Indian Institute of Technology, Gandhinagar

Database for IITGN Maintenance

Authors:
Pulkit Gautam 21110169
Manav Parmar 21110120
Shubh Agarwal 21110205
Gaurav Rawat 21110066
Hrishikesh Birje 21110044
Aman Singh 21110020
Ishva Patel 21110082
Atal Gupta 21110037

Deploying the DBMS

Databases

CS 432

Under the guidance of
Prof. Mayank Singh

April 16, 2024

Responsibility of G1

Question 1

Feedback Cycle 1

Feedback Session with Mr. Dilip

Mr. Tej Gaurang and Mr. Dilip were partners in developing the web page and related database for IIT Gandhinagar, which they maintain. Mr. Dilip has a good idea about the databases and websites in general. He advised us that some data may be deleted as unnecessary, such as 'middle name' in the user table. He proposes making the user experience more convenient and faster by adding an opportunity to auto-fill the user ID into the request form. He had also proposed adding timestamps to the requests and sorting them. In general, he expressed satisfaction with the system's performance.

Feedback Session with Mr. Dharmendra Panchal

Mr. Dharmendra Panchal, who handles the civil aspects of maintenance, frequently utilizes the admin page of the current website. He also recommended enhancing the user-friendliness of the status change feature. He wanted the request form to appear as a single row but was fine without it. He liked our addition of worker information on the webpage and was overall happy with the page.

Changes Implemented

Deletion of middle name

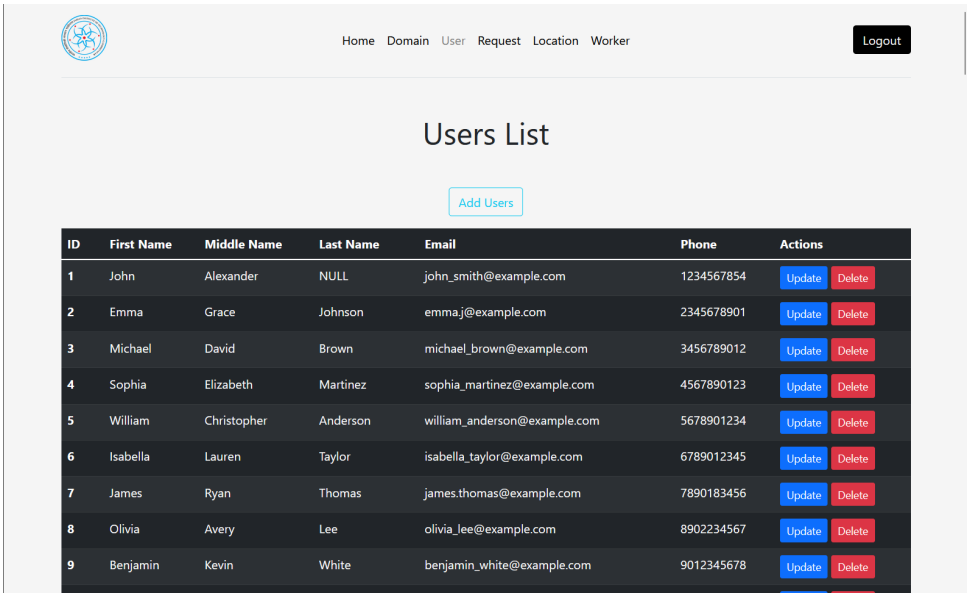


Figure 1: Middle name present in the User Table

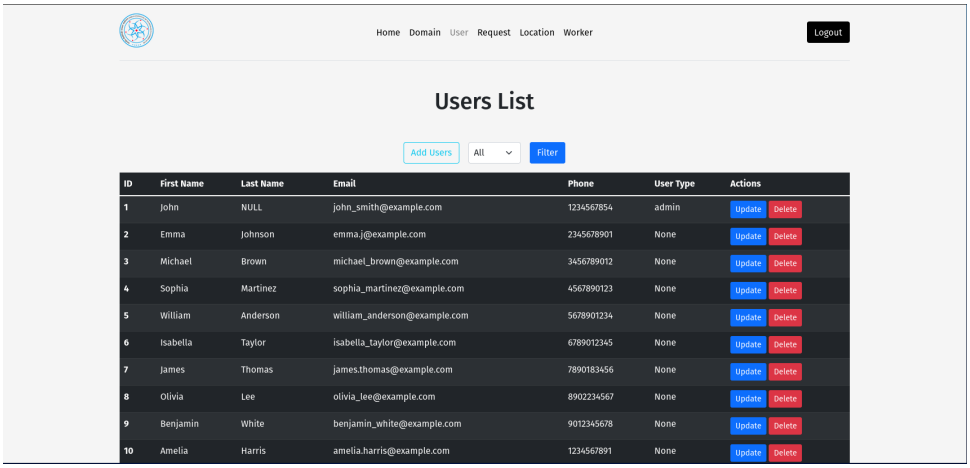


Figure 2: Middle name deleted from User Table

Auto-filling of user ID

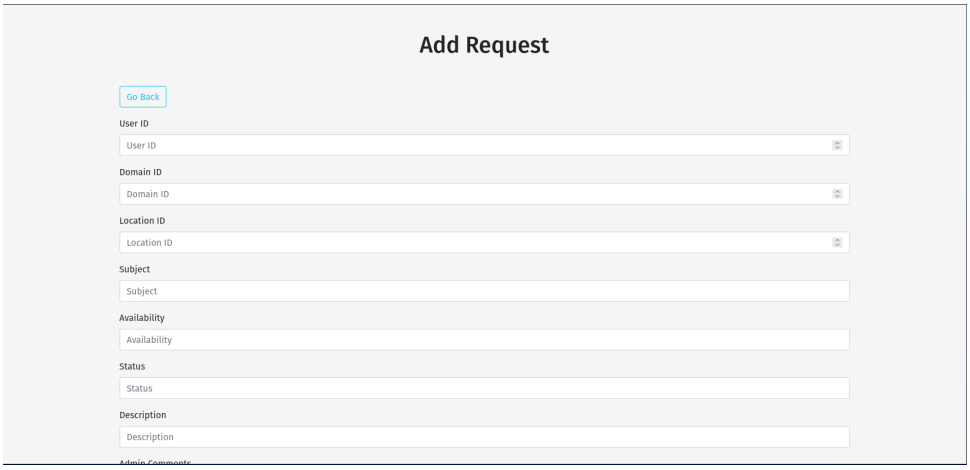


Figure 3: No auto-filling in request form

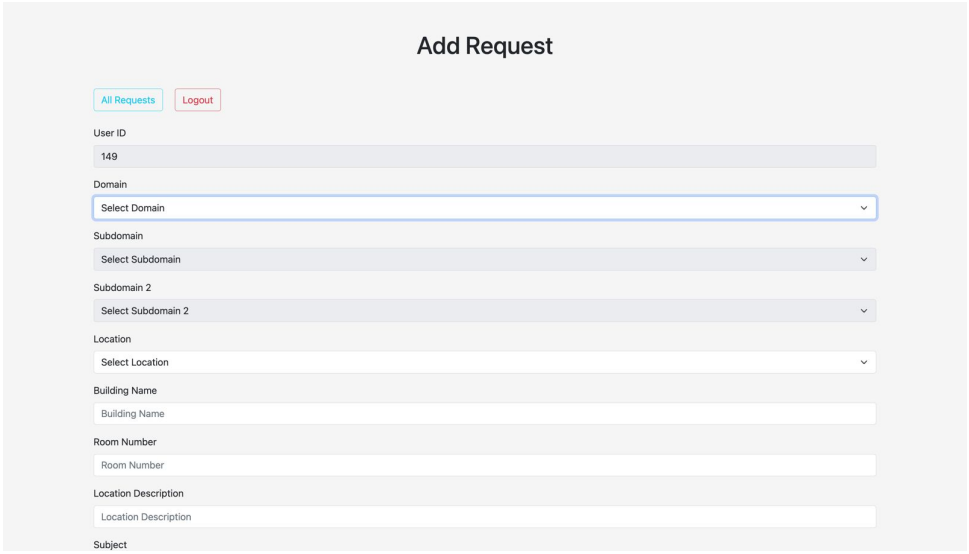


Figure 4: Auto-filling user-id in request form

Addition of time stamps and sorting requests accordingly

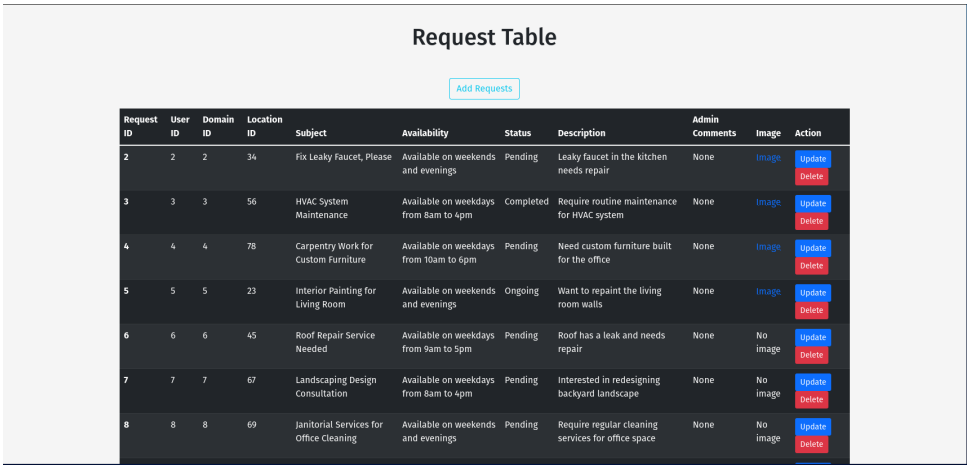


Figure 5: No timestamps in request table before Feedback1

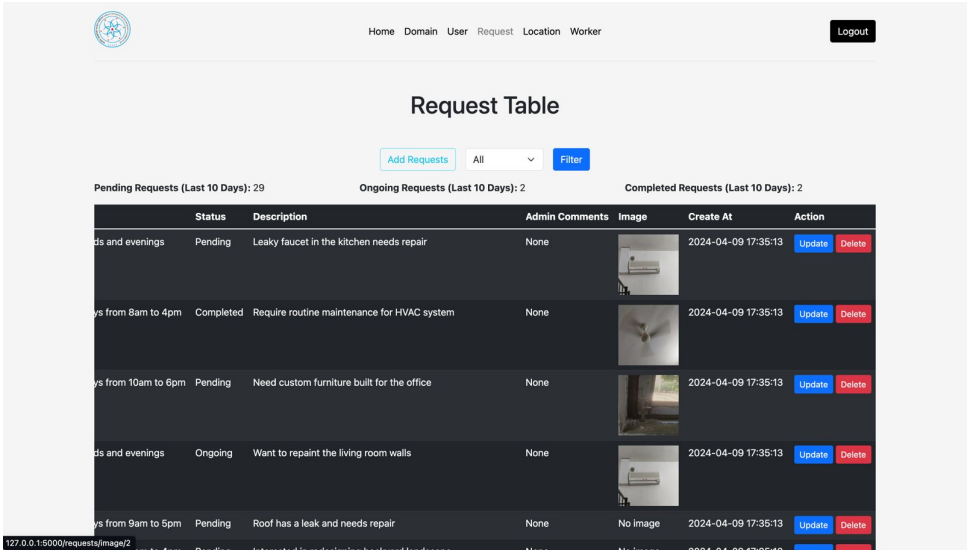


Figure 6: Timestamps added and requests sorted accordingly

Status change feature

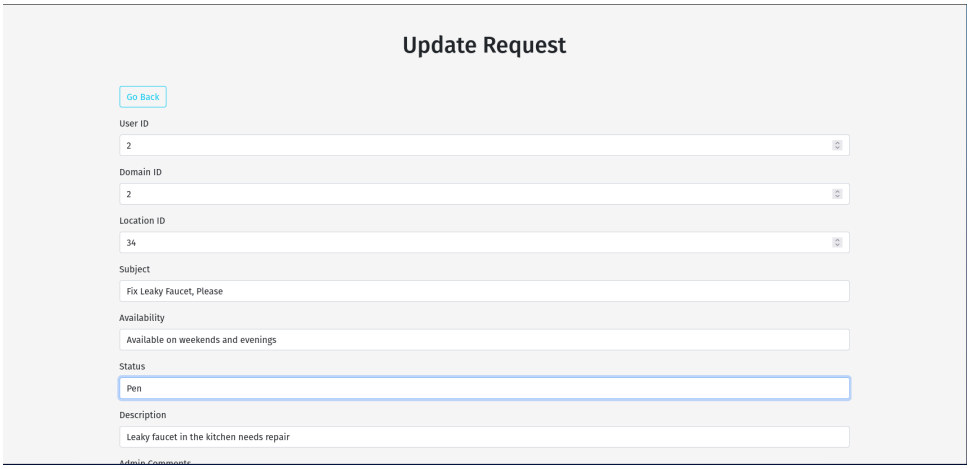


Figure 7: Manual nature of request status change before Feedback1

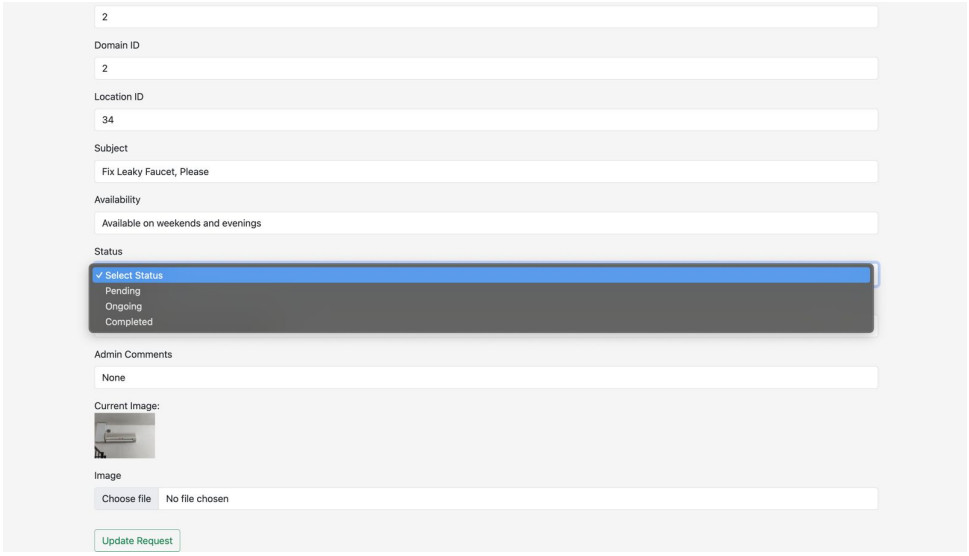


Figure 8: Dropdown added for status change

Feedback Cycle 2

Feedback Session with Mr Tej Gaurang and Mr Dilip

We went to these two individuals because we had interacted with them regarding this project in the past, and we wanted to maintain consistency. He asked us to display some details, such as request pending and some user information on the home admin page. He felt this feature was necessary for someone who uses the webpage daily. He also asked us to handle the overflow in the request table on the front end. He also asked us to change some parts of the UI to make it look better. Overall, he was satisfied with our efforts.

Changes Implemented

Home page of admin

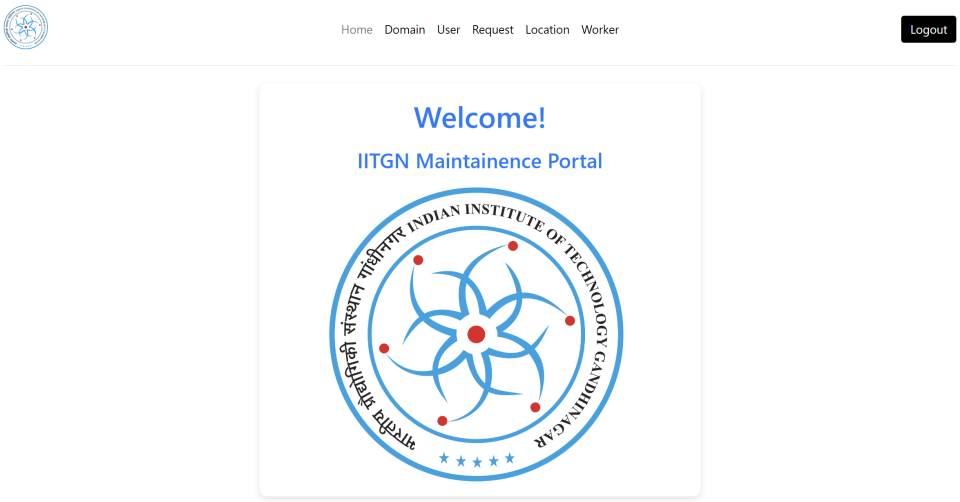


Figure 9: Home page before Feedback

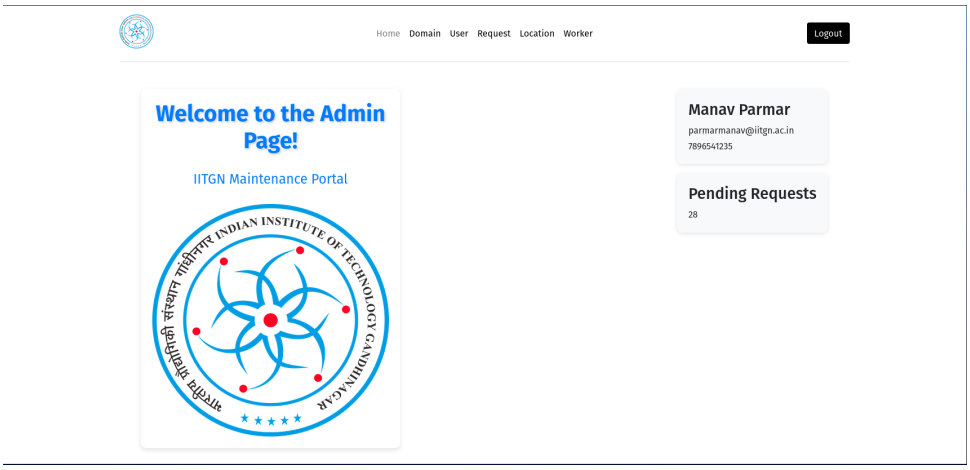


Figure 10: New home page with additional information

Overflow in request table handling

Figure 11: No maximum character cap through frontend

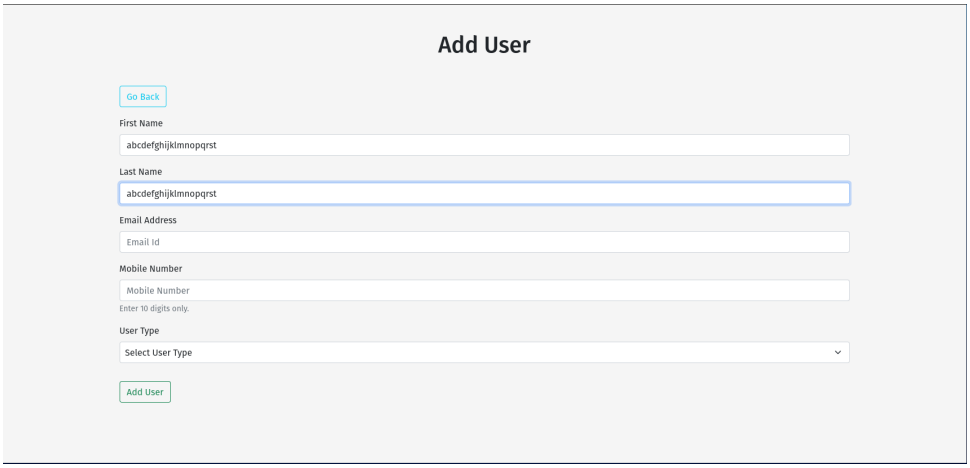


Figure 12: Maximum character cap through frontend

Login page UI updated

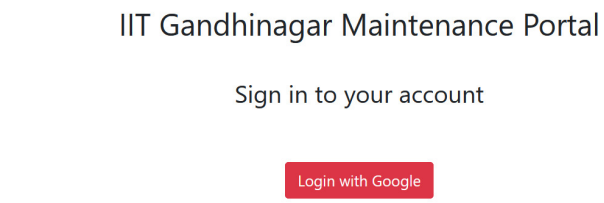


Figure 13: Previous Login page

Figure 14: Current Login Page

Request Table UI updated

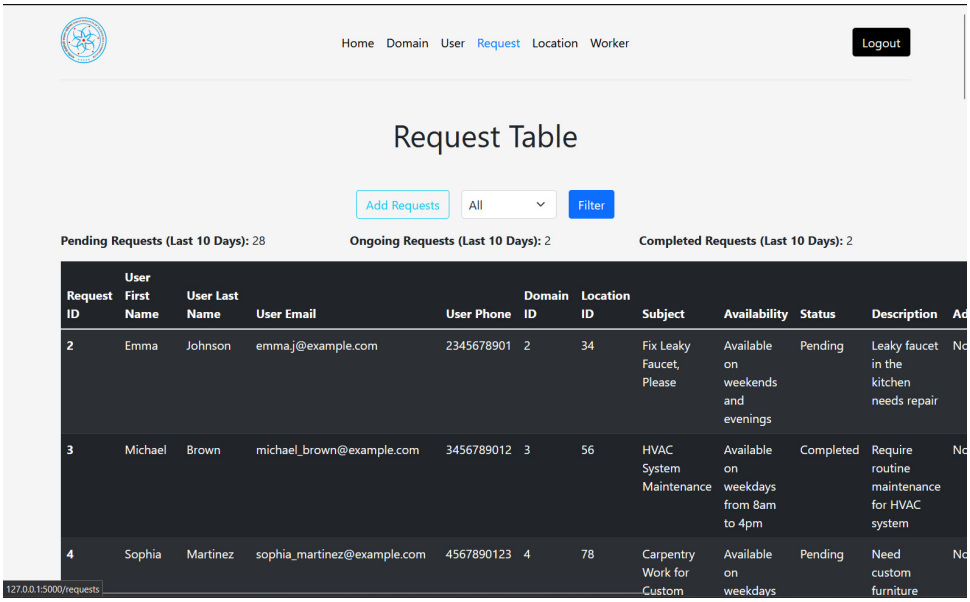


Figure 15: Previous Login page

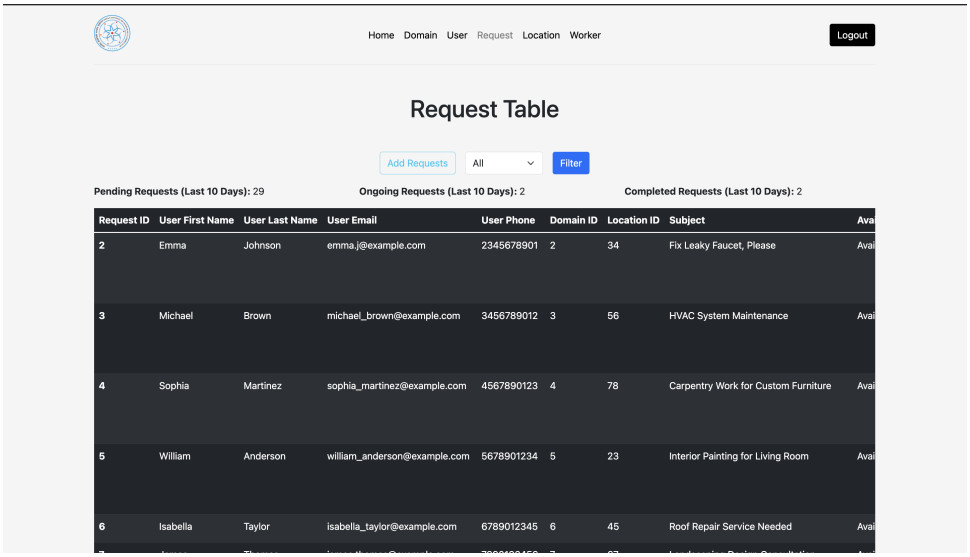


Figure 16: Current Login Page

Question 2

Types of Users

In our webpage, like in the currently existing page, we have categorized the users into two types, namely admin and user type. The nature of the 'user' work will be to add maintenance complaints through the form provided. The 'admin' would have access to all database tables and could update them as she likes. Detailed privileges are provide

Admin

The purpose of the admin user type is to supervise and handle the user type's complaints. Admin would have access to all the tables in the database. She could update or delete requests, workers, users, domains and locations from their respective tables. Attached are the screenshots of various tabs and the functionality of the admin user.

Admin Home Page

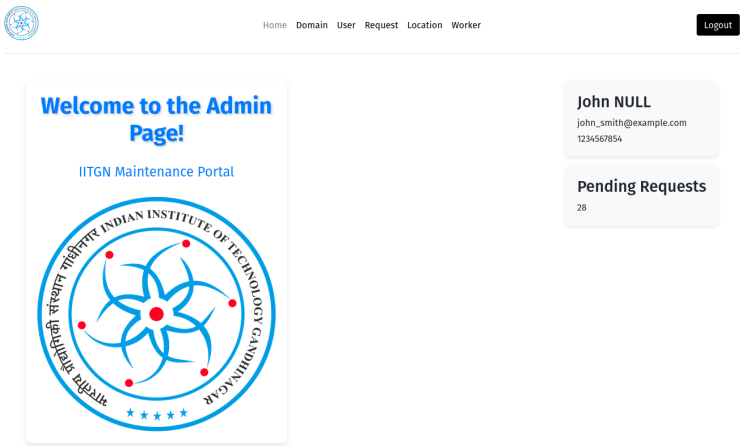


Figure 17: Admin home page

Admin View of Domain Table

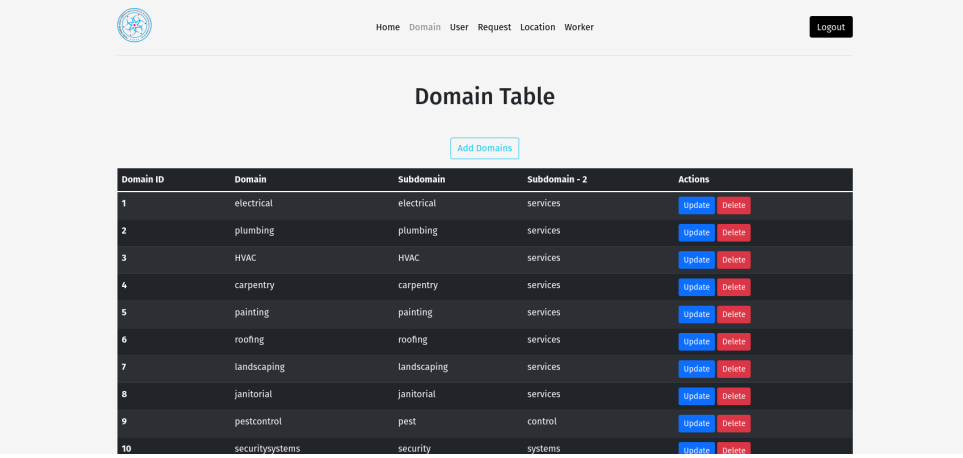


Figure 18: Admin Domain table

Inserting Domain using Admin privileges

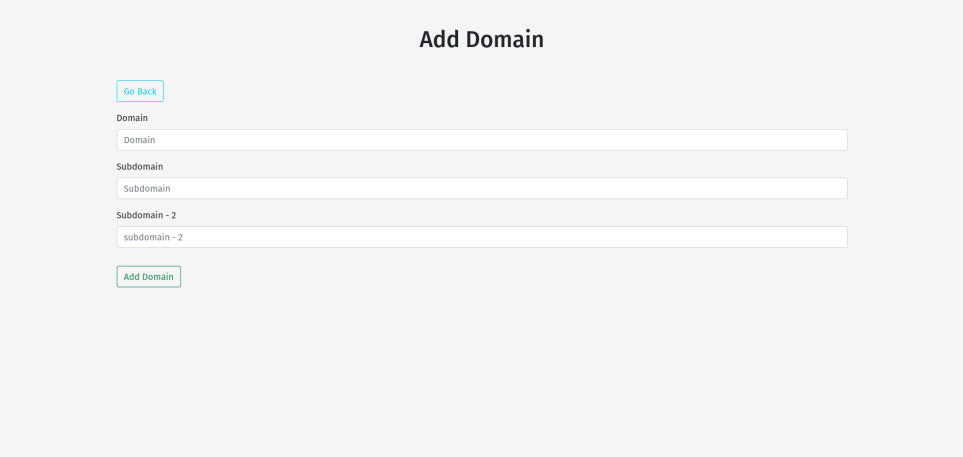


Figure 19: Domain Insertion

Admin View of User Table

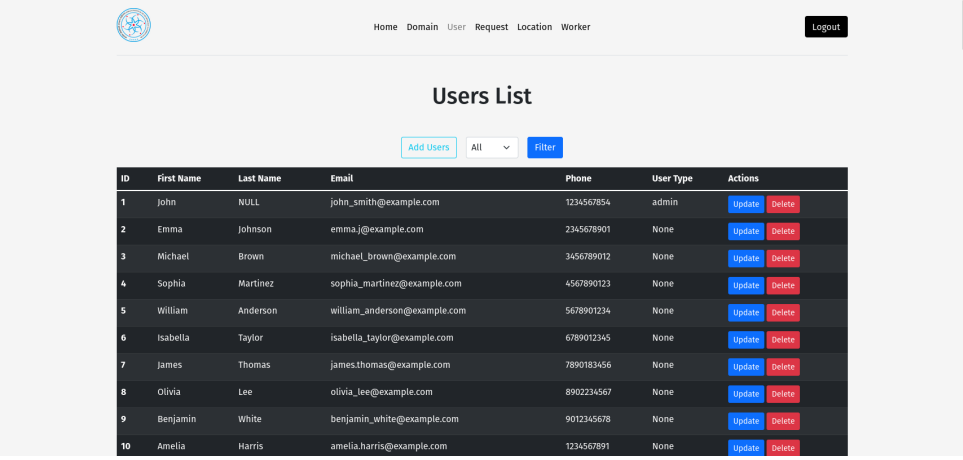


Figure 20: Admin User table

Inserting User using Admin privileges

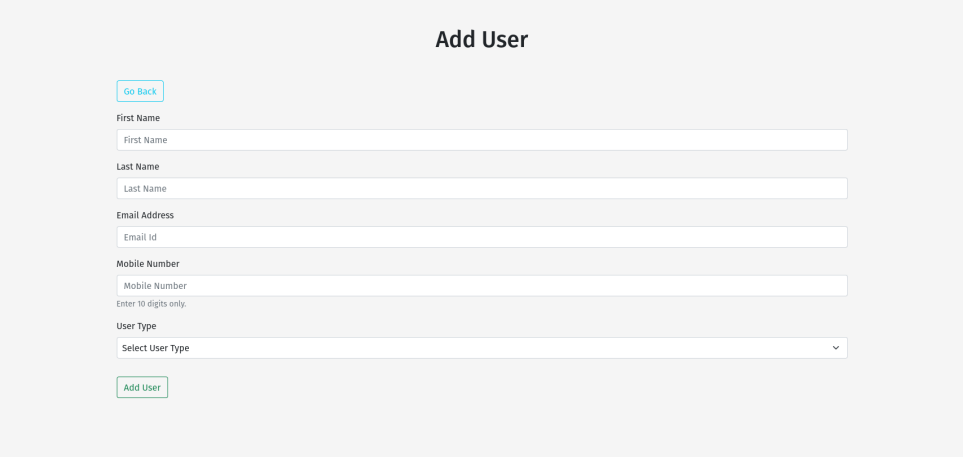


Figure 21: User Insertion

Admin View of Request Table

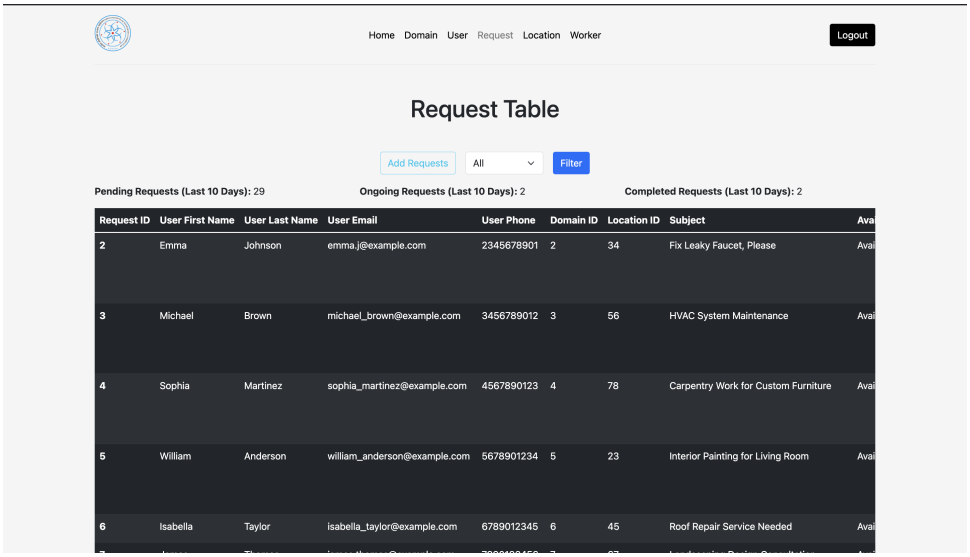


Figure 22: Admin User table

Inserting request using Admin privileges

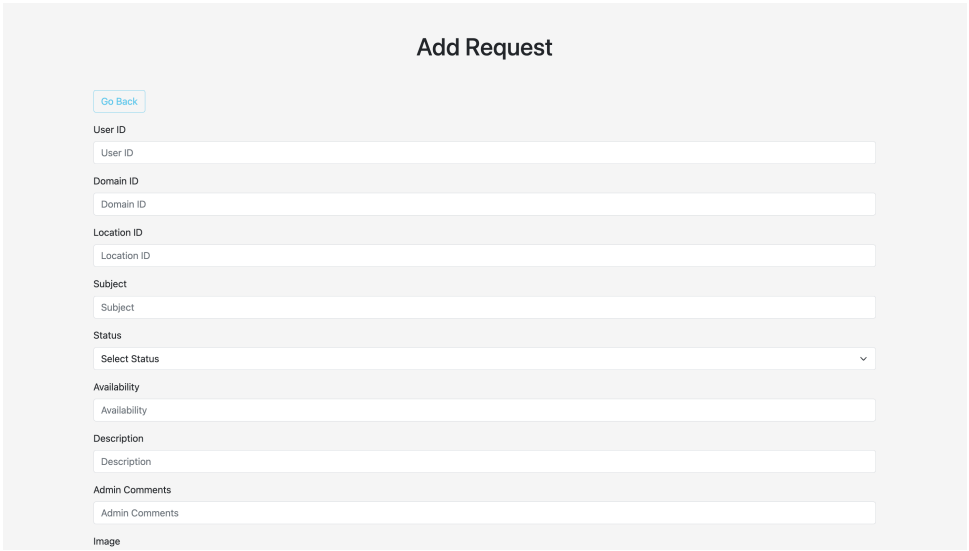


Figure 23: Admin User table

Admin View of Location Table

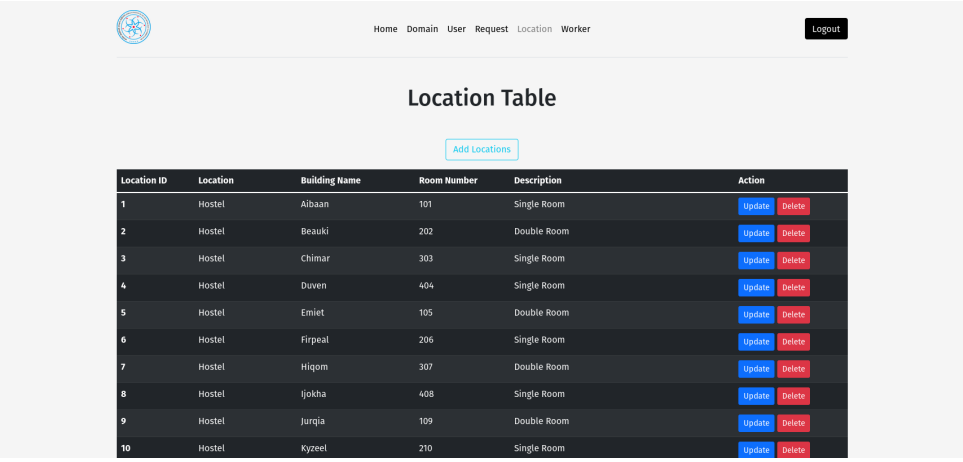


Figure 24: Admin Location table

Inserting Location using Admin privileges

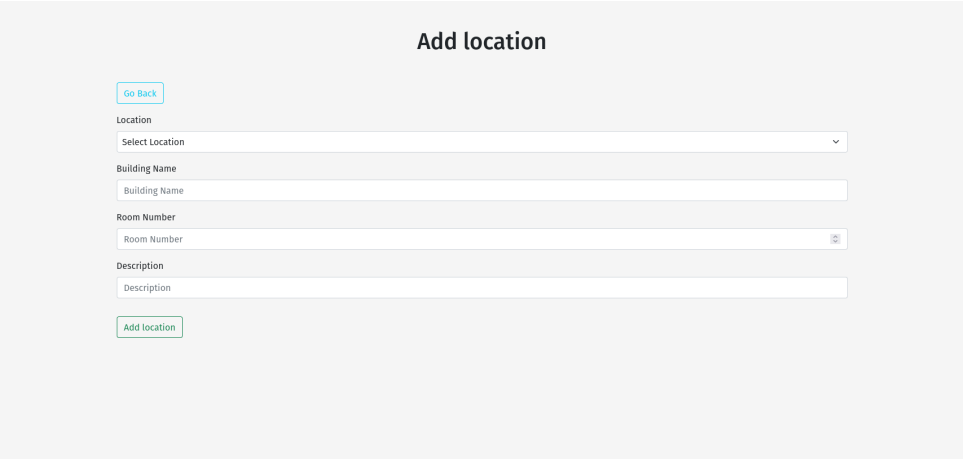


Figure 25: Location Insertion

Admin View of Worker Table

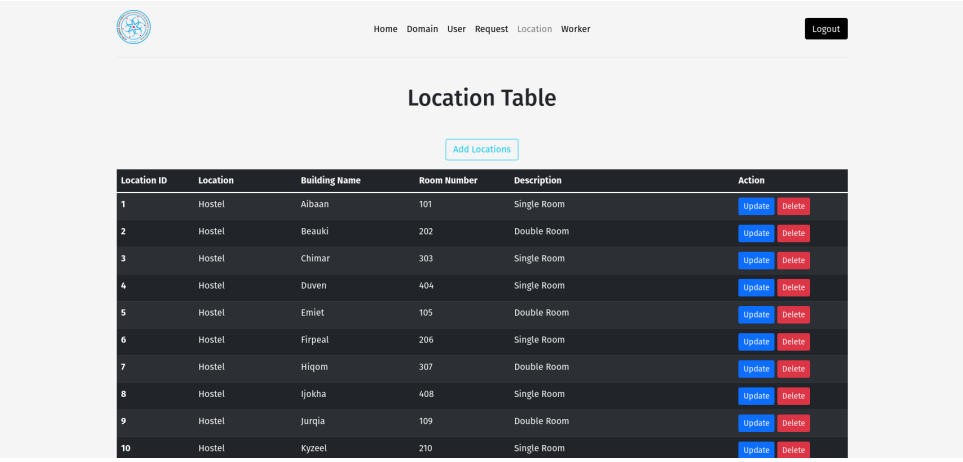


Figure 26: Admin Worker table

Inserting worker using Admin privileges

Figure 27: Worker Insertion

User

The purpose of this user type is mainly to add complaint requests. Most of the users will be in this class. The user is not required to do anything more than make a complaint request and, hence, has no dedicated home page. Additionally, the user could also view his previous requests.

Requesting through user

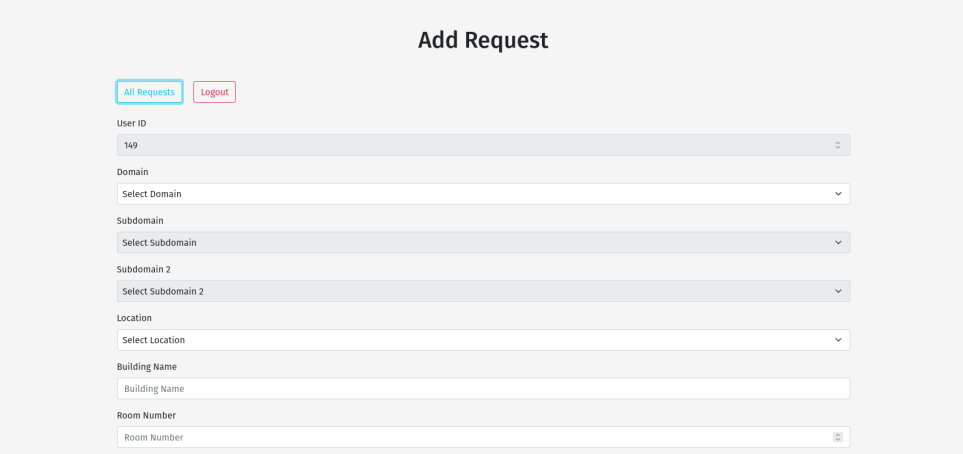


Figure 28: Request Making through user type

Viewing previous Requests

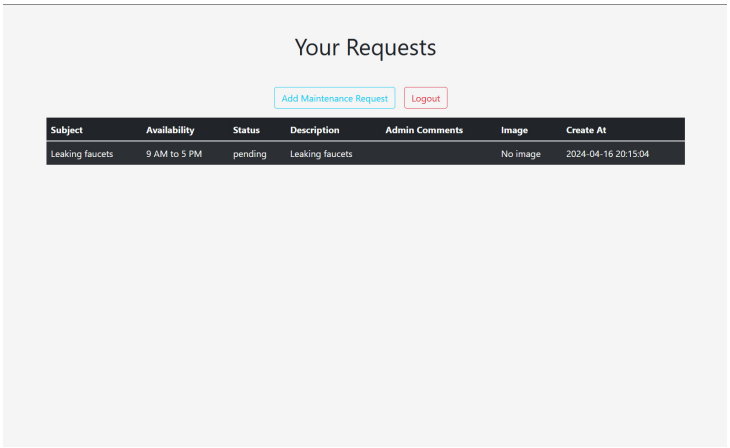


Figure 29: previous requests

Responsibility of G2

Concurrent Multi-User Access

We've categorized users into two types: Users, who can submit requests, and Admins, who possess full access to the database tables, enabling them to update, delete, or add entries. To ensure consistency in multi-user environments, we've implemented commit and rollback mechanisms for every table alteration, preserving transaction atomicity.

Updates Based on Feedback

In response to user feedback, we've enhanced the portal by incorporating user information display within the request table. Additionally, we now showcase the timestamp of each request and provide statistics on request counts per status. These backend modifications accommodate the aforementioned queries efficiently from the database.

Google Authentication for Login and Registration

We've seamlessly integrated Google authentication into our Flask application, leveraging the following libraries:

- **Flask-Login**: Facilitates user session management and authentication functionalities.
- **Flask-OAuthlib**: Empowers OAuth authentication flows, enabling user authentication via Google.

Initially, we define a User model using SQLAlchemy, representing users within our application, inclusive of fields such as user ID, first name, last name, email, and mobile number.

Subsequently, we configure the Google OAuth client utilizing the `OAuth` object provided by Flask-OAuthlib. This involves specifying the client ID and client secret obtained from the Google Developers Console, alongside the requisite scopes for accessing user email information.

The `login_google()` function initiates the Google OAuth flow upon user login attempts, redirecting them to Google's authorization endpoint.

Upon successful authentication with Google, the `google_authorized()` function is triggered. Here, we retrieve the access token and user information from the OAuth response. Validating the email, we proceed to authenticate the user within our application by querying the User model.

For user registration, we offer a signup form where users input their details. Upon submission, a new User instance is instantiated and added to the database, ensuring registration exclusivity to users with IITGN email addresses.

The `logout()` function handles user logout, clearing the session and revoking any associated Google tokens.

Here are some relevant code snippets illustrating the implementation:

```
@auth.route('/login/google')
def login_google():
    return google.authorize(callback=url_for('auth.google_authorized', _external=True))

@auth.route('/login/google/authorized')
@google.authorized_handler
def google_authorized(resp):
    access_token = resp['access_token']
    session['google_token'] = (access_token, '')
    user_info = google.get('userinfo')
```

Responsibility of G1 & G2

Attacks on Deployed Interface

SQL Injection

SQL injection is a cyberattack where malicious SQL code is inserted into input fields to manipulate a database, potentially granting unauthorized access or causing data loss.

Our system is made robust to various SQL injection techniques using proper sanitization of the user-supplied data.

To perform SQL injection attacks, we have insert malicious SQL code in the input fields. However, since our front-end permits a fixed format for various fields (eg, 20 characters in First and Last Name, 10 digit int for mobile number), we had to intercept the packet before it gets passed on to the servers using Burp Suite and Foxy Proxy.

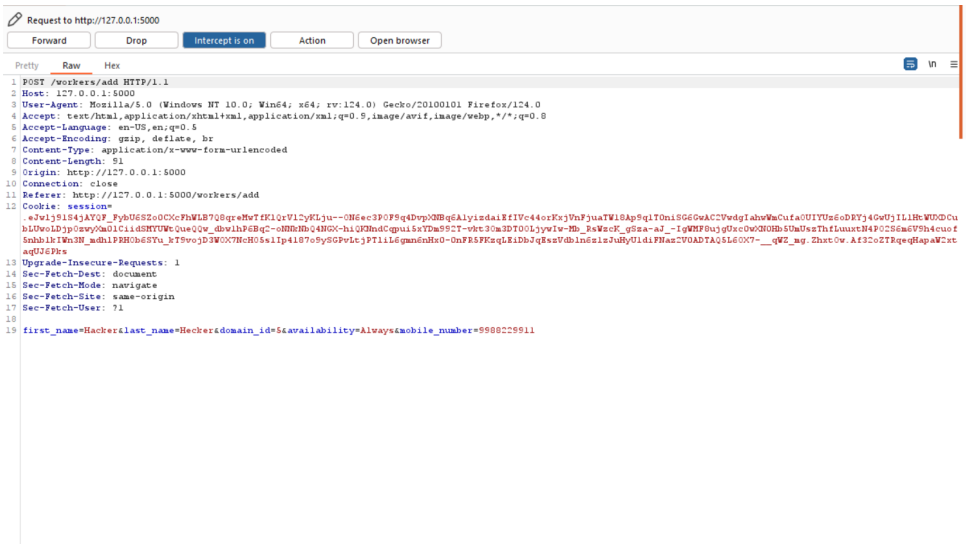


Figure 30: Packet captured with Burp Suite

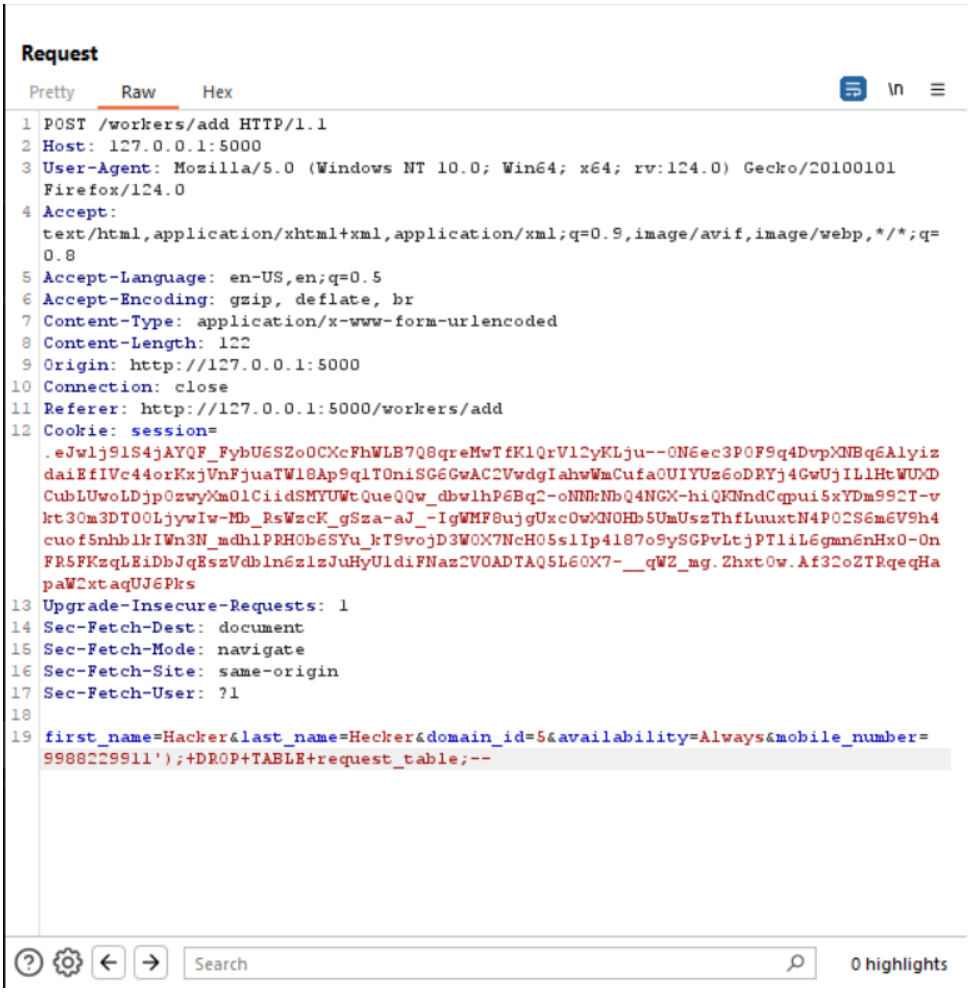


Figure 31: Modifying the packet to inject SQL

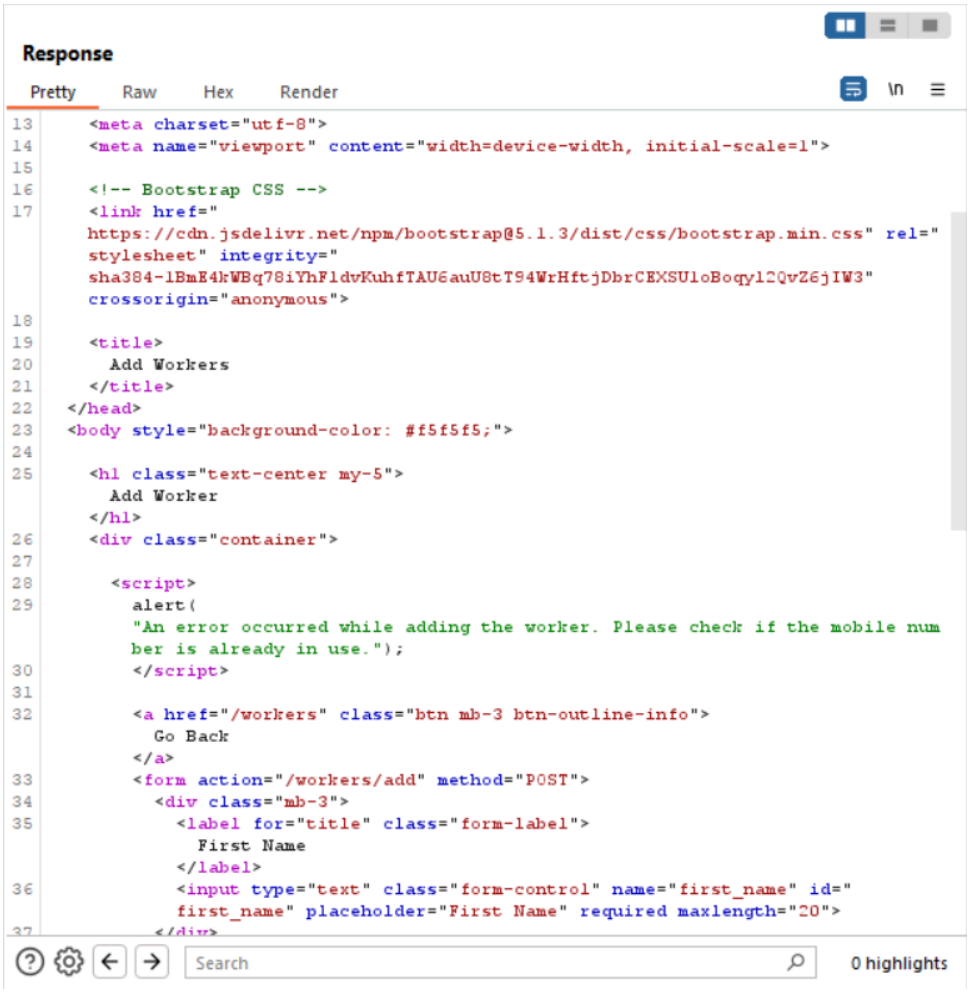


Figure 32: Unsuccessful SQL injection

As we can see the server clearly rejects the input. We explored the reason for this.

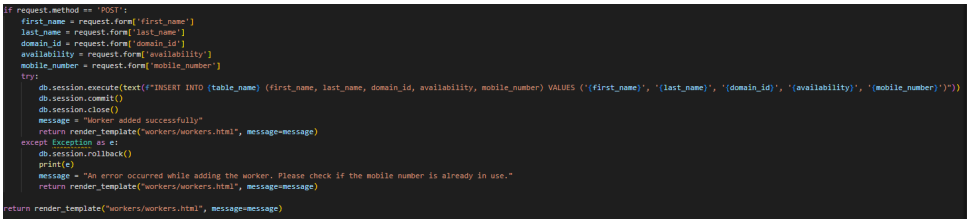


Figure 33: SQL query back-end implementation

The implementation includes the statement:

db.session.execute(text(f"INSERT INTO {table_name} (first_name, last_name, domain_id, availability, mobile_number) VALUES ('{first_name}', '{last_name}', '{domain_id}', '{availability}', '{mobile_number}')"))

If we replace the the values from the sample input we get the query as:

INSERT INTO worker_table (first_name, last_name, domain_id, availability, mobile_number) VALUES ('Hacker', 'Hecker', '5', 'Always', '5566442277'); DROP TABLE request_table;–') which seems to be pretty dangerous SQL injection but it is reasonable handled using text() from SQLAlchemy Library. By using text() to construct the SQL query string, the code can safely incorporate variables like table_name, first_name, last_name, etc., without directly inserting them into the query string. This helps prevent SQL injection because the library or framework can handle the necessary escaping and quoting of the dynamic values to ensure they are treated as data rather than executable SQL code.

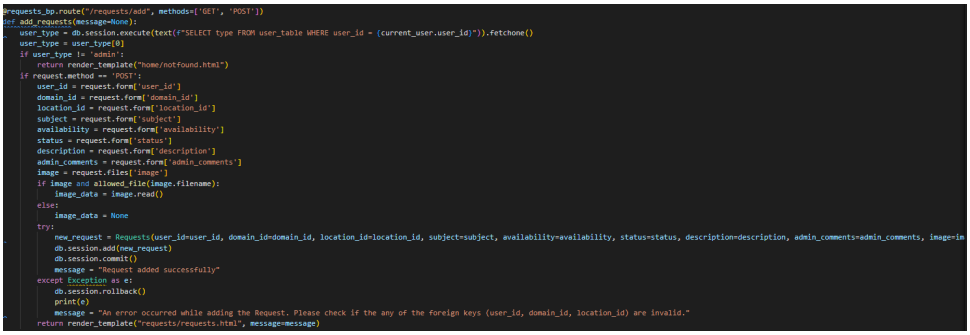


Figure 34: Another SQL query back-end implementation

In this implementation the user-input is handled-well using the proper sanitization of the inputs using the Request Class which inherits from the SQLAlchemy as well. Similarly all the input fields are made invulnerable to SQL Injection attacks.

Cross Site Scripting (XSS)

XSS (Cross-Site Scripting) is a type of cyberattack where malicious scripts are injected into web pages viewed by other users. These scripts can execute in the context of a user's browser, potentially stealing sensitive information, hijacking sessions, or modifying page content. XSS attacks typically fall into two main categories:

- Stored XSS
- Reflected XSS

In Stored XSS, malicious scripts are permanently stored on a website and executed when users access the affected page. This is already properly handled using proper sanitization of the data as explained in case of SQL Injection. In Reflected XSS, the injected script is reflected off the web server, often through a URL or input field, and executed when the victim visits a crafted link or submits a form. To perform this attack we used multiple XSS payloads and automized the attack using the Burp Suite and Foxy Proxy.

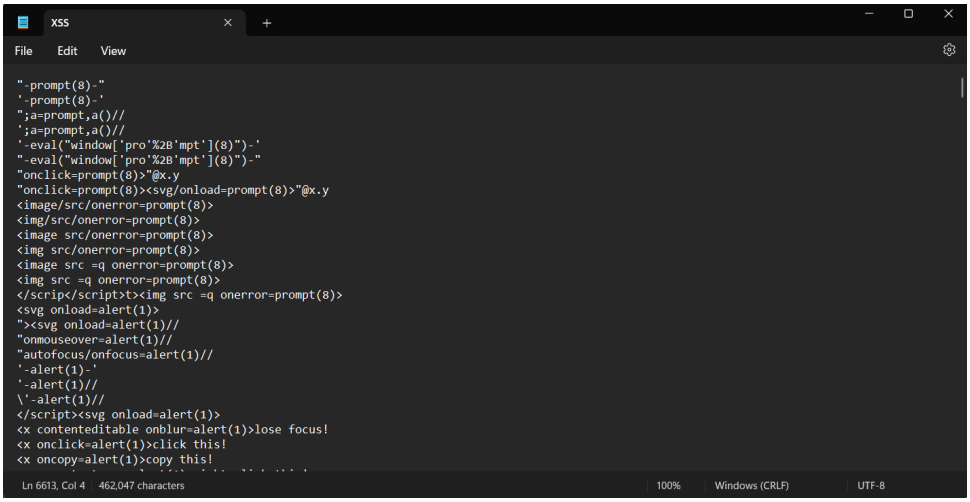


Figure 35: XSS payloads

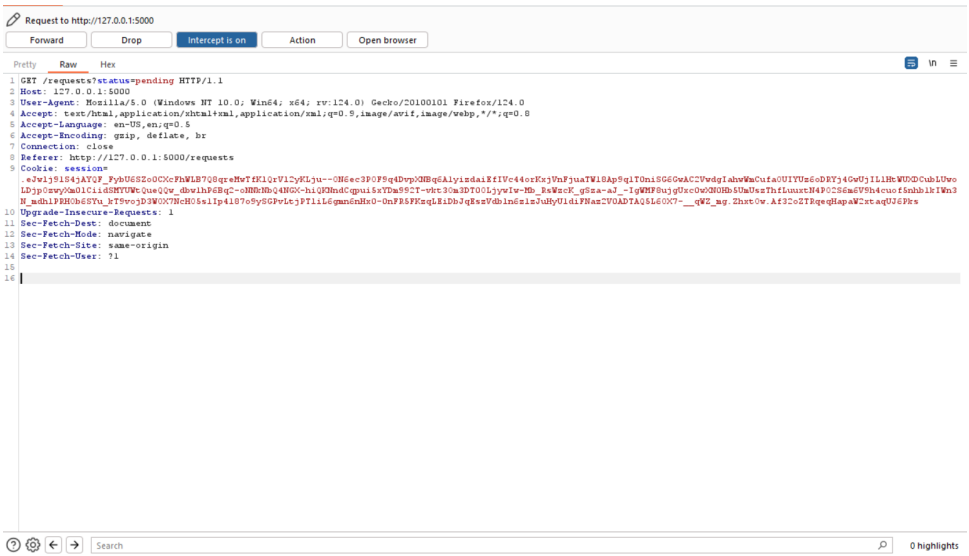


Figure 36: Intercepted Packet

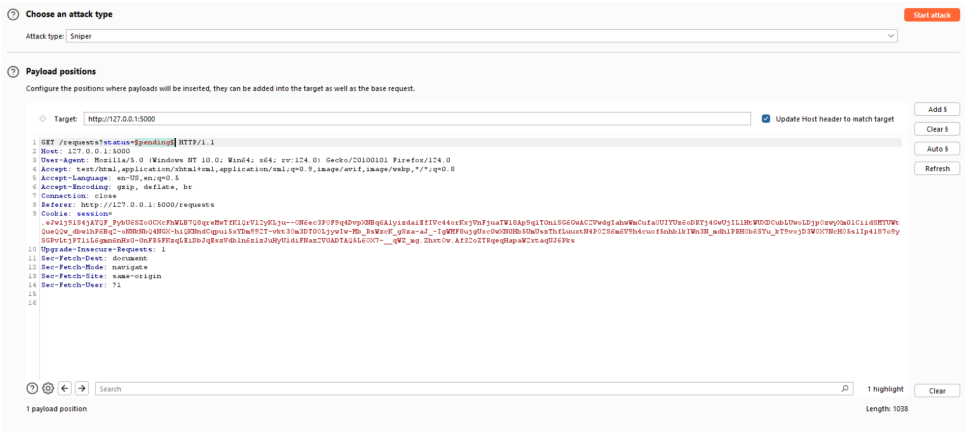


Figure 37: Sending the packet over to Intruder and Beginning the attack

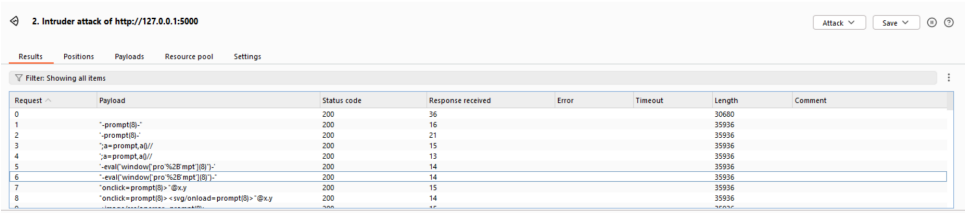


Figure 38: Attack Result

The XSS attacks also failed. We can find the reason on inspecting how the frontend is implemented.

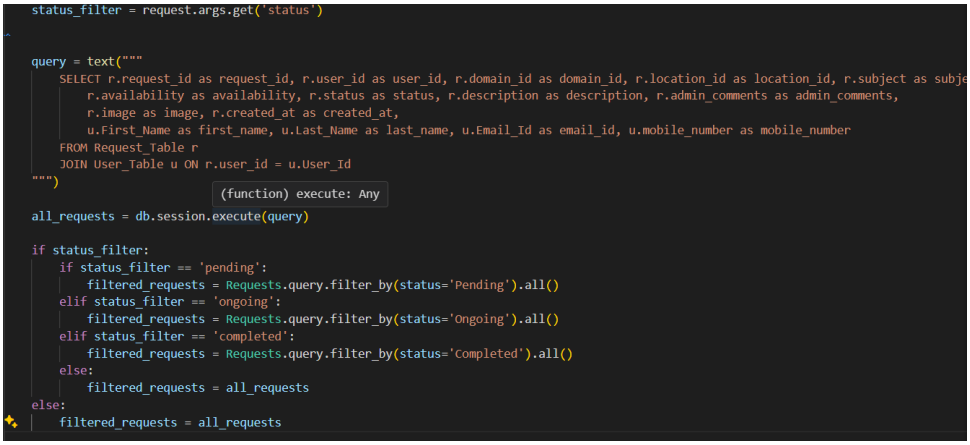


Figure 39: Frontend Implementation

As you can see in the case of the frontend the status_filter is only being used for comparison and its value is not being accessed anywhere in the code. This makes the code robust to Reflected XSS attacks, which rely on the property of the variable being accessed somewhere in the code.

Cross-Site Request Forgery (CSRF)

A Cross-Site Request Forgery (CSRF) attack occurs when a malicious website or program causes a user's web browser to perform an unwanted action on a trusted site for which the user is currently authenticated.

For Implementing this Attack we have made a malicious webpage for Free Coupon Voucher that has pre-filled values for the form Action that we are performing from the main website. When the user clicks the malicious web page it results in automatically submitting the request in the database.

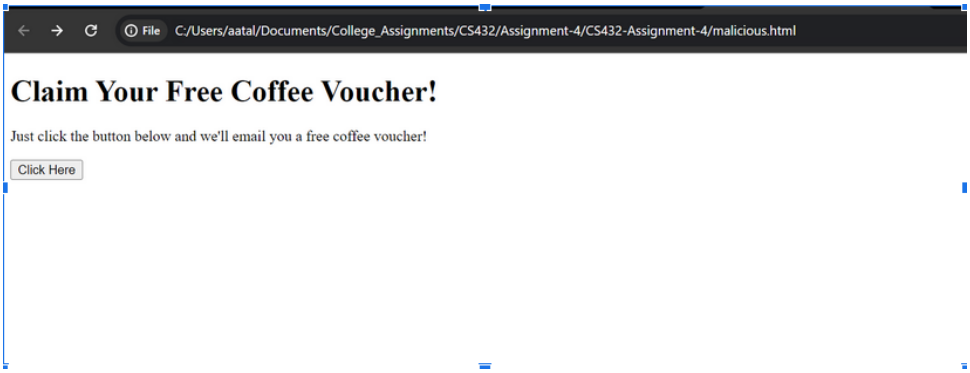


Figure 40: Implementing Malicious Webpage

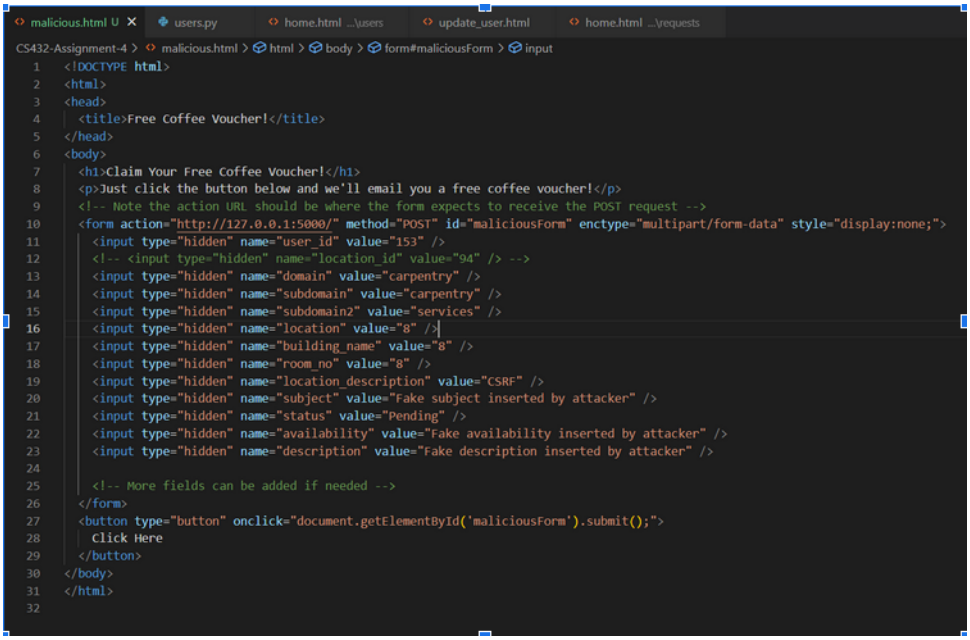


Figure 41: Frontend-Implementation of the Webpage

To prevent this kind of attack, Implementing Anti-CSRF Tokens. The server should generate a unique anti-CSRF token for each user session. This token is included as a hidden field in every form that causes a state change on the server when submitted. The server should then verify the token upon form submission before taking any action.

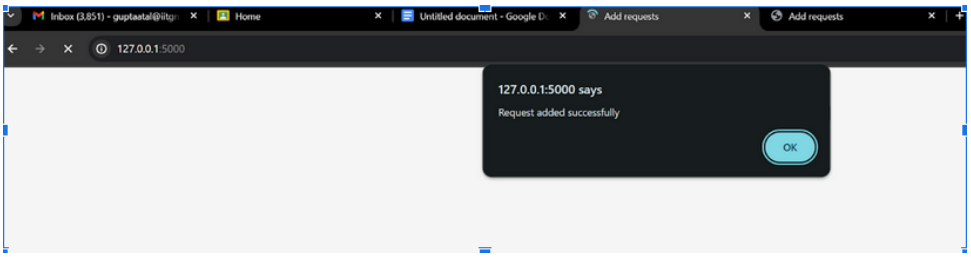
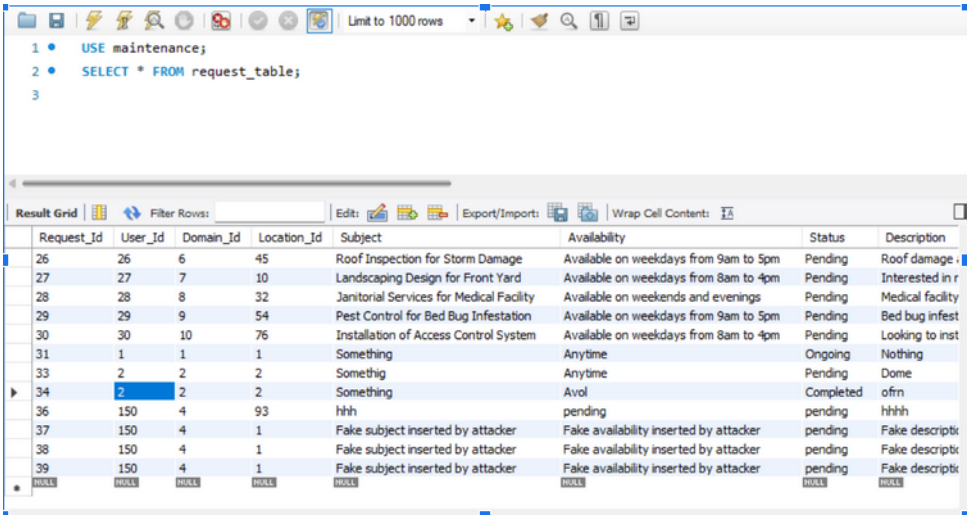


Figure 42: Alert for fake request is being added



Request_Id	User_Id	Domain_Id	Location_Id	Subject	Availability	Status	Description
26	26	6	45	Roof Inspection for Storm Damage	Available on weekdays from 9am to 5pm	Pending	Roof damage
27	27	7	10	Landscaping Design for Front Yard	Available on weekdays from 8am to 4pm	Pending	Interested in r
28	28	8	32	Janitorial Services for Medical Facility	Available on weekends and evenings	Pending	Medical facility
29	29	9	54	Pest Control for Bed Bug Infestation	Available on weekdays from 9am to 5pm	Pending	Bed bug infest
30	30	10	76	Installation of Access Control System	Available on weekdays from 8am to 4pm	Pending	Looking to inst
31	1	1	1	Something	Anytime	Ongoing	Nothing
33	2	2	2	Somethig	Anytime	Pending	Dome
34	2	2	2	Something	Avol	Completed	ofrn
36	150	4	93	hhh	pending	pending	hhhh
37	150	4	1	Fake subject inserted by attacker	Fake availability inserted by attacker	pending	Fake descripti
38	150	4	1	Fake subject inserted by attacker	Fake availability inserted by attacker	pending	Fake descripti
39	150	4	1	Fake subject inserted by attacker	Fake availability inserted by attacker	pending	Fake descripti

Figure 43: Fake data is displayed in the database

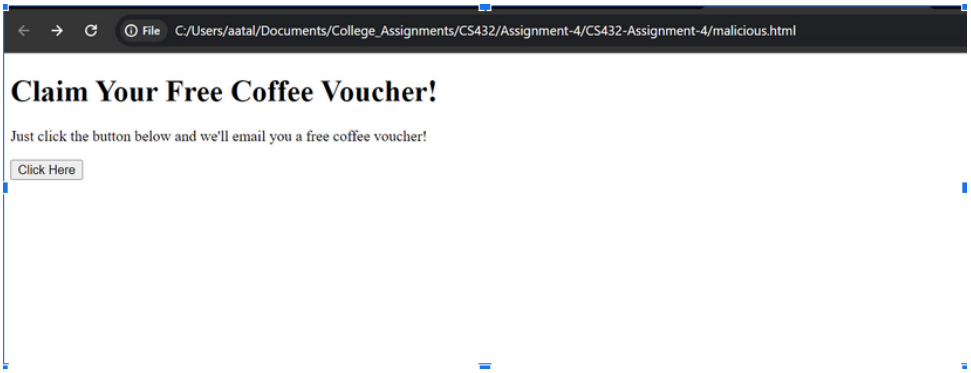


Figure 44: Implementing Malicious Webpage

Figure 45: Prevented a CSRF attack

Directory Traversal Attack

Directory Traversal attacks exploit vulnerabilities to navigate beyond restricted directories, potentially accessing sensitive files or executing malicious scripts, posing a significant threat to web server security. Directory Traversal Attacks are prevented using proper input validation and access controls. Error page is designed for invalid address and user-inputs aren't directly used into the url.

None of our user-inputs is being used directly to access any of document/file in our directory.

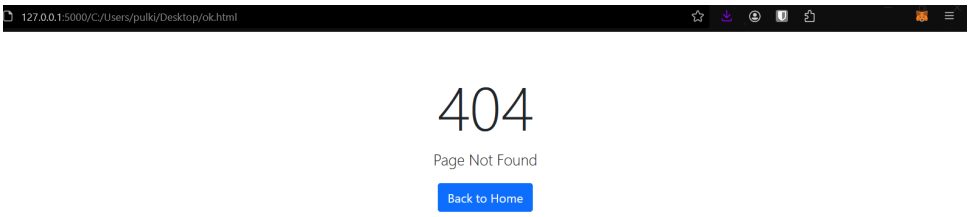


Figure 46: Error Page is displayed

We automized the attack sing Burp Suite to try for multiple types of payload

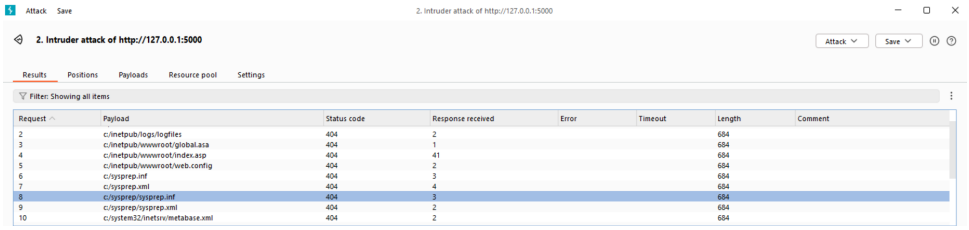


Figure 47: Automated Attacks results in 404

Hence, our website is secure from Directory Traversal Attacks.

Finalized Relations and Constraints

```
mysql> DESCRIBE domain_table;
+--------------+-------------+------+-----+---------+----------------+
| Field        | Type        | Null | Key | Default | Extra          |
+--------------+-------------+------+-----+---------+----------------+
| Domain_Id    | int         | NO   | PRI | NULL    | auto_increment |
| Domain       | varchar(20) | NO   |     | NULL    |                |
| Subdomain    | varchar(40) | NO   |     | NULL    |                |
| Subdomain_2  | varchar(40) | NO   |     | NULL    |                |
+--------------+-------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

Figure 48: Domain Table

```
mysql> DESCRIBE worker_table;
+-----------------+--------------+------+-----+---------+----------------+
| Field           | Type         | Null | Key | Default | Extra          |
+-----------------+--------------+------+-----+---------+----------------+
| Worker_Id       | int          | NO   | PRI | NULL    | auto_increment |
| First_Name      | varchar(20)  | NO   |     | NULL    |                |
| Last_Name       | varchar(20)  | YES  |     | NULL    |                |
| Domain_Id       | int          | NO   | MUL | NULL    |                |
| Availability    | varchar(100) | NO   |     | NULL    |                |
| Mobile_Number   | char(10)     | NO   | UNI | NULL    |                |
+-----------------+--------------+------+-----+---------+----------------+
6 rows in set (0.00 sec)
```

Figure 49: Worker Table

```
mysql> DESCRIBE location_table;
+---------------+-----------------------------------------------------------------------------------------------------------+------+-----+---------+----------------+
| Field         | Type                                                                                                      | Null | Key | Default | Extra          |
+---------------+-----------------------------------------------------------------------------------------------------------+------+-----+---------+----------------+
| Location_Id   | int                                                                                                       | NO   | PRI | NULL    | auto_increment |
| Location      | enum('Hostel','Housing','Academic','Infrastructure','Guest House','Central Arcade','Sports Complex','Research Park') | YES  |     | NULL    |                |
| Building_Name | varchar(100)                                                                                              | YES  |     | NULL    |                |
| Room_No       | int                                                                                                       | YES  |     | NULL    |                |
| Description   | varchar(200)                                                                                              | YES  |     | NULL    |                |
+---------------+-----------------------------------------------------------------------------------------------------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

Figure 50: Location Table

```
mysql> DESCRIBE user_table;
+---------------+-----------------------+------+-----+---------+----------------+
| Field         | Type                  | Null | Key | Default | Extra          |
+---------------+-----------------------+------+-----+---------+----------------+
| User_Id       | int                   | NO   | PRI | NULL    | auto_increment |
| First_Name    | varchar(20)           | NO   |     | NULL    |                |
| Last_Name     | varchar(20)           | YES  |     | NULL    |                |
| Email_Id      | varchar(255)          | NO   | UNI | NULL    |                |
| mobile_number | char(10)              | NO   | UNI | NULL    |                |
| type          | enum('user','admin') | YES  |     | NULL    |                |
+---------------+-----------------------+------+-----+---------+----------------+
6 rows in set (0.00 sec)
```

Figure 51: User Table

```
mysql> DESCRIBE request_table;
+----------------+--------------+------+-----+--------------------+--------------------+
| Field          | Type         | Null | Key | Default            | Extra              |
+----------------+--------------+------+-----+--------------------+--------------------+
| Request_Id     | int          | NO   | PRI | NULL               | auto_increment     |
| User_Id        | int          | NO   | MUL | NULL               |                    |
| Domain_Id      | int          | NO   | MUL | NULL               |                    |
| Location_Id    | int          | NO   | MUL | NULL               |                    |
| Subject        | varchar(100) | NO   |     | NULL               |                    |
| Availability   | varchar(100) | NO   |     | NULL               |                    |
| Status         | varchar(20)  | NO   |     | NULL               |                    |
| Description    | varchar(200) | YES  |     | NULL               |                    |
| Admin_Comments | varchar(200) | YES  |     | NULL               |                    |
| Image          | longblob     | YES  |     | NULL               |                    |
| created_at     | timestamp    | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED |
+----------------+--------------+------+-----+--------------------+--------------------+
11 rows in set (0.04 sec)
```

Figure 52: Request Table

All the relations and their constraints are present and valid as per the ER diagram drawn as a part of Assignment 1.

Contributions

Group 1

- **Pulkit Gautam:** Led Group 1, worked on implementation and prevention of SQL Injection, XSS and Directory Traversal attacks and wrote the documentation of G1 & G2.
- **Manav Parmar:** Responsible for re-designing the front-end and collecting the feedback during both the cycles. Also, helped with the documentation of G1.
- **Gaurav Rawat:** Responsible for collecting the feedback during cycle 1 and helped with the documentation.
- **Hrishikesh Birje:** Responsible for re-designing the interface.

Group 2

- **Shubh Agarwal:** Led Group 2, worked on implementing Login by Google, Concurrent Multi-User Access and wrote documentation for G2.
- **Atal Gupta:** Responsible for implementing CSRF Attack and implementing security against it, helped implement changes to the backend as specified by stakeholders.
- **Ishva Patel:** Helped with implementing Multi-User Access, helped with documentation.
- **Aman Singh:** Updated the backend based on the feedback received from stakeholders also, updated the login page UI and helped with documentation.