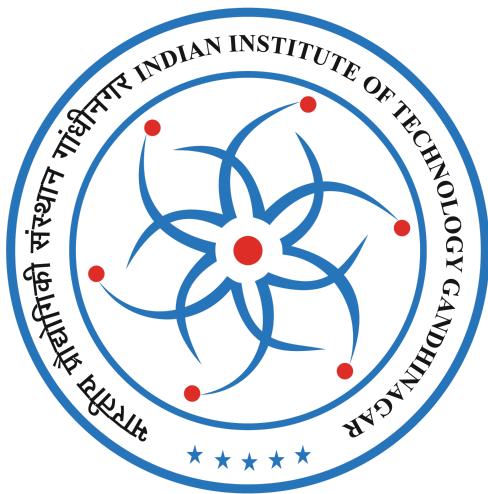


CS 432 Final Report

Food Delivery



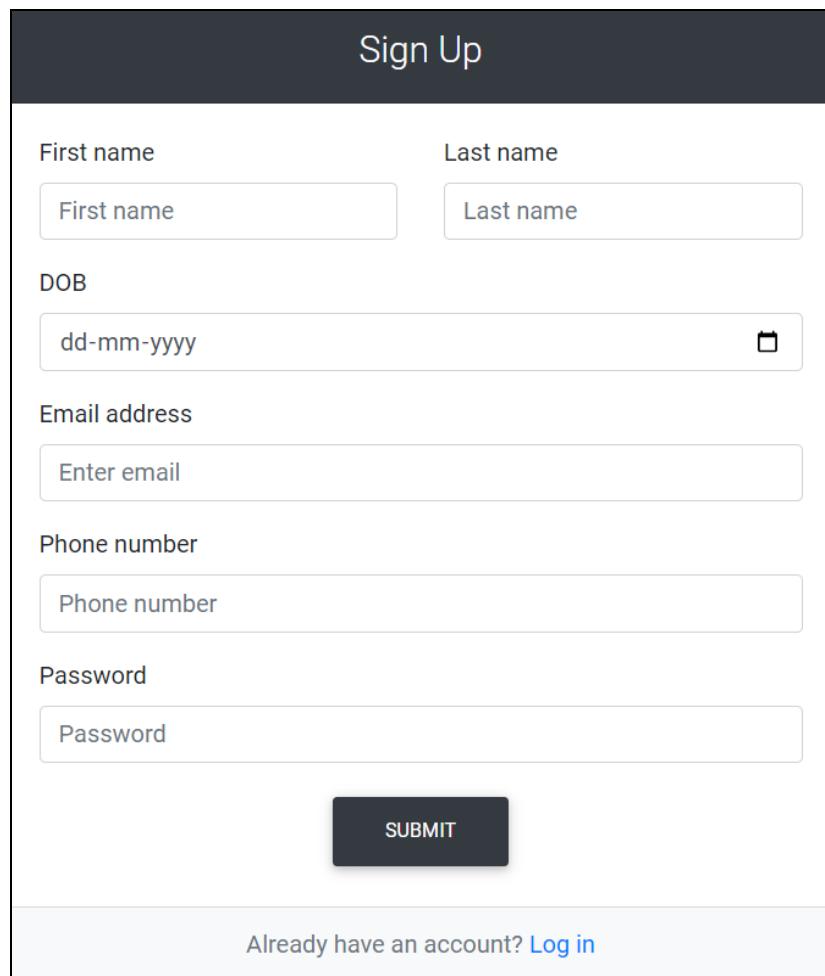
Osborg
4th September, 20XX

3.1.1

We took two rounds of feedback from all stakeholders and, based on the review, made the following changes in our food delivery system. Changes post the first feedback, compared to the initial version.

Customer:

Filling up signup details. All the mentioned fields are mandatory. Each signup page is customized as per the user's authorization.



The image shows a 'Sign Up' form interface. At the top, there is a dark header bar with the text 'Sign Up'. Below the header, there are several input fields: 'First name' and 'Last name' in separate input boxes; 'DOB' with a date input field ('dd-mm-yyyy') and a calendar icon; 'Email address' with an input field containing placeholder text 'Enter email'; 'Phone number' with an input field containing placeholder text 'Phone number'; and 'Password' with an input field containing placeholder text 'Password'. A large 'SUBMIT' button is centered below these fields. At the bottom of the form, there is a link 'Already have an account? Log in'.

Based on feedback, the modification was made so that stakeholders can now enter their addresses at the time of signup.

Sign Up

First name Middle name Last name

DOB

□

Email address Phone number

Address

Password

SUBMIT

Already have an account? [Log in](#)

This page showcases the personal details of the user and the status of the orders they have placed.

User Name: Ulises Gottlieb
UserPhone number : 1559563370
user email: leon82@example.org

Your Orders

Order ID	Restaurant	Amount	Order Time	Status	Action
1	Marquardt, Roob and Green	1134.00	2023-03-20 23:08:03	In Progress	
2	Prohaska, Carroll and Hil	2898.00	2023-01-13 17:24:58	Delivered	<button>DELETE</button>

© 2023 Copyright@Osborg [Dashboard](#) [About Us](#)

The view was modified so that the user can now choose the address for their account at the time of placing the order.

Item	Description	Price	Type	Quantity
Butter chicken	744.00	Main Course	<input type="text" value="0"/>	
Chana masala	759.00	Main Course	<input type="text" value="0"/>	
Chapati	444.00	Main Course	<input type="text" value="0"/>	
Chicken razala	431.00	Main Course	<input type="text" value="0"/>	
Chicken Tikka masala	107.00	Main Course	<input type="text" value="0"/>	
Chicken Tikka	192.00	Starter	<input type="text" value="0"/>	
Chole bhature	182.00	Main Course	<input type="text" value="0"/>	

Is Paid

© 2023 Copyright@Osborg [Dashboard](#) [About Us](#)

Restaurant:

Restaurant owners can access these pages. They will be able to see the “Name,” “Address,” “Email,” and “Rating” of the restaurant. They can view the latest 10 orders given to this restaurant.

The screenshot shows a web application interface for a restaurant. At the top, there are navigation links for "Home" and "Menu" on the left, and "Restaurant Detail" on the right. Below this is a "Restaurant Details" section containing the following information:

Name: Hyatt-Pfannerstill	Email: skylar.dach@example.com
Phone Number: 599477815	Rating: 3.2

Below this is a "Your Deliveries" section. It displays two separate order tables, each with an image of a dish, item name, price, and quantity. To the right of the first table, summary information is provided:

Item Name: Qubani ka meetha	Item Price: 232.00	Item Quantity: 3
Item Name: Chikki	Item Price: 447.00	Item Quantity: 1
Item Name: Sev tameta	Item Price: 285.00	Item Quantity: 2

Total price: 1713.00
Order time: 2023-02-07 07:02:38
Order status: delivered

Below the first order table is another similar table for a different order:

Item Name: [Image]	Item Price: [Redacted]	Item Quantity: [Redacted]
--------------------	------------------------	---------------------------

Total price: 225.00
Order time: 2023-01-30

This was modified to give them access to edit their email address and phone numbers at any given time. Also, the opening and closing time of the restaurant is mentioned separately for weekdays and weekends.

The screenshot shows the same "Restaurant Detail" page as above, but with additional edit functionality. In the "Restaurant Details" section, there is a "EDIT" button below the initial information. Below this, there are input fields for "Email" and "Phone Number", each with a corresponding "SAVE CHANGES" button.

Name: Hyatt-Pfannerstill	Street Name: Iman	Weekday Opening Time: 15:00:00
Email: skylar.dach@example.com	City: Benjaminberg	Weekday Closing Time: 22:00:00
Phone Number: 599477815	State: WestVirginia	Weekend Opening Time: 8:00:00
Rating: 3.2	Pin Code: 918102	Weekend Closing Time: 2:00:00

Moreover, a feature to sort orders according to duration is also provided. They can filter orders for the day or the week.

The screenshot shows a web-based application for managing restaurant details and order history. At the top, there's a navigation bar with 'Home' and 'Menu' on the left and 'Restaurant Detail' on the right. Below the navigation is a section titled 'Restaurant Details' containing various information fields:

Name: Hyatt-Pfannerstill	Street Name: Iman	Weekday Opening Time: 15:00:00
Email: skylar.dach@example.com	City: Benjaminberg	Weekday Closing Time: 22:00:00
Phone Number: 599477815	State: WestVirginia	Weekend Opening Time: 8:00:00
Rating: 3.2	Pin Code: 918102	Weekend Closing Time: 2:00:00

A blue 'EDIT' button is located below these fields. Below the details is a dropdown menu labeled 'Orders of duration?' with options: 'Orders of duration?', 'Today', 'Last Week', and 'All'. The 'All' option is currently selected. To the right of the dropdown is a table showing order history:

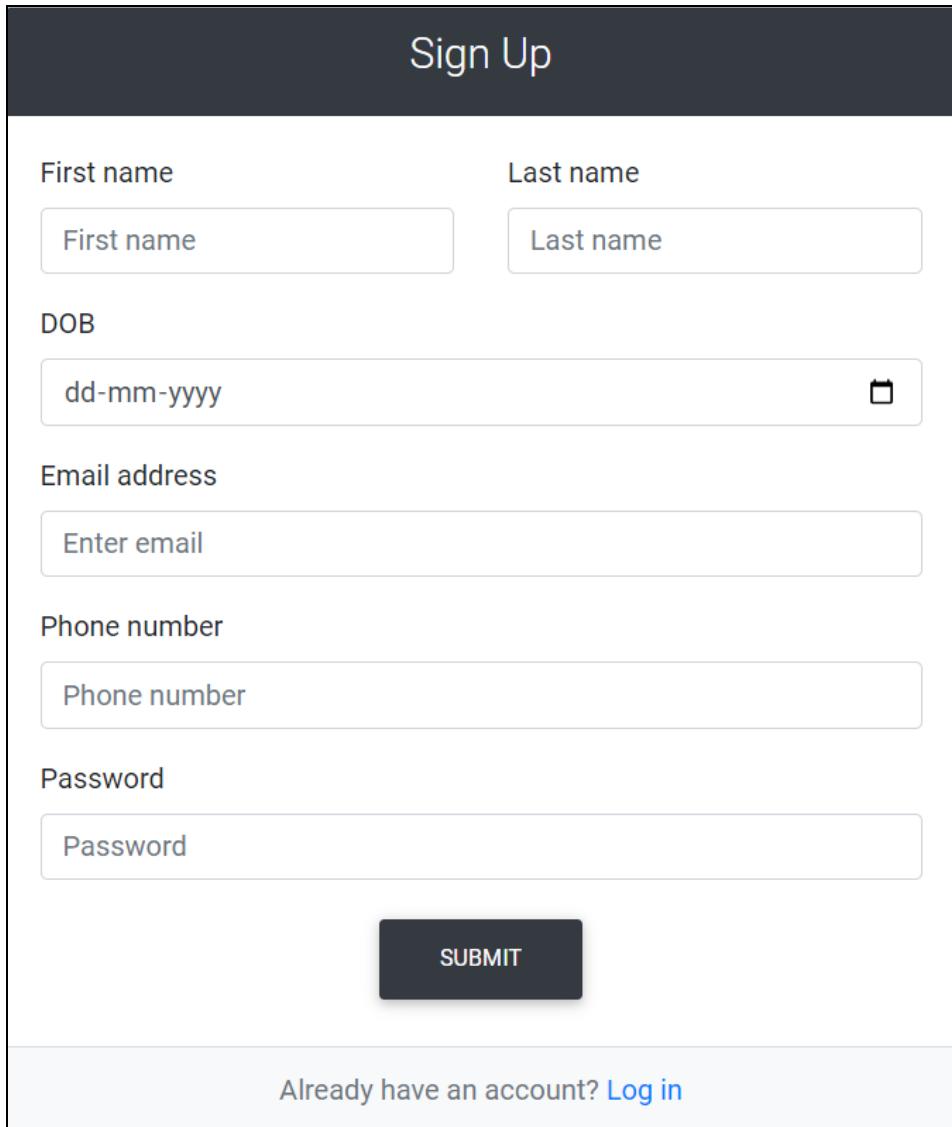
Item Name	Item Price	Item Quantity
Qubani ka meetha	232.00	3
Chikki	447.00	1

On the far right of the order table, the total price and order time are displayed:

Total price: 1713.00
Order time: 2023-02-07 07:02:38

Delivery Agent:

Filling up signup details. All the mentioned fields are mandatory. Each signup page is customized as per the user's authorization.



The image shows a 'Sign Up' form interface. At the top, there is a dark header bar with the text 'Sign Up'. Below the header, there are six input fields arranged in two columns: 'First name' and 'Last name', 'DOB', 'Email address', 'Phone number', and 'Password'. Each input field has a placeholder text and a clear button on the right. Below these fields is a large 'SUBMIT' button. At the bottom of the form, there is a link 'Already have an account? [Log in](#)'.

Sign Up	
First name	Last name
<input type="text" value="First name"/>	<input type="text" value="Last name"/>
DOB	<input type="text" value="dd-mm-yyyy"/> 
Email address	<input type="text" value="Enter email"/>
Phone number	<input type="text" value="Phone number"/>
Password	<input type="text" value="Password"/>
SUBMIT	
Already have an account? Log in	

Based on feedback, the modification was made so that delivery agents can enter their addresses at signup.

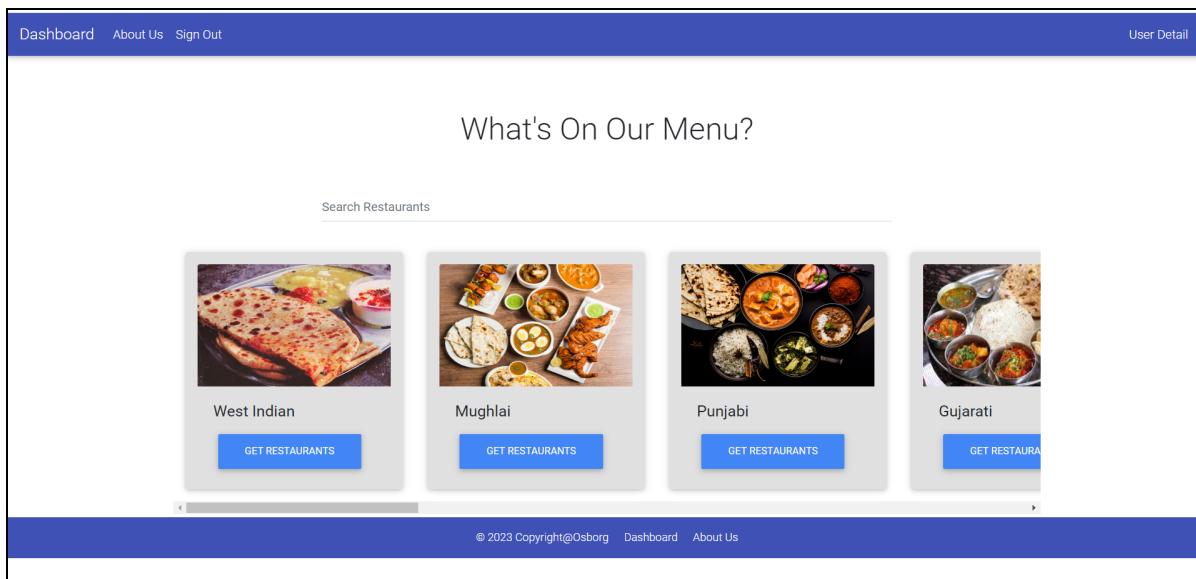
Sign Up

First name	Middle name	Last name
<input type="text" value="First name"/>	<input type="text" value="Middle name"/>	<input type="text" value="Last name"/>
DOB		
<input type="text" value="dd-mm-yyyy"/>		<input type="button" value=""/>
Email address	Phone number	
<input type="text" value="Enter email"/>	<input type="text" value="Phone number"/>	
Address		
<input type="text" value="Building Number"/>	<input type="text" value="Street"/>	
<input type="text" value="City name"/>	<input type="text" value="State Name"/>	
<input type="text" value="803110"/>		
Password		
<input type="text" value="Password"/>		
<input type="button" value="SUBMIT"/>		
Already have an account? Log in		

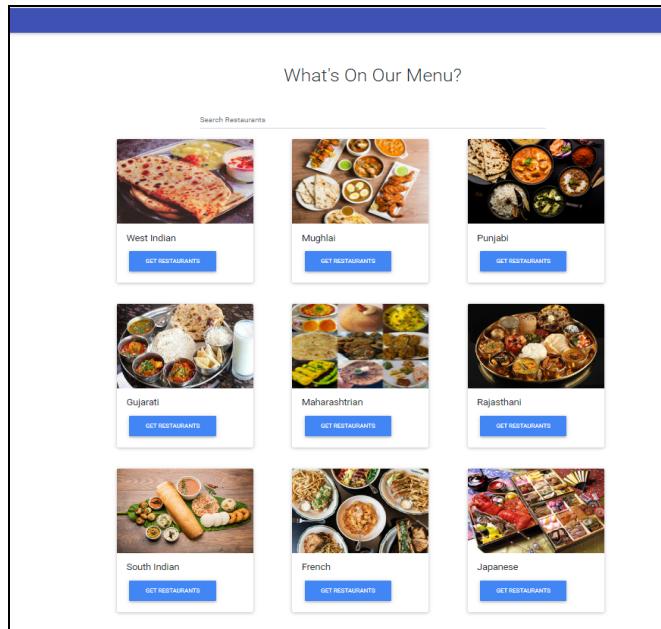
Further changes introduced after the second round of feedback:

Customer:

Dashboard for customers. It contains a search bar and cuisine types being served by restaurants. Earlier, the customers had to slide through a horizontal column to view all types being served.



It is now modified vertically, such that the users can view all cuisine types at the same time for a better customer experience.



The view was modified so that instead of choosing the order through a button, we can change it using a simple radio button for a better customer experience. This enables them to order from different places like home, office, etc.

<input checked="" type="radio"/>	Butter chicken	744.00	main course	<input type="text" value="0"/>
<input checked="" type="radio"/>	Chana masala	759.00	main course	<input type="text" value="0"/>
<input checked="" type="radio"/>	Chapati	444.00	main course	<input type="text" value="0"/>
<input checked="" type="radio"/>	Chicken razala	431.00	main course	<input type="text" value="0"/>
<input checked="" type="radio"/>	Chicken Tikka masala	107.00	main course	<input type="text" value="0"/>
<input checked="" type="radio"/>	Chicken Tikka	192.00	starter	<input type="text" value="0"/>
<input checked="" type="radio"/>	Chole bhature	182.00	main course	<input type="text" value="0"/>

is Paid

© 2023 Copyright@Osborg [Dashboard](#) [About Us](#)

The changed view with the radio buttons.

This screenshot shows a user profile page with a blue header bar containing 'Dashboard', 'About Us', 'Sign Out', and 'User Detail'. The main content area displays the user's name 'Gillian Kunde' and contact information: 'UserPhone number : 535960267' and 'user email: johnson@example.net'. Below this is a section for selecting an address, featuring three radio button options with their respective building numbers, street names, cities, states, and pin codes. A blue 'CHOOSE ADDRESS' button is positioned below the radio buttons. The next section, titled 'Your Orders', shows two recent orders from 'Hyatt-Pfannerstill'. Each order card includes a thumbnail image of a pizza, the restaurant name, the order time, the status ('In Progress'), and the order status ('placed').

Restaurant:

Restaurant owners can access these pages. They will be able to see “Name”, “Address”, “Email”, and “Rating” of the restaurant. They can view the latest 10 orders given to this restaurant.

This screenshot shows a restaurant owner's dashboard with a blue header bar containing 'Home', 'Menu', and 'Restaurant Detail'. The main content area starts with a 'Restaurant Details' section showing the name 'Hyatt-Pfannerstill', email 'skylar.dach@example.com', phone number '599477815', and a rating of '3.2'. Below this is a 'Your Deliveries' section displaying a table of recent deliveries. The table has columns for 'Item Name', 'Item Price', and 'Item Quantity'. It lists three items: 'Qubani ka meetha' (232.00), 'Chikki' (447.00), and 'Sev tameta' (285.00). To the right of the table, delivery details are shown: 'Total price: 1713.00', 'Order time: 2023-02-07 07:02:38', and 'Order status: delivered'. At the bottom of the page, there is another partial view of a delivery card showing a pizza thumbnail and some table headers.

This was changed so that restaurant owners could update the status of the order, and change it to ready whenever the food is prepared.

The screenshot shows a web-based application interface for managing restaurant orders. At the top, there is a blue header bar with 'Home' and 'Menu' on the left and 'Restaurant Detail' on the right. Below the header, the main content area is divided into two sections: 'Restaurant Details' and 'Your All Deliveries'.

Restaurant Details: This section contains form fields for entering restaurant information. The fields include:

Name: Hyatt-Pfannerstill	Street Name: iman	Weekday Opening Time: 15:00:00
Email: skylar.dach@example.com	City: Benjaminberg	Weekday Closing Time: 22:00:00
Phone Number: 599477815	State: WestVirginia	Weekend Opening Time: 8:00:00
Rating: 3.2	Pin Code: 918102	Weekend Closing Time: 2:00:00

A blue 'EDIT' button is located below the form fields.

Your All Deliveries: This section displays a list of delivered orders. It includes a dropdown menu labeled 'Orders of duration?'. Below the dropdown, the heading 'Your All Deliveries' is displayed. A thumbnail image of a pizza is shown next to a table listing the order details:

Item Name	Item Price	Item Quantity
Chhena poda	264.00	1
Cham.cham	255.00	1

To the right of the table, the total price is listed as 'Total price: 519.00', the order time as 'Order time: 2023-04-15 20:55:56', and the order status as 'Order status: placed'. A blue 'READY' button is located at the bottom right of this section.

3.1.2

Customer:

The customer has the following rights and privileges that enable them to make the following changes to the database:

- At the time of signup, they enter their personal details. These changes are directly reflected in the customer's table. Since customers can add multiple addresses, they can also access the customer address table.

[Sign Up](#)

First name

Middle name

Last name

First name

Middle name

Last name

DOB

dd-mm-yyyy

Email address

Phone number

Enter email

Phone number

Address

Building Number

Street

City name

State Name

803110

Password

Password

SUBMIT

Already have an account? [Log in](#)

- Place an order from any restaurant to create a new entry in the orders table. They can also view various cuisines, allowing them to read data from the cuisine-type table.

Result is Here

Restaurant Name: Breitenberg, Hartmann and Email: bailey.emilio@example.org Phone: 22338308 Rating: 5.0	Restaurant Addr City: South Friedrichburgh State: Indiana Pincode: 913117	CHECK MENU
Restaurant Name: Hansen-Greenholt Email: vward@example.com Phone: 1154754225 Rating: 4.9	Restaurant Addr City: Benjaminberg State: WestVirginia Pincode: 918102	CHECK MENU
Restaurant Name: Stehr-Brown Email: parker.bailey@example.com Phone: 845052598 Rating: 4.8	Restaurant Addr City: South Joany State: Pennsylvania Pincode: 173619	CHECK MENU

- They can view the menu for any restaurant, thus having read access to the menu items table.

Dashboard About Us Sign Out User Detail

Powlowski Ltd

Menu Items

Veg/Non veg	Item Name	Unit Price	Type	Quantity
▢	Daal Dhokli	662.00	main course	0
▢	Kutchi dabeli	352.00	snack	0
▢	Dalithoy	115.00	main course	0
▢	Dhokla	220.00	snack	0
▢	Dudhi halwa	774.00	dessert	0
▢	Gatta curry	760.00	main course	0
▢	Gud peddi	725.00	dessert	0

is Paid **SUBMIT**

© 2023 Copyright@Osborg Dashboard About Us

- Choose any addresses they provided for the order, reflecting the corresponding change in the delivery_detail table.

The screenshot shows a 'Restaurant Details' section with the following data:

Name: Hyatt-Pfannerstill	Street Name:Iman	Weekday Opening Time: 15:00:00
Email: skylar.dach@example.com	City:Benjaminberg	Weekday Closing Time: 22:00:00
Phone Number:599477815	State: WestVirginia	Weekend Opening Time: 8:00:00
Rating:3.2	Pin Code: 918102	Weekend Closing Time: 2:00:00

EDIT

A dropdown menu titled 'Orders of duration?' is open, showing options: Today, Last Week, All.

Item Name	Item Price	Item Quantity
Qubani ka meetha	232.00	3
Chikki	447.00	1

Total price: 1713.00
Order time: 2023-02-07 07:02:38

- Customers can also edit their order's payment details, thus having access to the payments table. They also can view their earlier orders, whose data they can view from the order_items table.

The screenshot shows an 'Order Submitted!!' message with the following details:

Ordering from Carroll Ltd
[Go to the dashboard](#)

Order Summary:

item_name	unit_price	price
Boondi	325.00	325
Gulab jamun	101.00	202

Total: 527

At the bottom, there is a footer bar with links: © 2023 Copyright@Osborg, Dashboard, About Us.

Restaurant:

The restaurant owners have the following rights and privileges that enable them to make the following changes to the database:

- At the time of signup, they enter their personal details. These changes are directly reflected in the restaurant table.

The form is titled "Sign Up". It contains fields for "Restaurant name", "Email address", "Phone number", "Address" (with fields for "Building Number", "Street", "City name", "State Name", and a zip code field), "Password", and a "SUBMIT" button. At the bottom, there is a link "Already have an account? Log in".

- They can edit their personal details, like email, phone number, etc., at any point. These changes will be directly reflected in the restaurant's table.

The page shows "Restaurant Details" for a restaurant named "Hyatt-Pfannerstill". It lists details such as Name, Email, Phone Number, Rating, Street Name, City, State, Pin Code, Weekday Opening Time, Weekday Closing Time, Weekend Opening Time, and Weekend Closing Time. Below this, there is an "EDIT" button, fields for "Email" and "Phone Number", and a "SAVE CHANGES" button. The top navigation bar includes "Home", "Menu", and "Restaurant Detail".

- They can change the order status by flagging when the food is ready. This change is directly reflected in the orders table.

This screenshot shows a restaurant detail interface. At the top, there are tabs for 'Home' and 'Menu'. On the right, it says 'Restaurant Detail'. Below this, there's a 'Restaurant Details' section with fields for Name, Street Name, Weekday Opening Time, Email, City, Weekday Closing Time, Phone Number, State, Weekend Opening Time, Rating, Pin Code, and Weekend Closing Time. An 'EDIT' button is located at the bottom of this section. Below this is a section titled 'Your All Deliveries' with a dropdown menu set to 'Orders of duration?'. It shows a table of delivered items:

Item Name	Item Price	Item Quantity
Chhena poda	264.00	1
Cham cham	255.00	1

Total price: 519.00
Order time: 2023-04-15 20:55:56
Order status: placed

A blue 'READY' button is visible next to the delivery details.

- They can also change and update the items from their menu. That gives them access to edit menu_items and cuisine_type tables.

This screenshot shows a menu management interface. At the top, there are tabs for 'Home' and 'Menu'. On the right, it says 'Restaurant Detail'. In the center, there's a button 'ADD NEW MENU' and the text 'Restaurant Name: Hyatt-Pfannerstill'. Below this is a table of menu items:

Veg/Non-veg	Availability	Item Name	Unit Price	Type	Edit
<input checked="" type="checkbox"/>	YES	Chhena poda	264.00	dessert	<input type="button" value="EDIT"/>
<input checked="" type="checkbox"/>	YES	Cham cham	255.00	dessert	<input type="button" value="EDIT"/>
<input checked="" type="checkbox"/>	YES	Kheer sagar	265.00	dessert	<input type="button" value="EDIT"/>
<input checked="" type="checkbox"/>	YES	Ledikeni	661.00	dessert	<input type="button" value="EDIT"/>
<input checked="" type="checkbox"/>	YES	Lyangcha	411.00	dessert	<input type="button" value="EDIT"/>

Delivery Agent:

- At the time of signup, they enter their personal details. These changes are directly reflected in the delivery agent table.

Sign Up

First name	Middle name	Last name
<input type="text" value="First name"/>	<input type="text" value="Middle name"/>	<input type="text" value="Last name"/>
DOB	<input type="text" value="dd-mm-yyyy"/> CALENDAR	
Email address	Phone number	
<input type="text" value="Enter email"/>	<input type="text" value="Phone number"/>	
Address		
<input type="text" value="Building Number"/>	<input type="text" value="Street"/>	
<input type="text" value="City name"/>	<input type="text" value="State Name"/>	
<input type="text" value="803110"/>		
Password		
<input type="text" value="Password"/>		

Already have an account? [Log in](#)

- They can view the orders to be picked and choose to take up the order. This gives them access to the view and edits access to the order table

The screenshot shows a web application interface for a delivery agent. At the top, a green header bar displays the message "Logged in successfully!". Below this is a blue "SIGN OUT" button. The main title "Delivery Agent Page" is centered above a table. To the left of the table, a sidebar titled "Agent Details" contains the agent's name, email, and phone number.

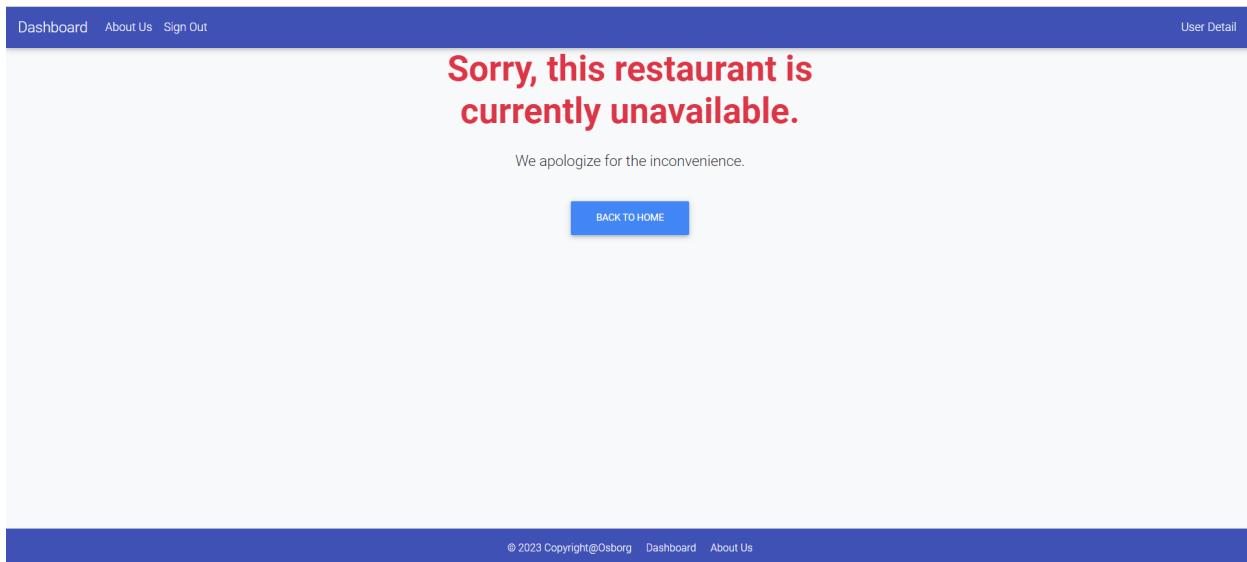
Order ID	Customer Phone number	Restaurant Address	Delivery Address	Delivery Charge	Pickup Time	Delivery Time	Current Order Status
1501	535980287	y785 Nata Ernestoborough NewYork 687998	Missing for now	None	None	None	placed
244	932982480	z774 Doug Fridabury Michigan 161719	s775 Manu Kendallton Pennsylvania 967481	289	2023-02-10 11:14:09	2023-02-10 13:08:36	delivered
1010	685548462	o927 Hami Savannahview NorthCarolina 680341	p160 Donn Waelchberg Ohio 488578	385	2023-02-10 09:05:25	2023-02-10 11:17:29	delivered
800	1317250753	g883 Mito Myrtieburgh Wyoming 750649	w534 Blan Gloverville NorthDakota 143913	894	2023-02-10 09:06:23	2023-02-10 10:26:50	delivered
3	23056547	x809 Creo Lake Friedrich NorthCarolina 186118	y952 Okun Denesikmouth Idaho 159482	407	2023-02-10 08:51:18	2023-02-10 09:28:03	delivered
872	388838774	c654 Boga East Maymie SouthCarolina 164434	e806 Summ Palmaview Arkansas 216564	205	2023-02-10 08:57:07	2023-02-10 09:01:28	delivered
848	1445065110	z259 Shay Hickleborough Arizona 110576	x595 Cleo East Pamela Idaho 157300	410	2023-02-10 08:59:32	2023-02-10 09:57:53	delivered
809	1400295995	o927 Hami Savannahview NorthCarolina 680341	z311 Geov Lake Piper District of Columbia 101829	323	2023-02-10 04:46:36	2023-02-10 05:57:15	delivered
210	1082541047	e787 Pasc Lake Abdielville WestVirginia 639057	s438 Soph Wolfstad Vermont 788299	772	2023-02-10 04:32:16	2023-02-10 07:15:46	delivered
1118	184318055	j739 Alys East Camille Nevada 116925	d575 Alba West Hymanchester Connecticut 107834	313	2023-02-10 01:24:49	2023-02-10 04:18:26	delivered

At the bottom of the page, a blue footer bar contains the copyright information "© 2023 Copyright@Osborg" and links to "Home" and "About Us".

3.2.1

We have a table named 'menu_item' that is accessed by both customers and restaurants. Customers only read from it, while restaurants write to it. To prevent conflicts, we need to use a locking mechanism to ensure that these operations do not occur simultaneously. This is important because we do not want a customer to read menu items that are being updated by a restaurant.

To implement this, we use a lock as soon as a restaurant begins editing their menu. If a customer tries to access that restaurant's menu during this time, we redirect them to a page that says 'restaurant currently unavailable'. We release the lock once the menu update is complete.



This is the page a customer sees when they are trying to access the menu of a restaurant that the restaurant is currently updating.

3.2.2

We have made multiple changes in the frontend part also but as the question asks about the changes in the backend part we are mentioning the snippets of the code for the backend changes.

Adding customer address on the sign-up page.

```
deliverysystem > cd app.py > login
# making routes for sign up for all three types of users
@app.route('/signupcustomer',methods=['GET', 'POST'])
def signupcustomer():
    if request.method == 'POST':
        msg = 'CUSTOMER: Please fill out the form again'
        userdetails = request.form
        firstname = userdetails['firstname']
        lastname = userdetails['lastname']
        email = userdetails['email']
        DOB = userdetails['DOB']
        phone_number = userdetails['phone_number']
        password = userdetails['password']
        building_name = userdetails['building_name']
        street_name = userdetails['street_name']
        city = userdetails['cityname']
        state = userdetails['statename']
        pin_code = userdetails['pincode']
        cur = mysql.connection.cursor()
        cur.execute("select max(customer_ID) from customers")
        ID = cur.fetchone()
        ID = str(int(ID[0]) + 1)
        cur.execute("select address_ID from address where building_name=%s and street_name=%s and pin_code=%s and city=%s and state=%s", (building_name, street_name, pin_code, city, state))
        address_ID = cur.fetchone()
        try:
            if (address_ID == None):
                cur.execute("select max(address_ID) from address")
                address_ID = cur.fetchone()
                address_ID = str(int(address_ID[0]) + 1)
                cur.execute("insert into address(address_ID, building_name, street_name, pin_code, city, state) values(%s, %s, %s, %s, %s, %s)", (building_name, street_name, pin_code, city, state))
                mysql.connection.commit()
            else:
                address_ID = address_ID[0]
                cur.execute("insert into customers(customer_ID, first_name, last_name, email, phone_no, password, DOB) values(%s, %s, %s, %s, %s, %s, %s)", (ID, firstname, lastname, email, phone_number, password, DOB))
                cur.execute("insert into customer_address(customer_ID, address_ID) values(%s, %s);", (ID, address_ID))
                mysql.connection.commit()
```

Adding restaurant address on the signup page.

```
@app.route('/signuprestaurant', methods=['GET', 'POST'])
def signuprestaurant():
    if request.method == 'POST':
        msg = 'Restaurant please fill out the form again'
        restdetail = request.form
        name = restdetail['name']
        email = restdetail['email']
        phoneno = restdetail['Phone number']
        password = restdetail['password']
        # Keeping weekend_time and weekday_time as null values
        # For now keeping rest_address from our side
        cur = mysql.connection.cursor()
        cur.execute('select max(restaurant_ID) from restaurant')
        ID = cur.fetchone()
        ID = str(int(ID[0]) + 1)
        building_name = restdetail['buildingnumber']
        street_name = restdetail['streetname']
        city = restdetail['cityname']
        state = restdetail['statename']
        pin_code = restdetail['pincode']
        cur.execute("select address_ID from address where building_name=%s and street_name=%s and pin_code=%s and city=%s and state=%s", (building_
        address_ID = cur.fetchone()
        try:
            if (address_ID == None):
                cur.execute("select max(address_ID) from address")
                address_ID = cur.fetchone()
                rest_address = str(int(address_ID[0]) + 1)
                cur.execute("insert into address(address_ID, building_name, street_name, pin_code, city, state) values(%s, %s, %s, %s, %s, %s)", (r
                mysql.connection.commit()
            else:
                rest_address = address_ID[0]
                cur.execute('insert into restaurant(restaurant_ID, name, email, phone_number, rest_address, password, weekday_time, weekend_time) value
        except:
```

Storing multiple addresses and allowing customers to choose a particular address for delivery.

```
cursor.execute("SELECT * FROM Customers WHERE customer_ID = %s", (ID,))
account = cursor.fetchone()
if account:
    cursor.execute("select a.address_ID, building_name, street_name, city, state, pin_code from address a inner join customer_address c on a.a
    addressCust = cursor.fetchall()
    addresses = []
    for address in addressCust:
        temp = {
            'address_ID':address[0],
            'building_no':address[1],
            'street_no':address[2],
            'city':address[3],
            'state':address[4],
            'pin_code':address[5]
        }
        addresses.append(temp)

    cursor.execute("SELECT Orders.order_ID, Orders.order_placed_time, Orders.order_status, restaurant.name as restaurant_name, order_totals.net
    orders_by_cust = cursor.fetchall()
    orders = []
    for placed_order in orders_by_cust:
        temp = {
            'ID': placed_order[0],
            'rest_name': placed_order[3],
            'time': placed_order[1].strftime('%Y-%m-%d %H:%M:%S'),
            'status': placed_order[2],
            'price': placed_order[4]
        }
        orders.append(temp)
    context = {
        "first_name":account[1],
        "last_name": account[2],
        "email": account[4],
```

We have added the opening time and closing time for the restaurants. We also provide the option to edit the email id and phone number of a restaurant. Moreover, we also allow restaurants to do multiple queries on their placed/completed orders based on today, last week and all.

```
# Update for assign 04

order_update_id = request.form.get('order_update')
if (order_update_id != None):
    cur.execute("update orders set order_status=%s where order_ID=%s", ("ready", order_update_id))
    mysql.connection.commit()

try:
    query = "SELECT Orders.order_ID, Orders.order_placed_time, Orders.order_status, order_totals.net_price FROM Orders inner join restaurant on Orders.restaurant_ID=restaurant.restaurant_ID"
    if request.method == 'POST' and 'email' in request.form and 'phone' in request.form:
        email = request.form['email']
        phone = request.form['phone']
        cur.execute("update restaurant set email=%s where restaurant_ID=%s", (email, rest_ID,))
        cur.execute("update restaurant set phone_number=%s where restaurant_ID=%s", (phone, rest_ID,))
        mysql.connection.commit()
        cur.execute(query, (rest_ID,))
    elif request.method == 'POST' and 'duration' in request.form:
        duration = request.form['duration']
        if(duration=="all"):
            cur.execute(query, (rest_ID,))
        elif(duration=="today"):
            duration = str(datetime.date.today())
            cur.execute("SELECT Orders.order_ID, Orders.order_placed_time, Orders.order_status, order_totals.net_price FROM Orders inner join restaurant on Orders.restaurant_ID=restaurant.restaurant_ID WHERE order_placed_time >=%s AND order_placed_time <=%s", (duration, duration))
        elif (duration=="lastweek"):
            base = datetime.datetime.today()
            week_time = str(base - datetime.timedelta(days=6)).split(" ")[0]
            cur.execute("SELECT Orders.order_ID, Orders.order_placed_time, Orders.order_status, order_totals.net_price FROM Orders inner join restaurant on Orders.restaurant_ID=restaurant.restaurant_ID WHERE order_placed_time >=%s AND order_placed_time <=%s", (week_time, week_time))
        else:
            cur.execute(query, (rest_ID,))
    orders_rest = cur.fetchall()
except:
    cur.execute(query, (rest_ID,))
    orders_rest = cur.fetchall()
# update ends for assign 04
```

3.3.1

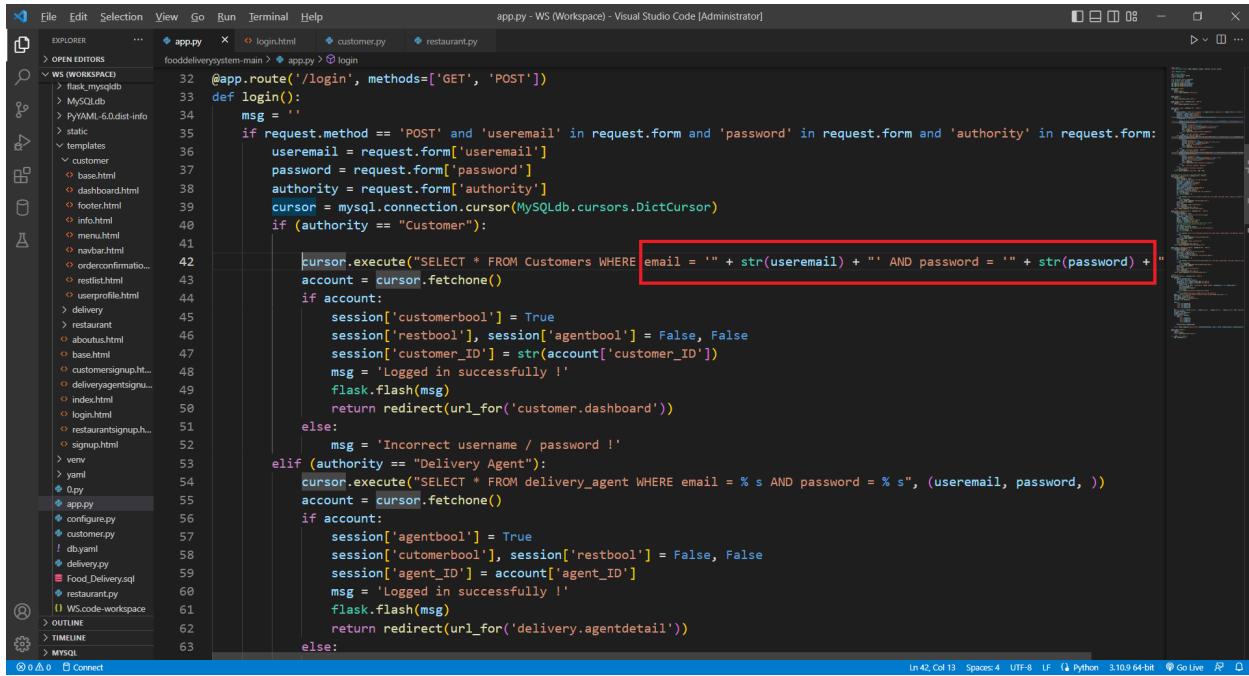
Attacks and Defenses

[Link](#) to the Folder containing Recordings/Snapshots of Attacks, Defenses, Results

1) Incorrect Queries (That fail on runtime):

- **Description:** This is an SQL Injection attack. We enter logically and grammatically correct query(s) into the field. However, we make sure that they will fail when we submit, i.e., at runtime. This will cause the system to throw an error, which will leave a trace that we will be able to see. From the traceback, we can extract the information about the back-end database system, like the name of the table, columns, rows' data, etc. We collect this info so that we can target some specific aspects of the database instead of the entire database. It can also be useful in building other attacks, more particularly.
- **Attack:**
 - Email : ' OR select * from nsdkjfnnfn --
 - Email : ' OR SELECT accounts FROM users WHERE login = convert ((select name from sysobjects where xtype='U' limit 1), unsigned integer)
- **Defense:** Do not allow single quotes (') or double quotes (") in the input fields, except the case that the input is just going to be stored by the database, and not going to be used in the query execution.

Before:

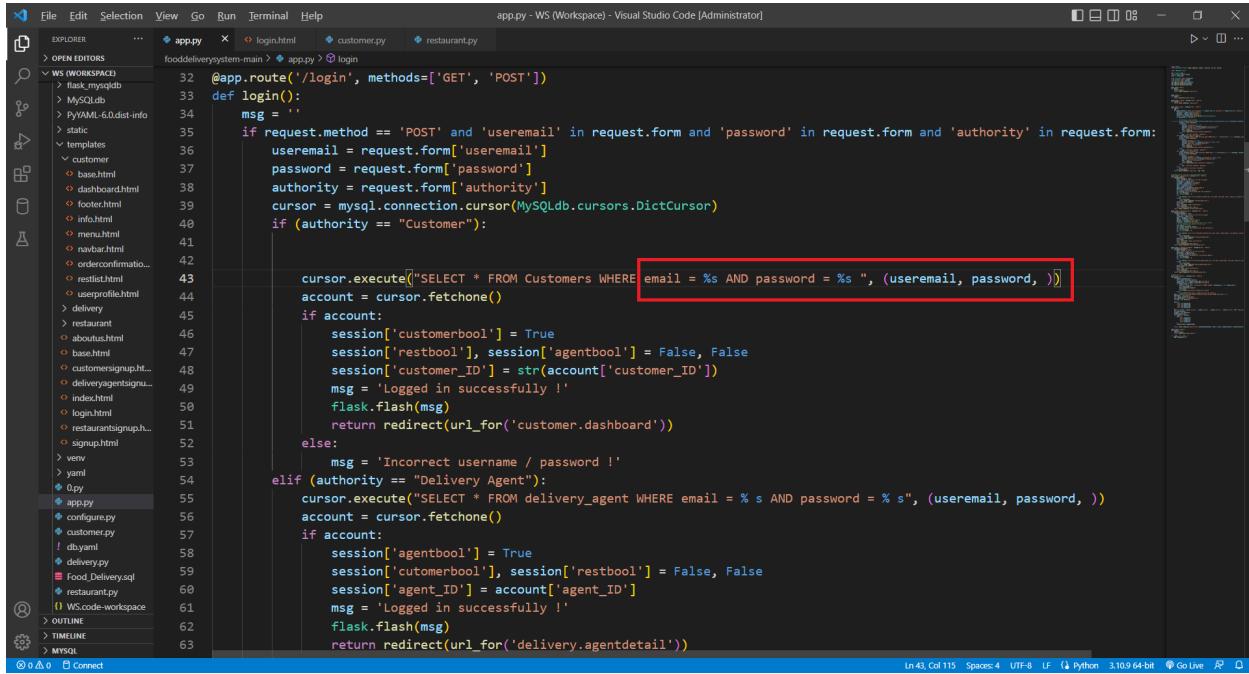


```

File Edit Selection View Go Run Terminal Help app.py - WS (Workspace) - Visual Studio Code [Administrator]
OPEN EDITORS
WS (WORKSPACE)
> flask_myqldb
> MySQLdb
> PyYAML-6.0.dist-info
> static
templates
customer
base.html
dashboard.html
footer.html
info.html
menu.html
navbars.html
orderconfirmation.html
restist.html
userprofile.html
> delivery
> restaurant
aboutus.html
base.html
customersignup.html
deliveryagentsignup.html
index.html
login.html
resturantsignup.html
signup.html
> venv
> yaml
0.py
app.py
configure.py
customer.py
! dbyaml
delivery.py
Food_Delivery.sql
restaurant.py
WS.code-workspace
OUTLINE
TIMELINE
MYSQL
Line 32: cursor.execute("SELECT * FROM Customers WHERE email = '" + str(useremail) + "' AND password = '" + str(password) + "'")
Line 33: account = cursor.fetchone()
Line 34: if account:
Line 35:     session['customerbool'] = True
Line 36:     session['restbool'], session['agentbool'] = False, False
Line 37:     session['customer_ID'] = str(account['customer_ID'])
Line 38:     msg = 'Logged in successfully !'
Line 39:     flask.flash(msg)
Line 40:     return redirect(url_for('customer.dashboard'))
Line 41: else:
Line 42:     msg = 'Incorrect username / password !'
Line 43: elif (authority == "Delivery Agent"):
Line 44:     cursor.execute("SELECT * FROM delivery_agent WHERE email = % s AND password = % s", (useremail, password))
Line 45:     account = cursor.fetchone()
Line 46:     if account:
Line 47:         session['agentbool'] = True
Line 48:         session['customerbool'], session['restbool'] = False, False
Line 49:         session['agent_ID'] = account['agent_ID']
Line 50:         msg = 'Logged in successfully !'
Line 51:         flask.flash(msg)
Line 52:         return redirect(url_for('delivery.agentdetail'))
Line 53:     else:
Line 54:         msg = 'Incorrect username / password !'
Line 55: elif (authority == "Delivery Agent"):
Line 56:     cursor.execute("SELECT * FROM delivery_agent WHERE email = % s AND password = % s", (useremail, password))
Line 57:     account = cursor.fetchone()
Line 58:     if account:
Line 59:         session['agentbool'] = True
Line 60:         session['customerbool'], session['restbool'] = False, False
Line 61:         session['agent_ID'] = account['agent_ID']
Line 62:         msg = 'Logged in successfully !'
Line 63:         flask.flash(msg)
Line 64:         return redirect(url_for('delivery.agentdetail'))

```

After:



```

File Edit Selection View Go Run Terminal Help app.py - WS (Workspace) - Visual Studio Code [Administrator]
OPEN EDITORS
WS (WORKSPACE)
> flask_myqldb
> MySQLdb
> PyYAML-6.0.dist-info
> static
templates
customer
base.html
dashboard.html
footer.html
info.html
menu.html
navbars.html
orderconfirmation.html
restist.html
userprofile.html
> delivery
> restaurant
aboutus.html
base.html
customersignup.html
deliveryagentsignup.html
index.html
login.html
resturantsignup.html
signup.html
> venv
> yaml
0.py
app.py
configure.py
customer.py
! dbyaml
delivery.py
Food_Delivery.sql
restaurant.py
WS.code-workspace
OUTLINE
TIMELINE
MYSQL
Line 32: cursor.execute("SELECT * FROM Customers WHERE email = %s AND password = %s", (useremail, password))
Line 33: account = cursor.fetchone()
Line 34: if account:
Line 35:     session['customerbool'] = True
Line 36:     session['restbool'], session['agentbool'] = False, False
Line 37:     session['customer_ID'] = str(account['customer_ID'])
Line 38:     msg = 'Logged in successfully !'
Line 39:     flask.flash(msg)
Line 40:     return redirect(url_for('customer.dashboard'))
Line 41: else:
Line 42:     msg = 'Incorrect username / password !'
Line 43: elif (authority == "Delivery Agent"):
Line 44:     cursor.execute("SELECT * FROM delivery_agent WHERE email = % s AND password = % s", (useremail, password))
Line 45:     account = cursor.fetchone()
Line 46:     if account:
Line 47:         session['agentbool'] = True
Line 48:         session['customerbool'], session['restbool'] = False, False
Line 49:         session['agent_ID'] = account['agent_ID']
Line 50:         msg = 'Logged in successfully !'
Line 51:         flask.flash(msg)
Line 52:         return redirect(url_for('delivery.agentdetail'))
Line 53:     else:
Line 54:         msg = 'Incorrect username / password !'
Line 55: elif (authority == "Delivery Agent"):
Line 56:     cursor.execute("SELECT * FROM delivery_agent WHERE email = % s AND password = % s", (useremail, password))
Line 57:     account = cursor.fetchone()
Line 58:     if account:
Line 59:         session['agentbool'] = True
Line 60:         session['customerbool'], session['restbool'] = False, False
Line 61:         session['agent_ID'] = account['agent_ID']
Line 62:         msg = 'Logged in successfully !'
Line 63:         flask.flash(msg)
Line 64:         return redirect(url_for('delivery.agentdetail'))

```

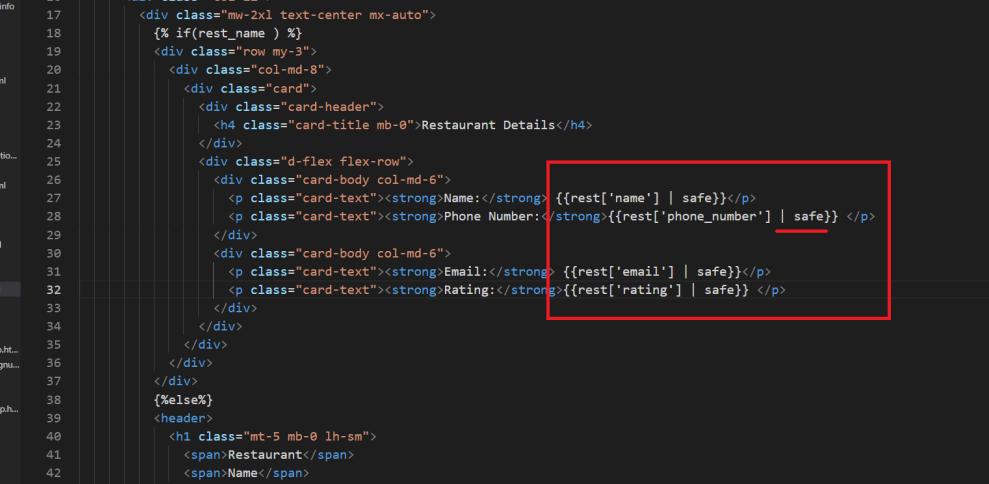
2) Cross Site Scripting (XSS) :

- **Description :** Cross-site scripting (XSS) is a type of security vulnerability that allows an attacker to inject malicious HTML/JavaScript code into the web page. The attacker can exploit a vulnerability in a web application by injecting a script or code that is executed when the user visits the compromised page. This can result in the theft of user data, such as cookies or sensitive information, or the execution of malicious actions on behalf of the user. In persistent XSS, the malicious code is stored in a server-side database, and every time the web page is loaded, the code is executed. In non-persistent XSS, the malicious code is injected into a URL or a form, and is executed when the user interacts with the affected page.

- **Attack :**
 - Adding a restaurant named as follow:
`<script> alert("YOUR ACCOUNT IS HACKED"); </script>`
 - If this name is added then when any other customer wants a restaurant list, this name is loaded in html from MySQL and the html code runs this as script instead of just displaying it as plain text.
 - This script will give alert if there is some other script written then the user can be forwarded to some other malicious website.

- **Defense :** We had used “safe” keyword-function in them html code-file for displaying the data collected from database, as {{rest['name'] | safe}}. This safe function considers the data from the database to be safe and doesn't use any filters. For defense against XSS, we removed all such “safe” functions. After this change the html will consider data from the database as a plain text and simply display it without running it as a script.

Before:



```
<div class="card-body col-md-6">
    <p class="card-text"><strong>Name:</strong> {{rest['name'] | safe}}</p>
    <p class="card-text"><strong>Phone Number:</strong> {{rest['phone_number'] | safe}} </p>
</div>
<div class="card-body col-md-6">
    <p class="card-text"><strong>Email:</strong> {{rest['email'] | safe}}</p>
    <p class="card-text"><strong>Rating:</strong> {{rest['rating'] | safe}}</p>
</div>
```

After:



```
File Edit Selection View Go Run Terminal Help             restdetail.html - WS (Workspace) - Visual Studio Code [Administrator]
EXPLORER ... app.py customer.py restaurant.py restdetail.html orderlist.html menu.html
OPEN EDITORS restdetail.html > section.pb-24-position-relative.overflow-hidden > div.container > div.row.mt-12 > div.col-12 > div.mw-2xText-center.mx-auto > div.row.my-3 > div.col-md-8 > div.card > div.d-flex.flex-row > div.card-body.col-md-6 > p.card-text
WS (WORKSPACE)
> flask_mySqlDB
> MySQLdb
> PyMySQL-0.6.0.dist-info
> static
> templates
    < customer
        > base.html
        > dashboard.html
        > footer.html
        > info.html
        > menu.html
        > navheader.html
        > orderconfirmation.html
        > restdetail.html
        > userprofile.html
    < delivery
    < restaurant
        > base.html
        > editmenu.html
        > menu.html
        > orderlist.html
        > restdetail.html
        > restnav.html
        > aboutus.html
        > base.html
        > customersignup.html
        > deliveryagentsignup.html
        > index.html
        > login.html
        > restaurantsignup.html
        > signup.html
    > ven
    > yaml
    > .py
    > app.py
    > configure.py
> OUTLINE
> TIMELINE
> MYSQL
    > Connect
18     {% if rest['rest_name'] %} 
19         <div class="row my-3">
20             <div class="col-md-8">
21                 <div class="card">
22                     <div class="card-header">
23                         <h4 class="card-title mb-0">Restaurant Details</h4>
24                     </div>
25                     <div class="d-flex flex-row">
26                         <div class="card-body col-md-6">
27                             <p class="card-text"><strong>Name:</strong> {{rest['name']}}</p>
28                             <p class="card-text"><strong>Phone Number:</strong> {{rest['phone_number']}}</p>
29                         </div>
30                         <div class="card-body col-md-6">
31                             <p class="card-text"><strong>Email:</strong> {{rest['email']}}</p>
32                             <p class="card-text"><strong>Rating:</strong> {{rest['rating']}}</p>
33                         </div>
34                     </div>
35                 </div>
36             </div>
37         </div>
38     {%else%}
39         <header>
40             <h1 class="mt-5 mb-0 lh-sm">
41                 <span>Restaurant</span>
42                 <span>Name</span>
43             </h1>
44         </header>
45     {%endif%}
46     </div>
47     </div>
48     </div>
49     </div>
50     </div>
```

3) Tautology :

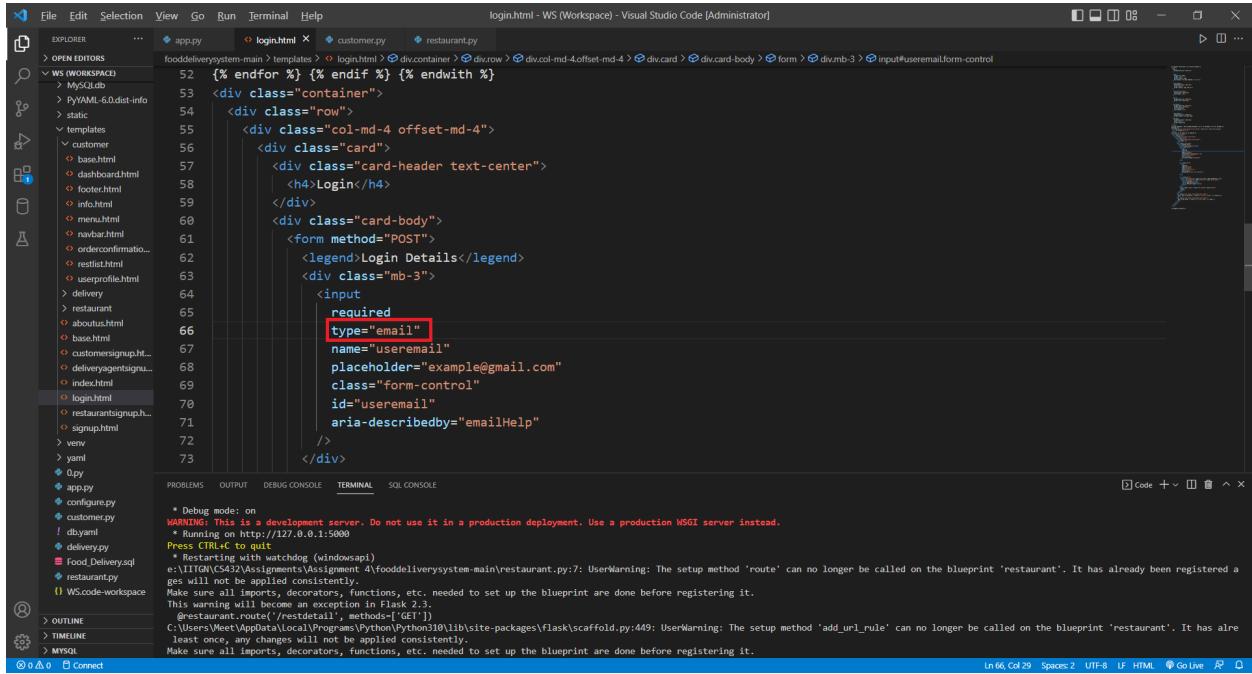
- **Description** : This is an SQL Injection attack. We write a query that always evaluates to TRUE, so that it will allow us to proceed anyways.
 - **Attack** :
 - Email : admin' OR '1'='1' --
 - **Defense** : Change the input field from text to email/phone_number, so that an email/number verification test is executed.

Before:

The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** Shows a tree view of the workspace. The "templates" folder contains subfolders for "customer" and "restaurant", each with their respective HTML files like base.html, dashboard.html, etc.
- Code Editor:** Displays the content of `app.py`. It includes imports for `flask`, `configparser`, `customer.py`, `delivery.py`, and `restaurant.py`. A warning message at the bottom states: "WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead." It also shows a `route` method being registered on the `restaurant` blueprint.
- Terminal:** Shows the command `python app.py` being run, outputting the warning message and the registration of the `route` method.
- Output:** Shows logs from the application, including a warning about the setup method for the `restaurant` blueprint.
- Problems:** Shows several errors related to the `route` method being registered multiple times.
- Terminal:** Shows the command `python app.py` being run, outputting the warning message and the registration of the `route` method.
- Output:** Shows logs from the application, including a warning about the setup method for the `restaurant` blueprint.
- Terminal:** Shows the command `python app.py` being run, outputting the warning message and the registration of the `route` method.
- Output:** Shows logs from the application, including a warning about the setup method for the `restaurant` blueprint.

After:



```

File Edit Selection View Go Run Terminal Help
WS (WORKSPACE) login.html - WS (Workspace) - Visual Studio Code [Administrator]
EXPLORER ... app.py login.html customers.py restaurant.py
WS (WORKSPACE)
> MySQLab
> PyYAML-6.0.dist-info
> static
> templates
> customer
> base.html
> dashboard.html
> footer.html
> info.html
> menu.html
> navbar.html
> orderconfirmation.html
> restlist.html
> userprofile.html
> delivery
> restaurant
> aboutus.html
> base.html
> customergroup.html
> deliveryagentgroup.html
> index.html
> login.html
> restauratnsignup.html
> signup.html
> venv
> yaml
> 0.py
app.py
configure.py
customer.py
! dbyaml
delivery.py
Food_Delivery.sql
restaurant.py
WS.code-space
OUTLINE
TIMELINE
MYSQL
File Explorer
Terminal
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL SQL CONSOLE
TERMINAL
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with watchdog (windosapi)
e:\ITGM\CS432\Assignments\Assignment 4\fooddeliverysystem-main\restaurant.py:7: UserWarning: The setup method 'route' can no longer be called on the blueprint 'restaurant'. It has already been registered a
ges will not be applied consistently.
Make sure all imports, decorators, functions, etc. needed to set up the blueprint are done before registering it.
The setup method 'route' can no longer be applied consistently in Flask 2.3
@restaurant.route('/restdetail', methods=['GET'])
C:\Users\Meet\AppData\Local\Programs\Python\Python38\lib\site-packages\flask\scaffold.py:449: UserWarning: The setup method 'add_url_rule' can no longer be called on the blueprint 'restaurant'. It has alre
ast once, any changes will not be applied consistently.
Make sure all imports, decorators, functions, etc. needed to set up the blueprint are done before registering it.
Ln 66, Col 29 Spaces: 2 UTF-8 LF, HTML Go Live

```

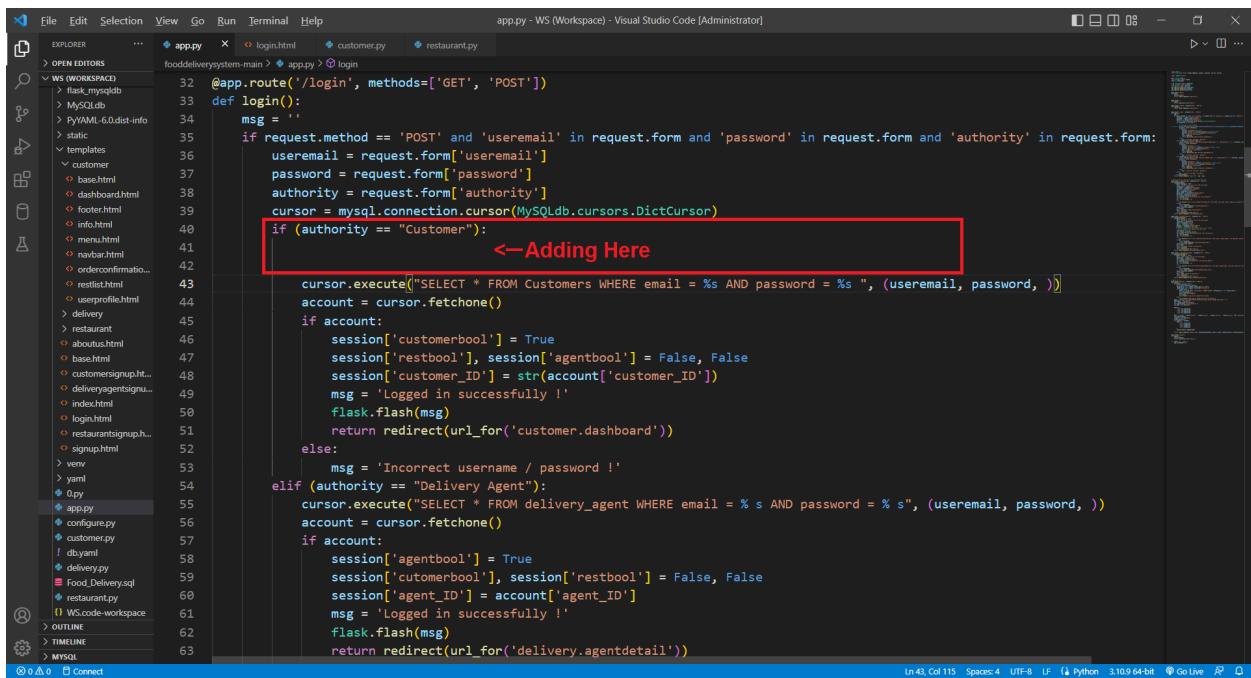
4) Union Query :

- Description :** This also is an SQL Injection attack. We do not care about the query that is already inside the database system. Instead, we finish the original query in a way that its output is NULL. Then, we put the UNION keyword, and write another query that we want to be executed. (Note : We use the information obtained from the first attack to write these queries.)
- Attack :**
 - Email : ' UNION select * from customers where cusomer_ID=5 --
 - Email : ' UNION select name, phone_no from customers --
 - Email : ' UNION drop table customers --
- Defense :** We can add both the defenses mentioned above. Instead of them, we can perform a quality check on the input obtained from the fields, about whether or not they contain any sub queries. We search for the keyword UNION and check if the following statement can be executed as another query. If so, we reject showing the output and throw an alert signal.

5) Piggy-Backed Queries :

- **Description** : This, too, is an SQL Injection attack. In this type of attack, we let the original query run as usual and correctly. But, we instead insert a new query after the original one. The original query is run and executed as usual. But, after that, the injected query is also run, which, thus, piggy-backs the original query.
- **Attack :**
 - Password : `some_pass_word' ; DROP TABLE customers ; --`
- **Defense** : We need to look for special characters in the input fields like semicolon(;), quotes(' and ") etc. Defenses from the previous attacks can also be used for this type of attack.

Before:



```

File Edit Selection View Go Run Terminal Help app.py - WS (Workspace) - Visual Studio Code [Administrator]
EXPLORER WS (WORKSPACE)
> flask_myqldb
> MySQLdb
> PyYAML-6.0.dist-info
> static
> templates
  > customer
    > base.html
    > dashboard.html
    > footer.html
    > info.html
    > menu.html
    > navbar.html
    > orderconfirmation.html
    > restishml
    > userprofile.html
> delivery
> restaurant
> aboutus.html
> base.html
> customersignup.html
> deliveryagentsignup.html
> index.html
> login.html
> restaurantsignup.html
> signup.html
> venv
> yaml
app.py
configure.py
customer.py
! dbyaml
delivery.py
Food_Delivery.sql
restaurant.py
WS.code-workspace
OUTLINE
TIMELINE
MySQL

```

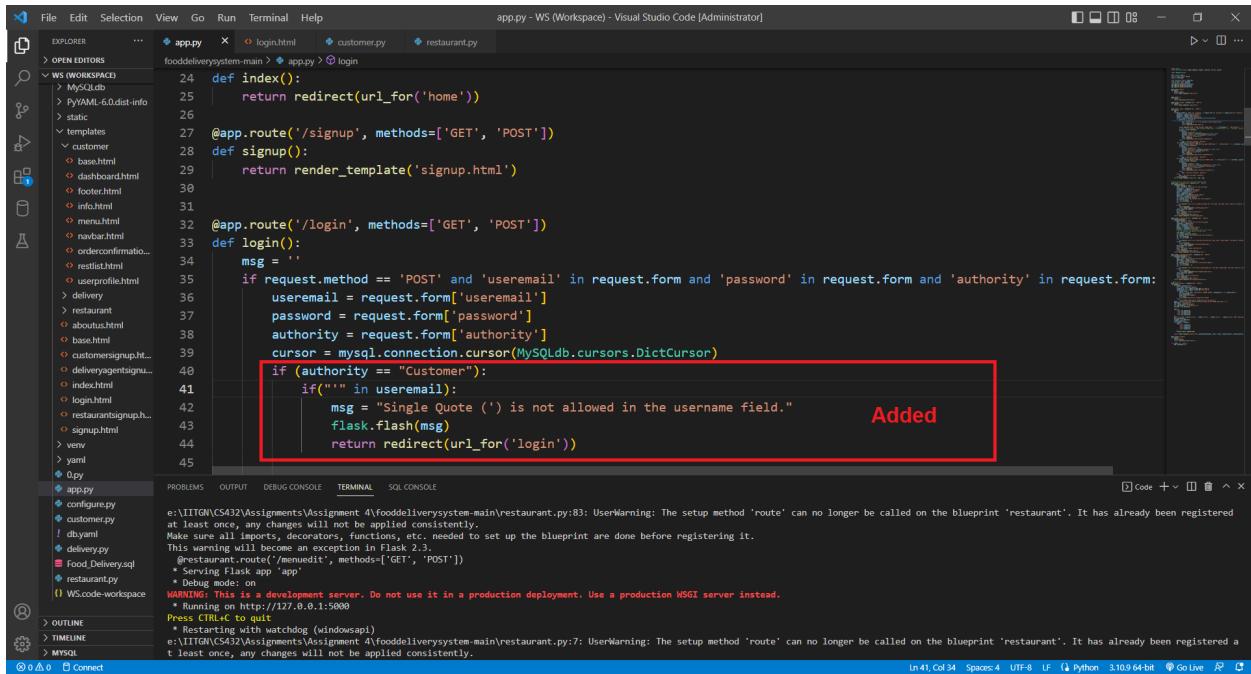
```

32 @app.route('/login', methods=['GET', 'POST'])
33 def login():
34     msg = ''
35     if request.method == 'POST' and 'useremail' in request.form and 'password' in request.form and 'authority' in request.form:
36         useremail = request.form['useremail']
37         password = request.form['password']
38         authority = request.form['authority']
39         cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)
40         if (authority == "Customer"):
41             cursor.execute("SELECT * FROM Customers WHERE email = %s AND password = %s", (useremail, password, ))
42             account = cursor.fetchone()
43             if account:
44                 session['customerbool'] = True
45                 session['restbool'], session['agentbool'] = False, False
46                 session['customer_ID'] = str(account['customer_ID'])
47                 msg = 'Logged in successfully !'
48                 flask.flash(msg)
49                 return redirect(url_for('customer.dashboard'))
50             else:
51                 msg = 'Incorrect username / password !'
52         elif (authority == "Delivery Agent"):
53             cursor.execute("SELECT * FROM delivery_agent WHERE email = % s AND password = % s", (useremail, password, ))
54             account = cursor.fetchone()
55             if account:
56                 session['agentbool'] = True
57                 session['customerbool'], session['restbool'] = False, False
58                 session['agent_ID'] = account['agent_ID']
59                 msg = 'Logged in successfully !'
60                 flask.flash(msg)
61                 return redirect(url_for('delivery.agentdetail'))
62
63

```

In 43, Col 115 Spaces:4 UTF-8 LF Python 3.10.9 64-bit Go Live

After:



```

24 def index():
25     return redirect(url_for('home'))
26
27 @app.route('/signup', methods=['GET', 'POST'])
28 def signup():
29     return render_template('signup.html')
30
31
32 @app.route('/login', methods=['GET', 'POST'])
33 def login():
34     msg = ''
35     if request.method == 'POST' and 'useremail' in request.form and 'password' in request.form and 'authority' in request.form:
36         useremail = request.form['useremail']
37         password = request.form['password']
38         authority = request.form['authority']
39         cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)
40         if (authority == "Customer"):
41             if("'" in useremail):
42                 msg = "Single Quote ('') is not allowed in the username field."
43                 flask.flash(msg)
44             return redirect(url_for('login'))
45

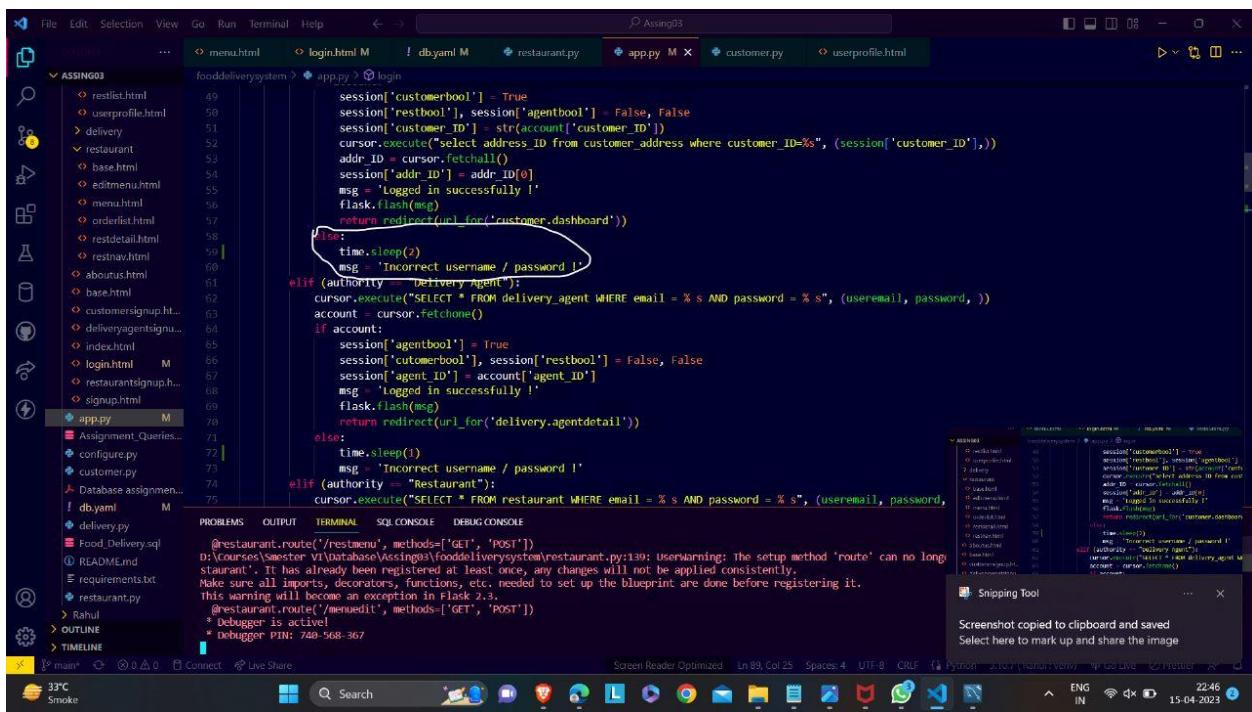
```

Added

6) Brute Force :

- **Description :** A brute force attack is a type of cyber attack where an attacker attempts to gain unauthorized access to a system or data via trying every possible password or encryption key until the correct one is found.
- **Attack :**
 - For some valid username we will try all possible password combinations. If the possible combinations are very less or if the user has a weak password then we can brute force the password.
 - We have used “hydra tool” for this purpose.
 - We had username.txt and password.txt as possible passwords and usernames.
 - `$ hydra -l username.txt -p password.txt 127.0.0.1 8000 -o safe.txt`
 - This code will go through combinations of username and passwords and return correct combinations in the safe.txt file.

- **Defense** : To defend against brute force attacks we need to make passwords stronger. For this we can force the users to choose a password which has length more than 8 or 10 characters. In addition to that, we can impose a constraint that those passwords should include at least one unit of the following : capital letter, small letter, digit and special character. We have also added 3 seconds delay between submit button click and the displaying of the same webpage, in case the input username and password are incorrect. After this, the brute force attack will take 3 seconds per check. In this way, we have optimized our defense against brute force attacks.



```

File Edit Selection View Go Run Terminal Help ... ASSING03 login.html M db.yaml M restaurant.py app.py M customer.py userprofile.html
ASSING03
foodeliverysystem > app.py > login
foodeliverysystem > app.py > login
session['customerbool'] = True
session['restbool'], session['agentbool'] = False, False
session['customer_ID'] = str(account['customer_ID'])
cursor.execute("select address_ID from customer_address where customer_ID=%s", (session['customer_ID'],))
addr_ID = cursor.fetchall()
session['addr_ID'] = addr_ID[0]
msg = 'Logged in successfully !'
flask.flash(msg)
return redirect(url_for('customer.dashboard'))
else:
    time.sleep(2)
    msg = 'Incorrect username / password !'
elif (authority == 'Delivery Agent'):
    cursor.execute("SELECT * FROM delivery_agent WHERE email = % s AND password = % s", (useremail, password, ))
    account = cursor.fetchone()
    if account:
        session['agentbool'] = True
        session['customerbool'] = False, False
        session['agent_ID'] = account['agent_ID']
        msg = 'Logged in successfully !'
        flask.flash(msg)
        return redirect(url_for('delivery.agentdetail'))
    else:
        time.sleep(1)
        msg = 'Incorrect username / password !'
elif (authority == "Restaurant"):
    cursor.execute("SELECT * FROM restaurant WHERE email = % s AND password = % s", (useremail, password, ))
    account = cursor.fetchone()
    if account:
        session['customerbool'] = True
        session['restbool'] = False, False
        session['customer_ID'] = account['customer_ID']
        msg = 'Logged in successfully !'
        flask.flash(msg)
        return redirect(url_for('customer.dashboard'))
    else:
        time.sleep(1)
        msg = 'Incorrect username / password !'
else:
    time.sleep(1)
    msg = 'Incorrect username / password !'
if account:
    session['customerbool'] = True
    session['restbool'], session['agentbool'] = False, False
    session['customer_ID'] = str(account['customer_ID'])
    cursor.execute("select address_ID from customer_address where customer_ID=%s", (session['customer_ID'],))
    addr_ID = cursor.fetchall()
    session['addr_ID'] = addr_ID[0]
    msg = 'Logged in successfully !'
    flask.flash(msg)
    return redirect(url_for('customer.dashboard'))
else:
    msg = 'Incorrect username / password !'
    flask.flash(msg)
    return redirect(url_for('customer.login'))

```

The screenshot shows a code editor with Python code for a login system. The code uses sessions to track user roles (customer, delivery agent, or restaurant) and handles user input with a 2-second delay via `time.sleep(2)`. It queries a MySQL database for user credentials and logs them in if successful. The code is part of a Flask application, as indicated by the `flask.flash` and `redirect` calls. A circled portion of the code highlights the `time.sleep(2)` line, which is used to prevent brute-force attacks by delaying password checks.

7) Frontend attack:

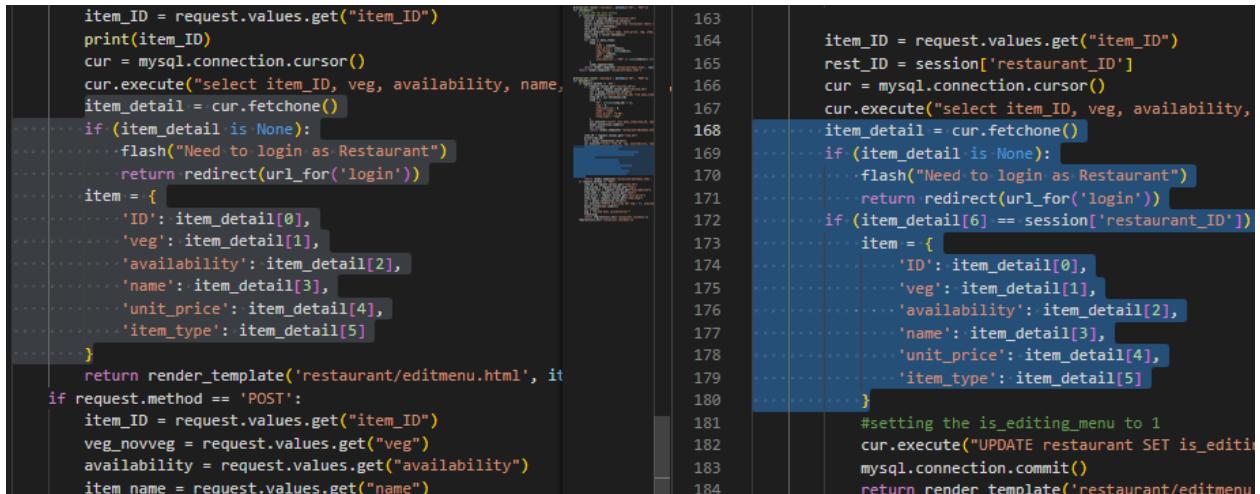
- When the restaurant menu goes to edit a certain menu item then through playing with url a restaurant is able to change the details of any menu item in the database.
- We prevented it by checking whether the menu_item belongs to the current restaurant_ID stored in session details.
- We flash up the message if the session's restaurant id doesn't match with the one related to the menu_item being edited.

Location of vulnerability:

The screenshot shows a web browser window with the following details:

- URL Bar:** 127.0.0.1:5000/menuedit?item_ID=1
- Page Title:** Restaurant Detail
- Form Fields:**
 - Availability: 0
 - Item Name: Balu shahi
 - Item Price: 256.00
 - Type: dessert
- Buttons:** A blue "SAVE CHANGES" button.
- Page Bottom:** © 2023 Copyright@DatabaseGroup Home Menu Aboutus

Before and after:



```

item_ID = request.values.get("item_ID")
print(item_ID)
cur = mysql.connection.cursor()
cur.execute("select item_ID, veg, availability, name,
item_detail = cur.fetchone()
if (item_detail is None):
    flash("Need to login as Restaurant")
    return redirect(url_for('login'))
item = {
    'ID': item_detail[0],
    'veg': item_detail[1],
    'availability': item_detail[2],
    'name': item_detail[3],
    'unit_price': item_detail[4],
    'item_type': item_detail[5]
}
return render_template('restaurant/editmenu.html', it
if request.method == 'POST':
    item_ID = request.values.get("item_ID")
    veg_novveg = request.values.get("veg")
    availability = request.values.get("availability")
    item.name = request.values.get("name")

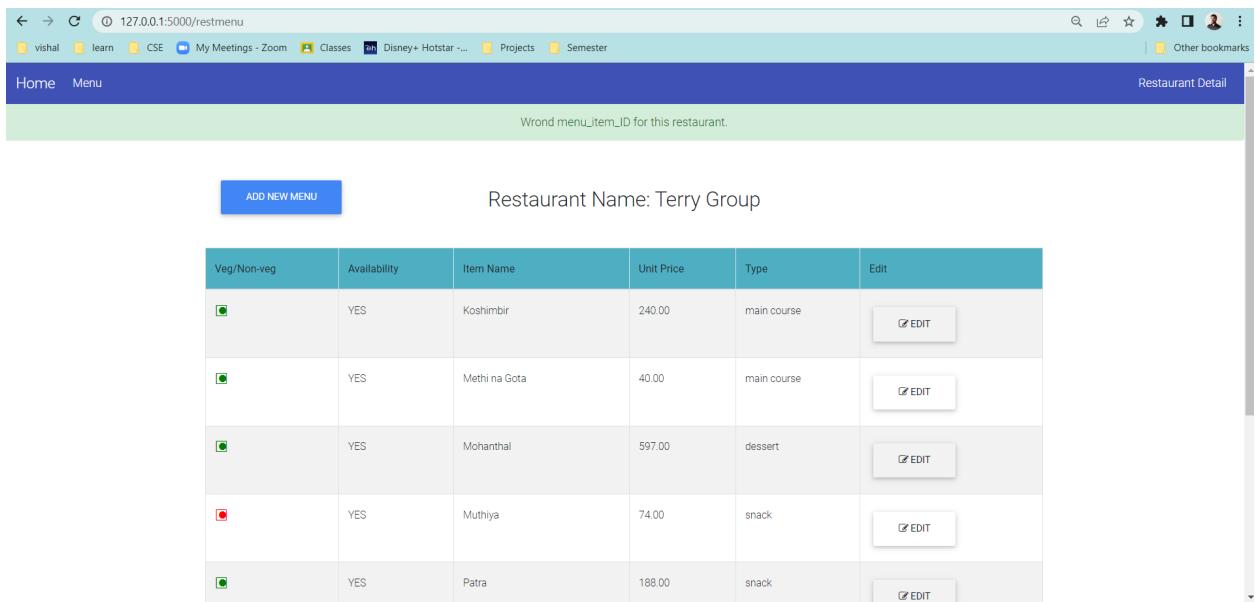
```

```

163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184

item_ID = request.values.get("item_ID")
rest_ID = session['restaurant_ID']
cur = mysql.connection.cursor()
cur.execute("select item_ID, veg, availability,
item_detail = cur.fetchone()
if (item_detail is None):
    flash("Need to login as Restaurant")
    return redirect(url_for('login'))
if (item_detail[6] == session['restaurant_ID']):
    item = {
        'ID': item_detail[0],
        'veg': item_detail[1],
        'availability': item_detail[2],
        'name': item_detail[3],
        'unit_price': item_detail[4],
        'item_type': item_detail[5]
    }
    #setting the is_editing_menu to 1
    cur.execute("UPDATE restaurant SET is_editing_menu = 1")
    mysql.connection.commit()
return render_template('restaurant/editmenu.html', it

```



Veg/Non-veg	Availability	Item Name	Unit Price	Type	Edit
<input checked="" type="checkbox"/>	YES	Koshimbir	240.00	main course	<input checked="" type="button"/> EDIT
<input checked="" type="checkbox"/>	YES	Methi na Gota	40.00	main course	<input checked="" type="button"/> EDIT
<input checked="" type="checkbox"/>	YES	Mohanthal	597.00	dessert	<input checked="" type="button"/> EDIT
<input checked="" type="checkbox"/>	YES	Muthiya	74.00	snack	<input checked="" type="button"/> EDIT
<input checked="" type="checkbox"/>	YES	Patra	188.00	snack	<input checked="" type="button"/> EDIT

3.3.2

These are constraints in our database management system that enforce rules and restrictions on the data stored in the database tables. These we have implemented in the submitted SQL-dump. They hold true even after the changes implemented after the second feedback.

1. delivery_rating: This constraint ensures that the delivery rating given by the customer is between 0 and 5 (inclusive).
2. delivery_charges: This constraint ensures that the delivery charges for an order are non-negative (i.e., greater than or equal to 0).
3. delivery_time: This constraint ensures that the delivery time for an order is greater than the pickup time.
4. rating: This constraint ensures that the restaurant rating the customer gives is between 0 and 5 (inclusive).
5. payment_method: This constraint ensures that the payment method used for an order is one of the allowed options (UPI, Netbanking, Debit Card, or Credit Card).
6. amount: This constraint ensures that the amount paid for an order is greater than 0.
7. order_status: This constraint ensures that the order status is one of the allowed options (placed, ready, picked, or delivered). This is the only change committed in our database from the previous version as we have added the option to have the statues “ready”.
8. order_rating: This constraint ensures that the order rating given by the customer is between 0 and 5 (inclusive).

Contributions:

Index	Tasks	Members taking responsibility
1	Getting Feedback	Vishal, Pratik and Juhil
2	Implementing the feedback	Rahul and Vishal
3	SQL injection and XSS and frontend attacks	Prey, Meet and Tejas
4	Prevention of concurrent transaction	Dhruv, Prakram
5	Checking constraints and relations after second feedback	Juhil, Dhiren
6	Documentation	Dablu and Dhiren