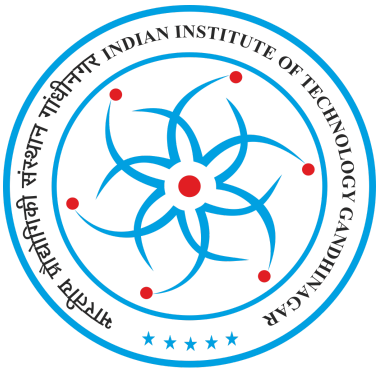


IITGN Guest House Management System (GHMS)

Assignment 4: Deploying the DBMS

Team: First Order

Gaurav Shah - 21110064

Soham Rahatal - 21110173

Pratik Agarwal - 21110166

Rohit Srivastav - 21110180

Banavath Diraj Naik - 22110044

Sohitha Sonalika - 22110151

Shivamani - 22110062

3.1 Responsibility of G1:

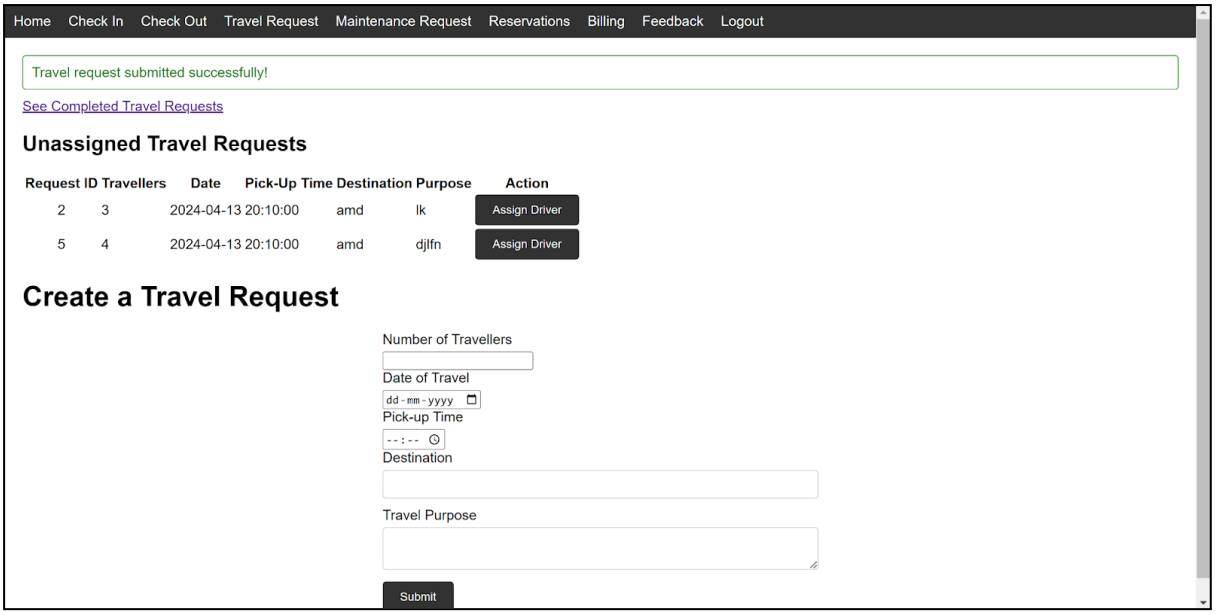
1.1 Initial Feedback

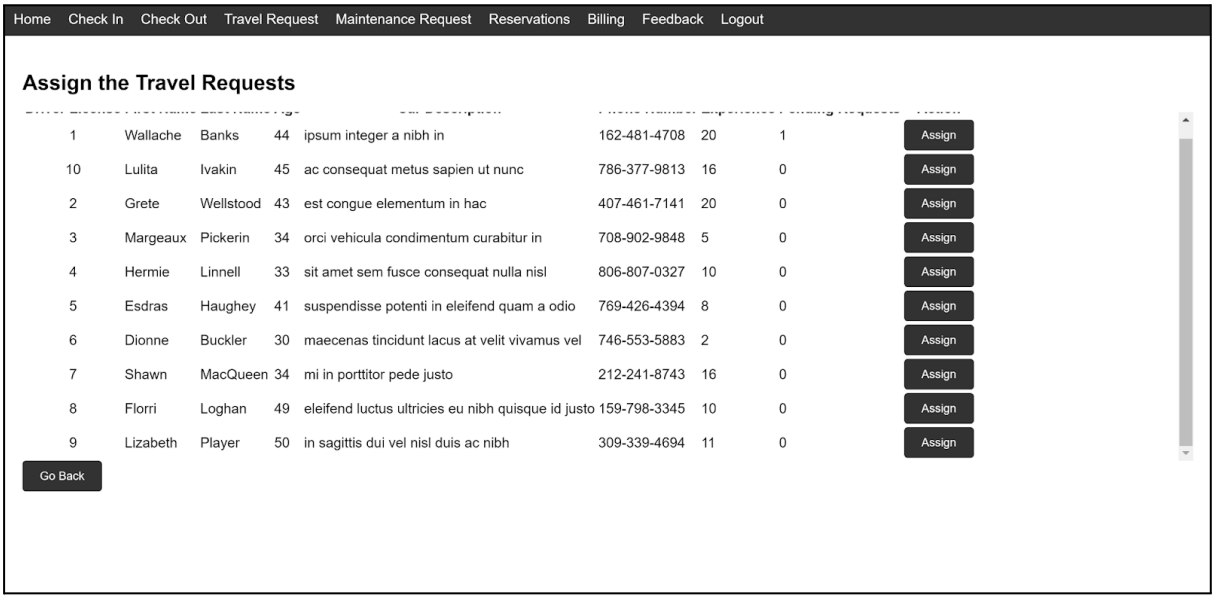
From the initial feedback by M. Yashwant Chouhan from the Guesthouse, we have added the following functionalities to our WebApp:

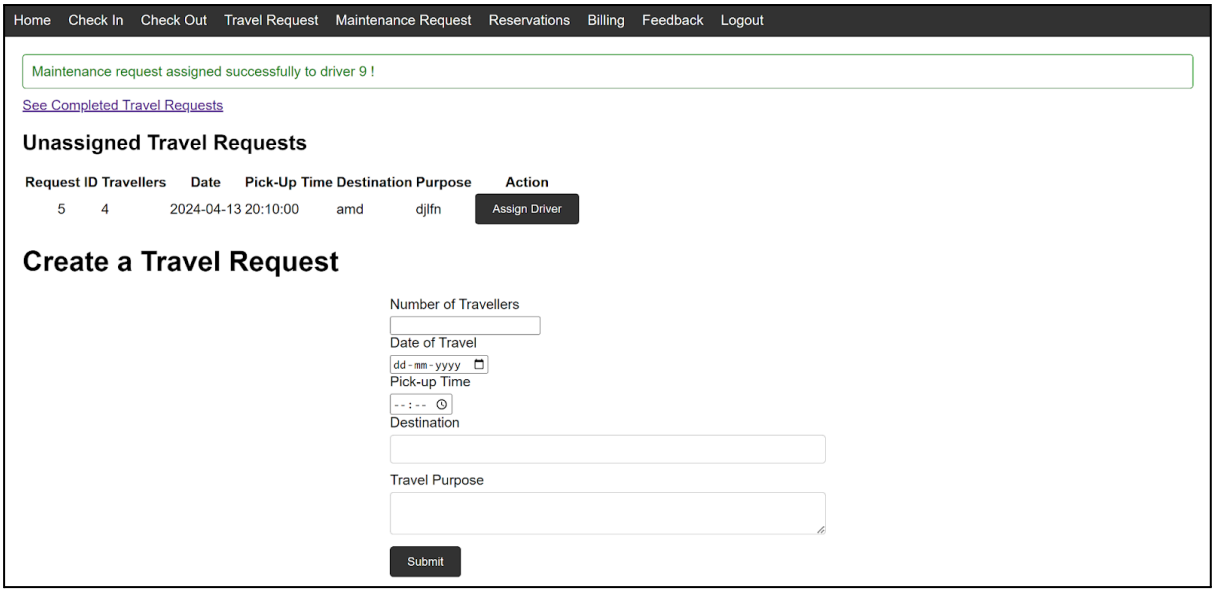
I. Added changes to the Travel Request Management part of the admin dashboard. In the first version of the web app, when a travel request is raised by the hospitality staff or the current guest, it gets displayed as an unassigned request on the 'Travel Request' page of the hospitality_staff dashboard until the driver is not assigned. Then, one of the admin users assigns the request to one of the drivers by looking at his/her records and the pending requests. This version showed the drivers on the available list even when another pending request was assigned at the exact pick-up time and date. We have fixed this by updating the SQL query, initially fetching the list of available drivers by fetching the drivers that don't have a pending request $\pm$ 6 *hours* to the given request.

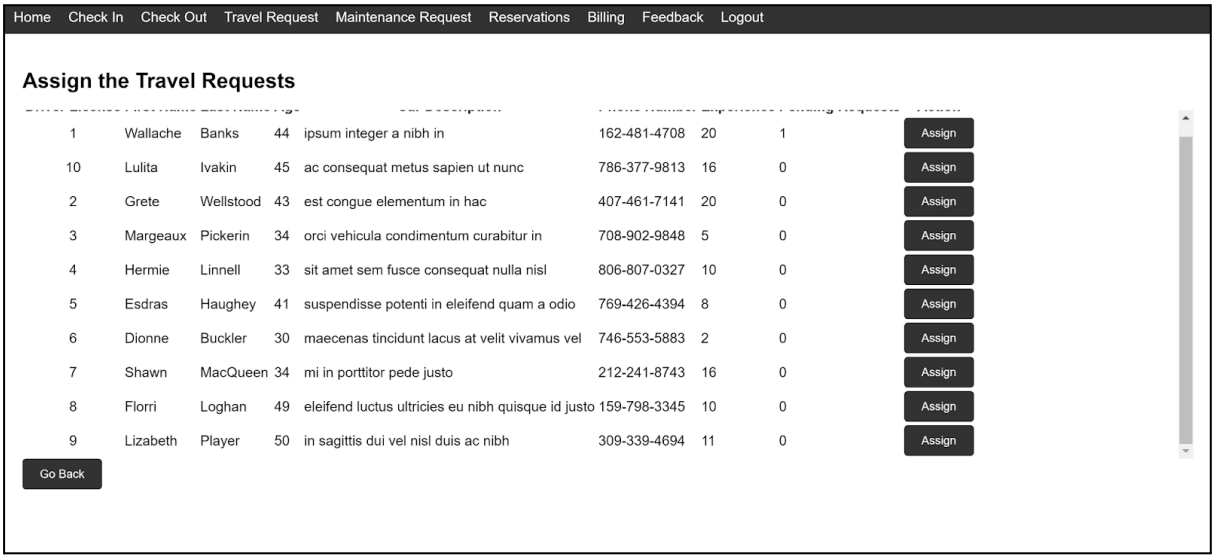
```
    driver_available = []
    for drivers in driver_all:
        pending_requests =
travel_request.query.filter(travel_request.driver_license ==
drivers.driver_license, travel_request.date_of_travel >=
request.date_of_travel).all()
        l = len(pending_requests)
        i = 0
        for pending_request in pending_requests:
            if pending_request.date_of_travel == request.date_of_travel and
abs(request.pick_up_time.hour - request.pick_up_time.hour) < 6:
                print(pending_request)
                break
            i += 1
        if i == l:
            driver_available.append(drivers)
    print(driver_available)
```

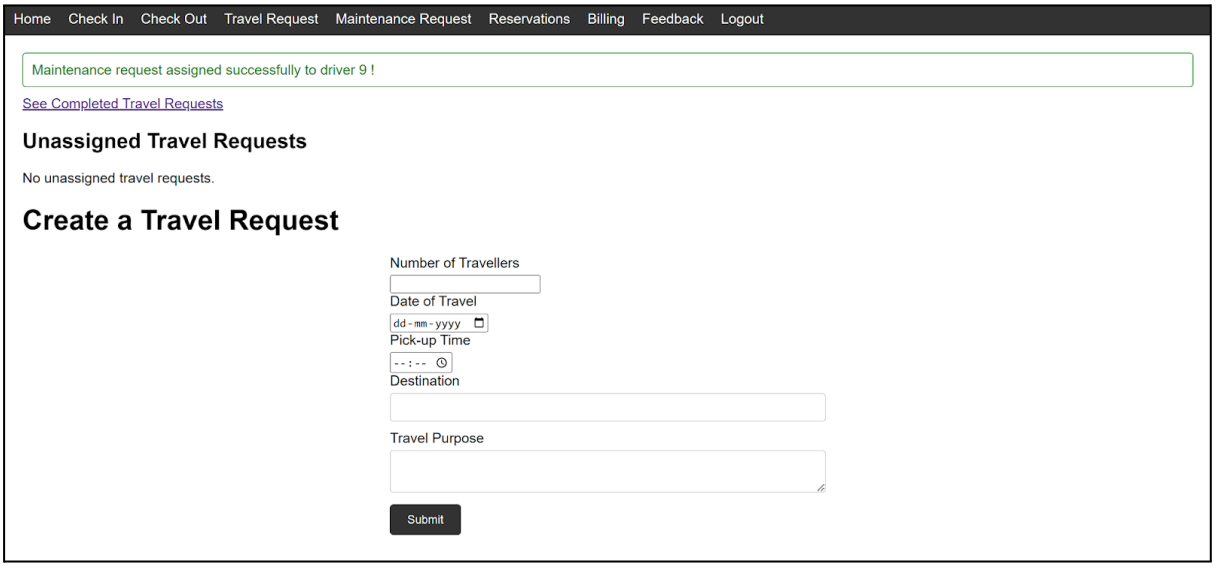
Before:


Home Check In Check Out Travel Request Maintenance Request Reservations Billing Feedback Logout
Travel request submitted successfully!
See Completed Travel Requests
Unassigned Travel Requests
Request ID Travellers Date Pick-Up Time Destination Purpose Action
2 3 2024-04-13 20:10:00 amd lk Assign Driver
5 4 2024-04-13 20:10:00 amd djlfn Assign Driver
Create a Travel Request
Number of Travellers
Date of Travel
dd-mm-yyyy
Pick-up Time
--:--
Destination
Travel Purpose
Submit


Home Check In Check Out Travel Request Maintenance Request Reservations Billing Feedback Logout
Assign the Travel Requests
1 Wallache Banks 44 ipsum integer a nibh in 162-481-4708 20 1 Assign
10 Lulita Ivakin 45 ac consequat metus sapien ut nunc 786-377-9813 16 0 Assign
2 Grete Wellstood 43 est congue elementum in hac 407-461-7141 20 0 Assign
3 Margeaux Pickerin 34 orci vehicula condimentum curabitur in 708-902-9848 5 0 Assign
4 Hermie Linnell 33 sit amet sem fusce consequat nulla nisl 806-807-0327 10 0 Assign
5 Esdras Haughey 41 suspendisse potenti in eleifend quam a odio 769-426-4394 8 0 Assign
6 Dionne Buckler 30 maecenas tincidunt lacus at velit vivamus vel 746-553-5883 2 0 Assign
7 Shawn MacQueen 34 mi in porttitor pede justo 212-241-8743 16 0 Assign
8 Florri Loghan 49 eleifend luctus ultricies eu nibh quisque id justo 159-798-3345 10 0 Assign
9 Lizabeth Player 50 in sagittis dui vel nisl duis ac nibh 309-339-4694 11 0 Assign
Go Back

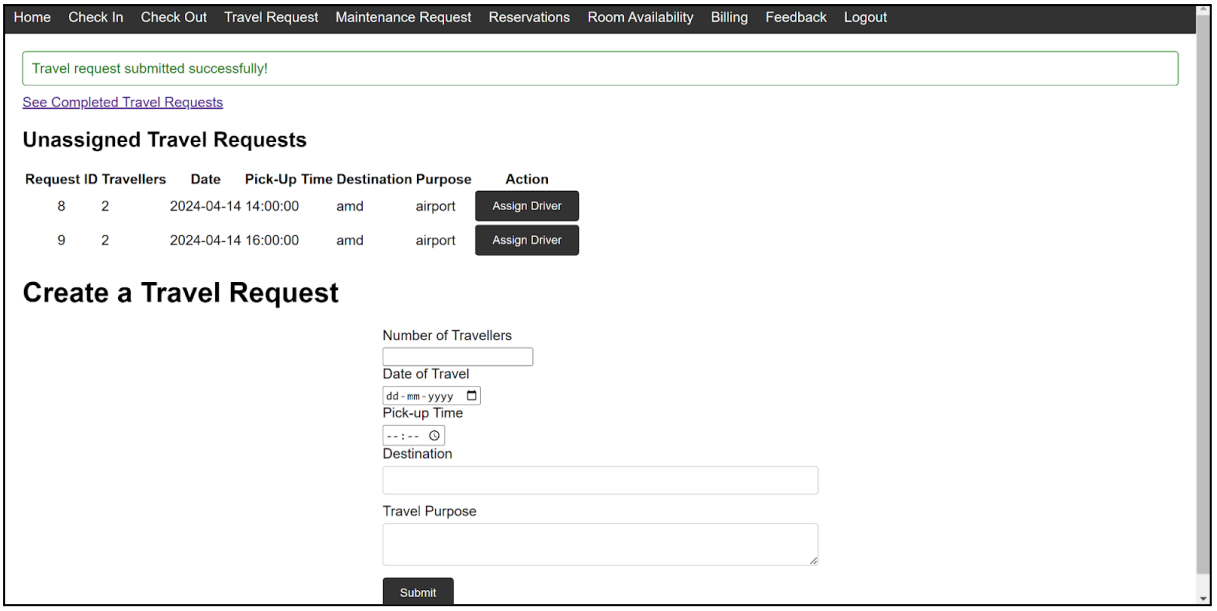


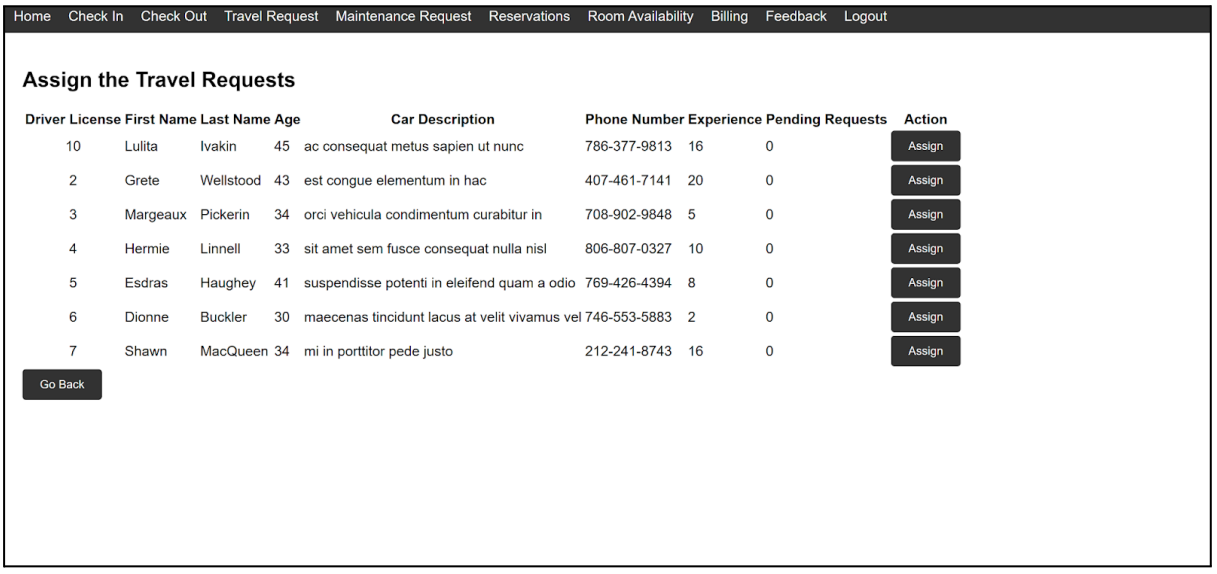


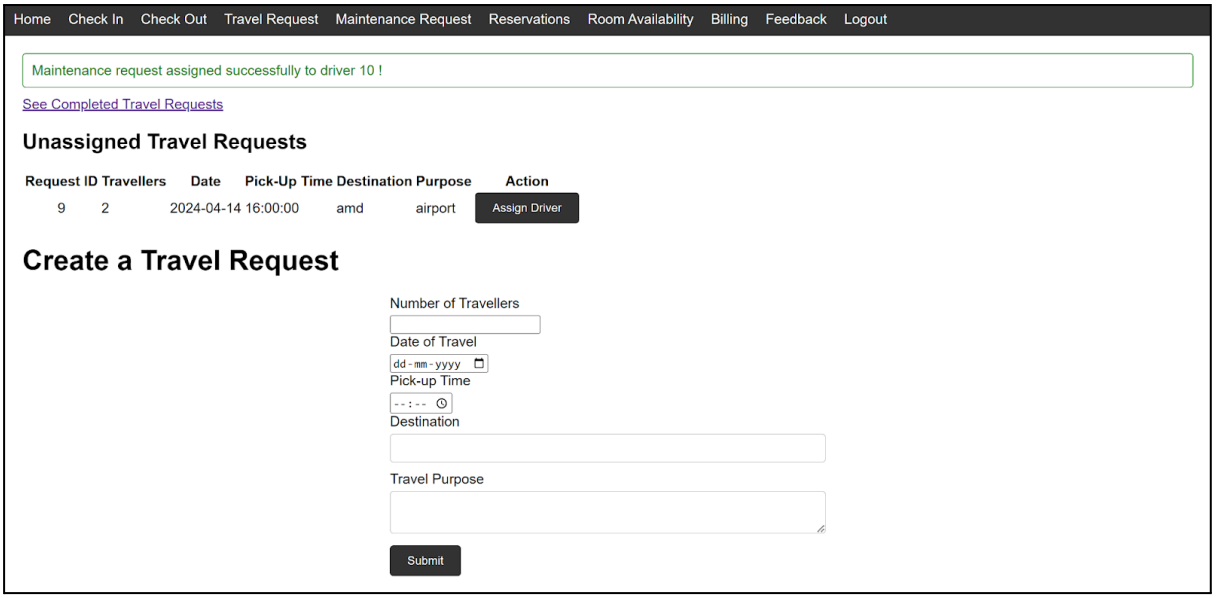


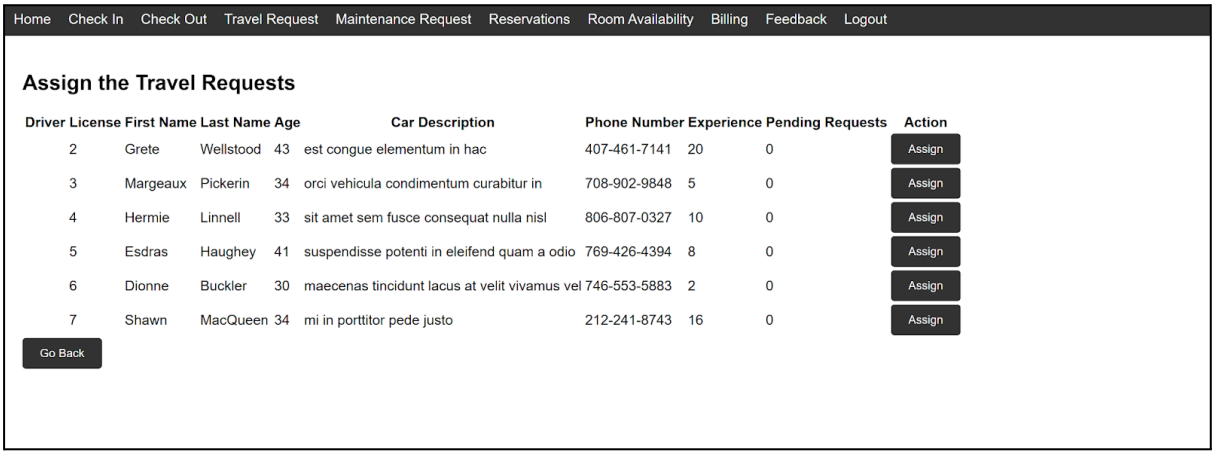
Both travel requests get assigned to driver 9.

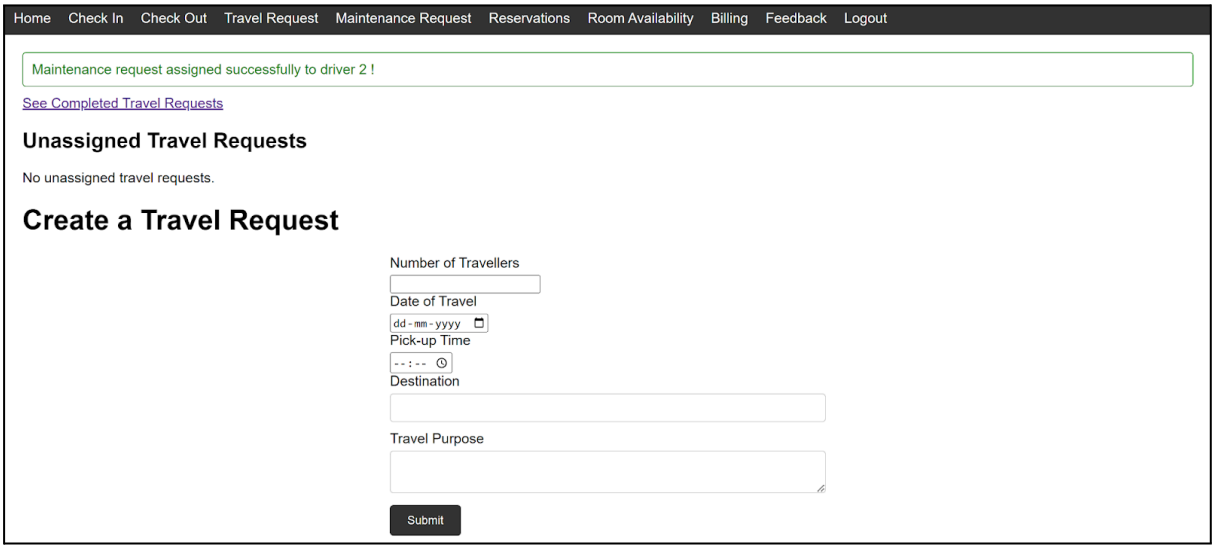
After:


Home Check In Check Out Travel Request Maintenance Request Reservations Room Availability Billing Feedback Logout
Travel request submitted successfully!
See Completed Travel Requests
Unassigned Travel Requests
Request ID Travellers Date Pick-Up Time Destination Purpose Action
8 2 2024-04-14 14:00:00 amd airport Assign Driver
9 2 2024-04-14 16:00:00 amd airport Assign Driver
Create a Travel Request
Number of Travellers
Date of Travel
dd-mm-yyyy
Pick-up Time
--:--
Destination
Travel Purpose
Submit


Home Check In Check Out Travel Request Maintenance Request Reservations Room Availability Billing Feedback Logout
Assign the Travel Requests
Driver License First Name Last Name Age Car Description Phone Number Experience Pending Requests Action
10 Lulita Ivakin 45 ac consequat metus sapien ut nunc 786-377-9813 16 0 Assign
2 Grete Wellstood 43 est congue elementum in hac 407-461-7141 20 0 Assign
3 Margeaux Pickerin 34 orci vehicula condimentum curabitur in 708-902-9848 5 0 Assign
4 Hermie Linnell 33 sit amet sem fusce consequat nulla nisl 806-807-0327 10 0 Assign
5 Esdras Haughey 41 suspendisse potenti in eleifend quam a odio 769-426-4394 8 0 Assign
6 Dionne Buckler 30 maecenas tincidunt lacus at velit vivamus vel 746-553-5883 2 0 Assign
7 Shawn MacQueen 34 mi in porttitor pede justo 212-241-8743 16 0 Assign
Go Back





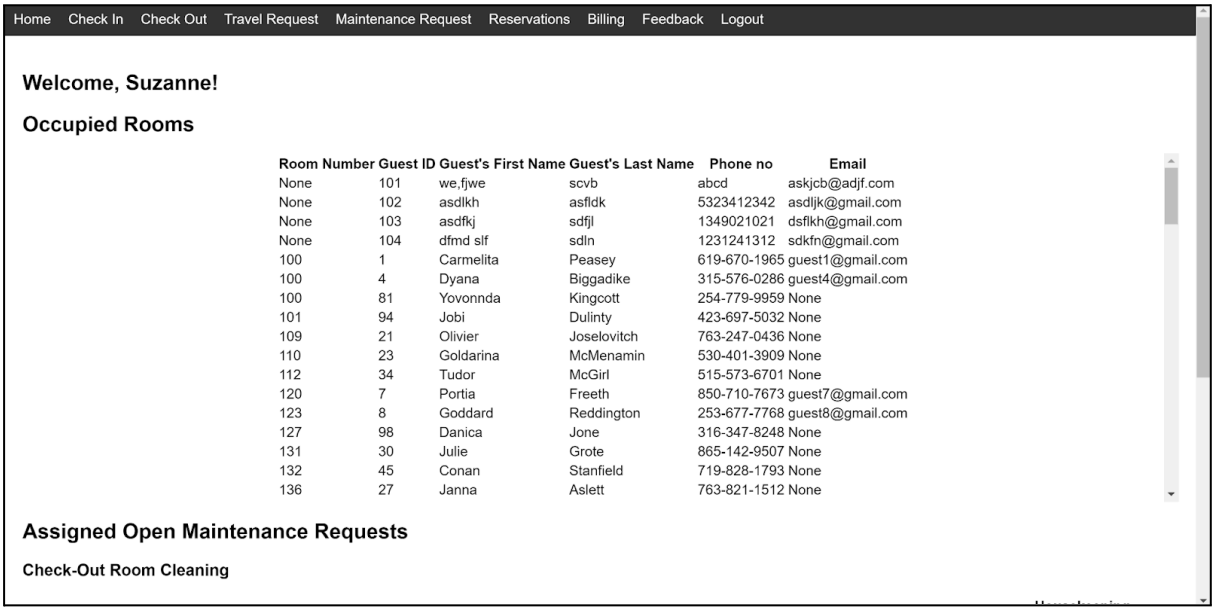


As the first request is assigned to driver 10, and the second request's pick-up time is within one hour of the first request, driver 10 is not on the list of available drivers on the driver assignment page. Therefore, driver two is selected.

II. We have also added a new webpage called 'Room Availability' in the hospitality_staff dashboard, enabling the staff to check the room availability of the rooms in the guesthouse. This will help the staff not to underbook or overbook reservations. Staff can enter a date on which he/she wants to see the room availability and then will be redirected to a new page where they can see the number of rooms available by type. They can't see exactly which room (by its number) is available as the room assignment is done at check-in and not while reserving.

Implemented by creating new routes in admin.py, writing the necessary SQL queries, and creating the necessary HTML pages.

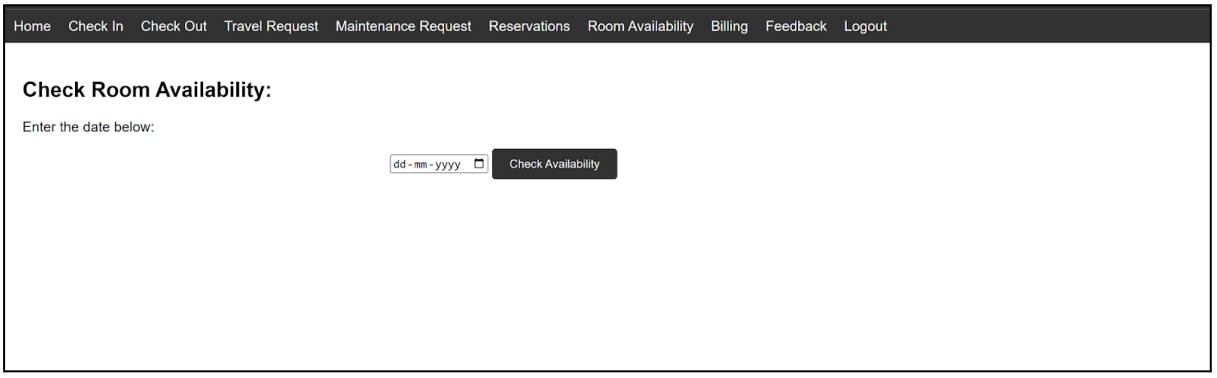
Before:

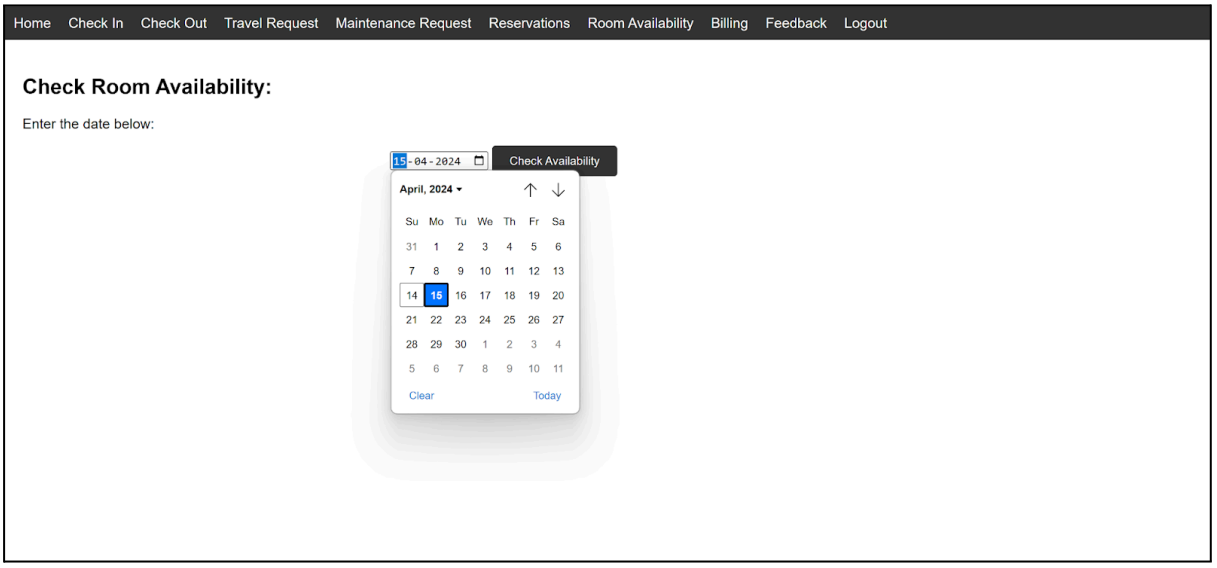


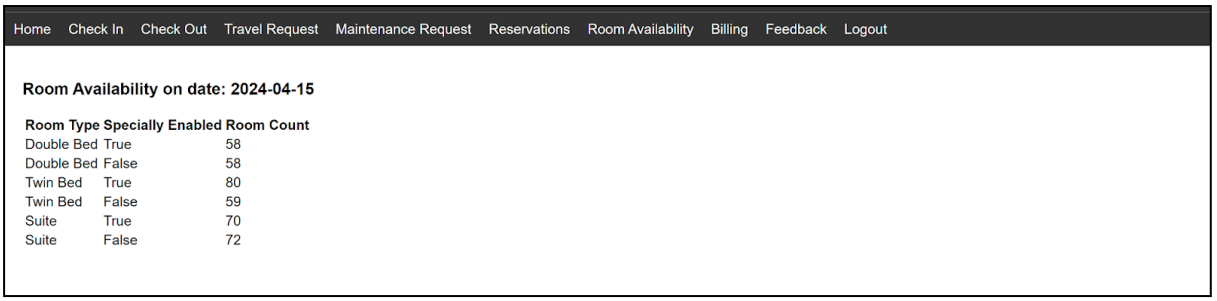
There is no page called "Room Availability" in the initial version.

After:


Home Check In Check Out Travel Request Maintenance Request Reservations Room Availability Billing Feedback Logout
Welcome, Suzanne!
Occupied Rooms
Room Number Guest ID Guest's First Name Guest's Last Name Phone no Email
184 28 Rutherford Tomkies 559-408-1299 None
189 26 Gwenny Peidro 619-973-9025 None
194 5 Gaby Domnin 240-580-9248 guest5@gmail.com
194 43 Nadia Aishford 212-795-4847 None
204 6 Tally Charopen 312-296-2463 guest6@gmail.com
204 59 Meyer Pendered 419-790-6207 None
204 66 Algernon Druery 502-718-9534 None
208 65 Marrissa Radnedge 814-664-9618 None
208 69 Lucky Millott 915-182-4875 None
211 88 Randene Robbings 516-443-8178 None
233 25 Georgette Coffey 423-542-1376 None
235 95 Blondy Reuther 434-556-2012 None
236 92 Calla Haszard 770-603-2400 None
237 51 Elianore Gayter 540-755-1437 None
242 29 Elianora Macieja 713-931-9692 None
244 87 Gregoor McDowall 212-199-8244 None
246 48 Marline Philson 954-826-5033 None
Assigned Open Maintenance Requests
Check-Out Room Cleaning


Home Check In Check Out Travel Request Maintenance Request Reservations Room Availability Billing Feedback Logout
Check Room Availability:
Enter the date below:
dd-mm-yyyy
Check Availability


Home Check In Check Out Travel Request Maintenance Request Reservations Room Availability Billing Feedback Logout
Check Room Availability:
Enter the date below:
15-04-2024
Check Availability
April, 2024
Su Mo Tu We Th Fr Sa
31 1 2 3 4 5 6
7 8 9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 1 2 3 4
5 6 7 8 9 10 11
Clear
Today

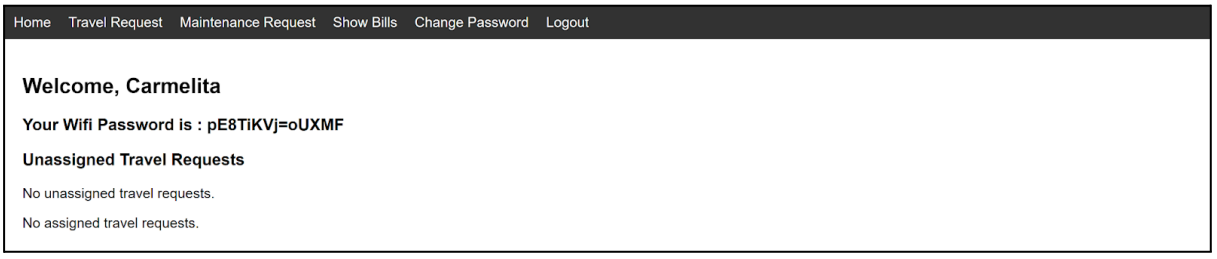


Room availability by date can now seen as above. Therefore, the staff can check whether the requested rooms are available when making a new reservation.

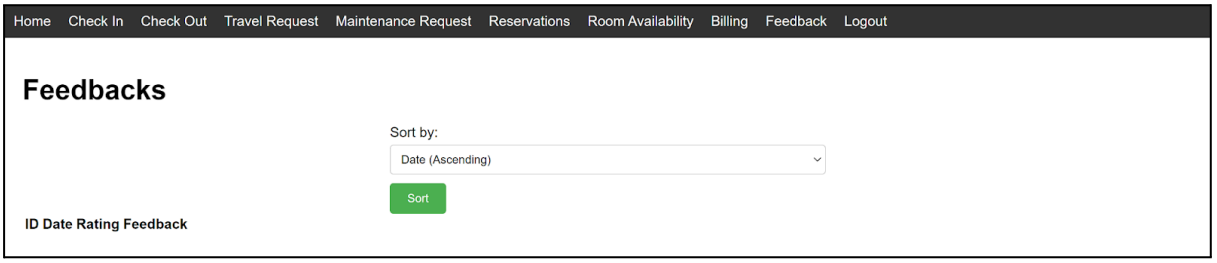
III. There wasn't a feedback form page in the current_guest dashboard for adding feedback. This has been corrected to allow current guests to write about their stay and the submitted feedback can be seen in the hospitality_staff dashboard.

<u>Before:</u>

Current Guest Dashboard



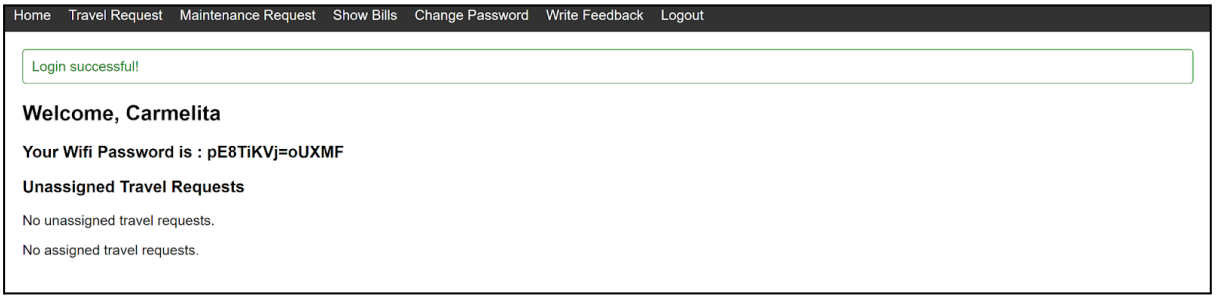
Hospitality Staff Dashboard

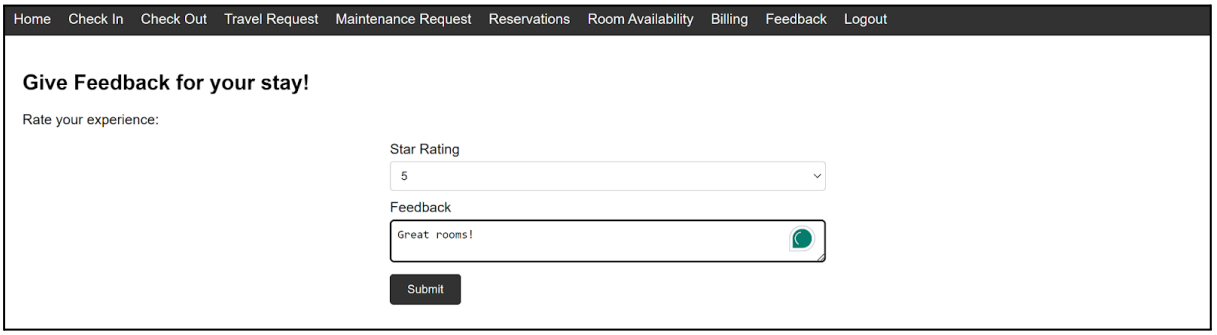


The current guests can't give their feedback, so the entries in the "Feedback" page in the hospitality staff's dashboard are NULL.

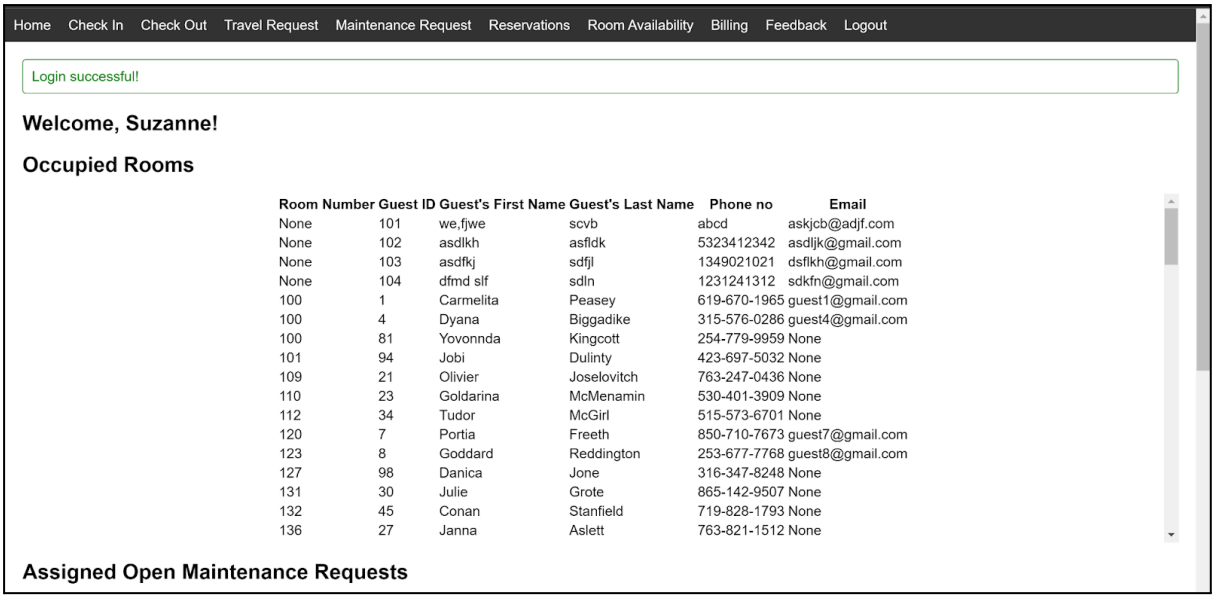
After:

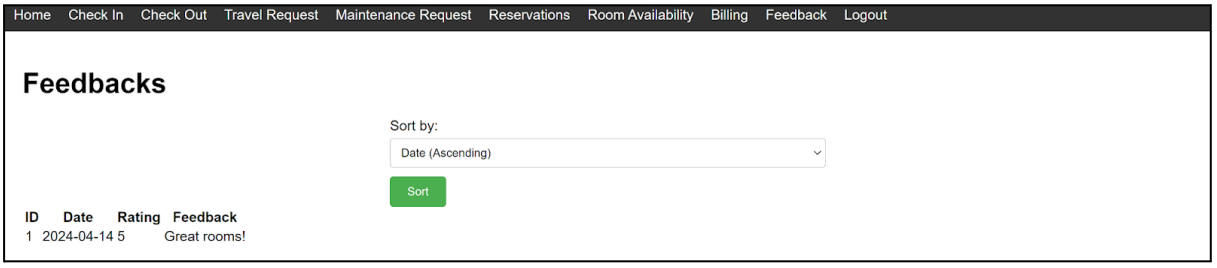
Current Guest Dashboard





Hospitality Staff Dashboard





Guests can give feedback from their dashboards, and the hospitality staff can see them from their dashboards.

1.2 Final Feedback

As per the final feedback by Mr. Yashwant Chouhan, we have added the following new features to our database:

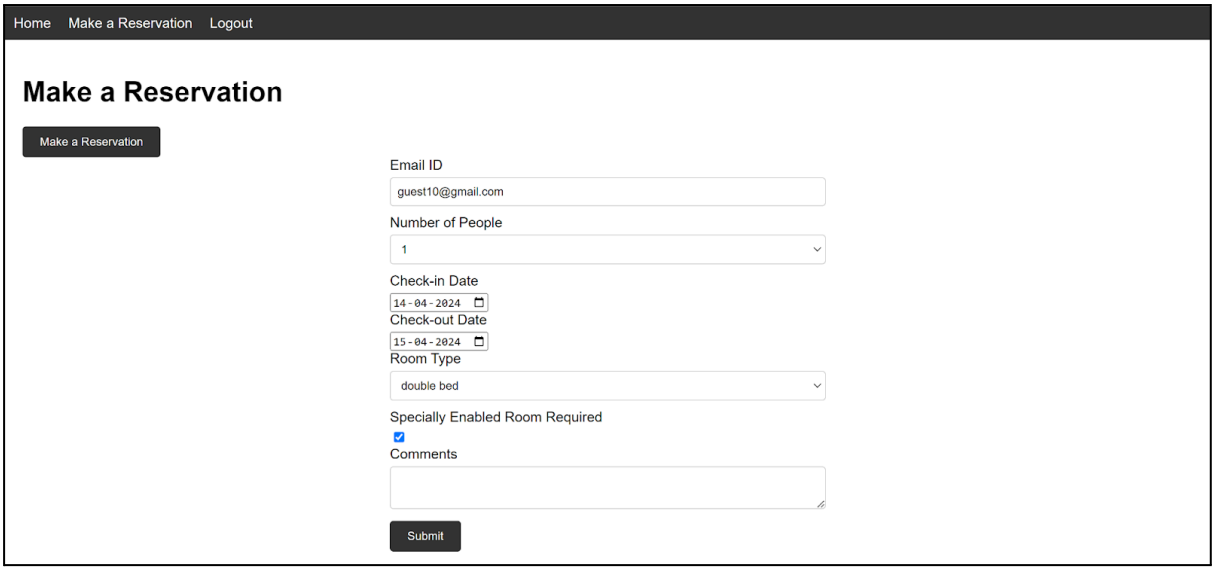
I. *Reservation Confirmation for requests made by IITGN members*
Initially, when the IITGN Members filled out the reservation form from their dashboards, a new reservation was directly created without the approval from the hospitality staff team. To tackle this issue, we have created a new column in the reservation table in the database called 'confirmed' using the query:

```
alter table reservation add column confirmed bool;
```

which will help us track the confirmation status of the reservations.
Now, when an IITGN Member creates a reservation, a new reservation does get added to the reservation table, but confirmed = FALSE until the hospitality staff approves the reservation. Also, if a reservation is not confirmed and the check-in date exceeds today, it automatically gets deleted from the reservation table.
The reservation made by hospitality staff automatically has confirmed = TRUE

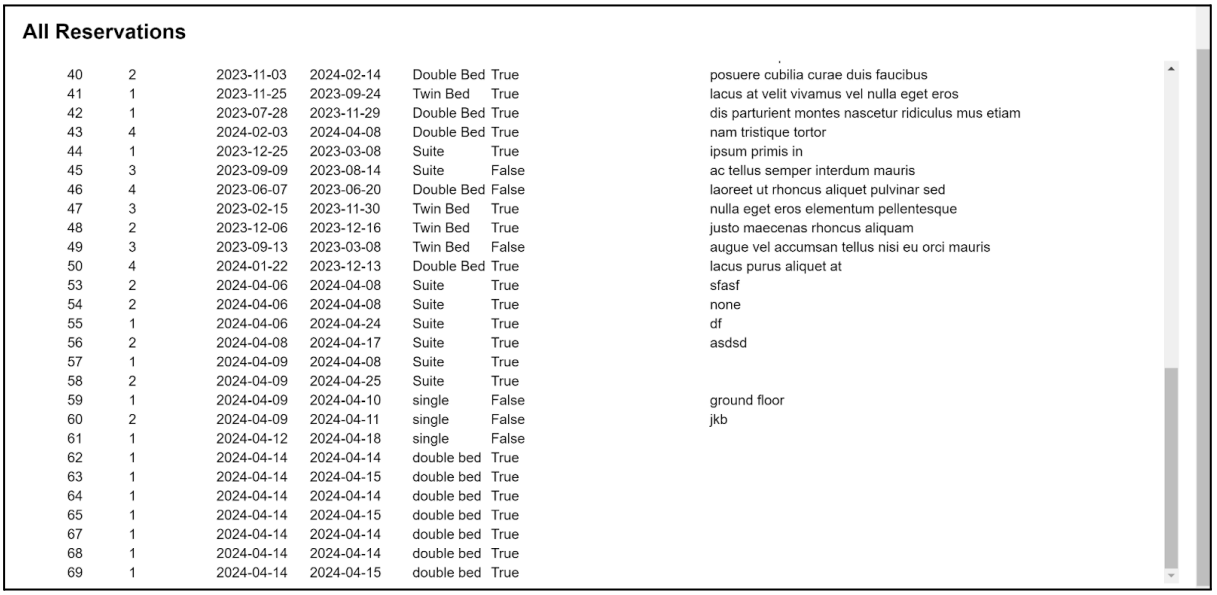
<u>Before:</u>

Reservation form in IITGN Member Dashboard



Reservation directly gets made without confirmation

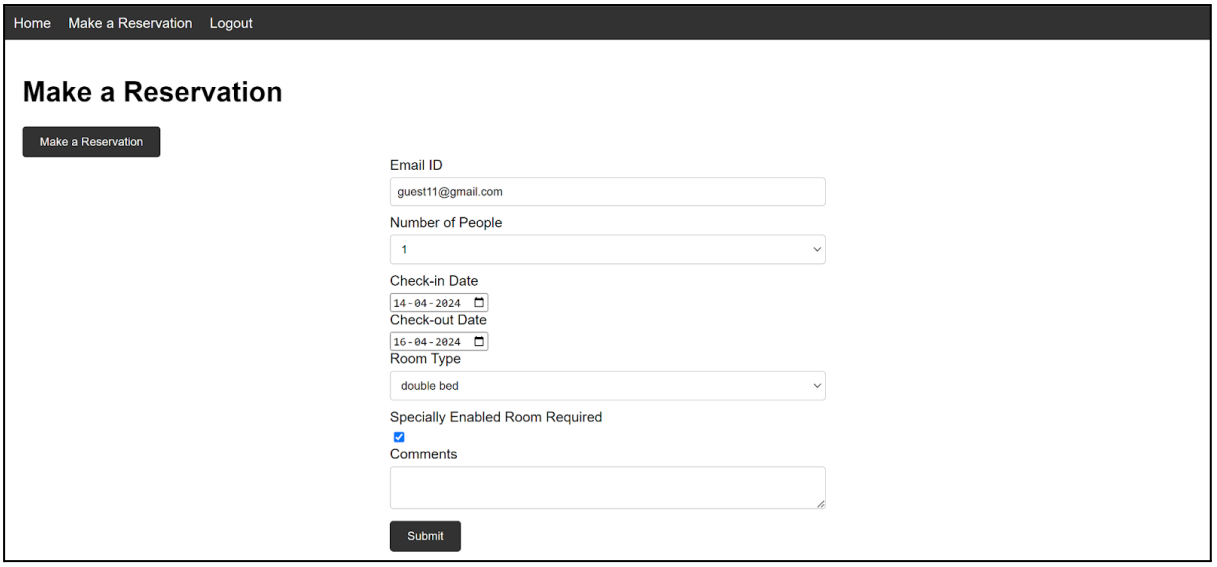
Reservation page in Hospitality Staff's dashboard



40	2	2023-11-03	2024-02-14	Double Bed	True	posuere cubilia curae duis faucibus
41	1	2023-11-25	2023-09-24	Twin Bed	True	lacus at velit vivamus vel nulla eget eros
42	1	2023-07-28	2023-11-29	Double Bed	True	dis parturient montes nascetur ridiculus mus etiam
43	4	2024-02-03	2024-04-08	Double Bed	True	nam tristique tortor
44	1	2023-12-25	2023-03-08	Suite	True	ipsum primis in
45	3	2023-09-09	2023-08-14	Suite	False	ac tellus semper interdum mauris
46	4	2023-06-07	2023-06-20	Double Bed	False	laoreet ut rhoncus aliquet pulvinar sed
47	3	2023-02-15	2023-11-30	Twin Bed	True	nulla eget eros elementum pellentesque
48	2	2023-12-06	2023-12-16	Twin Bed	True	justo maecenas rhoncus aliquam
49	3	2023-09-13	2023-03-08	Twin Bed	False	augue vel accumsan tellus nisi eu orci mauris
50	4	2024-01-22	2023-12-13	Double Bed	True	lacus purus aliquet at
53	2	2024-04-06	2024-04-08	Suite	True	sfasf
54	2	2024-04-06	2024-04-08	Suite	True	none
55	1	2024-04-06	2024-04-24	Suite	True	df
56	2	2024-04-08	2024-04-17	Suite	True	asdsd
57	1	2024-04-09	2024-04-08	Suite	True	
58	2	2024-04-09	2024-04-25	Suite	True	
59	1	2024-04-09	2024-04-10	single	False	ground floor
60	2	2024-04-09	2024-04-11	single	False	jkb
61	1	2024-04-12	2024-04-18	single	False	
62	1	2024-04-14	2024-04-14	double bed	True	
63	1	2024-04-14	2024-04-15	double bed	True	
64	1	2024-04-14	2024-04-14	double bed	True	
65	1	2024-04-14	2024-04-15	double bed	True	
67	1	2024-04-14	2024-04-14	double bed	True	
68	1	2024-04-14	2024-04-14	double bed	True	
69	1	2024-04-14	2024-04-15	double bed	True	

<u>After:</u>

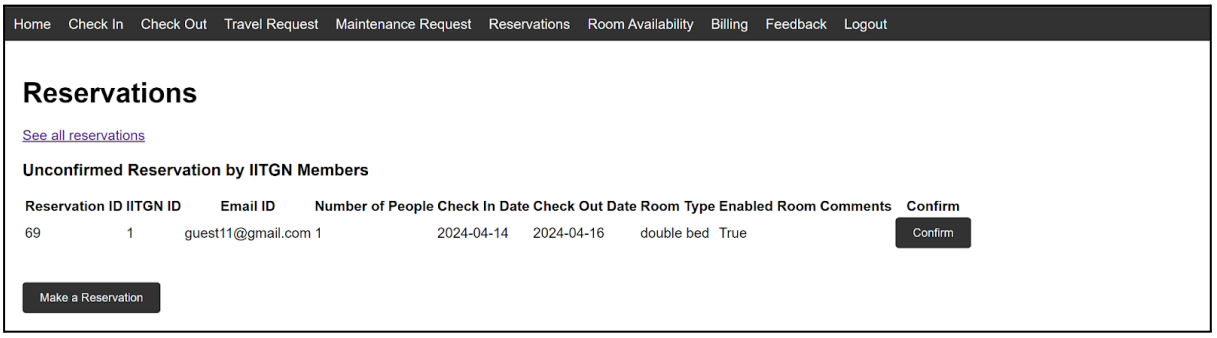
Reservation form in IITGN Member Dashboard



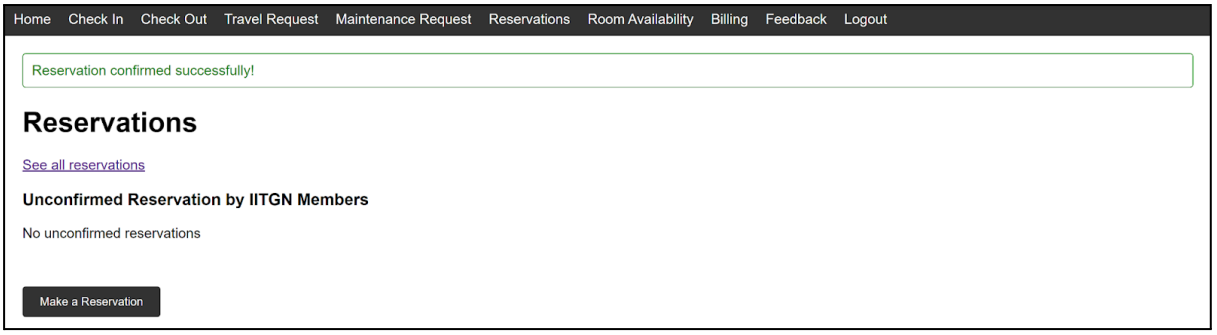
On submission, this creates a reservation request

Home page in IITGN Member Dashboard

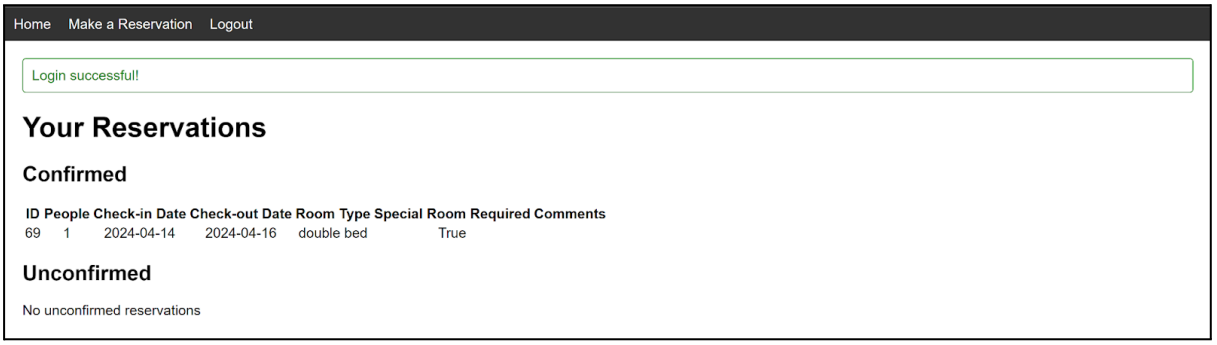
Reservation page in Hospitality Staff Dashboard



After clicking on confirm button



Home page in IITGN Member Dashboard after reservation confirmation

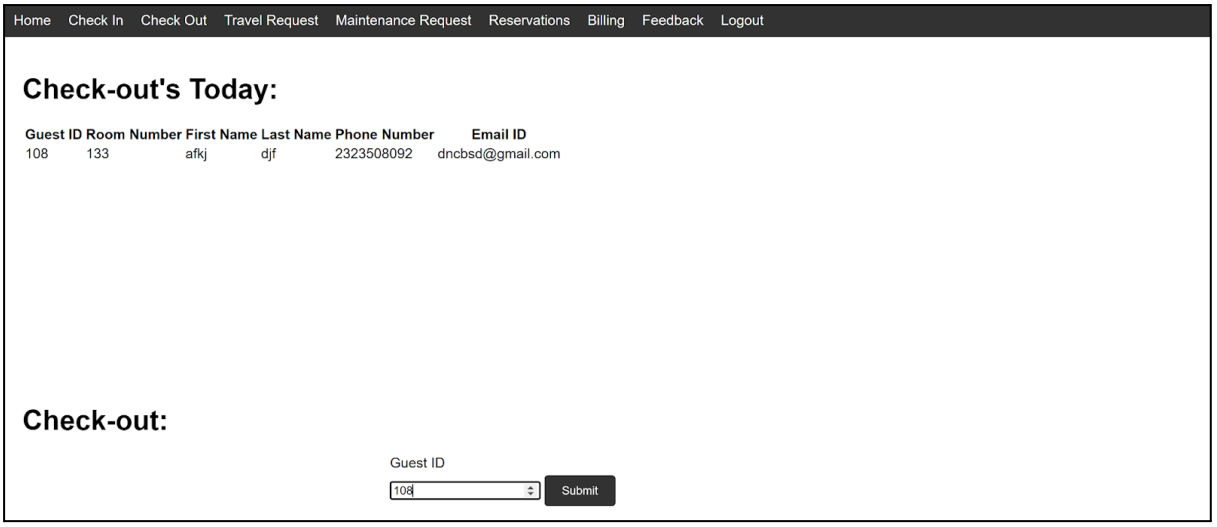


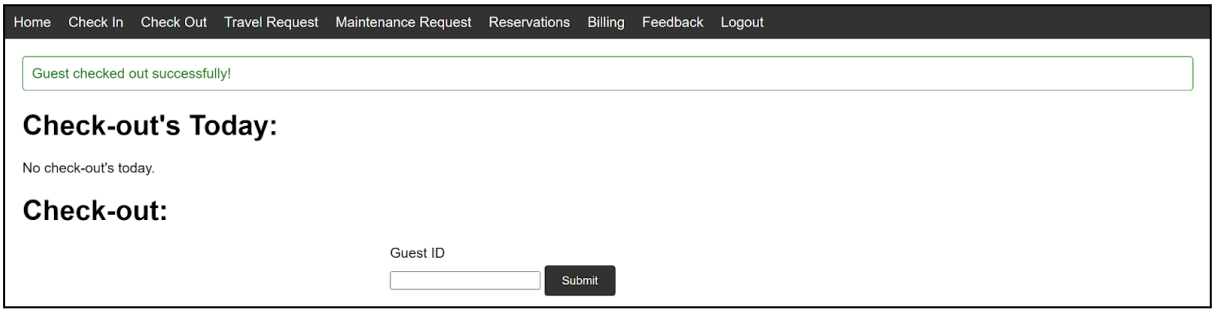
II. *Room cleaning maintenance request during check-out*

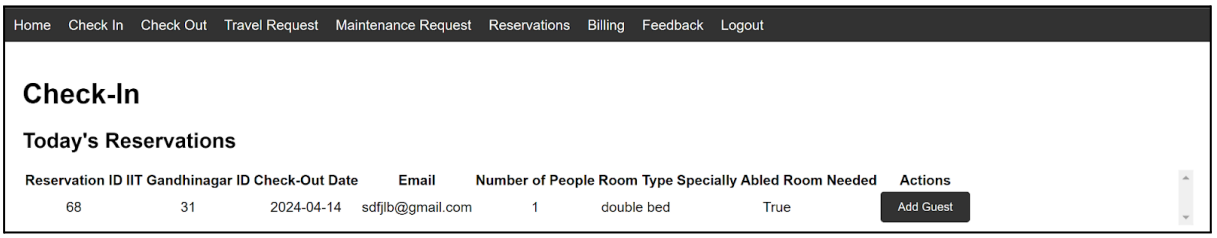
When a guest checks out, his/her room must be cleaned before it can be assigned to another guest during check-in. Therefore, we have implemented this as follows:

1. When a current guest checks out, a new maintenance request is created: "CHECK-OUT: Cleaning in room {room_no}."
2. A new entry is added to an existing table, 'requires_maintenance,' with request_id and room_no.
3. This maintenance request is assigned to a housekeeping staff following the same procedure as before, but now we have added a new page to the housekeeping staff dashboard called "Room Cleaning." This will help to keep track of check-out cleaning and other maintenance requests easily.
4. Until the room cleaning request is closed by the housekeeping staff to whom it was assigned, the room is not shown in the list of available rooms during the check-in.

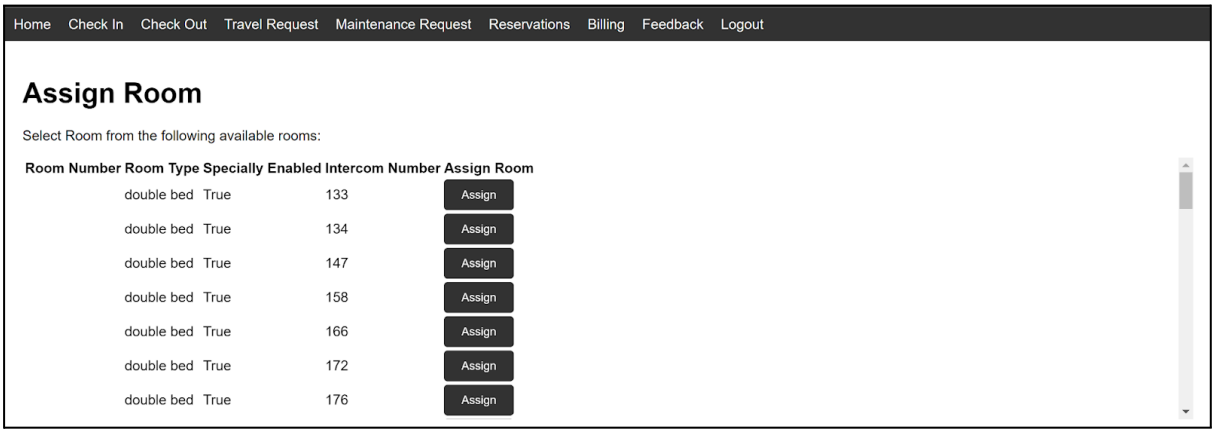
Before:

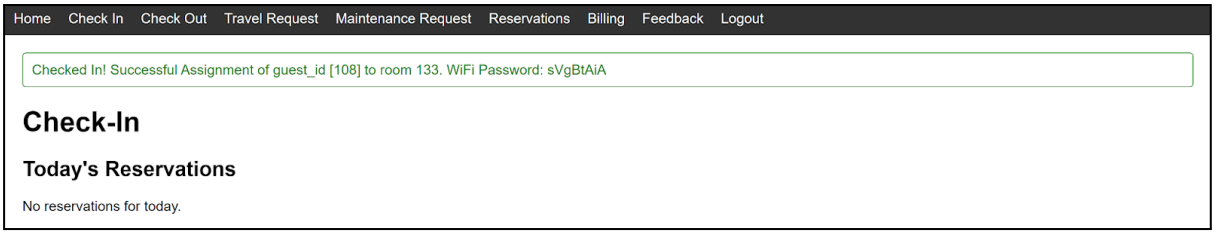




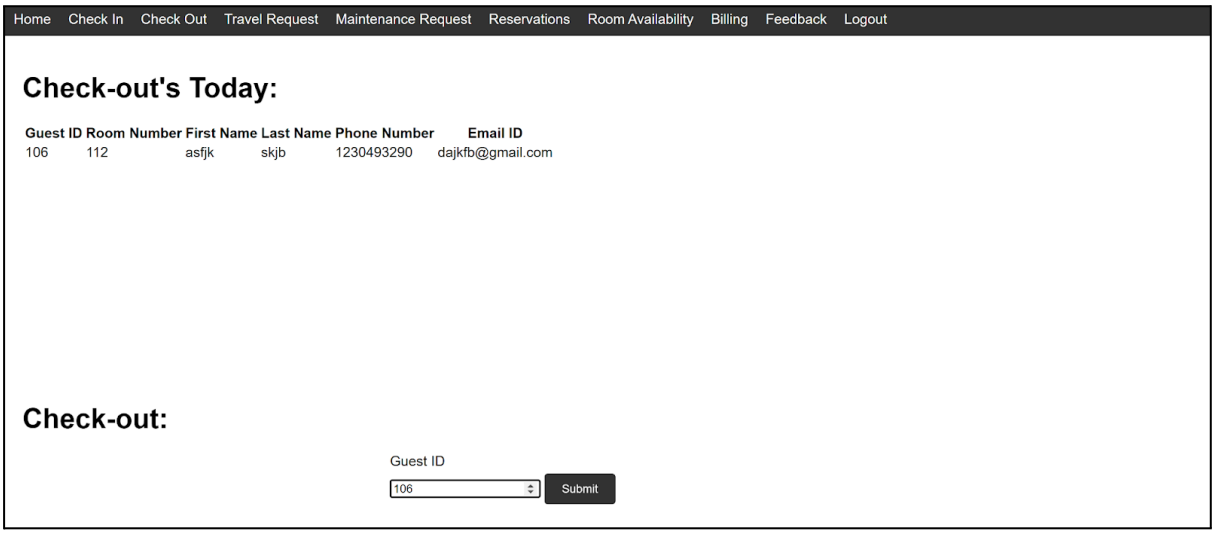


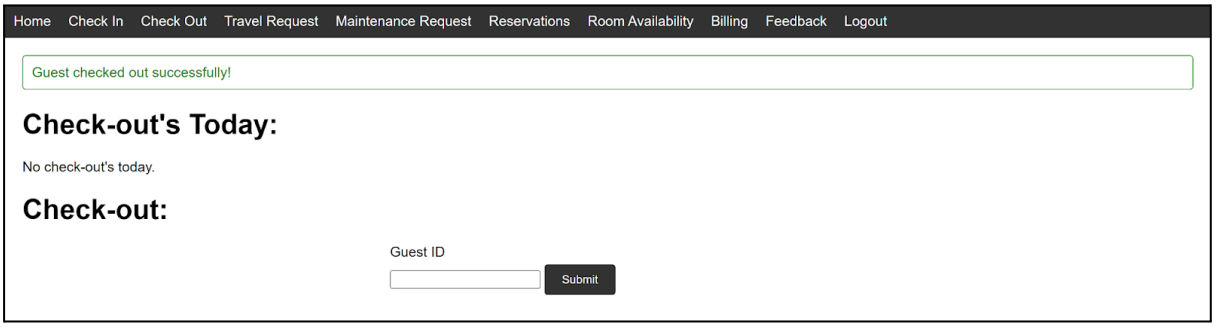
The room checked-out on the same day visible as available room for check-in



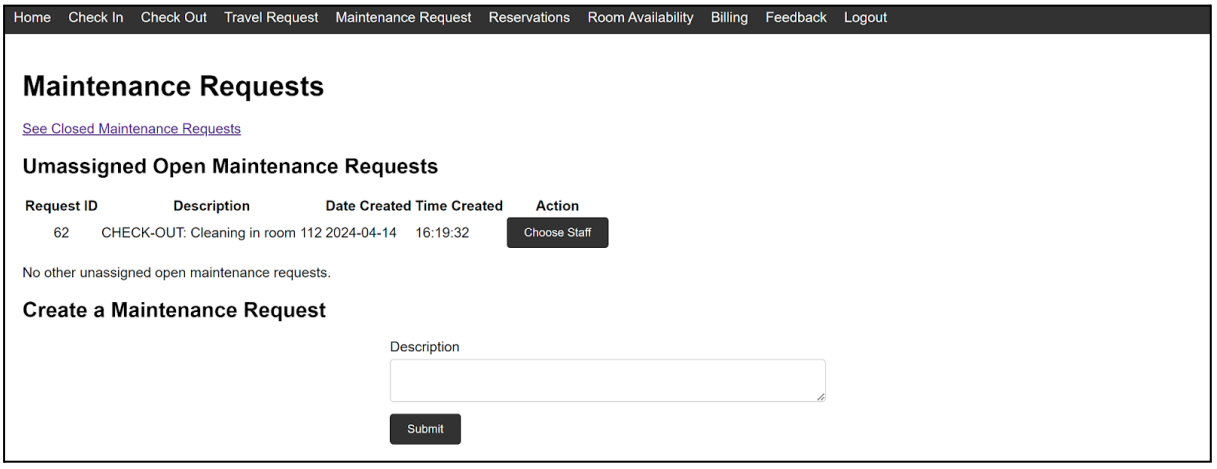


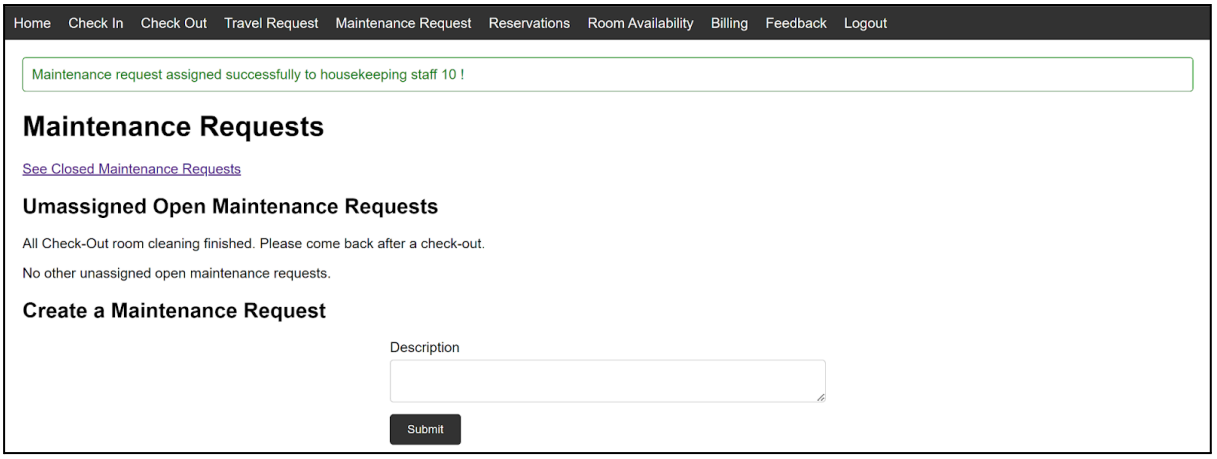
After:

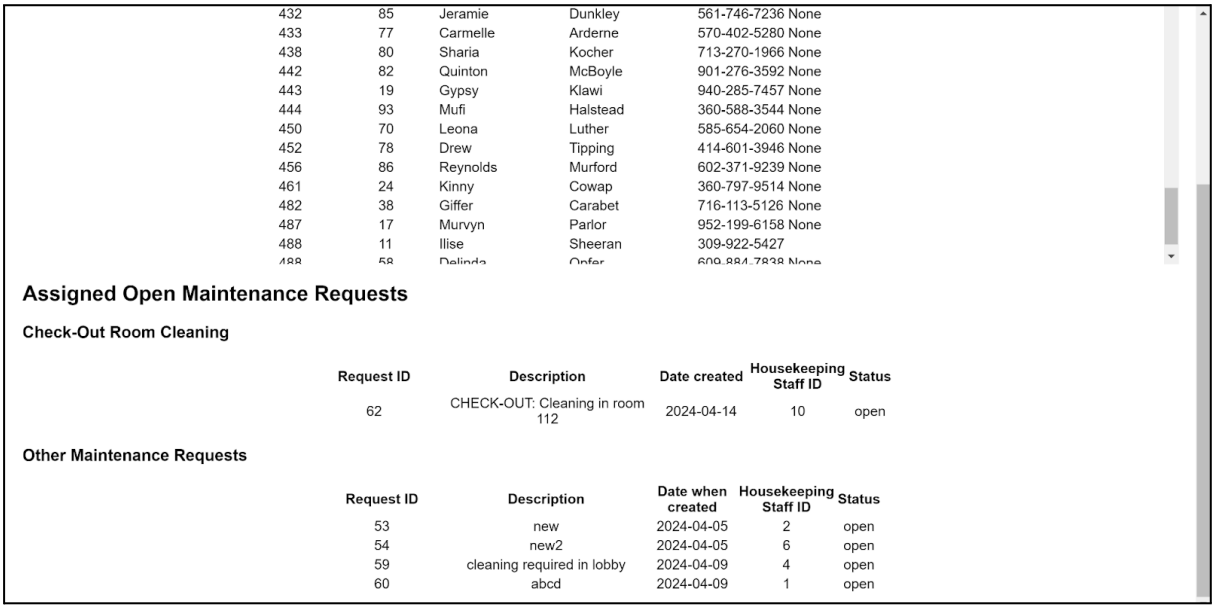




Room cleaning maintenance request created automatically







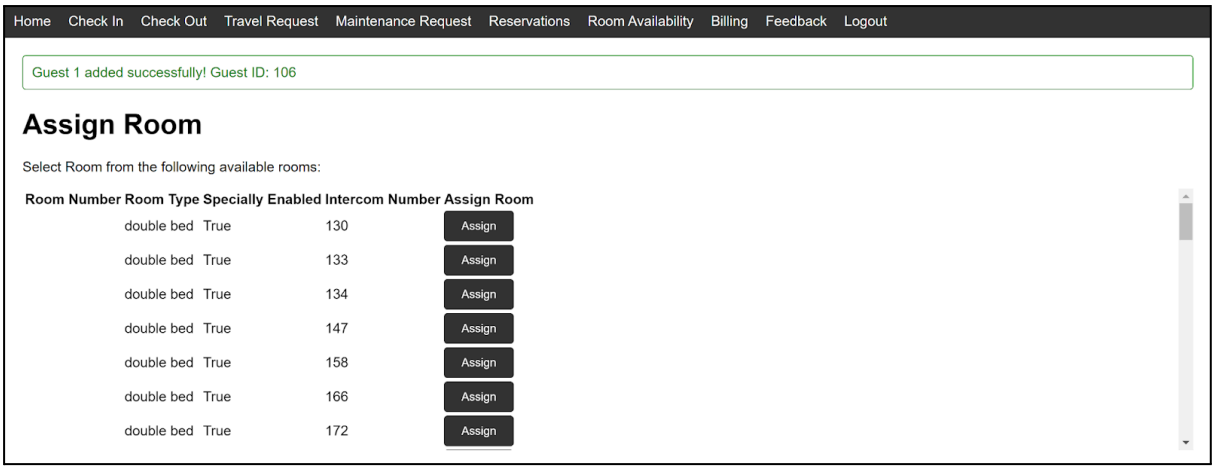
432	85	Jeramie	Dunkley	561-746-7236	None
433	77	Carmelle	Arderne	570-402-5280	None
438	80	Sharia	Kocher	713-270-1966	None
442	82	Quinton	McBoyle	901-276-3592	None
443	19	Gypsy	Klawi	940-285-7457	None
444	93	Mufi	Halstead	360-588-3544	None
450	70	Leona	Luther	585-654-2060	None
452	78	Drew	Tipping	414-601-3946	None
456	86	Reynolds	Murford	602-371-9239	None
461	24	Kinny	Cowap	360-797-9514	None
482	38	Giffer	Carabet	716-113-5126	None
487	17	Murvyn	Parlor	952-199-6158	None
488	11	Ilise	Sheeran	309-922-5427	

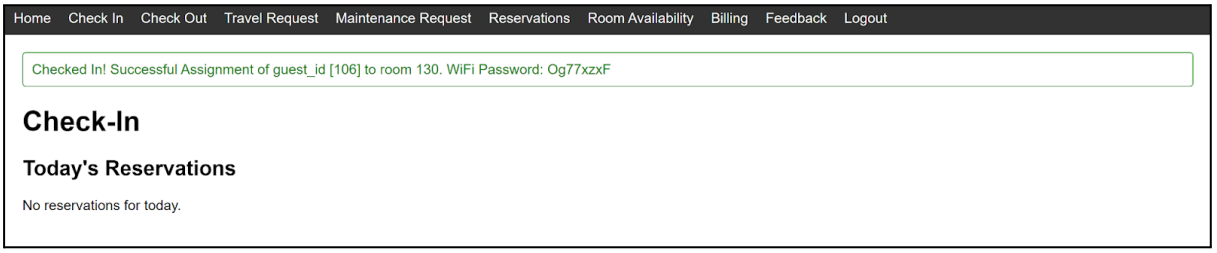
Request ID	Description	Date created	Housekeeping Staff ID	Status
62	CHECK-OUT: Cleaning in room 112	2024-04-14	10	open

Request ID	Description	Date when created	Housekeeping Staff ID	Status
53	new	2024-04-05	2	open
54	new2	2024-04-05	6	open
59	cleaning required in lobby	2024-04-09	4	open
60	abcd	2024-04-09	1	open

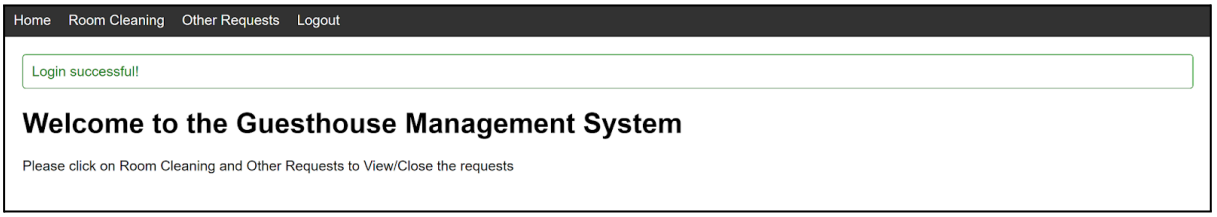
Reservation ID	IIT Gandhinagar ID	Check-Out Date	Email	Number of People	Room Type	Specially Abled Room Needed	Actions
65	1	2024-04-15	safdjn@gmail.com	1	double bed	True	Add Guest

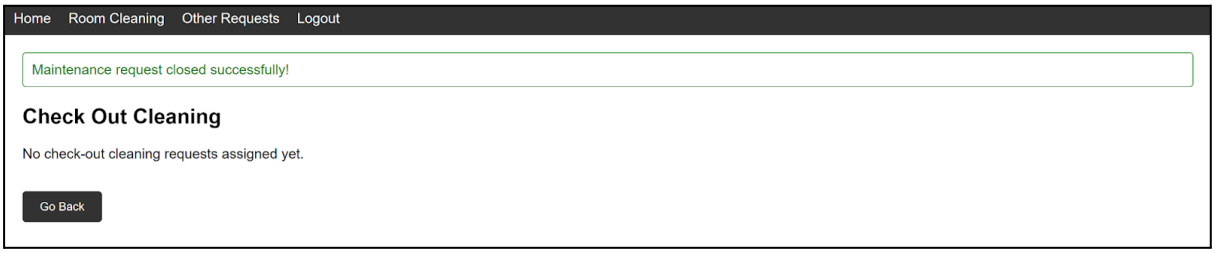
Room 112, which was checked out, is not visible in the check-in room assignment, as the maintenance request hasn't been closed .





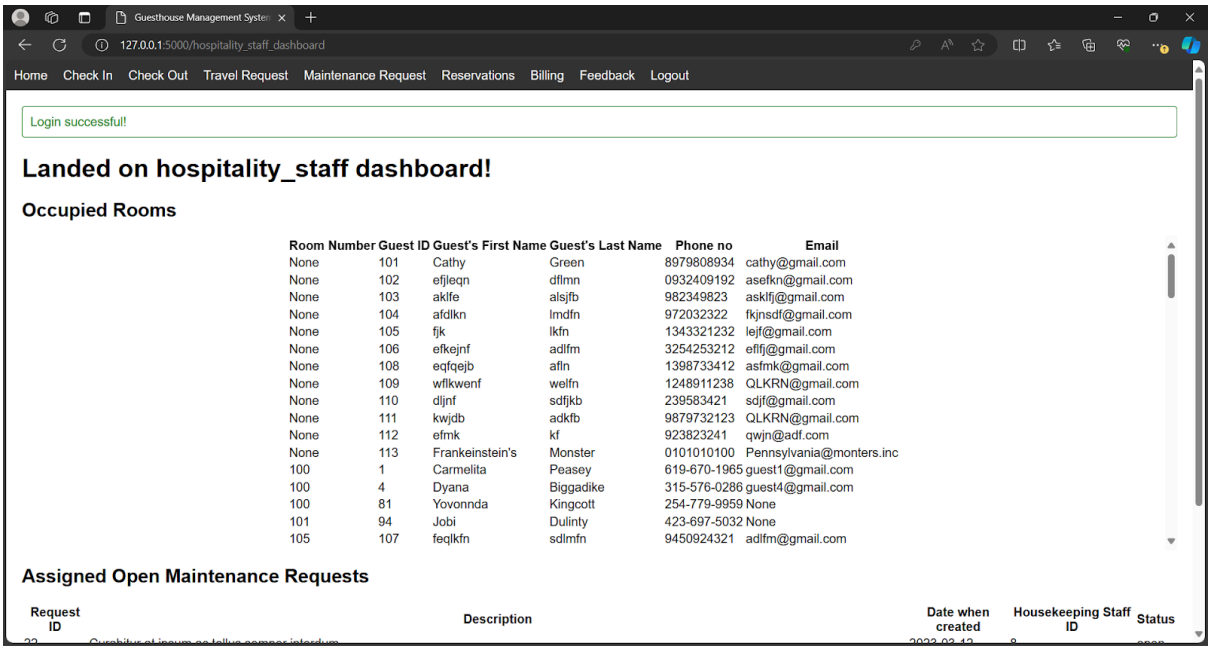
Housekeeping Staff Dashboard





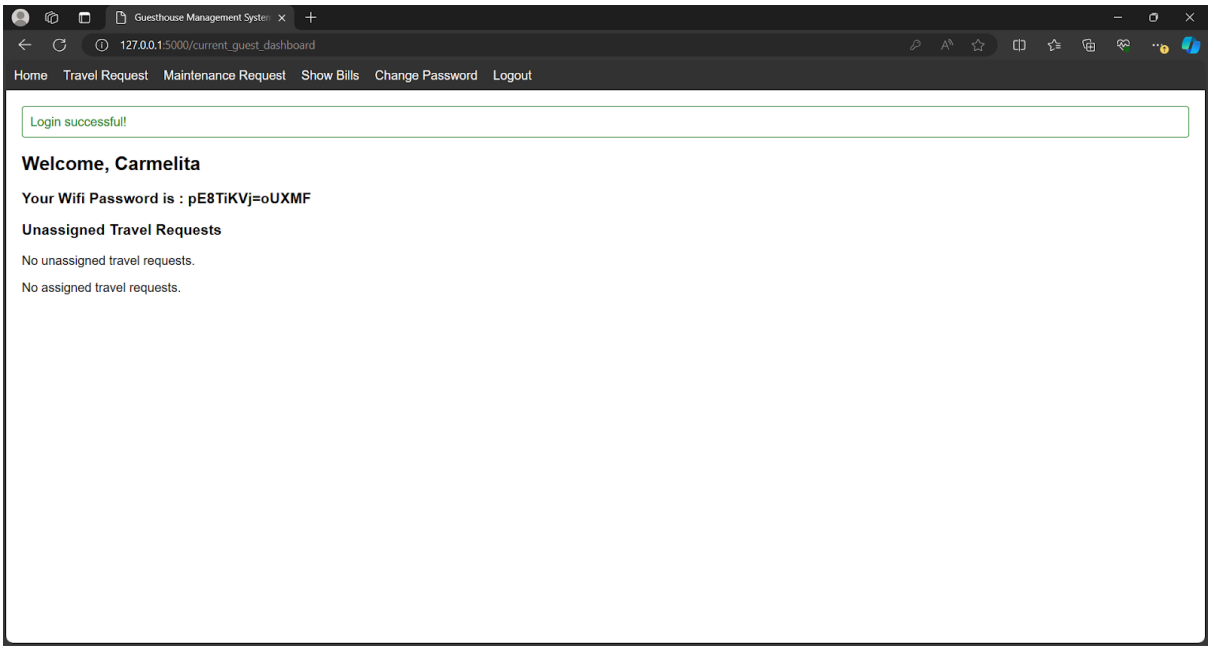
2. User Views and Privileges in the database

A. *Hospitality Staff Dashboard (Admin) :*



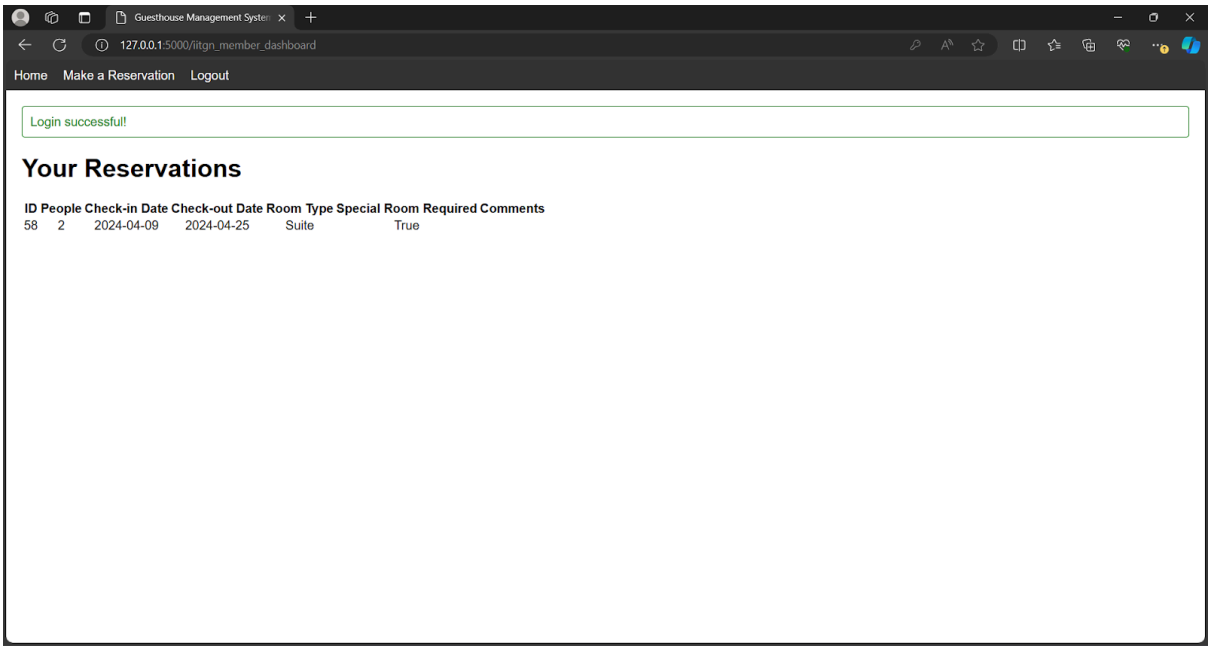
- This hospitality staff dashboard is the home page and we can see the options shown above are:
    - Check-in: This navigates to the page that shows that day's reservations.
    - Check-out: This navigates to the page that shows that day's check-outs and check-out billings.
    - Travel request: This navigates to the page that shows completed and unassigned travel requests.
    - Maintenance request: This navigates to the page that shows open and closed maintenance requests.
    - Reservations: This navigates to the page that shows all the reservations.
    - Billing: This navigates to the page where we can create or generate a Bill.
    - Feedback: This navigates to the page that shows all the feedbacks.
    - Logout: This can be used to logout from the current account.

B. *Current Guest Dashboard:*



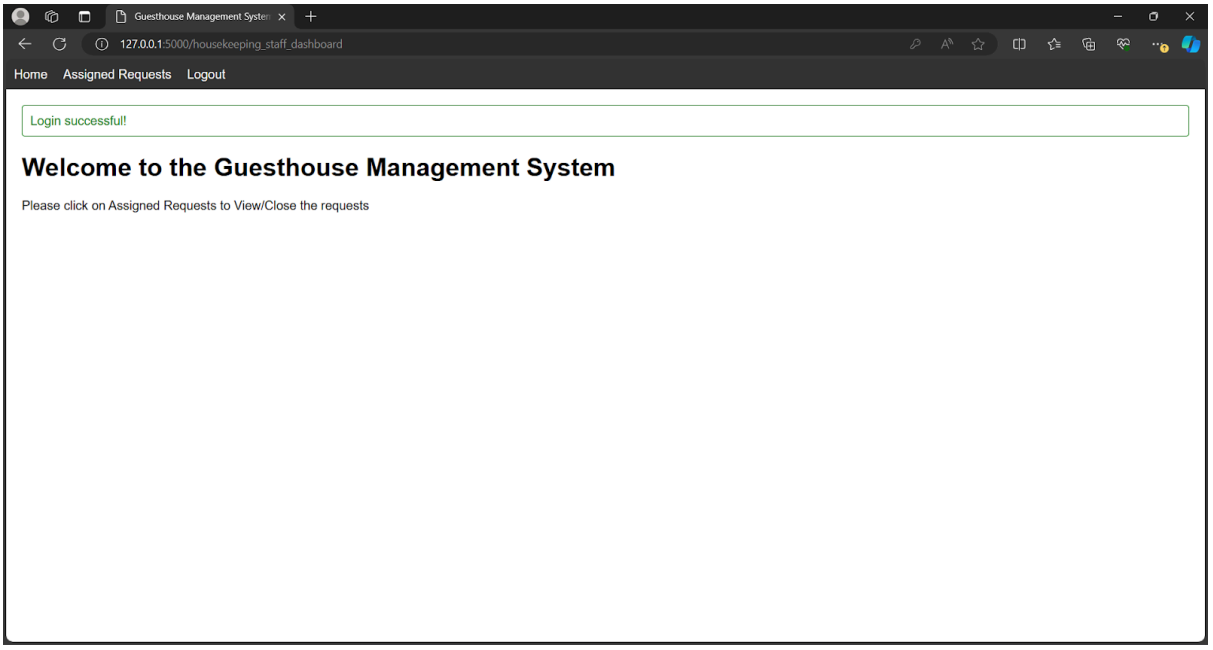
- This current guest dashboard is the home page of the current guest. We can see the options shown above are:
    - Travel request: This navigates to the page where the guest can make a travel request.
    - Maintenance request: This navigates to the page where the guest can make a maintenance request.
    - Show bills: This navigates to the page that shows all the bills
    - Change password: This navigates to the page where the user can change password.
    - Logout: This can be used to logout from the current account.

C. IITGN Member Dashboard:



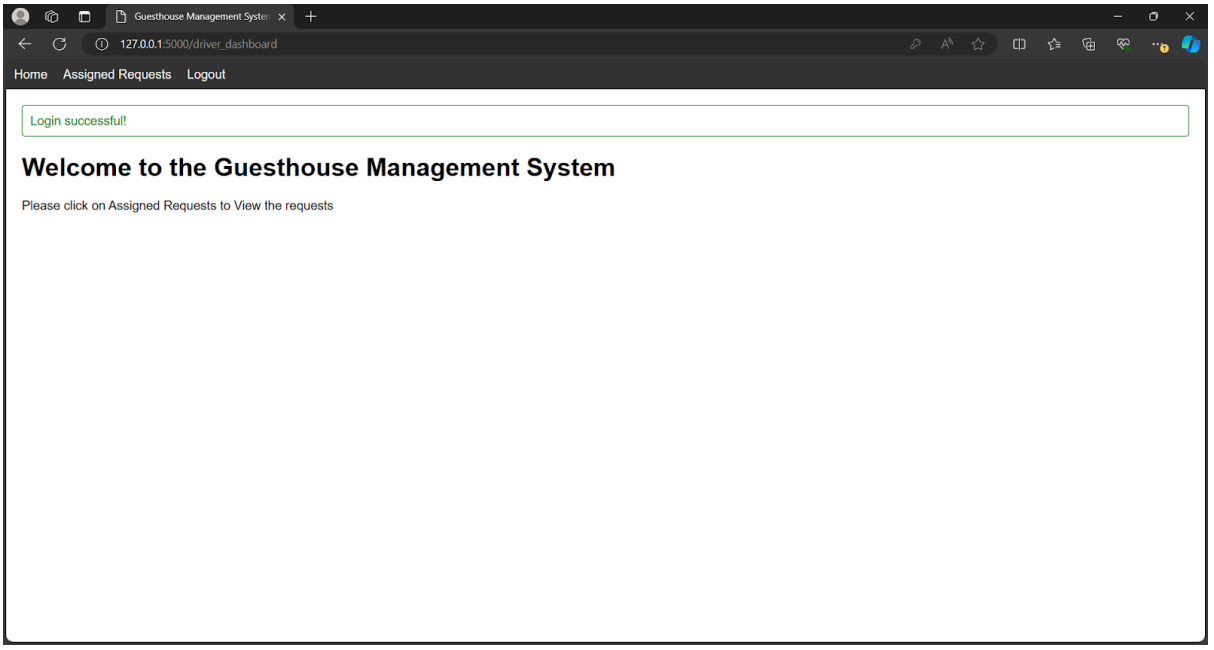
- This is the IITGN member dashboard and we can make a reservation through the above shown option.
- This current guest dashboard is the home page and the user can login with their credentials through the login button shown above.

D. Housekeeping Staff Dashboard:



- This is the housekeeping staff dashboard and we can see the assigned requests for housekeeping staff through the above shown option.
- And from the logout option we can logout from the dashboard.

E. Driver Dashboard:



- This is the driver dashboard and we can see the assigned requests for driver through the above shown option.
- And we can logout through the logout option beside the assigned requests option.

3.2 Responsibility of G2:

1. Concurrent multi-user access

As you may know, our web app has different logins, so different types of users can access our web app and can modify the database concurrently. For this, we applied locks for some of our database tables so that at a time, only a user can update the information, preventing concurrent transactions from interfering with each other.

For example, guest_travel_request form can be filled by multiple users/guests. So, we applied locks using SQLAlchemy's (python SQL toolkit, which is used to manage SQL databases using pythonic language) support.

```python
@guest.route('/current_guest_dashboard/guest_travel_request', methods=['GET', 'POST'])
@login_required
def guest_travel_request():
    form = TravelRequestForm()
    if form.validate_on_submit():
        # Handle travel request form submission
        # Acquire lock on the travel_request table
        locked_table = select(travel_request).with_for_update()
        db.session.execute(locked_table)

        with db.session.begin_nested():
            highest_id = db.session.query(db.func.max(travel_request.travel_request_id)).scalar()
            if highest_id is None:
                highest_id = 0
            travel_request_add = travel_request(
                travel_request_id=highest_id + 1,
                number_of_travellers=form.number_of_travellers.data,
                date_of_travel=form.date_of_travel.data,
                pick_up_time=form.pick_up_time.data,
                destination=form.destination.data,
                travel_purpose=form.travel_purpose.data
            )
            db.session.add(travel_request_add)
            db.session.flush()  # Flush to generate the primary key before adding the initiated_travel_request
            initiated_travel_request = InitiatedTravelRequest(
                travel_request_id=highest_id + 1,
                guest_id=current_user.get_id()
            )
            db.session.add(initiated_travel_request)
        db.session.commit()
        flash('Travel request submitted successfully!', 'success')
        return redirect(url_for('current_guest_dashboard.guest_travel_request'))
    return render_template('guest_travel_request.html', form=form)
```

Code for guest_travel_request page with locking mechanism

For acquiring locks :

```
locked_table = select(travel_request).with_for_update()
```

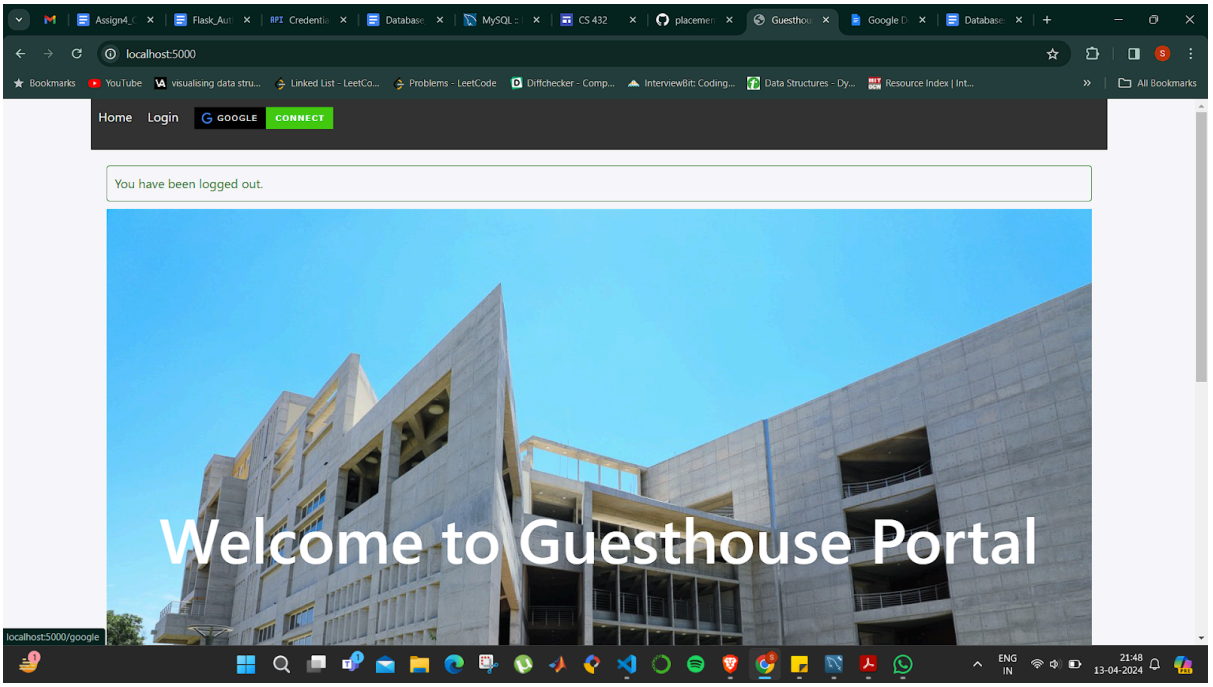
The above line creates an SQL statement “select _ _ _ for update”[6] that selects rows from the database table ‘travel_request’ and acquires a lock on them to prevent any other transaction from being modified until an existing transaction finishes.

`db.session.commit()`: This line commits the changes made within the transaction to the database and releases the lock acquired earlier.

To implement concurrent multi-user access, locks have been applied to the tables like travel_request, guest_travel_request, maintenance_request, guest_maintenance_request, reservation & bill.

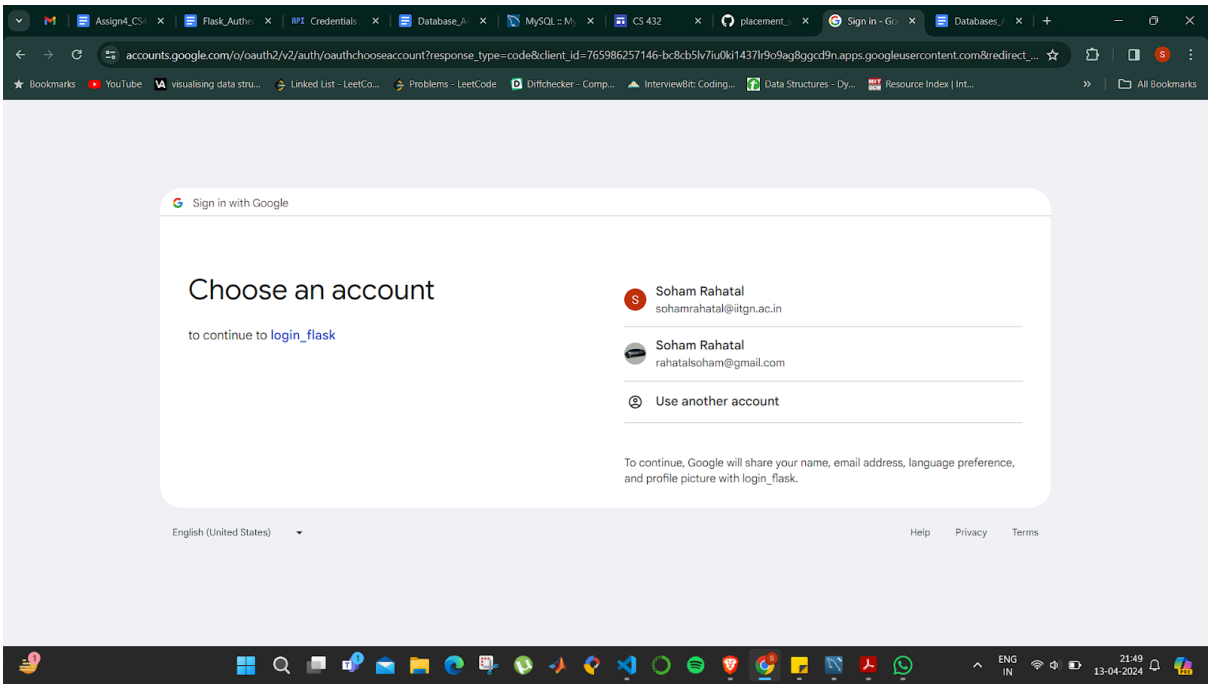
2. Google Authentication

Our web app has five different logins, and hospitality_staff (administrator) and iitgn_member are the only ones with an iitgn email ID. So, we added google authentication for hospitality_staff members and iitgn_member.

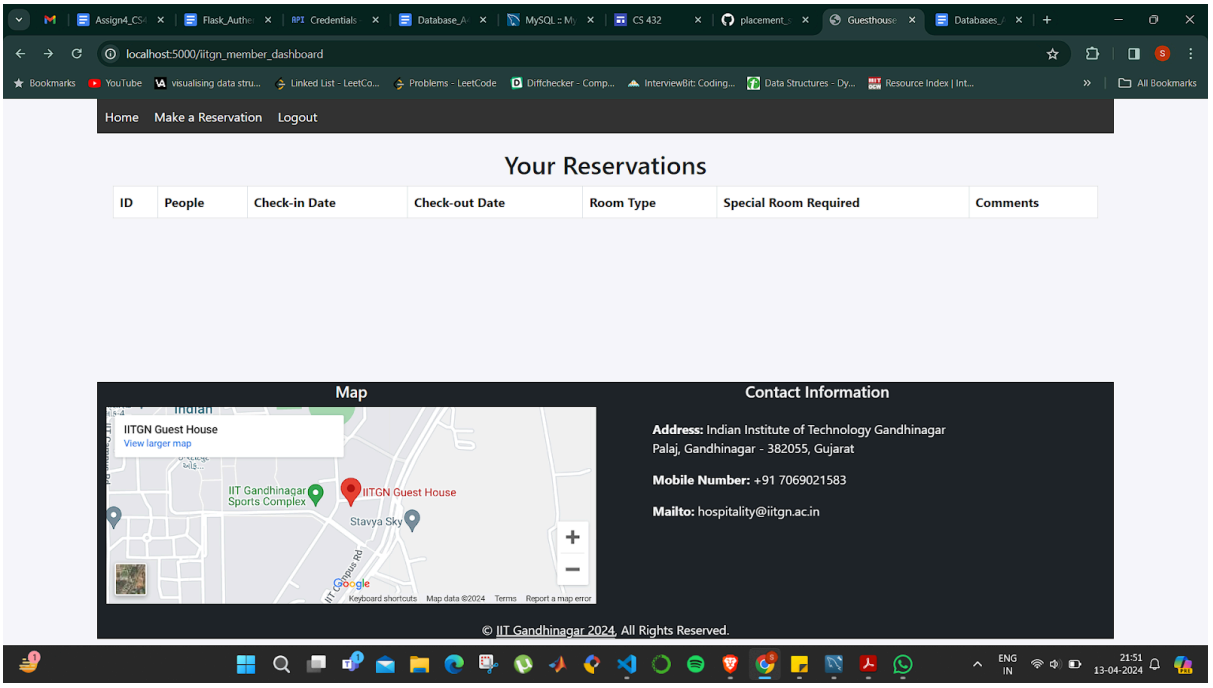


When you click on the [Google Connect] button to authenticate →

We are redirected to the page below to select the Google account.



After selecting the iitgn account i.e., sohamrahatal@iitgn.ac.in



We are redirected to the iitgn_member dashboard as my email_id/profile is present in the iitgn_member database table of our database 'guesthouse_db'.

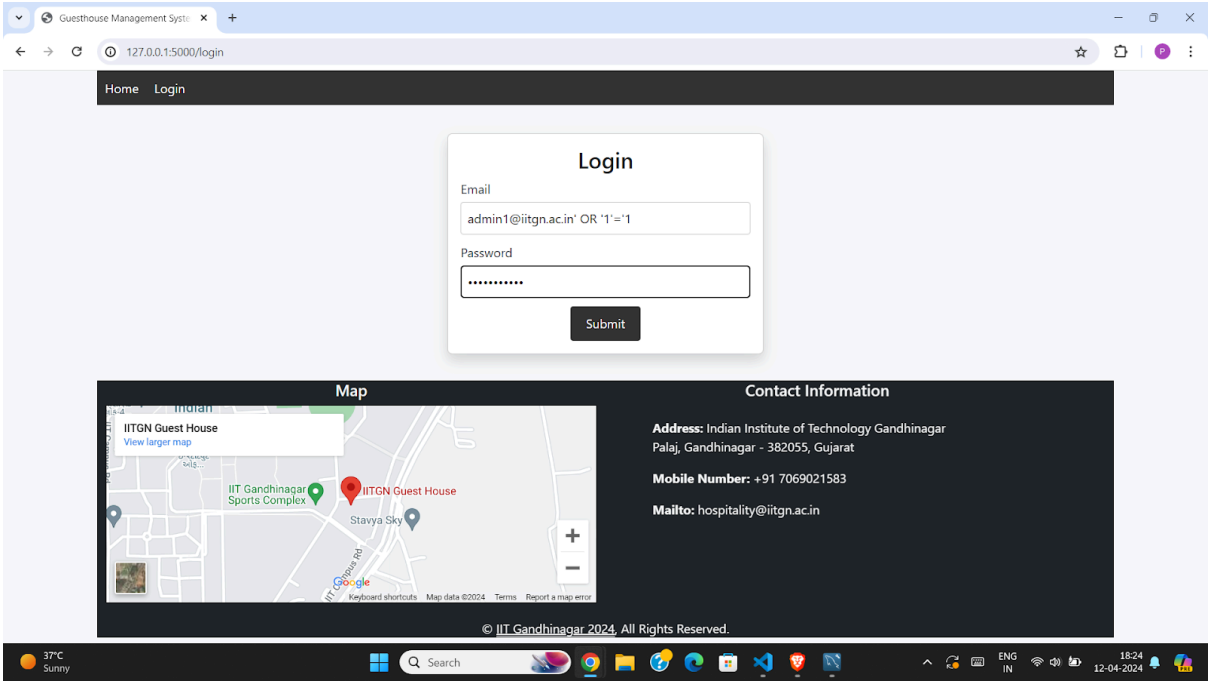
If an email_id is related to hospitality_staff, then that account will be redirected to the hospitality_staff dashboard of the web app.

For the rest logins, like driver, guest, and housekeeping_staff, they will have to login through the normal login system of the web app as they don’t have iitgn email_id.

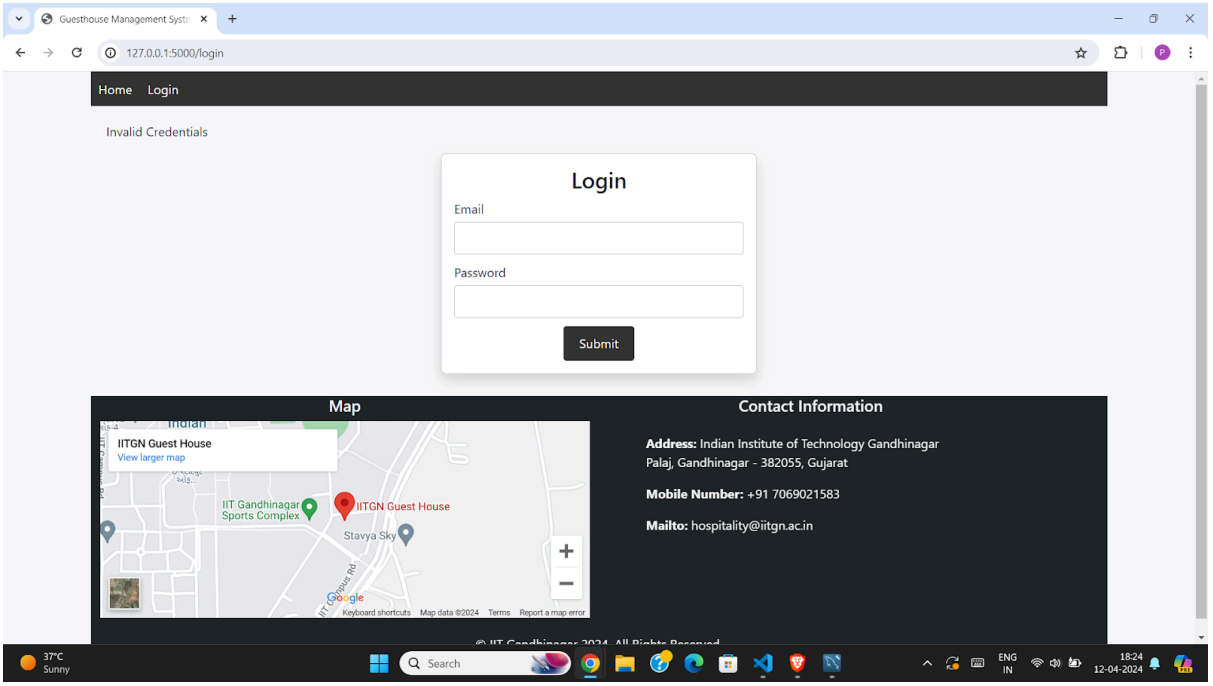
3.3 Responsibility of G1 & G2:

1. Attacks on our WebApp

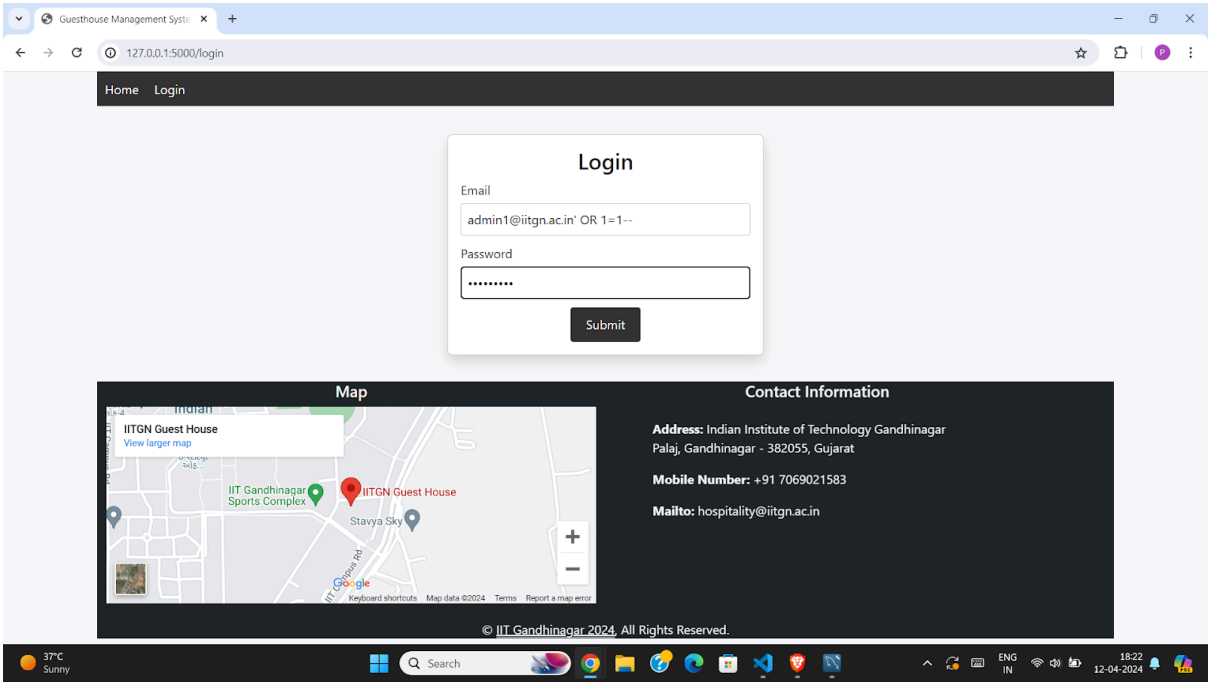
1. SQL Injection Error Attack:



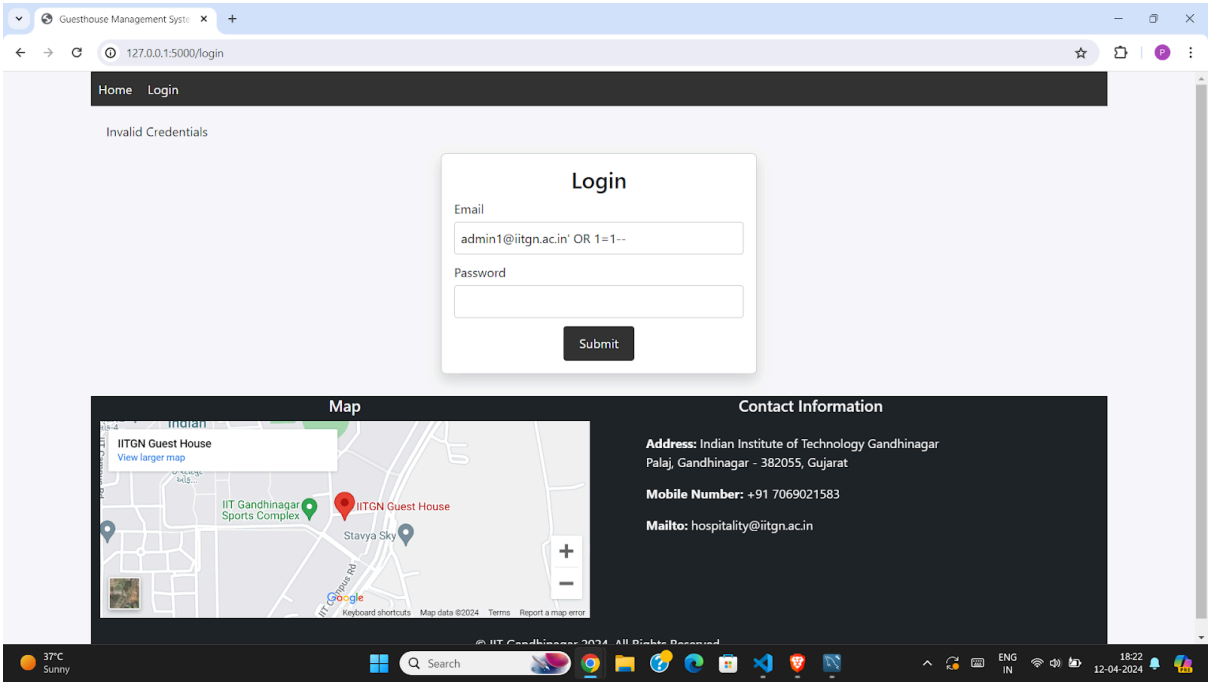
We try the first method, where we try to add an OR condition that is always true i.e. 1=1, to modify the SQL query generated through the form.



We received an ‘Invalid Credentials’ message, and we could not hack into the system.



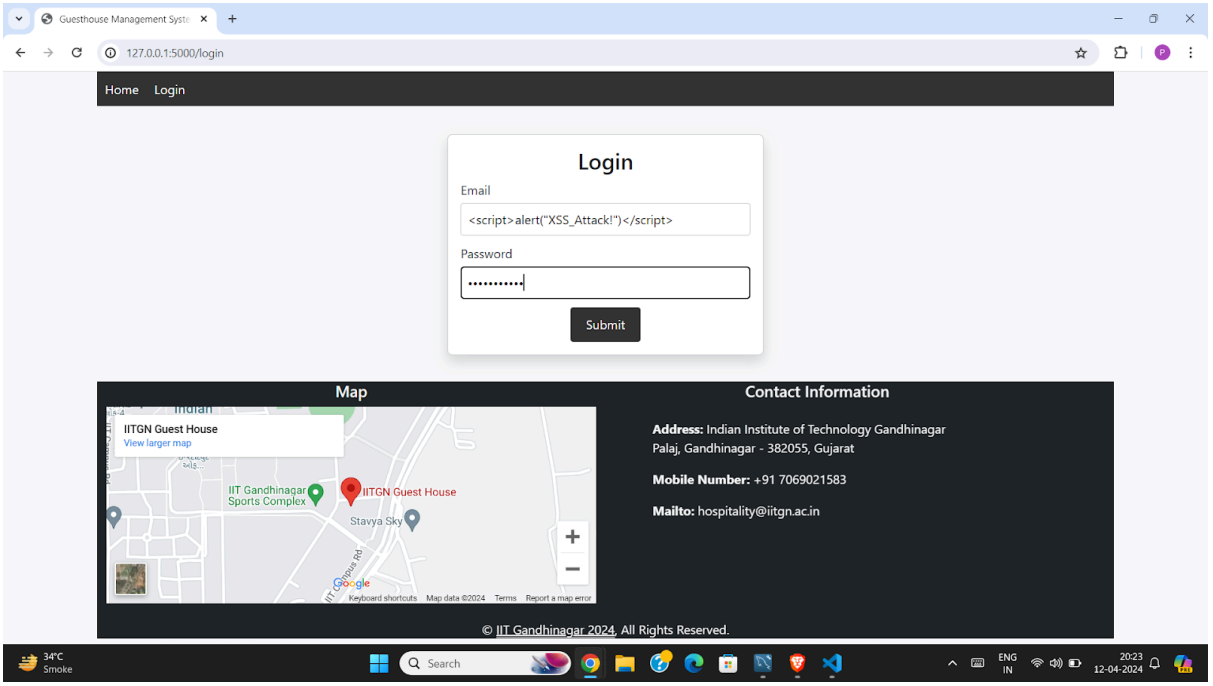
We tried the second method, where we used the symbol “--” which is used to comment in MySQL. It would comment the SQL code after it, such that we would be able to hack into the system.



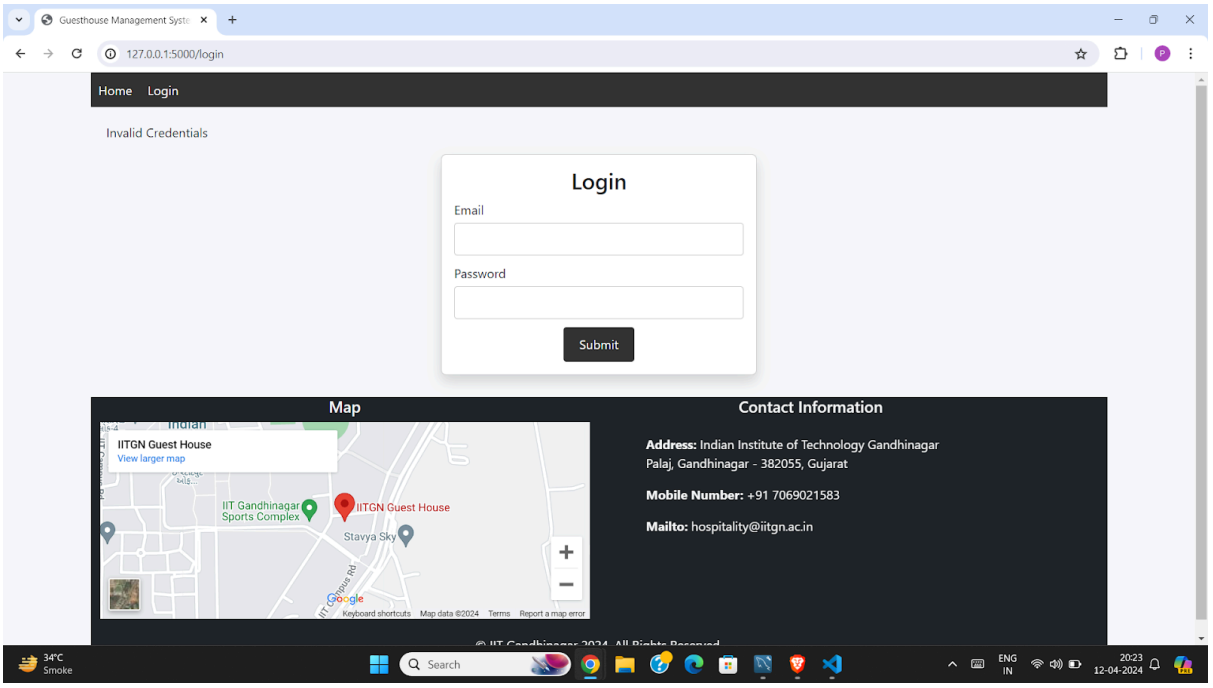
We received an ‘Invalid Credentials’ message, and we could not hack into the system.

Our System was able to withstand SQL Injection attacks because it uses parameterized queries, specifically through the ORM (Object-Relational Mapping) methods provided by the SQLAlchemy library to interact with the database.

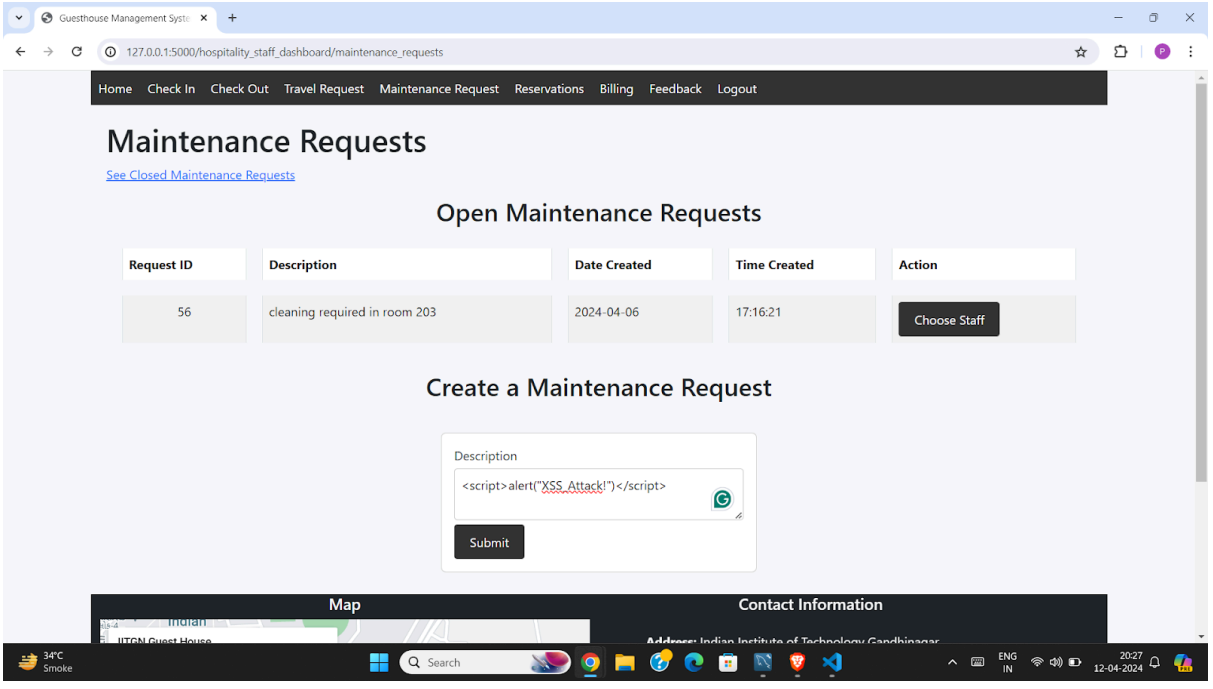
2. **XSS Attack:**



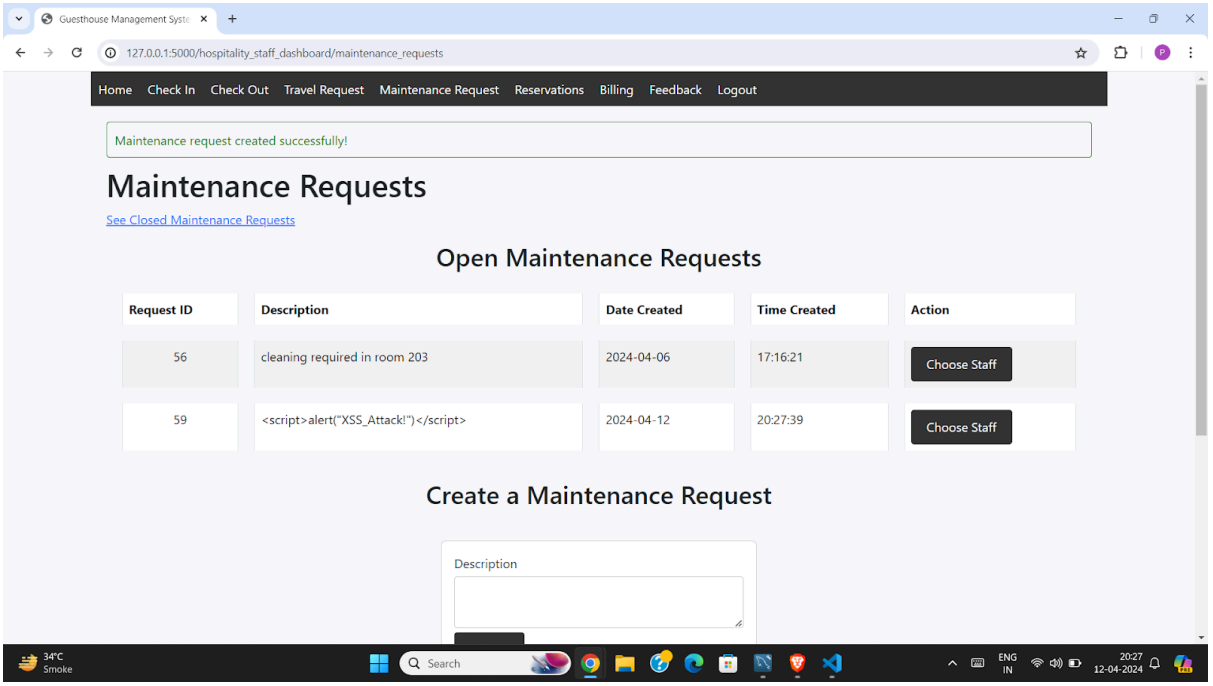
We tried to attempt an XSS attack at the login page in our website, if the attack is successful we would recieve an alert pop-up of “XSS_Attack!” on web-page.



We received an ‘Invalid Credentials’ message, and we could not conduct a XSS attack on the login page.



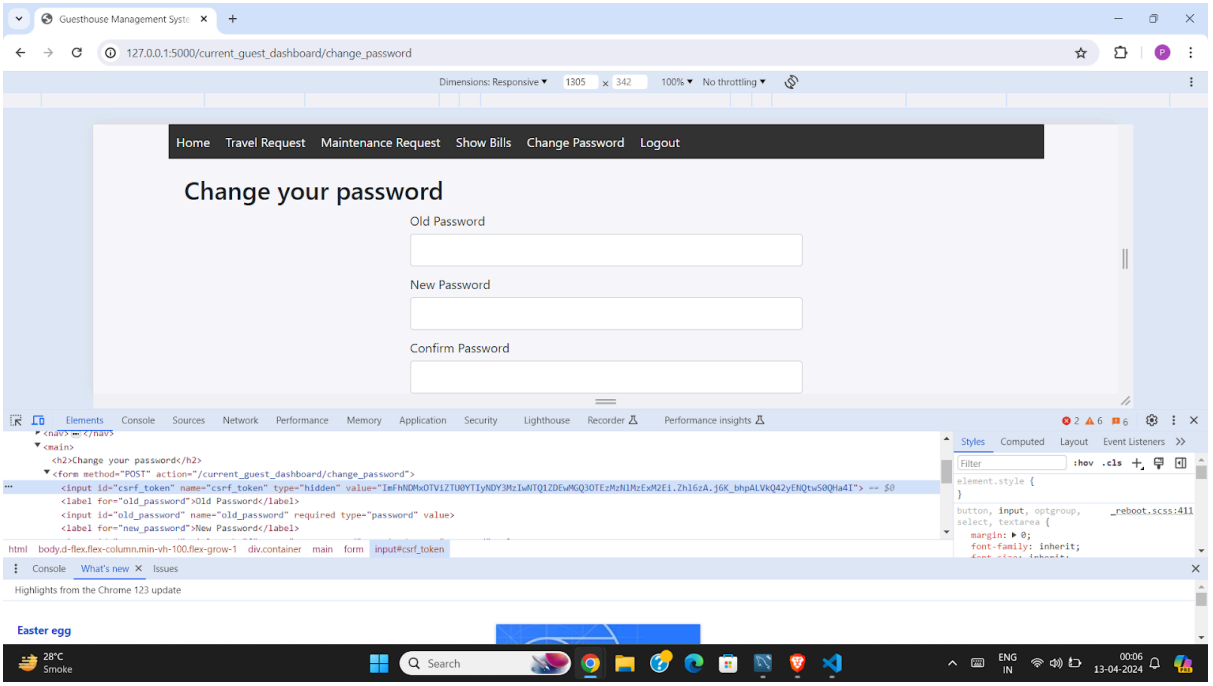
Next we tried to test for XSS attack in admin login where in Maintenance request form, I entered my own script. Upon sucessfull attack, we would recieve an alert pop-up of “XSS_Attack!” on the web-page.



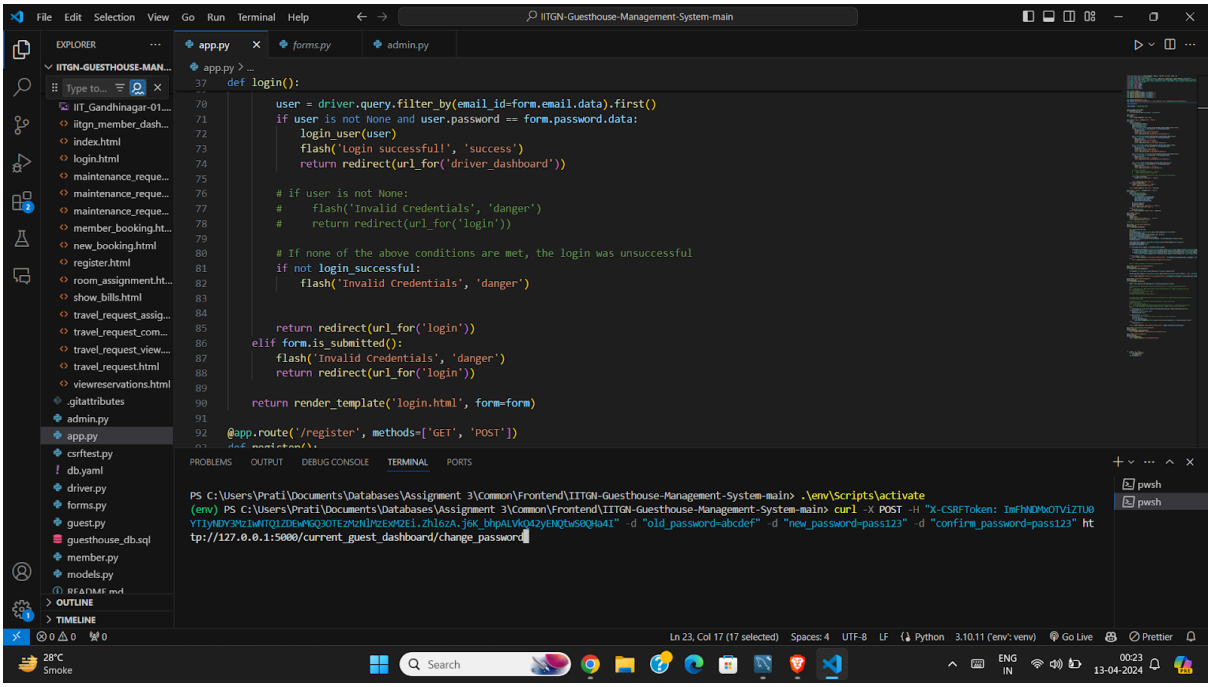
Our attempt to insert script is failed and the maintenance request form is submitted with our entered code as plain description text of the form. We could not conduct a sucessful XSS attack on the admin page.

We could not conduct XSS attacks sucessfully on our system, because Flask and its template engine Jinja2 have built-in protections against XSS. Flask-WTF provides built-in protection against XSS attacks by automatically escaping HTML entities when rendering form fields and Jinja2 automatically escapes HTML entities by default when rendering templates. Moreover, Modern web browsers like Google Chrome also have built-in XSS protection mechanisms that might prevent certain XSS attacks from being executed.

3. Cross-Site Request Forgery (CSRF) Attack:



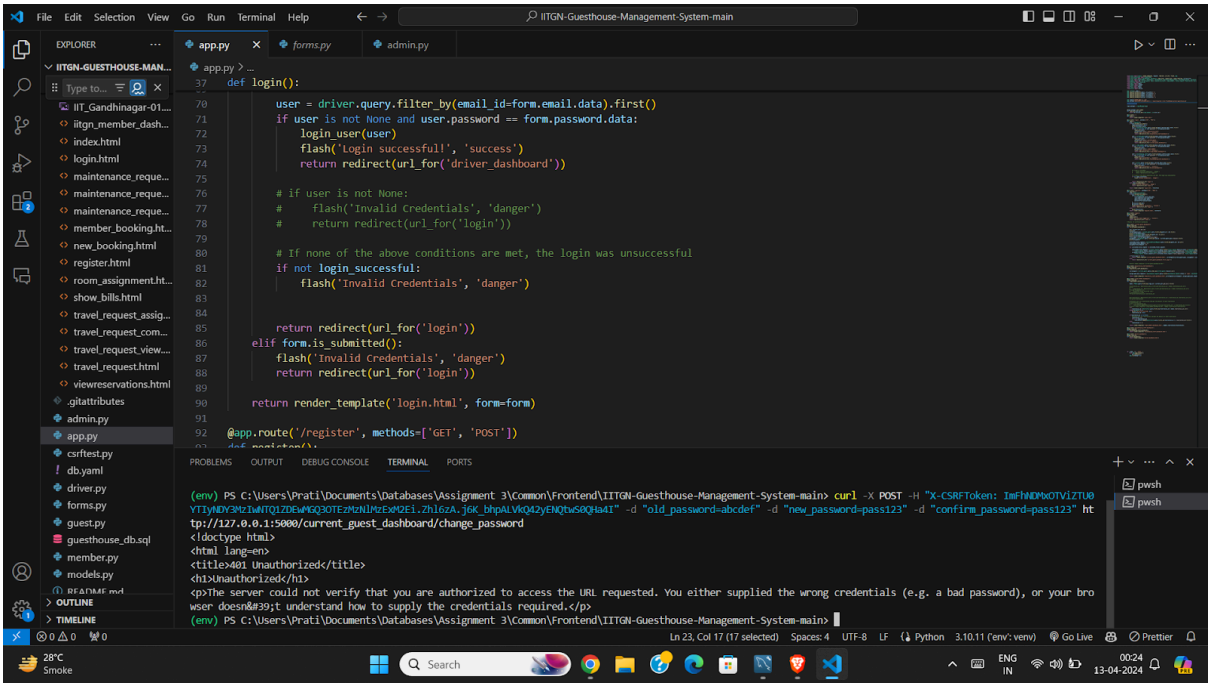
An external site might access the crsf_token for sensitive user information in the website, like changing passwords in the current guest account.



We tried to test if an external site can request to modify the user data without their information on their behalf.

```
curl -X POST -H "X-CSRFToken: ImFhNDMxOTViZTU0YTIyNDY3MzIwNTQ1ZDEwMGQ3OTEzMzNlMzExM2Ei.Zhl6zA.j6K_bhpALVkQ42yENQtwS0QHa4I" -d "old_password=abcdef" -d "new_password=pass123" -d "confirm_password=pass123" http://127.0.0.1:5000/current_guest_dashboard/change_password
```

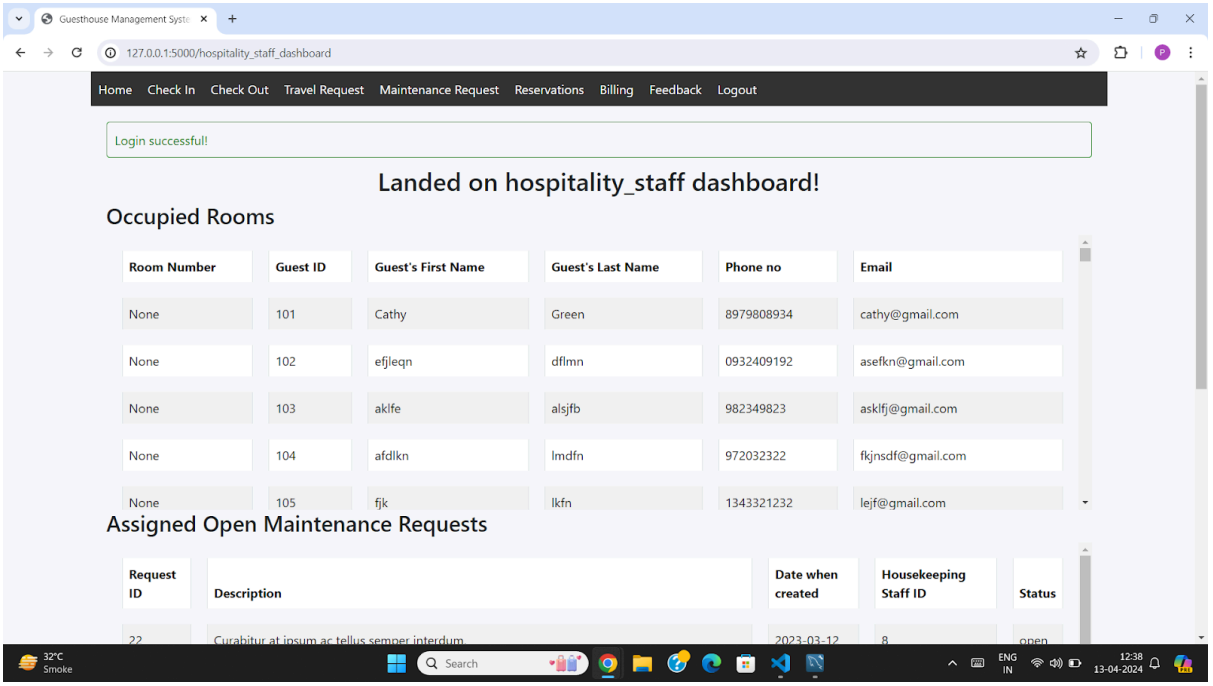
We executed the above code to try to change the current password of the guest (“abcdef”) to a new password (“pass123”).



We are unauthorized in changing the password for the user externally, and their data is protected.

Flask extensions such as Flask-WTF render a CSRF token that is automatically generated and included in the form. This token is unique per session and form. Therefore, Flask-WTF automatically adds CSRF protection to all forms created with it.

4. **Path Traversal Attack**



We are looking that if a housekeeping staff is able to hack into to the system and access sensitive information of current guests and members through locally stored files.

```python
import requests

# The URL of the webpage you're testing
url = 'http://127.0.0.1:5000/hospitality_staff_dashboard'

# A list of payloads to test
payloads = ['../', '../../', '../../../', '../../../../', '../../../../../']

user_input = input("Enter your input: ")
# Iterate over each payload
for payload in payloads:
    # Send the request to the URL with the payload as a parameter
    response = requests.get(url, params={'input': payload + user_input})
    # Print the response
    print(response.text)

    # If the payload is in the response, the application is vulnerable
    if 'root:x:0:0:' in response.text:
        print('The application is vulnerable to path traversal')
    else:
        print('The application is not vulnerable to path traversal')
```

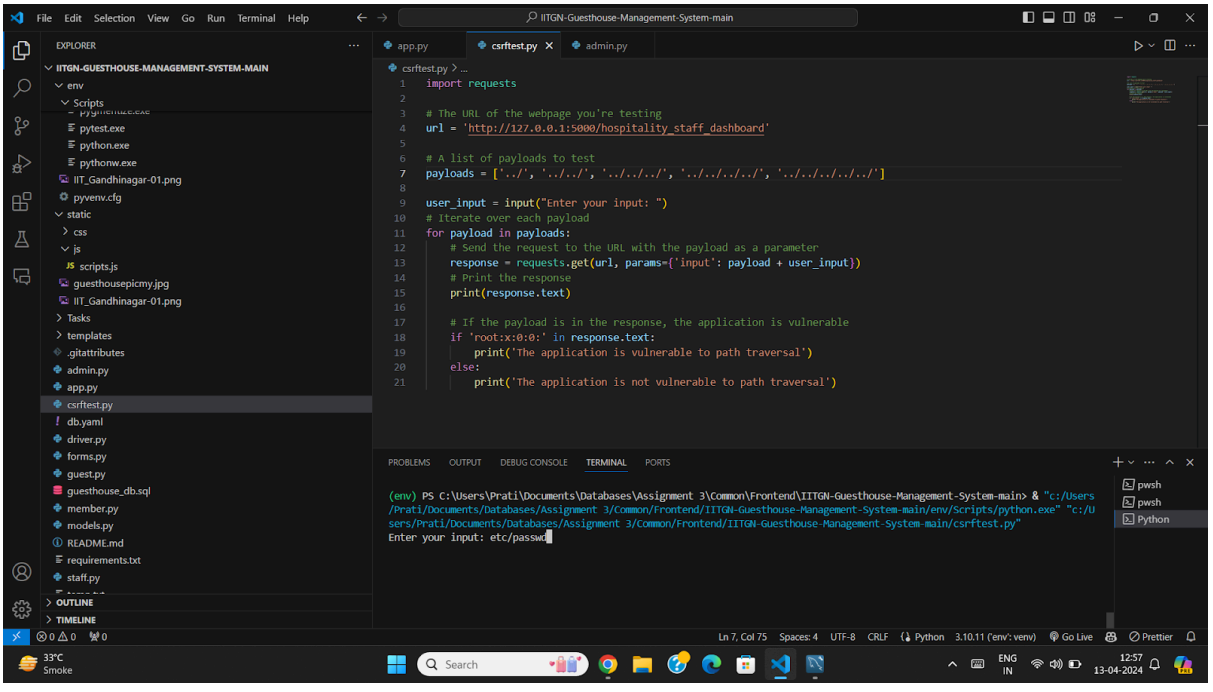
Code to test for path traversal attack

1. To test for file traversal attack, we imported the requests library, which is a popular Python library that makes HTTP requests.

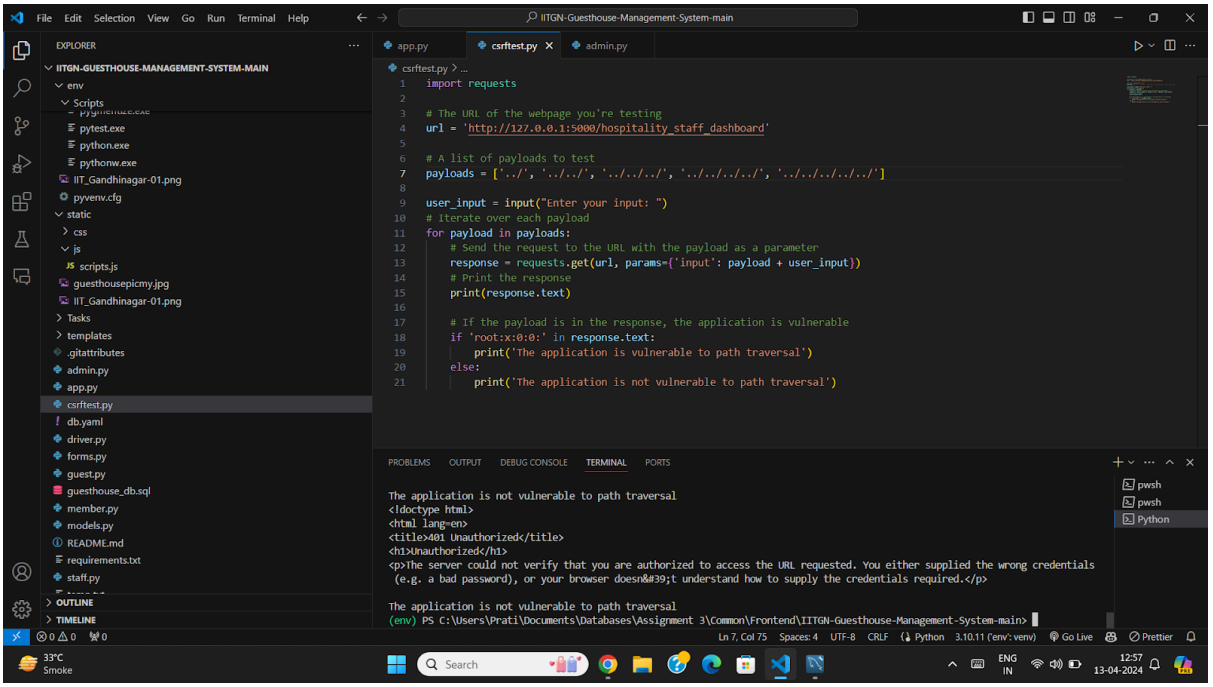
2. Setting the target URL: The URL of the web page you're testing is defined. In this case, it's set to 'http://127.0.0.1:5000/hospitality_staff_dashboard'.
3. Defining the payloads (../): A list of payloads to test is defined. These payloads are common path traversal sequences that attempt to move up in the directory structure.
4. Getting user input: The script asks for user input. This input is expected to be the remaining part of the path after the payload
5. Sending the requests and checking the responses: The script then enters a loop where it iterates over each payload. For each payload, it:
    a. Sends a GET request to the URL with the payload and user input as a parameter.
    b. Prints the response.
    c. Checks if the response includes sensitive information. If it does, it prints a message indicating that the application is vulnerable to path traversal. If it doesn't, it prints a message indicating that the application is not vulnerable to path traversal.

a. `etc/passwd`

This is a Unix file that contains user account information. It should not be accessible through a web application.

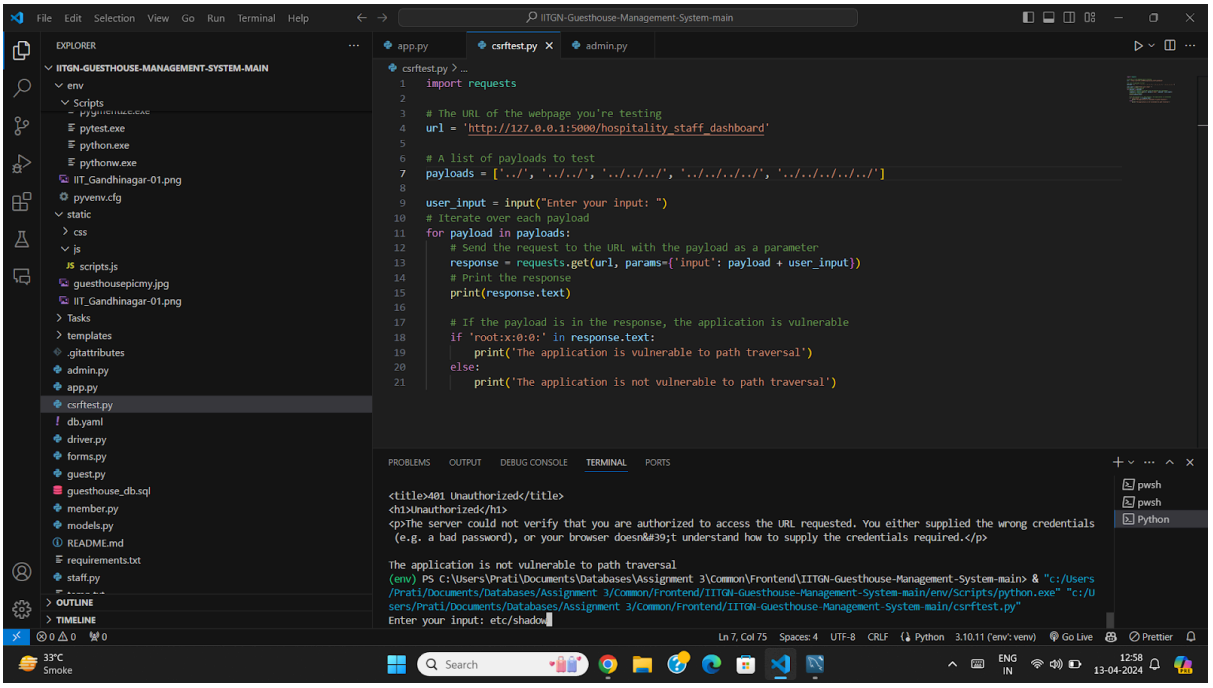


We tried to insert user input to test on the terminal

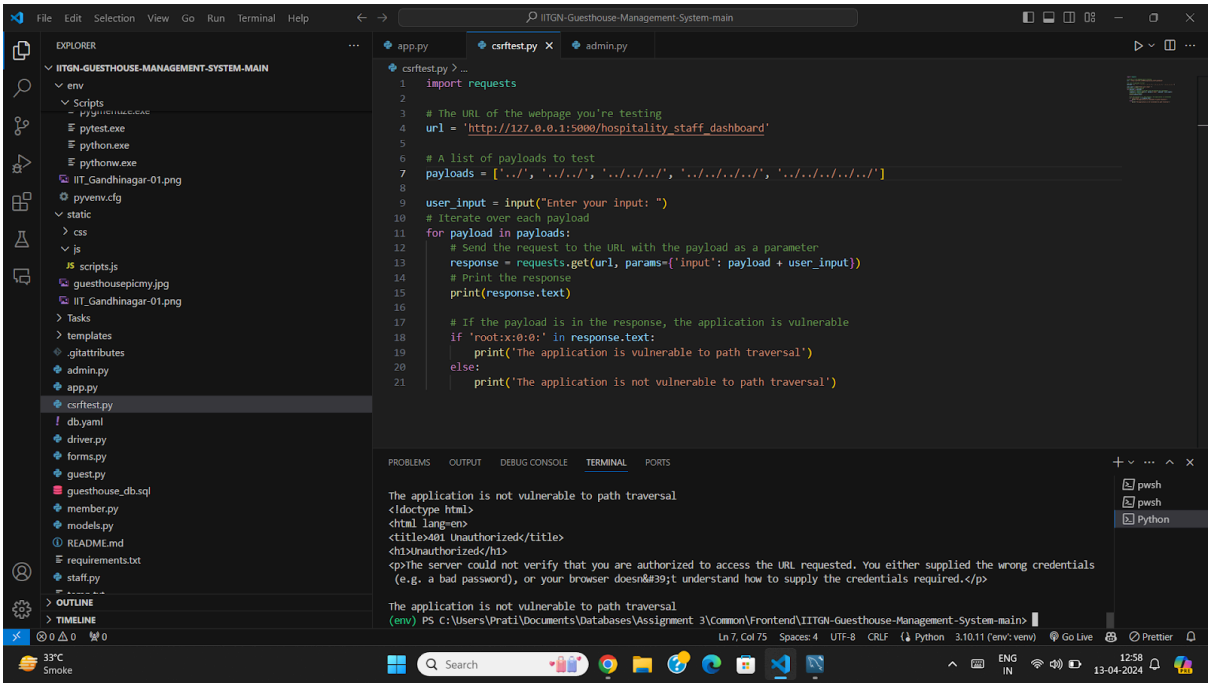


We received an unauthorized error response, and we could not access sensitive information

b. `etc/shadow`



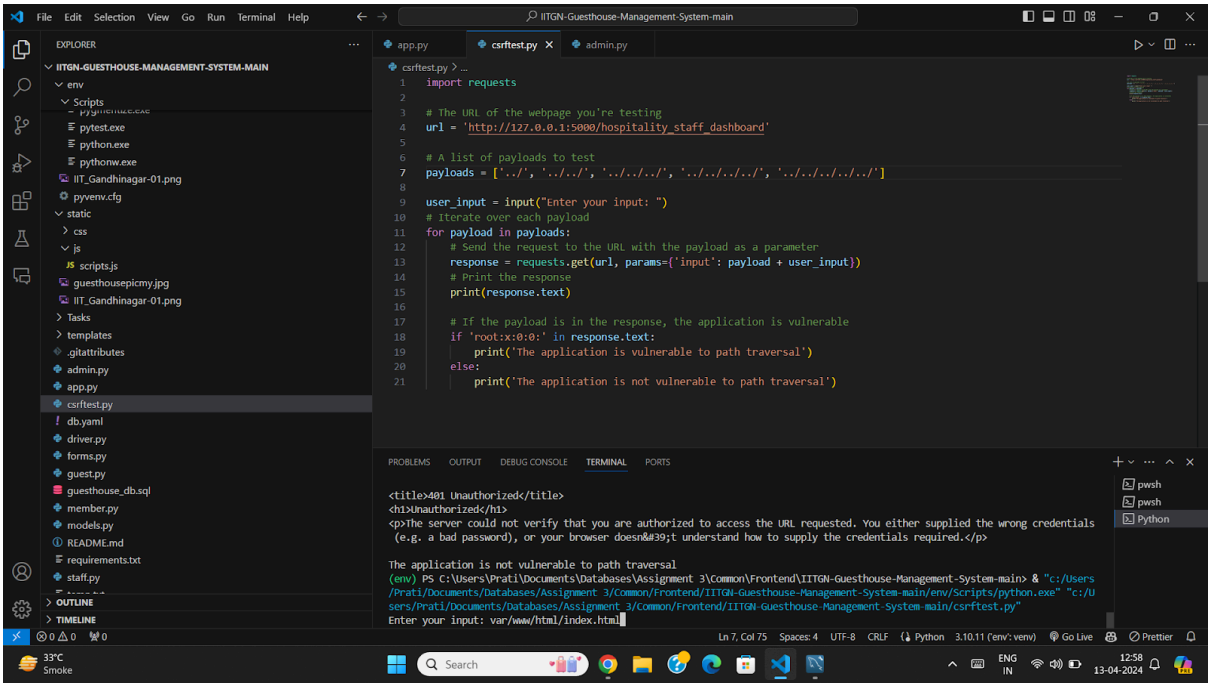
We tried to insert user input to test on the terminal



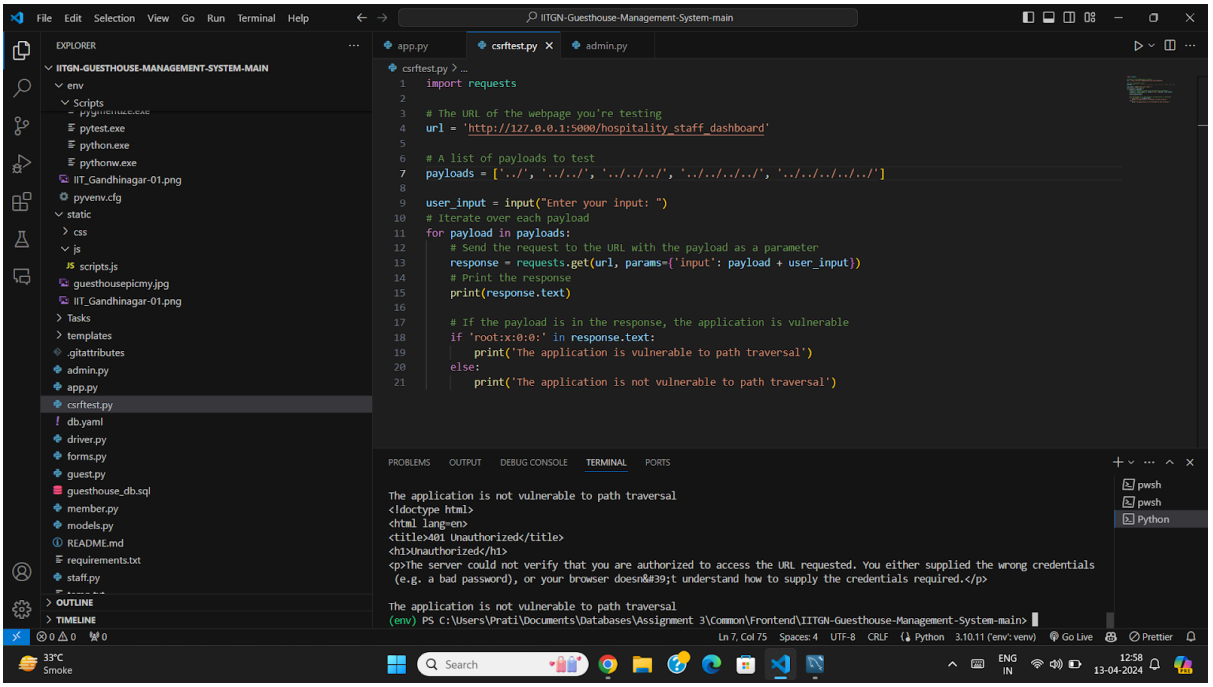
We received an unauthorized error response, and we could not access sensitive information This is another Unix file that contains encrypted password information. It should also not be accessible.

c. `var/www/html/index.html`

This is a common location for the main HTML file of a web server. If you can access this file, it might indicate a path traversal vulnerability.



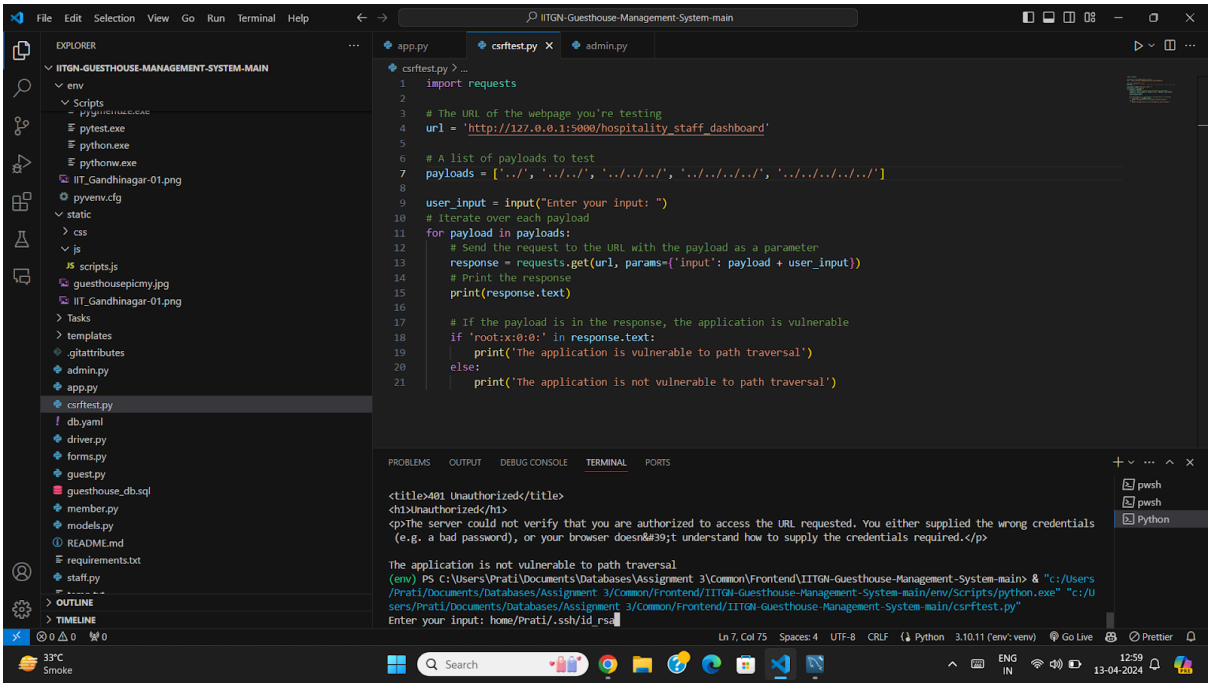
We tried to insert user input to test on the terminal



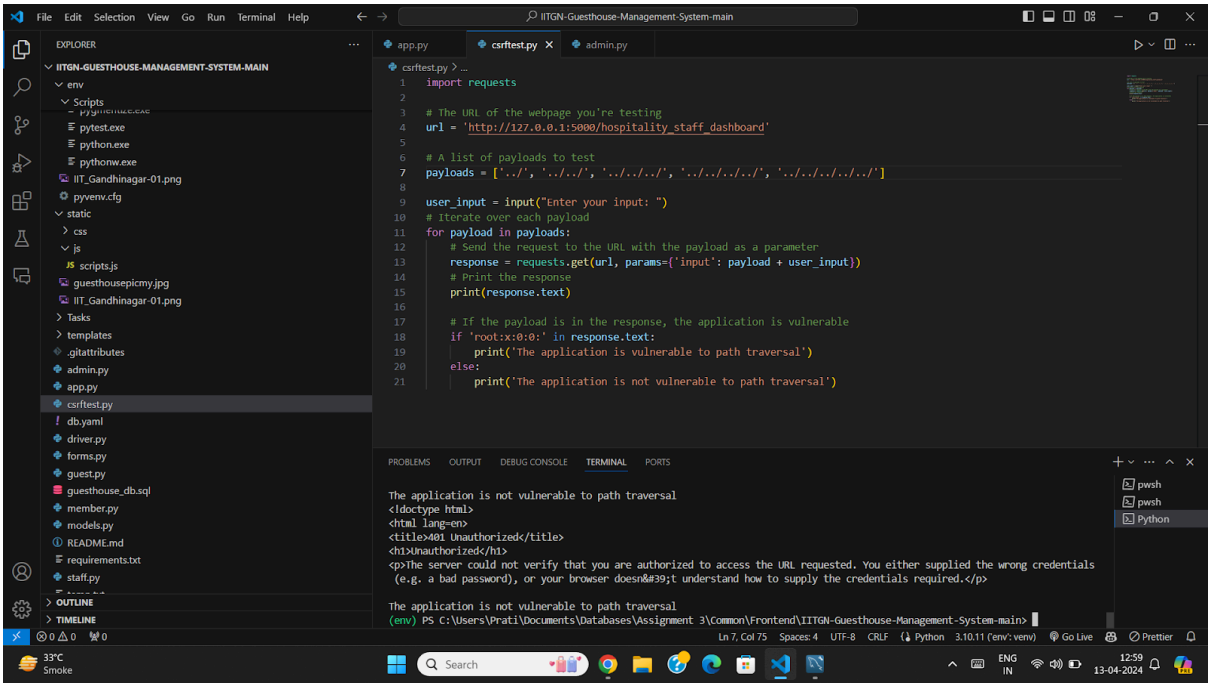
We received an unauthorized error response, and we could not access sensitive information

d. 4. `home/<username>/.ssh/id_rsa`

This is the location of the private key for SSH on a Unix system. If you can access this file, it's a serious security vulnerability.



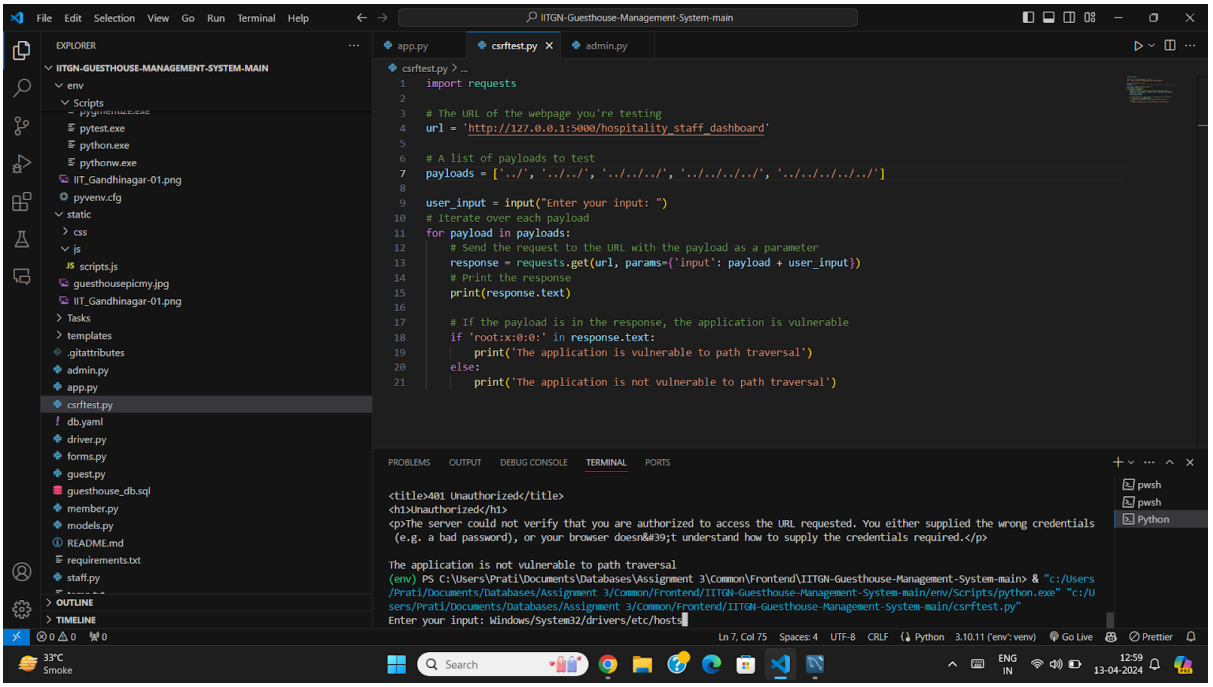
We tried to insert user input to test on the terminal



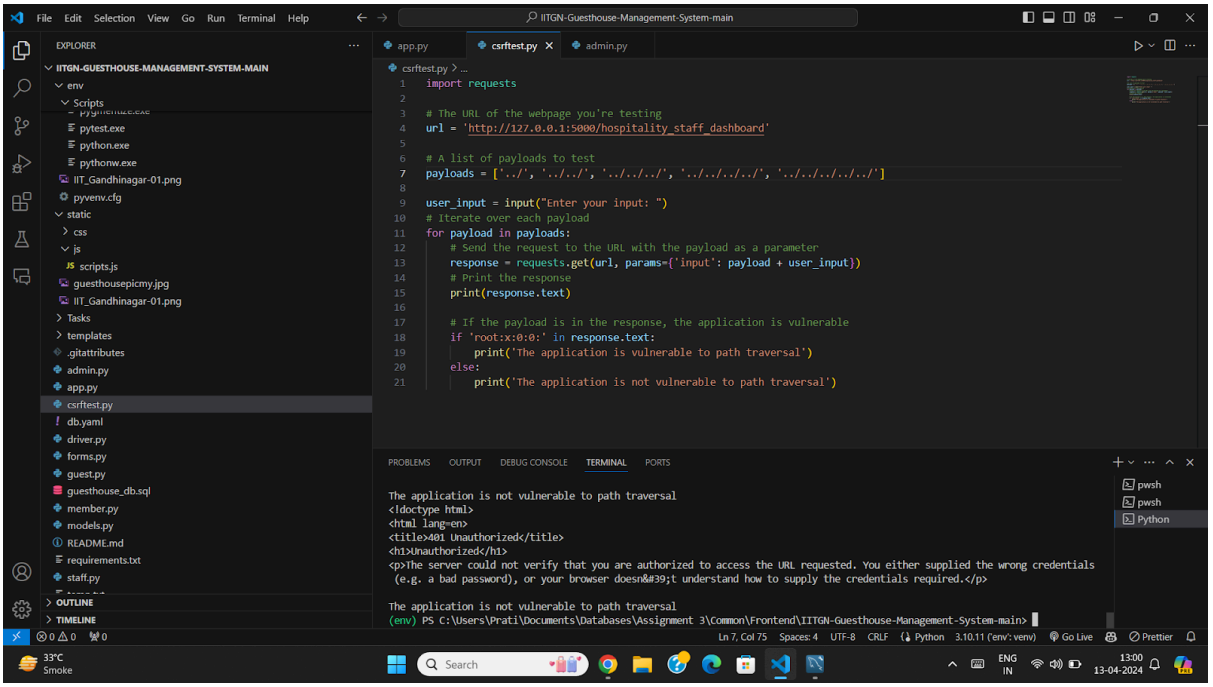
We received an unauthorized error response, and we could not access sensitive information

e. `Windows/System32/drivers/etc/hosts`

This is the location of the hosts file on a Windows system. If you can access this file, it might indicate a path traversal vulnerability.



We tried to insert user input to test on the terminal



We received an unauthorized error response, and we could not access sensitive information

The Flask web framework, has several built-in features that help protect against path traversal attacks:

URL Routing: Flask uses URL routing to map URLs to Python functions (known as routes). When a request is made to a URL, Flask executes the corresponding function and returns the result as a response. This means that the URLs in your application don't directly map to files or directories on your server, which makes path traversal attacks difficult.

Template Rendering: Flask uses the Jinja2 template engine to render views. When you call render_template, Flask looks for the specified template file in your templates folder, renders it, and returns the result as a response. This process is safe from path traversal attacks because Flask doesn't expose your file system structure to the client.

Form Handling: The request object in Flask provides a secure way to handle form data. When you call “request.form.get”, Flask retrieves the specified form field from the request. This process is safe from path traversal attacks because Flask doesn't use the form data to access files or directories on your server.

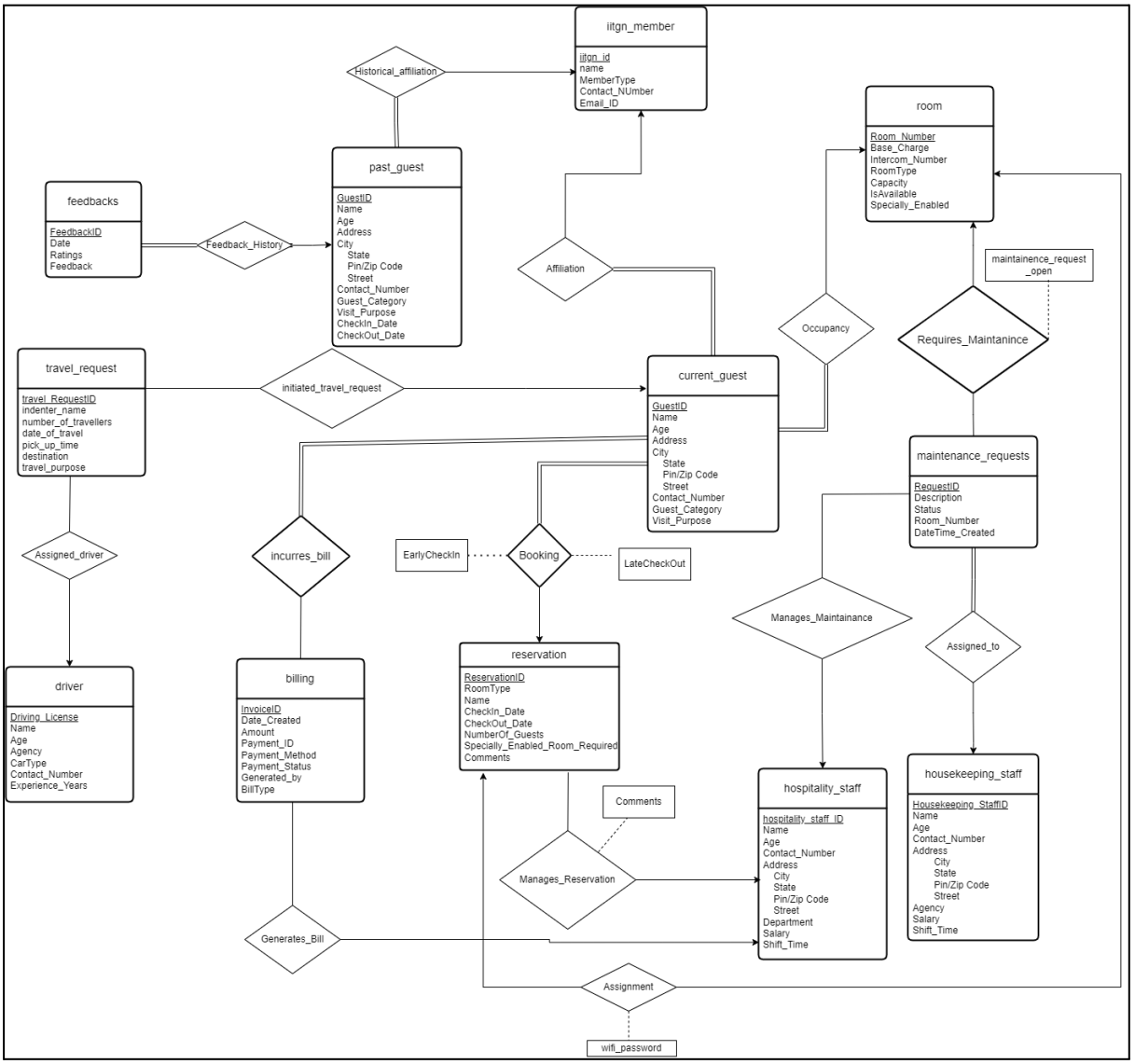
Database Access: Flask-SQLAlchemy, which you're using to interact with your database, uses SQLAlchemy's ORM (Object-Relational Mapping) to map Python classes to database

tables. This means that your database queries don't directly map to files or directories on your server, which makes path traversal attacks difficult.

Configuration: Flask's configuration system allows you to store sensitive information, like your secret key and database URI, in your application's configuration. This information is not exposed to the client, which helps protect against path traversal attacks.

2. Relations and their Constraints from feedback relation with ER Diagram in Assignment 1

ER Diagram as In Assignment 1



Relationships post Second Feedback

1. **`Occupancy:` current_guest** and **room**
2. **`affiliation:` current_guest** and **iitgn_member**
3. **`booking:` current_guest** and **reservation**
4. **`assignment:` room** and **reservation**
5. **`requires_maintainence:` room** and **maintainence_request**
6. **`manages_maintainence:` hospitality_staff** and **maintaince_request**
7. **`manages_reservation:` hospitality_staff** and **reservation**
8. **`assigned_to:` maintaince_request** and **housekeeping_staff**
9. **`incurres_bill:` guest** and **bill**
10. **`generates_bill:` hospitality_staff** and **bill**
11. **`travel_Request:` current_guest** and **travel_request**
12. **`assigned_driver:` travel_request** and **drivers**
13. **manage_travel_request: hospitality_staff and travel_Request**
14. **`feedback_history:` current_guest** and **feedback**

Validation with ER Diagram

1. Occupancy: current_guest and room

Constraint: The guest is always assigned a room, which may or may not be assigned to a particular guest.
Validity: This constraint is present and valid as it allows for the relationship between the current guest and the room.

2. Affiliation: current_guest and iitgn_member

Constraint: A current guest must be affiliated with the IITGN member.
Validity: This relationship is present and valid, adhering to the constraint.

3. Booking: current_guest and reservation

Constraint: A current guest must have a reservation under their name.
Validity: This relationship is present and valid as it allows for multiple reservations per guest.

4. Assignment: room and reservation

Constraint: A new room shall be assigned for every single reservation.
Validity: This relationship is present and valid, allowing for multiple reservations to be associated with a room.

5. Requires_maintenance: room and maintenance_request

Constraint: A room can require maintenance, and there can be multiple maintenance requests for different rooms.
Validity: This relationship is present and valid, adhering to the constraint.

6. Manages_maintenance: hospitality_staff and maintenance_request

Constraint: Hospitality staff can manage multiple maintenance requests.
Validity: This relationship is present and valid, allowing staff to manage multiple maintenance requests.

7. Manages_reservation: hospitality_staff and reservation

Constraint: Hospitality staff can manage multiple reservations.
Validity: This relationship is present and valid, adhering to the constraint.

8. Assigned_to: maintenance_request and housekeeping_staff

Constraint: Maintenance requests can be assigned to multiple housekeeping staff members.
Validity: This relationship is present and valid, allowing for multiple staff members to be assigned to maintenance requests.

9. Incurs_bill: guest and bill

Constraint: A guest can incur multiple bills.
Validity: This relationship is present and valid, allowing for multiple bills to be associated with a guest.

10. Generates_bill: hospitality_staff and bill

Constraint: Hospitality staff can generate multiple bills.
Validity: This relationship is present and valid, adhering to the constraint.

11. Travel_Request: current_guest and travel_request

Constraint: A current guest can have multiple travel requests.
Validity: This relationship is present and valid, allowing for multiple travel requests per guest.

12. Assigned_driver: travel_request and drivers

Constraint: A travel request must be assigned to a single driver.
Validity: This relationship is present and valid, allowing for drivers to be assigned to a travel request.

13. Manage_travel_request: hospitality_staff and travel_request

Constraint: Hospitality staff can manage multiple travel requests.
Validity: This relationship is present and valid, adhering to the constraint.
Feedback_history: current_guest and feedback

Constraint: A current guest can have multiple feedback entries.
Validity: This relationship is present and valid, allowing for multiple feedback entries per guest.

NOTE:

In app.py, line 88 add these lines,

```
GOOGLE_CLIENT_SECRET = "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"
GOOGLE_CLIENT_ID =
"765986257146-bc8cb5lv7iu0ki1437lr9o9ag8ggcd9n.apps.googleusercontent.com"
```

4. Contributions:

1. **Gaurav Shah** (**Group Leader**)

- Completed all the changes in the WebApp as mentioned in both feedbacks.
- Created the final report.

2. **Soham Rahatal**

- Added locks for some database tables for concurrent multi-user access
- Added the Google authentication for the admin and iitgn_member
- Collected feedback from stakeholders with Gaurav & Pratik
- Involved in the drafting of the final report.

3. **Pratik Agrawal**

- Conducted tests for SQL Injection, XSS, CRSF, & Path Traversal attacks
- Completed Validity of Relationships after feedback with ER diagram
- Collected feedback from stakeholders with Gaurav & Soham
- Involved in the drafting of the final report.

4. **Rohit Srivastava**

- NO CONTRIBUTION

5. **Banavath Diraj Naik**

- Q2 in Responsibility for G1

6. **Sohitha Sonalika**

- Q2 in Responsibility for G1

7. **Shivamani**

- NO CONTRIBUTION

Sub-groups:

G1: Pratik Agarwal, Banavath Diraj Naik, Sohitha Sonalika, Rohit Srivastav

G2: Gaurav Shah, Soham Rahatal, Shivamani

References:

1. https://guesthouse.iitgn.ac.in/booking.php
2. https://guesthouse.iitgn.ac.in/accommodation_facility.html

---

The End