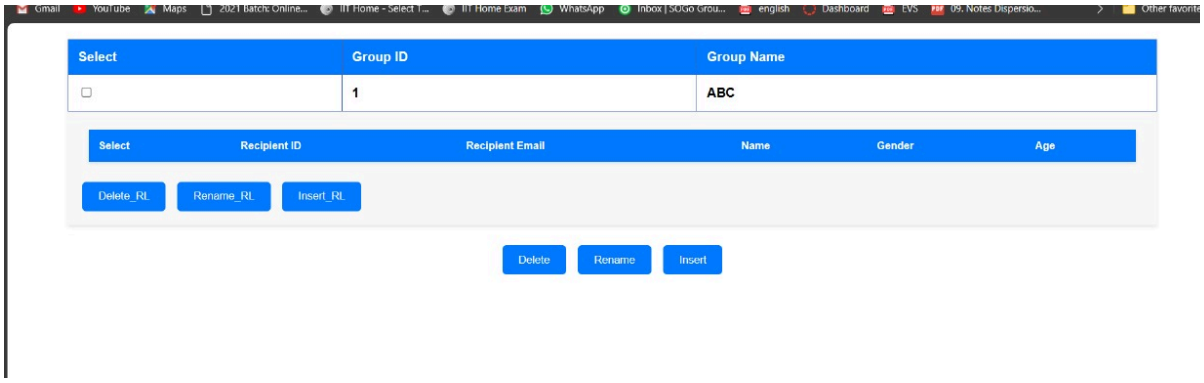
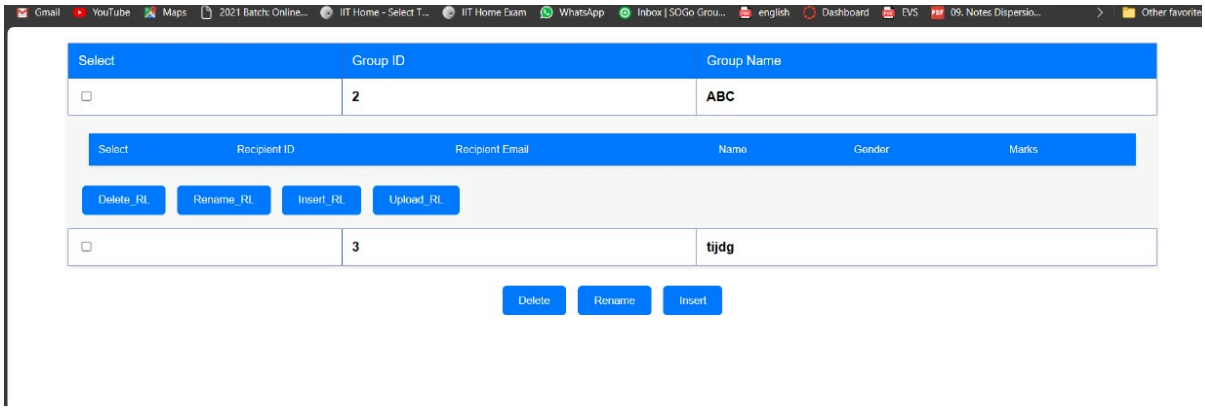
Assignment 4

3.1 Responsibility of G1:

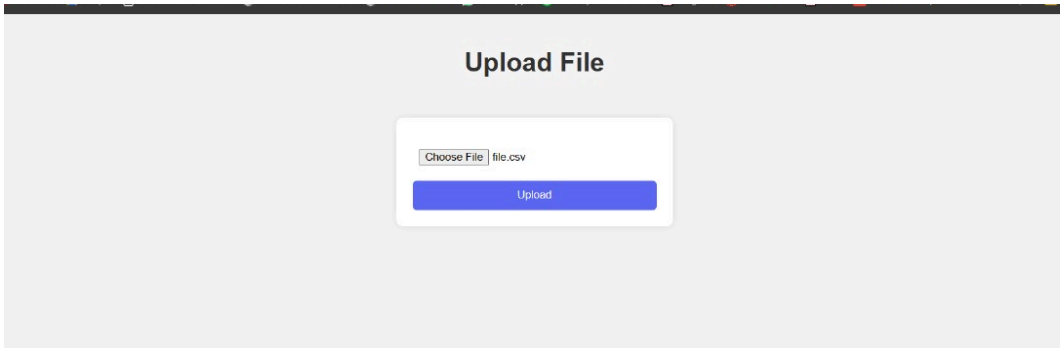
1) Changes according to feedback received.
   Feedback 1:
   1. Add an option to upload the csv file of recipients so that large recipients data can be inserted.
      In Groups page
      Before:



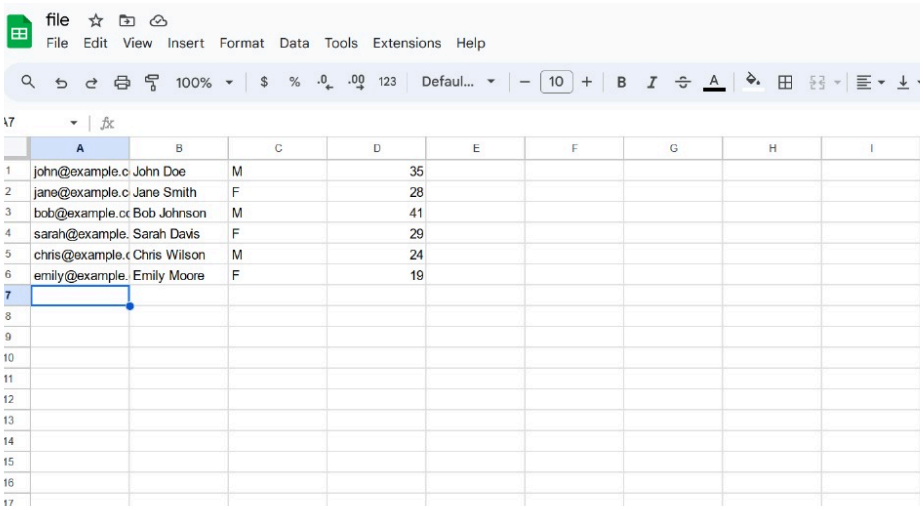
After:
Select the group where you wanted to add, Click the button Upload_RL then you can upload a file (.csv) containing recipient list.



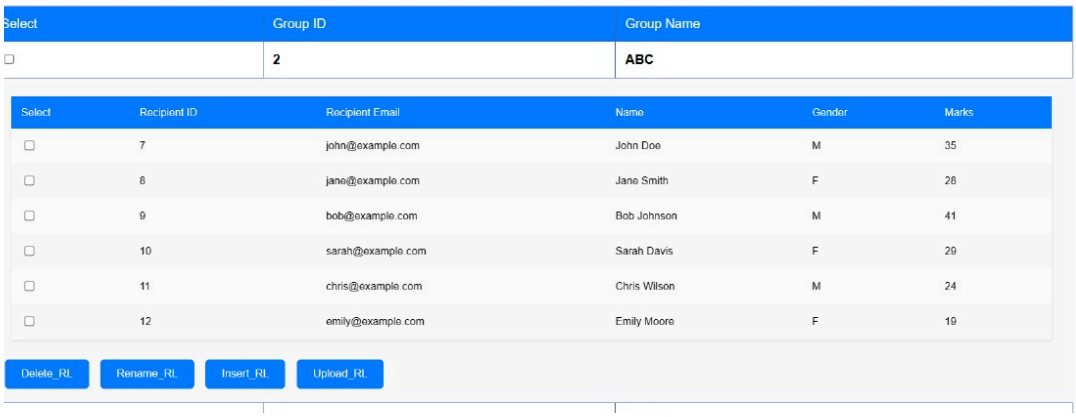
Here you can upload file



The csv file is as follows:

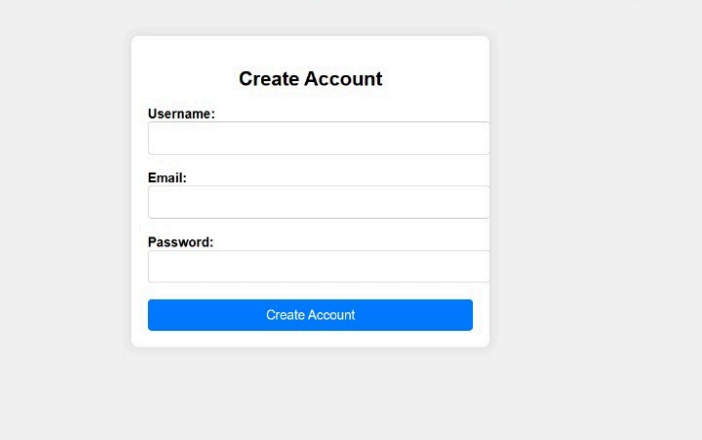


Now when you see in Groups all these recipient list will be inserts and now can you can take this group and send the email at once for all the recipients.

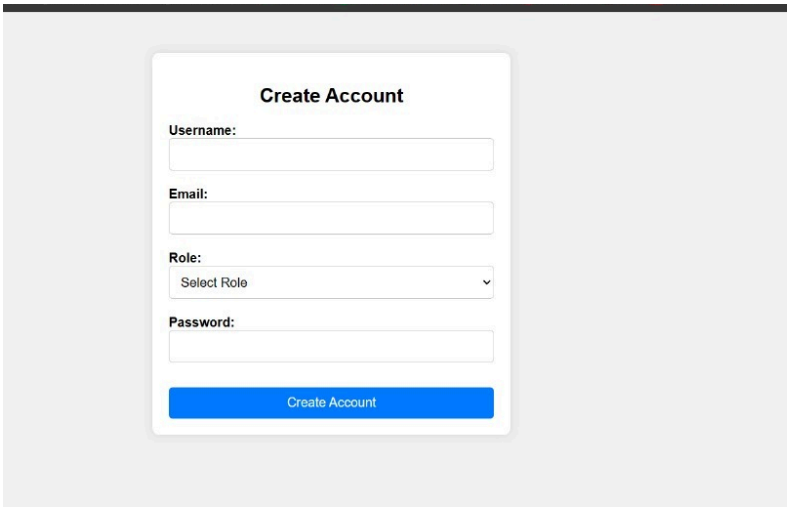


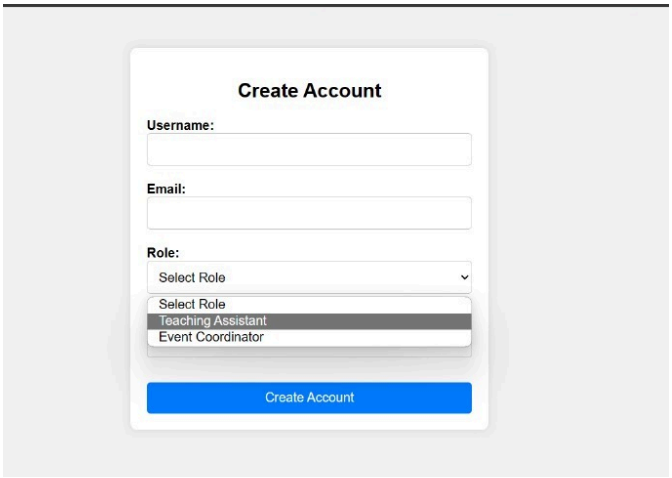
2. Better to have the option to choose from two different roles of users instead of sticking to one role.

Before:



After:



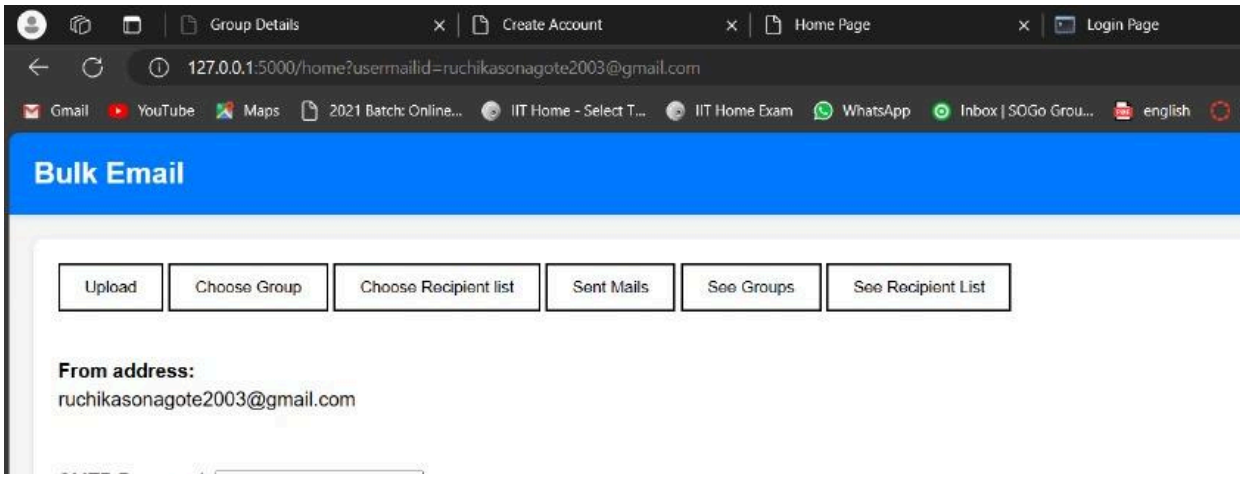


You can see that there is dropdown box of role where you can create account as per your role and here the roles can be either Teaching Assistant (to send respective marks

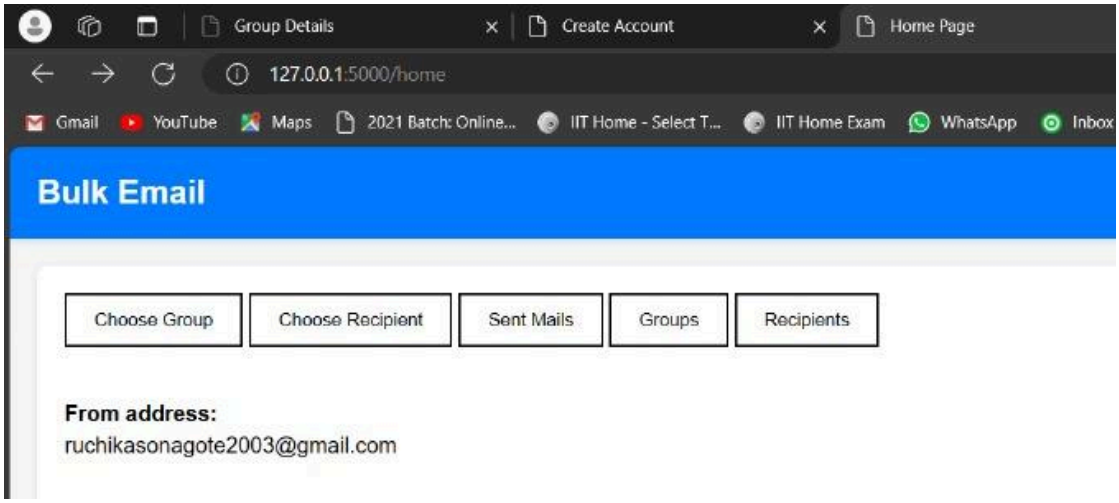
of all recipients) or Event Coordinator (to send request for funds for organizing event to all companies at once)

3. In the user_interface, change the names of the buttons of seeGroup and seeRecipientList.

Before:



After:



You can see that names of buttons of seGroups buttons and seeRecipientList are changed to Groups, Recipients.

Feedback2:

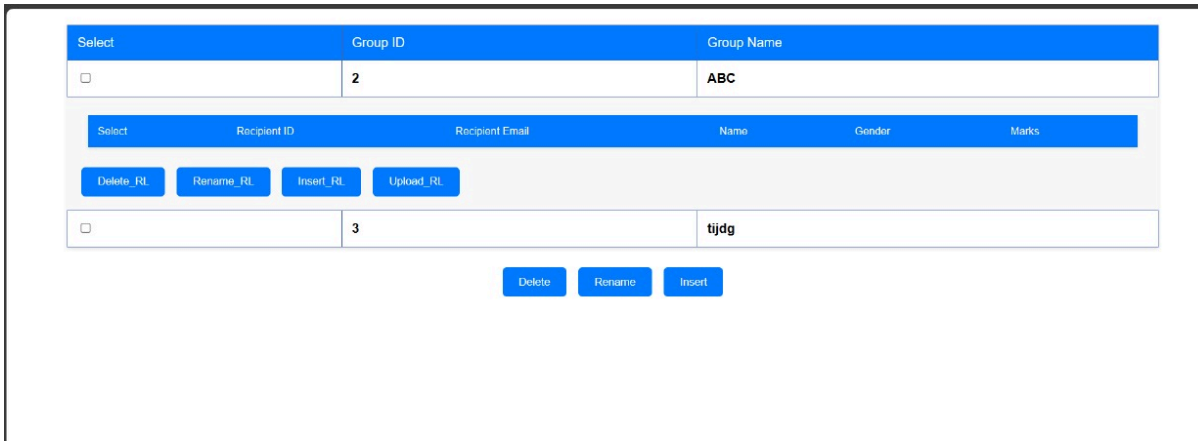
All the suggested changes were made and said the website looks fine and no further changes needed.

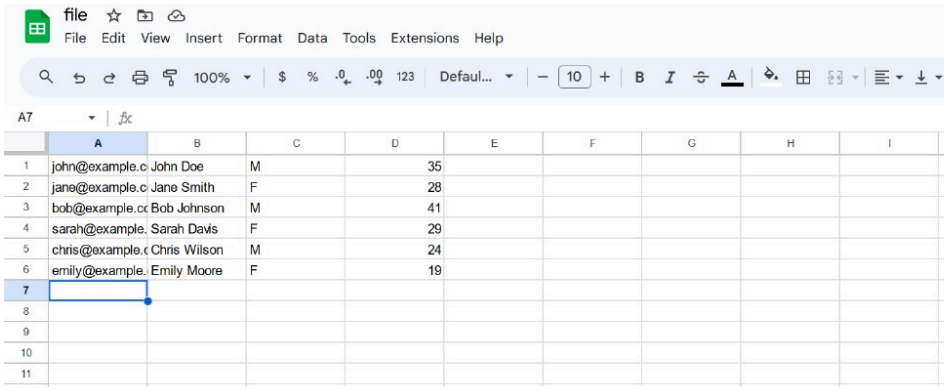
2) There are two types of class of users on our website. One is a Teaching Assistant and the other is Event Coordinator. So in our RecipientList we have columns of their id, name, gender, company and marks. But there is no permission for the Teaching Assistant to have access to the column of company similarly Event Coordinator should not access or view the column of marks. For this we will be creating views for different

roles in such a way that all columns of the recipient list will be displayed for Teaching Assistant except company similarly for Event Coordinator Marks will not be displayed and will be displayed.

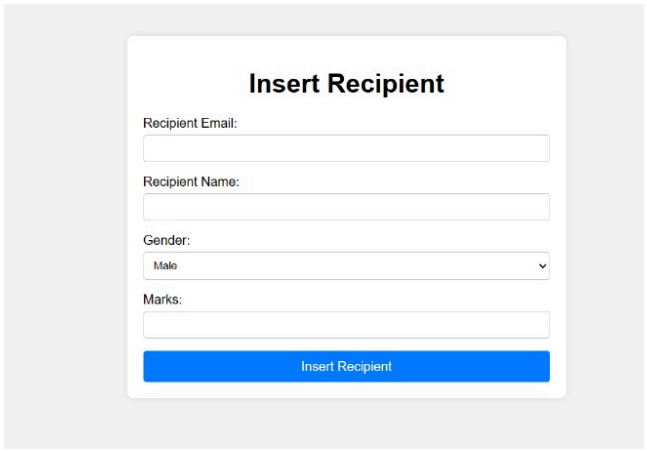
<u>In Groups page:</u>

View of Teaching Assistant:



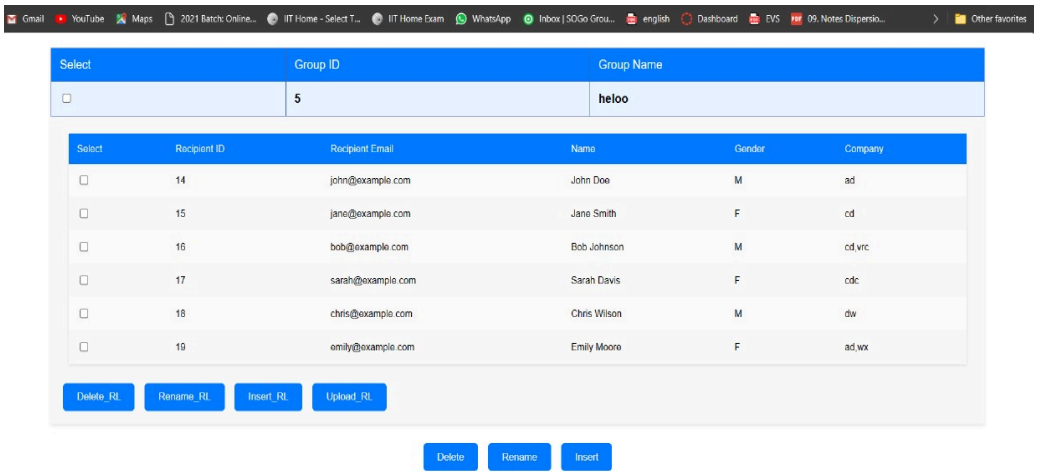


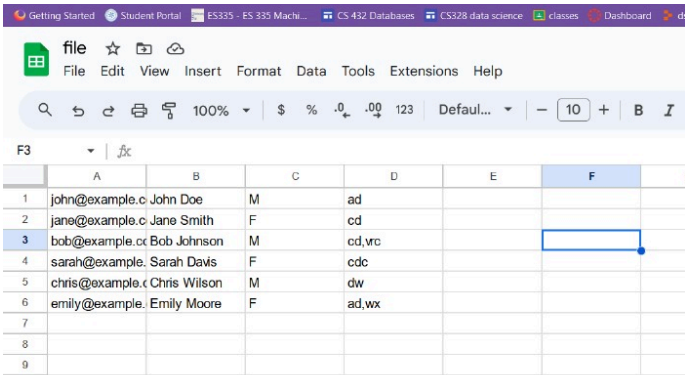
The csv file for that view will be like this.

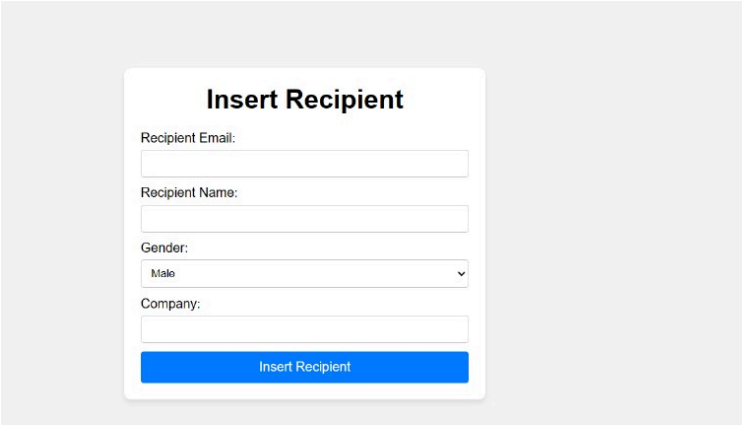


In the above view you can see that if you login as a Teaching Assistant then you will not be able to see the Column Company and cannot insert the column Company.

View of Event Coordinator:



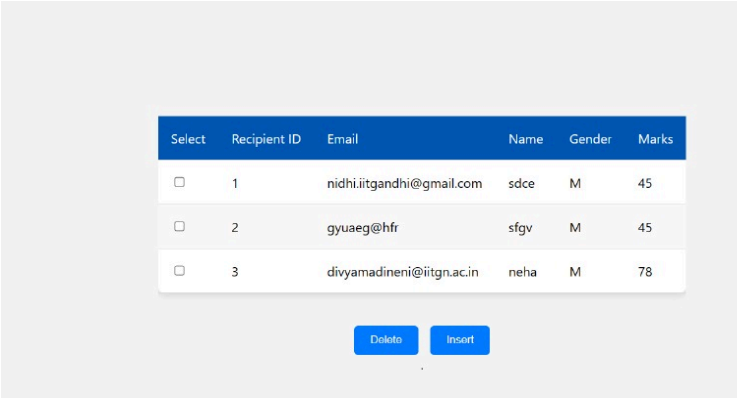




Similarly here you can see that the column ‘Marks’ for Event Coordinator. And in the csv file also you can see how it should be uploaded.
Even when you upload a csv file then you can just add your corresponding columns and upload.

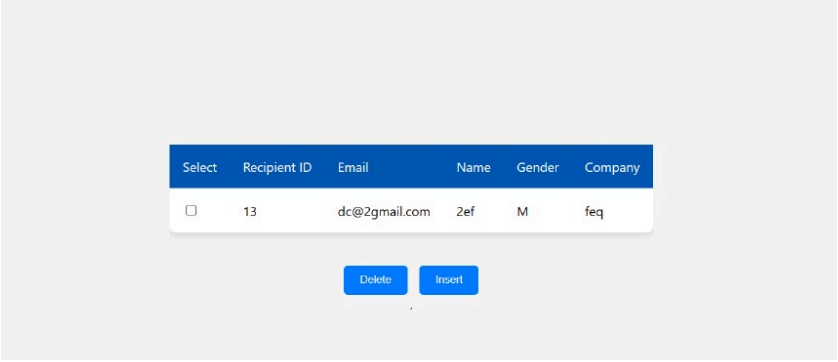
In Recipients Page:

View of Teaching Assistant:



You can see that Company cannot be accessed by this user and cannot be inserted. As shown above in groups.

View of Event Coordinator:



Similarly Marks column cannot be accessed by this user and cannot be inserted as shown above in groups.

3.2 Responsibility of G2:

1) We have added ‘lock_table’ and ‘unlock_table’ functions which will lock the tables before executing the relevant query and unlock it after performing the operation.
   In case of login or registration, as multiple users can login or register at the same time, we can use the above two functions. Here one transaction will take place at a time.
   Following images show the code for the same.

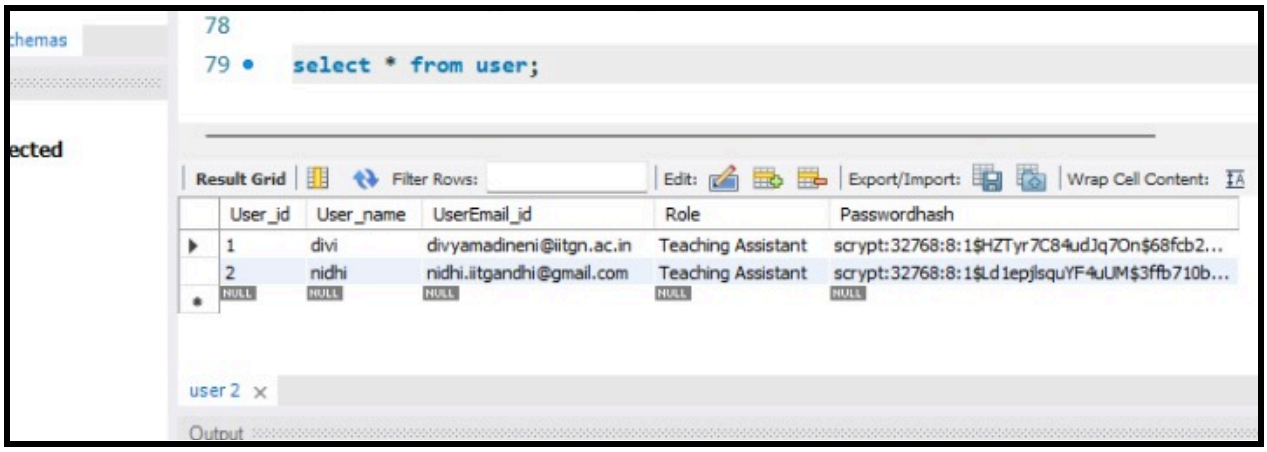
```python
def lock_table(): #Rutuja
    cur = mysql.connection.cursor()
    cur.execute("LOCK TABLES User WRITE")
    return cur

def unlock_table(cur): #rutuja
    cur.execute("UNLOCK TABLES")
    cur.close()
```

```python
@app.route('/', methods=['GET', 'POST'])
def login():
    error = None
    usermailid=None
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        # cur = mysql.connection.cursor()
        cur = lock_table() #Rutuja
        cur.execute("SELECT * FROM User WHERE UserEmail_id=%s", [email])
        User = cur.fetchone()
        if User and check_password_hash(User[4], password):
            session['user_id'] = User[0]
            usermailid=User[2]
            session['usermailid'] = usermailid
            session.pop('selected_groups', None)
            session.pop('selected_recipients', None)
            unlock_table(cur) #Rutuja
            return redirect(url_for('home',usermailid=usermailid))
```

Same function is also used in registration, send_emails and other routes. In case of sending the emails, the email table is being updated. When two users try to send the email at the same time, one email will be sent at a time and after completing this transaction another email will be sent.

2) All the suggested changes were made as per the feedback received and can be seen in 3.1.1
   The modified User table after adding the ‘role’ column.



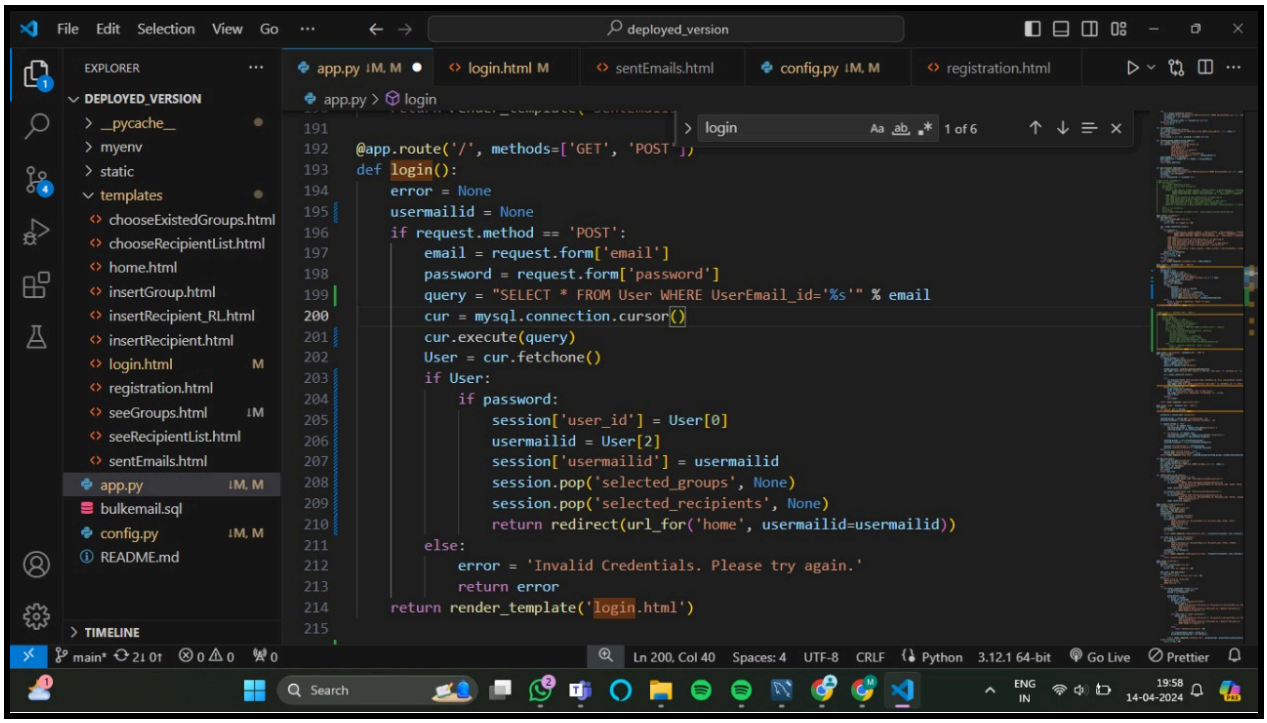
‘

The modified RecipientList table after adding ‘marks’ attribute to the table.

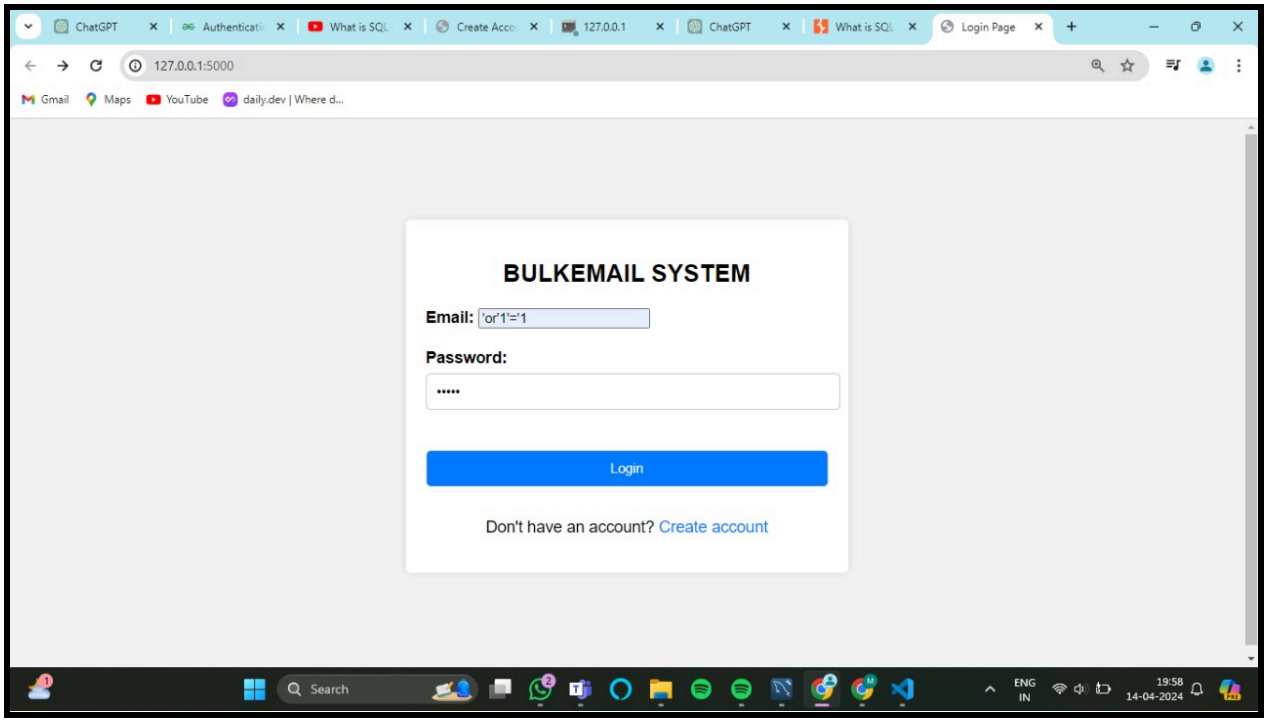
Recipient_id	RecipientEmail_id	User_id	Recipient_name	Gender	Company	Marks
1	nidhi.iitgandhi@gmail.com	1	sdce	M	NULL	45
2	gyuaeg@hfr	1	sfgv	M	NULL	45
3	divyamadineni@iitgn.ac.in	1	neha	M	NULL	78
5	kernjdgz@dfgd	2	fhrth	M	srth	NULL
6	awdQ@FG	2	se	M	er	NULL
13	dc@2gmail.com	4	2ef	M	feq	NULL
14	john@example.com	4	John Doe	M	ad	NULL
15	jane@example.com	4	Jane Smith	F	cd	NULL
16	bob@example.com	4	Bob Johnson	M	cd,vrc	NULL
17	sarah@example.com	4	Sarah Davis	F	cdc	NULL
18	chris@example.com	4	Chris Wilson	M	dw	NULL
19	emily@example.com	4	Emily Moore	F	ad,wx	NULL
20	hgdde@hgfg	1	jhhfyg	M	NULL	86
NULL	NULL	NULL	NULL	NULL	NULL	NULL

3.3 Responsibility of G1 & G2:

1. **SQL Injection**

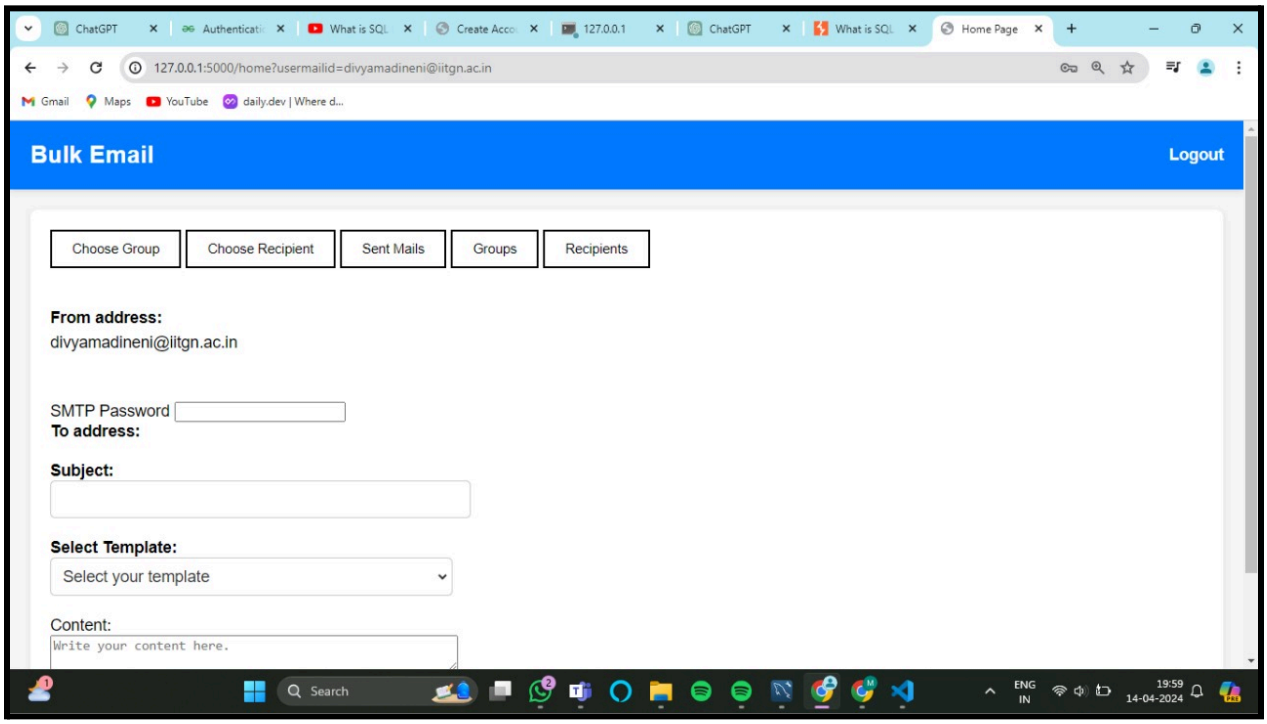


This is our backend code handling the login of the user who has already created account.

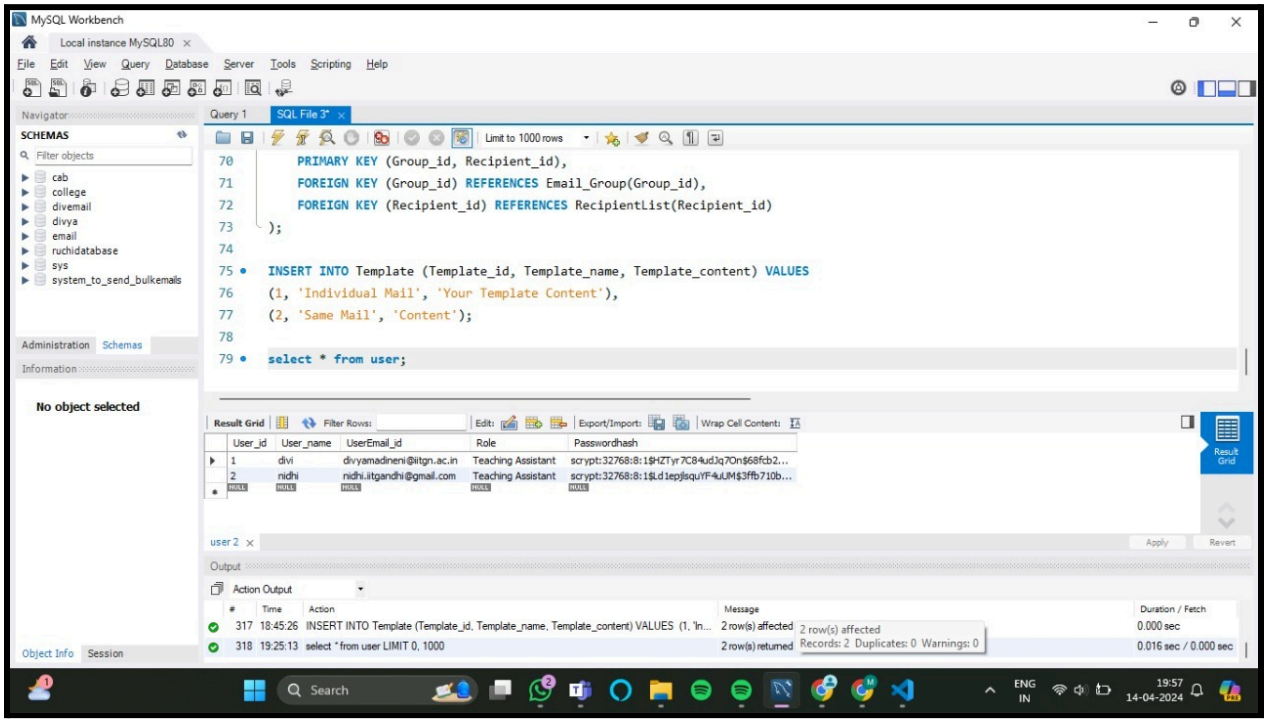


When we go to the login page, we write the "*'or'1'='1*" in the input email field. In the password field, we can write anything. After clicking on the login button, it redirects to the home page while the email_id and password is not correct. This query works because when this query execute, it becomes the part of the SQL command: Select * from User Where UserEmail_id = ' or '1'='1'. In SQL, '1' = '1' is always true, so this query will select all rows of the 'User' table regardless of the value of the UserEmail_id. As a

result, this query will bypass the password check and retrieves all row of the ‘User’ table, allowing unauthorised access.



After this unauthorised login, the application retrieves the first user from the result set, which results, the attacker will redirect to the home page with user_id = 1.



Here we can see that user_id of the "divyamadineni@iitgn.ac.in" is 1 and the attacker after unautorised access has redirected to the homepage with User_id = 1.

For defending the SQL injection:

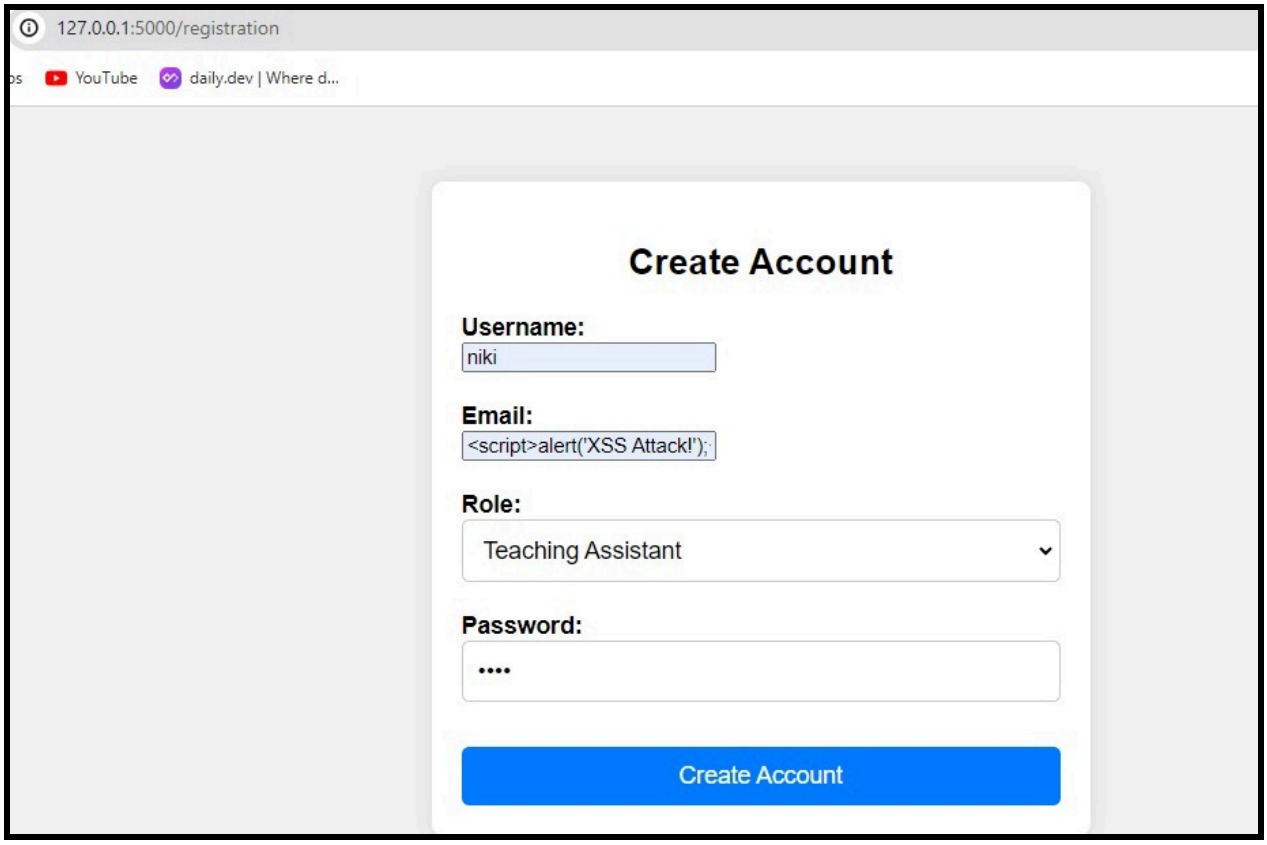
```python
@app.route('/', methods=['GET', 'POST'])
def login():
    error = None
    usermailid=None
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        # cur = mysql.connection.cursor()
        cur = lock_table()
        cur.execute("SELECT * FROM User WHERE UserEmail_id=%s", [email])
        User = cur.fetchone()
        if User and check_password_hash(User[4], password):
            session['user_id'] = User[0]
            usermailid=User[2]
            session['usermailid'] = usermailid
            session.pop('selected_groups', None)
            session.pop('selected_recipients', None)
            unlock_table(cur)
            return redirect(url_for('home',usermailid=usermailid))
        else:
            unlock_table(cur)
            error = 'Invalid Credentials. Please try again.'
            return escape(error)
    return render_template('login.html')
```

We did hashed the password.

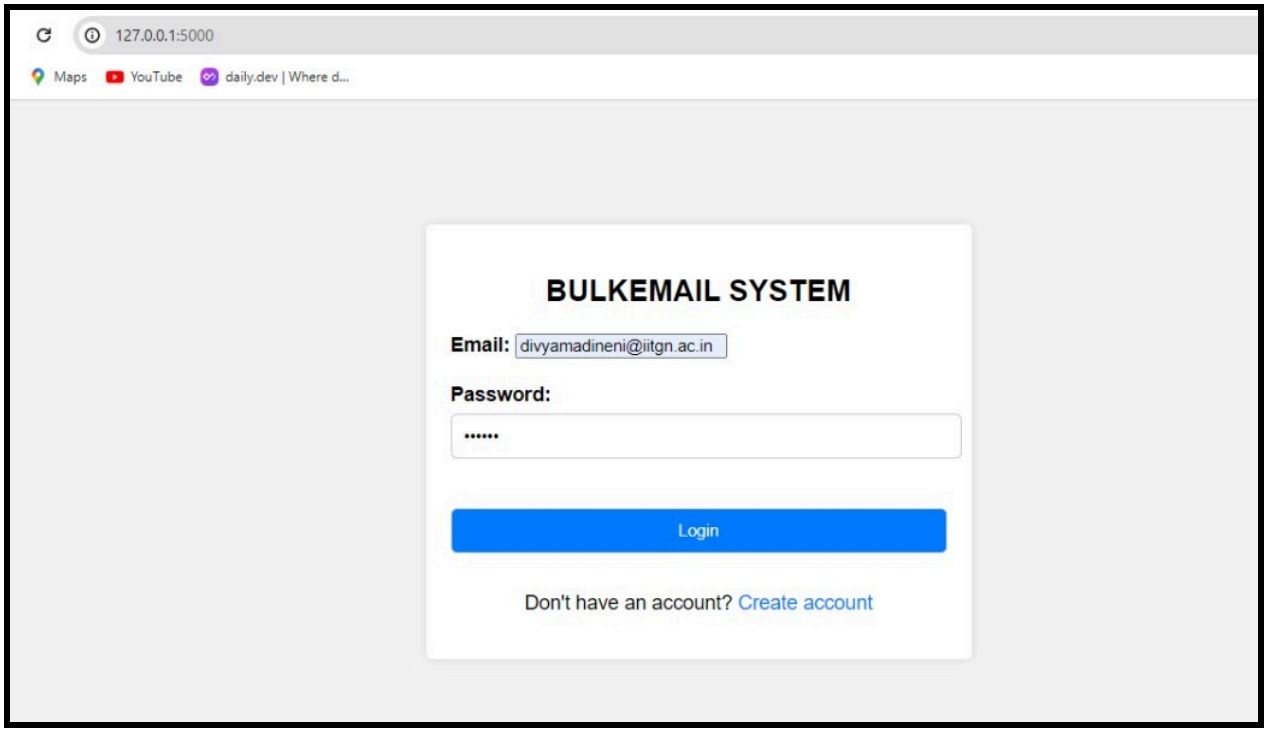
XSS

Before defend:

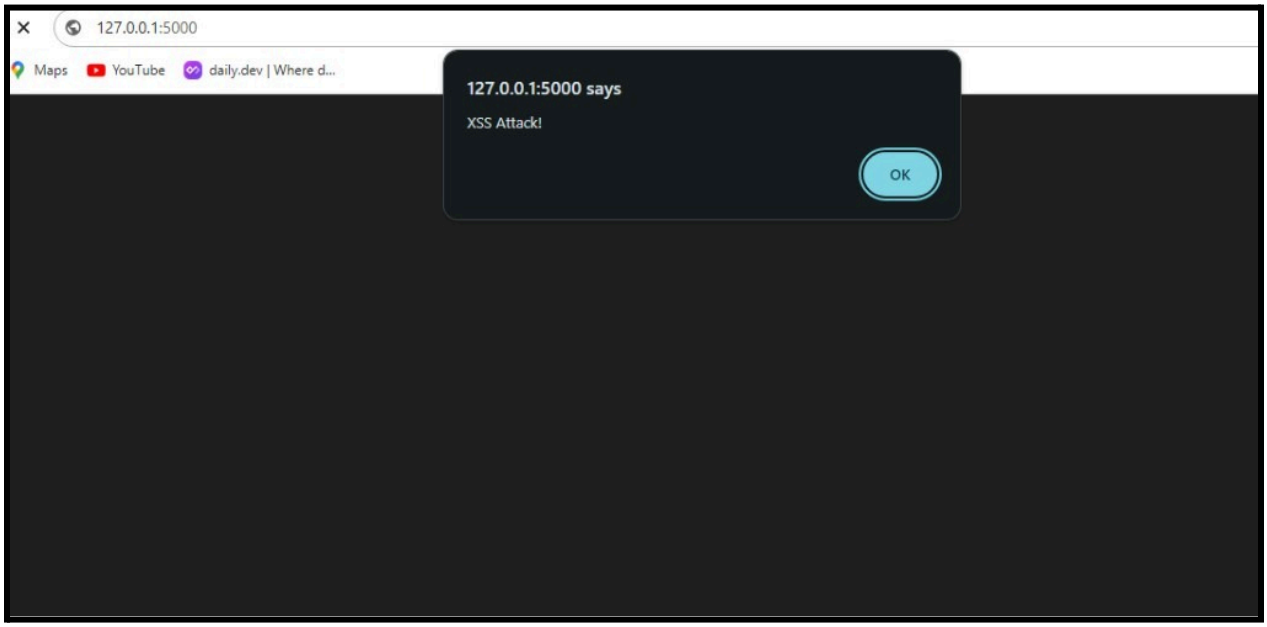
```
@app.route('/', methods=['GET', 'POST'])
def login():
    error = None
    usermailid=None
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        # cur = mysql.connection.cursor()
        cur = lock_table()
        cur.execute("SELECT * FROM User WHERE UserEmail_id=%s", [email])
        User = cur.fetchone()
        if User and check_password_hash(User[4], password):
            session['user_id'] = User[0]
            usermailid=User[2]
            session['usermailid'] = usermailid
            session.pop('selected_groups', None)
            session.pop('selected_recipients', None)
            unlock_table(cur)
            return redirect(url_for('home',usermailid=usermailid))
        else:
            unlock_table(cur)
            error = 'Invalid Credentials. Please try again.'
            return error
    return render_template('login.html')
```



We use the query ‘<script>alert('XSS Attack!');</script>@example.com’ in the email_id field. This query is an potential attack because it is using the <script>. Without proper sanitization or escaping, the input containing the `<script>` tag can be executed as JavaScript code on the web page, allowing attackers to display an alert with the message "XSS Attack!" to users visiting the page.



When any user login with their credential, then after clicking the login button it will show: XSS Attack alert.

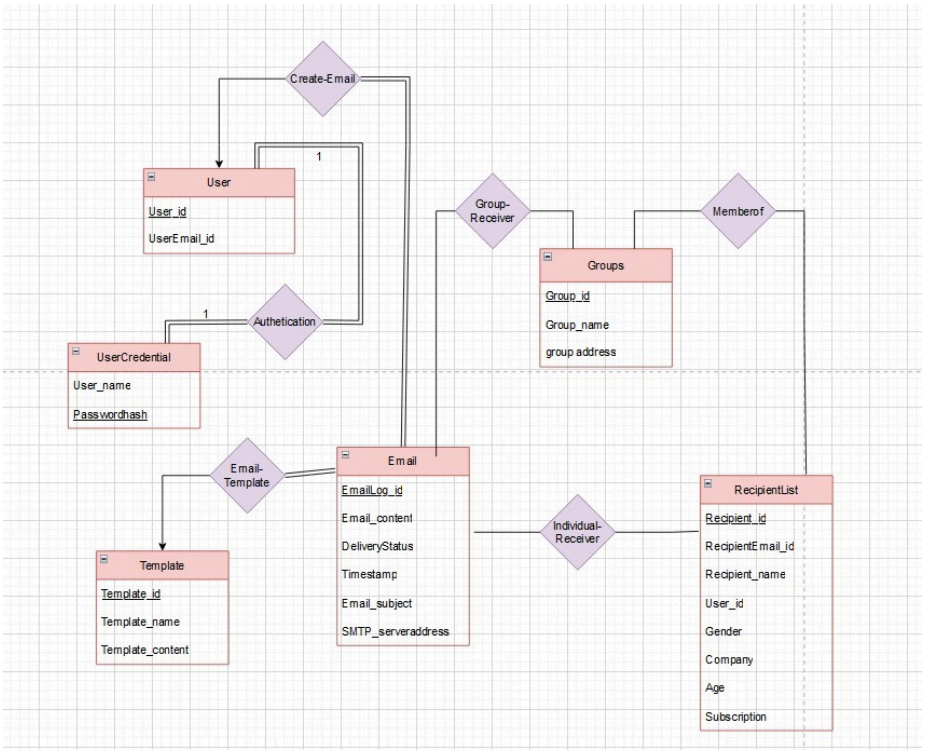


After defend:

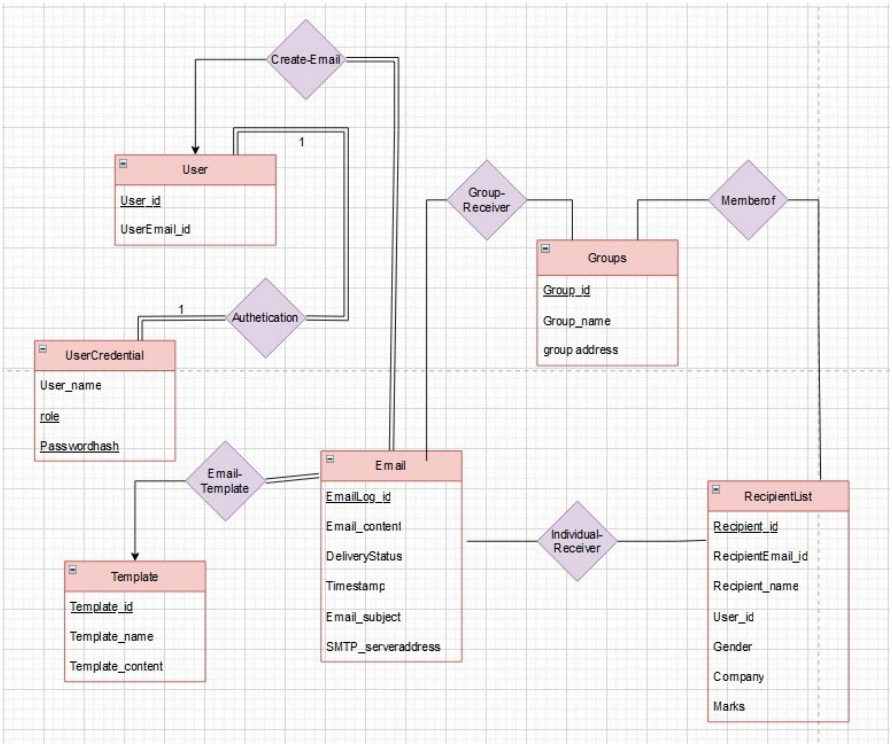
```python
@app.route('/', methods=['GET', 'POST'])
def login():
    error = None
    usermailid=None
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        # cur = mysql.connection.cursor()
        cur = lock_table()
        cur.execute("SELECT * FROM User WHERE UserEmail_id=%s", [email])
        User = cur.fetchone()
        if User and check_password_hash(User[4], password):
            session['user_id'] = User[0]
            usermailid=User[2]
            session['usermailid'] = usermailid
            session.pop('selected_groups', None)
            session.pop('selected_recipients', None)
            unlock_table(cur)
            return redirect(url_for('home',usermailid=usermailid))
        else:
            unlock_table(cur)
            error = 'Invalid Credentials. Please try again.'
            return escape(error)
    return render_template('login.html')
```

2.ER-diagram:

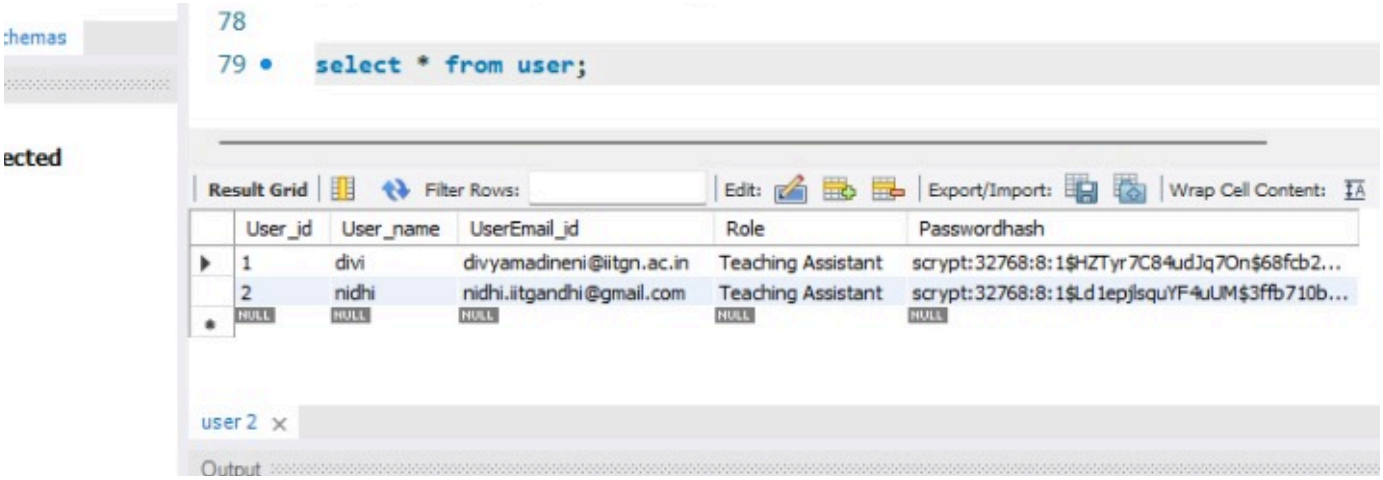
Before:


Create-Email
1
User
User_id
UserEmail_id
Group-
Receiver
Memberof
Groups
Group_id
Group_name
group address
1
Authetication
UserCredential
User_name
Passwordhash
Email-
Template
Email
EmailLog_id
Email_content
DeliveryStatus
Timestamp
Email_subject
SMTP_serveraddress
Individual-
Receiver
RecipientList
Recipient_id
RecipientEmail_id
Recipient_name
User_id
Gender
Company
Age
Subscription
Template
Template_id
Template_name
Template_content

After:


Create-Email
1
User
User_id
UserEmail_id
Group-
Receiver
Memberof
Groups
Group_id
Group_name
group address
1
Authetication
UserCredential
User_name
role
Passwordhash
Email-
Template
Email
EmailLog_id
Email_content
DeliveryStatus
Timestamp
Email_subject
SMTP_serveraddress
Individual-
Receiver
RecipientList
Recipient_id
RecipientEmail_id
Recipient_name
User_id
Gender
Company
Marks
Template
Template_id
Template_name
Template_content

You can see that User_credntial and User will be combined any way as User table as they are one to one relation. Changes in ER-diagrams are as follows:
1.Role has been added to the User table and all other constraints remain the same.



2. Dropped column Age and added marks so that marks column is used by Teaching assistant and Company can be used by Event Coordinator and removed the column subscription at the time of third phase as there is no use of having that column.

Recipient_id	RecipientEmail_id	User_id	Recipient_name	Gender	Company	Marks
1	nidhi.iitgandhi@gmail.com	1	sdce	M	NULL	45
2	gyuaeg@hfr	1	sfgv	M	NULL	45
3	divyamadineni@iitgn.ac.in	1	neha	M	NULL	78
5	kernjdgz@dfgd	2	fhrth	M	srth	NULL
6	awdQ@FG	2	se	M	er	NULL
13	dc@2gmail.com	4	2ef	M	feq	NULL
14	john@example.com	4	John Doe	M	ad	NULL
15	jane@example.com	4	Jane Smith	F	cd	NULL
16	bob@example.com	4	Bob Johnson	M	cd,vrc	NULL
17	sarah@example.com	4	Sarah Davis	F	cdc	NULL
18	chris@example.com	4	Chris Wilson	M	dw	NULL
19	emily@example.com	4	Emily Moore	F	ad,wx	NULL
20	hgdde@hgfg	1	jhhfyg	M	NULL	86
NULL	NULL	NULL	NULL	NULL	NULL	NULL

All the relations and constraints were the same as before, just the column has been deleted and added.

[**NOTE:**You have to input the SMTP server address in the home.html so that you send teh email correctly.

SMTP server address can be taken as follows:
1)Go to Manage your Google Account.
2)Go to security and click on 2-step verification and click on get started and ci=ontinue the steps to complete verification as directed.
3)Once you are done with the above search app passwords in settings and go to app passwords.
4) Now you just create a random name and generate a password, store the password somewhere with you and click on it. Remove the spaces in between the password generated and input this in the home.html SMTP address to send email.]

Contribution:

Members of G1:

- Yalamanchili Pavithra
- Sunil Jatav
- Madineni Divya
- Kanamarlapudi Hema

Member of G2:

- Harshal Sonawane
- Nidhi Kumari
- Rutuja Kadam
- Ruchika Sonagote

The responsibilities for group 1 and group 2 were done as given in the assignment. There is discussion among respective groups, and work is divided equally. Later, the responsibility of group 1 and group 2 was also fulfilled equally by all the group members with discussion and documentation.