
# CS 432
# Databases

## Assignment IV

## DEPLOYING THE DBMS

## QUERY CRAFTERS

### Group Members


| | |
|---|---|
| Saurabh Kumar Sah | 21110188 |
| Shreya Patel | 21110155 |
| Riya Jain | 21110178 |
| Twinkle Devda | 21110228 |
| Ishika Raj | 21110081 |
| Saumya Jaiswal | 21110186 |
| Darsh Dalal | 21110049 |
| Het Trivedi | 21110226 |

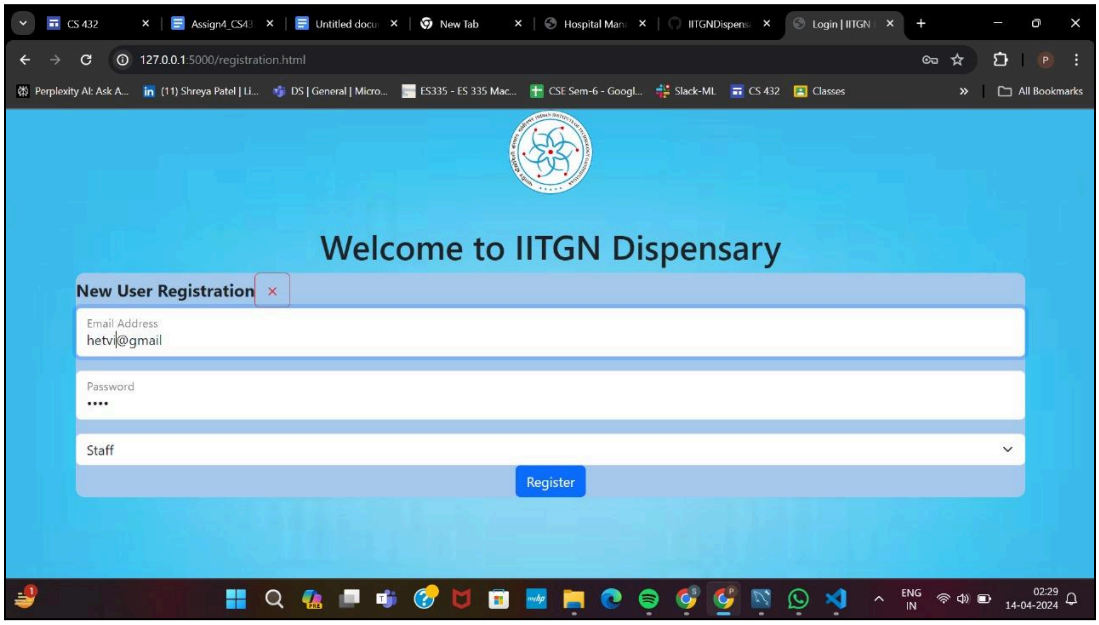

**April 16, 2024**

## Responsibility of G1

**Q1.**

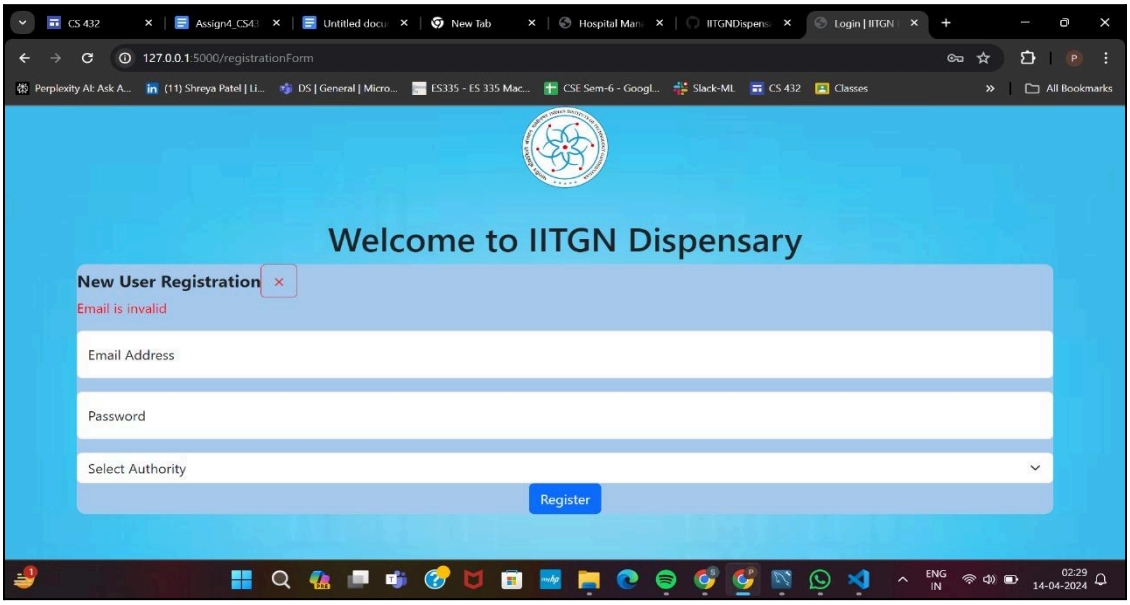
1. Changes in the Registration Form

    a. Before first feedback



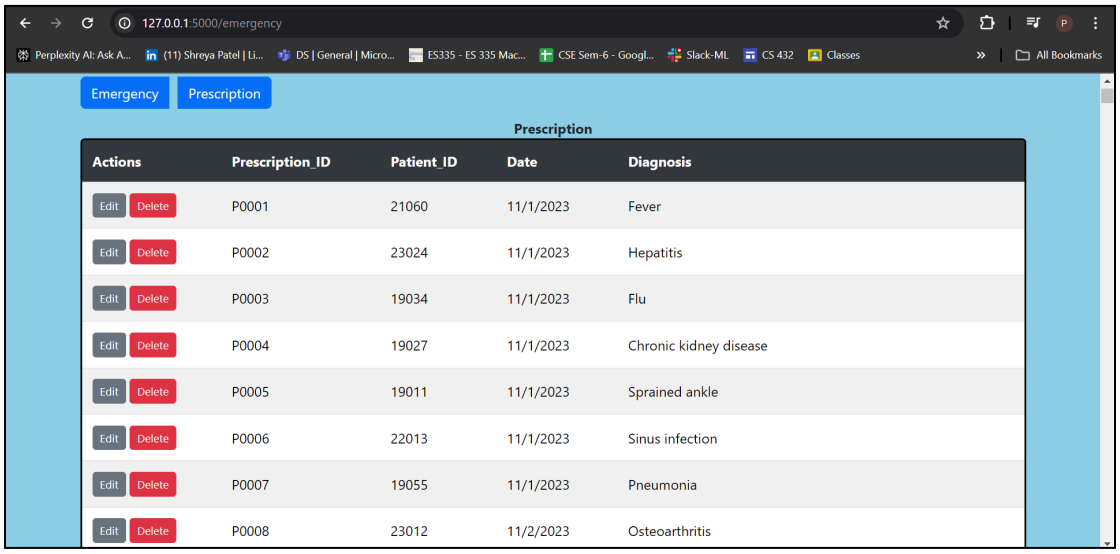

Anyone can register with an invalid email address and its entry also gets added to the authentication table. However, allowing registration with invalid email addresses can pose security risks and undermine the integrity of the authentication process.

b. After the first feedback



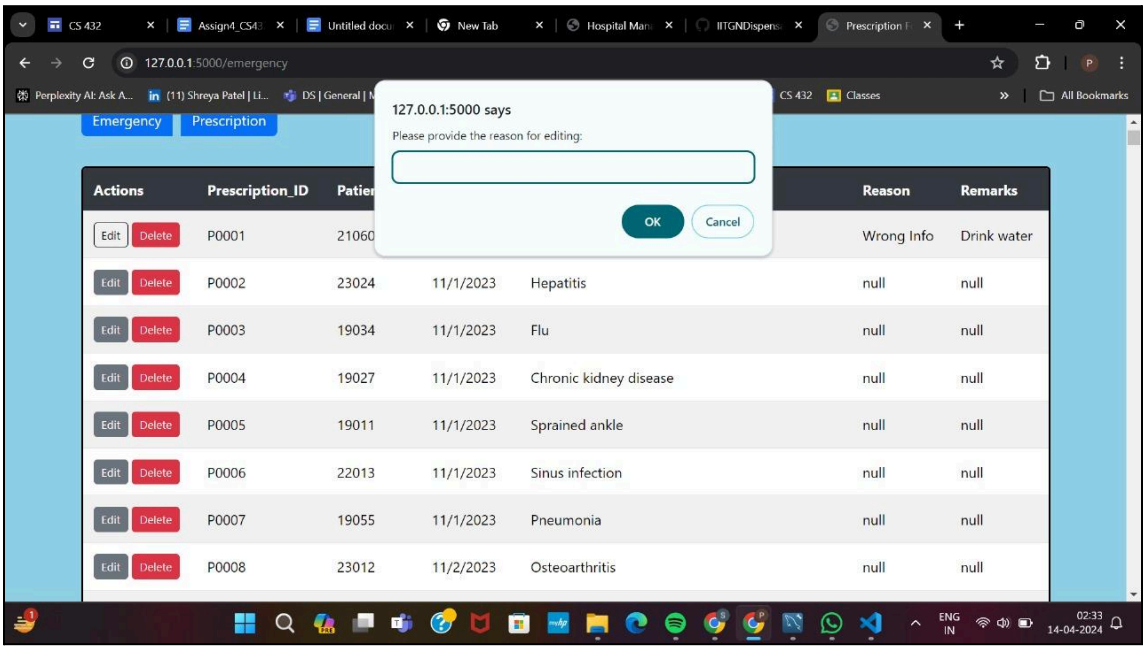
Following updates to the registration page, attempting to register with an invalid email address will prompt the message "**Email is invalid**." This enhancement aims to improve user experience by providing immediate feedback guiding users towards providing valid email addresses.

2. Added the reason and remarks column
   a. Before Feedback




Before the changes, users were not prompted to provide a reason for editing prescriptions, and doctors lacked the option to include additional remarks. By incorporating these features, the system now fosters better transparency, enabling users to provide context for edits and doctors to offer supplementary information, enhancing the overall quality of care.

   b. After Feedback

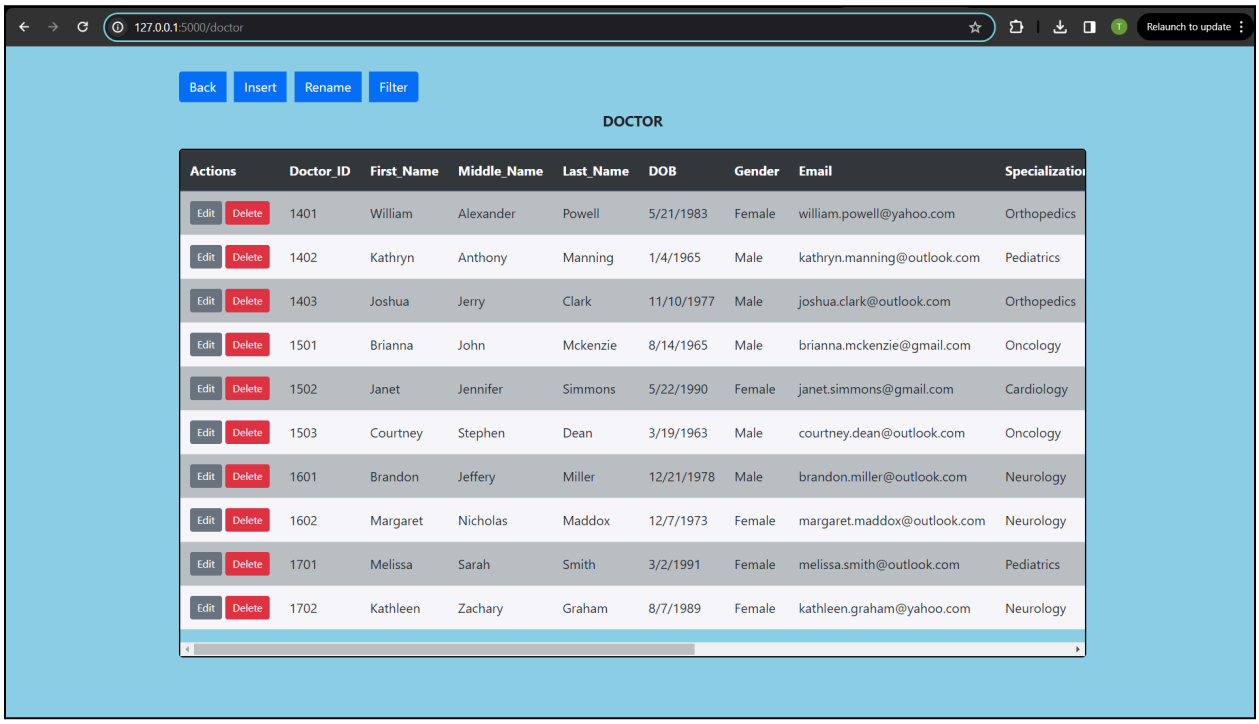


Here, upon clicking the edit button, a dialogue box prompts the user to input the reason for the modification. Once submitted, this reason is recorded alongside the edited prescription in the "Reason" column. An extra column titled "Remarks" also allows doctors to provide supplementary information or comments. These enhancements streamline the editing process, facilitate better communication, and ensure all relevant information is captured within the prescription record.

3. Staff and Doctor can see the OPD checked by the Doctor

a. Before Feedback

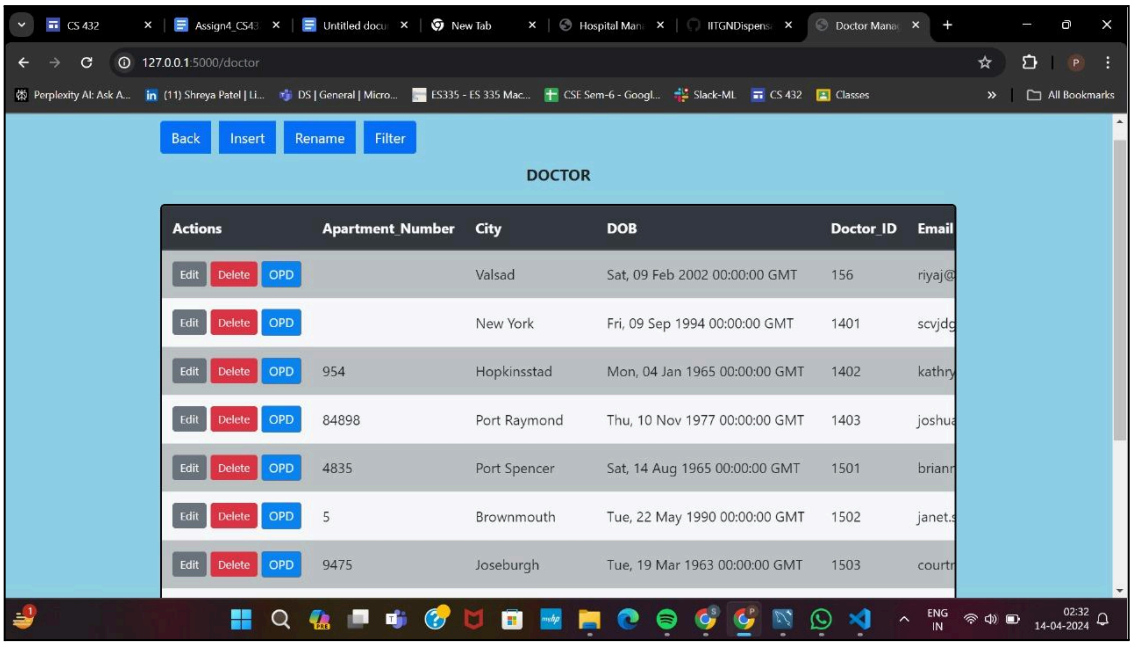
In the image below, we can see that there is no such option to track which patients a particular doctor has attended to in the OPD for both the current day and cumulative records.

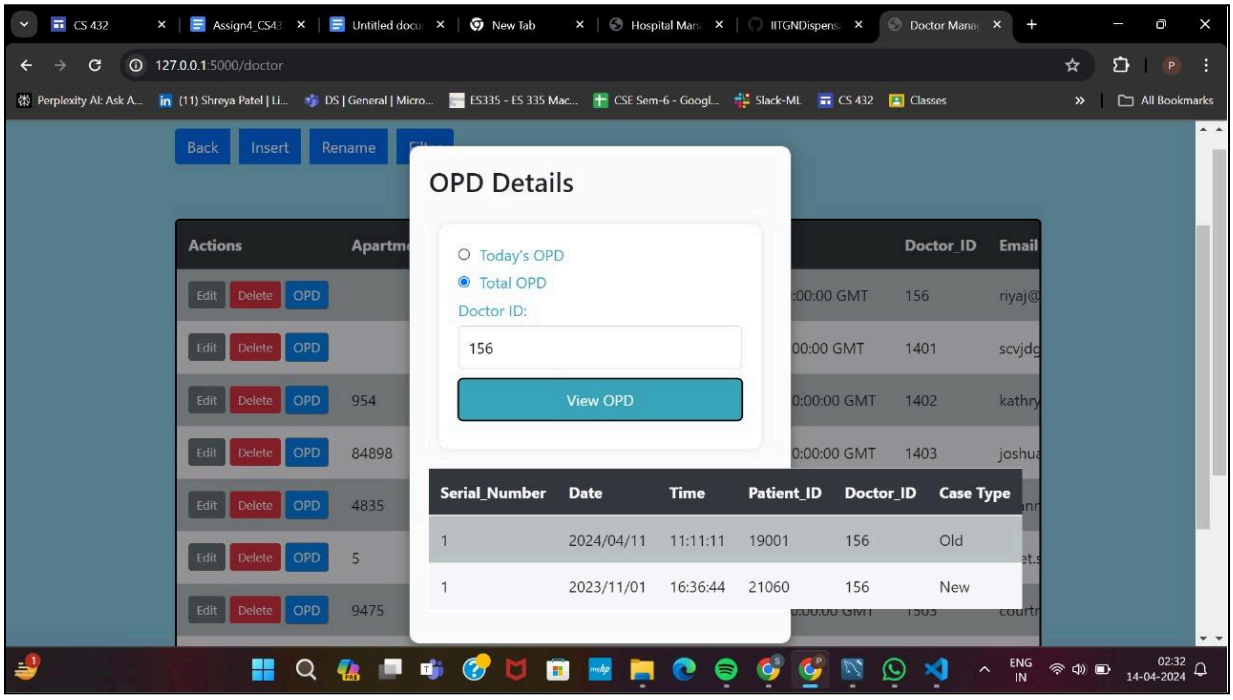


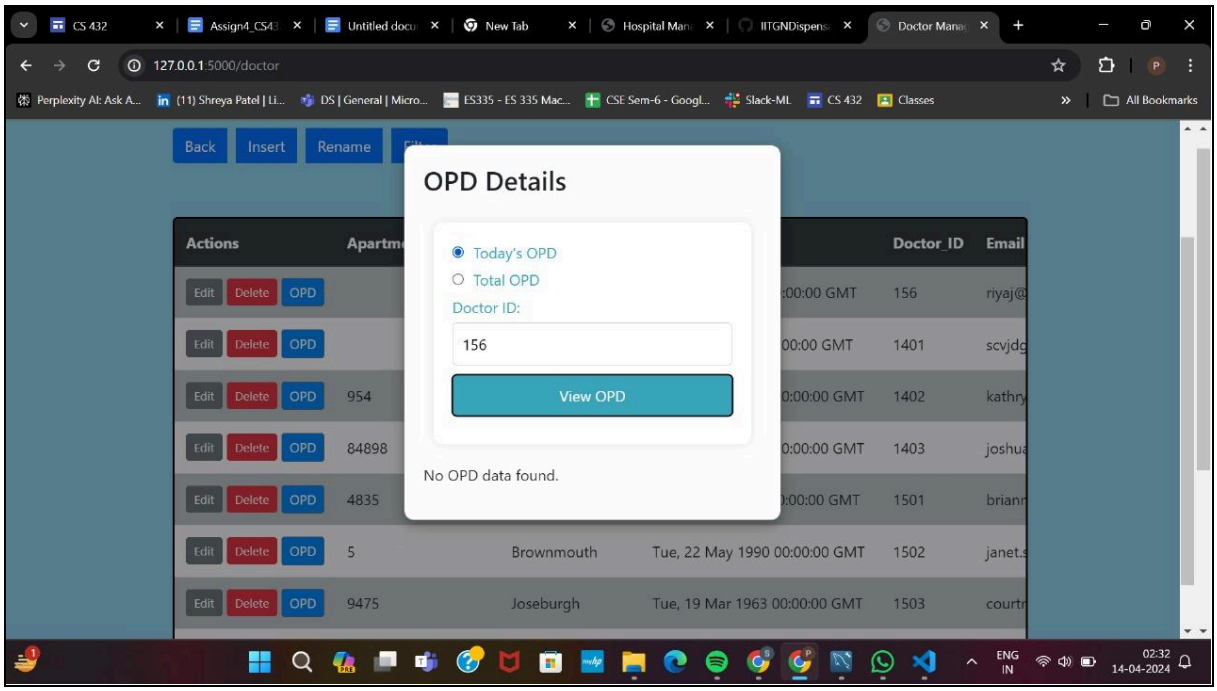

b. After Feedback

We've introduced the "**OPD**" option, triggering a dialogue box when clicked. Within this box, users can select whether they wish to view the total patients checked by the doctor either for the current day or cumulatively till the present. Users can access the OPD entries corresponding to the selected doctor upon submission. If there are no entries for that doctor in the OPD, the system will display a message stating **"No OPD data found,"** ensuring clarity and transparency

in information retrieval.

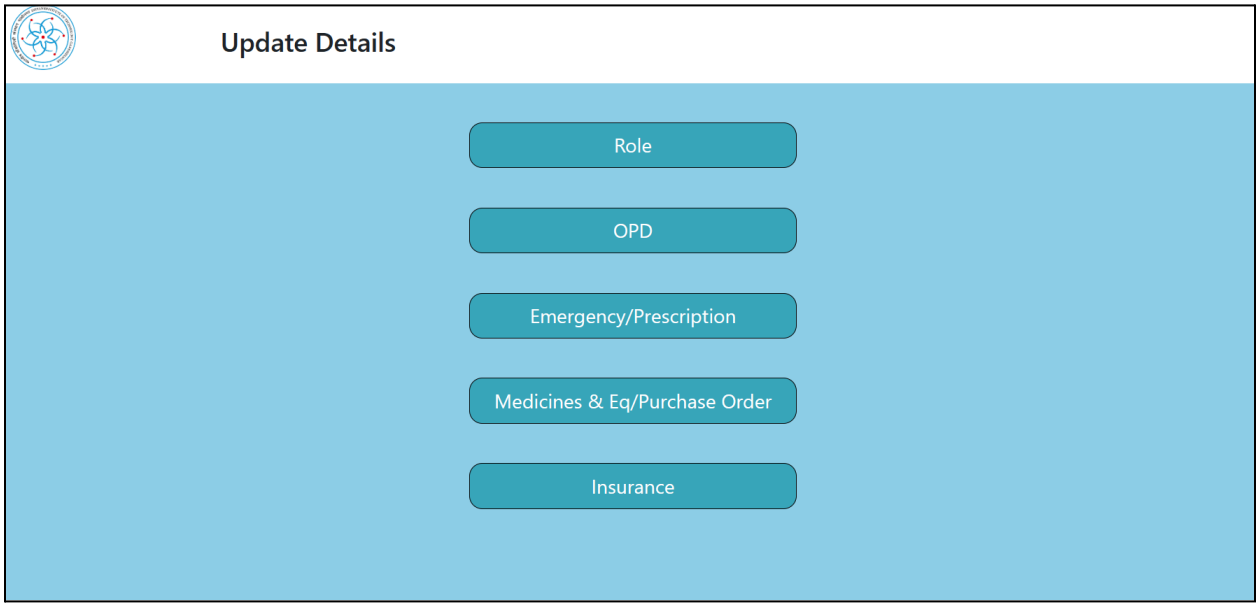




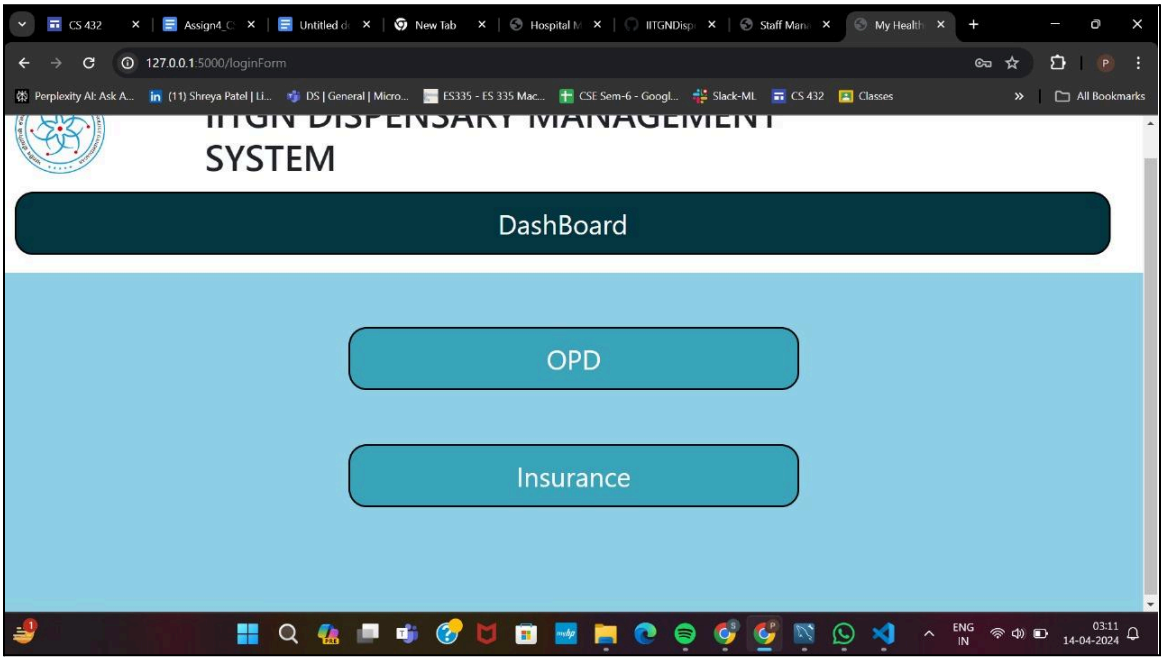

4. Remove view access of some tables to the patient

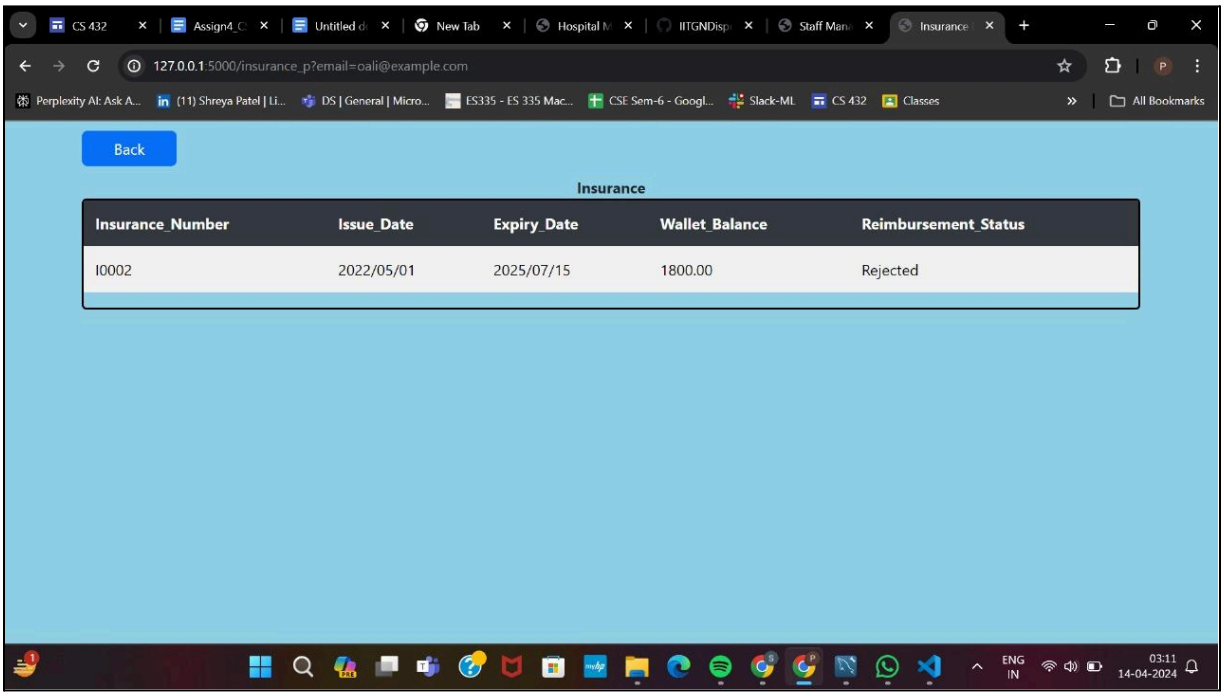
   a. Before Feedback

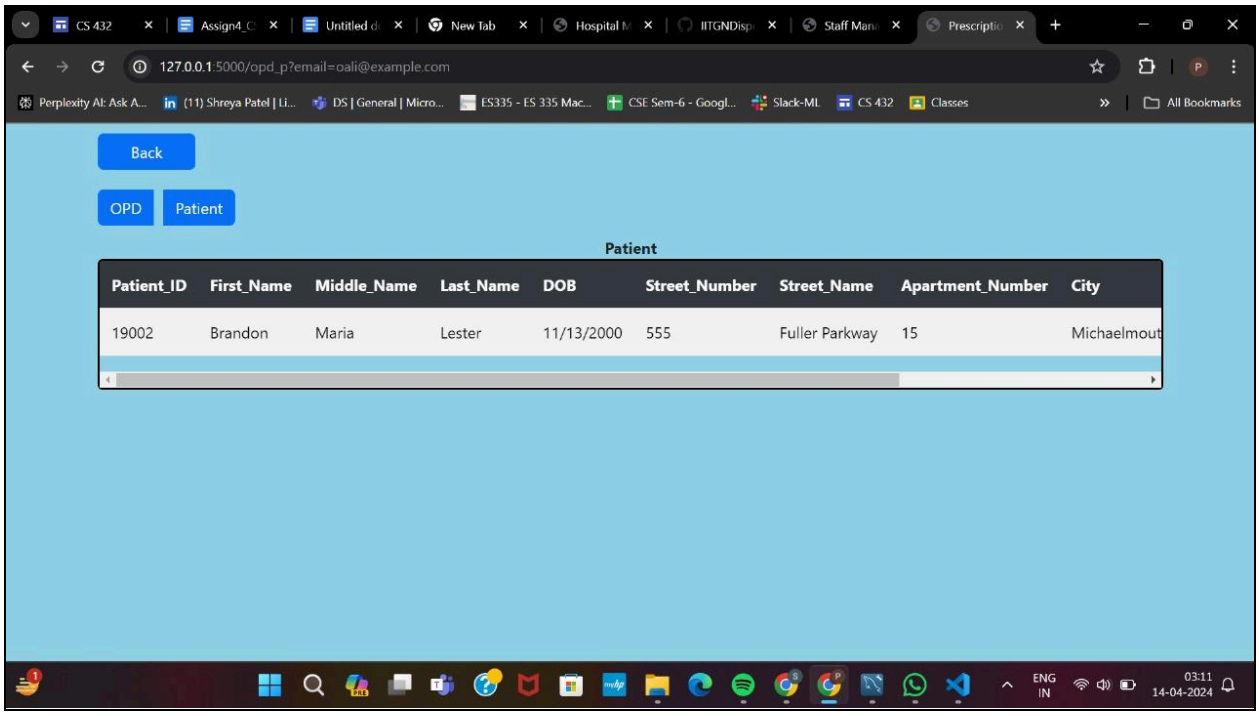


Initially, patients had view access to all tables, but following discussions with stakeholders, it was determined that patients should only have access to specific tables: OPD, Insurance, and Patient. In these tables, patients can only view entries corresponding to their data, ensuring privacy and confidentiality.

b. After Feedback



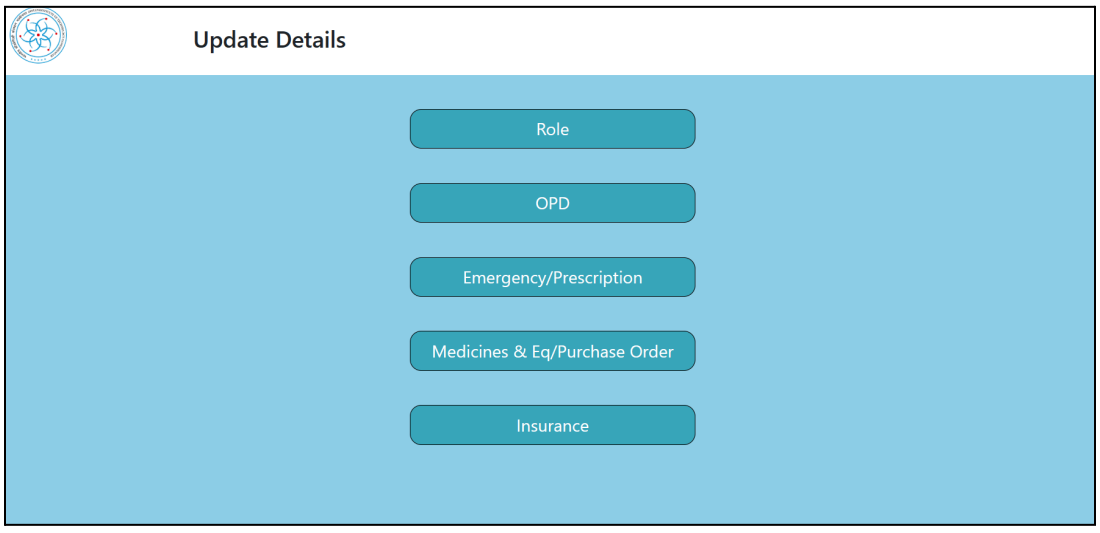






Patients are limited to exclusively viewing the Patient, OPD, and Insurance tables. Moreover, patients can access their entries within these tables, ensuring privacy and confidentiality. This restriction ensures that patients can only access information pertinent to their own healthcare records, aligning with privacy regulations.

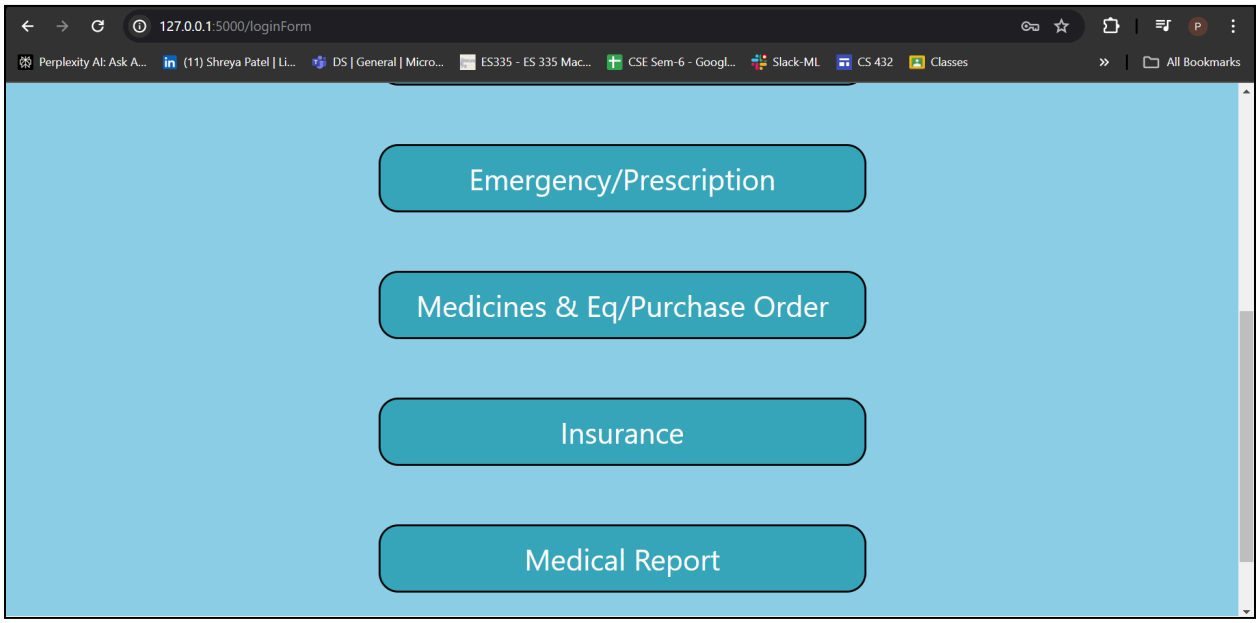
5. Added new table to show the entries for the medical test report

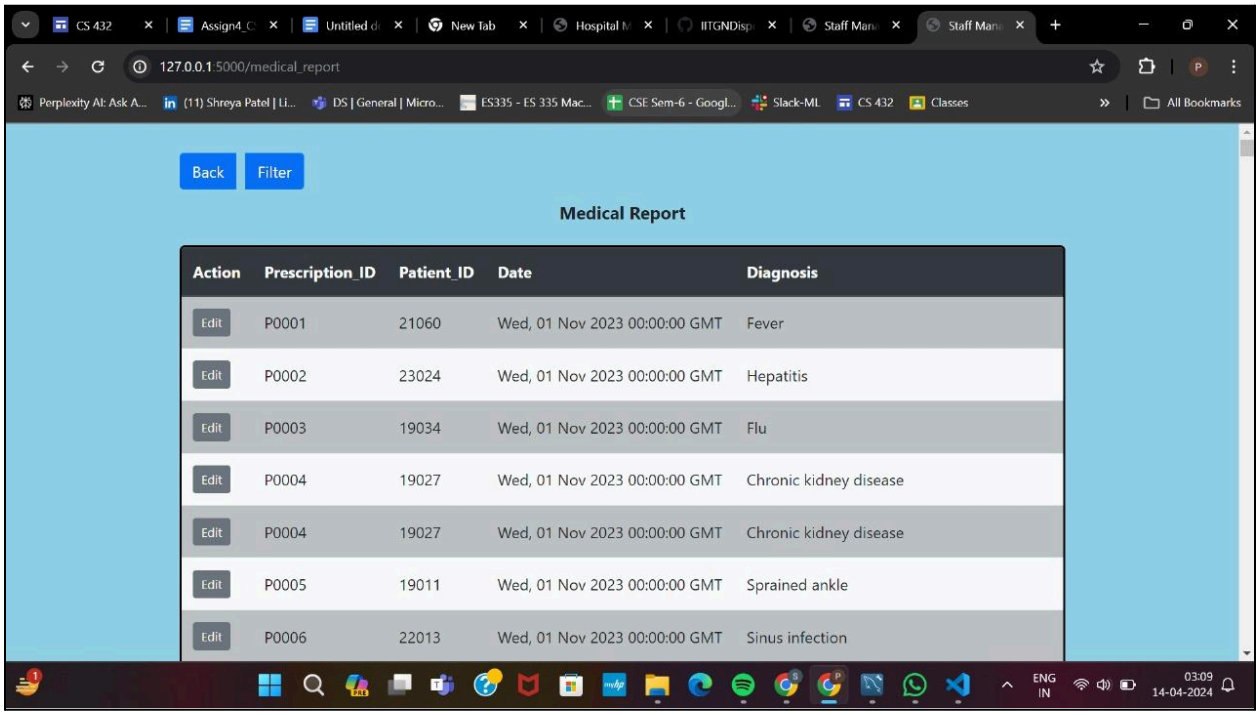
a. Before Feedback



As you can see that initially there is no option to see the information regarding the medical report.

b. After Feedback




The Medical Report table contains essential information such as patient ID, prescription ID, date of test, diagnosis, test name, and result. Additionally, there is an option to edit entries, facilitating

adjustments by staff or doctors. This edit functionality is particularly useful for updating test results from "Pending" to "Positive" or "Negative" based on the outcome, ensuring accurate and up-to-date medical records.

Q2.

We are dealing total 3 types of the users :

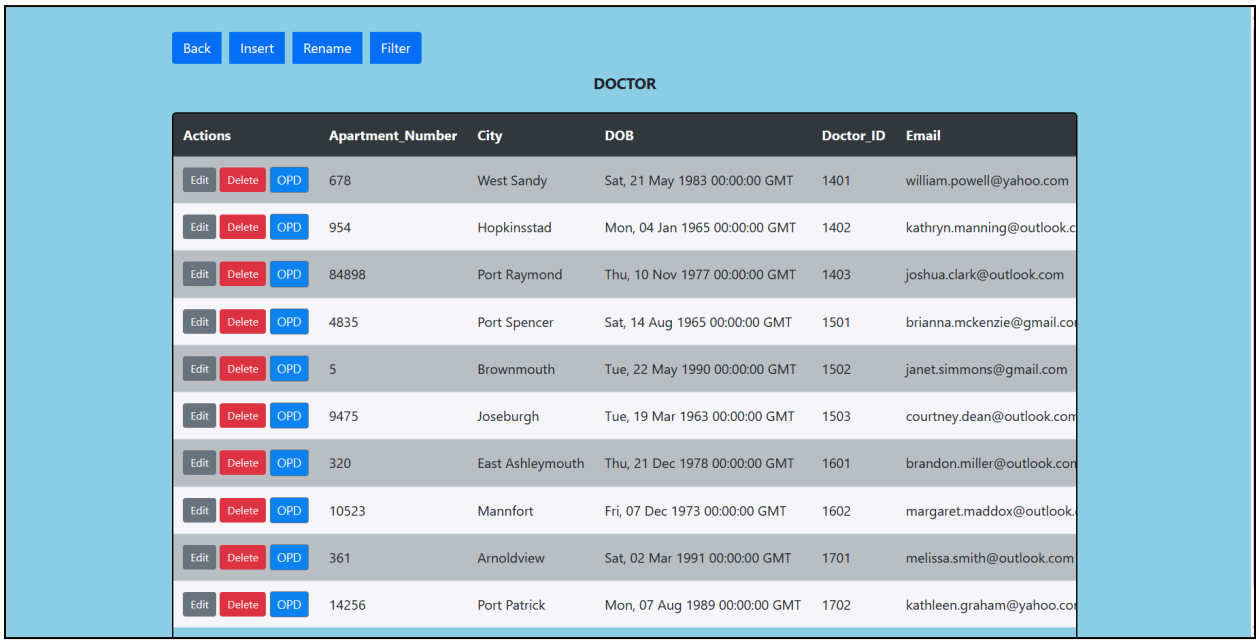
1) Staff: Responsible for administrative and operational tasks within the healthcare facility. They might handle patient registrations, appointments, and general administrative duties.

2) Doctor: Professionals who diagnose, treat, and provide medical care to patients. They may prescribe medications, order tests, and oversee treatment plans.

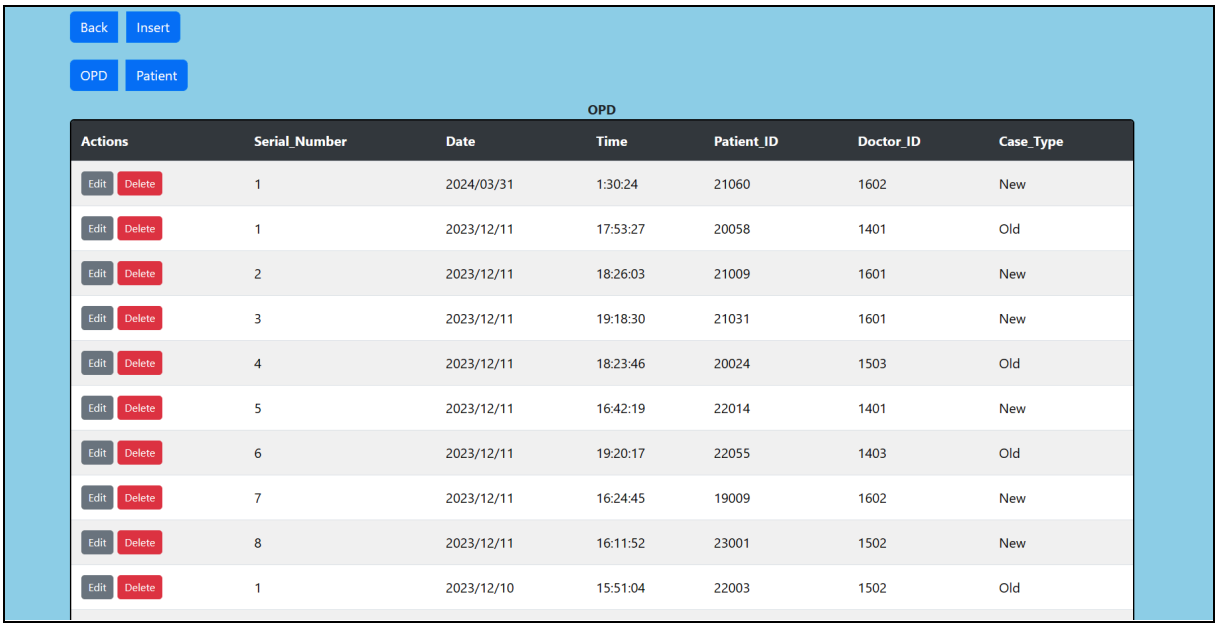
3) Patient: Individuals seeking medical treatment within the healthcare facility. Patients access services, view their medical records, and participate in their treatment plans.

**1) Staff**


When a staff member logs in, they'll see the page depicted above. They have the freedom to select any of the provided options and perform actions such as viewing, editing, deleting, and inserting data into the corresponding tables within the database. For instance, if they click on the "Role" option, a page will appear with choices like "Doctor," "Staff," "Medical Test," "Hospital," and "Supplier." By selecting any of these tables, they can then view the data contained within it, as illustrated in the image below.

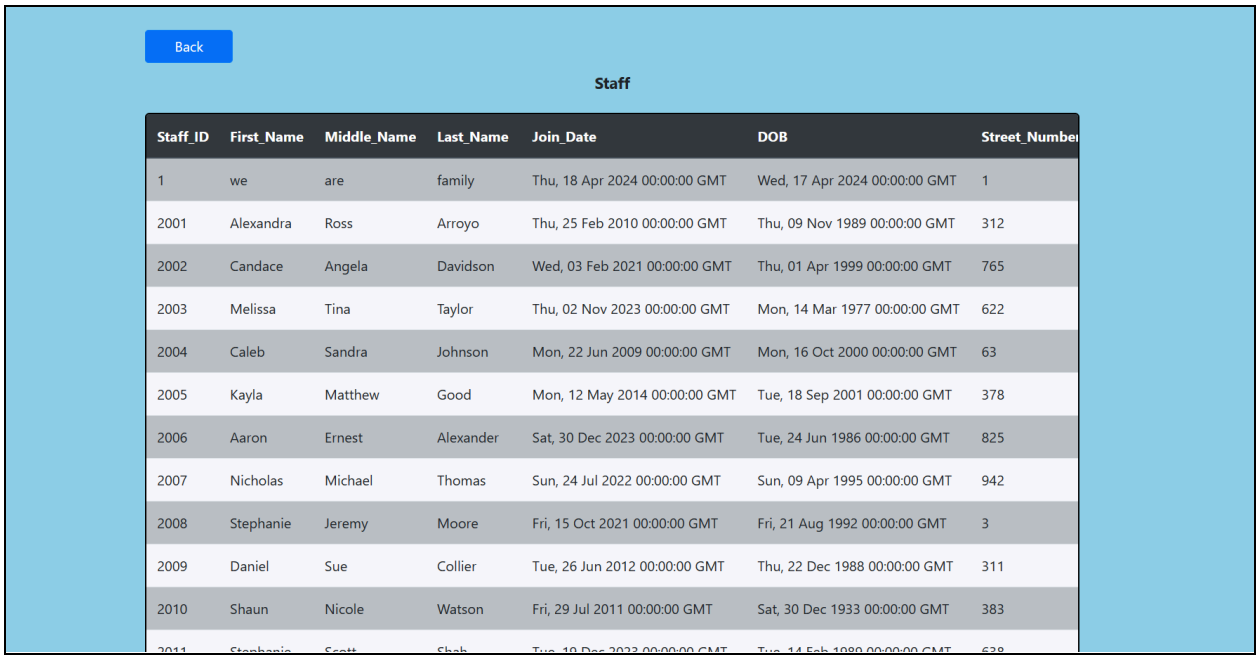

Likewise, if the user selects the "OPD" option, they will have access to view and perform various operations on the OPD table. Additionally, they can also choose to perform operations on the patient table. Below is an example interface for the OPD table, where options for editing and deleting entries are visible.




**Privilege** : staff members are granted full access privileges across all tables present in the database. This includes the ability to view, edit, and delete entries within any table.

## 2) Doctor

If a user is logged in as a doctor, they will have access to all the tables just like staff members. However, doctors will be restricted from editing or deleting entries in certain tables. For example, as shown in the image below, a doctor can edit and delete entries from the doctor table, but such options are not available for the staff table. This restriction ensures that doctors can manage their own data effectively while maintaining the integrity and security of other tables within the database.




**Privilege:** Doctors can view all table data in the databases through different pages. They are granted the following access permissions:

- Edit and Delete Access:
  - Doctor table
  - Prescription table
  - Emergency table

- Edit-Only Access:
  - Medical Report table

### 3) Patient

If the User is logged in as a patient then the website view for the patient is as per shown in the change number 4.

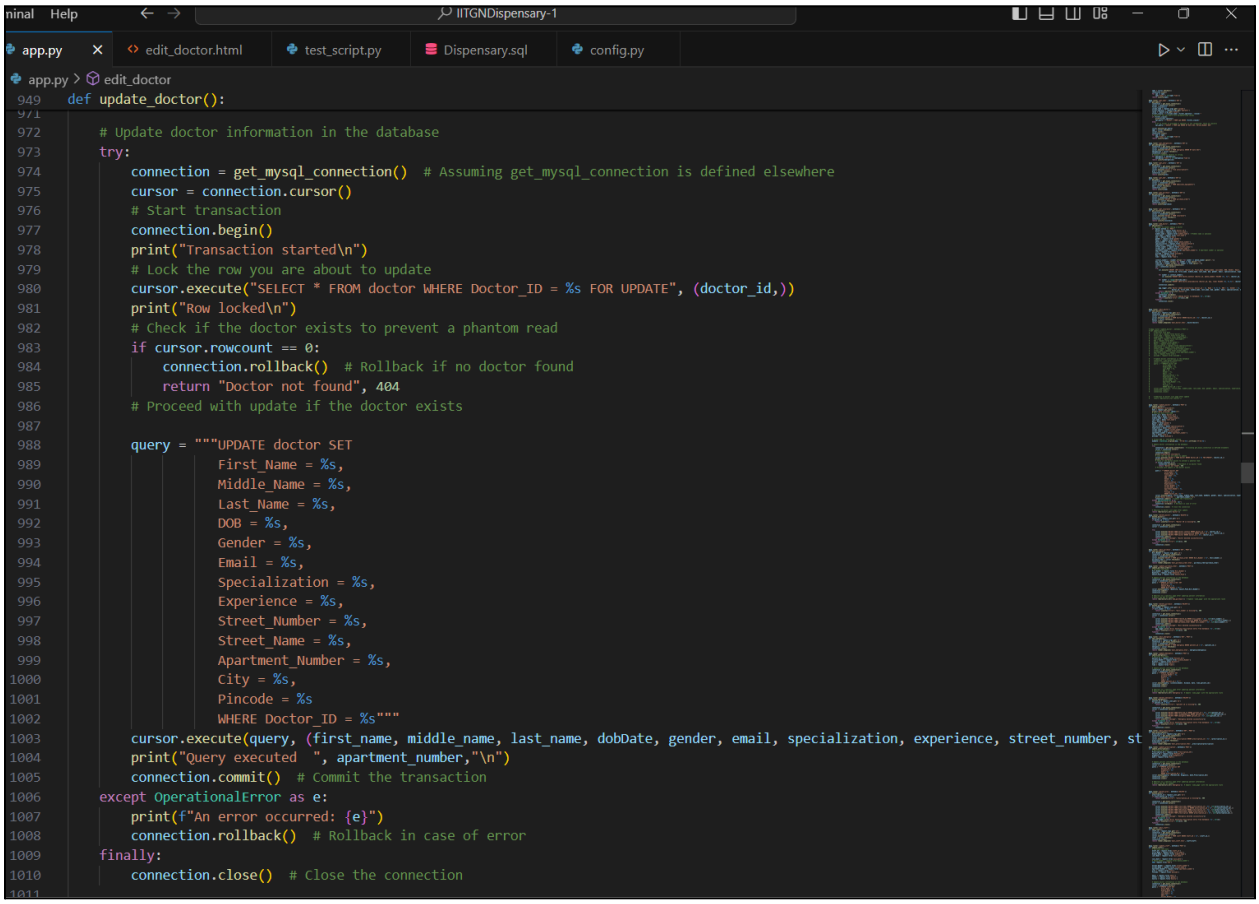
**Privilege:** patients will have restricted access to the following tables: OPD, Patient, Emergency, and Insurance. Specifically, patients will only be able to view their own entries in these tables and will not have access to view entries belonging to other patients. Additionally, patients will not have the ability to edit or delete any entries in these tables.

## Responsibility of G2

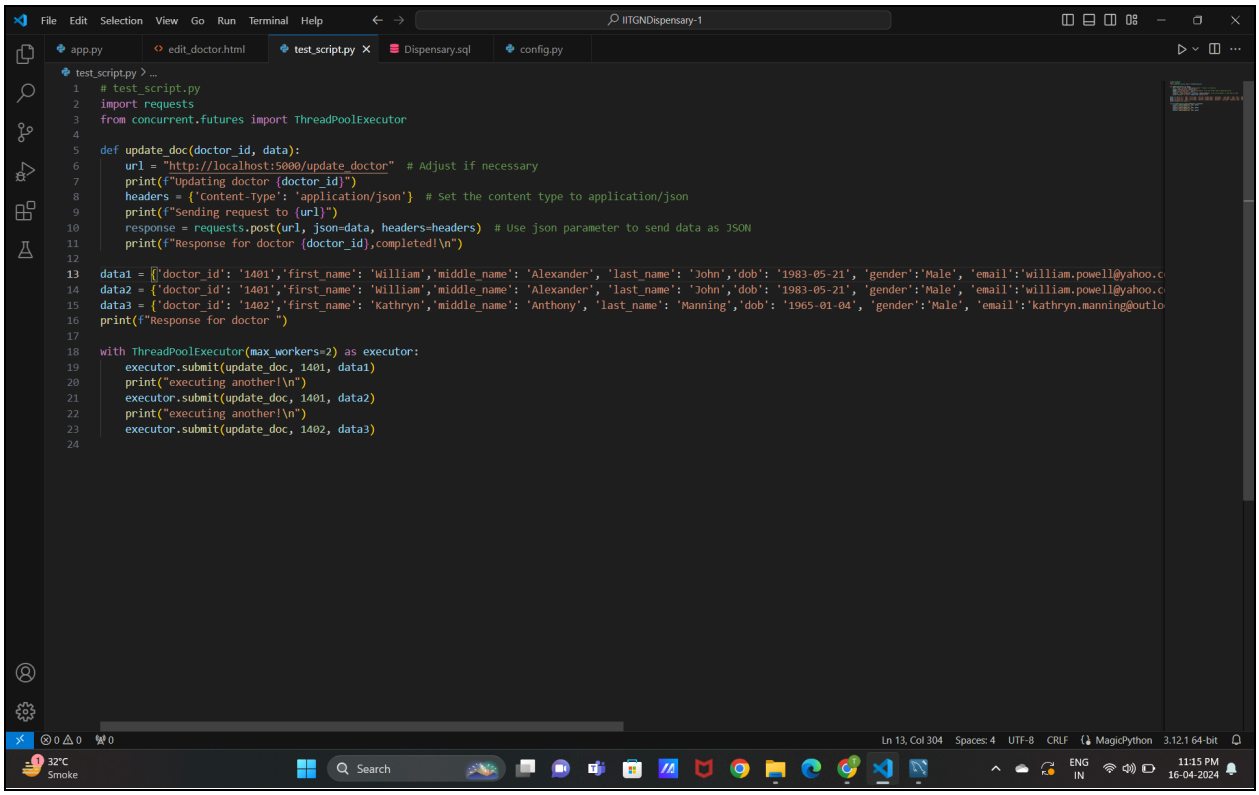
### Q1. Implementing Concurrent multi-user access:

Added Row-Level locks in the “update_doctor” function to control access to individual rows of a table by different transactions. If multiple users try to edit a single doctor entry at the same time, sequential edit requests will be executed. Randomly, one request is chosen, the specific doctor row gets locked till the first request completely gets executed, then next requests in queue get executed. It prevents the data inconsistencies that can occur when multiple transactions attempt to modify the same data simultaneously.

- When a transaction selects a row for updating, the database system places a lock on that specific row.




- Used test_script.py to implement concurrent transactions. Here, we are editing a specific row of “Doctor” with multiple edit requests concurrently, here as Data1 and Data2 trying to edit Doctor entry with Doctor_ID = “1401”.
- Data 1 = Apartment Number → 400
- Data 2 = Apartment Number → 401



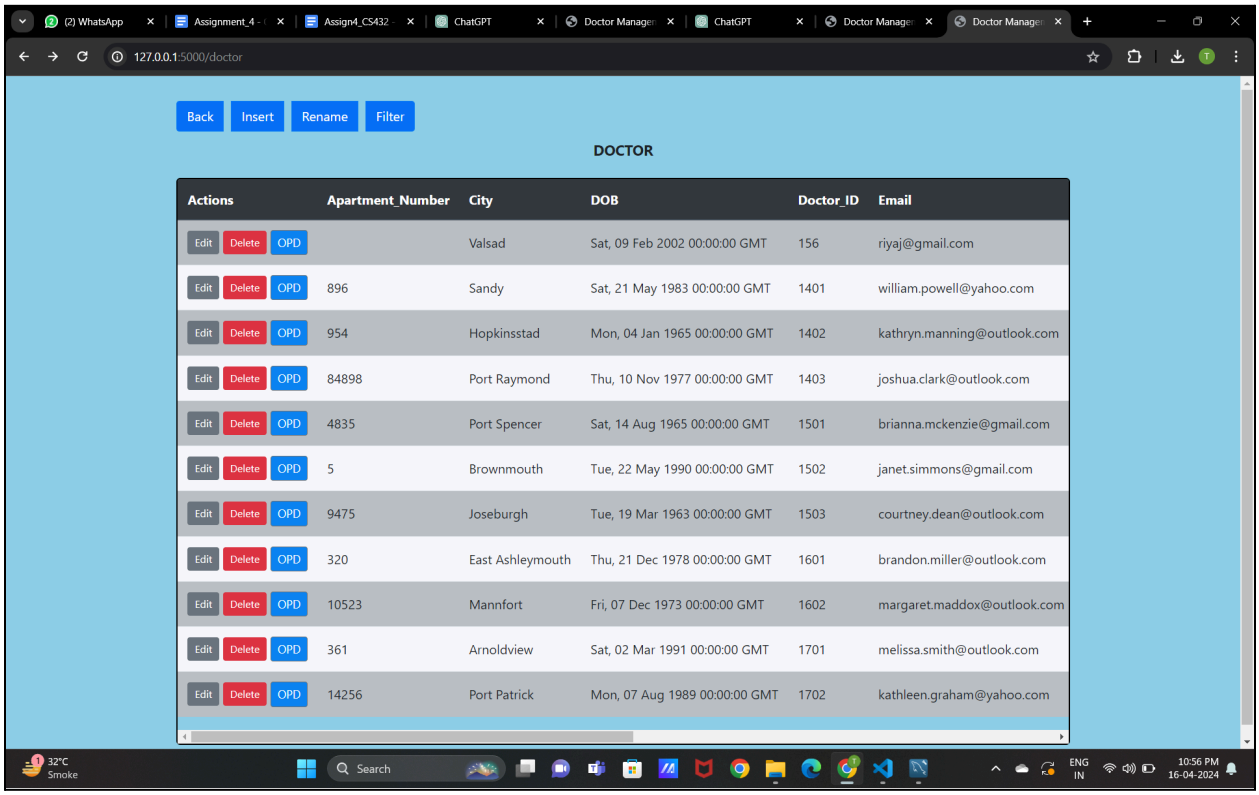
```python
# test_script.py
import requests
from concurrent.futures import ThreadPoolExecutor

def update_doc(doctor_id, data):
    url = "http://localhost:5000/update_doctor"  # Adjust if necessary
    print(f"Updating doctor {doctor_id}")
    headers = {'Content-Type': 'application/json'}  # Set the content type to application/json
    print(f"Sending request to {url}")
    response = requests.post(url, json=data, headers=headers)  # Use json parameter to send data as JSON
    print(f"Response for doctor {doctor_id},completed!\n")

data1 = {'doctor_id': '1401','first_name': 'William','middle_name': 'Alexander', 'last_name': 'John','dob': '1983-05-21', 'gender':'Male', 'email':'william.powell@yahoo.c
data2 = {'doctor_id': '1401','first_name': 'William','middle_name': 'Alexander', 'last_name': 'John','dob': '1983-05-21', 'gender':'Male', 'email':'william.powell@yahoo.c
data3 = {'doctor_id': '1402','first_name': 'Kathryn','middle_name': 'Anthony', 'last_name': 'Manning','dob': '1965-01-04', 'gender':'Male', 'email':'kathryn.manning@outlo
print(f"Response for doctor ")

with ThreadPoolExecutor(max_workers=2) as executor:
    executor.submit(update_doc, 1401, data1)
    print("executing another!\n")
    executor.submit(update_doc, 1401, data2)
    print("executing another!\n")
    executor.submit(update_doc, 1402, data3)
```

- Before concurrent editing, for Doctor_ID = 1401 , Apartment Number = 896 & Doctor_ID = 1402, Apartment Number = 954.

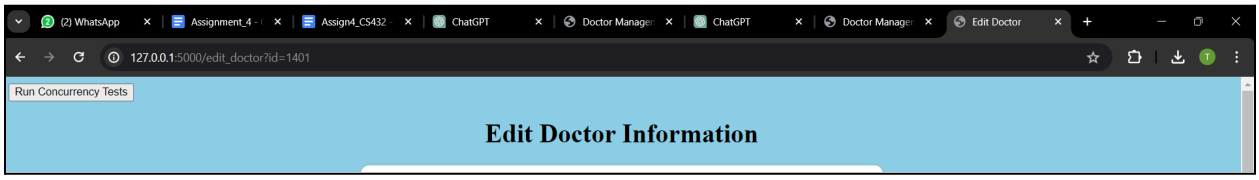


Back Insert Rename Filter

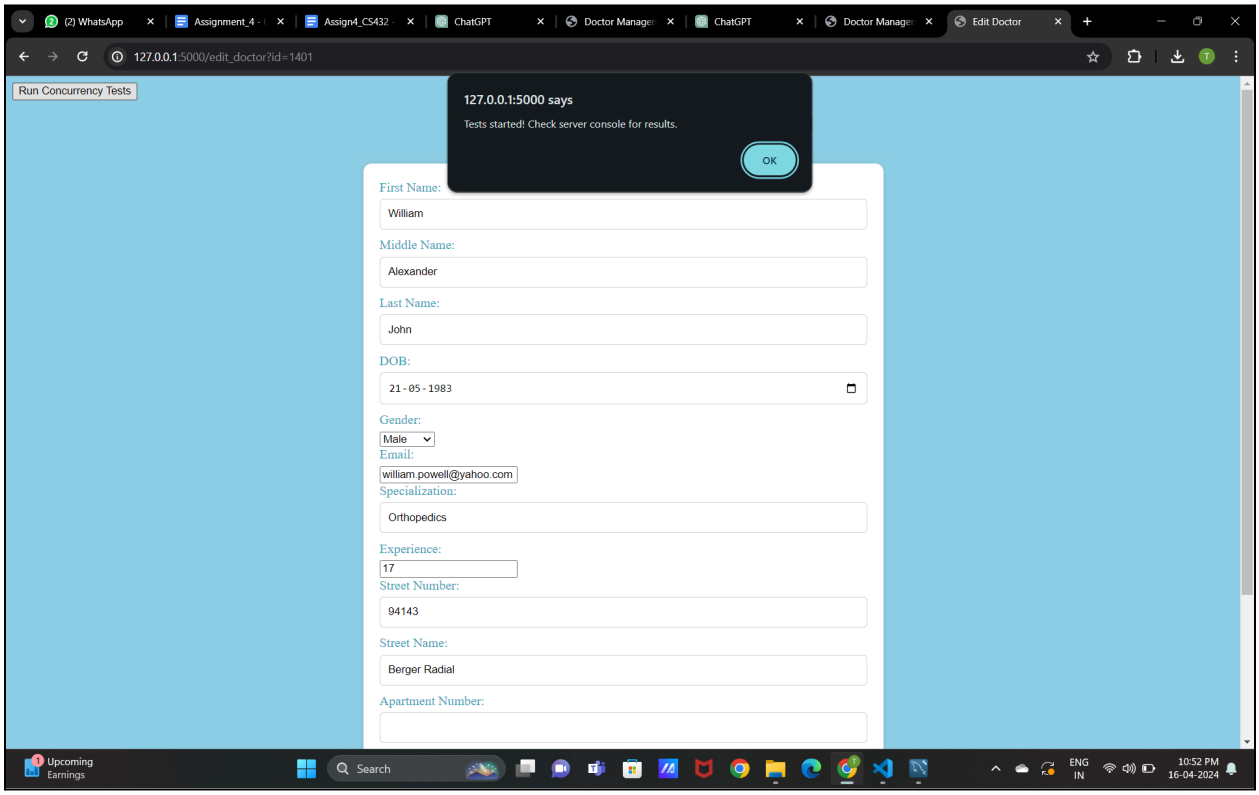
DOCTOR

| Actions | Apartment_Number | City | DOB | Doctor_ID | Email |
|---|---|---|---|---|---|
| Edit Delete OPD | | Valsad | Sat, 09 Feb 2002 00:00:00 GMT | 156 | riyaj@gmail.com |
| Edit Delete OPD | 896 | Sandy | Sat, 21 May 1983 00:00:00 GMT | 1401 | william.powell@yahoo.com |
| Edit Delete OPD | 954 | Hopkinsstad | Mon, 04 Jan 1965 00:00:00 GMT | 1402 | kathryn.manning@outlook.com |
| Edit Delete OPD | 84898 | Port Raymond | Thu, 10 Nov 1977 00:00:00 GMT | 1403 | joshua.clark@outlook.com |
| Edit Delete OPD | 4835 | Port Spencer | Sat, 14 Aug 1965 00:00:00 GMT | 1501 | brianna.mckenzie@gmail.com |
| Edit Delete OPD | 5 | Brownmouth | Tue, 22 May 1990 00:00:00 GMT | 1502 | janet.simmons@gmail.com |
| Edit Delete OPD | 9475 | Joseburgh | Tue, 19 Mar 1963 00:00:00 GMT | 1503 | courtney.dean@outlook.com |
| Edit Delete OPD | 320 | East Ashleymouth | Thu, 21 Dec 1978 00:00:00 GMT | 1601 | brandon.miller@outlook.com |
| Edit Delete OPD | 10523 | Mannfort | Fri, 07 Dec 1973 00:00:00 GMT | 1602 | margaret.maddox@outlook.com |
| Edit Delete OPD | 361 | Arnoldview | Sat, 02 Mar 1991 00:00:00 GMT | 1701 | melissa.smith@outlook.com |
| Edit Delete OPD | 14256 | Port Patrick | Mon, 07 Aug 1989 00:00:00 GMT | 1702 | kathleen.graham@yahoo.com |

- Running “Run Concurrency Tests”

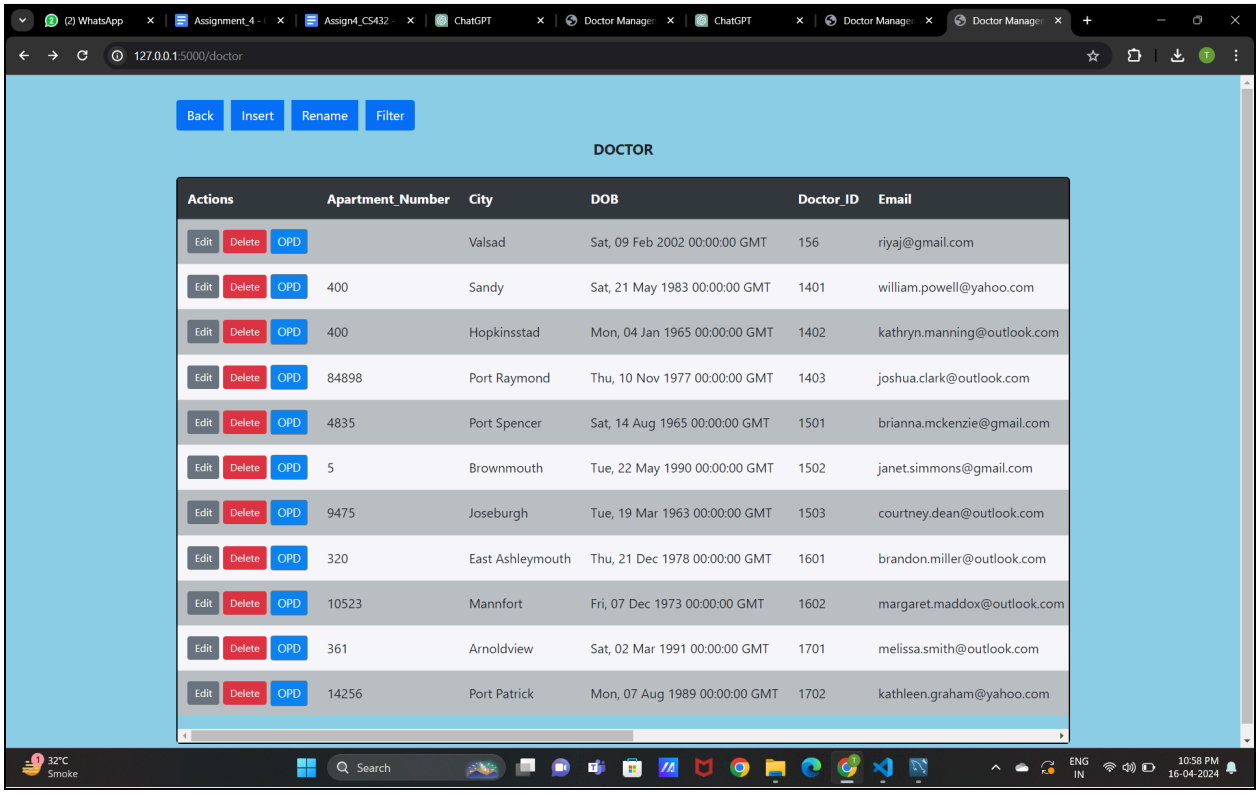



- Test Started Confirmation



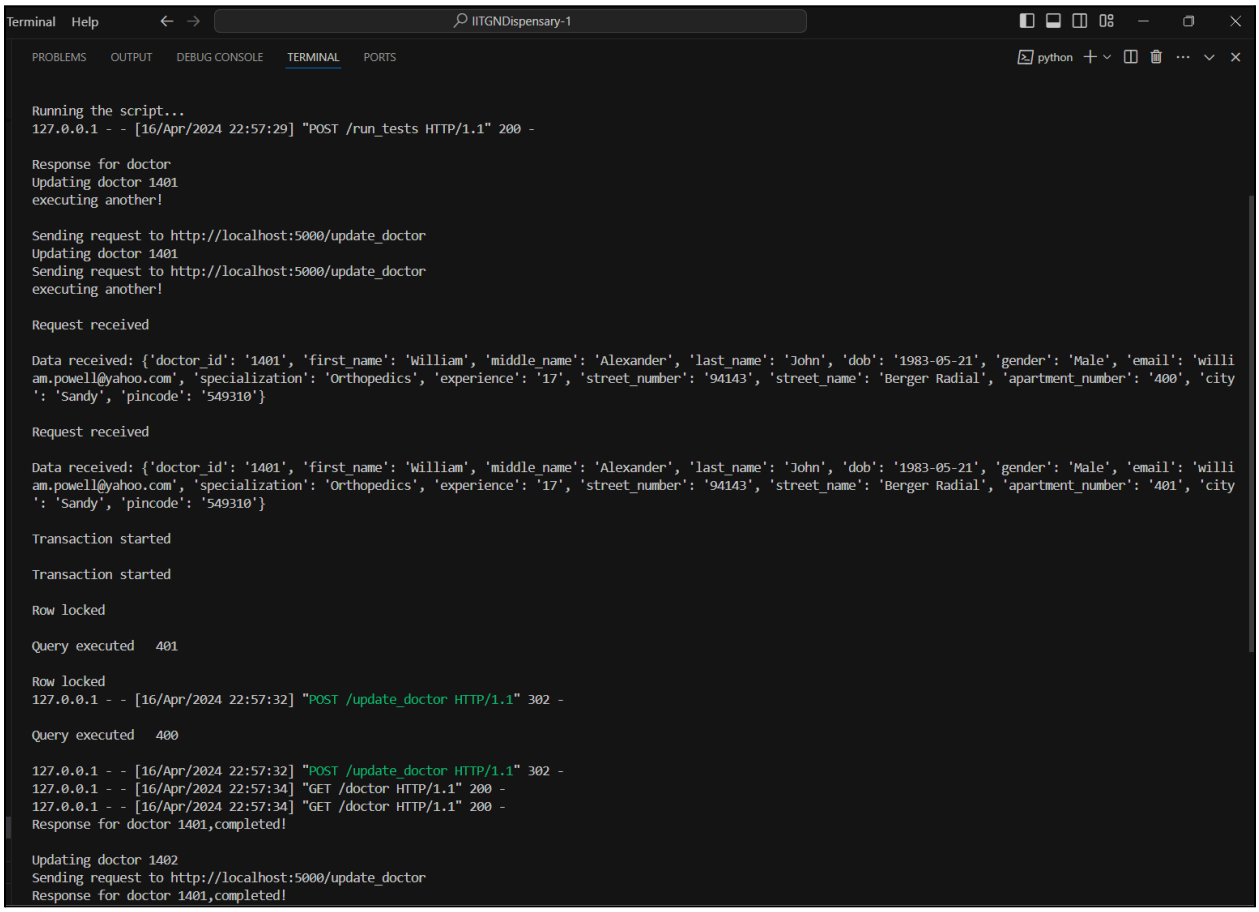

After editing, for Doctor_ID = “1401”, Apartment Number gets updated to “400” and Doctor_ID = “1402”, Apartment Number gets updated to “400”

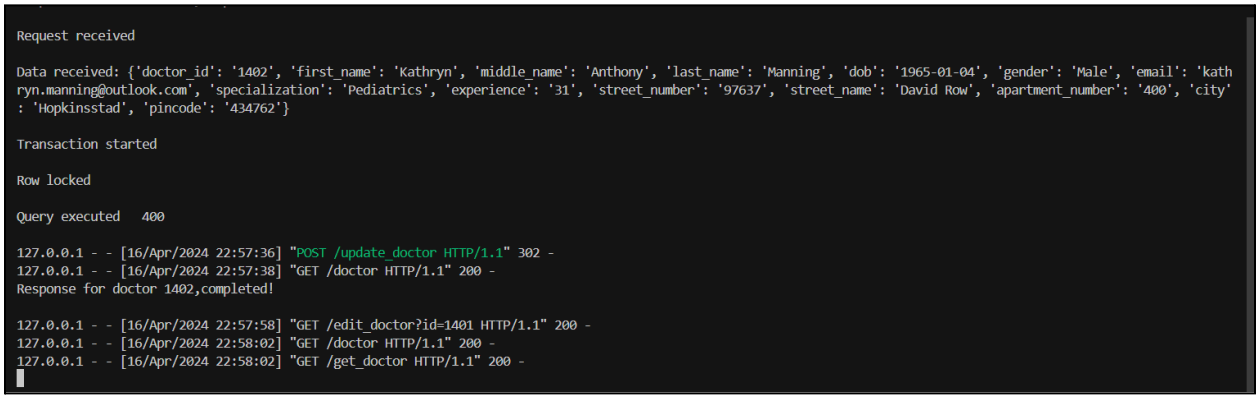
- Updated Doctor Table entries for Doctor_ID = 1401 & 1402



- Output from terminal on executing the concurrent requests.
  For Doctor_ID = 1401, both data1 and data2 requests are being sent concurrently, but are being executed sequentially.

  Here, in the output shown,
  First Query (Data2) which has Apartment Number = 401 gets executed, while being executed, the row gets locked so that further queries can't be executed in the midway.
  After transaction is successful, another query for Data 1 gets executed and that remains final.

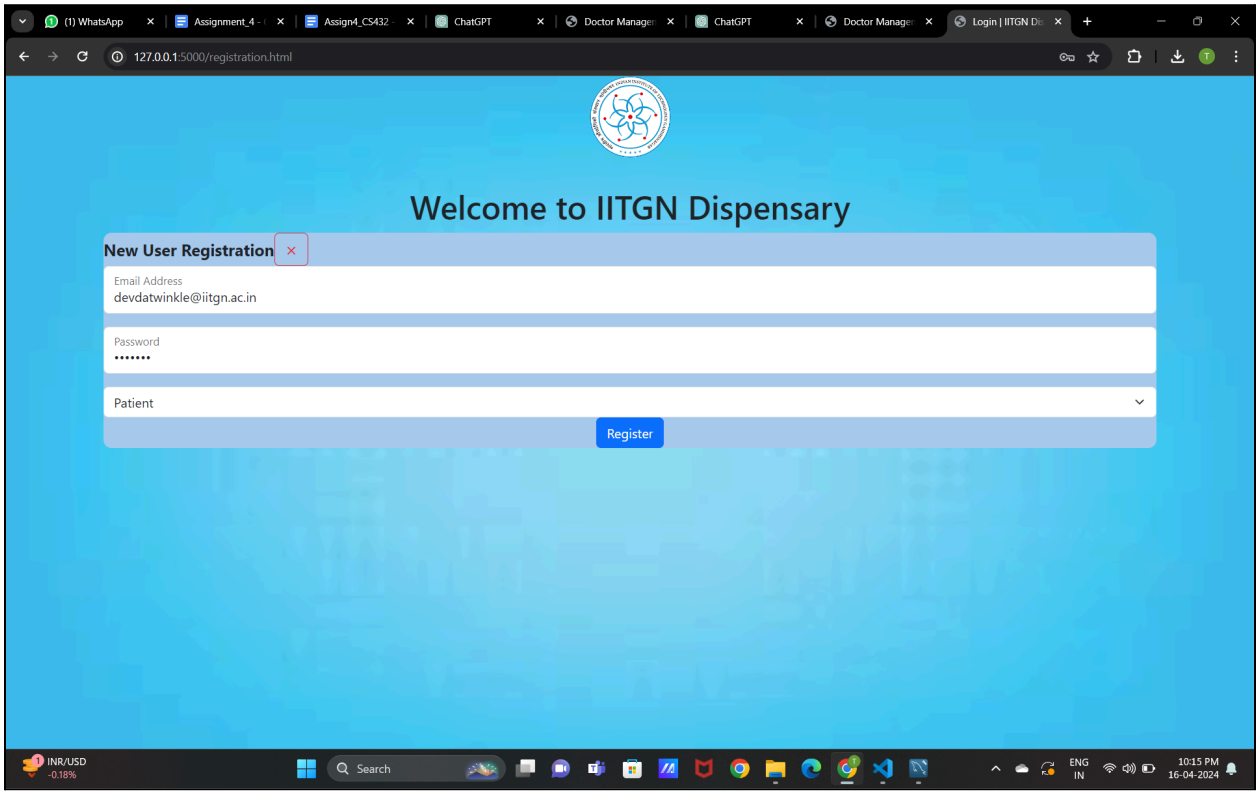


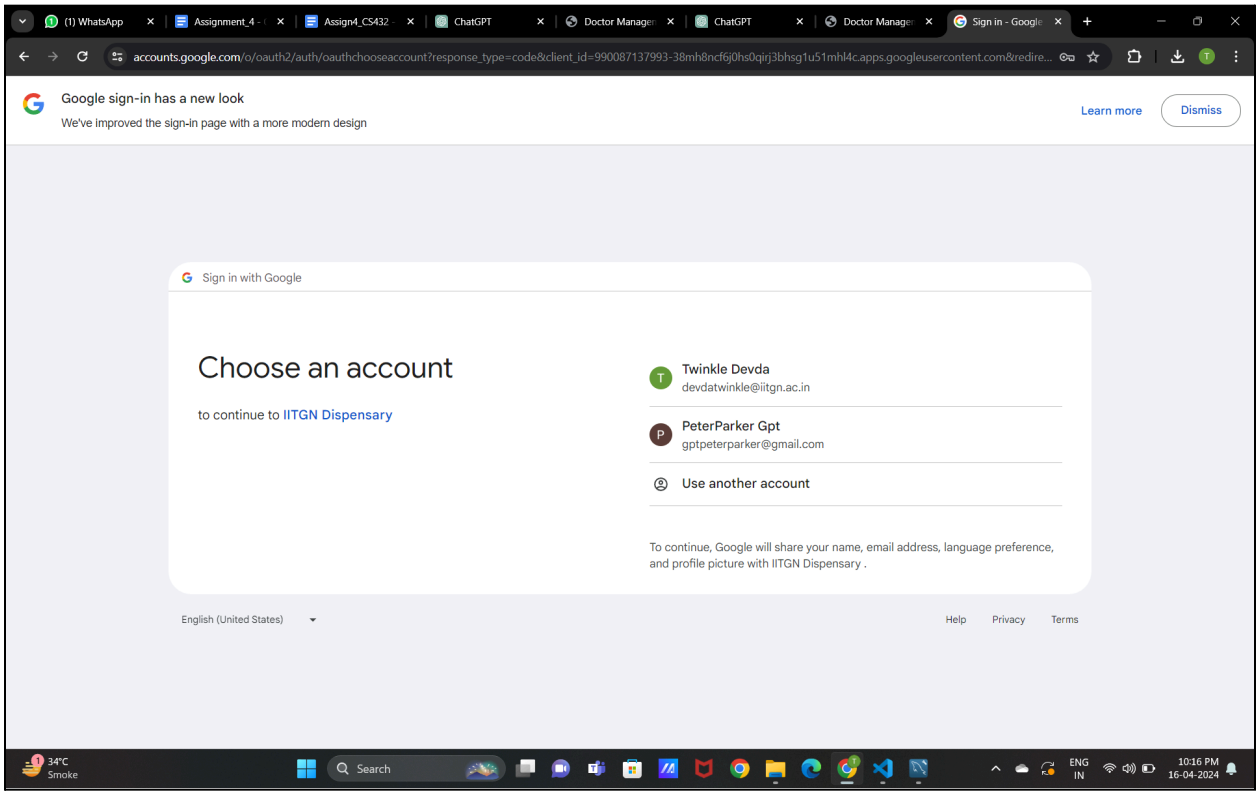
**Q2. Already done above (all the changes after the feedback has been compiled altogether by both the groups)**

**Q3. Google Authentication for Login & Register**
(Only IITGN User can Login and Register in the Webapp)

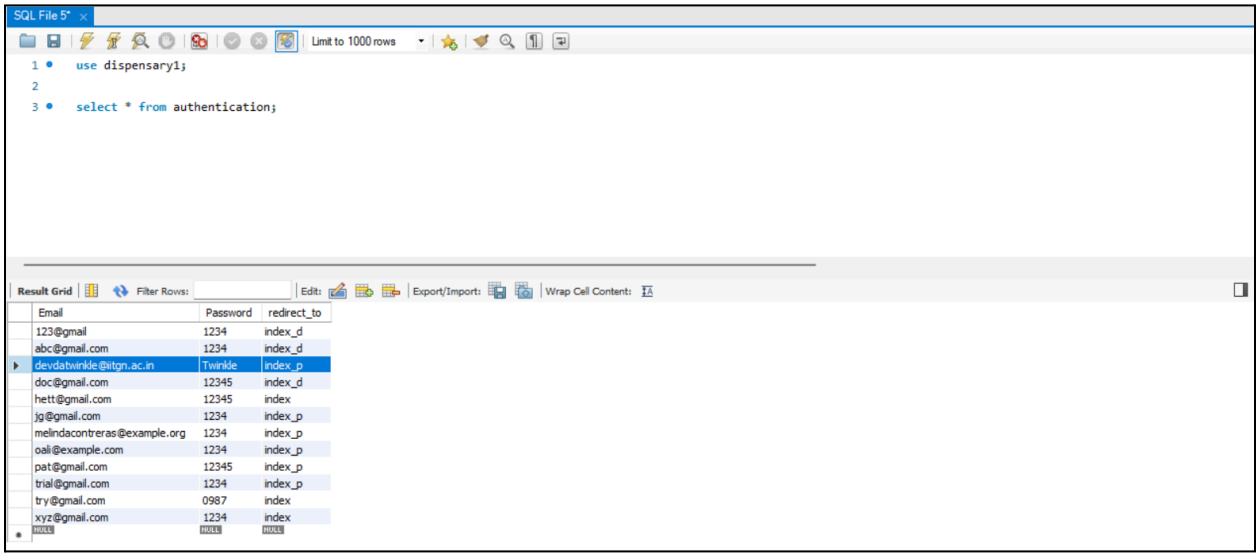
- Creating a new user with email id **"devdatwinkle@iitgn.ac.in" which is a valid IITGN user.**



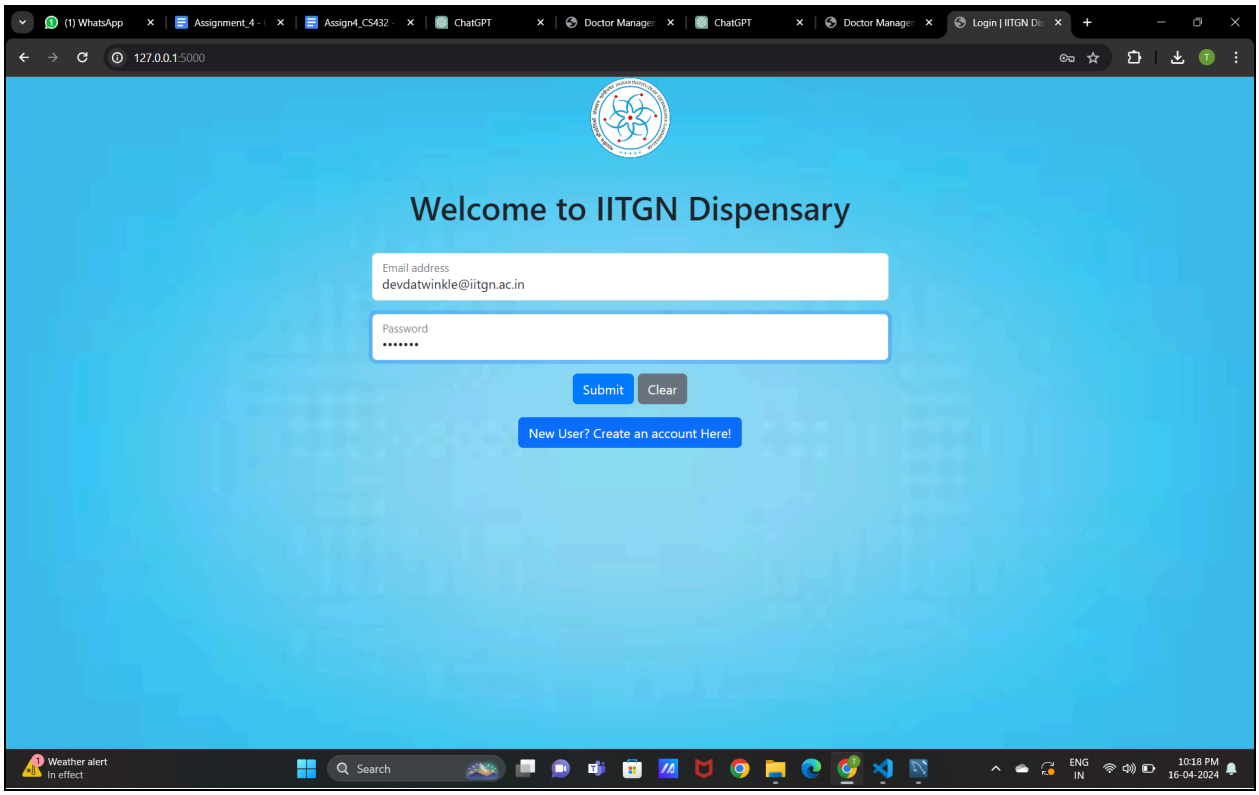
- **Checking if the given email id is Valid Google email Id or not!**




- **New user gets added in Authorisation Table**

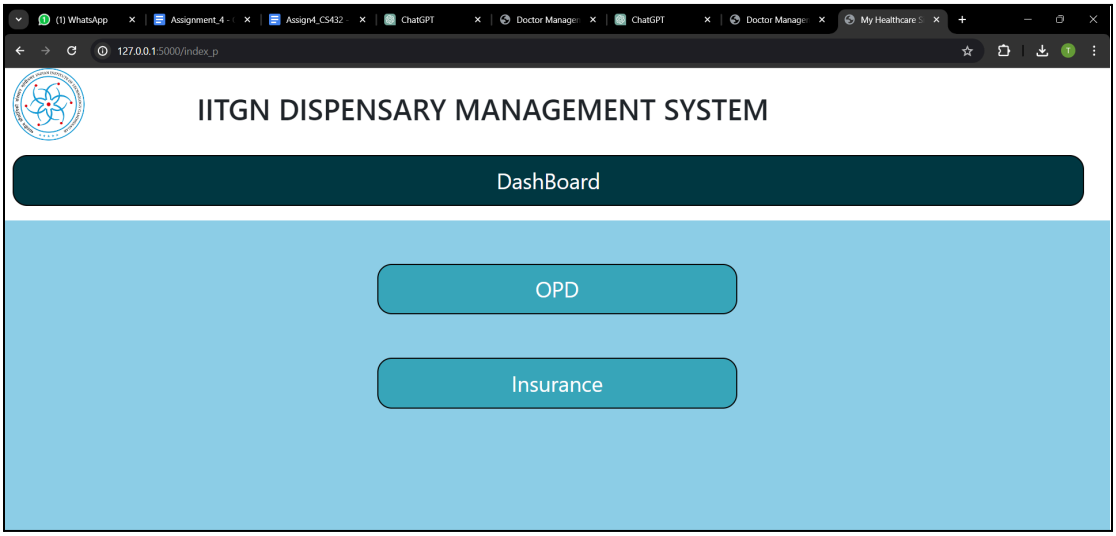



- **Logging In in the newly created account**

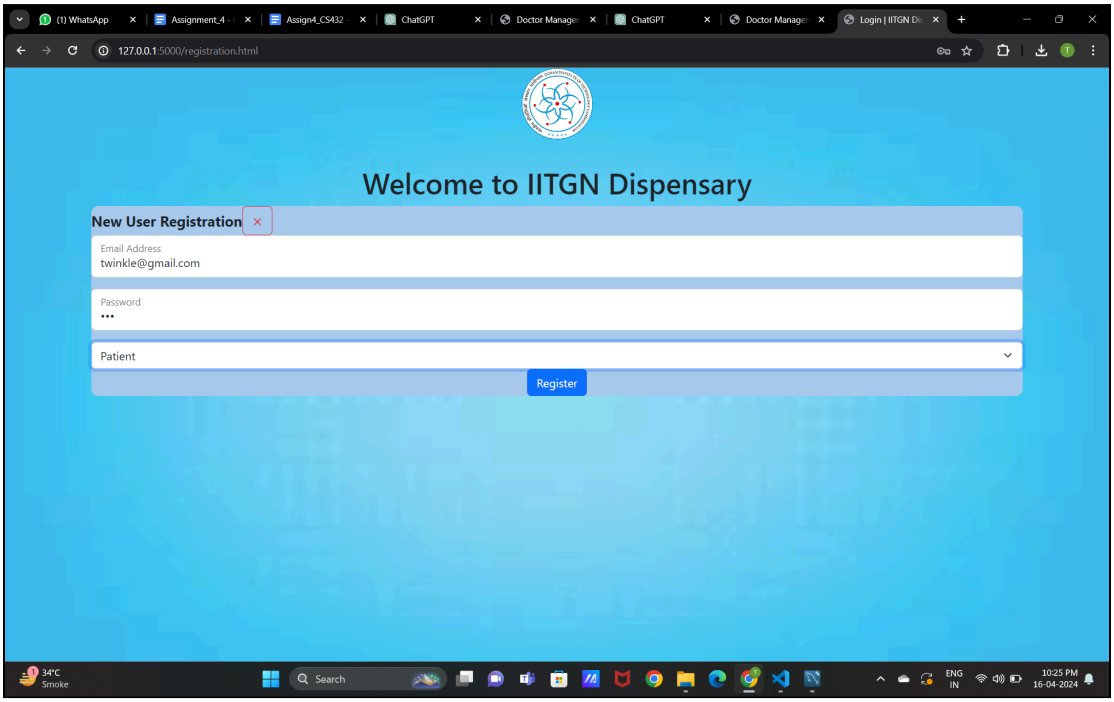



- **Authority = ‘Patient’ as the user is Patient**

- **Patient has acces to patient dashboard**

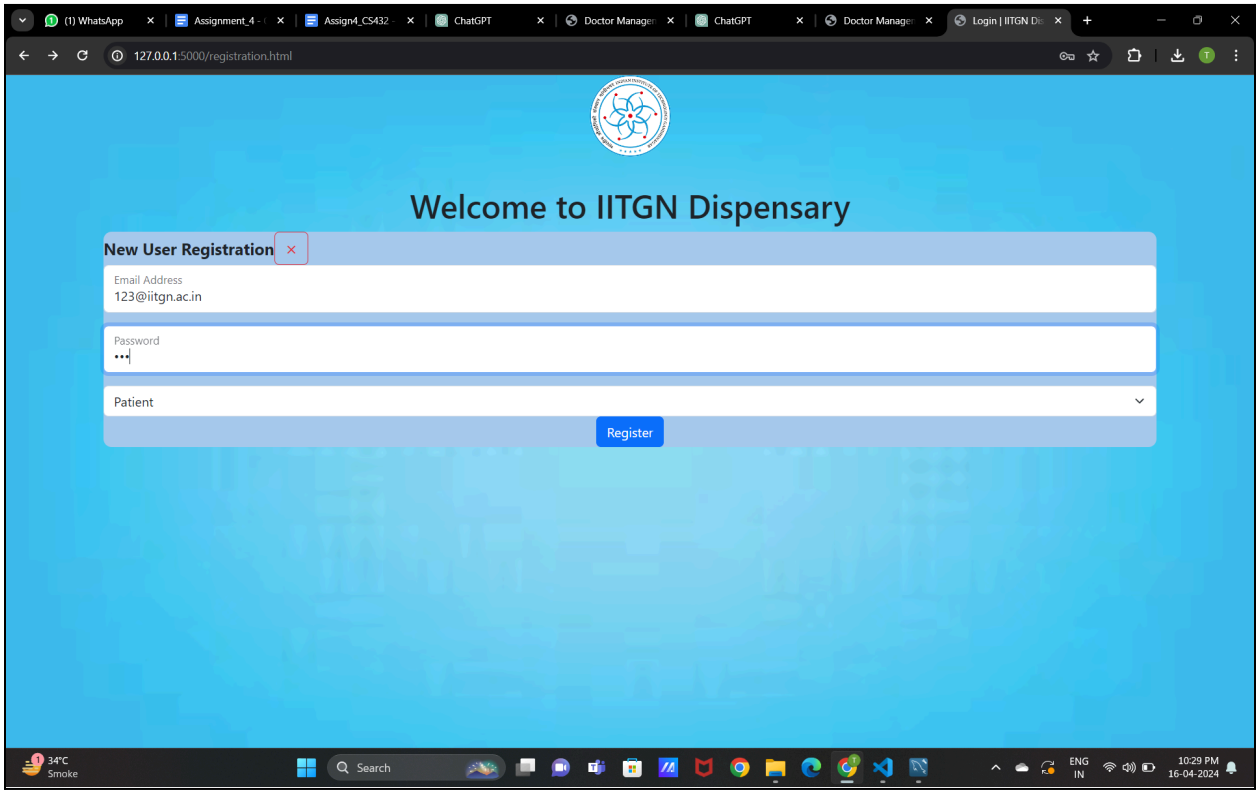



- **Trying to Register with a false Email-id (twinkle@gmail.com)**

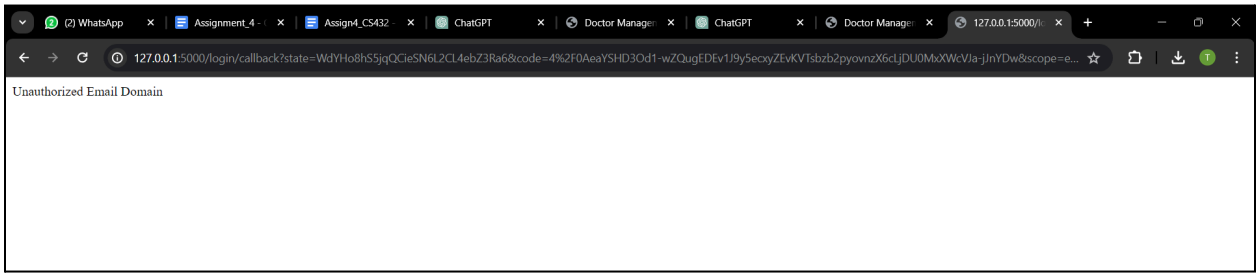



- **Error because this email-id is not supported, only IITGN email id is accepted.**

- **Trying to register with a non existing email id having domain name “iitgn.ac.in”**



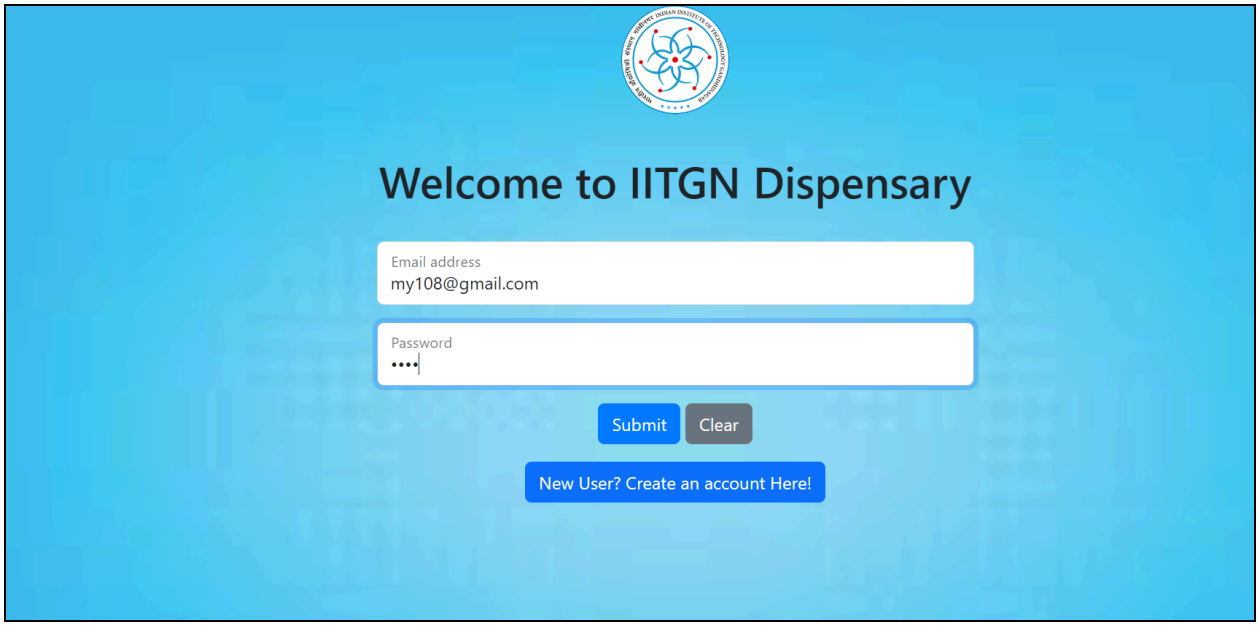

- **Since “123@iitgn.ac.in” is a non valid email-id, Google authorisation fails and we get error.**

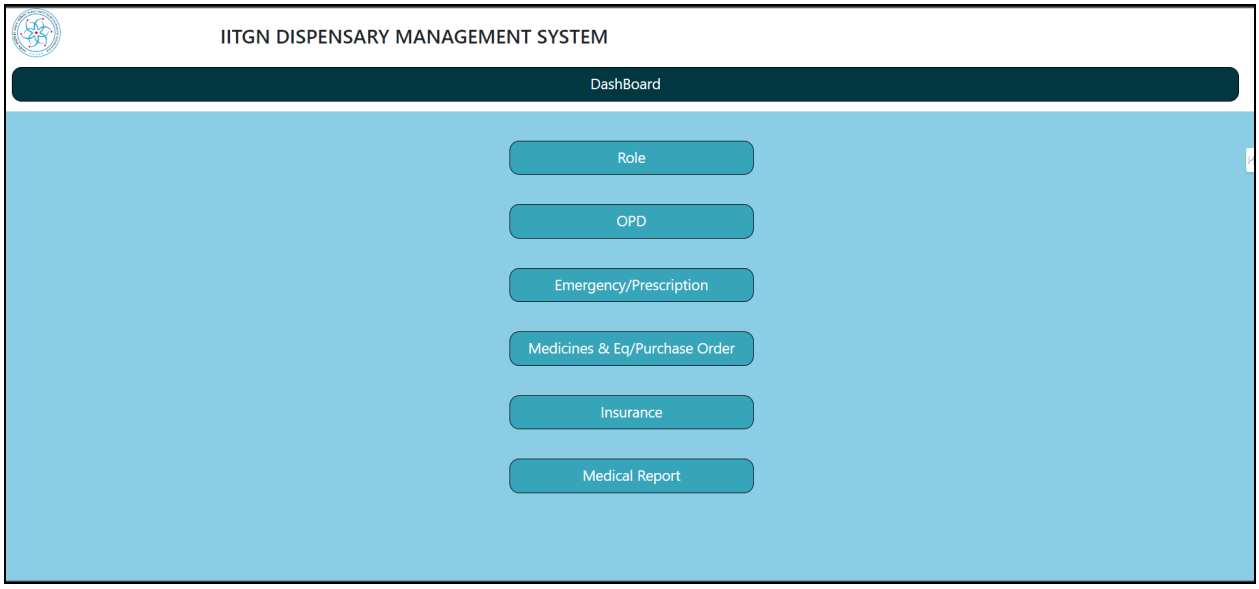


## Responsibility of G1 & G2

### Q1.

- **SQL Injection:**
  - An attacker can use a technique known as SQL Injection to insert malicious SQL code into a web application's query parameters or input fields.
  - Generally, SQL Injection vulnerabilities arise when web applications create SQL queries using user-supplied data without conducting adequate validation or sanitisation.

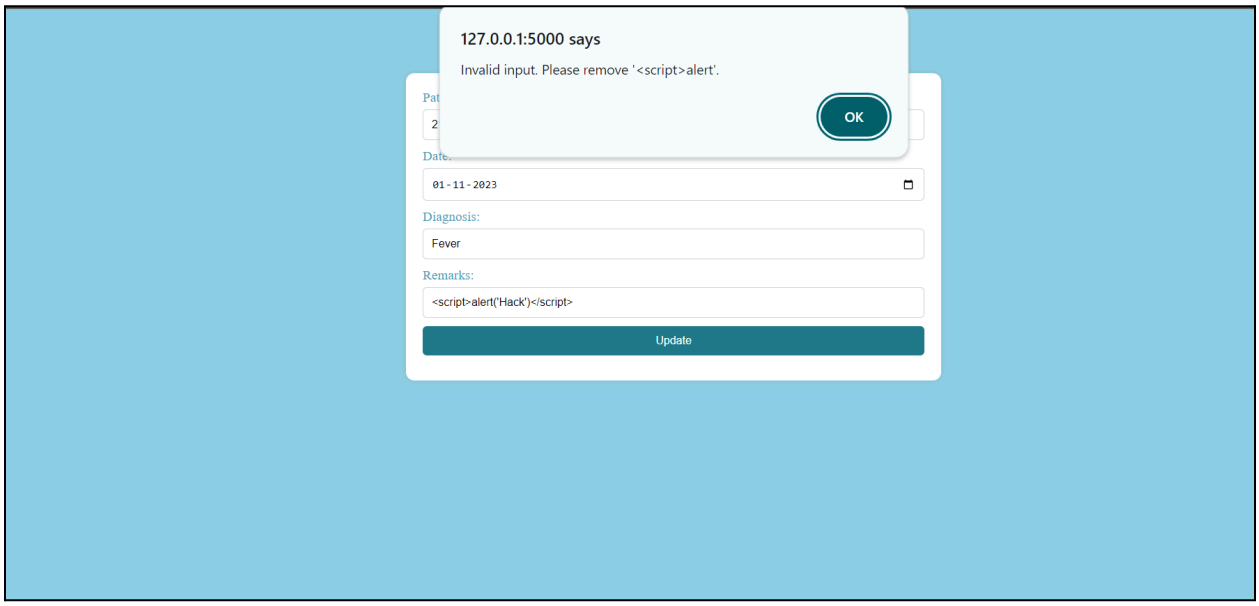






- **XSS:**
  - Cross-site scripting (XSS) enables attackers to insert malicious scripts into web pages that other users are viewing.
  - In the context of other users' browsers, the injected scripts have the ability to run arbitrary code, possibly stealing cookies, session tokens, or other private data, or acting as the user.
  - Three types of XSS vulnerabilities can be distinguished: DOM-based XSS (where the vulnerability is present in client-side JavaScript code), stored XSS (where the injected script is stored on the server and executed when viewed by other users), and reflected XSS (where the injected script is reflected off a web server).

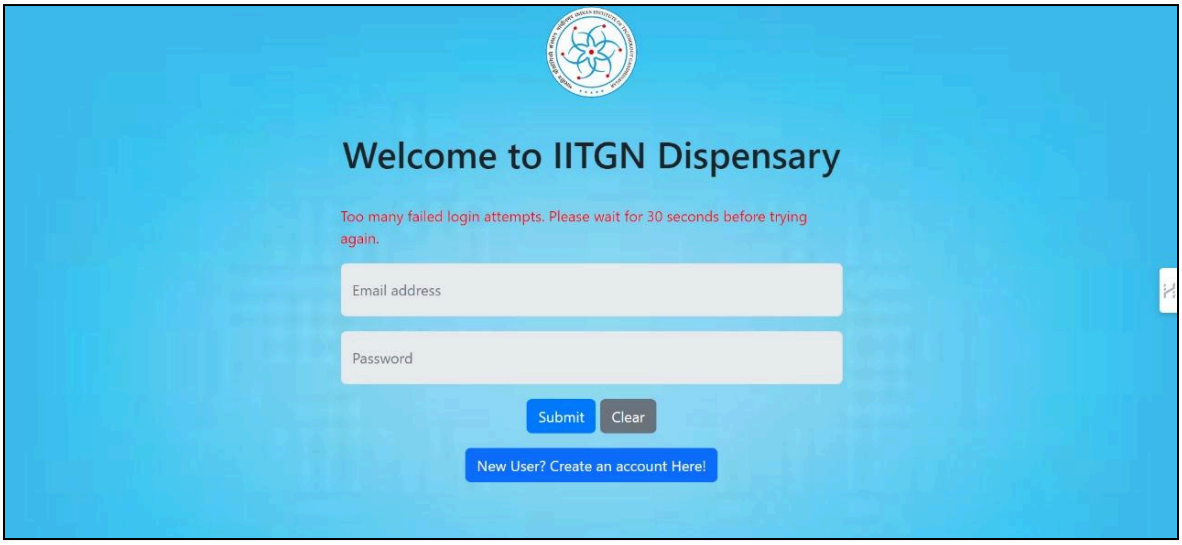

  - Without a defense, entering will cause a dialog box to appear and may cause the application to crash.

- We have included a dialog box that displays "invalid input. Please remove <script>alert" as a defense, preventing the user from inserting any dangerous content.



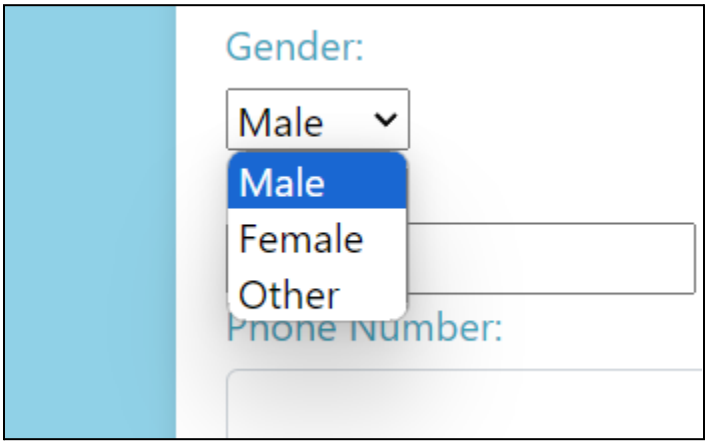

# BONUS ATTACK - BY G1 + G2

- **Brute Force:**
  - A Brute Force Attack is a kind of cybersecurity attack in which the attacker deliberately tries every possible combination of characters, numbers, and symbols until they find the one that works, in an attempt to guess the target's encryption keys, login credentials, or other sensitive information.
  - We have prevented this attack by limiting the user for 30 seconds after 3 login attempts and displaying the message " Too many login attempts. Please wait for 30 seconds before trying again."

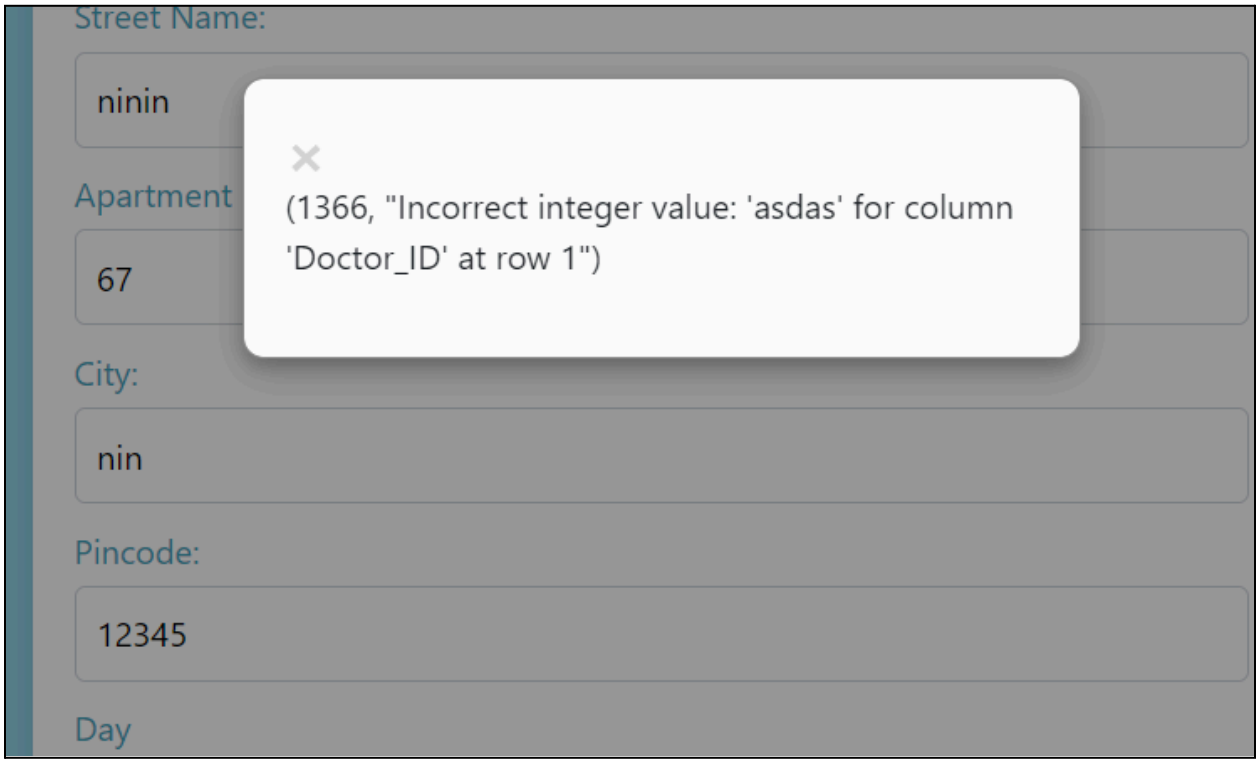


**Q2.**

Example 1) For gender, a drop-down box has been added so that values out of only these 3 can be added.

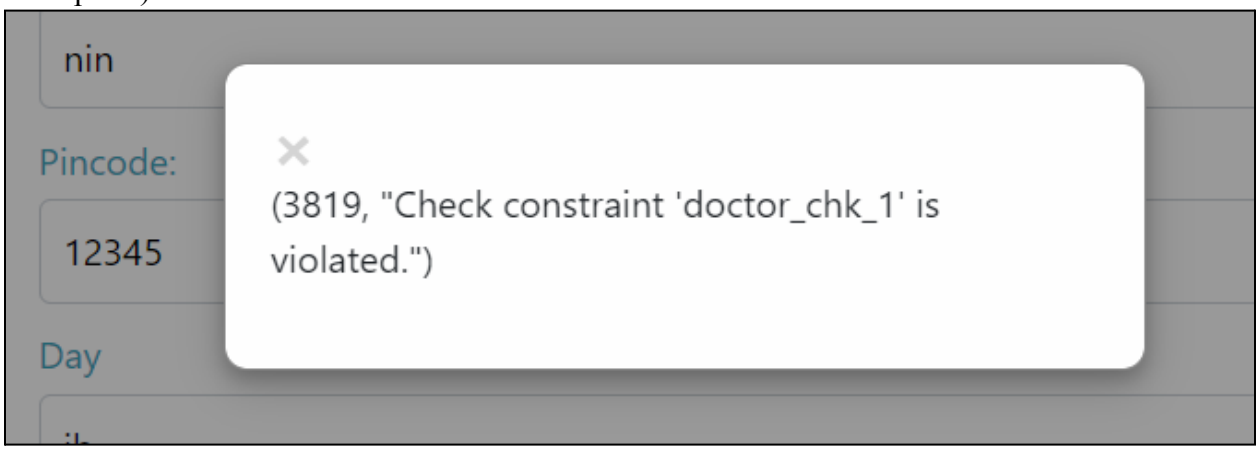



Example-2)



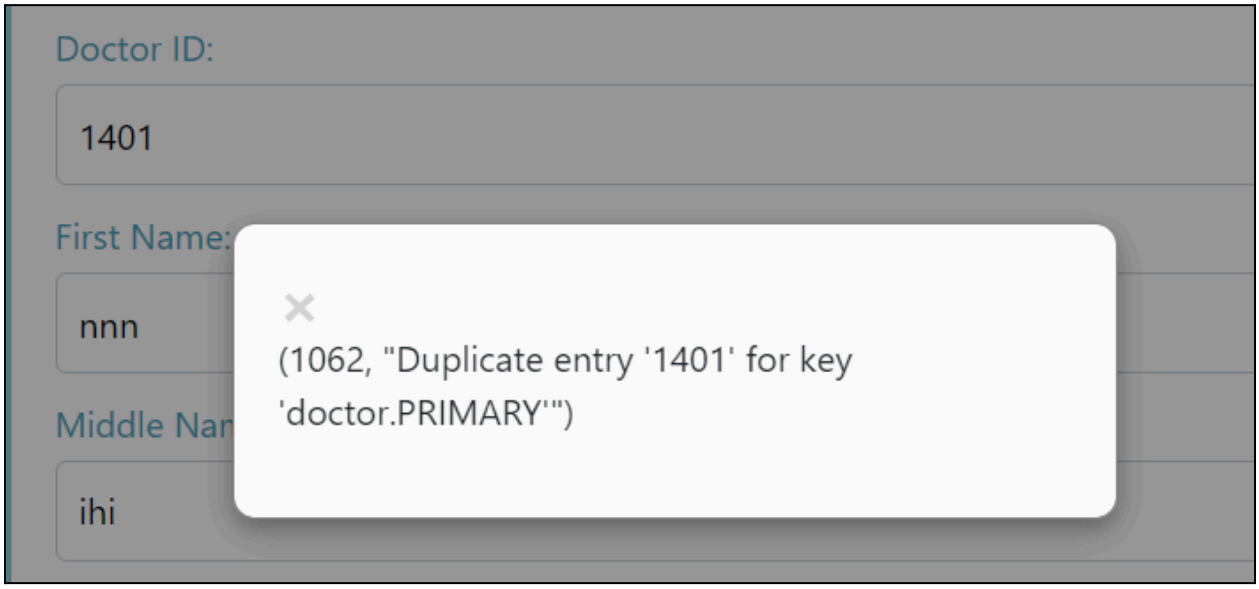

The primary key(Doctor_ID), which should be an integer gave an error upon adding a non-integer value.

Example-3)



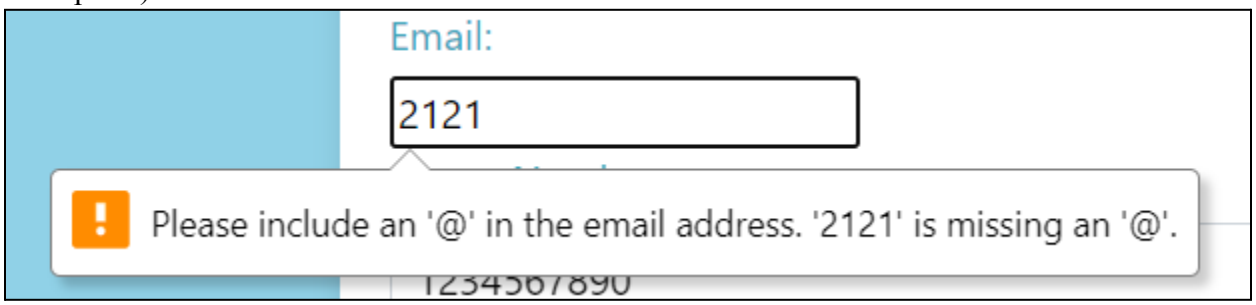

For a pin code not of  6 digits, the constraint error appeared.

Example-4 )



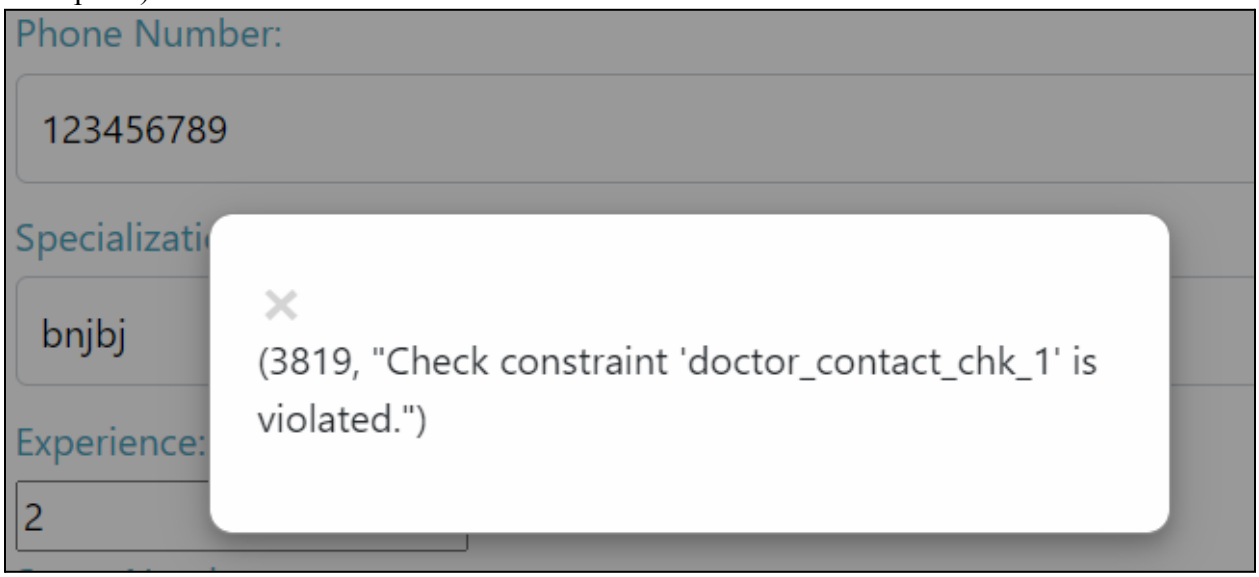

As we can see if a duplicate entry for the primary key (doctor_id) is added then error message would appear.

Example-5)




As we can see if @ is not included in the email, an error appears.

Example-6)




For a mobile number not of  10 digits, the constraint error appeared.

## Contribution:

| Name | Group | Contribution |
|---|---|---|
| Ishika Raj | G2 | Multiple roles access, update databases without overlapping edits.Secure login and authorisation by Google, restricted to IITGN users,Documenting additional security attacks, Document SQL Injection, XSS attacks, and defenses, Documentation |
| Twinkle Devda | G2 | Multiple roles access, update databases without overlapping edits.Secure login and authorisation by Google, restricted to IITGN users, Documenting additional security attacks, Document SQL Injection, XSS attacks, and defenses, Documentation |
| Saumya Jaiswal | G2 | Multiple roles access, update databases without overlapping edits. Secure login and authorisation by Google, restricted to IITGN users,Documenting additional security attacks, Document SQL Injection, XSS attacks, and defenses. |
| Darsh Dalal | G2 | Multiple roles access, update databases without overlapping edits.Secure login and authorisation by Google, restricted to IITGN users,Documenting additional security attacks,Document SQL Injection, XSS attacks, and defenses, ER Diagram Validation |
| Shreya Patel | G1 | Document changes before and after stakeholder feedback,Documenting additional security |

| | | |
|---|---|---|
| | | attacks,Document SQL Injection, XSS attacks, and defenses. Showcase database access levels for different user classes, Documentation |
| Riya Jain | G1 | Document changes before and after stakeholder feedback,Documenting additional security attacks,Document SQL Injection, XSS attacks, and defenses. Showcase database access levels for different user classes, Documentation |
| Trivedi HetKumar | G1 | Document changes before and after stakeholder feedback,Documenting additional security attacks,Document SQL Injection, XSS attacks, and defenses. Showcase database access levels for different user classes,Secure login via Google, restricted to IITGN users, Documentation |
| Saurabh Kumar Sah | G1 | Document changes before and after stakeholder feedback,Documenting additional security attacks,Document SQL Injection, XSS attacks, and defenses. Showcase database access levels for different user classes, Documentation |