CS 432 - DATABASES | ASSIGNMENT 4

MESS MANAGEMENT - DEPLOYING THE DBMS

April 15, 2023

GROUP MEMBERS

DHRUV DARDA | 19110012

MUHAMMAD YUSUF HASSAN | 19110020

HARSHIT RAMOLIA | 19110024

PATEL AGAM | 19110038

SHANTANU SAHU | 19110100

V P SHIVASANKARAN | 19110104

PATEL RAJAN GIRISHBHAI | 19110129

INSHA MANSURI | 19110182

MD AMIR SOHAIL | 19110188

PULKIT JAIN | 19110196

SOMESH PRATAP SINGH | 19110206

PRIYA GUPTA | 20110147

COURSE INSTRUCTOR:

PROF. MAYANK SINGH

Github Link: https://github.com/Harshit-Ramolia/Mess-managment-frontend

1. Responsibility of G1

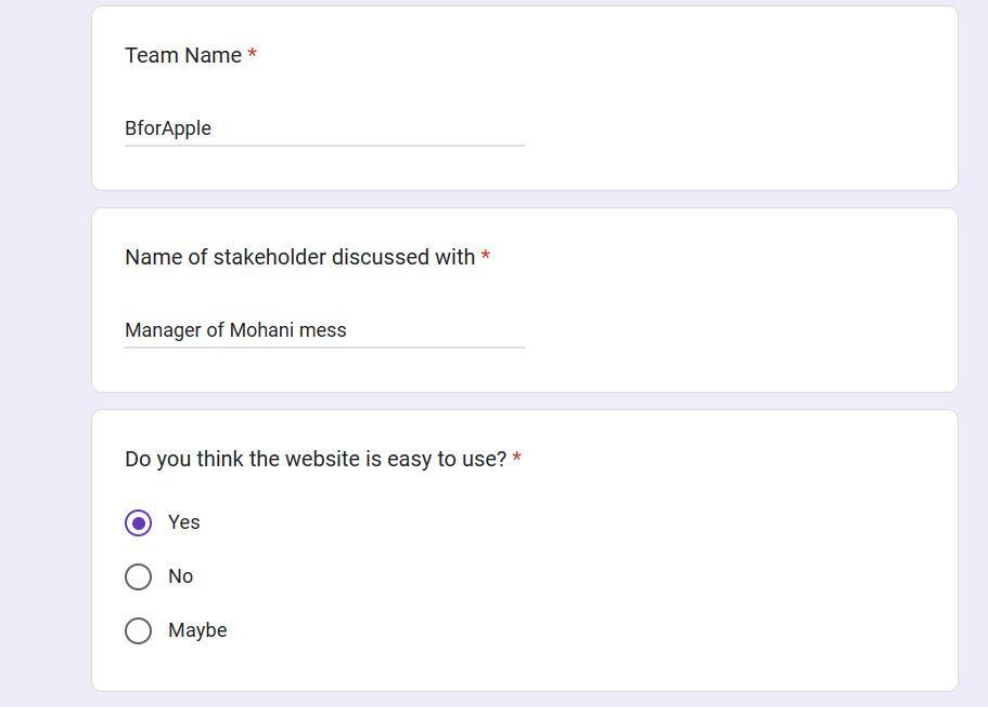
Note: All the screenshots after different levels of feedback are attached in response to Question 2 of Section 3.2

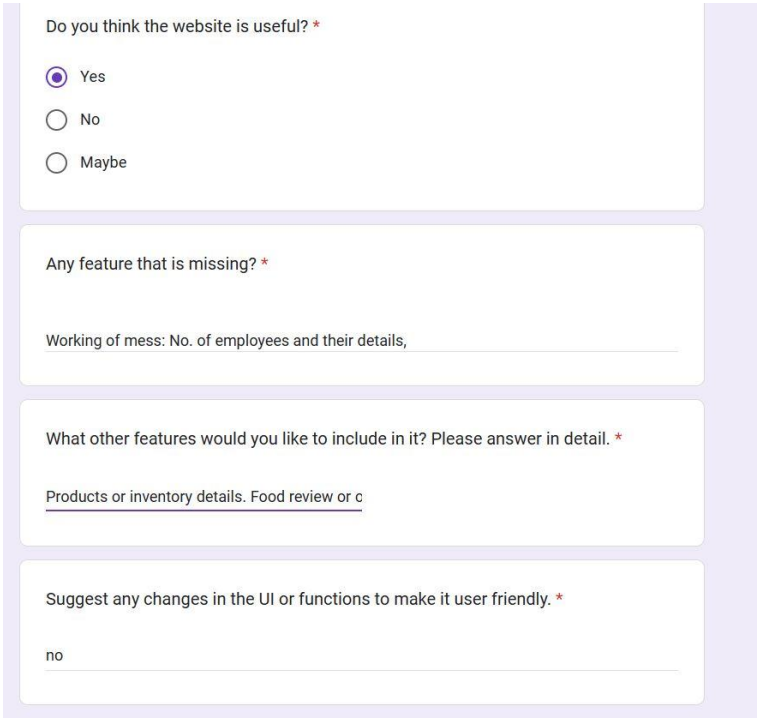
1.1 Feedback from stakeholders:

The G1 takes two feedbacks from the stakeholders, initial feedback and final feedback after making relevant changes as suggested in the first feedback

Initial feedback was taken from mess manager, mess supervisor, and two students of mohani mess.

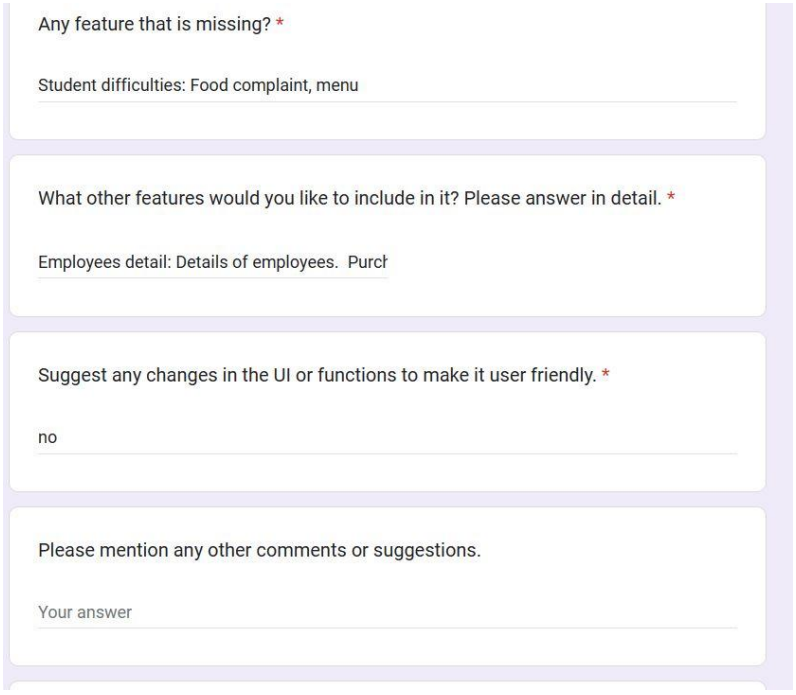
Screenshot of initial feedback from stakeholder 1 [manager of Mohani mess (Mr. Suresh bhai)]:
Note: We talked to the Mess manager and then based on the interaction with him we filled the feedback forms that were submitted.





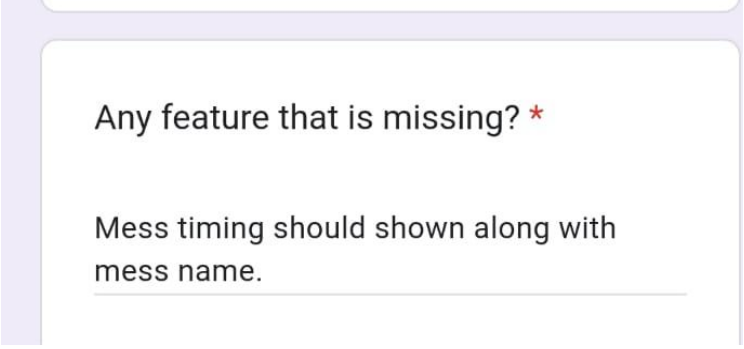
Summary of the feedback received from mess manager: The mess manager wants to see the worker details who are working in their mess. He also wants to see details about inventory, and student feedback about each meal.

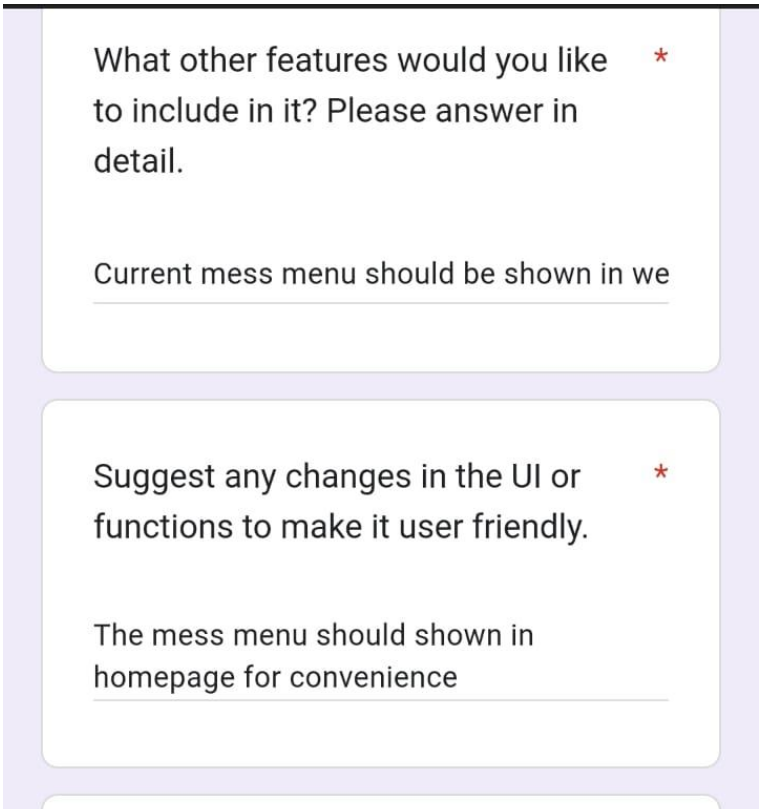
Screenshot of initial feedback of stakeholder 1 [supervisor of Mohani mess (Mr. Parvat singh)]:
Note: We talked to the Mess supervisor and then based on the interaction with him we filled the feedback forms that were submitted.



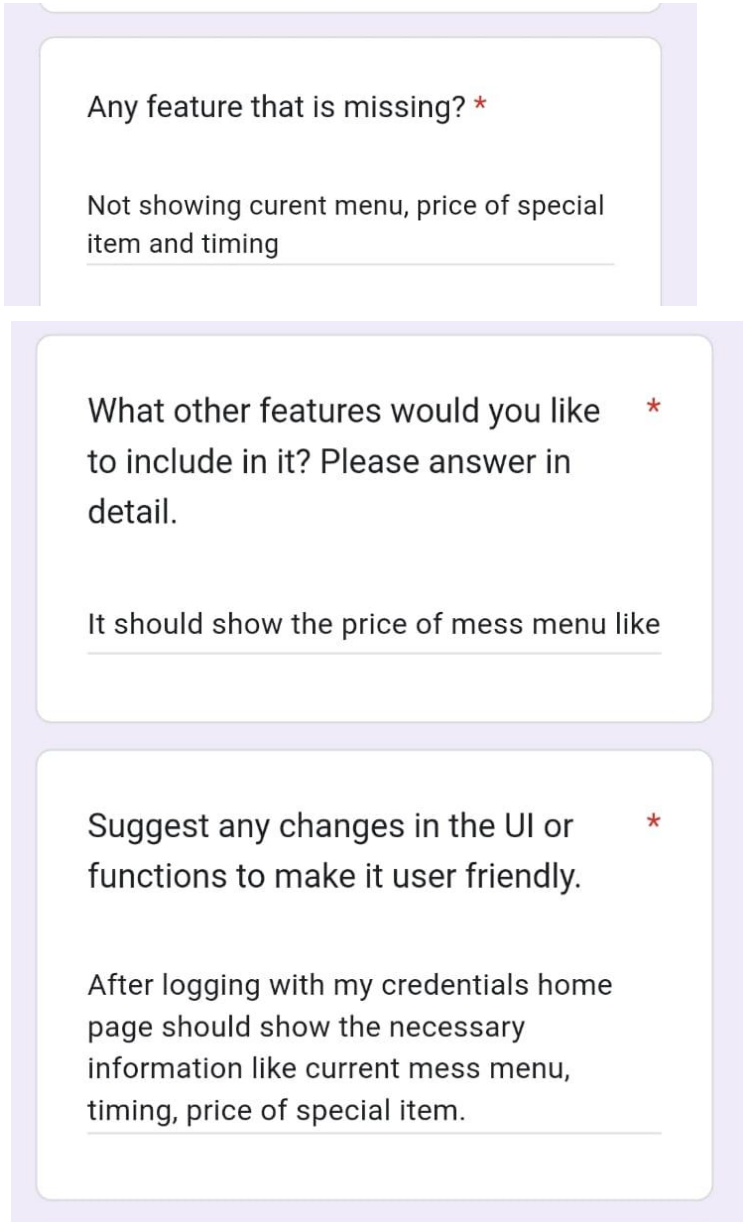
Summary of the feedback received from mess manager: Supervisor wants to see purchase details and employee details on the website.

Screenshot of first feedback of stakeholder 3 & 4 [Student's feedback].
First student feedback





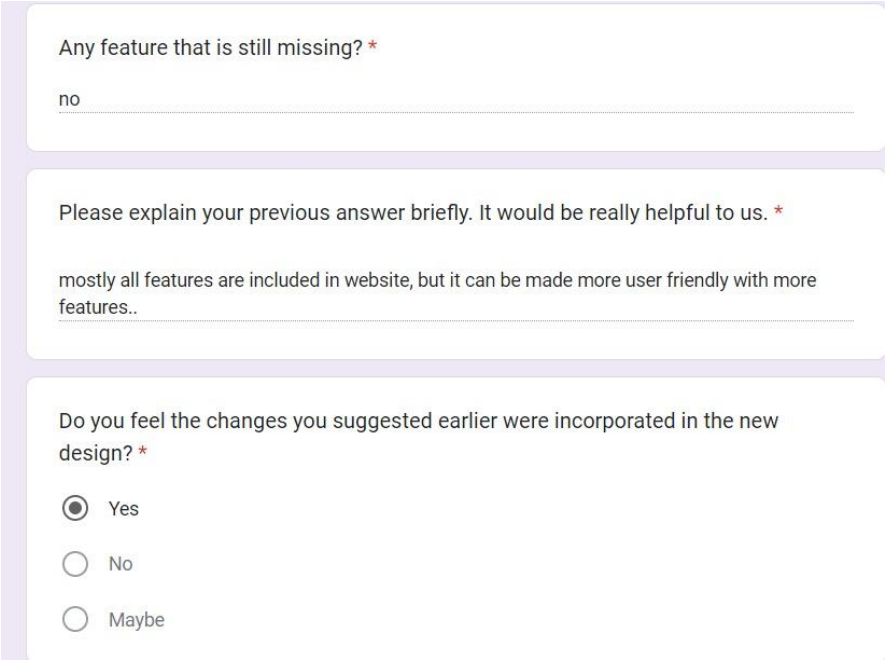
Second student's feedback



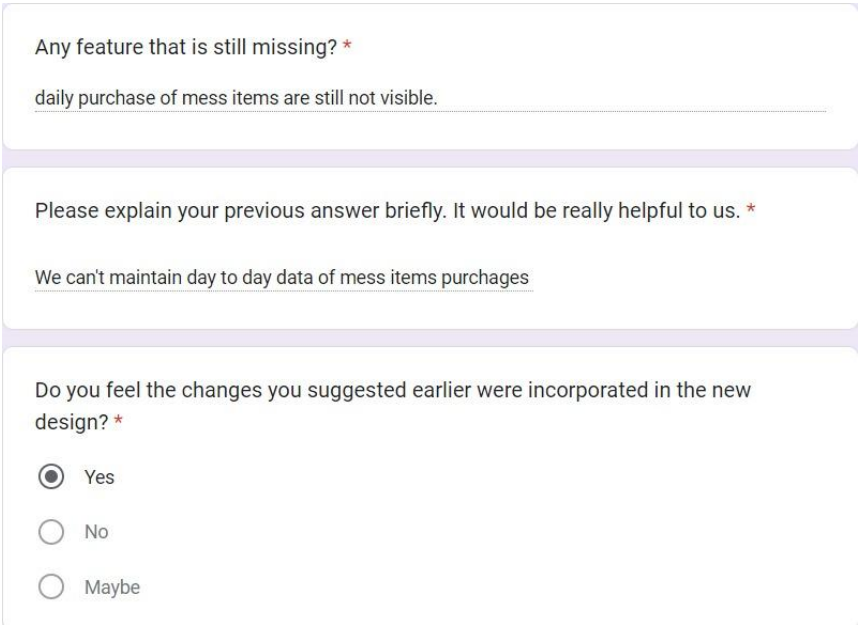
Summary of the feedback received from mess manager: Students of that mess want to see the mess opening and closing timing for every meal along with other mess details. They also want to see the current mess menu along with the price of special item that is served by the mess on a paid basis.

Final feedback was taken from mess manager, mess supervisor, and two students of mohani mess.

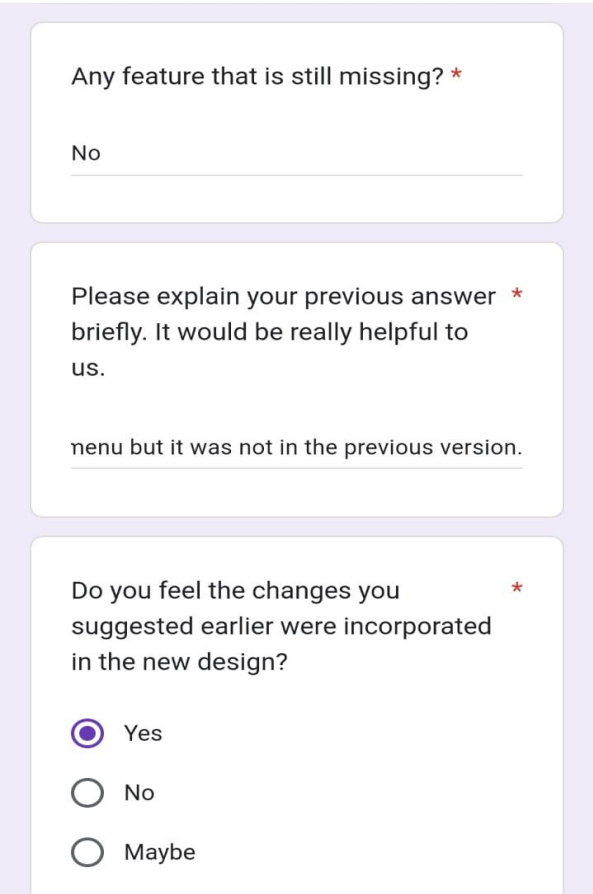
Screenshot of second feedback from stakeholders [Manager of mohani mess(Mr. Suresh bhai)].



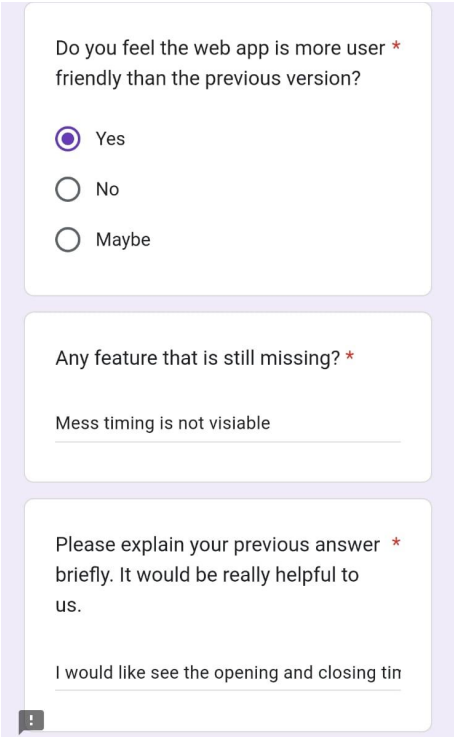
As per the interaction with the mess manager, all his requirements are almost full filled but the User Interface can be done in a better way according to him.
Screenshot of second feedback of stakeholder [Supervisor of Mohani mess (Mr. Parvat singh)].



Supervisor’s feedback: The daily purchase detail of the mess item is not visible on the website.
Screenshot of second feedback from stakeholders [Student’s feedback].
First student’s feedback

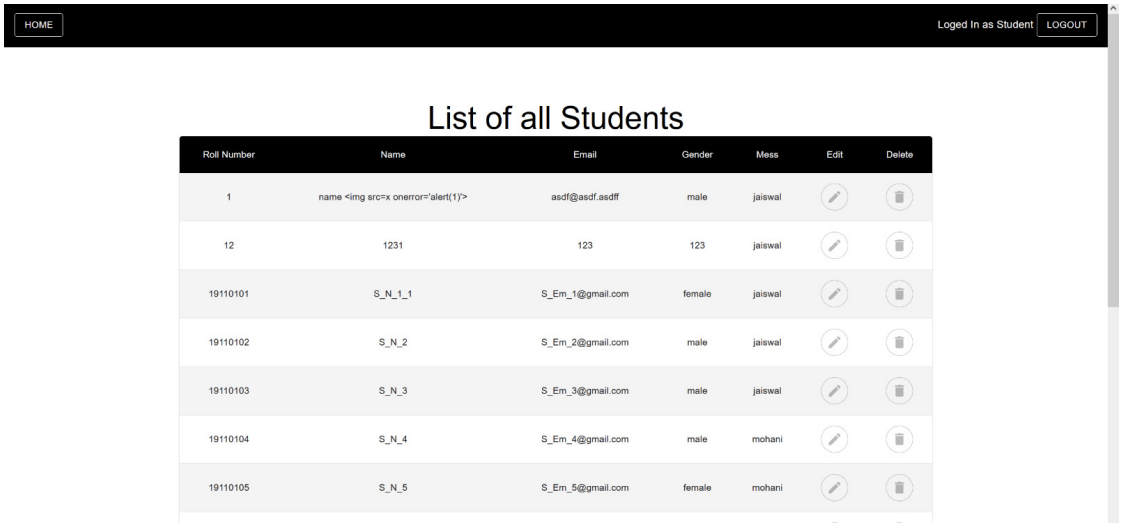


Second student's feedback

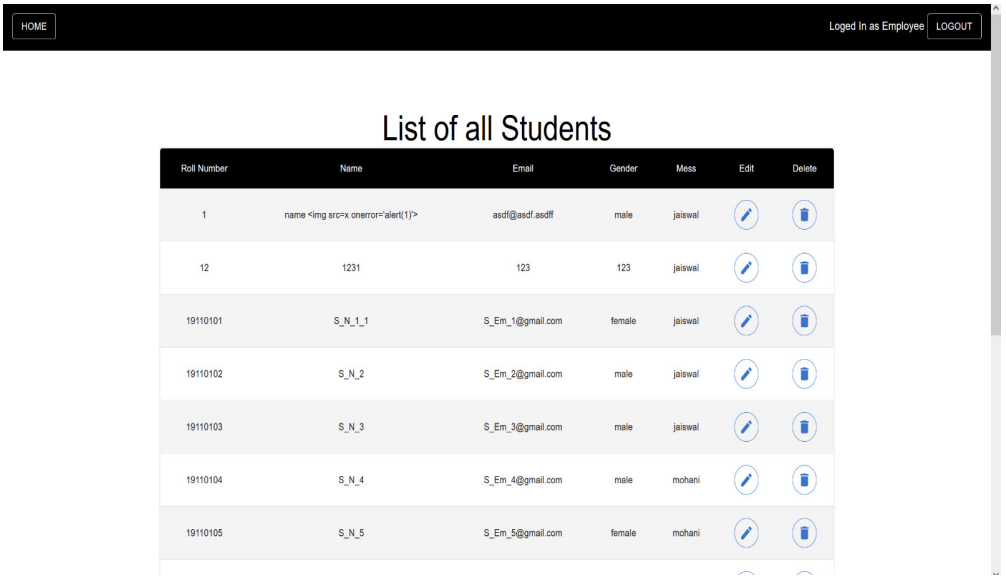


1.2 Screenshots of different views:

Student view:



Employee view:

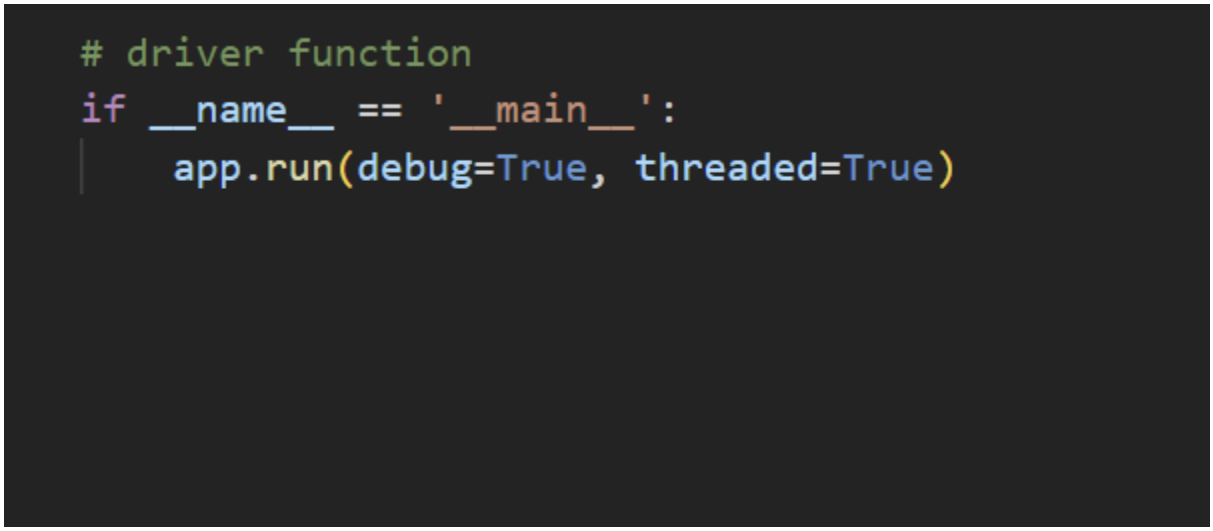


Writeup on privileges: The students do not have admin rights and can only view list of messes, students, student representatives, mess menu, wastage details, feedback and their attendance while the manager and the mess council with admin rights can view and edit the above mentioned pages along with access to other pages like inventory details, balance sheet, stock details, etc.

2. Responsibility of G2

2.1 Concurrent multi-user access:

Flask by default doesn't have multithreading option, but we can use it if we wants to



When multiple users concurrently access a Flask app, there is a risk of data inconsistency if two or more users attempt to modify the same data at the same time. To avoid this issue, we used locking mechanisms to prevent concurrent modification of data by multiple users. Locking allows a process or thread to acquire exclusive access to a resource or data, preventing other processes or threads from accessing or modifying the same resource at the same time. In our case we used Database level locking to provide concurrent modification of data.

Database-level locking involves applying locks at the database level to prevent multiple users from modifying the same data simultaneously. We can apply table-level locks to prevent concurrent updates to a table. By using locks, only one user can modify the table at a time, ensuring that data consistency is maintained.

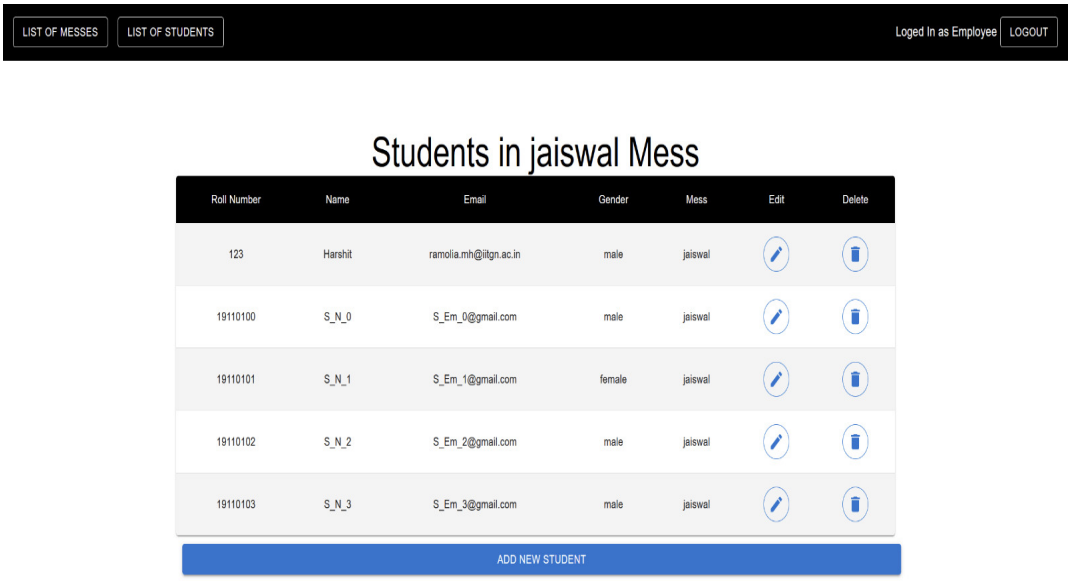
```python
class student_update(Resource):
    # http://127.0.0.1:5000/api/students/update?roll_no=19110104&name=shiva&mess_id=3&email=vp.shivasan@iitgn.ac.in&password=fluffy&gender=male
    # Sample post request object
    #  myobj = {'roll_no': 19110104,
    #                 'name' : "my_newName",
    #                 'email' : "joblessmf@unknown.com",
    #                 'mess_id':2,
    #                 'password':"new_password",
    #                 'gender' : 'male'
    #                 }
    def post(self):
        lock.acquire()
        print("Student Update")
        print(request.json)

        roll_no = request.json.get('Roll Number')
        mess_id = request.json.get('Mess_id')
        name = request.json.get('Name')
        email = request.json.get('Email')
        password = request.json.get('password')
        gender = request.json.get('Gender')
        cursor = mysql.connection.cursor()
        print("Roll no:", roll_no)
        cursor.execute(
            "select roll_number,name, email, gender,mess_id,password from `student_allocated` where roll_number=%s", [roll_no])
        old_student_data = cursor.fetchone()
        _, old_name, old_email, old_gender, old_mess_id, old_password = old_student_data

        name = old_name if name == None else name
        email = old_email if email == None else email
        gender = old_gender if gender == None else gender
        mess_id = old_mess_id if mess_id == None else mess_id
        password = old_password if password == None else password

        cmd = 'update student_allocated set mess_id={},name="{}",email="{}",password="{}",gender="{}" where roll_number=%s'.format(
            mess_id, name, email, password, gender)
        output = cursor.execute(cmd, [roll_no])
        mysql.connection.commit()
        lock.release()
        return("Updated")
# another resource to calculate the square of a number
```

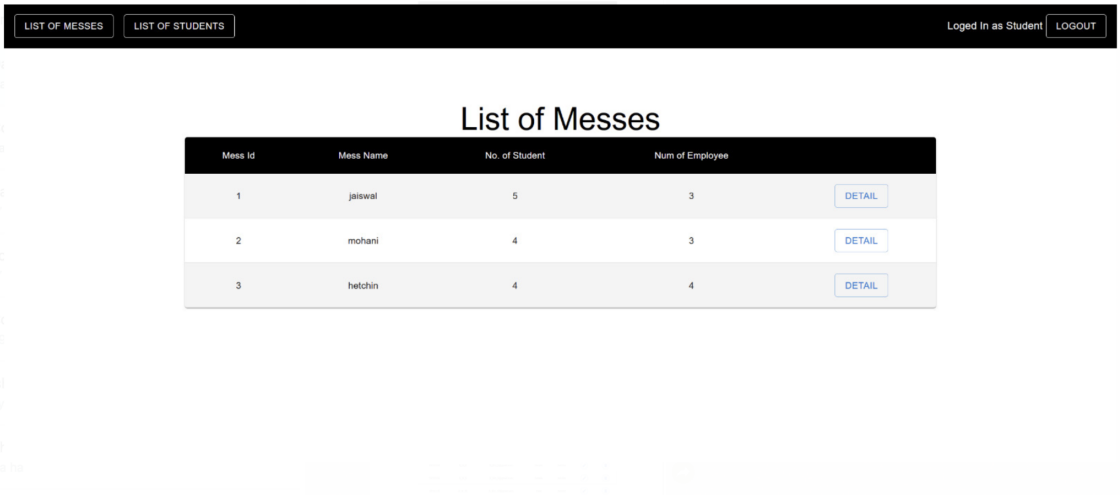
2.2 Implemented changes in the database as per the feedback received from stakeholders:

All the changes that were suggested by the stakeholders were attempted and completed that were within the scope of the course and student level of web-development skills. For example, earlier we had a basic application showing the major information only, now after incorporating two levels of feedback we have all the information showing up on the website. In addition to various tables being added, we have also improved on the login/logout functionality.
Given below are the screenshots of our Application/Database after various levels of feedback from various stakeholders.

Original application before any feedback

Pretty rudimentary website showing just list of students and list of messes with a basic login functionality





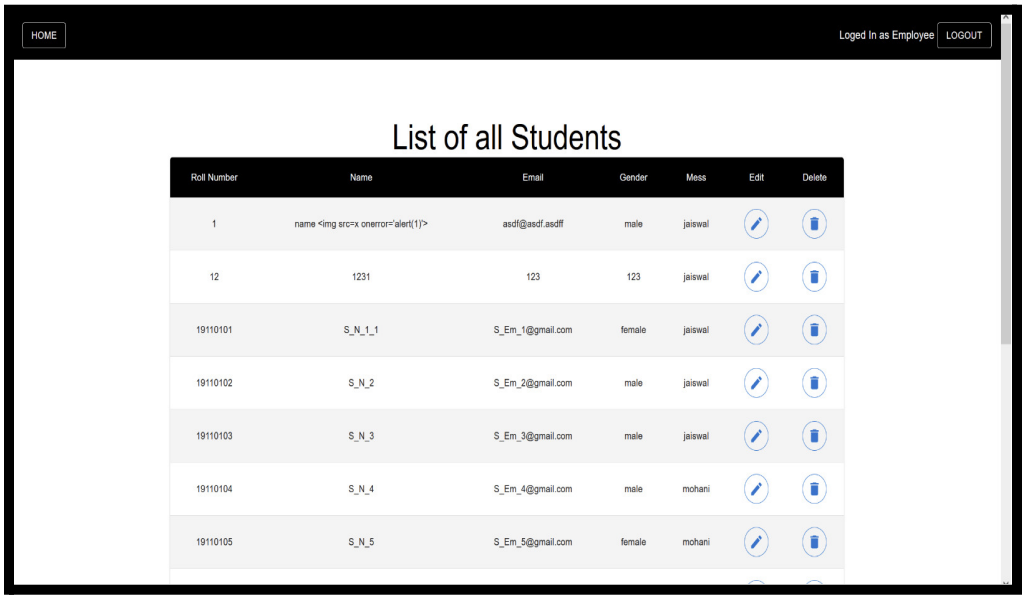
After first feedback

Functionalities added:

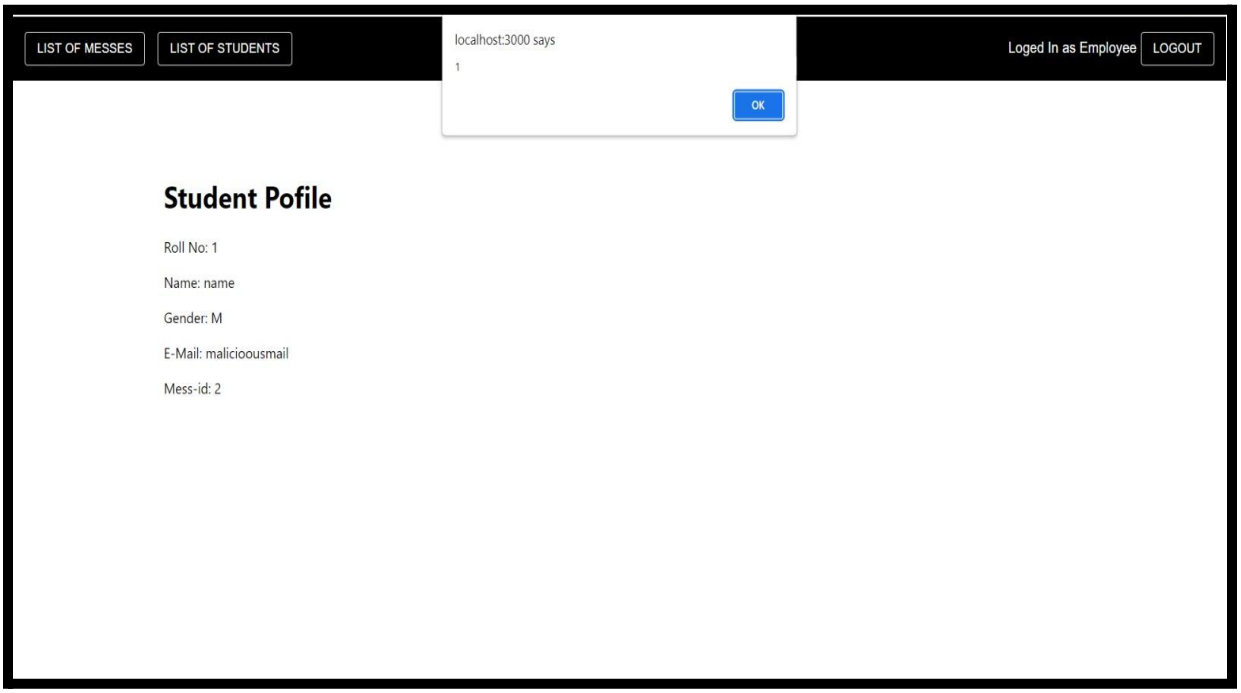
Screen shot of tables are added below with others.

After final feedback

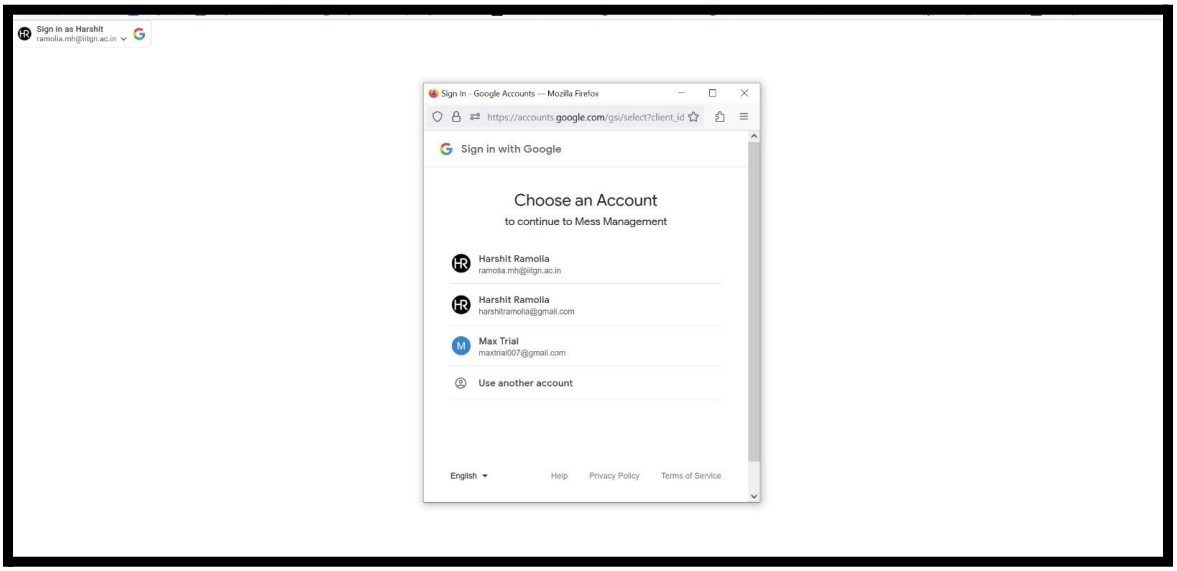
Functionalities added:



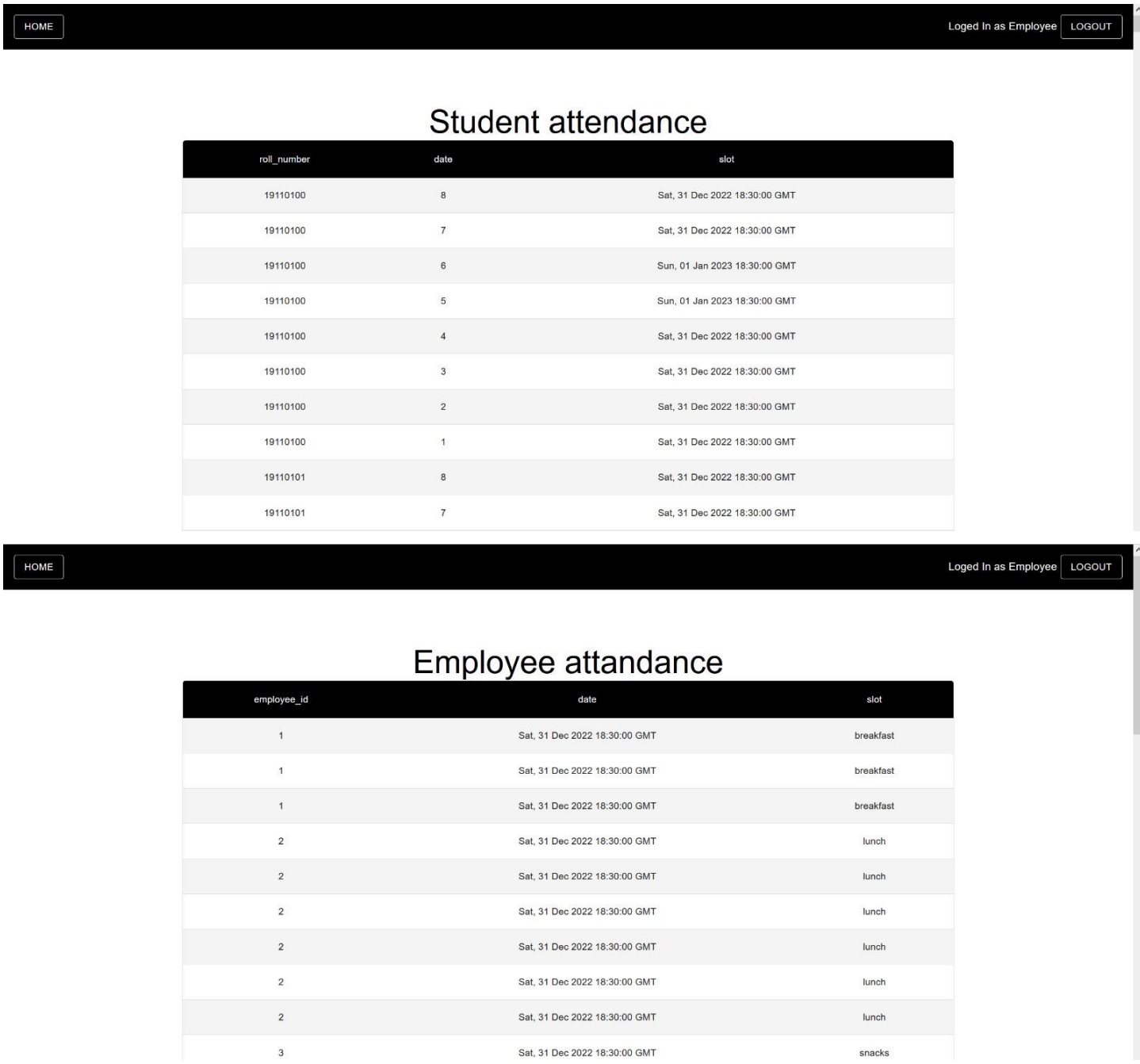
Employee view added



Profile Page



Better Login/Logout


HOME
Loged In as Employee
LOGOUT
Student attendance
roll_number
date
slot
19110100 8 Sat, 31 Dec 2022 18:30:00 GMT
19110100 7 Sat, 31 Dec 2022 18:30:00 GMT
19110100 6 Sun, 01 Jan 2023 18:30:00 GMT
19110100 5 Sun, 01 Jan 2023 18:30:00 GMT
19110100 4 Sat, 31 Dec 2022 18:30:00 GMT
19110100 3 Sat, 31 Dec 2022 18:30:00 GMT
19110100 2 Sat, 31 Dec 2022 18:30:00 GMT
19110100 1 Sat, 31 Dec 2022 18:30:00 GMT
19110101 8 Sat, 31 Dec 2022 18:30:00 GMT
19110101 7 Sat, 31 Dec 2022 18:30:00 GMT

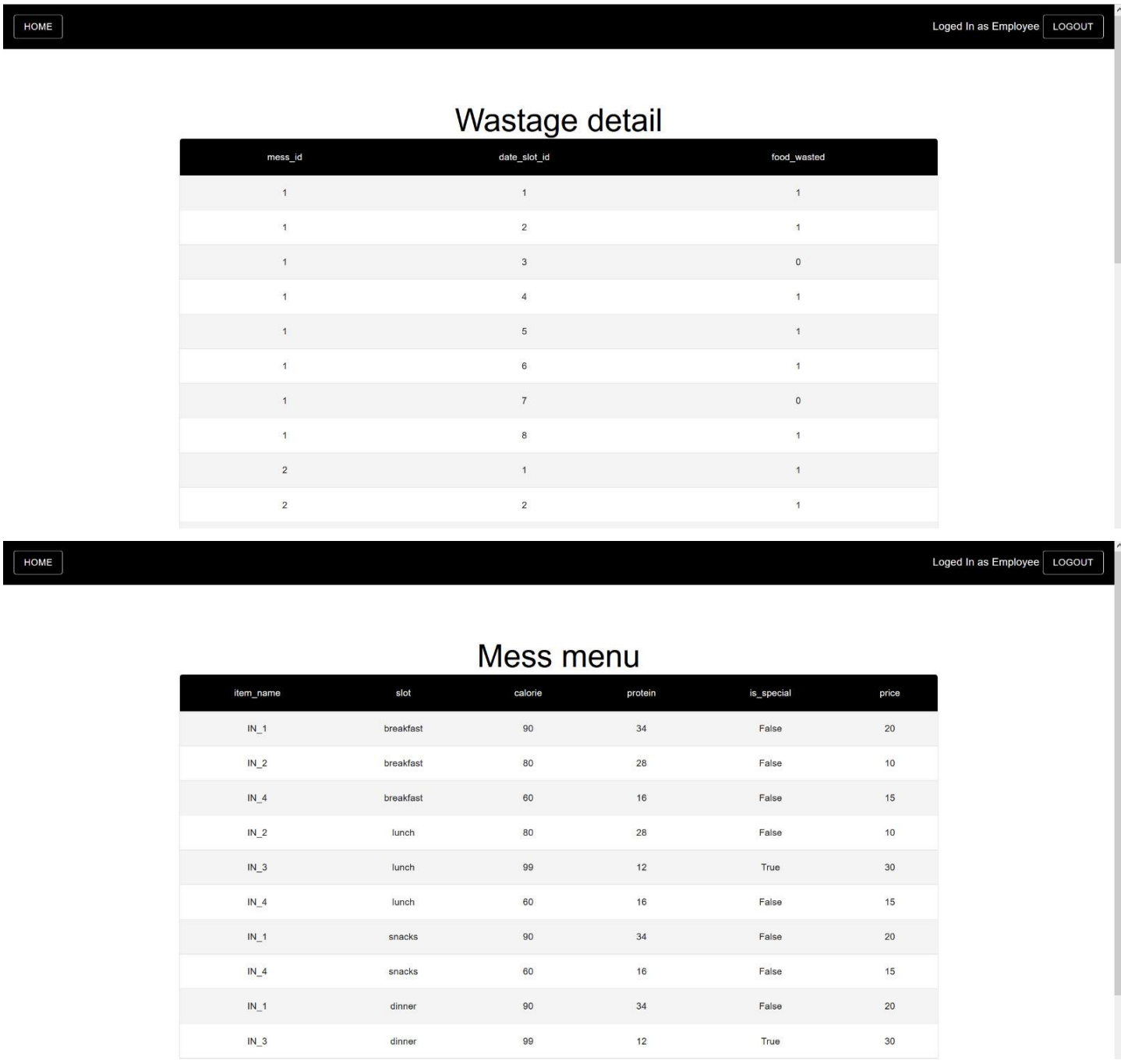
HOME
Loged In as Employee
LOGOUT
Employee attandance
employee_id
date
slot
1 Sat, 31 Dec 2022 18:30:00 GMT breakfast
1 Sat, 31 Dec 2022 18:30:00 GMT breakfast
1 Sat, 31 Dec 2022 18:30:00 GMT breakfast
2 Sat, 31 Dec 2022 18:30:00 GMT lunch
2 Sat, 31 Dec 2022 18:30:00 GMT lunch
2 Sat, 31 Dec 2022 18:30:00 GMT lunch
2 Sat, 31 Dec 2022 18:30:00 GMT lunch
2 Sat, 31 Dec 2022 18:30:00 GMT lunch
2 Sat, 31 Dec 2022 18:30:00 GMT lunch
3 Sat, 31 Dec 2022 18:30:00 GMT snacks

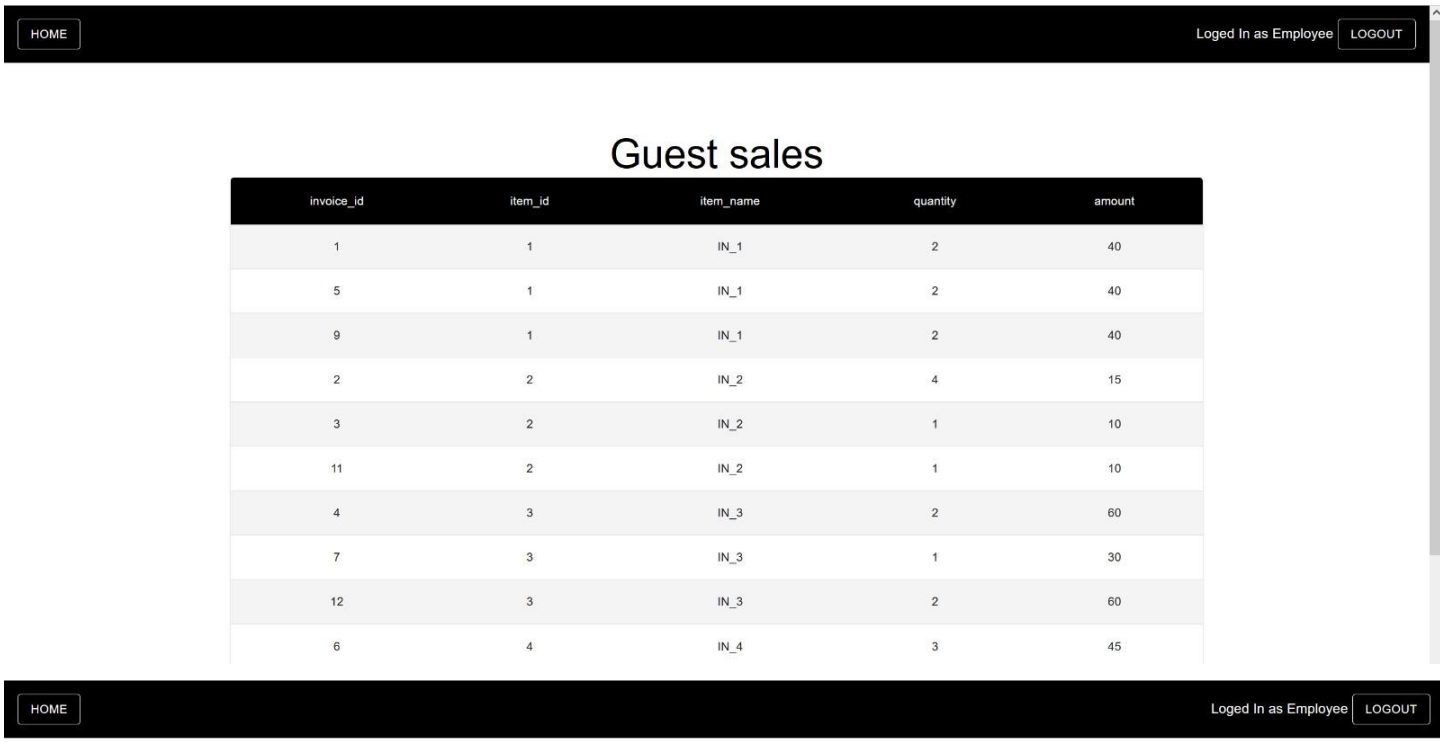
Feedback

Mess_id	Name	Date	Slot	Comment	Star Rating
1	S_N_1	Sat, 31 Dec 2022 18:30:00 GMT	breakfast	good	5
2	S_N_2	Sat, 31 Dec 2022 18:30:00 GMT	lunch	normal	4
3	S_N_3	Sun, 01 Jan 2023 18:30:00 GMT	dinner	not bad	4

Contractor detail

contractor Id	Contractor Name	Mess Id	HQ Office Number	HQ Street Number	HQ Street Name	HQ city	HQ State	HQ Pincode
1	abc	1	1	1	shivaji	s1	GJ	382355
2	def	2	2	2	ratrani	s2	GJ	382350
3	ghi	3	3	3	panchayat	s3	GJ	382340


HOME
Loged In as Employee
LOGOUT
Wastage detail
mess_id | date_slot_id | food_wasted
1 | 1 | 1
1 | 2 | 1
1 | 3 | 0
1 | 4 | 1
1 | 5 | 1
1 | 6 | 1
1 | 7 | 0
1 | 8 | 1
2 | 1 | 1
2 | 2 | 1
HOME
Loged In as Employee
LOGOUT
Mess menu
item_name | slot | calorie | protein | is_special | price
IN_1 | breakfast | 90 | 34 | False | 20
IN_2 | breakfast | 80 | 28 | False | 10
IN_4 | breakfast | 60 | 16 | False | 15
IN_2 | lunch | 80 | 28 | False | 10
IN_3 | lunch | 99 | 12 | True | 30
IN_4 | lunch | 60 | 16 | False | 15
IN_1 | snacks | 90 | 34 | False | 20
IN_4 | snacks | 60 | 16 | False | 15
IN_1 | dinner | 90 | 34 | False | 20
IN_3 | dinner | 99 | 12 | True | 30

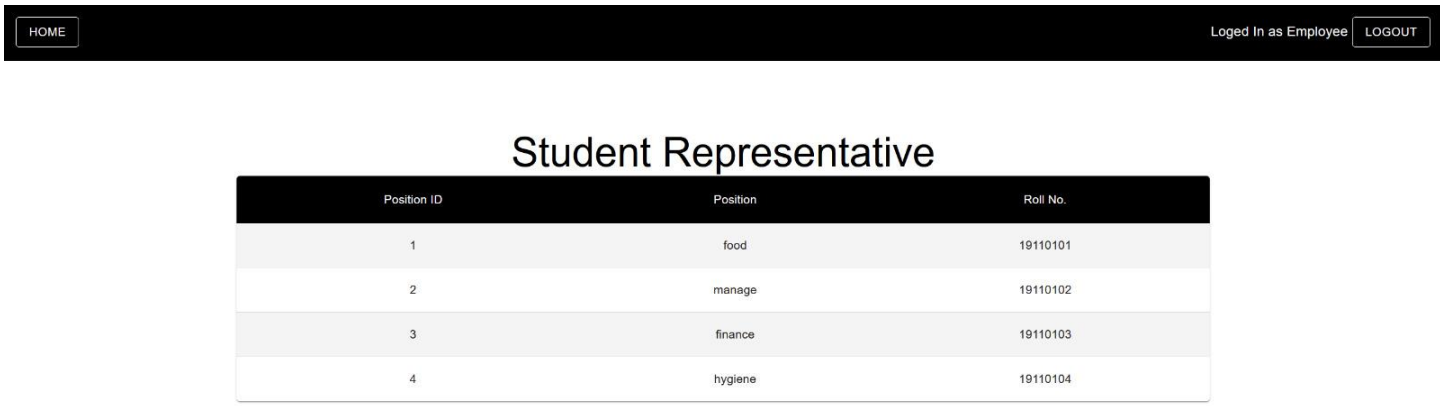


Guest sales

invoice_id	item_id	item_name	quantity	amount
1	1	IN_1	2	40
5	1	IN_1	2	40
9	1	IN_1	2	40
2	2	IN_2	4	15
3	2	IN_2	1	10
11	2	IN_2	1	10
4	3	IN_3	2	60
7	3	IN_3	1	30
12	3	IN_3	2	60
6	4	IN_4	3	45

Balance Sheet

balance_id	mess_id	month	year	from_guest	from_student	miscellaneous	to_employee	to_vendor
1	1	jan	2023	125	1840	1000	1534	5000
2	2	jan	2023	145	1840	1500	1534	5000
3	3	jan	2023	140	1840	1000	1534	5000

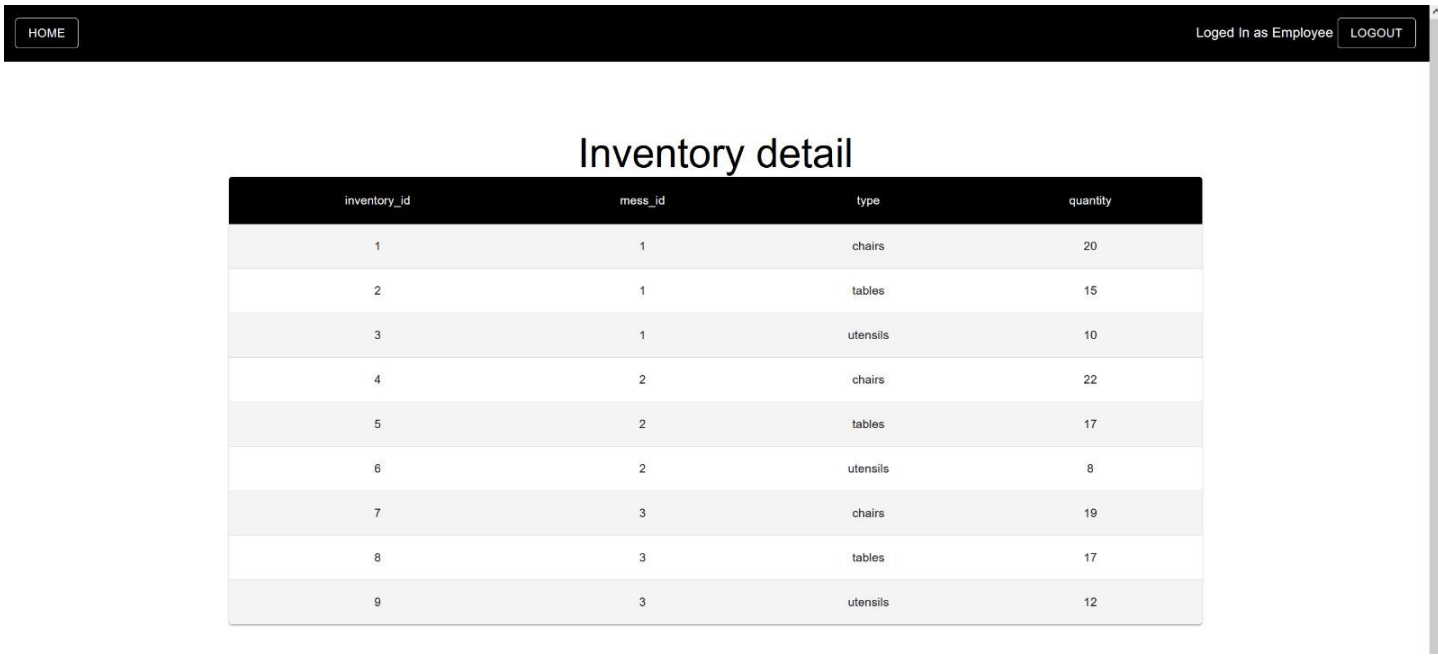


Student Representative

Position ID	Position	Roll No.
1	food	19110101
2	manage	19110102
3	finance	19110103
4	hygiene	19110104

Stock detail

product_id	mess_id	quantity	in_date	expiry
1	1	10	Thu, 29 Dec 2022 18:30:00 GMT	Sat, 29 Apr 2023 18:30:00 GMT
2	1	12	Thu, 29 Dec 2022 18:30:00 GMT	Sat, 29 Apr 2023 18:30:00 GMT
3	1	5	Thu, 29 Dec 2022 18:30:00 GMT	Sat, 29 Apr 2023 18:30:00 GMT
4	2	12	Fri, 30 Dec 2022 18:30:00 GMT	Wed, 29 Mar 2023 18:30:00 GMT
5	2	11	Fri, 30 Dec 2022 18:30:00 GMT	Wed, 29 Mar 2023 18:30:00 GMT
6	2	6	Fri, 30 Dec 2022 18:30:00 GMT	Wed, 29 Mar 2023 18:30:00 GMT
7	3	13	Wed, 28 Dec 2022 18:30:00 GMT	Mon, 29 May 2023 18:30:00 GMT
8	3	10	Wed, 28 Dec 2022 18:30:00 GMT	Mon, 29 May 2023 18:30:00 GMT
9	3	7	Wed, 28 Dec 2022 18:30:00 GMT	Mon, 29 May 2023 18:30:00 GMT

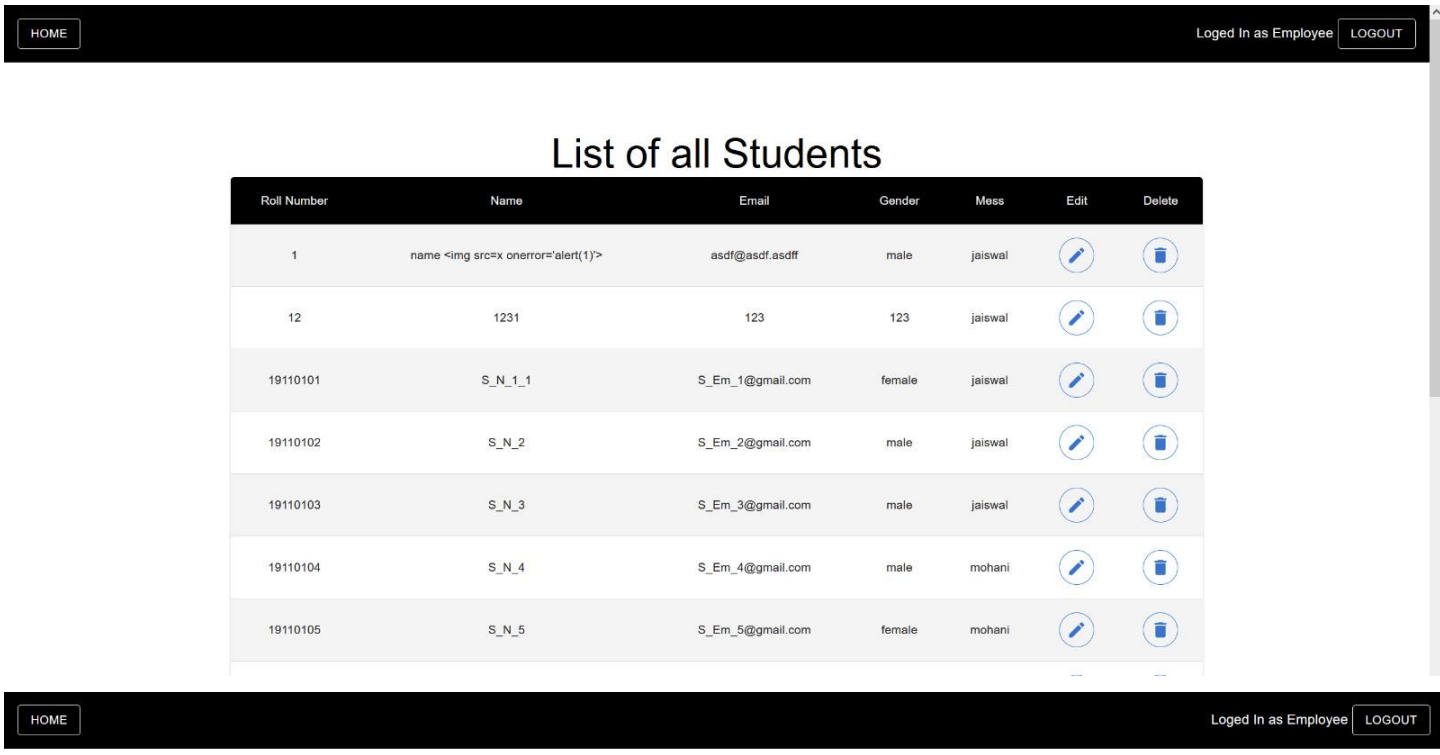


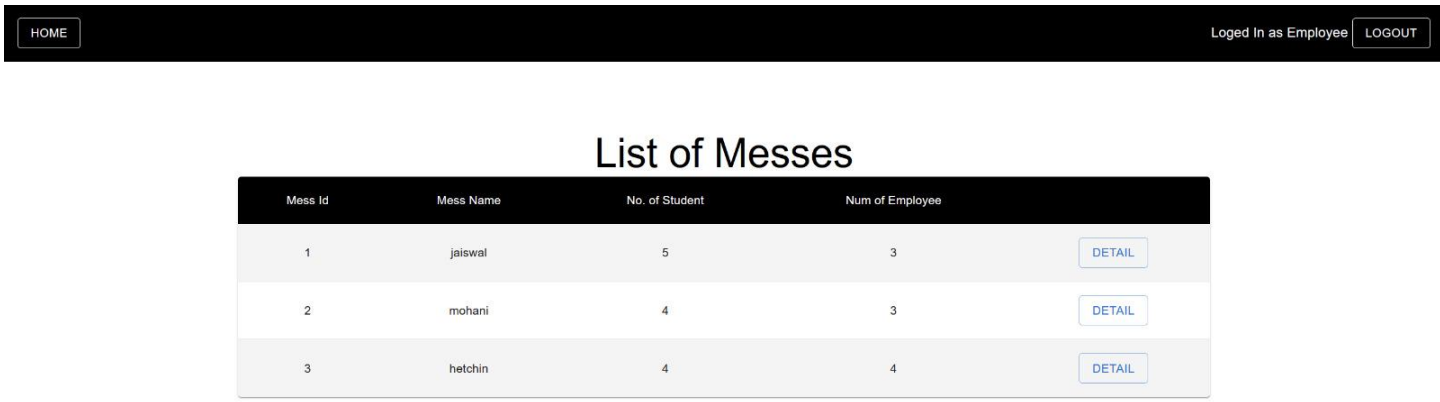
Inventory detail

inventory_id	mess_id	type	quantity
1	1	chairs	20
2	1	tables	15
3	1	utensils	10
4	2	chairs	22
5	2	tables	17
6	2	utensils	8
7	3	chairs	19
8	3	tables	17
9	3	utensils	12

List of all Empolyees

Designation	Name	Email	Mess_id	DoB	Gender	Address
manager	WN_1	W_EM_1@gmail.com	1	Sat, 31 Dec 1994 18:30:00 GMT	male	W_HN_1W_SN_1W_C_1W_S_1382388
chef	WN_2	W_EM_2@gmail.com	1	Tue, 31 Jan 1995 18:30:00 GMT	male	W_HN_2W_SN_2W_C_2W_S_2382389
worker	WN_3	W_EM_3@gmail.com	1	Tue, 28 Feb 1995 18:30:00 GMT	female	W_HN_3W_SN_3W_C_3W_S_3382390
manager	WN_4	W_EM_4@gmail.com	2	Fri, 31 Mar 1995 18:30:00 GMT	male	W_HN_4W_SN_4W_C_4W_S_4382391
chef	WN_5	W_EM_5@gmail.com	2	Sun, 30 Apr 1995 18:30:00 GMT	female	W_HN_5W_SN_5W_C_5W_S_5382392
worker	WN_6	W_EM_6@gmail.com	2	Wed, 31 May 1995 18:30:00 GMT	male	W_HN_6W_SN_6W_C_6W_S_6382393
manager	WN_7	W_EM_7@gmail.com	3	Fri, 30 Jun 1995 18:30:00 GMT	male	W_HN_7W_SN_7W_C_7W_S_7382394
chef	WN_8	W_EM_8@gmail.com	3	Mon, 31 Jul 1995 18:30:00 GMT	female	W_HN_8W_SN_8W_C_8W_S_8382395
worker	WN_9	W_EM_9@gmail.com	3	Thu, 31 Aug 1995 18:30:00 GMT	female	W_HN_9W_SN_9W_C_9W_S_9382396
worker	WN_10	vp.shivasan@iitgn.ac.in	3	Thu, 31 Aug 1995 18:30:00 GMT	female	W_HN_10W_SN_10W_C_10W_S_10382397





Almost all the tables giving full information about the databases added to the application

3. Responsibility of G1 and G2

3.1 Attacks [SQL Injection and XSS] performed and the defenses against those attacks:

SQL injection:

Use of python string formatters on user input for executing SQL queries creates a vulnerability where a malicious user could trick the code into executing custom queries which might lead to acts like invasion of privacy, data leaks.

Example: The following query was designed to showcase the inventory present for a particular mess_id.

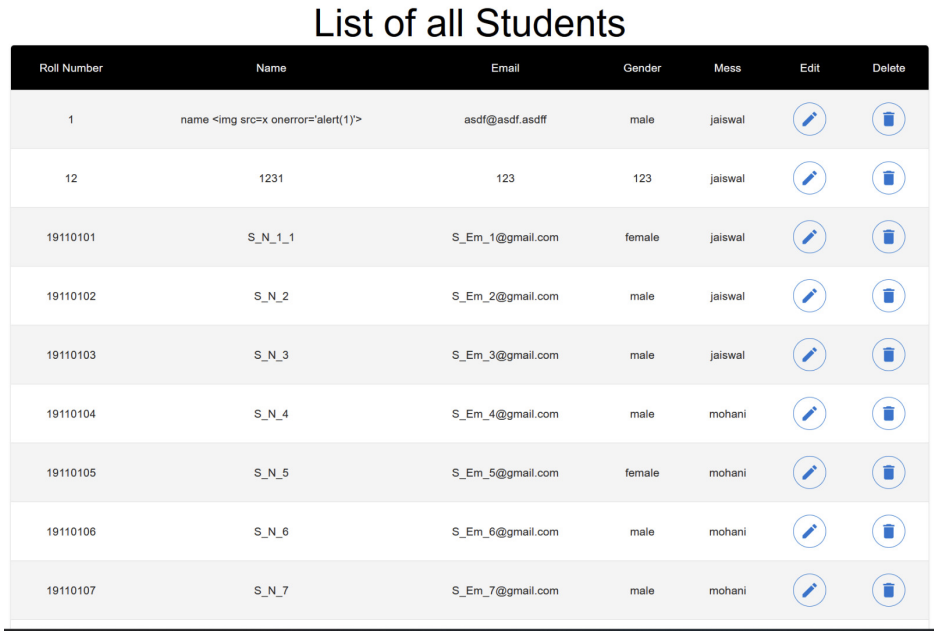
cursor.execute("""Select * from `inventory_present_at` where mess_id = '%s'""" % (mess_id))

However, due to python string formatter “%” a malicious user could exploit this vulnerability by sending mess_id as “1';update student_allocated set name="MySetNameWithoutRights" where roll_number=19110104 and email = 'S_Em_4@gmail.com”, which results in updating a student’s name.

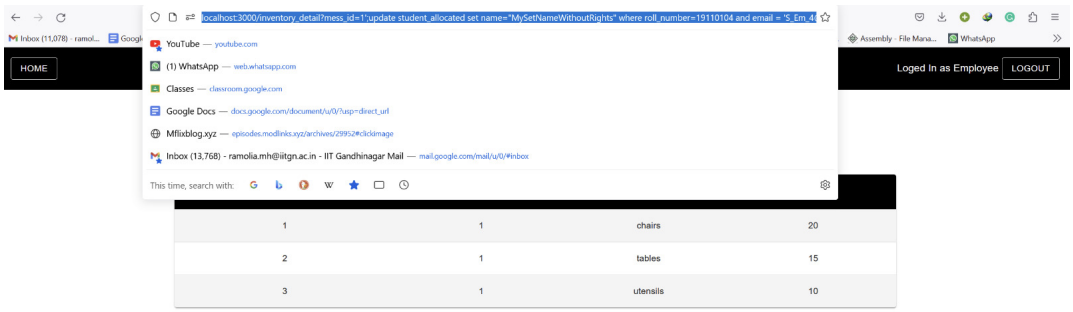
This vulnerability could be easily fixed by avoiding python string formatters and instead sending user inputs as argos to cursor.execute(). The fixed statement is as follows:

cursor.execute("""Select * from `inventory_present_at` where mess_id = '%s'""",[mess_id])

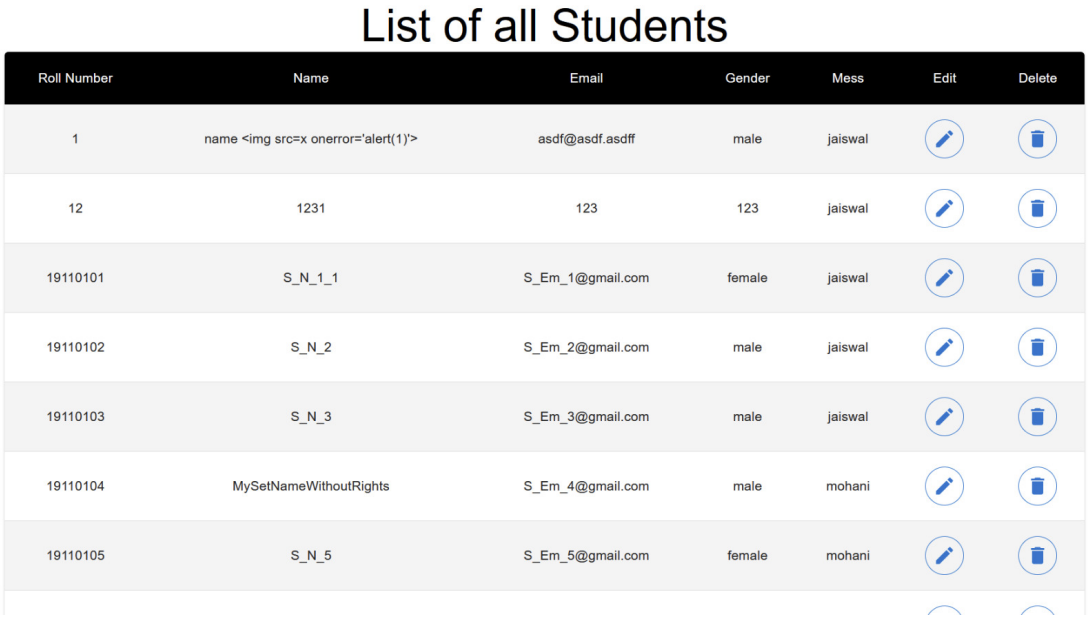
Before SQL injection:



Malicious Query:



After SQL injection:



List of all Students

Roll Number	Name	Email	Gender	Mess	Edit	Delete
1	name <img src=x onerror='alert(1)'>	asdf@asdf.asdff	male	jaiswal		
12	1231	123	123	jaiswal		
19110101	S_N_1_1	S_Em_1@gmail.com	female	jaiswal		
19110102	S_N_2	S_Em_2@gmail.com	male	jaiswal		
19110103	S_N_3	S_Em_3@gmail.com	male	jaiswal		
19110104	MySetNameWithoutRights	S_Em_4@gmail.com	male	mohani		
19110105	S_N_5	S_Em_5@gmail.com	female	mohani		

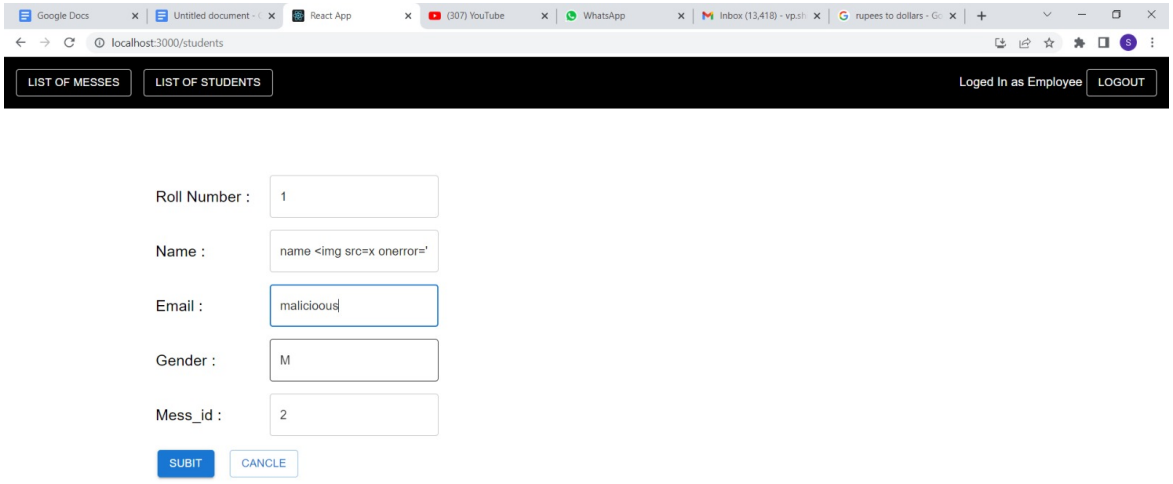
XSS:

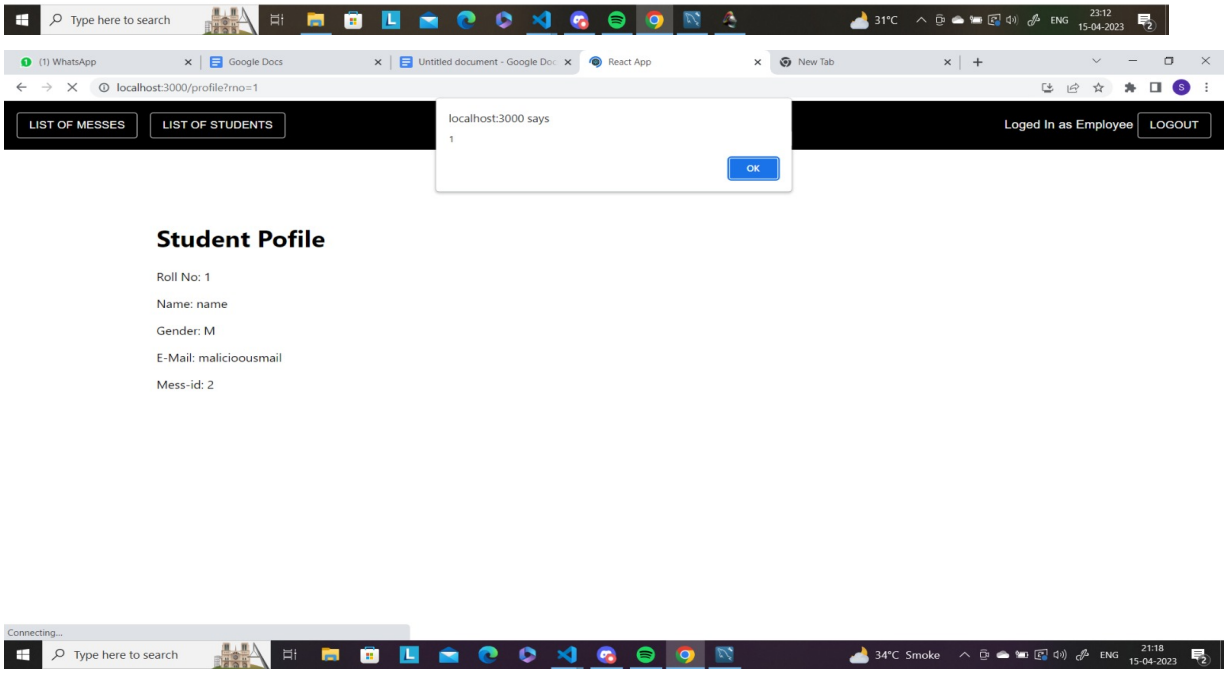
The webpage /profile is susceptible to XSS attacks. Here we showcase persistent XSS attack, where a malicious user changes/appends a malicious message to the database often containing javascript and whenever anyone visits the malicious user's profile the malicious javascript is executed.

Malicious query: StudentName <img src=x onerror='alert(1)'>

WebPage: http://localhost:3000/profile?rno=1

This vulnerability can be handled by escaping the rendering HTML, and in Flask this can be accomplished by "{% autoescape True %}" which escape the script tags and render any malicious code as plain text in the webpage rather than executing them.



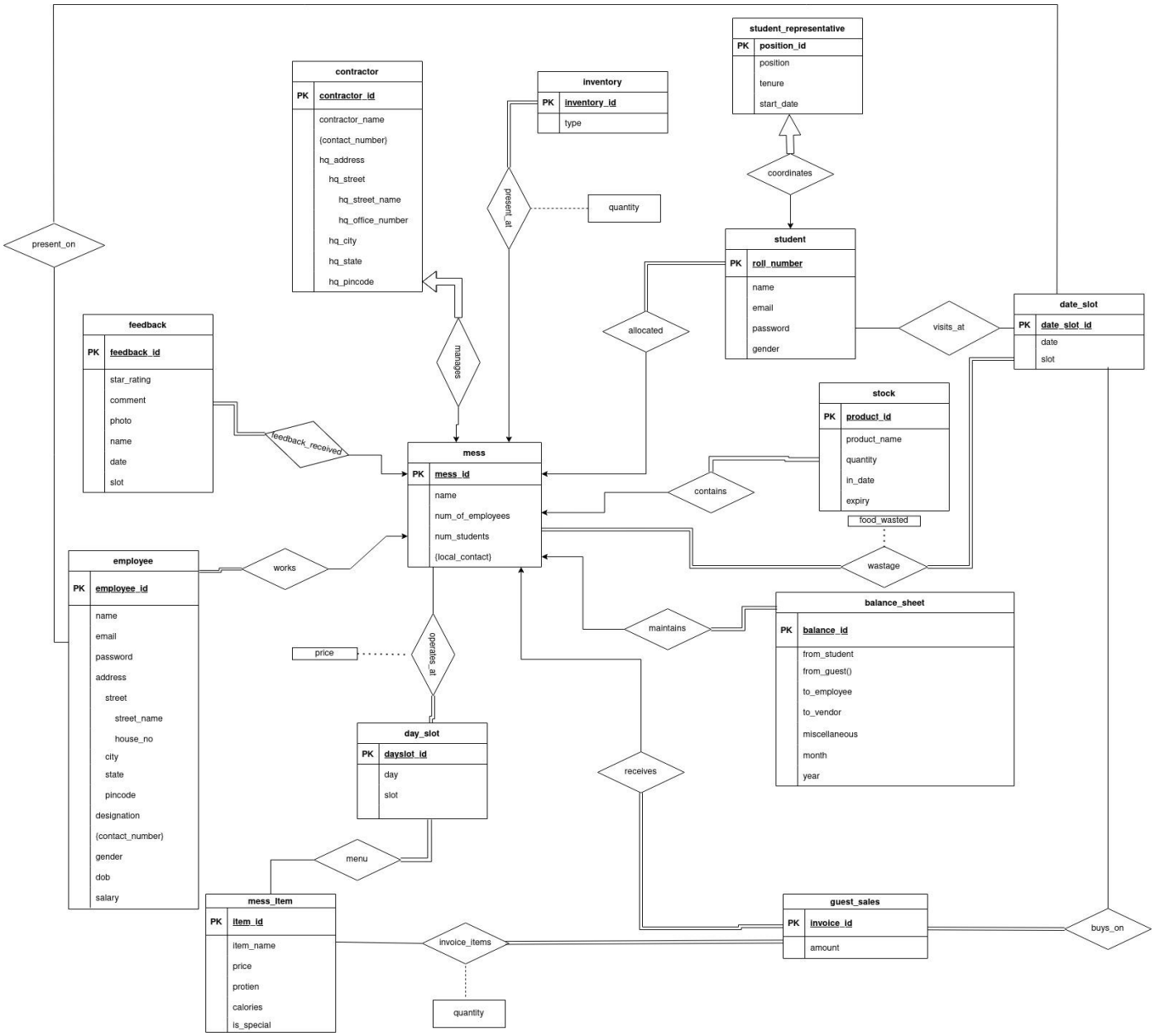


3.2 Show that all the relations and their constraints, finalized after the second feedback, are present and valid as per the ER diagram constructed in Assignment 1:

All of the 4 types of Constraints(mentioned in Assignment 1) in are satisfied in our DBMS along with ACID properties:

1. Domain Constraints: Salary is greater than 5,000.

2. Key Constraints and Entity Integrity Constraints: All of the entities and relations have primary keys with no null, so all tables have key constraints, tuple uniqueness constraints and entity integrity constraints satisfied.
3. Referential Integrity Constraints: all the referred tables are referenced using the primary keys of other tables as foreign keys.



Link to ER Diagram: https://drive.google.com/file/d/1OAhjqfx6VF9GBxos4WElF_Z4CB6ThwsR/view?usp=sharing

Contribution:

G1 Group:, Harshit Ramolia, Insha Mansuri, Somesh Pratap Singh, Priya Gupta, Pulkit Jain, Md Amir Sohail
G2 Group: VP Shivasankaran, Muhammad Yusuf Hassan, Patel Rajan Girishbhai, Shantanu Sahu, Patel Agam, Dhruv Darda