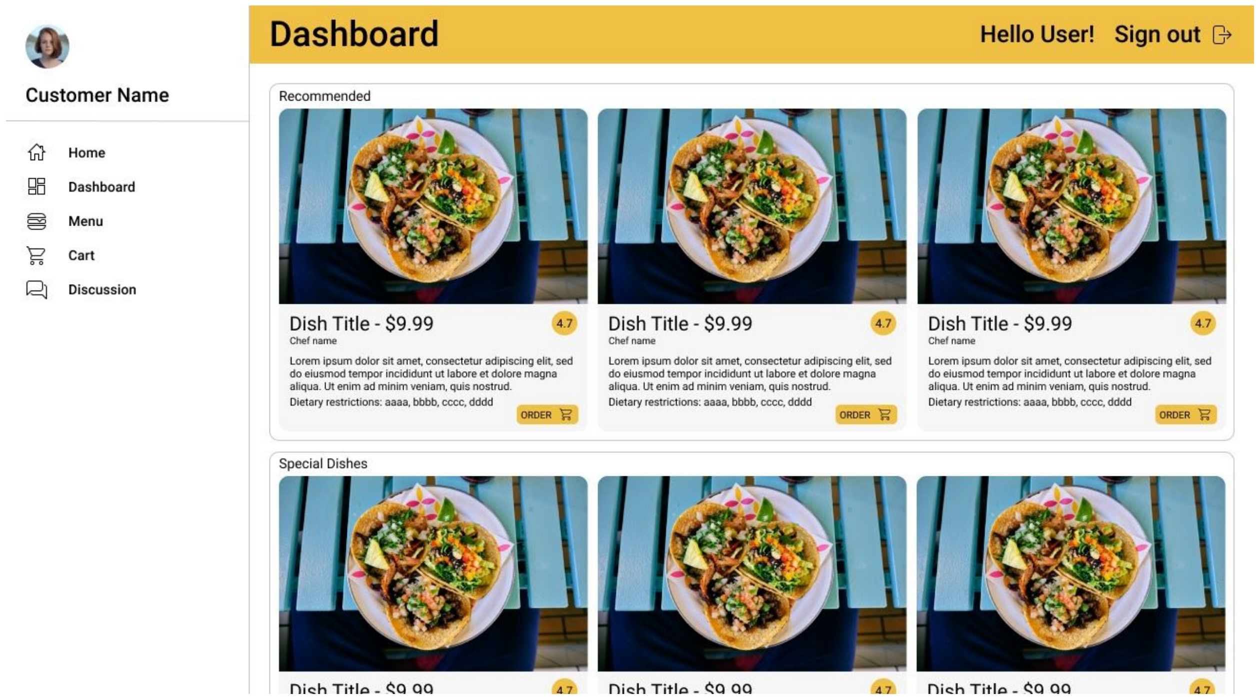
Dish Title - $9.99
4.7
Chef name
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim

Dish Title - $9.99
4.7
Chef name
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim

Dish Title - $9.99
4.7
Chef name
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim

Menu Page

Foodtopia	Version: 2.0
Software Requirements Specification	Date: 11/17/2020
SRS	



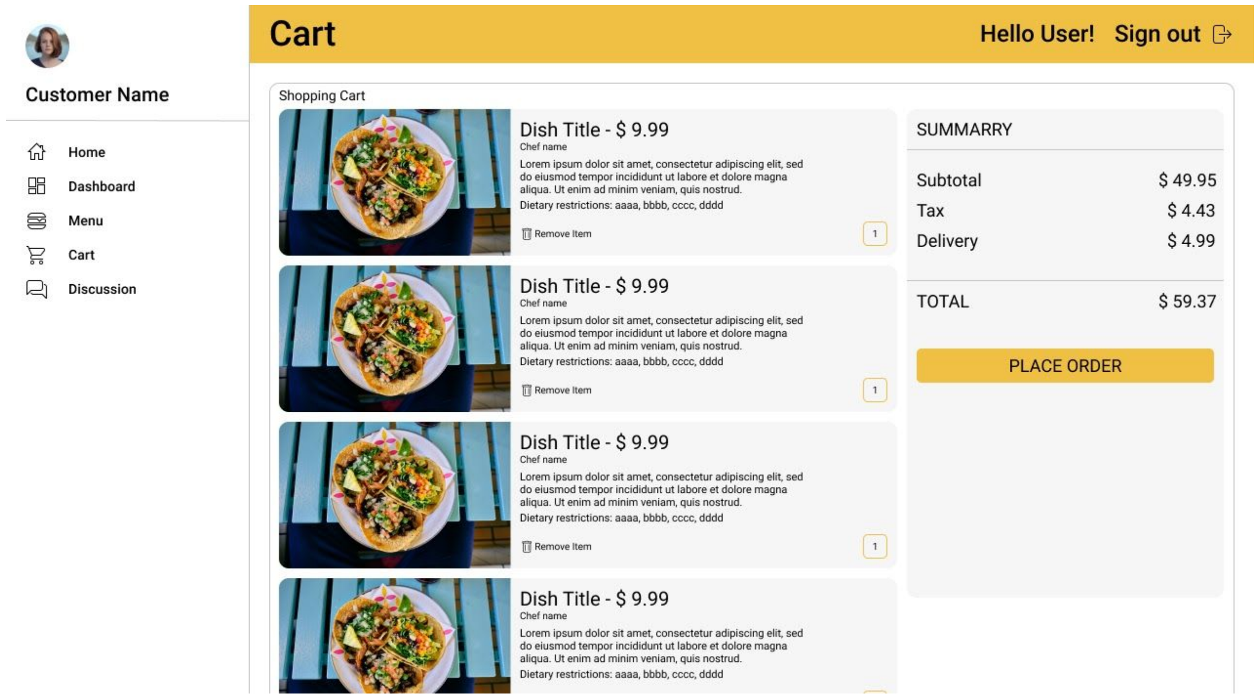
Dish Title - $9.99

Chef name

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud.

Dietary restrictions: aaaa, bbbb, cccc, dddd

Customer Dashboard



Dish Title - $ 9.99

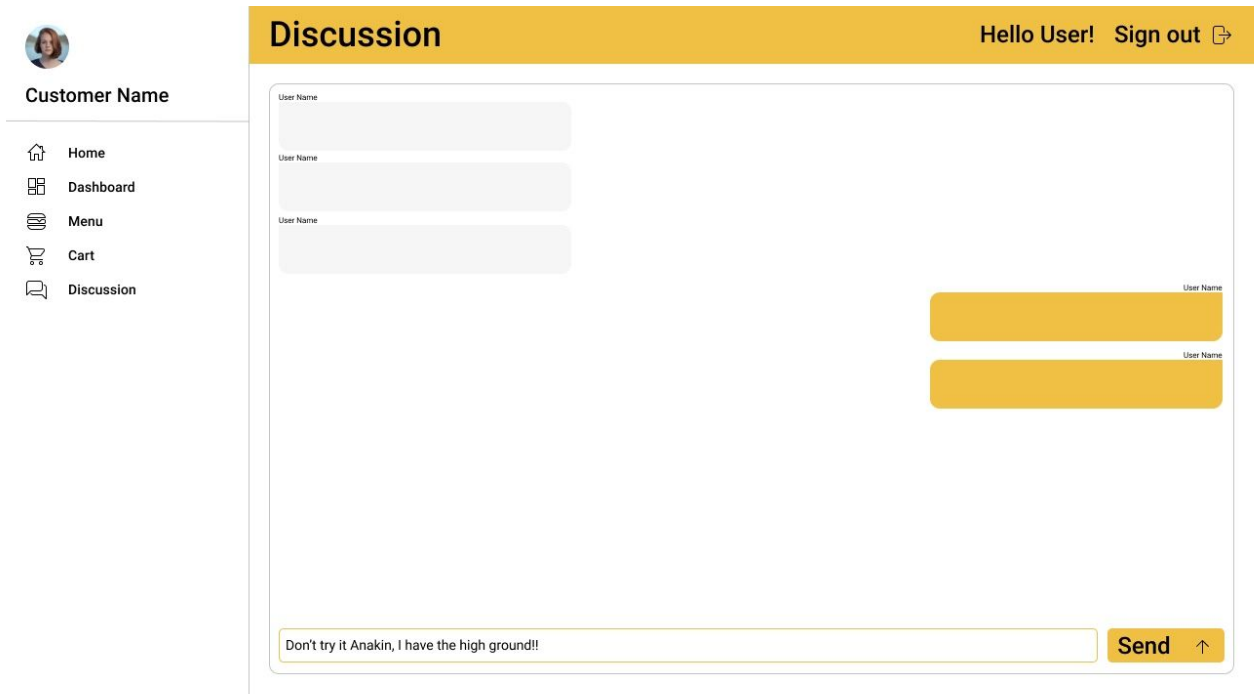
Chef name

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud.

Dietary restrictions: aaaa, bbbb, cccc, dddd

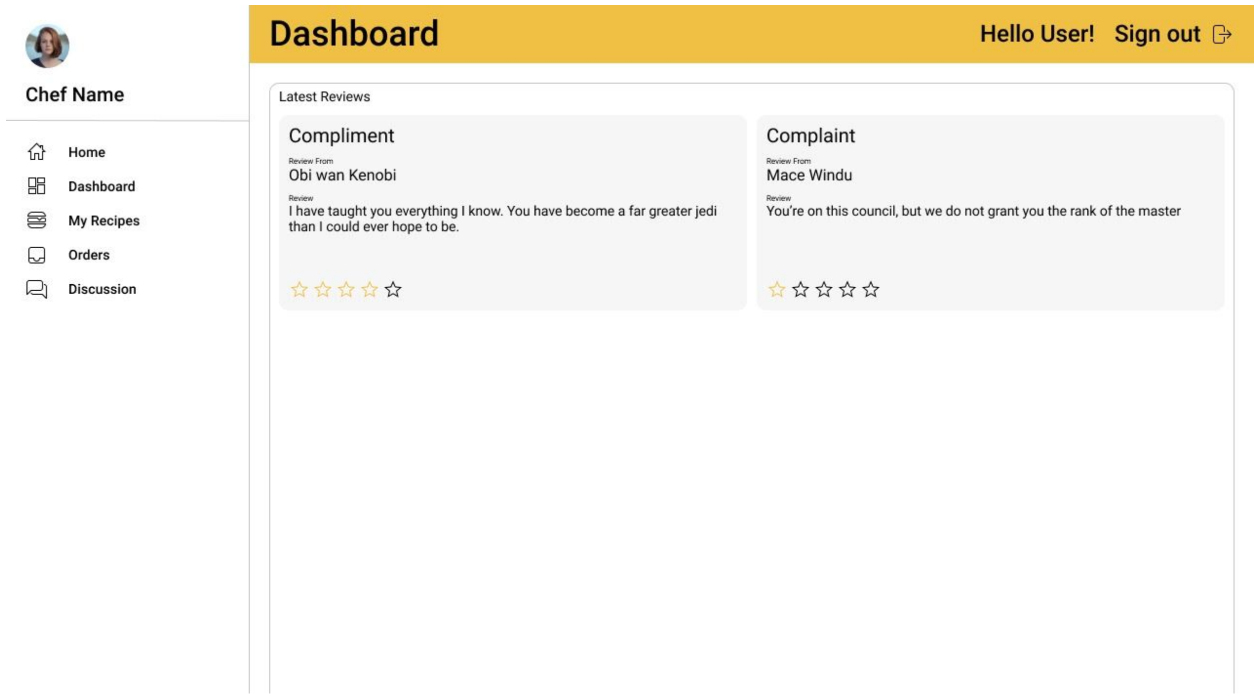
SUMMARRY	
Subtotal	$ 49.95
Tax	$ 4.43
Delivery	$ 4.99
TOTAL	$ 59.37

Customer Cart Page



Don't try it Anakin, I have the high ground!!

Discussion Page



Compliment

Review From
Obi wan Kenobi

Review
I have taught you everything I know. You have become a far greater jedi than I could ever hope to be.

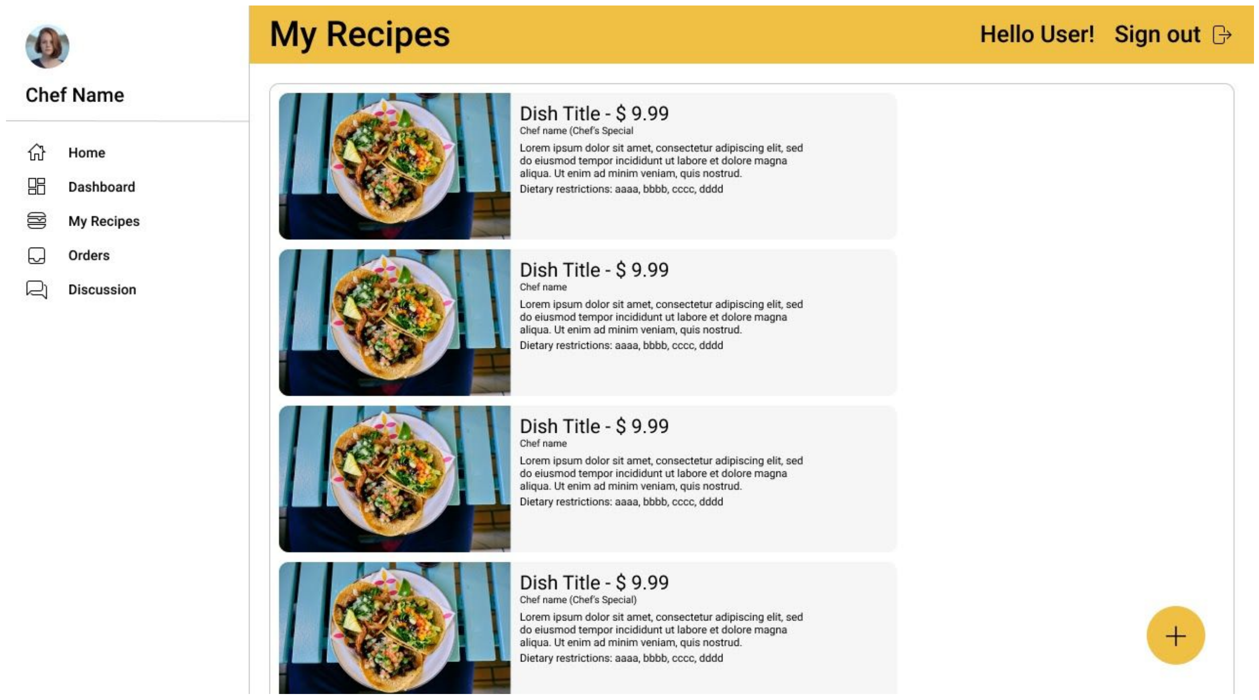
Complaint

Review From
Mace Windu

Review
You're on this council, but we do not grant you the rank of the master

Chef Dashboard

Foodtopia	Version: 2.0
Software Requirements Specification	Date: 11/17/2020
SRS	



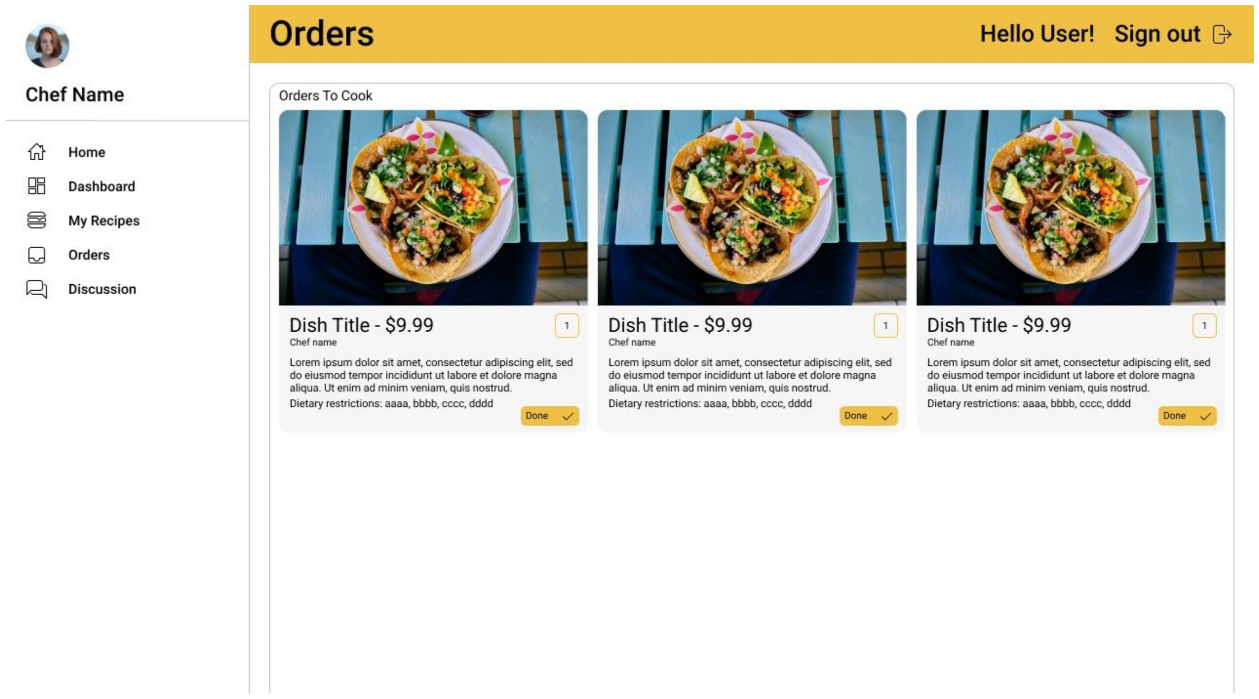
Dish Title - $ 9.99
Chef name (Chef's Special
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud.
Dietary restrictions: aaaa, bbbb, cccc, dddd

Dish Title - $ 9.99
Chef name
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud.
Dietary restrictions: aaaa, bbbb, cccc, dddd

Dish Title - $ 9.99
Chef name
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud.
Dietary restrictions: aaaa, bbbb, cccc, dddd

Dish Title - $ 9.99
Chef name (Chef's Special)
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud.
Dietary restrictions: aaaa, bbbb, cccc, dddd

Chef "My Recipes" Page

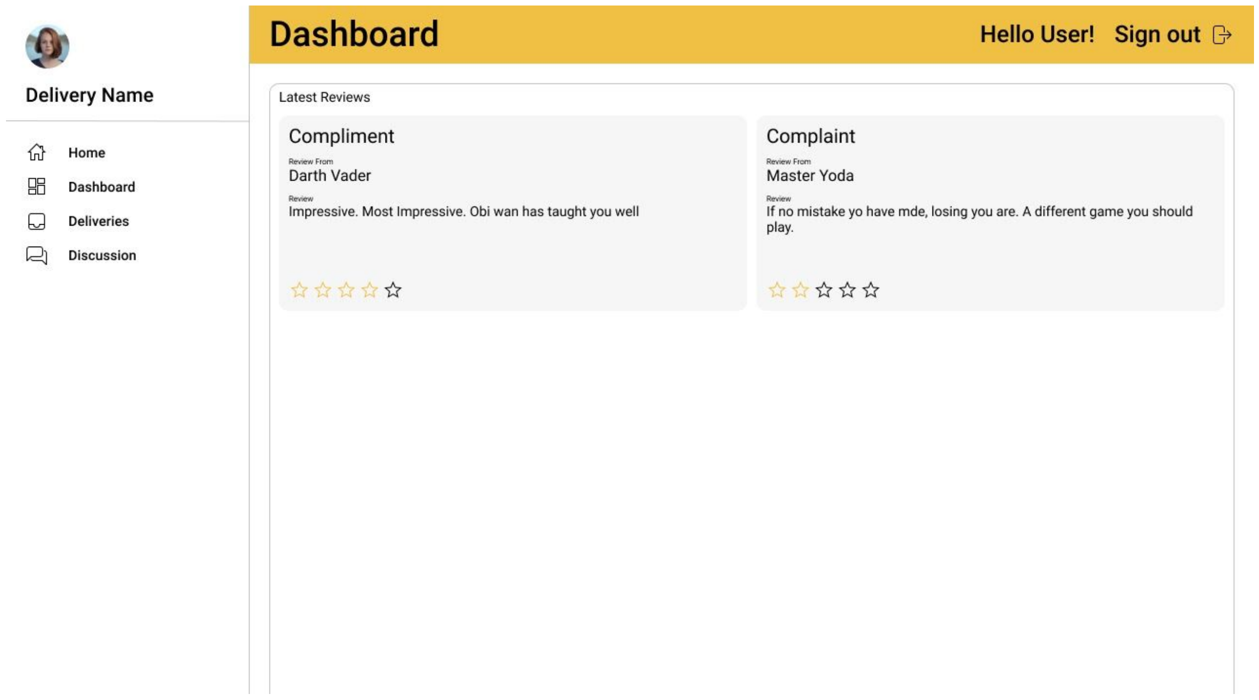


Dish Title - $9.99 1
Chef name
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud.
Dietary restrictions: aaaa, bbbb, cccc, dddd
Done

Dish Title - $9.99 1
Chef name
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud.
Dietary restrictions: aaaa, bbbb, cccc, dddd
Done

Dish Title - $9.99 1
Chef name
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud.
Dietary restrictions: aaaa, bbbb, cccc, dddd
Done

Chef "Orders To Make" Page



Dashboard

Hello User! Sign out

Latest Reviews

Compliment

Review From

Darth Vader

Review

Impressive. Most Impressive. Obi wan has taught you well

★★★★☆

Complaint

Review From

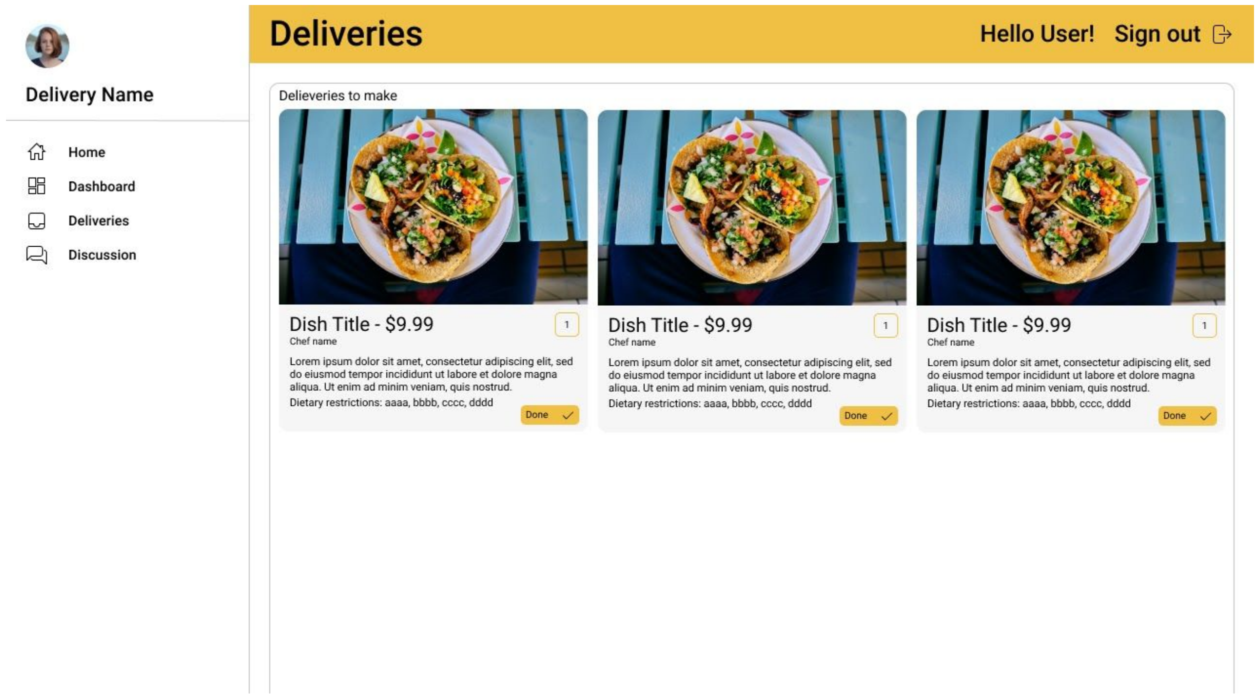
Master Yoda

Review

If no mistake yo have mde, losing you are. A different game you should play.

★★☆☆☆

Delivery Person Dashboard



Deliveries

Hello User! Sign out

Delieveries to make

Dish Title - $9.99 | 1

Chef name

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud.

Dietary restrictions: aaaa, bbbb, cccc, dddd

Done ✓

Dish Title - $9.99 | 1

Chef name

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud.

Dietary restrictions: aaaa, bbbb, cccc, dddd

Done ✓

Dish Title - $9.99 | 1

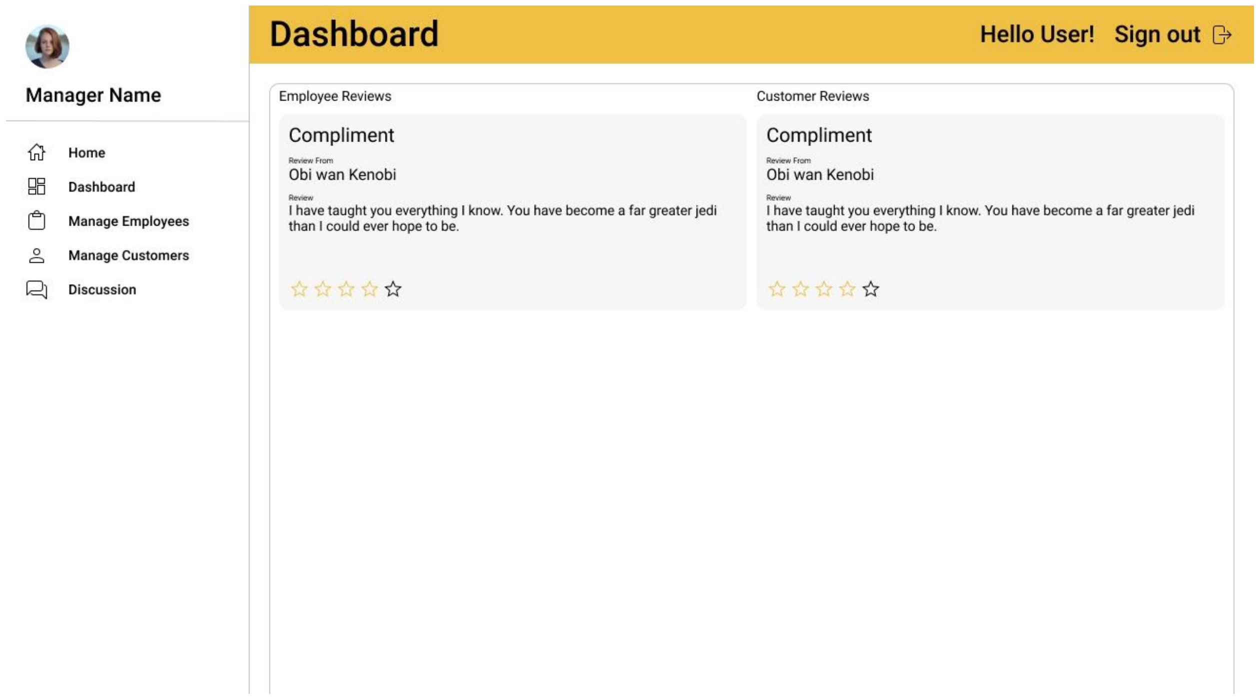
Chef name

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud.

Dietary restrictions: aaaa, bbbb, cccc, dddd

Done ✓

Delivery Person “Deliveries To Make” Page


Dashboard

Hello User! Sign out

Manager Name

- Home
- Dashboard
- Manage Employees
- Manage Customers
- Discussion

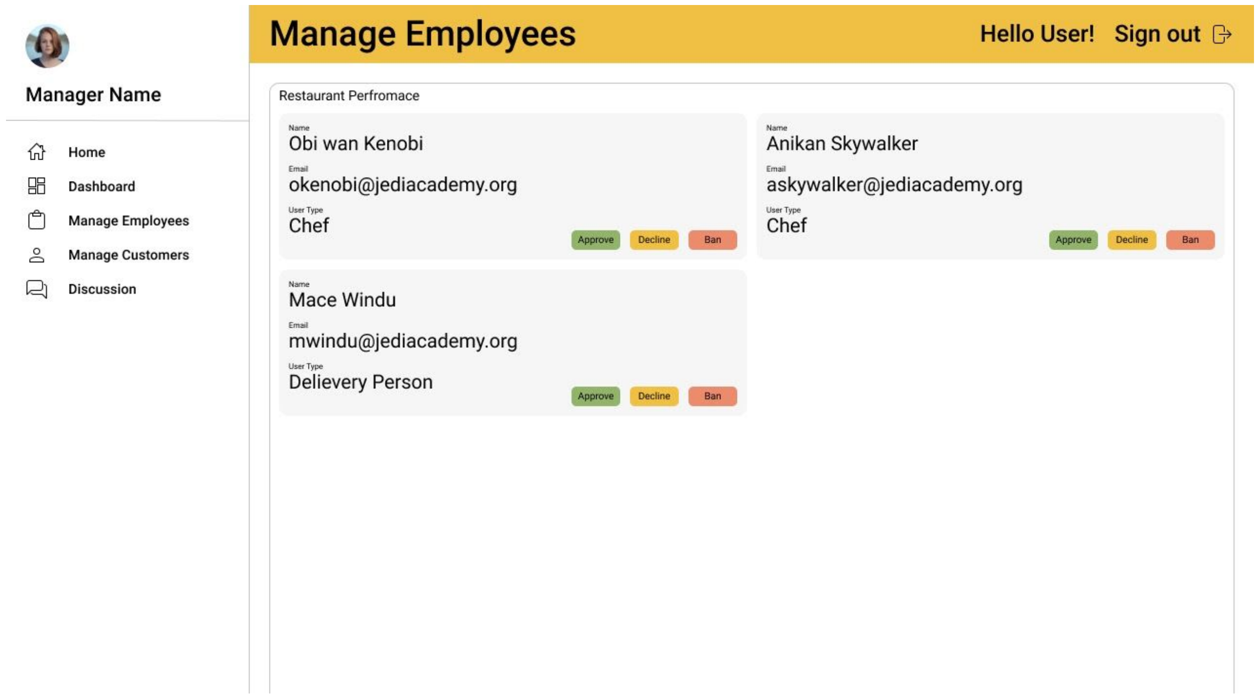
Employee Reviews

Compliment
Review From
Obi wan Kenobi
Review
I have taught you everything I know. You have become a far greater jedi than I could ever hope to be.

Customer Reviews

Compliment
Review From
Obi wan Kenobi
Review
I have taught you everything I know. You have become a far greater jedi than I could ever hope to be.

Manager Dashboard


Manage Employees

Hello User! Sign out

Manager Name

- Home
- Dashboard
- Manage Employees
- Manage Customers
- Discussion

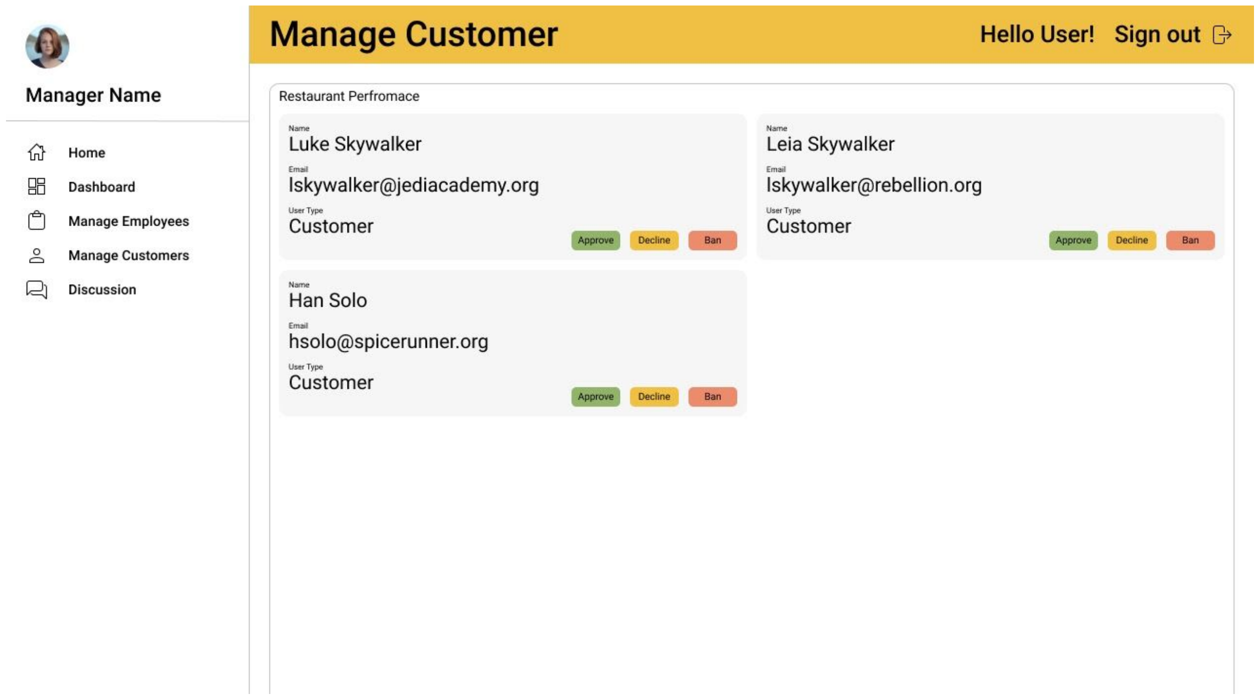
Restaurant Perfromace

Name
Obi wan Kenobi
Email
okenobi@jediacademy.org
User Type
Chef
Approve Decline Ban

Name
Anikan Skywalker
Email
askywalker@jediacademy.org
User Type
Chef
Approve Decline Ban

Name
Mace Windu
Email
mwindu@jediacademy.org
User Type
Delievery Person
Approve Decline Ban

Manager “Manage Employees” Page


Manage Customer

Hello User! Sign out

Manager Name

Home
Dashboard
Manage Employees
Manage Customers
Discussion

Restaurant Perfromace

Name
Luke Skywalker
Email
lskywalker@jediacademy.org
User Type
Customer
Approve Decline Ban

Name
Leia Skywalker
Email
lskywalker@rebellion.org
User Type
Customer
Approve Decline Ban

Name
Han Solo
Email
hsolo@spicerunner.org
User Type
Customer
Approve Decline Ban

Manager “Manage Customers” Page

Figma Design Link:
https://www.figma.com/file/47y9gFkwOwXnaY5towd9QI/Foodtopia-Design?node-id=0%3A1

Prototype Video Link: (created using figma):
https://drive.google.com/file/d/1oau_jMYhzmGjG22a5yaQK-WOOqCHqhzj/view?usp=sharing

6. Minutes of group meetings

10/02/2020: 2.5 hours long (initial planning, setup, specification discussion)

- Set up our backend and frontend repos, AWS server, MongoDB, and installed typescript and all the necessary dependencies. Discussed the restaurant system specifications, requirements, and additional features we would like to add. Planned our first sprint and divided up the tasks on Jira, to be completed before our next meeting.

10/16/2020: 2 hours long (phase 1 report)

- Discussed the phase 1 software requirements specification report, its requirements, our system's dependencies, and use cases. Divided up the tasks on Jira, to be completed by the 20th of October.

10/25/2020: 2 hours long (planning for next sprint)

- Reviewed completed tasks (mostly models and routes) and divided up the next set of tasks, which included routes, models, and controllers. We wrote out most of our system requirements in our first meeting in our team's backend specs Google Doc.

We plan to meet once a week until the final demo to ensure we are all staying on top of our tasks in each sprint and not running behind schedule.

7. Git repositories

Backend repo:

https://github.com/zealptl/swe-project-backend

Frontend repo:

https://github.com/zealptl/swe-project-frontend