CS387: Final Project Report

Team

Name	Email	Roll Number
Parth Laturia	parthlaturia@cse.iitb.ac.in	180050071
Anish Deshpande	anishdeshpande@cse.iitb.ac.in	180100013
Rajat Jain	rajat2001@cse.iitb.ac.in	180100091
Devansh Chandak	dchandak@cse.iitb.ac.in	180110027

Topic

Restaurant Management System

Table of Contents

a. Requirements

Descriptions

1. **Easy maneuvering** for the user over the **menu**.
2. It is a basic necessity that the user is able to **choose the dishes** he/she wants to eat and place an order of the list of items he has chosen for his/her meal. This use case is designed to support this need of the users. After placing an order, the orders relation will be updated.
3. The need to **update/manage employees** after sufficiently long intervals of time. Only the owner can update this relation. Many details of employees may change (eg: contact numbers, roles, address etc.). Moreover employees may leave or new employees may join. It is important to have such a use case.
4. We should be able to view and **analyse statistical data**: the consumption, food wastage being done, the profit being earned, the order frequency and the amount spent by different customers, the reviews of the dishes, the Total cost of dishes made by the chefs and the total servings done by the waiters.
5. After the user is done with the ordering of the corresponding dishes for his/her meal, we need to **allot** and assign the **ordered dishes to the appropriate chefs** using the mapping order -> chef by the cooks relation. This use case is designed for it.
6. the owner needs to have an account of the successful/unsuccessful **delivery of orders** to the customers
7. The need to **update inventory stock** after sufficiently long intervals of time. Only the owner can update this relation. Inventory here includes the need to update the stock in ingredients relation and the dishes relation.
8. After the chef completes making the dishes in the order and marks it we have to serve it to the customers. So a **mapping for order->waiter** has to be made which is done.
9. The **dish recommendation** part is quite common in today's E-Commerce/food delivery apps which helps users to decide faster and have a better experience by ordering the best quality & most loved products. There are two recommendations - most loved, your personal favorite. The other part we're automating is quantity flag

(flagging if quantity of any particular ingredient is below a certain threshold)so that the owner is always aware of what needs to be ordered without having to constantly keep checking the inventory.

Negative Requirements

10. The system must be able to **handle invalid inputs and searches** without crashing, and should direct the user appropriately to perform corrective action.
11. **Payments** for one order **mustn't be made twice**. A failed payment should be indicated and the customer shouldn't see that the order has been successfully placed. The order should not be allotted to the chef yet. (Not implemented).

Requirements based on who has logged in

After login:

The customer shall be able to:

- ❏ View his profile
- ❏ View recommended orders for him based on dishes ordered most times in the past by him
- ❏ Filter the menu of the restaurant categorically (cuisine, starter-main-dessert).
- ❏ View the cost and image of each dish.
- ❏ Then, he/she will select, one-by-one, the dishes desired (can select quantity)
- ❏ View his order history
- ❏ Finally, a payment option will be present. All of these details comprise the order. Upon completion, the order will be recorded in the order history.
- ❏ Logout

The owner/administrator shall be able to:

- ✓ View all the inventory (the pantry) that the restaurant has, update it, delete it or add a new ingredient
- ✓ Time Series Analytics 1: View all the dishes ordered on a date range specification. Owner can select start and end date (Order History)

- ✓ View Employee details (salary, name, ID, age etc.). Owner can add new employees, update employee information or delete employee from the relation
- ✓ Allot an order to a chef.
- ✓ Allot the delivery of a prepared order to a waiter.
- ✓ View analytics : popular dishes, waiters and chefs the with most completed orders, most loyal customers in both table and graph form
- ✓ Time Series Analytics 2: Get profit, expenditure, wastage of day-to-day basis based on filtering on start date and end date given by the owner
- ✓ Statistical Report: Owner can see a different page opens with statistical information: mean, standard deviation and day with maximum and minimum wastage/profit/expenditure
- ✓ Logout

The Employee shall be able to:

- ✓ View the orders allotted to them by the owner and fulfill those: if cook he can mark cooked and if a waiter he can marks it served
- ✓ View his profile
- ✓ See the number and total cost of orders they've fulfilled in the past
- ✓ Logout

b. Use case descriptions

List of Use Cases, elaborated on the bold ones after that -

Customer:

User registration, Logging into the system, **Searching for dishes(Search Engine)**, Add to Cart, **Place an Order**, Previous Orders, Cancel Order, My Preferred Order, Get Discount, Recommender System

Owner:

Add Employee, **Manage Employee Info**, **Analytics** of: Sales, Customer Visits, **Allot Order to Chef**, **View Order History**, Employee Leaderboard

Employee:

Mark completed orders, **Manage(update) inventory**, **Allot order to waiter(Serving Food)**, Orders completed, **Automation**

Following are our use cases:

Search Engine

1. Search Engine.

2. The user can search for dishes by name, id of dish.

3. Human trigger.
4. Customer will be the primary actor, even employees can use it to show the menu to the visitors.
5. Input will be the kind of Cuisine or Category, Dish name or Dish id.
6. Users must be logged in.
7. Based on keyword matching the database will return relevant results.
8. In case of wrong input, “Sorry, no matching dishes!”/no dishes will be displayed.
9. The user can go through the results displayed, check the ingredients and choose food/ search again.

Place an Order

1. Place an Order.
2. The user can place an order using the menu in the restaurant.
3. Human Trigger.
4. The customer who places the order.
5. The list of orders (specified by the dish_ID) that the user will provide using a form. The dish in the order must be present in the dishes entity .
6. The customer must be logged in.
7. The main set of interactions consist of filling the form and inputting the order dishes into the system. The final output is successful placing of order
8. The user places an order for some dishes not on the menu. (Not possible)
9. The order will have been placed and sent to the owner for further processing.

Manage Employee Information

1. Manage employee info.

2. Update the salary, add employees, find best performing employees, other personal information.
3. Human trigger.
4. The owner of the restaurant.
5. Enter employee name to edit their details, monitor their performance.
6. Must have admin rights+logged in.
7. Navigate to the employee tab on the admin interface.
8. None.
9. If the owner makes any edits and saves them then those changes will be pushed to the database and will comprise the new state.

Analytics: (with Time Series)

1. Analytics
2. The system generates some insightful analytics that provides a realistic view of the restaurant system and its quantitative features to the user.
3. Automatic Triggered
4. System and its computational logic
5. The statistical measures and values of restaurant's attributes and features (like expenditure, Food_wasted,...) on a daily basis
6. The user is logged in and must be an owner of the restaurant
7. The values are represented in an elegant and understandable format and successfully capture the trends of the provided data
8. The data type and attributes are not matching as per the conditions
9. The owner has captured the required analysis

Allot Order to Chef

1. Allot Order to Chef.
2. The Order will be allotted to the chosen/designated chef .
3. Human Trigger.
4. The owner is involved as he/she will assign the corresponding orders and so the owner will be the main actor interacting with the system.
5. The orders received from the users. Though the user may order multiple dishes, the ordered list will be broken down into smaller orders such that each order has only 1 dish in it. This will be done for each user and this will finally become the input to the system
6. The corresponding user must have given some order that the owner is currently transferring to the chef(s).

7. The chef(s) receive the order.
8. No chef is alloted the given order/ is allotted to an absent employee.
9. Those chefs who received the order will now start preparing it. The chefs have their own action, where they canteen designate an order as prepared and ready it for service.

Allot Order to Waiter

1. Allot Order to Waiter.
2. The order will be allotted to the chosen/designated waiter.
3. Human Trigger.
4. The manager/owner is involved as he/she will assign the corresponding orders and so the owner will be the main actor interacting with the system.
5. The orders prepared by the cooks. Though the user may order multiple dishes, the ordered list will be broken down into smaller orders such that each order has only 1 dish in it. This will be done for each user and this will finally become the input to the system
6. The corresponding user must have given some order that the manager is currently transferring to the chef(s)
7. The waiter(s) receive the order.
8. No waiter is alloted the given order/ is allotted to an absent employee
9. Those waiters who received the order will now have the option of serving it in their page.

Order History

1. Order History.
2. Get a list of all previous orders made by customers.
3. Triggered automatically.
4. Employee/Owner/Customers.
5. The user can input a particular date to see the order history.
6. Users must have non-customer status to see full order history.
7. Main set of interactions involved choosing the right filter for viewing the relevant history. The result is a table consisting of all relevant orders, their date, quantity, revenue etc.
8. None
9. The state doesn't change, since order history is view-only.

Update Inventory

1. Update Inventory.
2. Based on the dishes prepared and the incoming ingredients, update the inventory.
3. Human triggered.
4. Owner.
5. Choose the ingredient to update and increment quantity.
6. Users must have non-customer status.
7. Main set of interactions involve entering the right amount added/subtracted. The final result is the update of the database.
8. The quantity cannot be negative.
9. The new changes will be pushed to the database.

Cooking/Serving Food

1. Owners allot orders to waiters/cooks.
2. The Order will be allotted to the designated waiter/cook.
3. Human Triggers.
4. The employee is involved as he/she will assign the corresponding orders and so employee will be the main actor interacting with the system
5. As soon as the order is completed by the employee (owner), they can allot the order to a waiter, who will serve the client then or to a cook who will cook the order
6. Owner login
7. Employee (owner) checks available waiters and allots the task. The output is that the task is transferred to the waiter and marked completed by the employee.
8. Order is allotted to an absent waiter/cook
9. The order moves from the owner interface to waiter/cook interface, marked done on the employee interface, gets added to the employee performance table

Automation:

1. Personalization.
2. Track customer behaviour, ease ordering by recommending frequently ordered/ famous foods, notification to the owner if some ingredient's quantity falls below a threshold.
3. Automatically triggered.
4. Main actor is the backend, the customer is the beneficiary.

5. User does not give any specific input, the recommendation is based on the history of orders the user has made so it happens automatically.
6. Previous user data is required.
7. While searching food, placing an order - the automation will help reduce hassle by remembering previous information.
8. No serious complications, just that initially the recommendations may not be useful for users due to small size of training data.
9. Users can order on their own or choose from the recommendations, order ingredients if value falls below threshold.

c. Design

Final Schema

Our data is already normalised. (It is present in BCNF form)

DDL:

```
DROP TABLE IF EXISTS Accounts CASCADE;

DROP TABLE IF EXISTS Contains CASCADE;

DROP TABLE IF EXISTS Cooks CASCADE;

DROP TABLE IF EXISTS Customers CASCADE;

DROP TABLE IF EXISTS Customers_Contact CASCADE;

DROP TABLE IF EXISTS Delivers CASCADE;

DROP TABLE IF EXISTS Dishes CASCADE;

DROP TABLE IF EXISTS Ingredients CASCADE;

DROP TABLE IF EXISTS Non_waiting_customer CASCADE;

DROP TABLE IF EXISTS Orders CASCADE;

DROP TABLE IF EXISTS Pay_By CASCADE;

DROP TABLE IF EXISTS Payment CASCADE;

DROP TABLE IF EXISTS Sitting CASCADE;

DROP TABLE IF EXISTS Sit_Table CASCADE;

DROP TABLE IF EXISTS Staff CASCADE;
```

```
DROP TABLE IF EXISTS Staff_Contact CASCADE;
DROP TABLE IF EXISTS Tracks CASCADE;
DROP TABLE IF EXISTS Waiting_customer CASCADE;
CREATE Table Accounts(
        Work_Date Date PRIMARY KEY,
        Restaurant_profit int,
        expenditure int CHECK(expenditure>=0),
        Total_food_wasted real CHECK(Total_food_wasted>=0)
);
CREATE Table Staff (
        ID int PRIMARY KEY CHECK(ID>=1),
        Name_FN varchar(100),
        Name_MN varchar(100),
        Name_LN varchar(100),
        Gender varchar(20),
        Age int CHECK(Age>=1),
        Salary bigint CHECK(Salary>=0),
        Experience int CHECK(Experience>=0),
        Occupied int CHECK(Occupied<=1 and Occupied>=0),
        TCDM int CHECK(TCDM>=1),
        Role varchar(20)
);
CREATE Table Staff_Contact(
        Contact bigint PRIMARY KEY CHECK(Contact>=1000000000 and Contact<=9999999999),
        ID int,
        foreign key(ID) references Staff on delete cascade
);
```

```
CREATE Table Dishes (
        Dish_ID int PRIMARY KEY,
        Cuisine varchar(50),
        Category varchar(50),
        Cost int CHECK(Cost>=0),
        Name varchar(50)
);
CREATE Table Ingredients (
   Ingredient_ID int PRIMARY KEY,
   Ingredient_Name varchar(50),
   Quantity int CHECK(Quantity>=0)
   -- Quality int CHECK(Quality>=1 and Quality<=10)
);

CREATE Table Contains (
        Ingredient_ID int CHECK(Ingredient_ID>=1),
        Dish_ID int CHECK(Dish_ID>=1),
        Quantity_used int,
        PRIMARY KEY (Ingredient_ID, Dish_ID),
        foreign key (Dish_ID) references Dishes on delete cascade,
        foreign key (Ingredient_ID) references Ingredients on delete cascade
);
CREATE Table Non_waiting_customer (
        ID int PRIMARY KEY,
        Amount_Spent int CHECK(Amount_Spent>=0),
        Payment_Method varchar(20)
);
```

```
CREATE Table Orders (
        Order_ID int,
        ID int,
        Dish_ID int,
        Work_Date Date,
        Work_Time Time,
        Day varchar(9),
        Quantity_Ordered int,
        Review int CHECK(Review>=1 and Review<=5),
        Cost int CHECK(Cost>=0),
        PRIMARY KEY (Order_ID, Dish_ID),
        foreign key (ID) references Customers on delete cascade,
        foreign key (Dish_ID) references Dishes on delete cascade
);
CREATE Table Cooks (
        ID int,
        Order_ID int,
        Dish_ID int,
        Completed int CHECK(Completed>=0 and Completed<=1),
        PRIMARY KEY (ID, Dish_ID, Order_ID),
        foreign key (ID) references Staff on delete cascade,
        foreign key (Order_ID, Dish_ID) references Orders on delete cascade
);
CREATE Table Customers (
        ID int PRIMARY KEY CHECK(ID>=1),
        Name_FN varchar(100),
        Name_MN varchar(100),
```

```
	Name_LN varchar(100),
	Gender varchar(20),
	Age int CHECK(Age>=1),
	Order_Frequency int CHECK(Order_Frequency>=0)
);
CREATE Table Customers_Contact(
	Contact bigint PRIMARY KEY CHECK(Contact>=1000000000 and Contact<=9999999999),
	ID int,
	foreign key(ID) references Customers on delete cascade
);
CREATE Table Payment (
	Bill_ID int PRIMARY KEY,
	Work_Date Date,
	Discount_offered int CHECK(Discount_offered<=100 and Discount_offered>=0)
);
CREATE Table Pay_By (
	ID int,
	Bill_ID int,
	Payment_mode varchar(50),
	Invoice_ID int,
	Invoice_description varchar(50),
	Status varchar(50),
	PRIMARY KEY (ID, Bill_ID), -- will only Bill_ID work ?
	foreign key (ID) references Non_waiting_customer on delete cascade,
	foreign key (Bill_ID) references Payment on delete cascade
);
CREATE Table Sit_Table (
```

```
    Table_ID int PRIMARY KEY,
    Size int,
    Status varchar(20),
    Location varchar(50)
);
CREATE Table Sitting (
    ID int PRIMARY KEY,
    Table_ID int CHECK(Table_ID>=0),
    foreign key (ID) references Non_waiting_customer on delete cascade,
    foreign key (Table_ID) references Sit_Table on delete cascade);
CREATE Table Tracks (
    ID int,
    Work_Date Date,
    PRIMARY KEY (ID, Work_Date),
    foreign key (ID) references Staff on delete cascade,
    foreign key (Work_Date) references Accounts on delete cascade
);
CREATE Table Delivers (
    ID int,
    Order_ID int,
    Dish_ID int,
    Completed int CHECK(Completed>=0 and Completed<=1),
    PRIMARY KEY (ID, Dish_ID, Order_ID),
    foreign key (ID) references Staff on delete cascade,
    foreign key (Order_ID, Dish_ID) references Orders on delete cascade
);
CREATE Table Waiting_customer (
```

```
    ID int PRIMARY KEY,
    Waiting_Number int CHECK(Waiting_Number>=0));
CREATE INDEX Order_Serial_Number on Orders(Order_ID, Dish_ID);
```

Note: There are 18 other indices - one corresponding to the primary key of each relation which is created automatically. So total - 19 indices in the DDL.

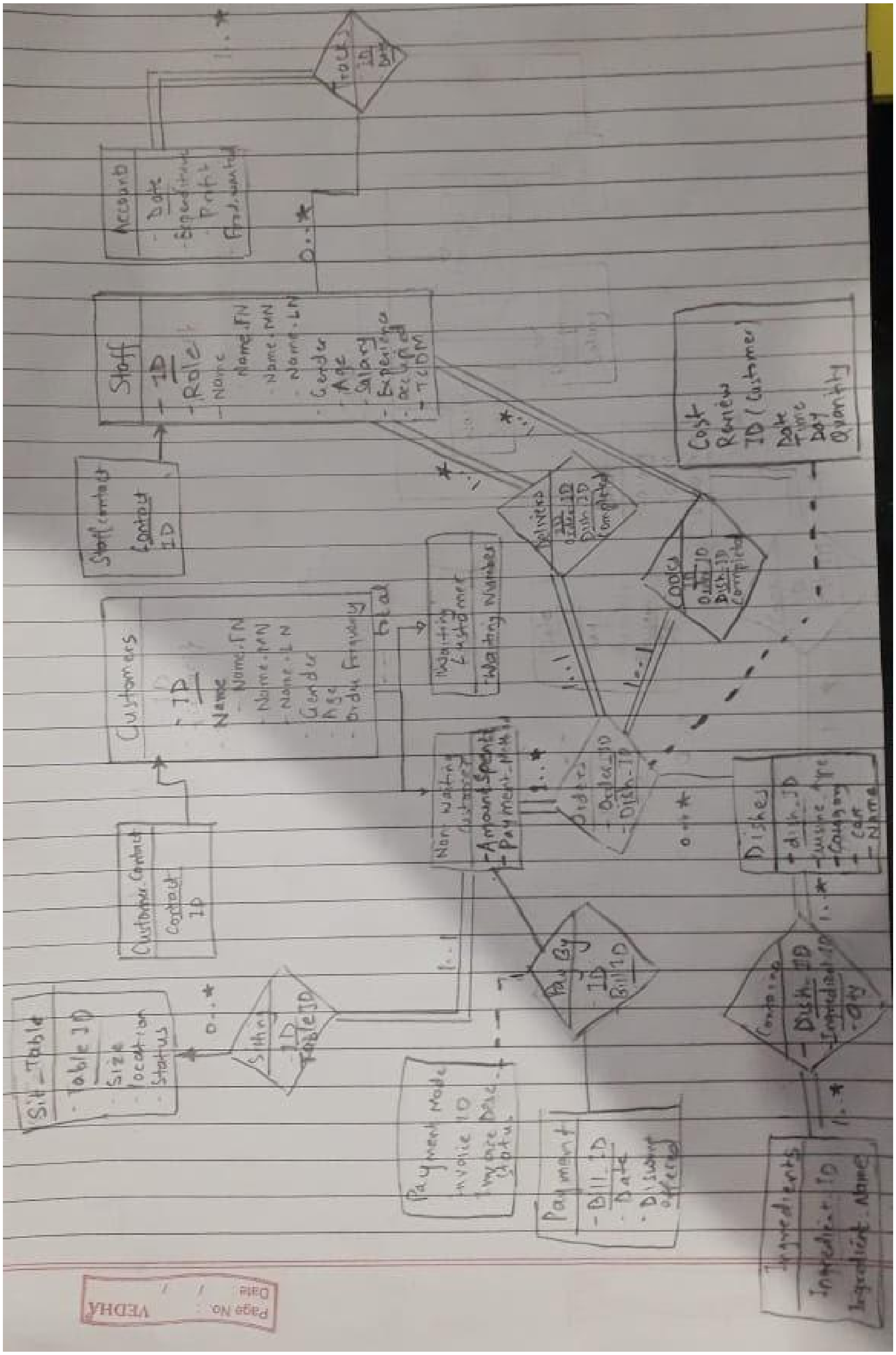
Normalization

Our schema has been decomposed to **BCNF** since all the functional dependencies *A -> B* valid for every relation are either trivial or *A* is a superkey for the relation.

The foreign key dependencies ensure **dependency preservation** while performing joins. This is ensured since we have only involved primary keys of the referencing and referenced relation in the dependency.

The above 2 points can easily be seen in the DDL given above.

ER Diagram



Staff
Accounts
Customers
Dishes
Payment
Ingredients
Sit_Table
Waiting Customer
Non Waiting Customer
Orders
Pay By
Staff contact
Customer Contact

Technology Choices

We have used the **MVC architecture in Node.Js** for our application. The database manager is PostgreSQL, accessed via either the terminal or via pgAdmin4.

The reason for this is its robustness and ease of use. We have been introduced to Node in this course, thus we decided to continue along those lines. The four of us worked on different screens and different features of the website. The Express framework simplified the coding process. Each could be constructed in parallel using separate model, controller, view and route files. This modularity enabled a relatively simple merge as well as the option to add and remove features with ease. Such possibilities are a testament to the efficacy of MVC.

We also have experience in connecting to and interacting with a database in PostgreSQL, and it was the natural choice for our project.

Activity with Bootstrap

Scripting with HTML and JavaScript.

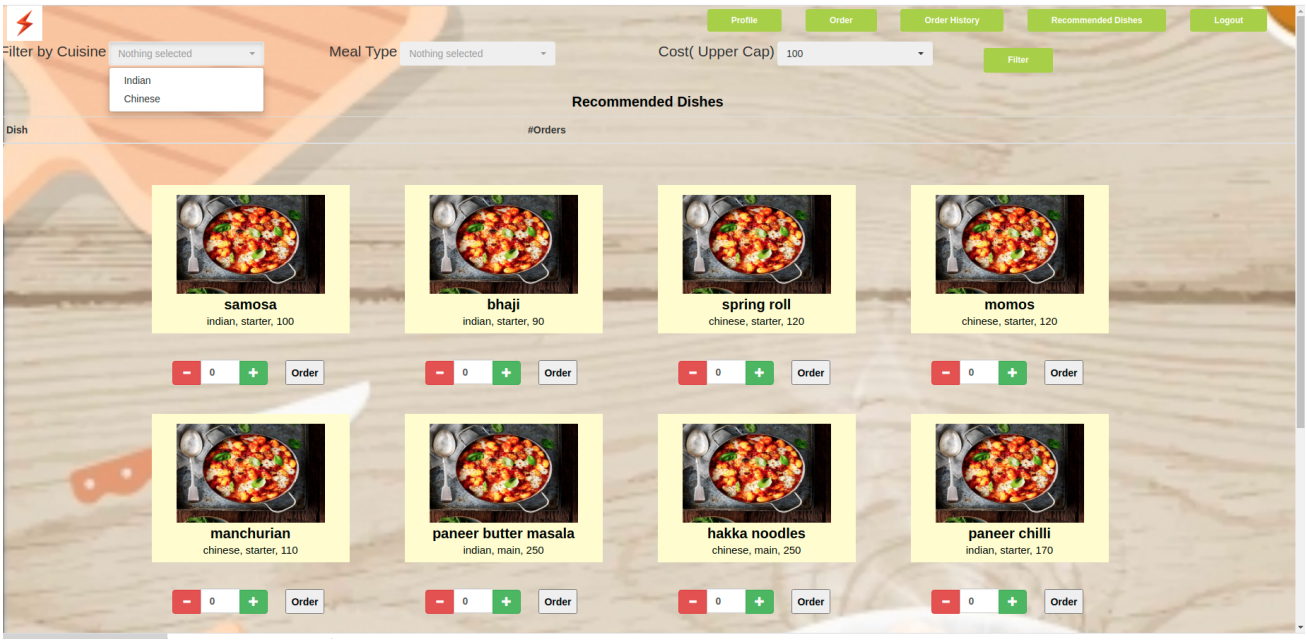
Styling with CSS.

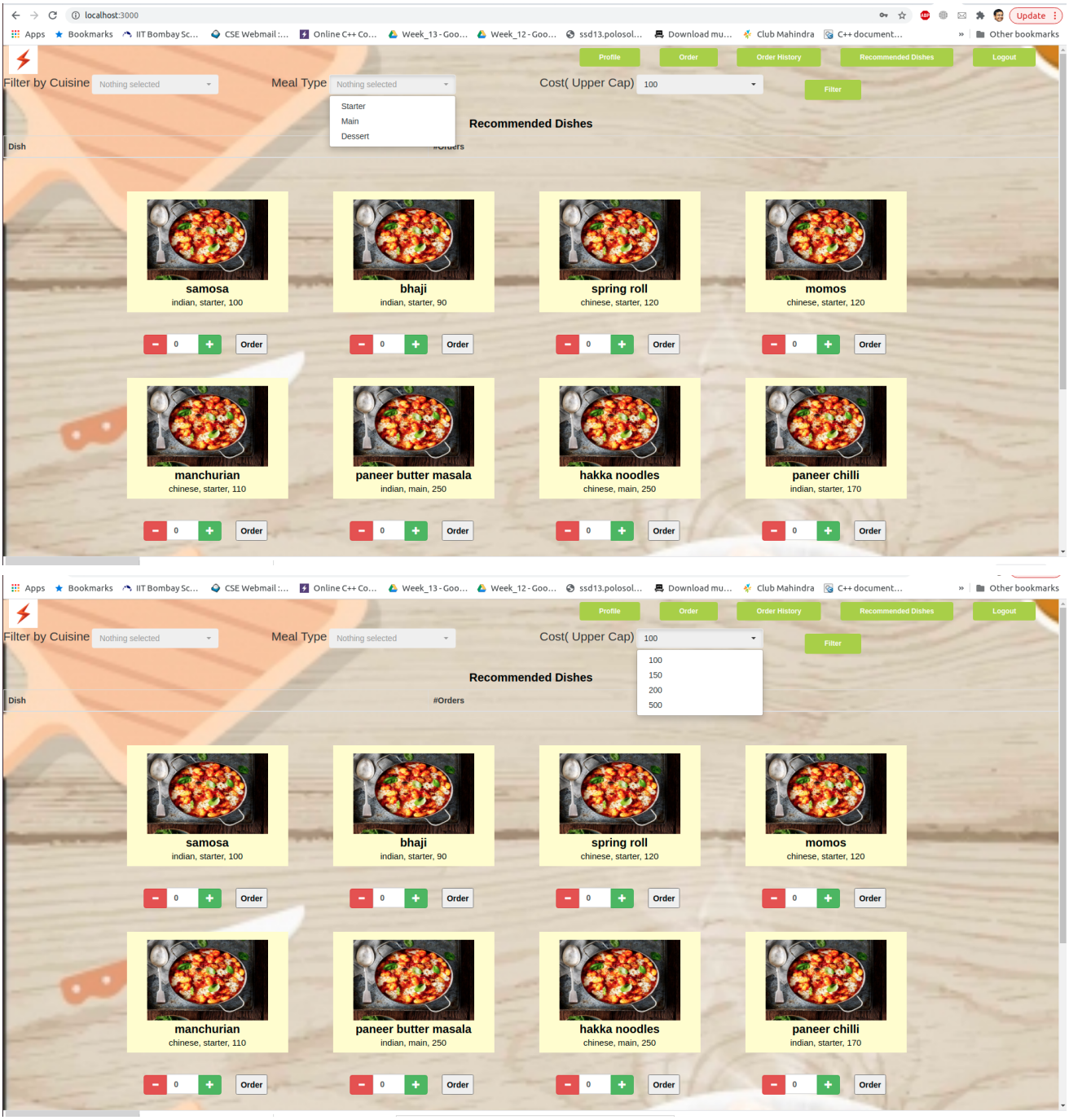
d. Test Results according to your test plan.

Test Results on individual use cases

- Test for Search Engine: PASSED

Our Final Screen(s):





Input:

Cuisine or category or price.

Output:

Relevant dishes

Test Procedure:

Homepage to load - click on the corresponding box for the dropdown that provides the options. Then, click on the filter button on the right to filter the items.

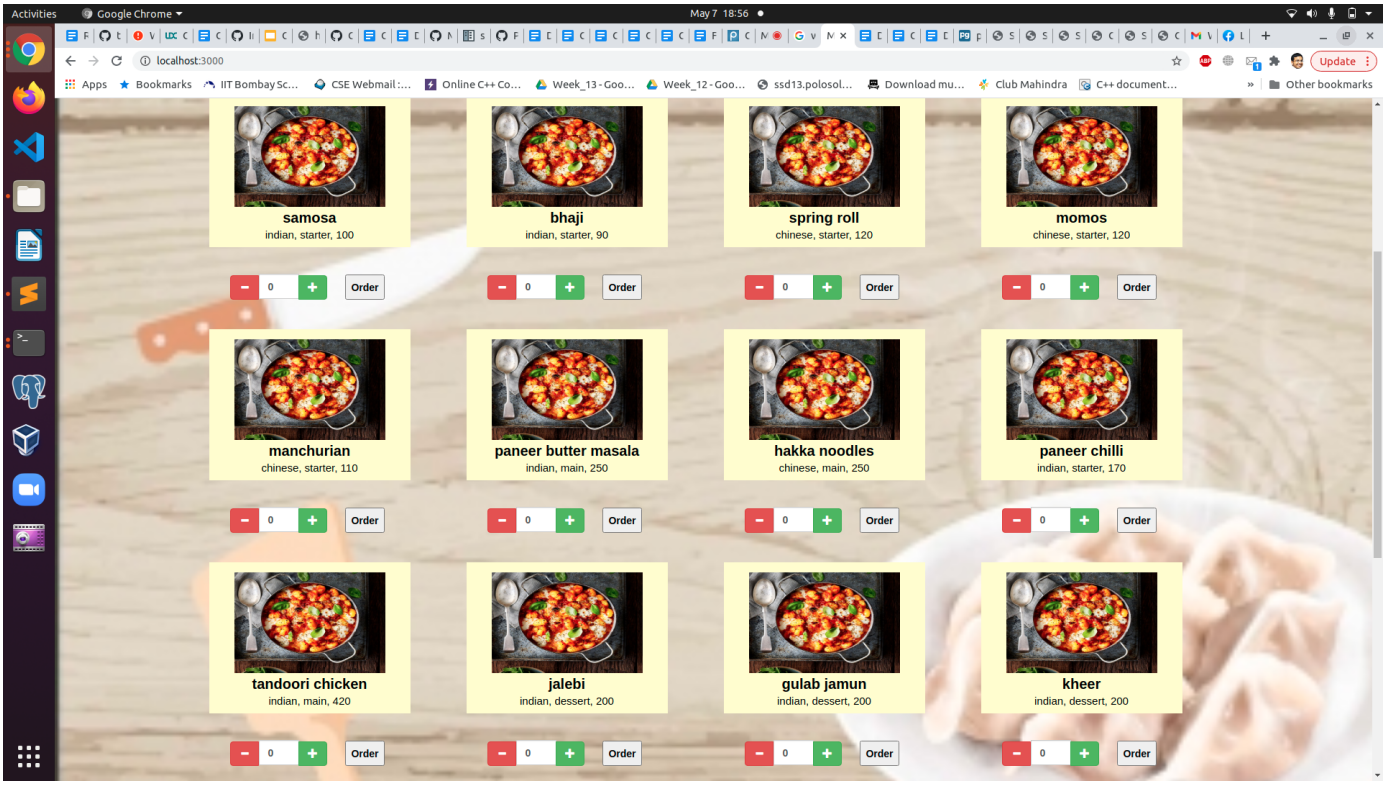
Negative tests:

Since we're providing drop-down for choosing cuisine/category and dish name, there is no chance that users can provide wrong inputs and hence no negative tests are needed here.

- **Test to Place an Order: PASSED**

Our Final Screen(s):

Input	Output
Order	PASSED



Input:

The primary input is the list of dishes and the customer ID that drives the further actions. Once, this is provided, this input is passed into the system using the “order” button. This will kind of put the customer’s order into a buffer so it can possibly be sent to chefs. Users can modify it until they do not click on “order”.

Output:

The order gets placed in the owner’s view.

Test procedure:

The customer should be able to place his/her order first by clicking on order and then on action. Once the user places the required order, the number of orders placed by that user till now, the number of times the corresponding dishes have been ordered till now

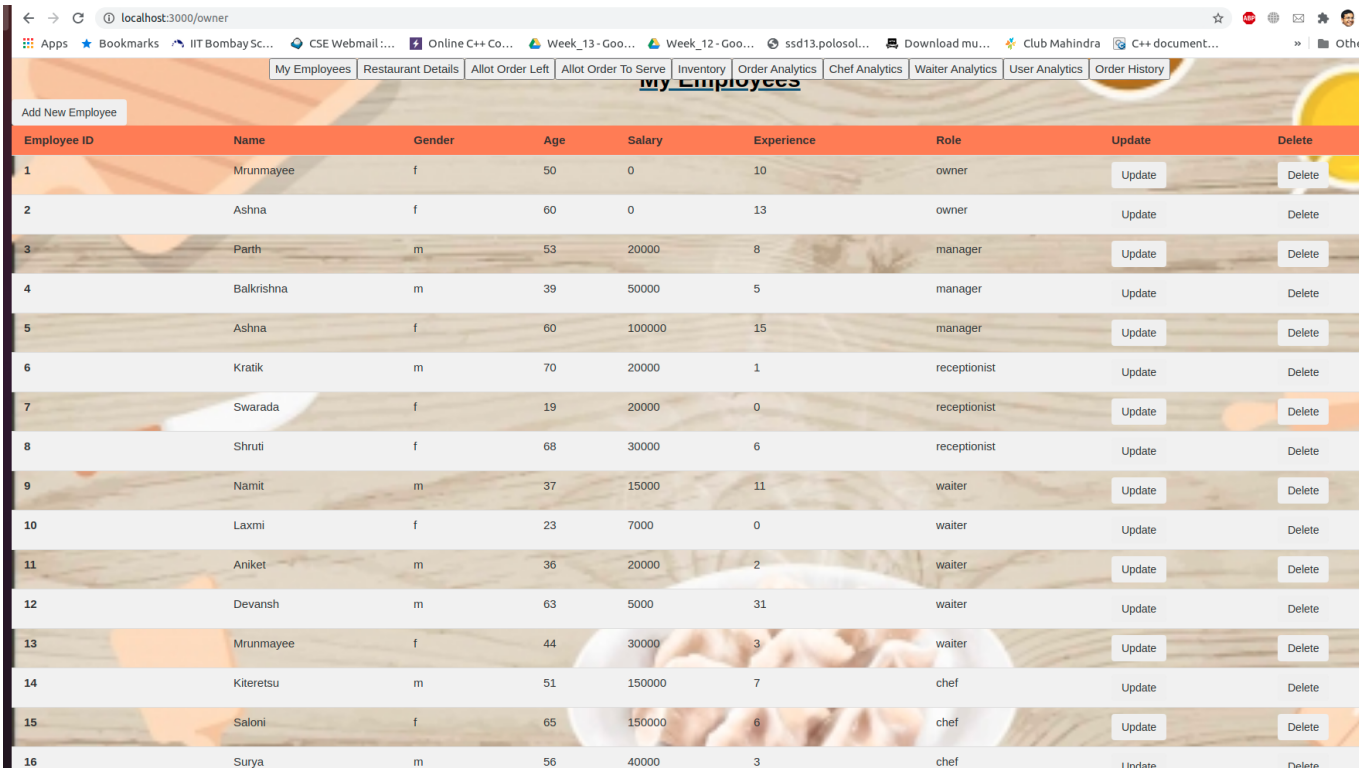
and other analytics related to this query are modified accordingly. If these conditions are satisfied, the test is successful.

- **Test to Manage Employee Information: PASSED**

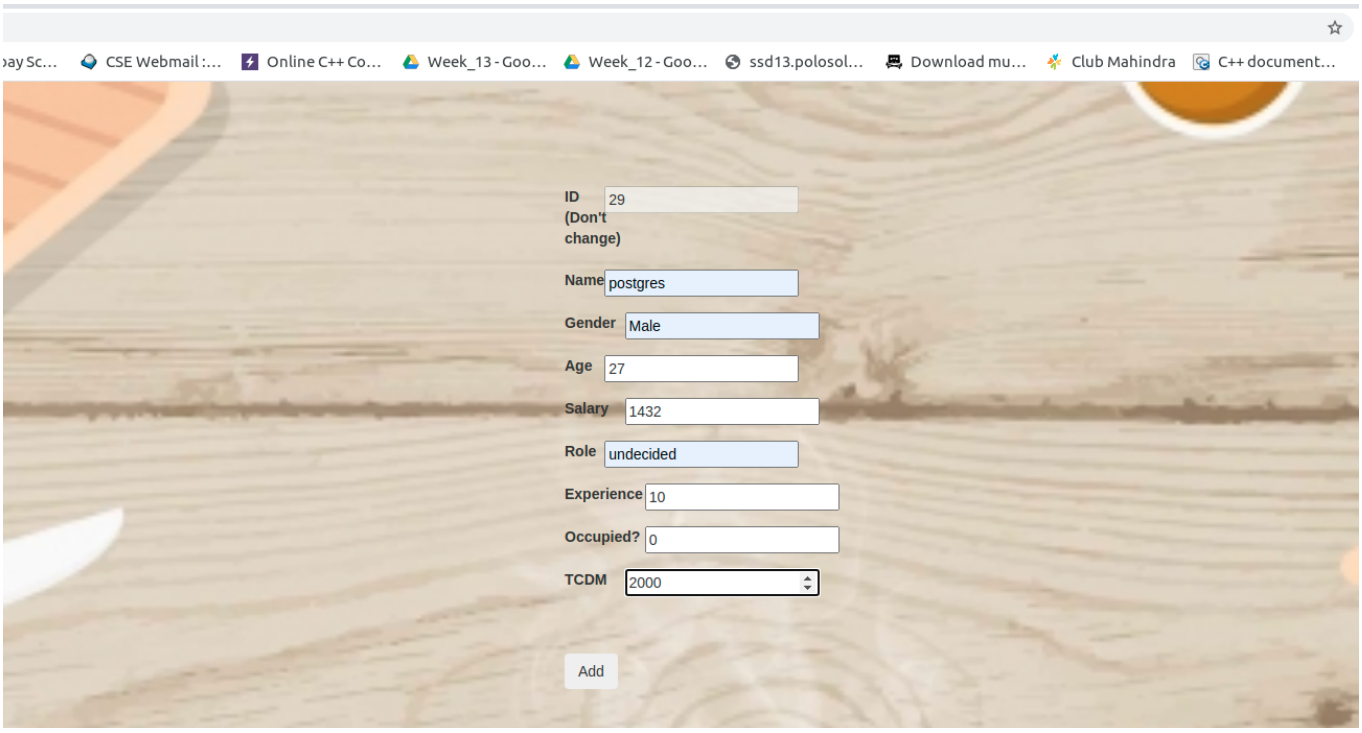
  Our Final Screen(s):

Input	Output
Create	PASSED
Update	PASSED
Delete	PASSED

Employee -> (ID, Name_FN, Gender, Age, Salary, Experience, Role, TCDM)

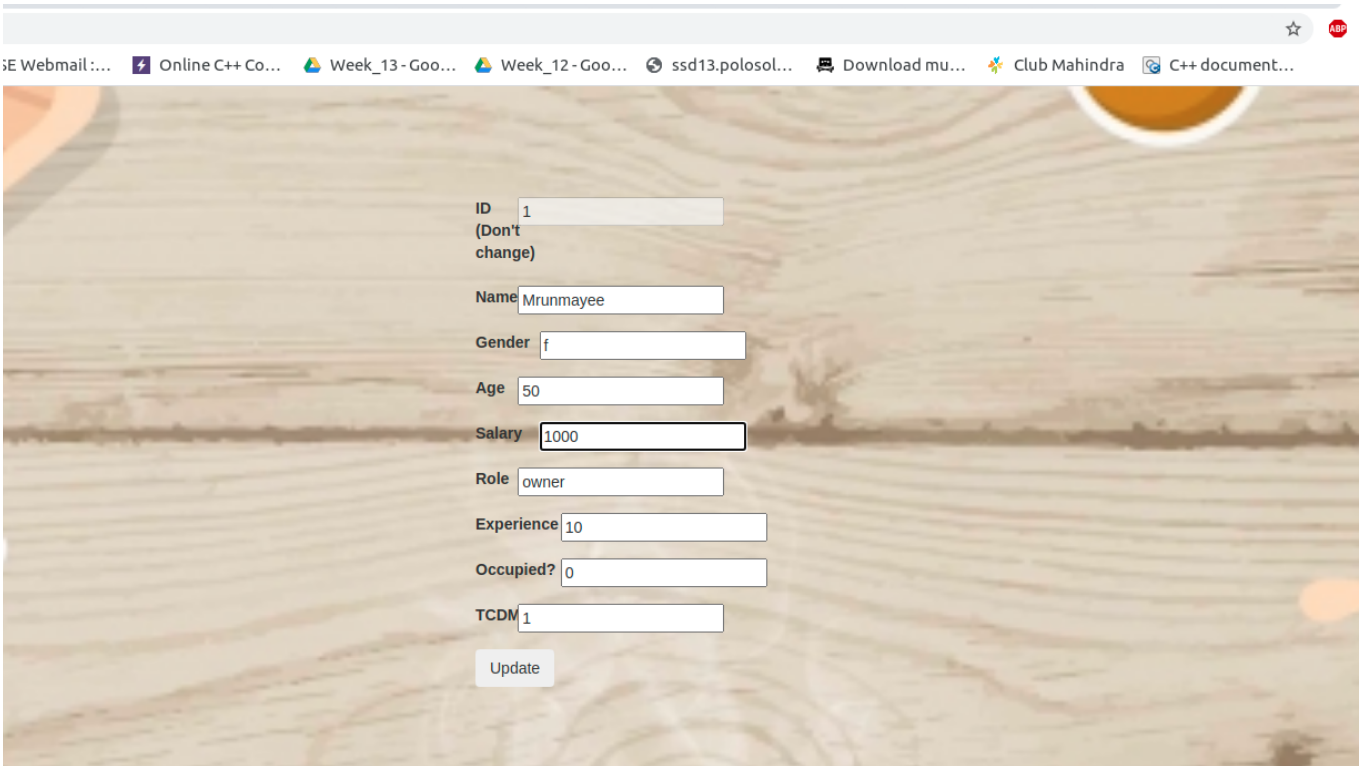


Employee ID	Name	Gender	Age	Salary	Experience	Role	Update	Delete
1	Mrunmayee	f	50	0	10	owner	Update	Delete
2	Ashna	f	60	0	13	owner	Update	Delete
3	Parth	m	53	20000	8	manager	Update	Delete
4	Balkrishna	m	39	50000	5	manager	Update	Delete
5	Ashna	f	60	100000	15	manager	Update	Delete
6	Kratik	m	70	20000	1	receptionist	Update	Delete
7	Swarada	f	19	20000	0	receptionist	Update	Delete
8	Shruti	f	68	30000	6	receptionist	Update	Delete
9	Namit	m	37	15000	11	waiter	Update	Delete
10	Laxmi	f	23	7000	0	waiter	Update	Delete
11	Aniket	m	36	20000	2	waiter	Update	Delete
12	Devansh	m	63	5000	31	waiter	Update	Delete
13	Mrunmayee	f	44	30000	3	waiter	Update	Delete
14	Kiteretsu	m	51	150000	7	chef	Update	Delete
15	Saloni	f	65	150000	6	chef	Update	Delete
16	Surya	m	56	40000	3	chef	Update	Delete



Tuples tried in create:

- (29, Cryptoknights, Male, 43, 14653, 6, undecided, 23)



Tuples tried in update:

- (1, Mrunmayee, Female, 65, 1002, 100, undecided, 107) -> (1, Boss, Male, 65, 1002, 100, undecided, 107)
- (1, Nathan, Male, 65, 1002, 100, undecided, 107) -> (1, Ramesh, Male, 65, 1002, 10, undecided, 36)

Input - Output

Input:

Update/Create (of CRUD) command. In case of delete the input is just the delete button press. In case of create/update after the button press a form opens up with old values/blank values for update and create respectively. Then the new values are typed in (as shown in screen design in Deliverable 3) and then the confirm button is pressed.

Note: Sample screens for input and output given below- they were also presented in Deliverable 3

Output:

Output is the updated grid view in the page (automatic display of the updated relation without need to reload the page). It will be part of the controller and backend logic.

Test procedure: (which screen to load, what action to take etc.)

The Manage Employee Information page is selected from the main menu/navigation bar of the website. Screen of this use case will then be loaded only if the login is of the owner else access is denied.

Then all 3 options will be tested. (on all the employee tables)

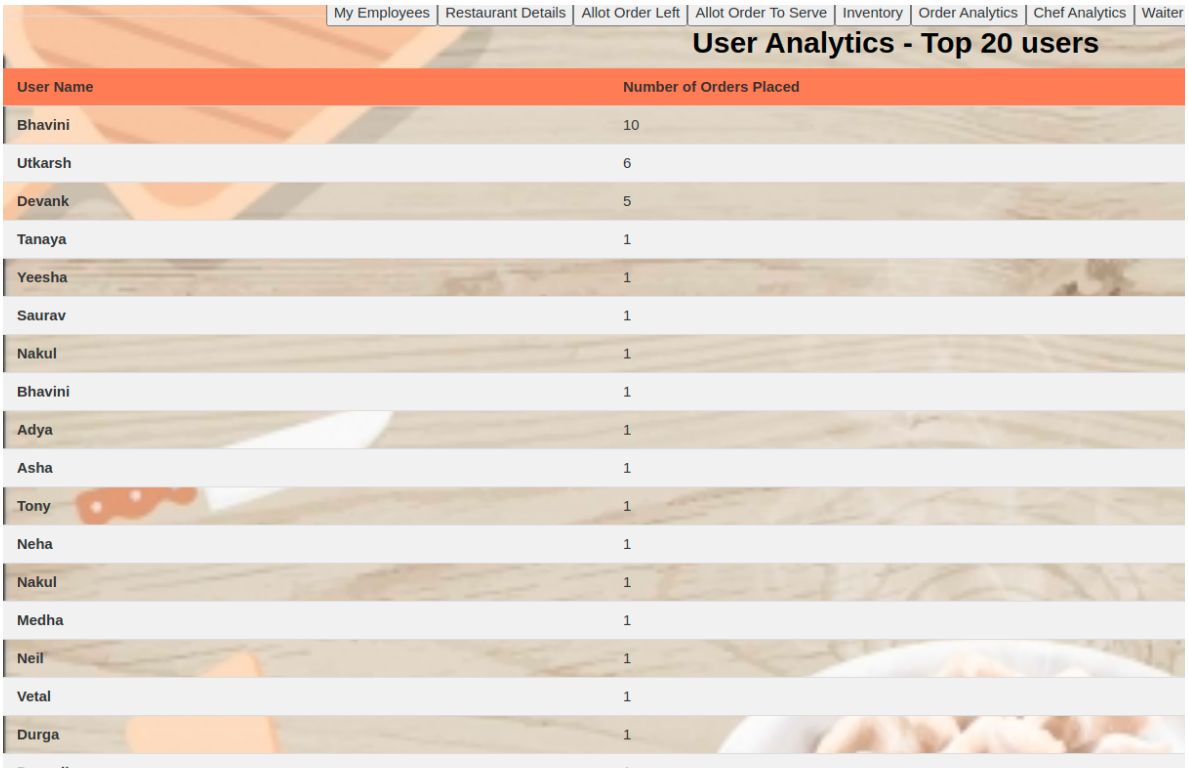
- Create: Add an employee with the asked attribute values in the form. This can be done by clicking on the “Add New Employee” Button.
- Update: Update existing employee information with the new attribute values in the form by clicking on the update button.
- Delete: Delete a tuple from a relation shown in the grid view by clicking on the delete button.

For the backend programmer one additional test can be to go to the database and check if the updates have taken place properly. This is in some ways redundant since grid view has a one-one correspondence with the backend relations.

- **Test for Analytics: PASSED**

Our Final Screen(s):

Input	Output
User Analytics	PASSED
Chef Analytics	PASSED
Waiter Analytics	PASSED
Order Analytics	PASSED
Restaurant Analytics	PASSED



Order History and Analytics

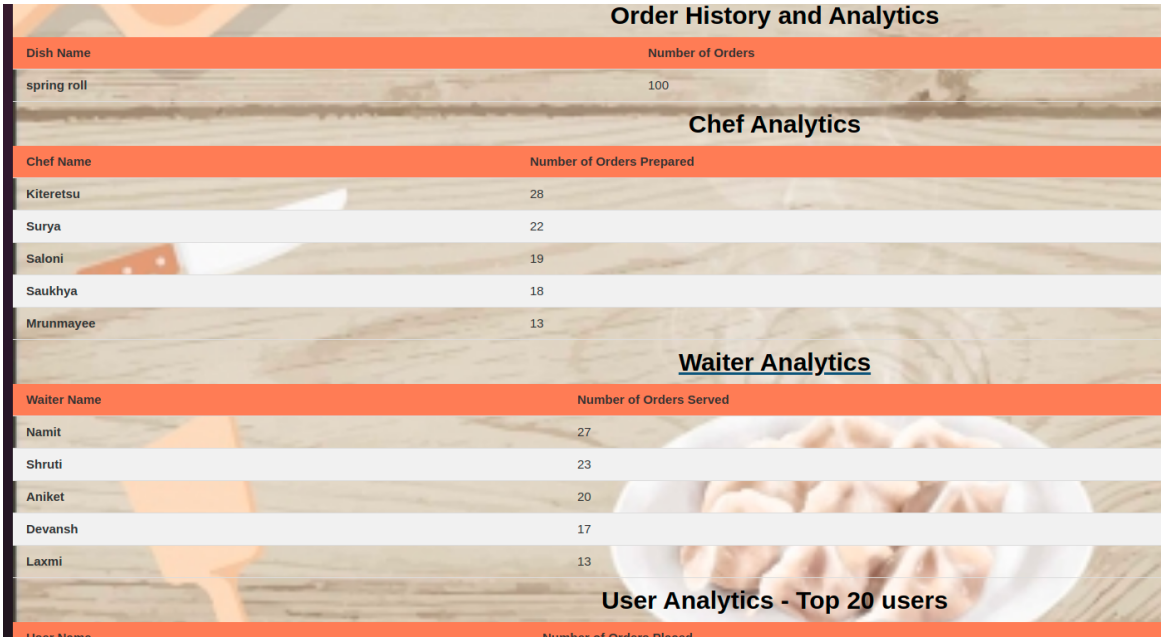
Dish Name	Number of Orders
spring roll	100
manchurian	2
gulab jamun	2
hakka noodles	2
momos	2
kheer	1
samosa	1

Chef Analytics

Chef Name	Number of Orders Prepared
Kiteretsu	30
Surya	22
Saloni	19
Mrunmayee	18
Saukhya	18

Waiter Analytics

Waiter Name	Number of Orders Served



Order History and Analytics

Dish Name	Number of Orders
spring roll	100

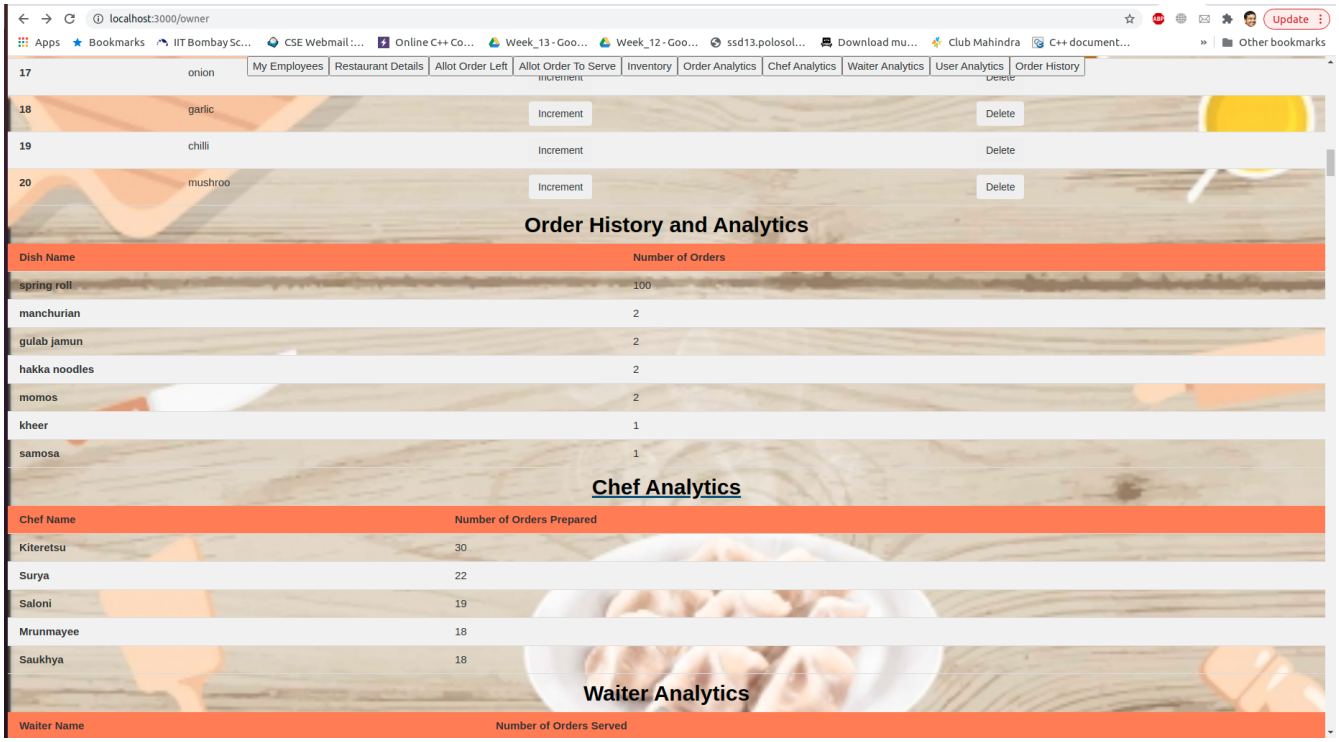
Chef Analytics

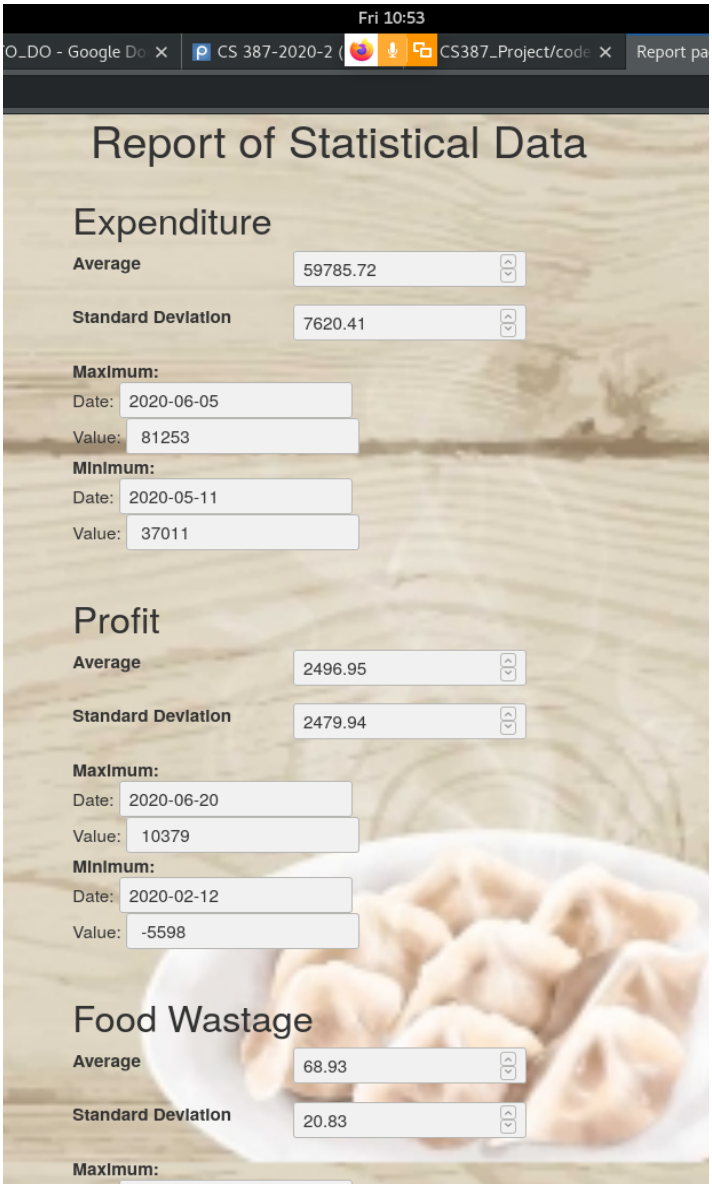
Chef Name	Number of Orders Prepared
Kiteretsu	28
Surya	22
Saloni	19
Saukhya	18
Mrunmayee	13

Waiter Analytics

Waiter Name	Number of Orders Served
Namit	27
Shruti	23
Aniket	20
Devansh	17
Laxmi	13

User Analytics - Top 20 users


localhost:3000/owner
Update
Apps
Bookmarks
IIT Bombay Sc...
CSE Webmail :...
Online C++ Co...
Week_13 - Goo...
Week_12 - Goo...
ssd13.polosol...
Download mu...
Club Mahindra
C++ document...
Other bookmarks
My Employees
Restaurant Details
Allot Order Left
Allot Order To Serve
Inventory
Order Analytics
Chef Analytics
Waiter Analytics
User Analytics
Order History
17 onion
18 garlic Increment Delete
19 chilli Increment Delete
20 mushroo Increment Delete
Order History and Analytics
Dish Name
Number of Orders
spring roll 100
manchurian 2
gulab jamun 2
hakka noodles 2
momos 2
kheer 1
samosa 1
Chef Analytics
Chef Name
Number of Orders Prepared
Kiteretsu 30
Surya 22
Saloni 19
Mrunmayee 18
Saukhya 18
Waiter Analytics
Waiter Name
Number of Orders Served



Input:

the quantities and amount of the required parameters to be analysed vis-a-vis the day, the amount of money spent on the day, the profit earned for the day, the food wasted for the day, the different orders placed by the different customers that arrived and ordered in the restaurant for the day, the different dishes cooked by different chefs and the different dishes served by different waiters. (First part implemented).

Output:

The output consists of plots and textual analysis for the attributes of different relations mentioned in the requirements.

Test Procedure:

The Analysis page is selected from the main menu/navigation bar of the website. Screen of this use case will then be loaded only if the login is of the owner else access is denied.

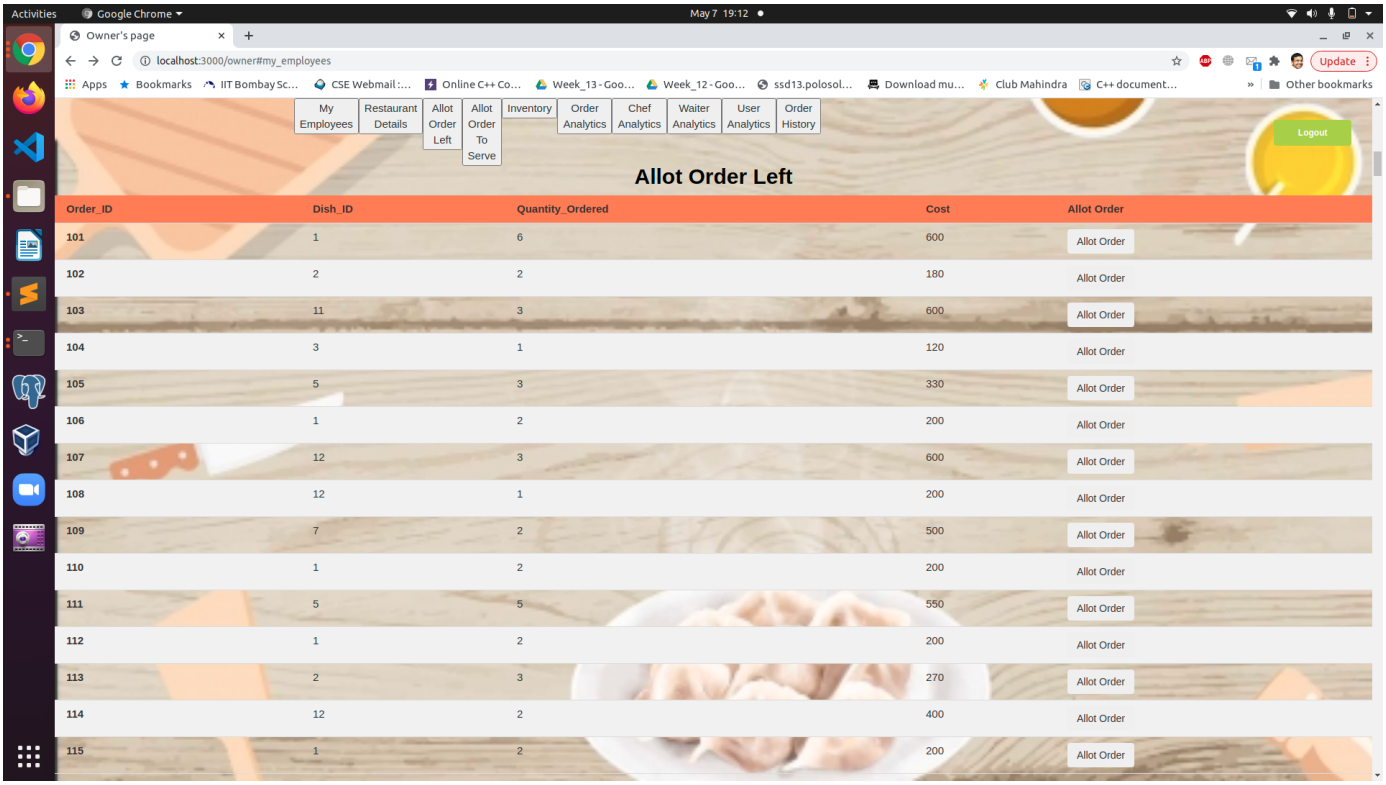
Once the login is successful, the owner has the option to view the analysis in the form of bar graphs, plots, etc. This will depend on the attribute to be analysed and the available analysis options.

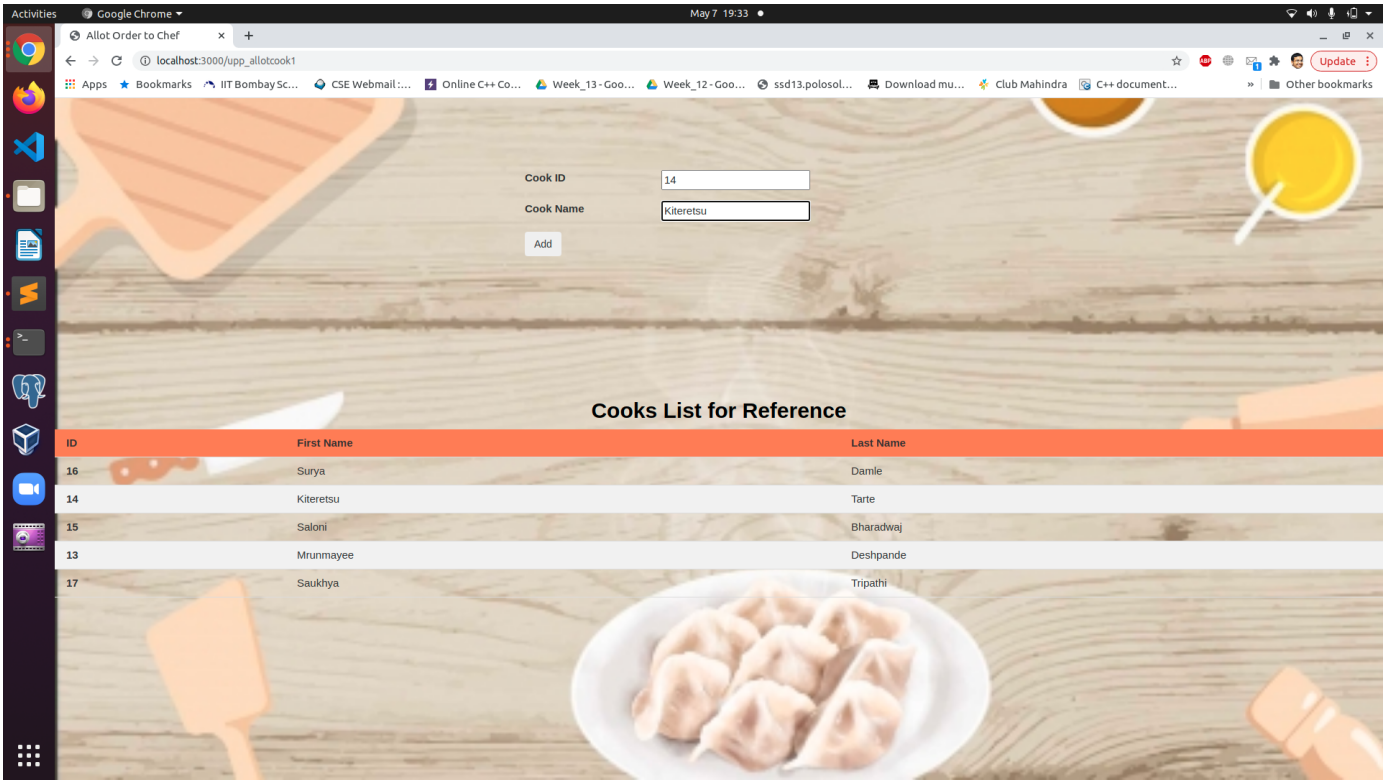
Also, this use case will also test whether the TCDM attribute of staff relation, Quantity, Review of the orders relation, the discount offered in the payment relation are modified for the day accordingly. (payment not implemented).

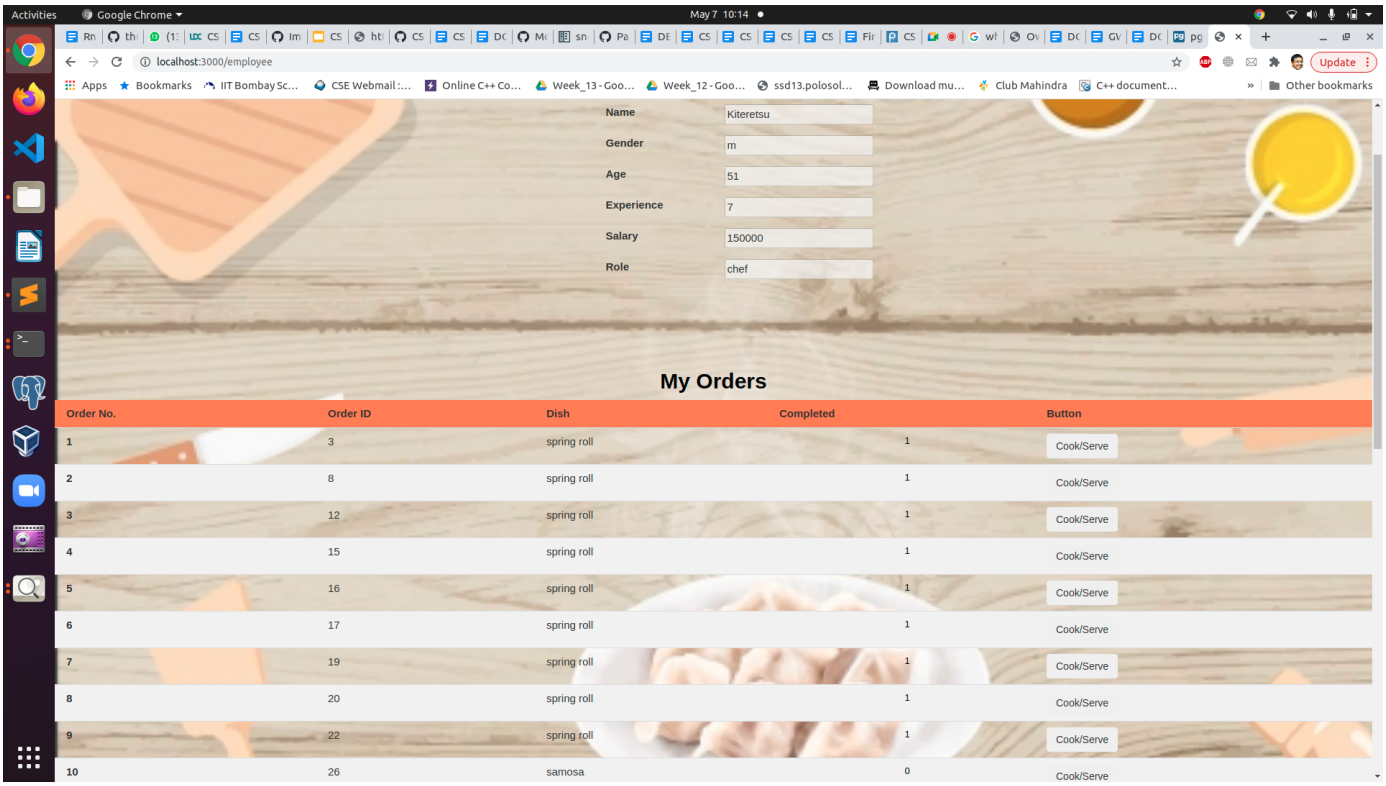
- **Test to Allot Order to Chef: PASSED**

Our Final Screen(s):

Input	Output
Allot Order	PASSED
Add	PASSED
Cook/Serve	PASSED




Activities
Google Chrome
May 7 19:33
Allot Order to Chef
localhost:3000/upp_allotcook1
Update
Apps
Bookmarks
IIT Bombay Sc...
CSE Webmail:...
Online C++ Co...
Week_13 - Goo...
Week_12 - Goo...
ssd13.polosol...
Download mu...
Club Mahindra
C++ document...
Other bookmarks
Cook ID
14
Cook Name
Kiteretsu
Add
Cooks List for Reference
ID
First Name
Last Name
16
Surya
Damle
14
Kiteretsu
Tarte
15
Saloni
Bharadwaj
13
Mrunmayee
Deshpande
17
Saukhya
Tripathi


Activities
Google Chrome
May 7 10:14
localhost:3000/employee
Update
Apps
Bookmarks
IIT Bombay Sc...
CSE Webmail:...
Online C++ Co...
Week_13 - Goo...
Week_12 - Goo...
ssd13.polosol...
Download mu...
Club Mahindra
C++ document...
Other bookmarks
Name
Kiteretsu
Gender
m
Age
51
Experience
7
Salary
150000
Role
chef
My Orders
Order No.
Order ID
Dish
Completed
Button
1
3
spring roll
1
Cook/Serve
2
8
spring roll
1
Cook/Serve
3
12
spring roll
1
Cook/Serve
4
15
spring roll
1
Cook/Serve
5
16
spring roll
1
Cook/Serve
6
17
spring roll
1
Cook/Serve
7
19
spring roll
1
Cook/Serve
8
20
spring roll
1
Cook/Serve
9
22
spring roll
1
Cook/Serve
10
26
samosa
0
Cook/Serve

Input:

The confirm button will confirm placing of the order and the order will be sent to the chefs for further processing.

Output:

After the confirm button is pressed, the dishes in the order will be allocated to different chefs. The output will also logically include the preparation of dishes by the chefs

Test procedure:

The Allot Food Button is selected from the main menu/navigation bar of the website. Screen of this use case will then be loaded only if the login is of the owner else access is denied.

Once the orders are placed by the user, they will be seen in the “Allot Order Left” Table. The orders can be assigned to chefs and we can test if the corresponding tuples are getting added into the grid view in the frontend and (the backend) of the corresponding Chef (Eg: Kiteretsu Here).

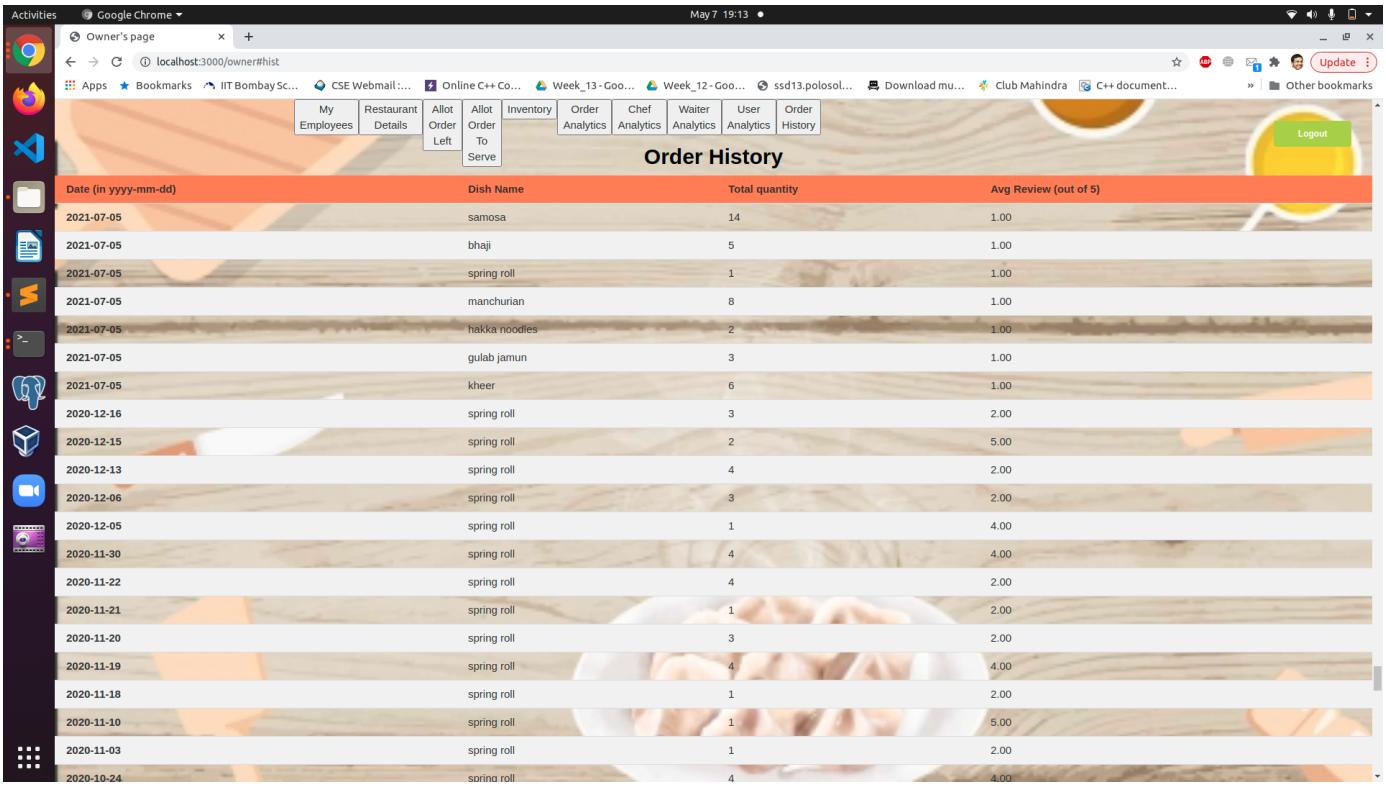
Once the order is assigned to some chef, it should be removed from the allot order table. We can move to the analytics page immediately and test if the cooks count of the concerned chef and other analytics related to the order have been modified accordingly or not. If these conditions are satisfied - Test is successful.

For the backend programmer one additional test can be to go to the database and check if the updates have taken place properly. This is in some ways redundant since grid view has a one-one correspondence with the backend relations.

- Test for Order History: PASSED

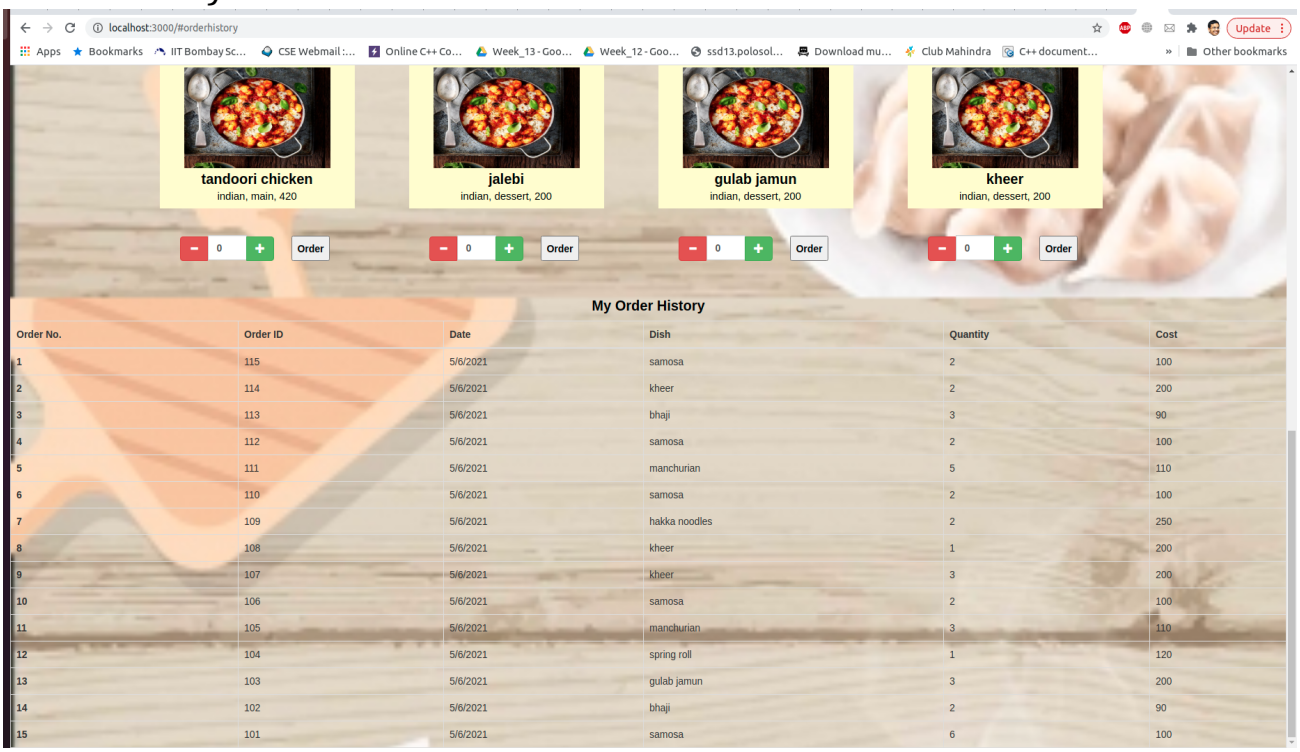
Our Final Screen(s):

Order History for Owner:



Date (in yyyy-mm-dd)	Dish Name	Total quantity	Avg Review (out of 5)
2021-07-05	samosa	14	1.00
2021-07-05	bhaji	5	1.00
2021-07-05	spring roll	1	1.00
2021-07-05	manchurian	8	1.00
2021-07-05	hakka noodles	2	1.00
2021-07-05	gulab jamun	3	1.00
2021-07-05	kheer	6	1.00
2020-12-16	spring roll	3	2.00
2020-12-15	spring roll	2	5.00
2020-12-13	spring roll	4	2.00
2020-12-06	spring roll	3	2.00
2020-12-05	spring roll	1	4.00
2020-11-30	spring roll	4	4.00
2020-11-22	spring roll	4	2.00
2020-11-21	spring roll	1	2.00
2020-11-20	spring roll	3	2.00
2020-11-19	spring roll	4	4.00
2020-11-18	spring roll	1	2.00
2020-11-10	spring roll	1	5.00
2020-11-03	spring roll	1	2.00
2020-10-24	spring roll	4	4.00

Order History for User:



Order No.	Order ID	Date	Dish	Quantity	Cost
1	115	5/6/2021	samosa	2	100
2	114	5/6/2021	kheer	2	200
3	113	5/6/2021	bhaji	3	90
4	112	5/6/2021	samosa	2	100
5	111	5/6/2021	manchurian	5	110
6	110	5/6/2021	samosa	2	100
7	109	5/6/2021	hakka noodles	2	250
8	108	5/6/2021	kheer	1	200
9	107	5/6/2021	kheer	3	200
10	106	5/6/2021	samosa	2	100
11	105	5/6/2021	manchurian	3	110
12	104	5/6/2021	spring roll	1	120
13	103	5/6/2021	gulab jamun	3	200
14	102	5/6/2021	bhaji	2	90
15	101	5/6/2021	samosa	6	100

Input:

Date range and then click on order history.

Output:

Orders placed in that date range sorted from newest to oldest.

Test procedure:

We will add orders placed on different dates in the database with the corresponding attributes such as dish name, quantity, cost, review etc. There are 2 views for this screen:

One for the owner which will show all the orders placed in his/her restaurant and will open only if the credentials are of the owner.

The other is for the user which will show all the previous records placed by that user. Whenever the user logins, his/her corresponding records will be displayed on his/her page.

Also, given the specified date range constraints, only and only those orders that were placed in that interval (start and end date included) should be outputted.

If these conditions are satisfied, the test is successful.

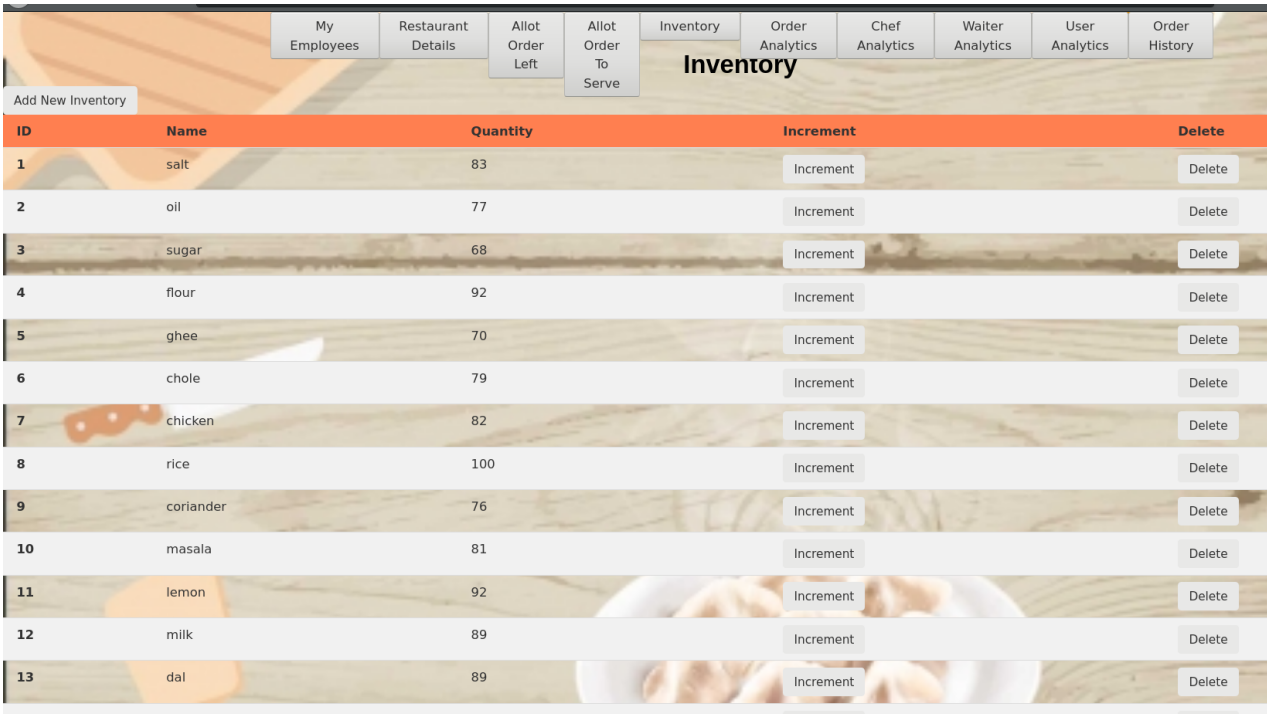
Negative Test:

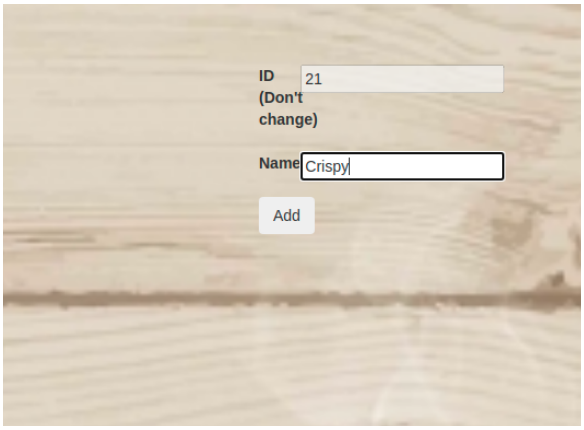
In the “to” and “from” of date range, put the “to” date which comes after “from” date and it should give no output.

- **Test for Update Inventory: PASSED**

Our Final Screen(s):

Input	Output
Add New Inventory	PASSED
Increment	PASSED
Delete	PASSED





Input:

Delete/Update/Create (of CRUD) command. In case of delete the input is just the delete button press. In case of create/update after the button press a form opens up with old values/blank values for update and create respectively. Then the new values are typed in (as shown in screen design in Deliverable 3) and then the confirm button is pressed.

Note: Sample screens for input and output given here in manage employee information - they were also presented in Deliverable 3

Output:

Output is the updated grid view in the page (automatic display of the updated relation without need to reload the page). It will be part of the controller and backend logic.

Test procedure: (which screen to load, what action to take etc.)

The Update Inventory page is selected from the main menu/navigation bar of the website. Screen of this use case will then be loaded only if the login is of the owner else access is denied.

Then all 3 options will be tested. (on all the inventory tables)

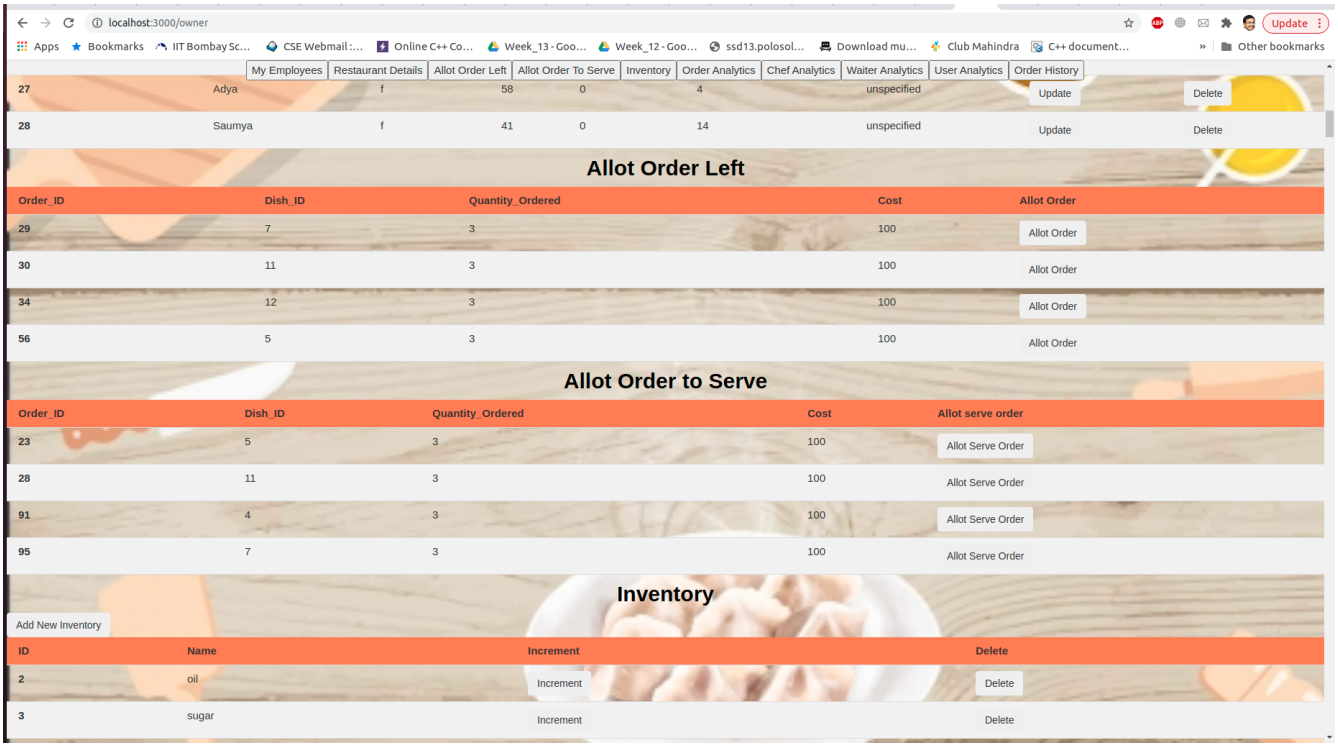
- Create: Add an ingredient/dish with the asked attribute values in the form using the Add new Inventory button.
- Update: Update existing quantity of the corresponding ingredient/dish using increment button.
- Delete: Delete a tuple from a relation shown in the grid view using the delete button.

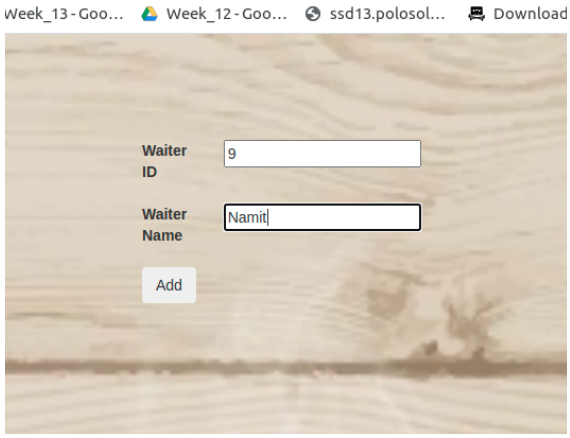
For the backend programmer one additional test can be to go to the database and check if the updates have taken place properly. This is in some ways redundant since grid view has a one-one correspondence with the backend relations.

- **Test for Serving Food: PASSED**

  Our Final Screen(s):

Input	Output
Allot Serve Order	PASSED
Add	PASSED
Cook/Serve	PASSED





Requirement:

After the chef completes making the dishes in the order and marks it we have to serve it to the customers. So a mapping for order->waiter has to be made which is done through this use case.

Use case:

The Serving Food Use Case (Use Case No. 8)

Input:

Input firstly is the serve button of a particular row in the (pending) orders relation's grid view shown. which will open up a form with a dropdown to choose a waiter. Then the confirm button.

Output:

After the confirm button is pressed the corresponding order is no longer pending and so it is deleted from the orders relation. (removed from grid view in the frontend) - Automatic display of the updated relation without need to reload the page). It will be part of the controller and backend logic.

Not only an output on the same page but the statistics in the analytics page is also updated. Here the concerned statistic is the number of orders served by the particular waiter.

Test procedure: (which screen to load, what action to take etc.)

The Allot Serve Order Button is selected from the main menu/navigation bar of the website. Screen of this use case will then be loaded only if the login is of the owner. Otherwise access is denied.

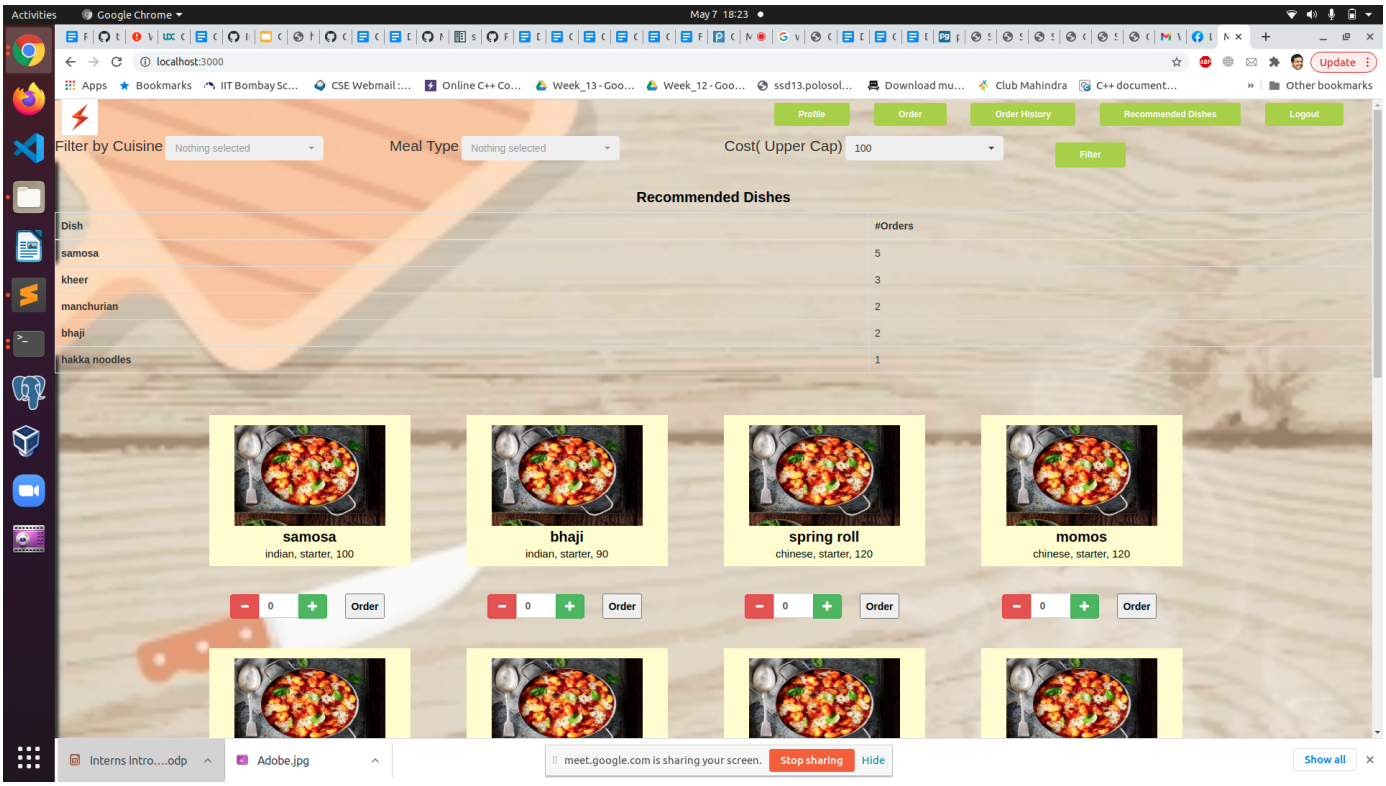
Once the order is cooked by the chef, it will be seen in the “Allot Order to Serve” Table. The orders can then be assigned to waiters and we can test if the corresponding tuples are getting added into the grid view in the frontend and (the backend) of the corresponding Waiter (Eg: Namit Here).

Once the order is assigned to some Waiter, it should be removed from the allot order to serve table. We can move to the analytics page immediately and test if the serve count of the concerned waiter and other analytics related to the order have been modified accordingly or not. If these conditions are satisfied - Test is successful.

For the backend programmer one additional test can be to go to the database and check if the updates have taken place properly. This is in some ways redundant since grid view has a one-one correspondence with the backend relations.

- **Automation: PASSED**

Our Final Screen(s):



Requirement:

The dish recommendation part is quite common in today's E-Commerce/food delivery apps which helps users to decide faster and have a better experience by ordering the best quality & most loved products. There are two recommendations - most loved, your personal favorite. The other part we're automating is quantity flag (flagging if quantity of any particular ingredient is below a certain threshold)so that the owner is always aware of what needs to be ordered without having to constantly keep checking the inventory.

Input:

Since this is automated there's no input from the user side. There will be sections on the user and admin interface where based on the entries in the database the recommendations will be added.

Output:

Most loved dishes will include the most ordered dishes (in the advanced version, we

can set the metric of most loved to include - ratings as well as number of times people ordered it (loyalty)) sorted to include top k dishes. And user favorite dishes will include the ones he/she has ordered most frequently. The output for quantity automation will be: “These ingredients need to be ordered asap” and will be followed by a list of ingredients. If the list is empty the “display” of the entire text will be “hidden”.

Test procedure:

Place a few orders from different users’ accounts and verify if they’re being recommended the right dishes. Print the output of the queries for most loved and personal favorite dishes to see if the counting is being done correctly. Based on the orders we’ll check if the quantity of ingredients is being updated properly.

Negative Tests

- **Invalid Order**

  Requirement:

  The system must be able to handle invalid inputs and searches without crashing, and should direct the user appropriately to perform corrective action.

  Use case:

  The Place and Order Use Case (Use Case No. 2)

  Input:

  The input is either in the form of selection of dishes/categories from a drop down list or plaintext input from the user in a text box. The former ensures no input error, as we have valid options. However the latter is completely up to the user, so must be handled.

  Output:

  If the specific combination that the selection of filters from the drop-down list yields is non-existent, then the system must indicate empty and provide the user with easy access to resubmit the dish selection options. If the user enters something altogether invalid, then the system must indicate as such without going to the back end, and request the user to retry.

  Test procedure:

  Same as the test for place an order (test 2)

- **The login credentials of the owner or manager or the concerned user is incorrect**

  Requirement:

The system must be able to handle invalid inputs without crashing, and should direct the user (owner/manager/customer etc) appropriately to perform corrective action.

Use cases: (for various users)

The Place and Order Use Case (Use Case No. 2),

The Manage Employee Information Use Case (Use Case No. 3),
The Update Inventory Use Case (Use Case No. 7)

Input:

The input is either in the form of a username and a password, or userID and password.

Output:

The concerned user should be unable to progress in the case where something invalid is entered. The system should inform the user of his/her error and prompt the user to retry.

Test procedure:

This will be tested on the screens where a login is required (i.e, before placing an order for a user, or when the owner logs in (the screen before he/she gets the options to manage employees, update inventory etc.

Error Handling

It is done through **console.logs** which is sufficient for our project. The user can also see such logs in the inspect element in the page. The website does not crash on such errors - it just stalls. Then if the user reloads and does something else which is error free, the website runs normally.

This is ensured since in every place in our controller we have caught all the errors and output them to the console using console.log. An alternate which could be used is output to a .log file - which can be replaced in the .catch lines in the code.

e. **Conclusions on how much of the requirements you could complete and why.**

Original features implemented

After login:

The customer shall be able to:

- ✓ **View his profile**
- ✓ **View recommended orders for him based on dishes ordered most times in the past by him**
- ✓ **Filter the menu of the restaurant categorically (cuisine, starter-main-dessert).**
- ✓ **View the cost and image of each dish.**
- ✓ **Then, he/she will select, one-by-one, the dishes desired (can select quantity)**
- ✓ **View his order history**
- ❏ Finally, a payment option will be present. All of these details comprise the order. Upon completion, the order will be recorded in the order history.
- ✓ **Logout**

The owner/administrator shall be able to:

- ✓ **View all the inventory (the pantry) that the restaurant has, update it, delete it or add a new ingredient**
- ✓ **Time Series Analytics 1: View all the dishes ordered on a date range specification. Owner can select start and end date (Order History)**
- ✓ **View Employee details (salary, name, ID, age etc.). Owner can add new employees, update employee information or delete employee from the relation**
- ✓ **Allot an order to a chef.**
- ✓ **Allot the delivery of a prepared order to a waiter.**
- ✓ **View analytics : popular dishes, waiters and chefs the with most completed orders, most loyal customers in both table and graph form**
- ✓ **Time Series Analytics 2: Get profit, expenditure, wastage of day-to-day basis based on filtering on start date and end date given by the owner**
- ✓ **Statistical Report: Owner can see a different page opens with statistical information: mean, standard deviation and day with maximum and minimum wastage/profit/expenditure**
- ✓ **Logout**

The Employee shall be able to:

- ✓ **View the orders allotted to them by the owner and fulfill those: if cook he can mark cooked and if a waiter he can marks it served**
- ✓ **View his profile**

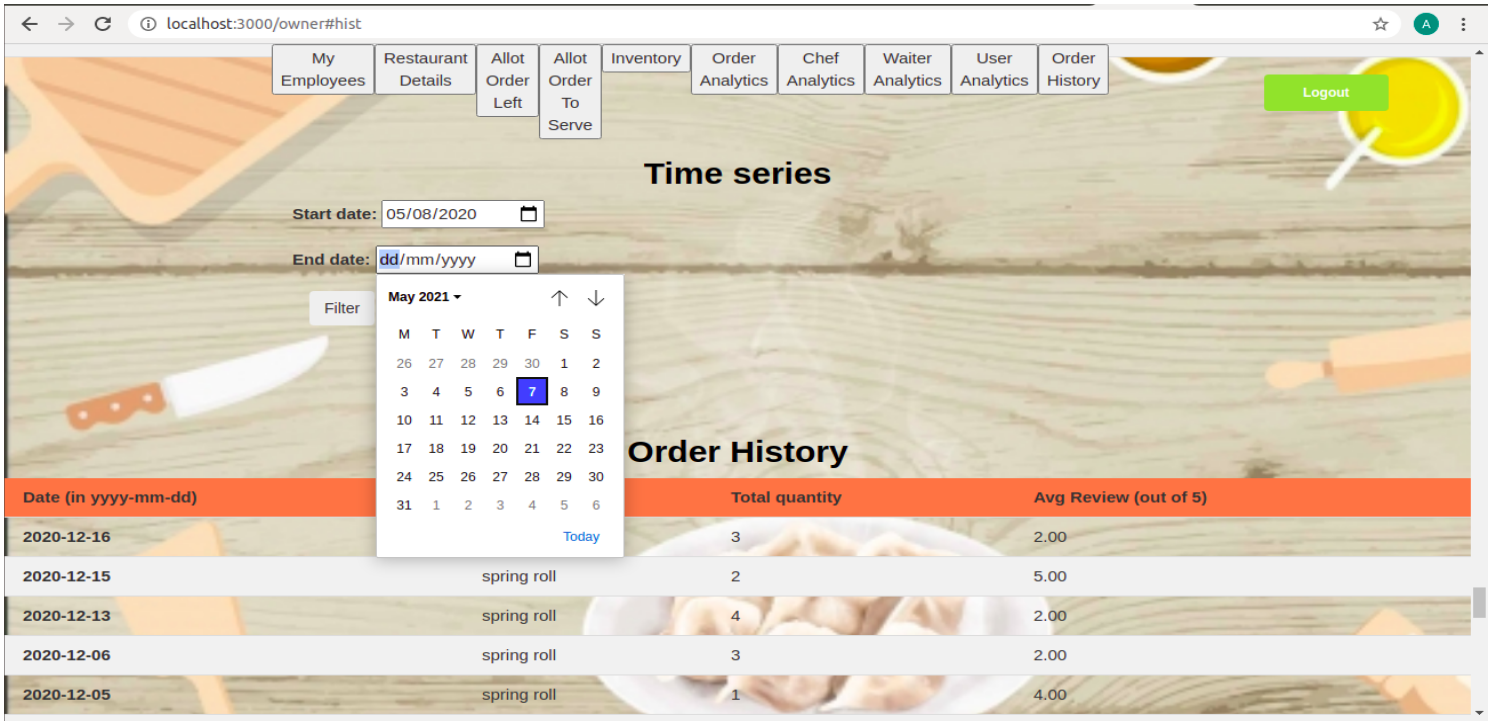
- ✓ **See the number and total cost of orders they've fulfilled in the past**
- ✓ **Logout**

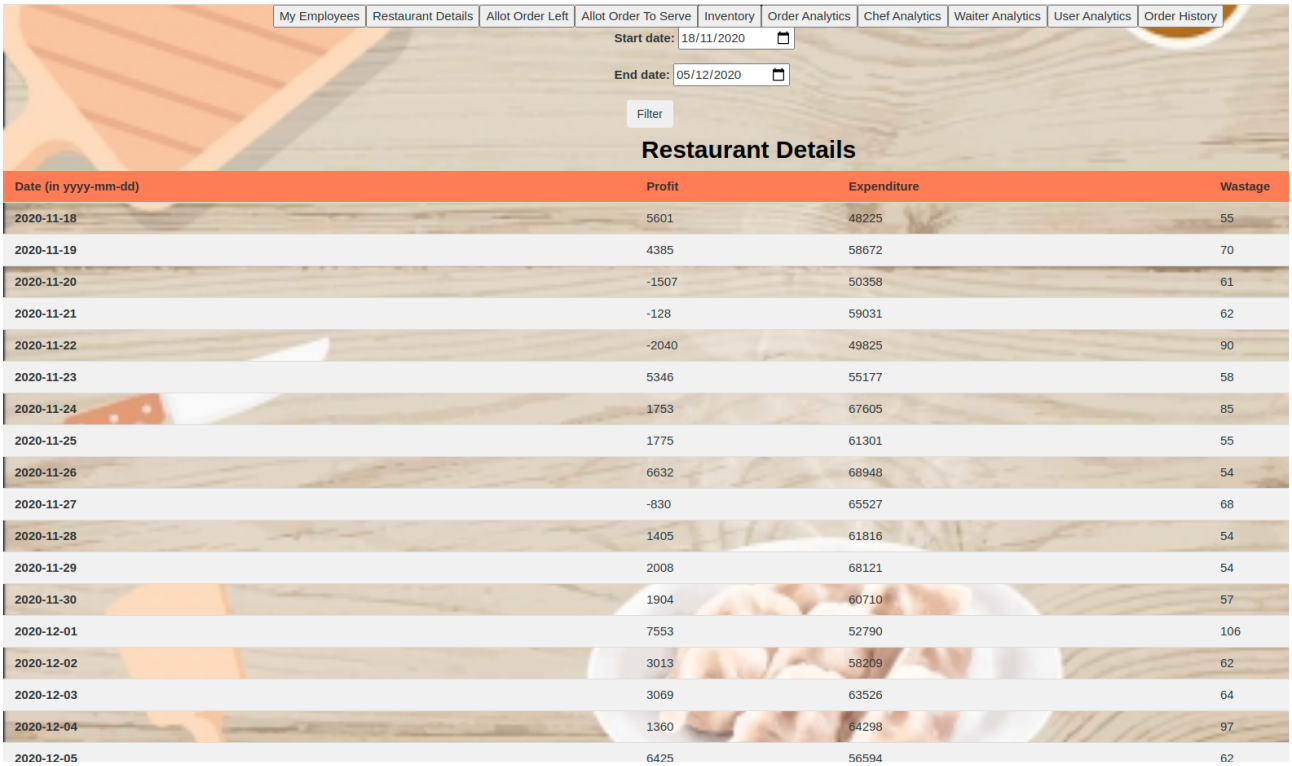
Why we could not complete some requirements:

-> The payment feature was not implemented. In our implementation: the order is placed, then it is allotted to a chef, then it is 'cooked', then it is allotted to a waiter for delivery. The order is added to the history of placement. And hence, there is no logical place at the end of this process for payment, it would be irrelevant. If the addition to the order history had been at the end, it would have made sense to complete the payment. It is also difficult to insert the payment process in between the dish selection/order and the update to the order history, at an implementation level.

Extra features added after feedback

- ✓ **Time series analytics:**
  - Owners can see dishes ordered on each day with their total quantity ordered and average review on that day. This can also be filtered using a date range.
  - Owner can see the wastage, expenditure and profits in a given date range.





Date (in yyyy-mm-dd)	Profit	Expenditure	Wastage
2020-11-18	5601	48225	55
2020-11-19	4385	58672	70
2020-11-20	-1507	50358	61
2020-11-21	-128	59031	62
2020-11-22	-2040	49825	90
2020-11-23	5346	55177	58
2020-11-24	1753	67605	85
2020-11-25	1775	61301	55
2020-11-26	6632	68948	54
2020-11-27	-830	65527	68
2020-11-28	1405	61816	54
2020-11-29	2008	68121	54
2020-11-30	1904	60710	57
2020-12-01	7553	52790	106
2020-12-02	3013	58209	62
2020-12-03	3069	63526	64
2020-12-04	1360	64298	97
2020-12-05	6425	56594	62

✓ **Statistical Report of Expenditure, Profit, Wastage:**
A button with the report page link is clicked and an analysis report is shown which has information about average and standard deviation of expenditure, price and wastage and days with maximum and minimum of each.

✓ **Atomicity Assurance:**
The way we have designed our add, updates and deletes button ensures that the atomicity constraints are not violated. Till the point when no click on any button is done, no changes will be made in the database. Also, is such that at any particular instant of time, given a particular button, only 1 click can be successfully executed even if that same button is being clicked from 100 devices at nearly the same time. And the update to the database is made immediately after the corresponding button is clicked (i.e. before any other click on any other button and before any other action on the system). This will ensure that once some action has started executing, the other action can take place only after the current action has either been successfully completed or there is some error in it. If there is some error, that corresponding action will be rolled back and the database will not be changed due to this action. Thus, this flow ensures that atomicity constraints are satisfied and taken care of in our system.

✓ **Security:**
The security of the application is an important addition we have made. Through the use of prepared statements, SQL injections in queries to our database are prevented. The login system has been made more robust by the use of cookies. After a logout, a simple 'back' (to go to the previous page) will not allow an entry to the system, as the concerned cookies are being cleared.

✓ **Pdf:**
We have designed a feature to download some selected analytics in pdf format for ease of view.

f. Link to the github repository containing the code

https://github.com/ParthLa/CS387_Project

PPT:
https://docs.google.com/presentation/d/1OmmtPuc1QhljApG6R6h4JzRuNLK8QPBWMgNGr4RCr-o/edit#slide=id.p

Mistakes rectified from previous deliverables:

1. Convert foreign keys in schema from on delete set null to on delete cascade since in most cases this is ideal and error free. For example tracks.id references staff.id and on deleting an employee by the owner in his screen on delete set null sets tracks.id to null which is undesirable for 2 reasons:
    - tracks.id does not make sense without staff.id
    - tracks.id is a primary key and cannot be null

2. Women Nobel Laureates graph removed.