CS 432: Databases
Sports Management System

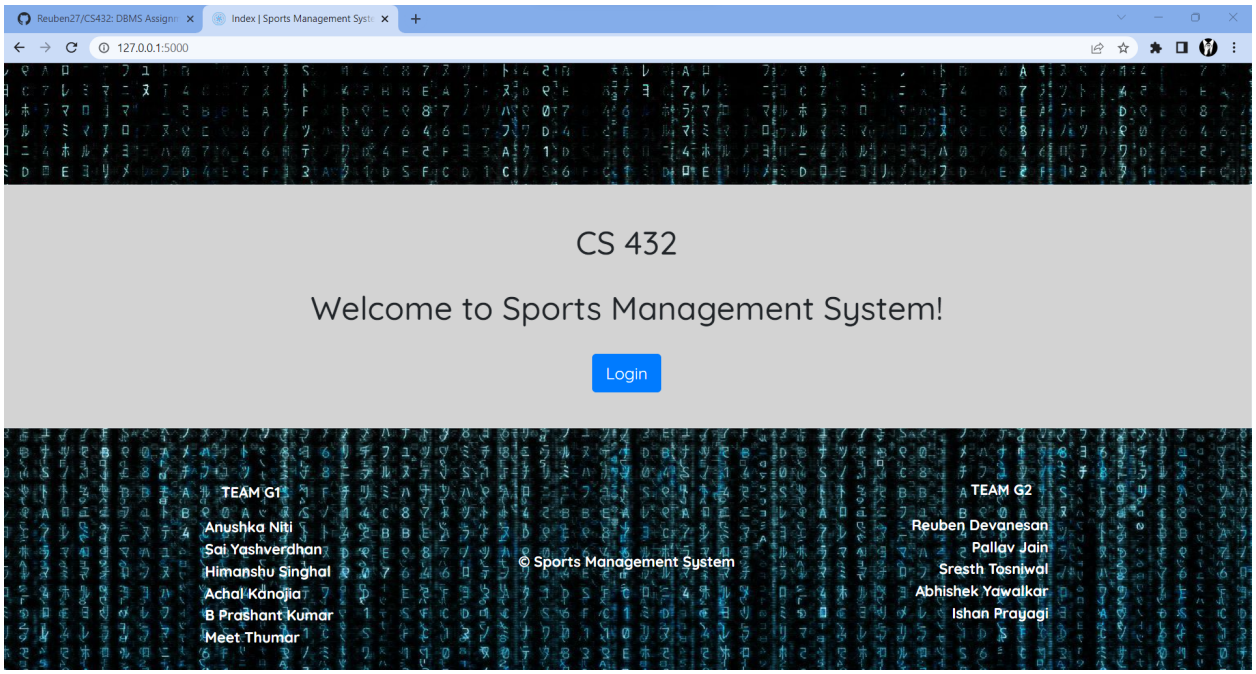
Group 5 (Group Name: BTech 19)

- Reuben Devanesan, 19110059
- Meet Thumar, 19110172
- Himanshu Singhal, 19110051
- Sai Yashverdhan, 19110027
- Sresth Tosniwal, 19110033
- Anushka Niti, 19110040
- Pallav Jain, 19110156
- Abhishek Yawalkar, 19110070
- B Prashant Kumar, 19110075
- Achal Kanojia, 19110108
- Aditya Shekhar, 19110002
- Ishan Prayagi, 19110194

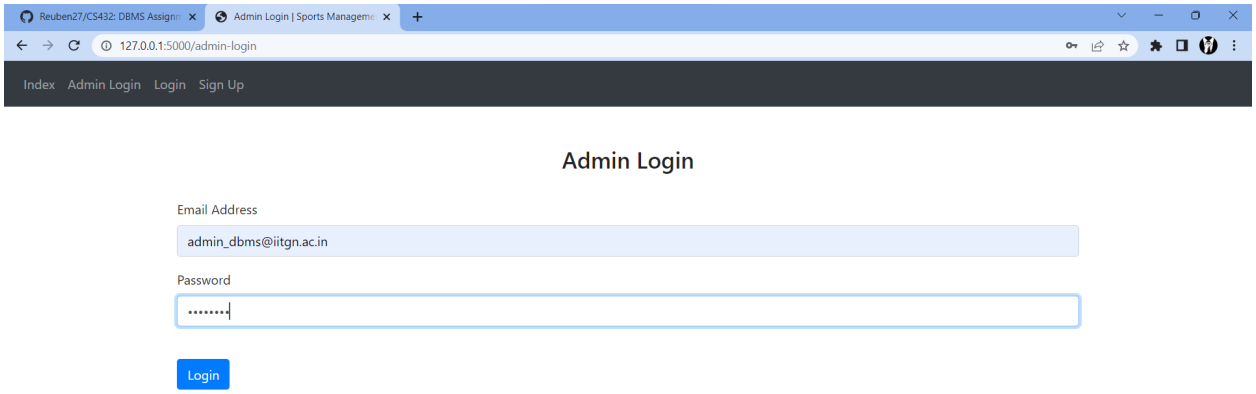
Responsibility of G1:

Q 1.

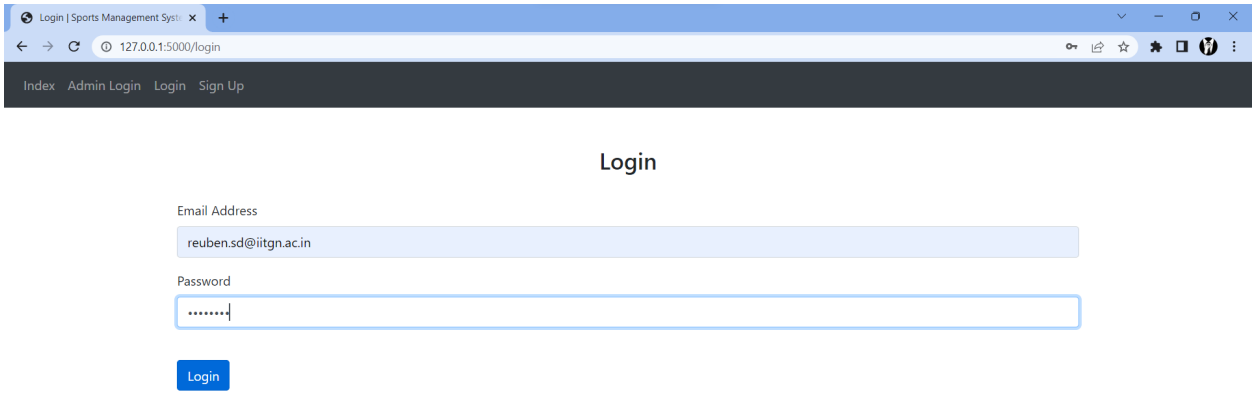
Screenshot before first feedback:



Home Page

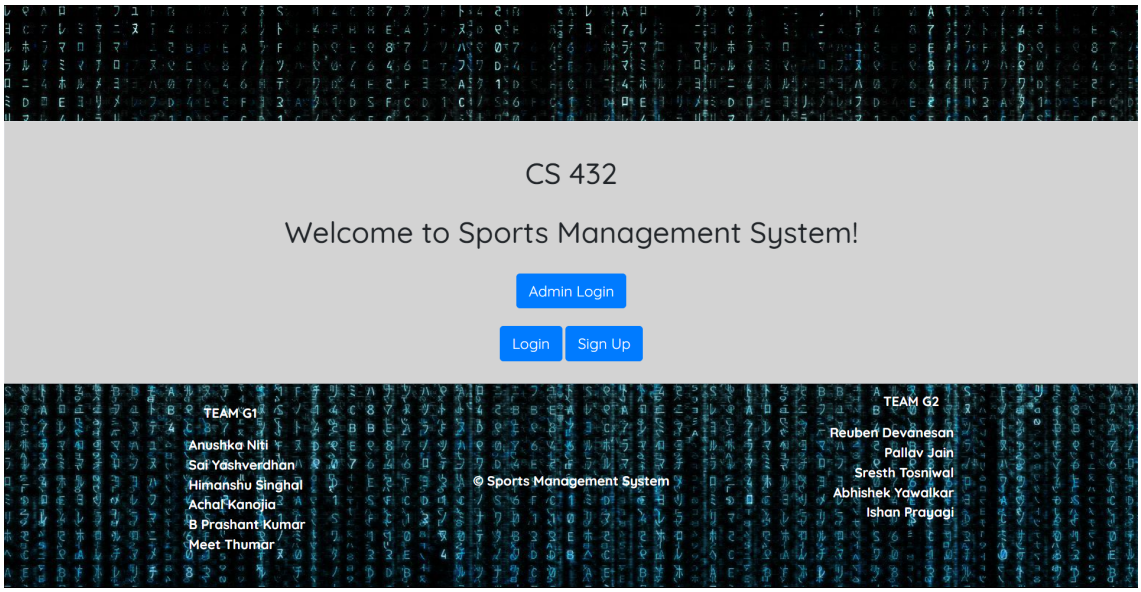


Admin Login Page

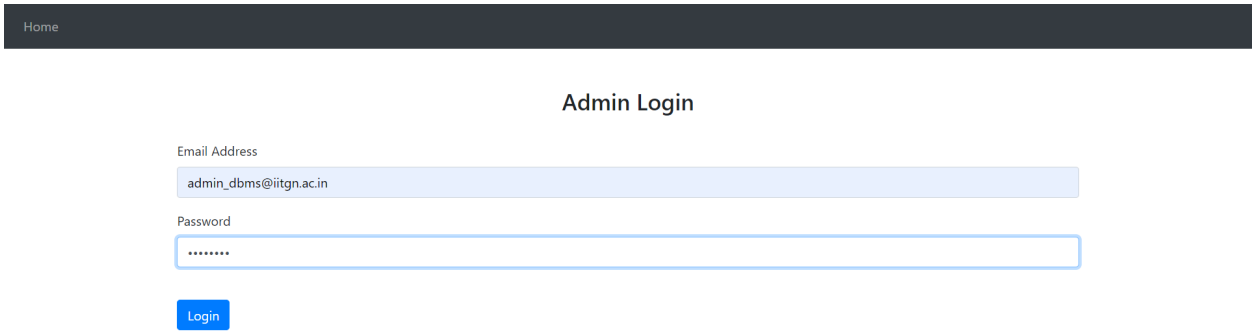


User Login Page

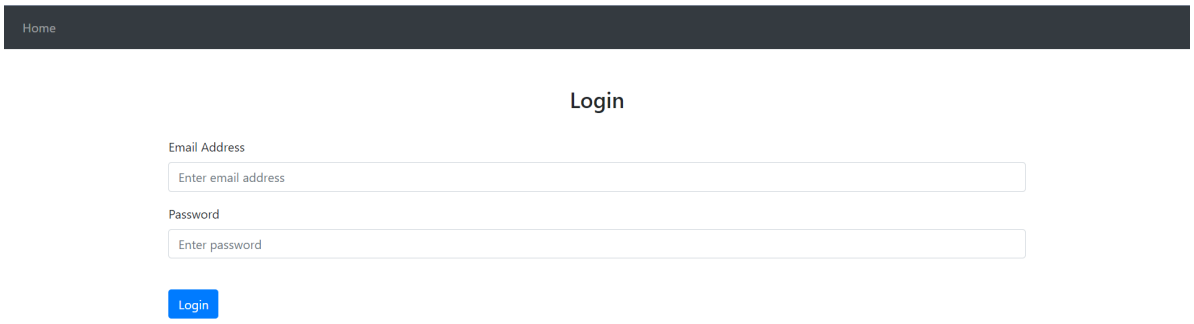
Screenshot after the final feedback



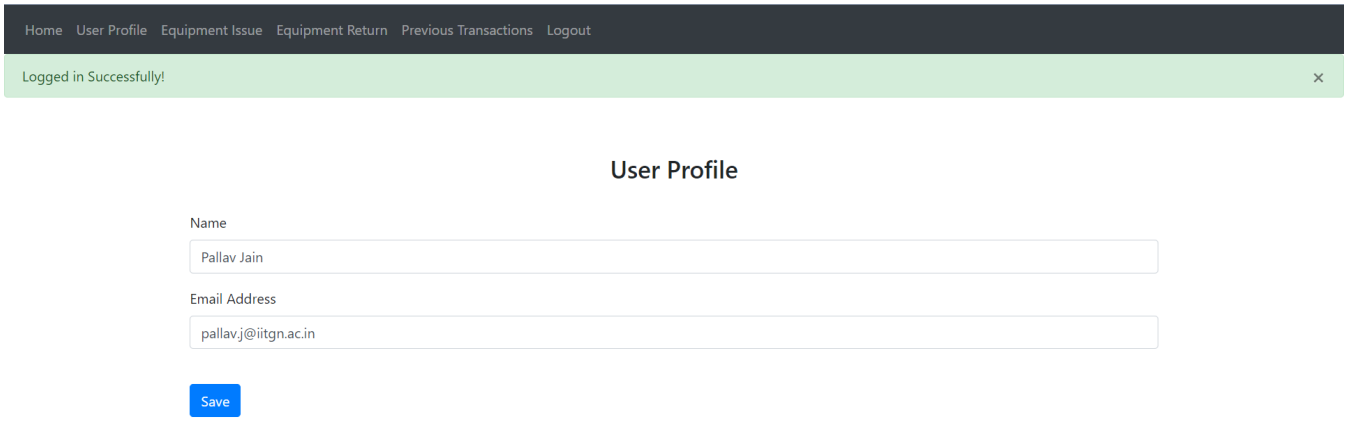
Home Page



Admin Login



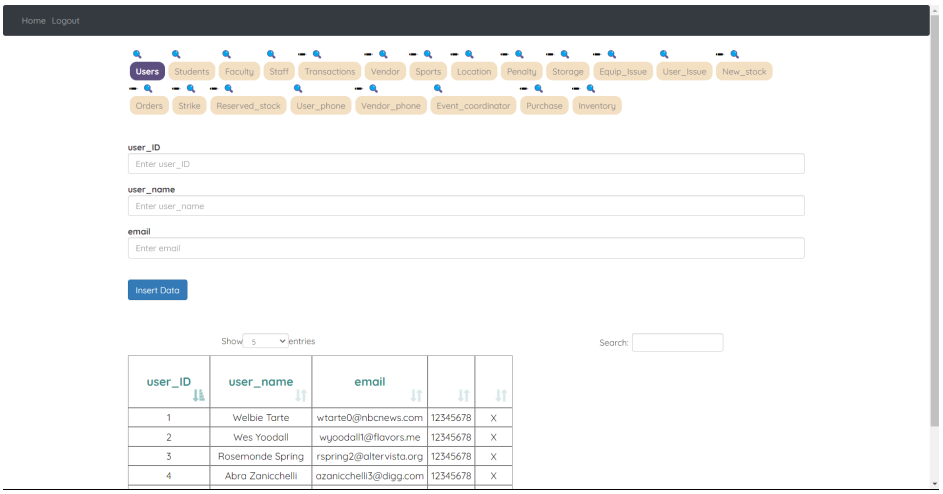
User Login



User Profile Page

Users can only access personal data and issue and return equipment

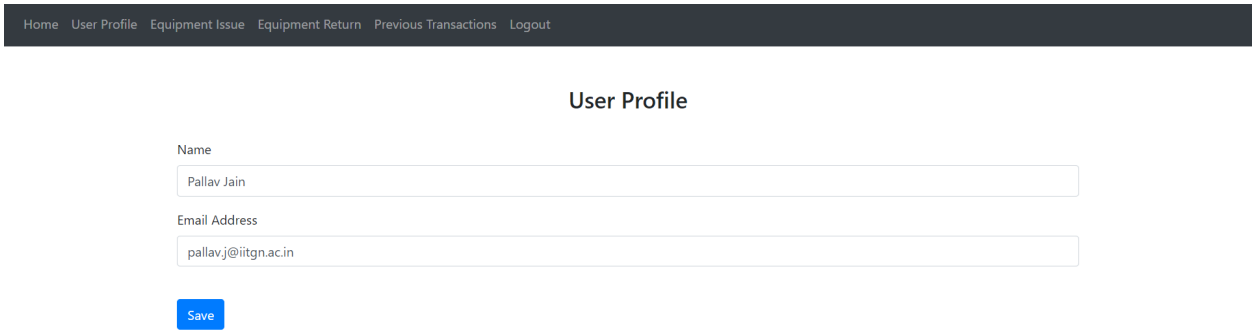
Q 2.



Admin can access all the tables

Privileges to the admin:

- Admin can see all the tables. He/she can perform various operations like update, change, and delete on all tables except the admin table.
- Admin cannot access the admin table, i.e., admin cannot add new admins. That can be done through the backend only.



A user can only access his data

Privileges to the user

- Users can create an account if he/she doesn't have an account by using the "Sign-In" button which is located at home.
- After Sign-in the user can log in to the site. Where users can issue the equipment **only for that user**.
- Update Profile details: Users can simply change their name by updating their name in "User Profile".
- Issue: entering details of available equipment and time&date. At a time users can issue only one piece of equipment.

- Return: The user can return an item by simply reverting back to the assigned transaction id in the drop-down in the equipment return window.
- Damage: If the item is damaged while returning, the user should mark the damage checkbox under the supervision of the issuer.
- Transaction: After issuing or returning equipment can see their whole transaction history, on the "Transaction" menu.

Responsibility of G2:

Q 1.
When multiple users with different roles can access and update the database concurrently, it is important to ensure that data is not being modified by more than one user at the same time to avoid data inconsistencies and conflicts.

We have used locking mechanisms, which prevent multiple users from modifying the same data at the same time. We used table **Write -Lock** to prevent concurrent access to a table by multiple users. For example, the **LOCK TABLES** statement to acquire a table lock in MySQL and **UNLOCK TABLES** to release the lock.

Changes are made in the **setup.sql** file

Q 2.
Changes are implemented as per the feedback received from stakeholders.

Responsibility of G1 & G2:

Q 1.

<u>ATTACK 01</u>: SQL Injection on login (User Authentication Page)

Initially, we created the user authentication page using SQL query as shown below.

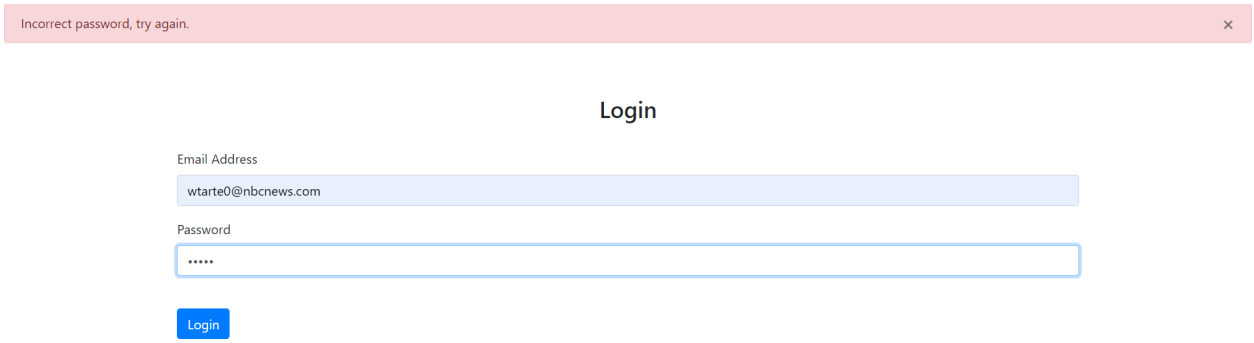
```
def login():
    if request.method == 'POST':
        email = request.form.get('email')
        password = request.form.get('password')

        cur = mysql.connection.cursor(MySQLdb.cursors.DictCursor)
        sql_query = "SELECT * FROM Users WHERE email='{}'".format(email)
        cur.execute(sql_query)
        user = cur.fetchone()
```

A snippet of our Users database is as follows:

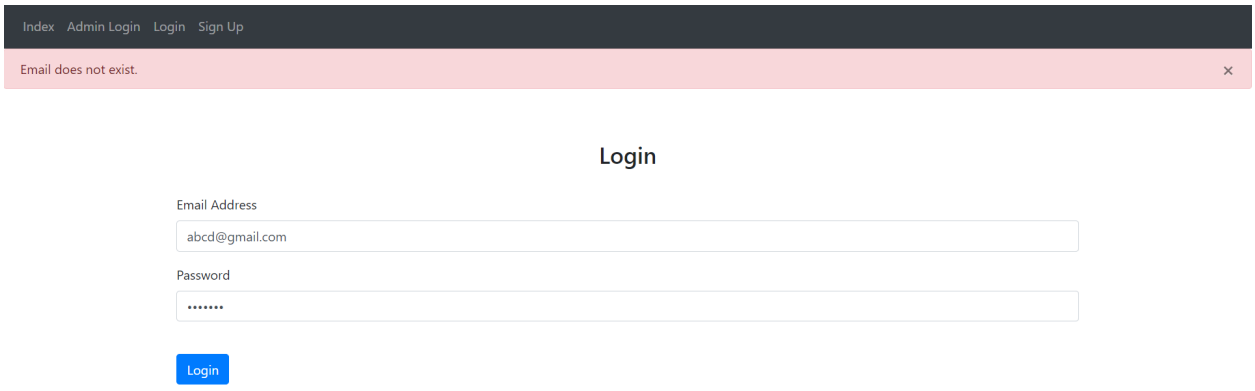
user_ID	user_name	email	password
1	Welbie Tarte	wtarte0@nbcnews.com	12345678
2	Wes Yoodall	wyoodall1@flavors.me	12345678
3	Rosemonde Spring	rspring2@altervista.org	12345678
4	Abra Zanicchelli	azanicchelli3@digg.com	12345678
5	Lanni Smyley	lsmyley4@github.io	12345678
6	Kathlin Flinders	kflinders5@aol.com	12345678
7	Anderson Lunny	alunny6@drupal.org	12345678
8	Dulcea Kiff	dkiff7@ihg.com	12345678

Working Demo #1: Existing email address with its incorrect password
When we enter an email address present in our Users database with the wrong password, it throws an error - ‘Incorrect password, try again.’



Working Demo #2: Non-existing email address

When we enter an email address that is not present in the Users database, the login page shows an error- 'Email does not exist.'



Vulnerability Exploitation:

Now, we will exploit the vulnerability possessed by the SQL query used for the login page.

The query is:

sql_query = "SELECT * FROM Users WHERE email='{}'".format(email)

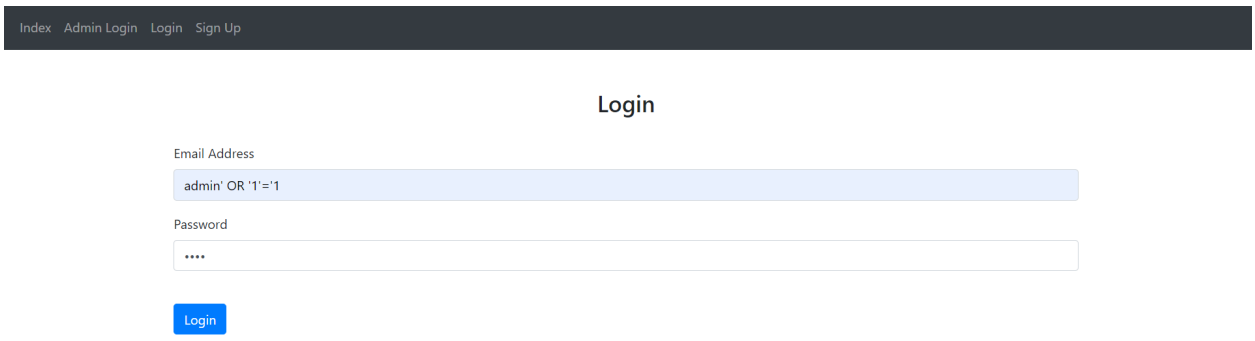
Where the condition will be **email = 'email'** where **email** is the one entered on the login page.

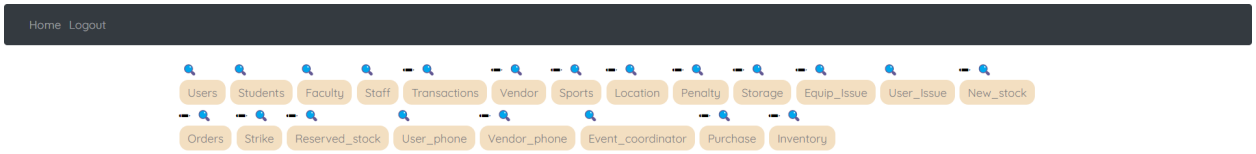
We can exploit the SQL query if we try to make the result of the where condition always TRUE.

One such example of an email address can be:

admin' or '1'='1

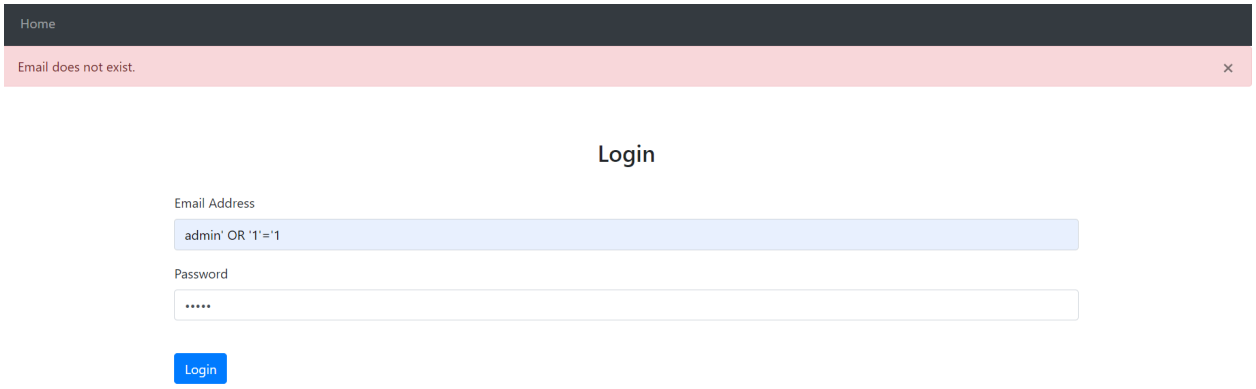
Using this email address with any password, we can bypass our code and get logged in as shown:



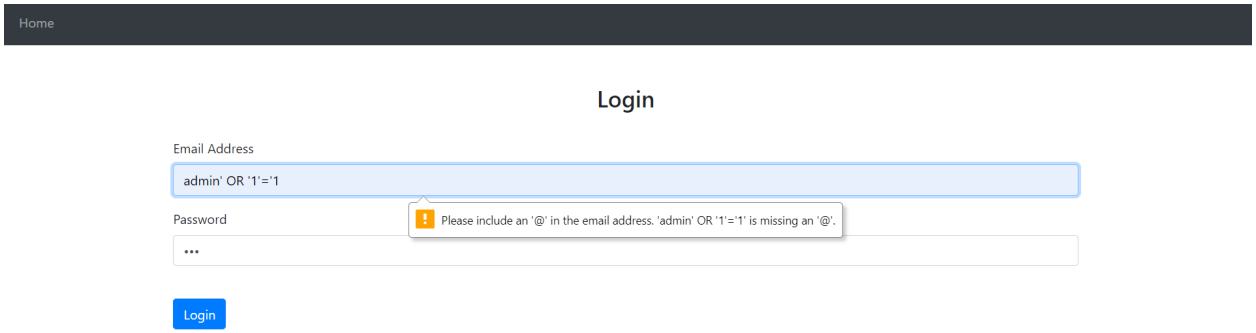


Prevention against attack: There are several ways to prevent a website from SQL injection attacks. We have used SQLAlchemy, a secured ORM, to interact with our database to defend against attacks.

Working Demo #1: After replacing our SQL query-based code with SQLLAchemy-based code. We were able to defend ourselves against the same vulnerability. As we enter the same text in the email address section, it gives an error message- 'Email does not exist.'



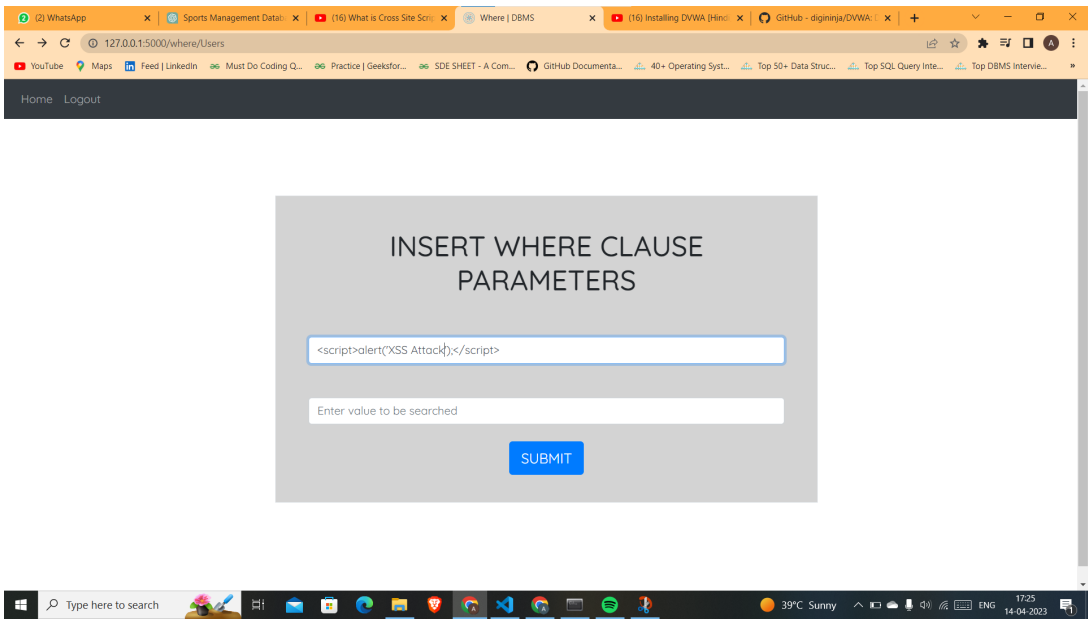
Working Demo #2: To ensure that the email address field will only take inputs in the form of an email only, we have also mentioned the input type= email in the HTML file created for the login page. In this case, it gives an error message- 'Please include an '@' in the email address. 'Admin' or '1'='1' is missing an '@'.'



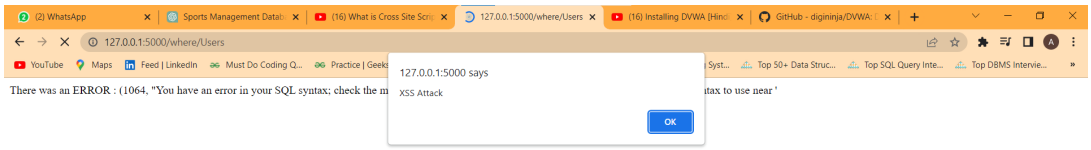
<u>ATTACK 02</u>: XSS (Cross-Site Scripting) [Non-reflected XSS Attack]

To perform an XSS attack

1. <script>alert("Attacked");</script>
2. <script>alert(document.cookie);</script>
3. <scr<script>ipt>alert("Attacked");<script>
4. <img src=x onMouseOver=alert("Hello")>



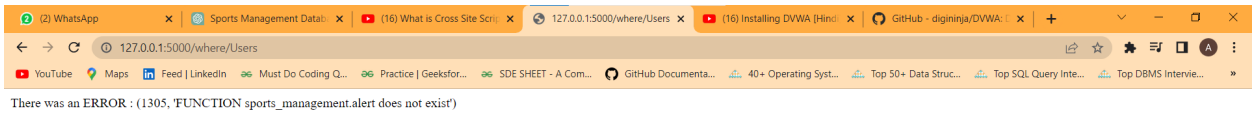
An alert message will pop up if the site is vulnerable to an XSS attack.



To prevent an XSS attack, we need to sanitise the user input, and for the same, we can use regular expressions.

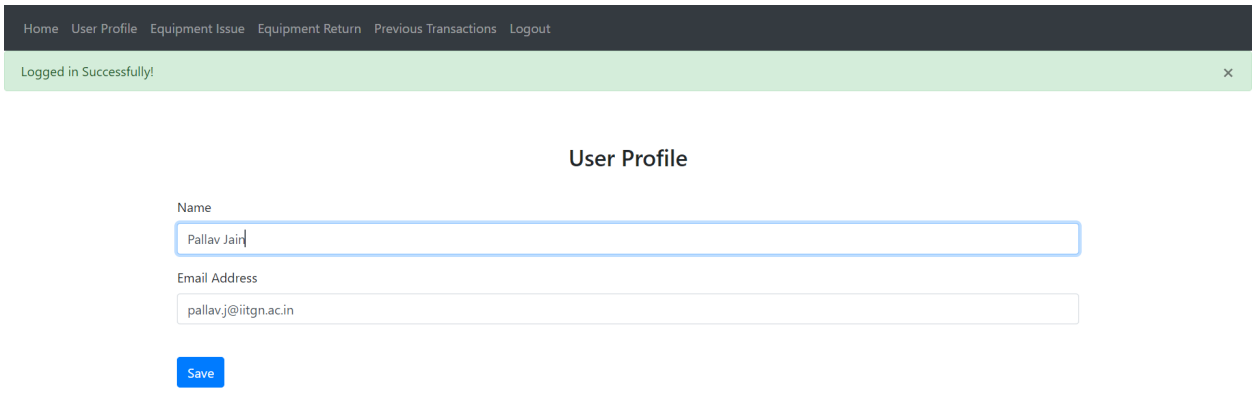
```
def where(key):
 print(key)
 msg = ''
 try:
  if request.method == 'POST':
    column_name = request.form['Column_Name']
    column_name = re.sub('<[^>]*>', '', column_name)
    print(column_name)
    value = request.form['Value']
    value = re.sub('<[^>]*>', '', column_name)
    print(value)
```

No message appeared since the user input was sanitised with the regular expression script.

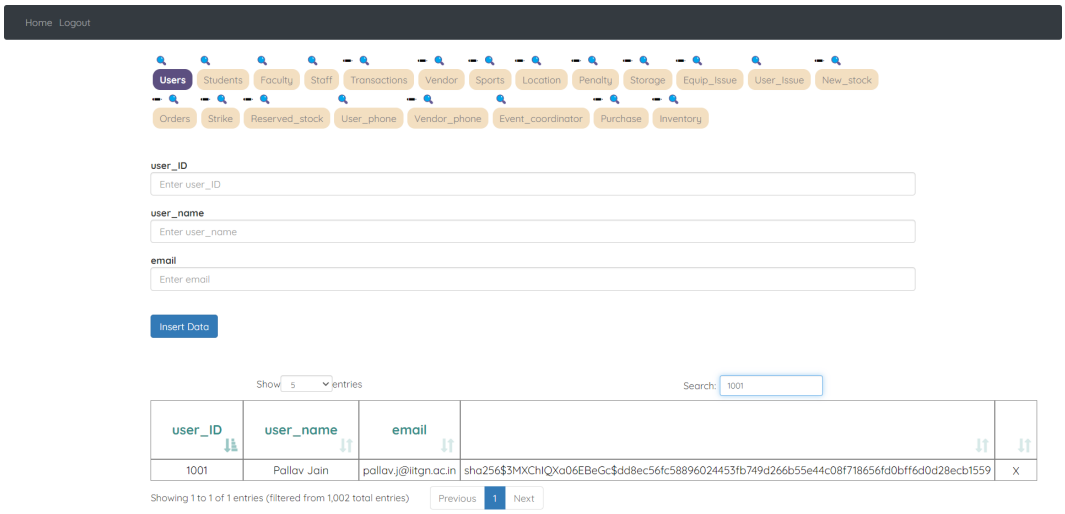


Q2.

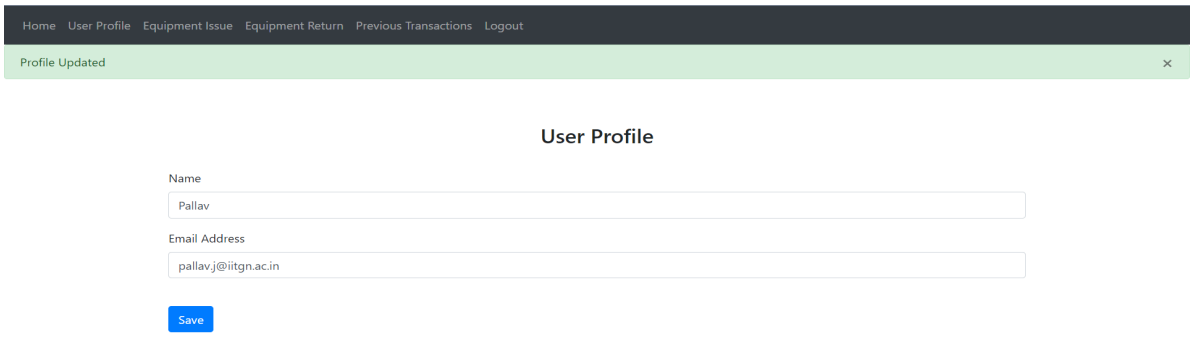
User can update their profile:



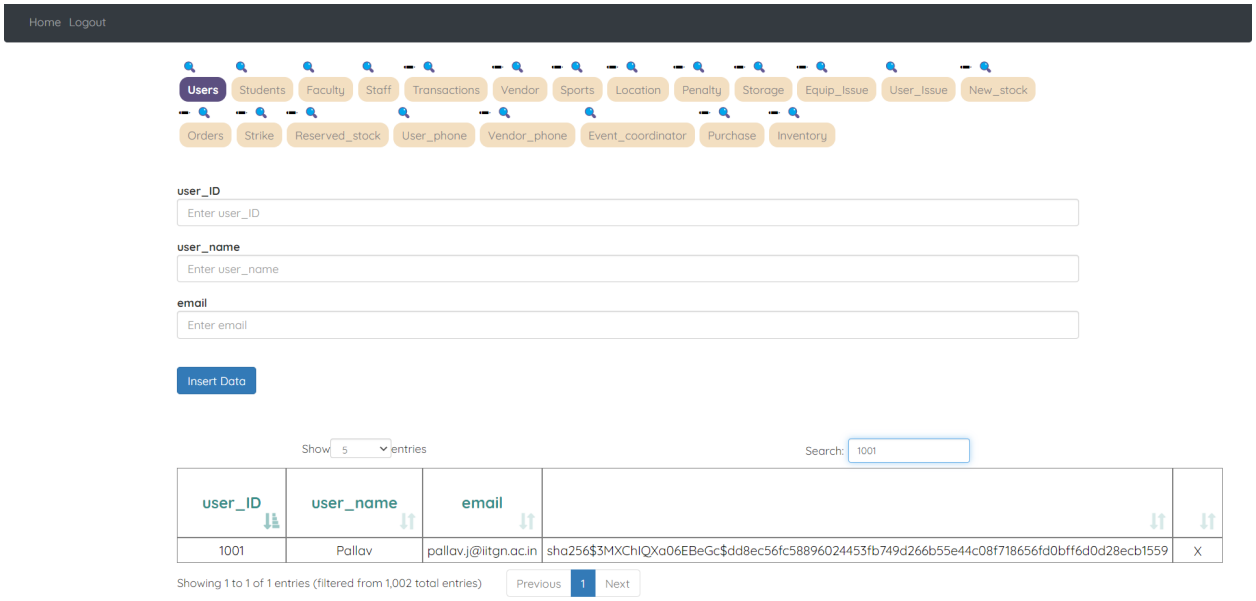
User Profile Page



User Original Data

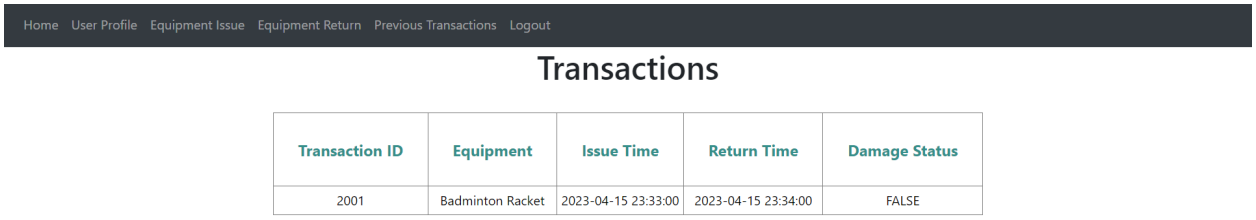


User updating his name

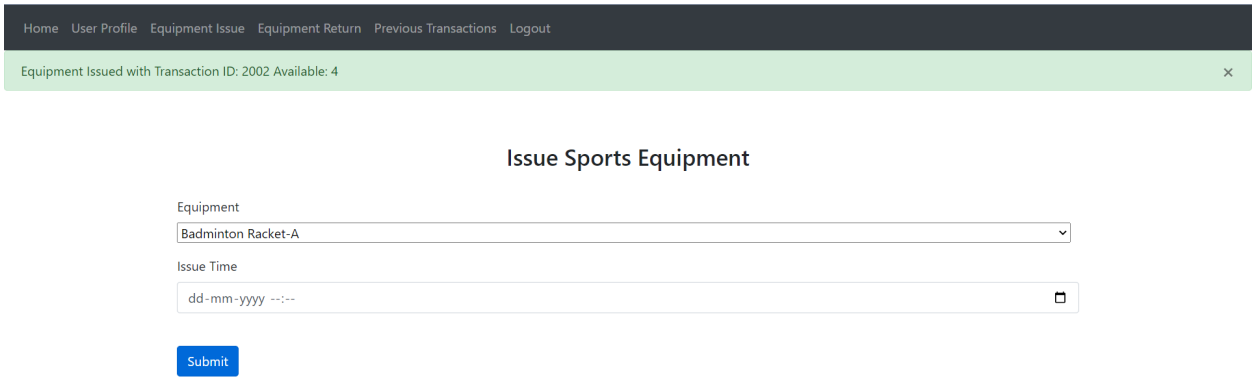


Change reflected in users table

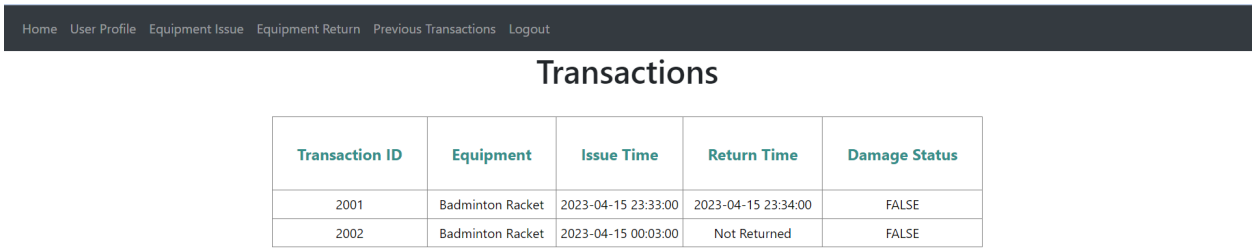
Users can view their Previous Transactions



Users can issue equipment

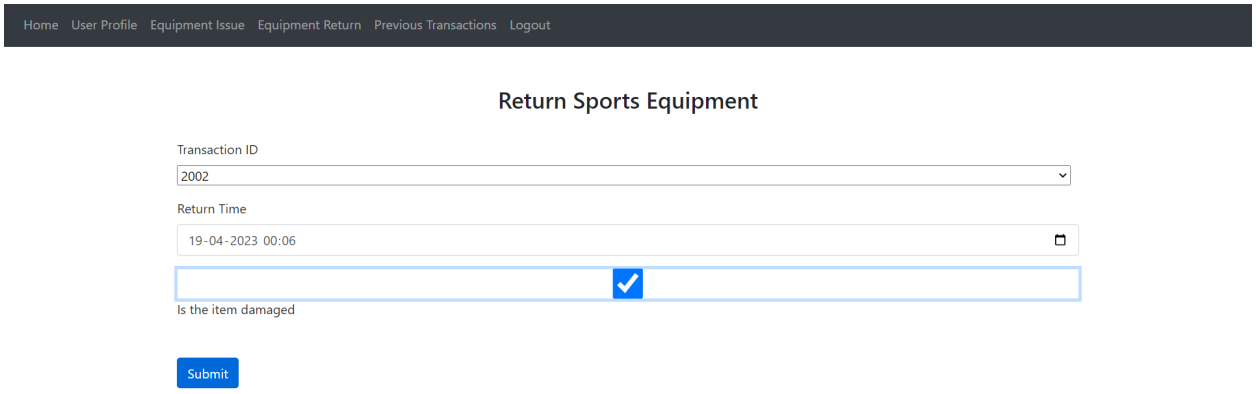


User issuing a Badminton Racket

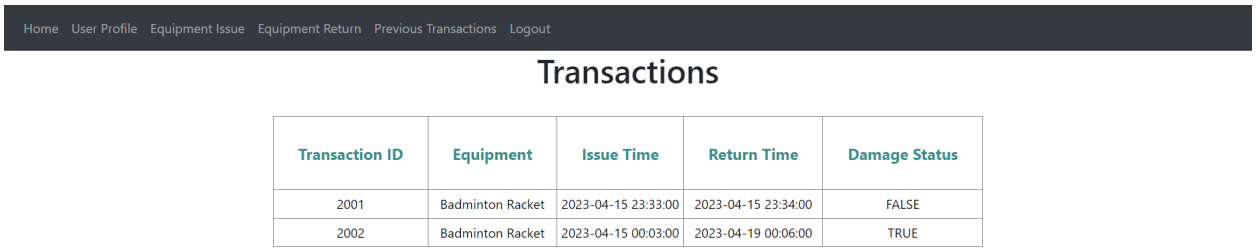


Users can see their transactions

Users can Return Equipment (Under the supervision of an officer/admin)

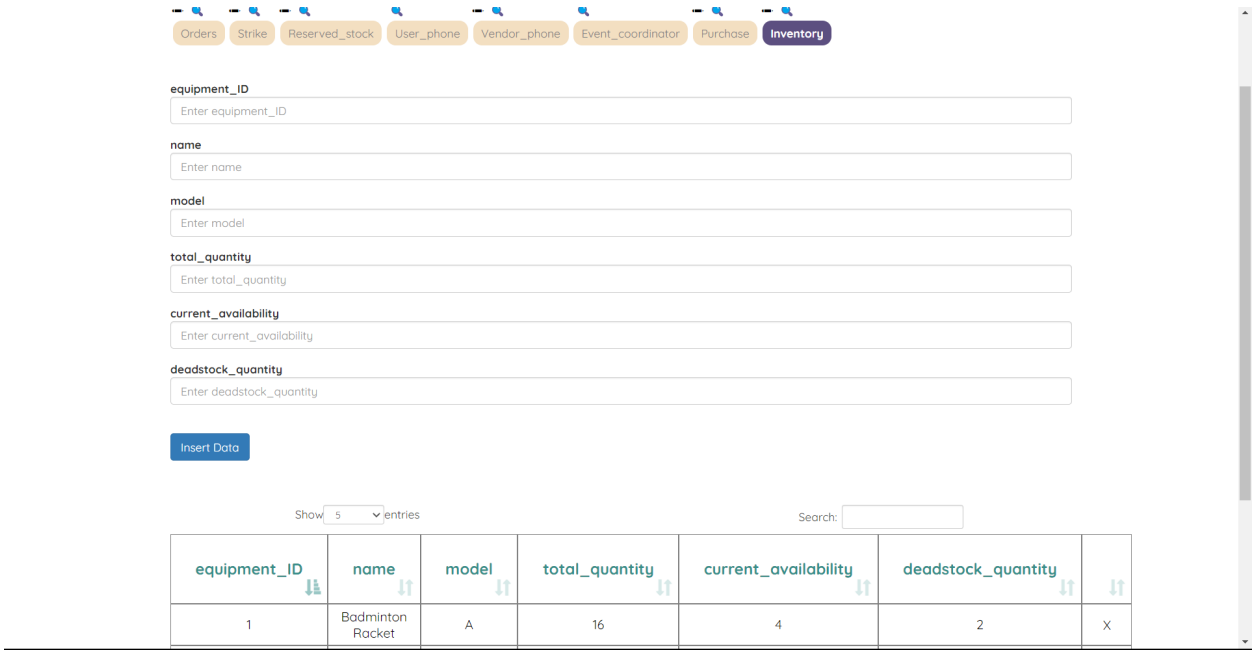


User returning the equipment

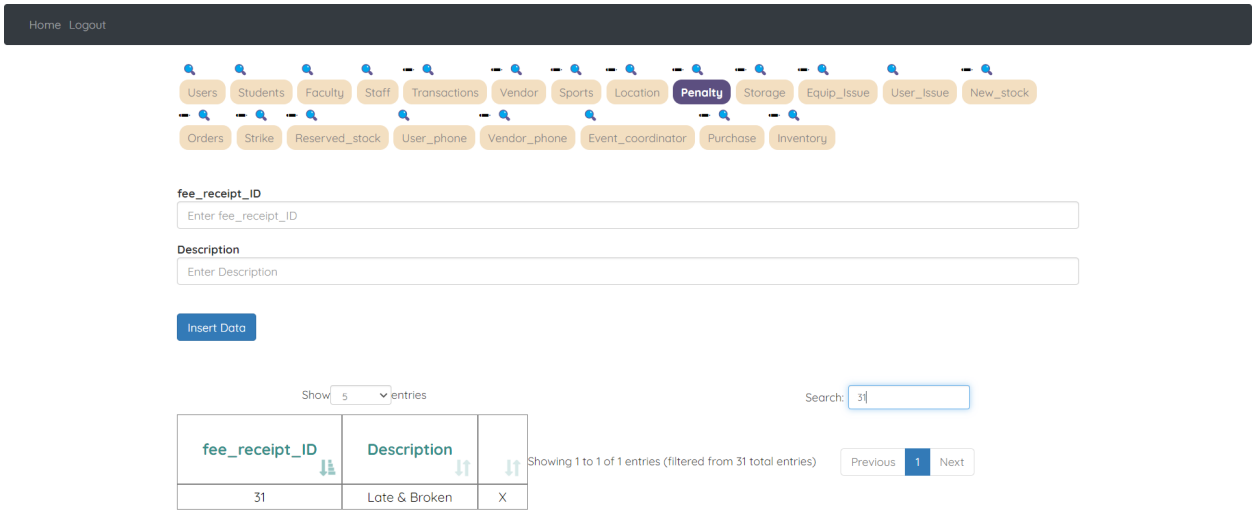


Transactions table updated after the user returned the equipment

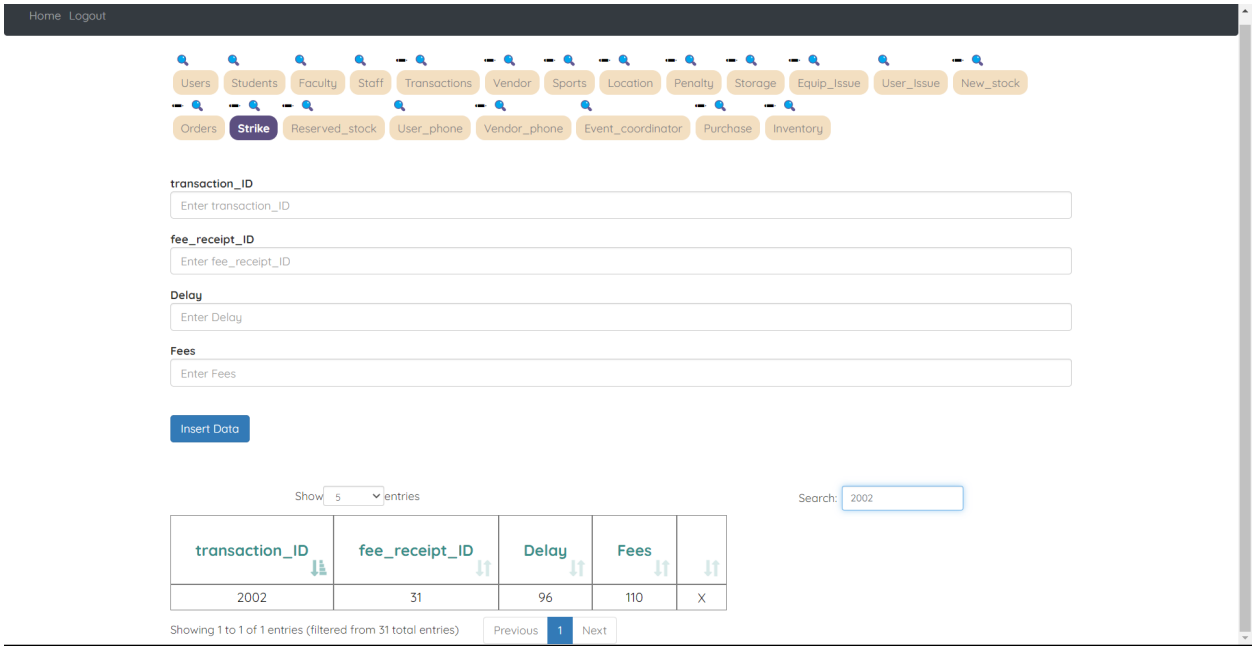
The damage status is also reflected in other tables



Inventory table

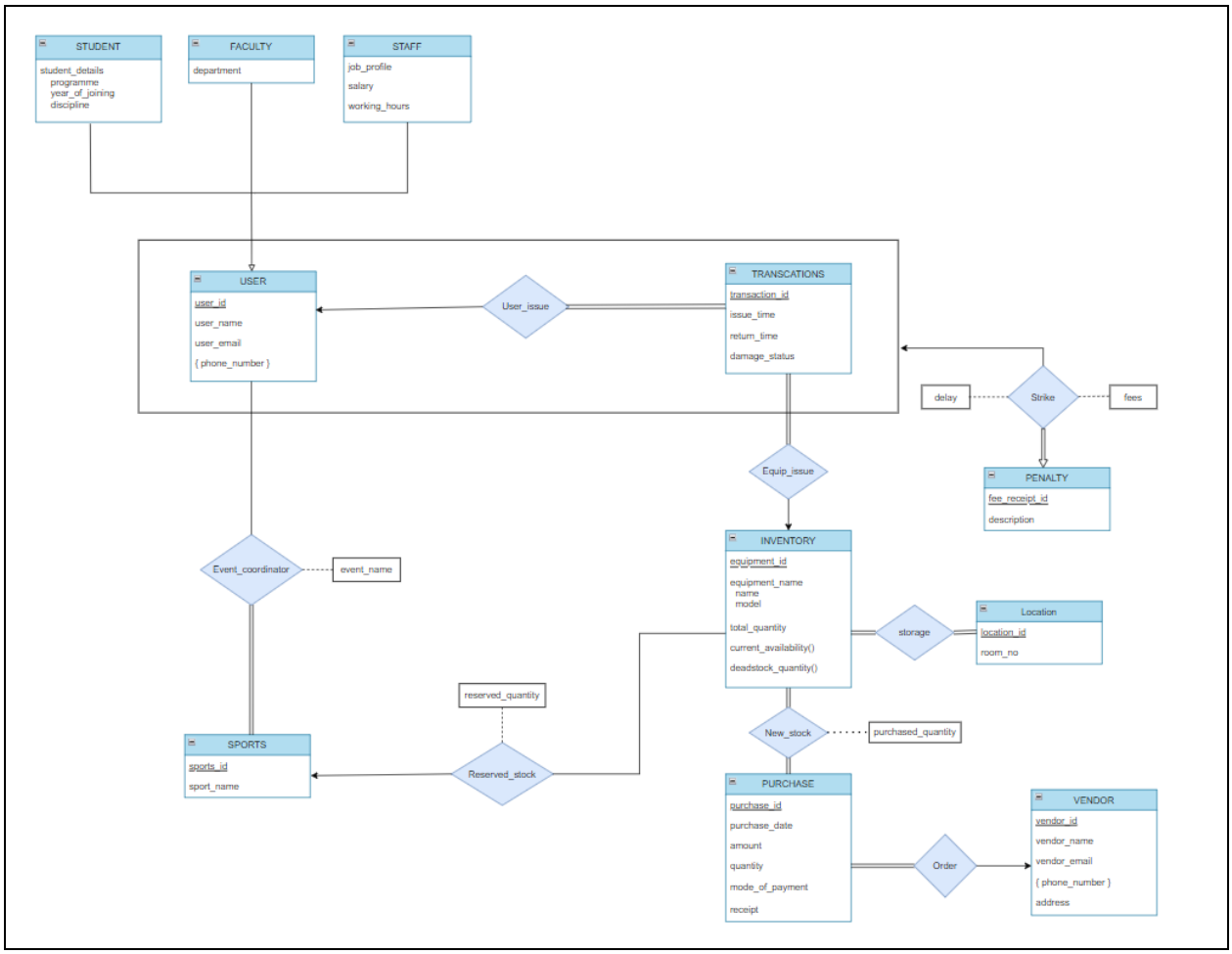


Penalty Table



Strike Table

ER Diagram is attached hereby for reference:



Contribution

Sr. No.	Name	Contribution
1	Pallav Jain	G1(q1,q2), G1&G2(q1,q2)
2	Abhishek Yawalkar	G2(q1,q2), G1&G2(q1,q2)
3	Reuben Devanesan	G1&G2(q1,q2)
4	Achal Kanojia	G2(q1,q2)
5	Sai Yashverdhan	G2(q1,q2)
6	Meet Thummar	G1(q1,q2)
7	Himanshu Singhal	G2(q1,q2)
8	Ishan Prayagi	G1(q1,q2)
9	Anushka Niti	G2(q1,q2)
10	Sresth Tosniwal	G2(q1,q2)
11	Aditya	-
12	Prashant	-