Indian Institute of Technology Gandhinagar

CS432: Databases

Semester II, AY 2023-24

ASSIGNMENT 4

GROUP - DataWeavers

TOPIC: *oCEO Activities Portal Management*

Group Members

Aditya Gupte	21110012
Balgopal Moharana	21110040
Darshi Doshi	21110050
Shah Faisal Khan	21110156
Pranjal	21110160
Rohit Raj	21110179
Tanish Phopalkar	21110221
Animesh Tumne	21110227

Under the guidance of

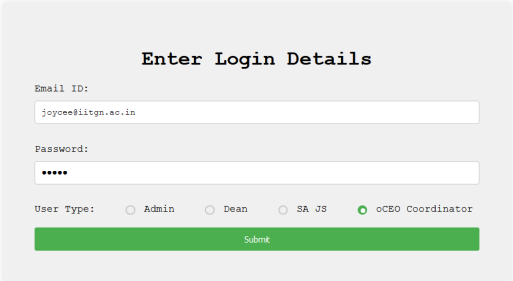
Prof. Mayank Singh

Tasks of G1:

1. Two feedbacks were taken from Mr. Sabarish Iyer from IMS services at IIT Gandhinagar. Relevant changes were made in the database and the webapp code to incorporate the same. Below shown are images of the webapp before and after both the feedbacks, which reflects the made changes.

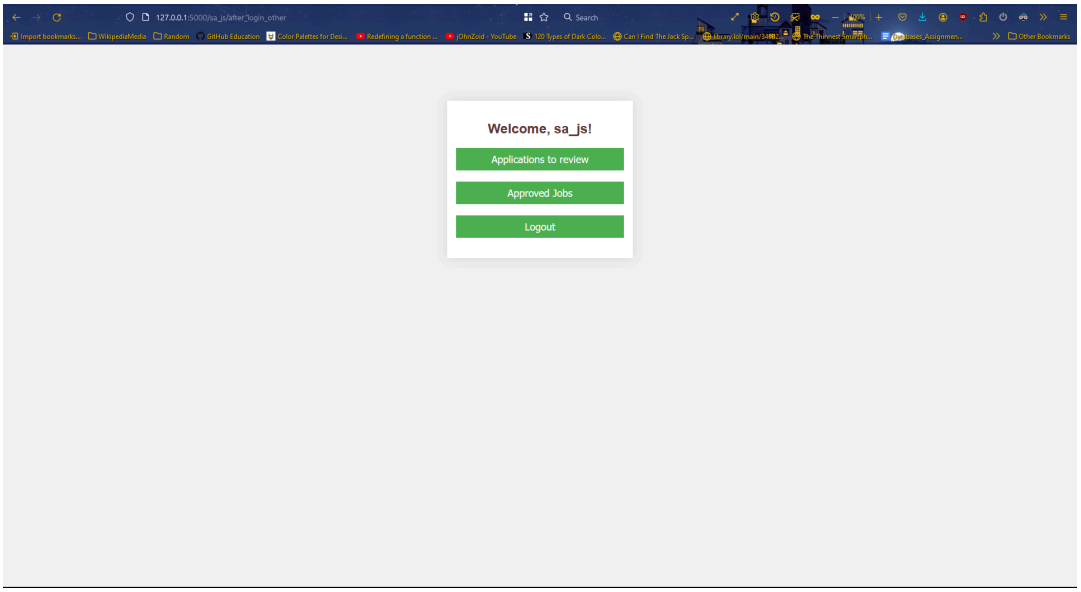
   I. Before first feedback:

      a. The oCEO coordinator is not able to login through their faculty email id:



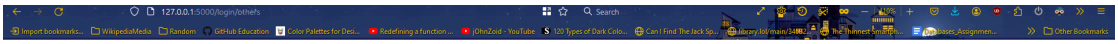
Bad Credentials!

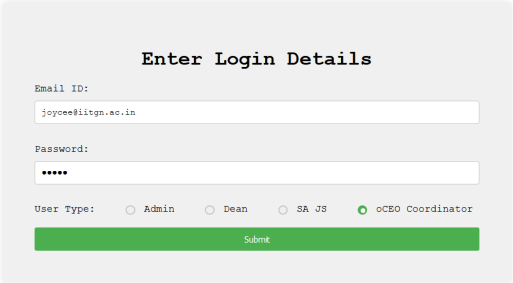
      b. SA JS can't see pending payments:

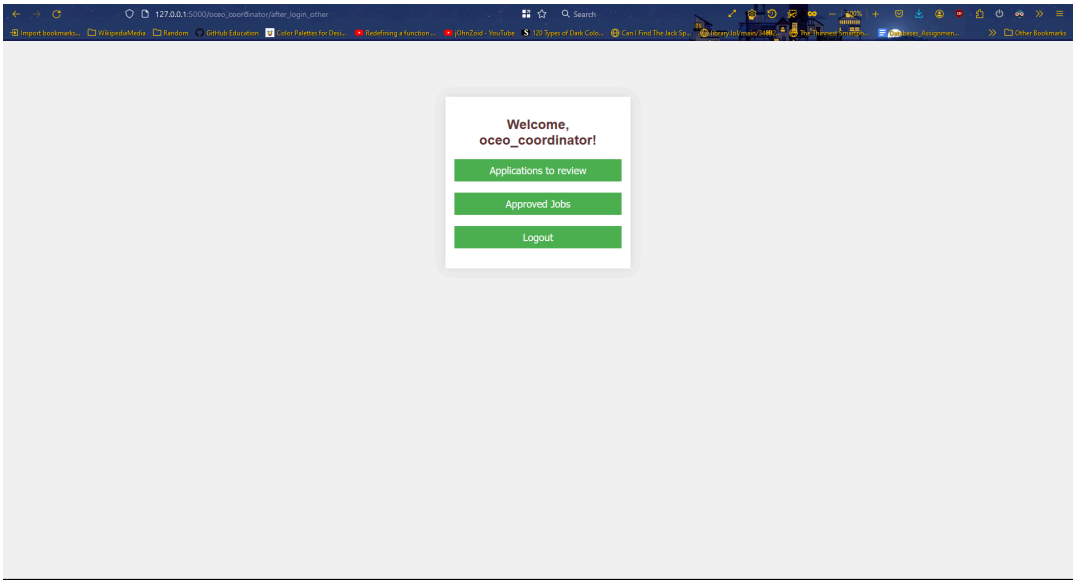


II. After first feedback:

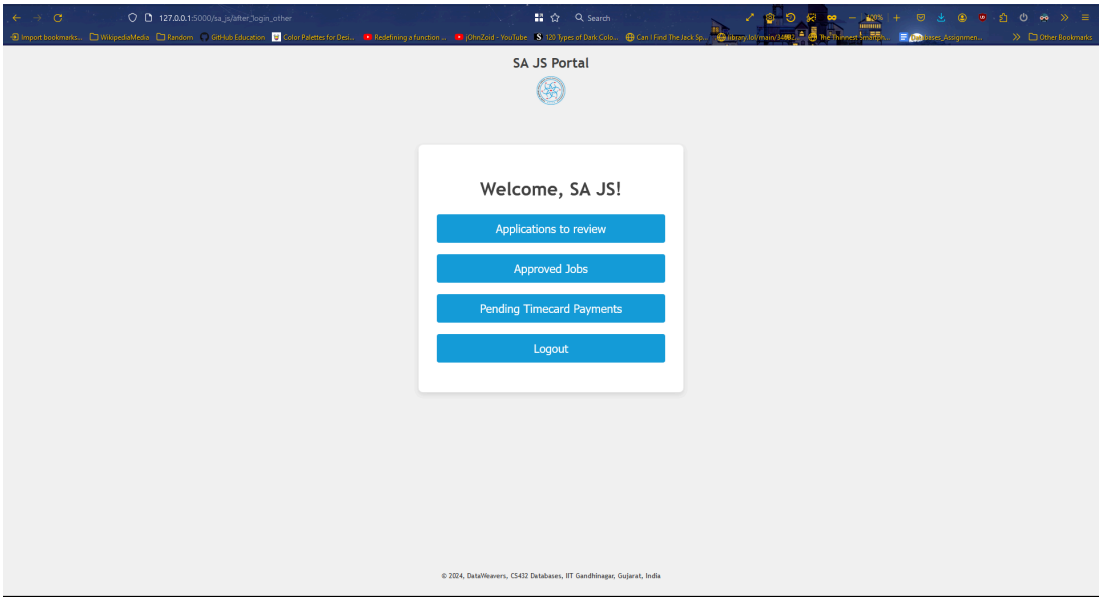
a. oCEO coordinator able to login:





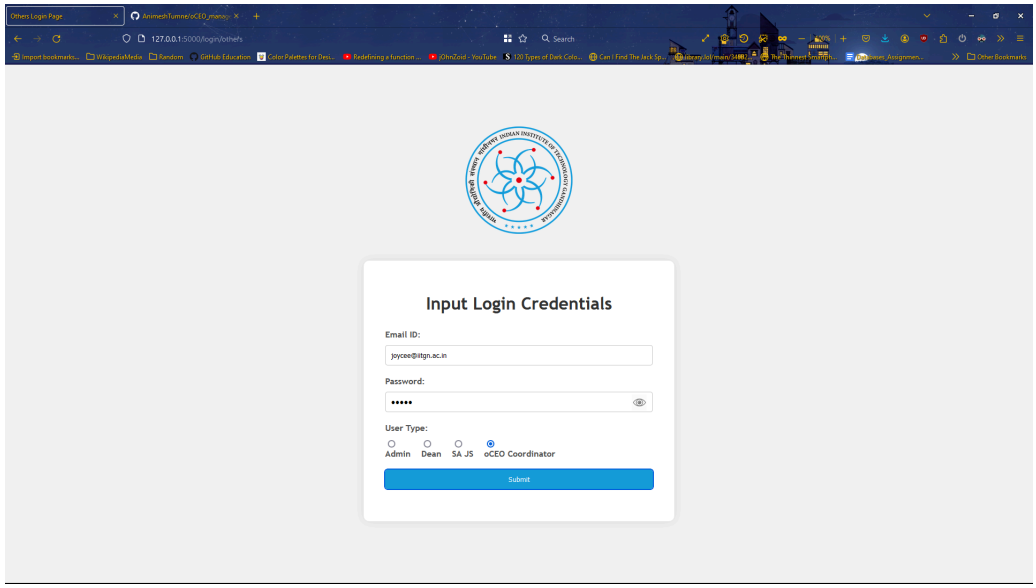


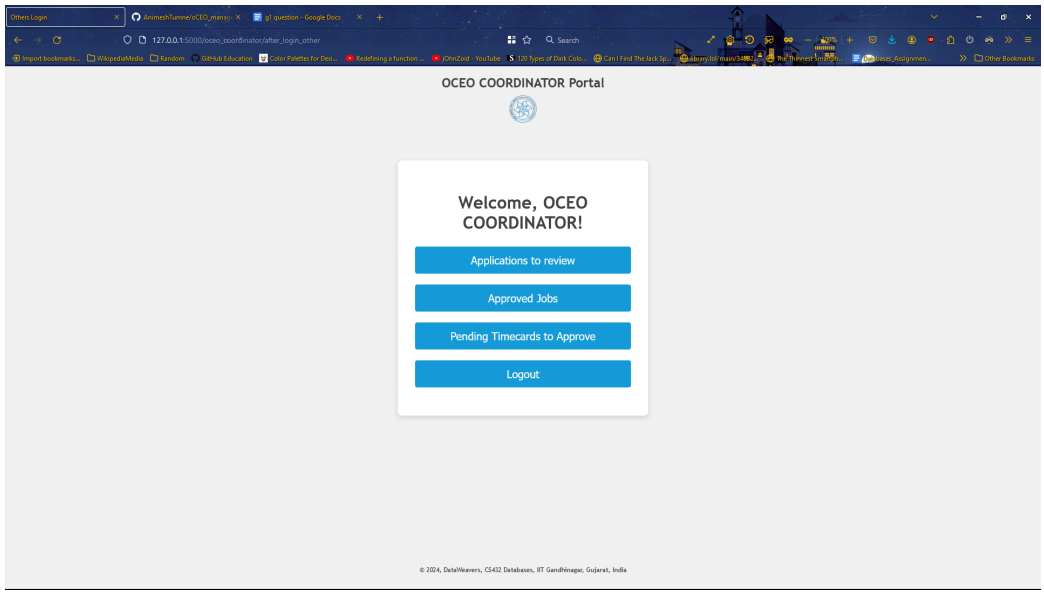
b. SA JS able to see pending payments:



III. After second feedback:

a. oCEO coordinator able to see pending timecards to approve:





2. Views and different privileges for different users:
   In the backend of our webapp, we are rendering different views with appropriate privileges for the different class of users (e.g. student, professor, oCEO coordinator, admin, dean and SA JS (Junior superintendent at Student Affairs department).

User Class:

1. **Student**

   Student have following privileges:

   - Change personal information
   - Change his/her password
   - See Available Jobs
   - See applied Jobs and the application status
   - Apply for a job and also delete applications.
   - View Time Details
   - Submit New Time Card
   - If employed in 'PAL' job , then the assigned mentees can also be seen

2. Faculty
   - Add a job
   - Delete a job
   - Approve/reject applications
   - approve/reject timecards
   - Change personal details
   - See employed students under them
   - Stop applications from accepting responses

3. Others

   In general,

   - See applications for review, and approve them.
   - See approved applications

   a. oCEO coordinator:
      - Timecards for approval
      - Approved timecards

   b. SA JS
      - Timecards for payment approval
      - Approved timecard payments

Tasks of G2

1. **Concurrent Multi-user Access**

   Concurrent access to multiple users has been implemented accordingly so that when one user makes an update to a given table, another user won't be simultaneously able to make an update; however, he/ she can view the database accordingly.

```python
cursor = db.connection.cursor()
cursor.execute("LOCK TABLES application_status WRITE")
# cursor.execute(f"SELECT application_id FROM application_status WHERE job_id = {job_id};")
cursor.execute(f"UPDATE application_status SET {perm} = 1 WHERE application_id = {application_id};")
if perm == 'dean_approved':
    cursor.execute(f"UPDATE application_status SET approval = 'approved' WHERE application_id = {application_id};")
db.connection.commit()
cursor.execute("UNLOCK TABLES")
cursor.close()
```

The above code changes have been made using *LOCK / UNLOCK* methods in SQL. When the user is accessing the database for an update, the tables are locked so that another user who is simultaneously trying to make an update in the table will have to wait once the update has been completed. Once the required update query has been executed, the other user will be able to make further updates in the table i.e., when it is unlocked.

This has been implemented for every update made by the Dean, Oceo Coordinator, SA_JS, and Oceo Admin since they will share a common table.

Further functions with locking implementation have been attached below.

```python
@app.route('/<type>/timecard_for_payment', methods=['GET', 'POST'])
def timecard_for_payment(type):
    if "email" in session:
        if request.method == 'POST':
            if request.form['submit_button'] == 'payment_done':
                job_id = request.form['job_id']
                roll_number = request.form['roll_number']
                month = request.form['month']
                print("#"*10)
                print(job_id, roll_number, month)
                print("#"*10)

                cursor = db.connection.cursor()
                cursor.execute("LOCK TABLES time_card  WRITE")
                cursor.execute(f"UPDATE time_card SET payment_status ='done'  WHERE job_id = {job_id} AND roll_number = {roll_number} AND
                db.connection.commit()
                cursor.execute("UNLOCK TABLES")
                cursor.close()
```

```python
        cursor.close()
        return redirect(url_for('review_application',type=type))
    elif request.form['submit_button'] == 'Reject':
        application_id = request.form['application_id']
        cursor = db.connection.cursor()
        cursor.execute("LOCK TABLES application_status WRITE")
        # cursor.execute(f"SELECT application_id FROM application_status WHERE job_id = {job_id};")
        cursor.execute(f"UPDATE application_status SET {perm}= 0 WHERE application_id = {application_id};")
        cursor.execute(f"UPDATE application_status SET approval = 'rejected' WHERE application_id = {application_id};")
        db.connection.commit()
        cursor.execute("UNLOCK TABLES")
        cursor.close()
        return redirect(url_for('review_application',type=type))
```

```python
@app.route('/<type>/pending_payments', methods=['GET', 'POST'])
def pending_payments(type):
    if "email" in session:
        if request.method == 'POST':
            if request.form['submit_button'] == 'payment_done':
                job_id = request.form['job_id']
                roll_number = request.form['roll_number']
                month = request.form['month']
                cursor = db.connection.cursor()
                cursor.execute("LOCK TABLES time_card WRITE")
                cursor.execute(f"UPDATE time_card SET payment_status ='done'  WHERE job_id = {job_id} AND roll_number = {roll_number} AND
                db.connection.commit()
                cursor.execute('UNLOCK TABLES')
                cursor.close()
            return redirect(url_for('pending_payments',type=type))
```

```python
@app.route('/<type>/review_application', methods=['GET', 'POST'])
def review_application(type):
    if "email" in session:
        if type =='admin':
            perm = 'faculty_approved'
        elif type =='dean':
            perm = 'dean_approved'
        elif type =='sa_js':
            perm  = 'SA_approved'
        elif type =='oceo_coordinator':
            perm = 'oceo_coordinator_approved'
        if request.method == 'POST':
            if request.form['submit_button'] == 'Approve':
                application_id = request.form['application_id']
                cursor = db.connection.cursor()
                cursor.execute("LOCK TABLES application_status WRITE")
                # cursor.execute(f"SELECT application_id FROM application_status WHERE job_id = {job_id};")
                cursor.execute(f"UPDATE application_status SET {perm} = 1 WHERE application_id = {application_id};")
                if perm == 'dean_approved':
                    cursor.execute(f"UPDATE application_status SET approval = 'approved' WHERE application_id = {application_id};")
                db.connection.commit()
                cursor.execute("UNLOCK TABLES")
                cursor.close()
```

```python
elif request.form['submit_button'] == 'Reject':
    application_id = request.form['application_id']
    cursor = db.connection.cursor()
    cursor.execute("LOCK TABLES application_status WRITE")
    # cursor.execute(f"SELECT application_id FROM application_status WHERE job_id = {job_id};")
    cursor.execute(f"UPDATE application_status SET {perm}= 0 WHERE application_id = {application_id};")
    cursor.execute(f"UPDATE application_status SET approval = 'rejected' WHERE application_id = {application_id};")
    db.connection.commit()
    cursor.execute("UNLOCK TABLES")
    cursor.close()
    return redirect(url_for('review_application',type=type))
```

2. Changes made based on received feedback:

- Based on first feedback, we made the following changes to improve our webapp:
    - Added faculty email id in the others table for oceo coordinator, since no separate email id is there in the name of oceo coordinator. Now, the oceo coordinator can login from the usual "Others" section in the webapp homepage, but now with their faculty email id (e.g. joycee@iitgn.ac.in). It is the role of admin to update the current oceo coordinator's email id.
    - The payment and bank details corresponding to a time card now shown to the SA JS as well, with the authority to complete the payment. For this,

suitable changes done in the backend, so that when SA JS approves a payment, the payment status is reflected as "done" in the time card details.

- Based on the second feedback, we made the following changes to improve our webapp:
    - Added authority of oceo coordinator to approve time cards, so that after being approved by the faculty and before being seen by SA JS, the time card should be approved by the oceo coordinator.

3. Adding Google Authentication:

- We added Google Authentication in order to allow only the users from IIT Gandhinagar to access our webapp (register/login).
- For this, we first created a project in Google Cloud Console, added relevant information like authorised URL and User Type as "Internal", after which we obtained the CLIENT_ID and CLIENT_SECRET.
- Please find the credentials (please use with care):

  CLIENT ID = "483725292816-7o6i235adoidqqhfv1hi0738paa4cfvr.apps.googleusercontent.com"

  CLIENT SECRET = "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"

- Code added in the backend for Google Auth:

```
from authlib.integrations.flask_client import OAuth
oauth = OAuth(app)
app.config['SERVER_NAME'] = '127.0.0.1:5000'

@app.route('/google/<user_type>')
def google(user_type):
   GOOGLE_CLIENT_ID="" # added the generated CLIENT ID
   GOOGLE_CLIENT_SECRET = "" # added the generated CLIENT SECRET
   CONF_URL =
'https://accounts.google.com/.well-known/openid-configuration'
   oauth.register(
       name='google',
       client_id=GOOGLE_CLIENT_ID,
       client_secret=GOOGLE_CLIENT_SECRET,
       server_metadata_url=CONF_URL,
       client_kwargs={
           'scope': 'openid email profile'
```

```
        }
    )
    redirect_uri = url_for('google_auth',user_type=user_type,
_external=True)
    return oauth.google.authorize_redirect(redirect_uri)

@app.route('/google/auth/<user_type>', methods =['GET', 'POST'])
def google_auth(user_type):
    if user_type=="student":
        redirect_uri = url_for('login_student')
    elif user_type=="professor":
        redirect_uri = url_for('login_professor')
    elif user_type=="others":
        redirect_uri = url_for('others_login')
    elif user_type=="new_user":
        redirect_uri = url_for('register')
    return redirect(redirect_uri)
```

- Relevant changes were made in the frontend files, in such a way that any button in our webapp homepage will only give the desired page when you are authenticated with Google.
- When authorising with Google, the users will be required to log in through their IITGN email id, otherwise, the webapp blocks them to access any further pages (not authorised).

Tasks of G1 and G2 Combined

1. **Performed Attacks and Defenses against them:**

   I. **SQL INJECTION:**

      A. Example 1

      In the below shown images, we see the job details page of a particular job (identified by job id) floated by a professor. Here, the URL is generated based on the entered job id written in the last. This job id value runs the required SQL query, which then renders the shown job page. Here, at the place of job id in URL, we can give 105 or 1=1, which then causes to render all the employed students in all jobs.

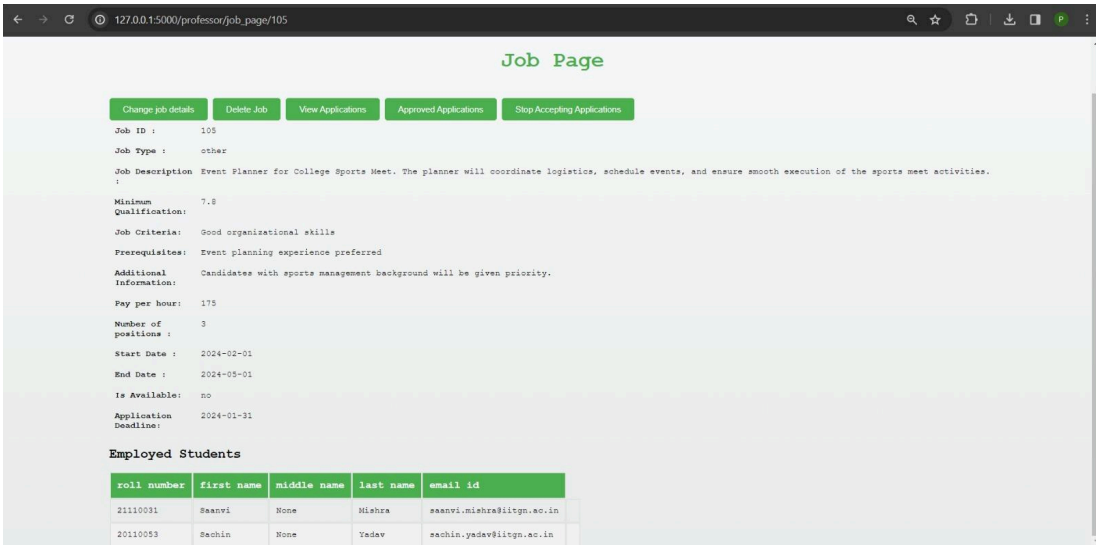


Fig: The desired page showing job details.

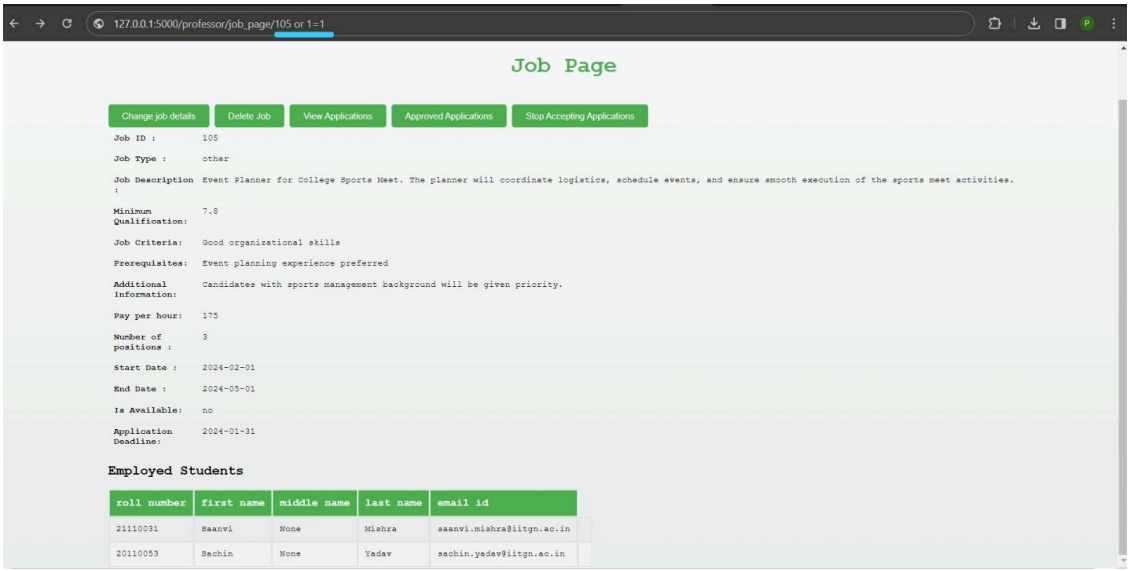
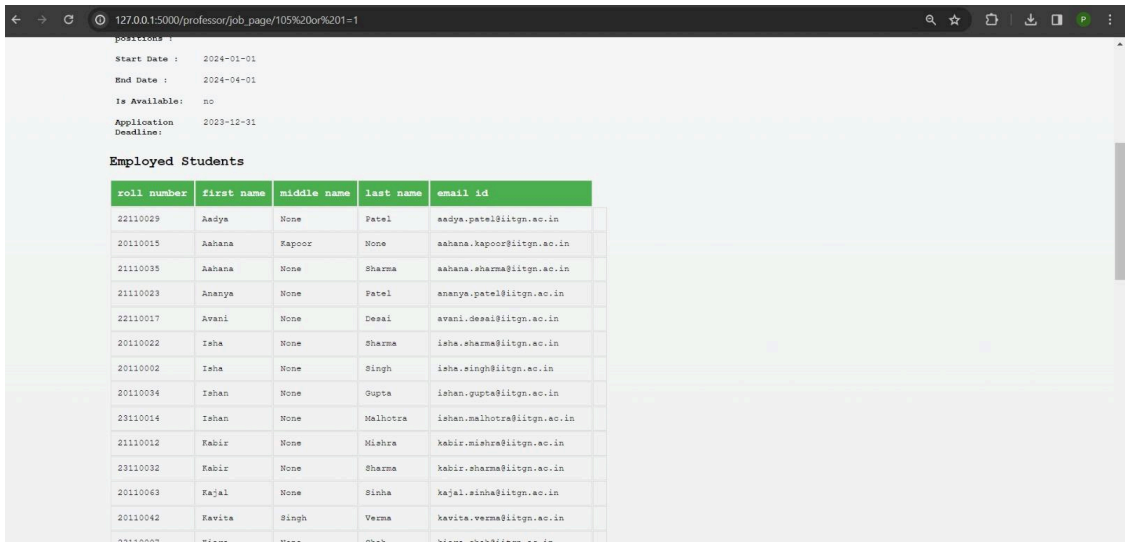


Fig: URL modified for SQL injection



roll number	first name	middle name	last name	email id
22110029	Aadya	None	Patel	aadya.patel@iitgn.ac.in
20110015	Aahana	Kapoor	None	aahana.kapoor@iitgn.ac.in
21110035	Aahana	None	Sharma	aahana.sharma@iitgn.ac.in
21110023	Ananya	None	Patel	ananya.patel@iitgn.ac.in
22110017	Avani	None	Desai	avani.desai@iitgn.ac.in
20110022	Isha	None	Sharma	isha.sharma@iitgn.ac.in
20110002	Isha	None	Singh	isha.singh@iitgn.ac.in
20110034	Ishan	None	Gupta	ishan.gupta@iitgn.ac.in
23110014	Ishan	None	Malhotra	ishan.malhotra@iitgn.ac.in
21110012	Kabir	None	Mishra	kabir.mishra@iitgn.ac.in
23110032	Kabir	None	Sharma	kabir.sharma@iitgn.ac.in
20110063	Kajal	None	Sinha	kajal.sinha@iitgn.ac.in
20110042	Kavita	Singh	Verma	kavita.verma@iitgn.ac.in

Fig: SQL injection caused to fetch details of all the employes students.

<u>Defense for this attack:</u> In the query which uses the job_id entered in URL, enclose {job_id} of the f-string under single quotes.

B. Example 2:

Here, we try to change the bank details of another student, from our update profile section.

For this, a student first goes to their bank details under personal information, then goes to edit bank details:

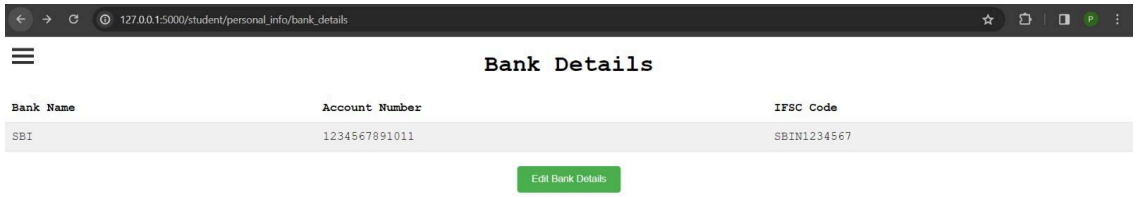


Fig: Current bank details of the student with option to edit.

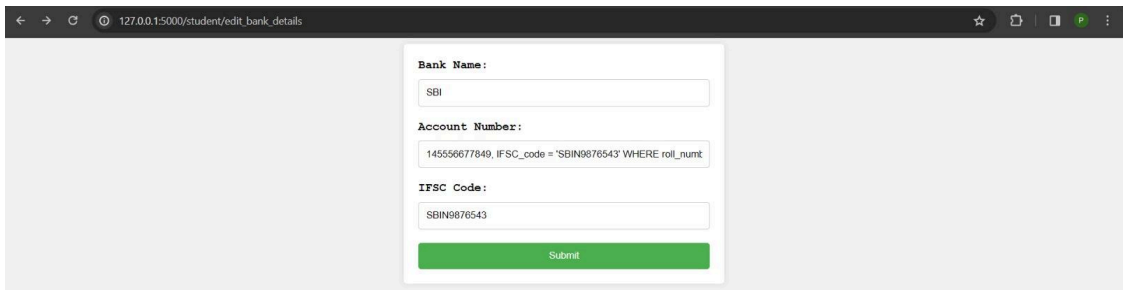


Fig: The student enters query to be injected in the Account number field.

Given input:
145556677849, IFSC_code = SBIN9876543 WHERE roll_number = 21110160; UPDATE bank_details SET bank_name ='SBI', account_number =1234567891011, IFSC_code = SBIN1234567 WHERE roll_number = 21110040;

In the bank_details table, account number is Primary key. In the injected query, we wish to change the account number of another student into our account number. For this, we first update our own account number into a random one. Then, the second query (written after ;) updates the bank details of another student.

This input is handled by the backend as:

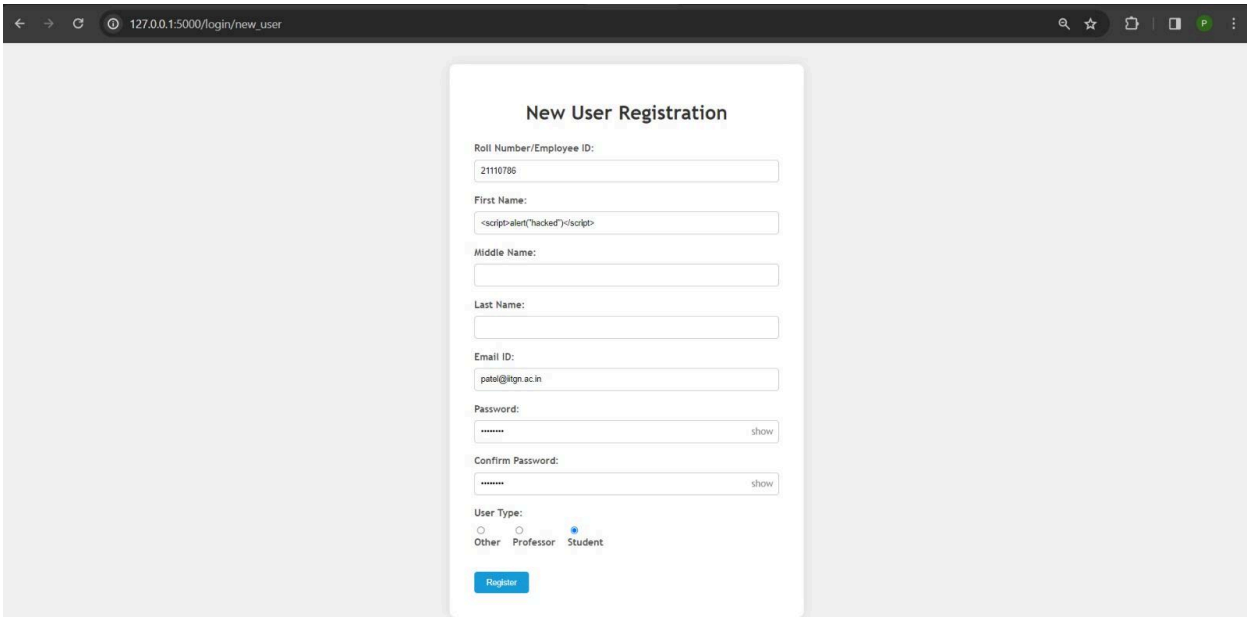
```
query = f"UPDATE bank_details SET bank_name = '{bank_name}',
account_number = {account_number}, IFSC_code = '{ifsc_code}' WHERE
roll_number = {roll_number};"
cursor.execute(update_query)
```

But SQL injection of this kind (multiple queries at once) doesn't apply and throws an error. This is because the cursor.execute() method here can process single SQL queries only, and hence acting as a defense for such SQL injections (source: https://stackoverflow.com/questions/20518677/mysqldb-cursor-execute-cant-run-multiple-queries/20539187#20539187).

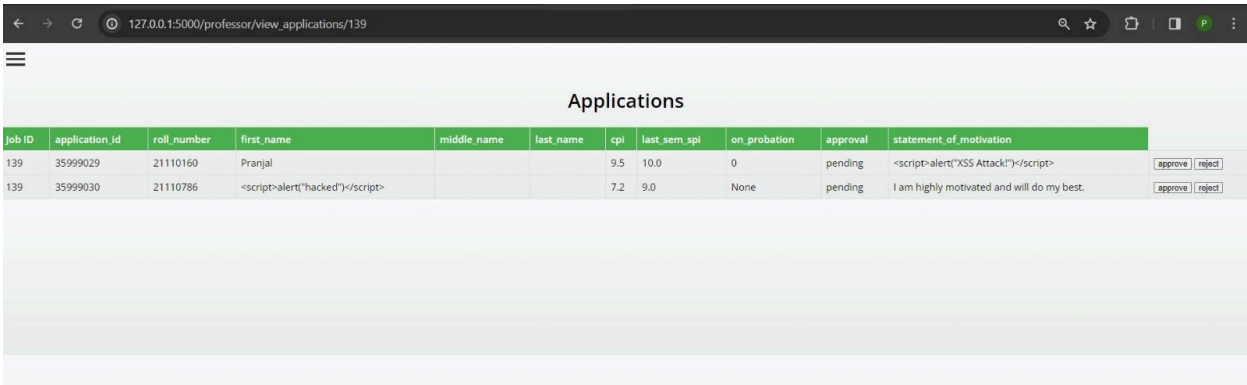
II. XSS Attack

In this type of attack, we try to inject malicious script into the code which runs when a user interacts with it. In our webapp, the data written in the input fields are stored in the database, and when required to render in the webpage (e.g. in a table), the data is fetched from the database, which only returns strings. Thus, when tried to inject script through such input blocks, the script does not run when encountered later while rendering a page. Instead, it renders as a string. In this way, malicious scripts can't be used in our webapp by storing them through input fields.

Example:
Entered alert script in the first name input when creating a new user.



This student then applies to a job. When the professor open the current applications under that job, the first name field should cause the alert to appear. But, the script is instead rendered as a string.



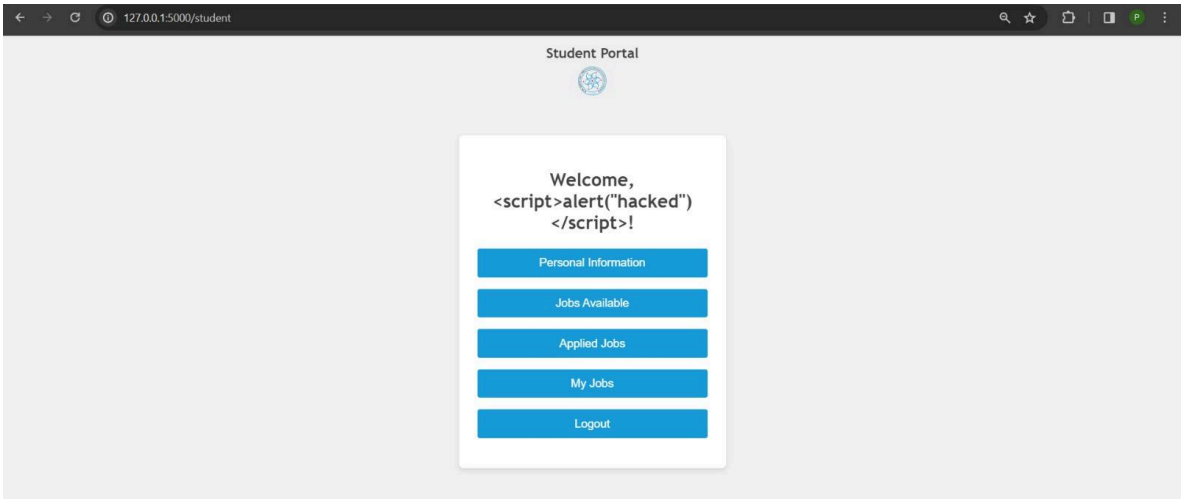
job ID	application_id	roll_number	first_name	middle_name	last_name	cpi	last_sem_spi	on_probation	approval	statement_of_motivation	
139	35999029	21110160	Pranjal			9.5	10.0	0	pending	<script>alert("XSS Attack!")</script>	approve reject
139	35999030	21110786	<script>alert("hacked")</script>			7.2	9.0	None	pending	I am highly motivated and will do my best.	approve reject

The above was an example of rendering values in a table. We tried another case, where the fetched data from the database is not explicitly being rendered as a string, but even in that case the script is printed instead of running.

Here, the script is sent into student_name below:

```
<form action="{{url_for('after_login_student')}}" method="post">
    <div class="container">
        <h2>Welcome, {{student_name}}!</h2>
        <button class='btn' name="submit_button" value="personal
```

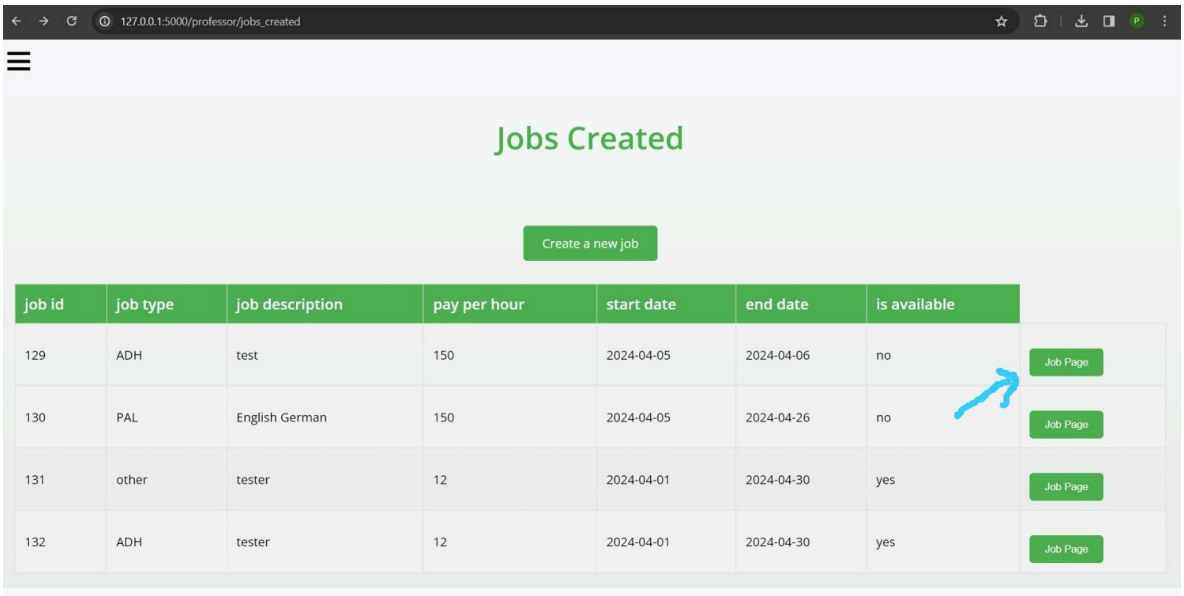
The output seen is:



III. Attack from URL

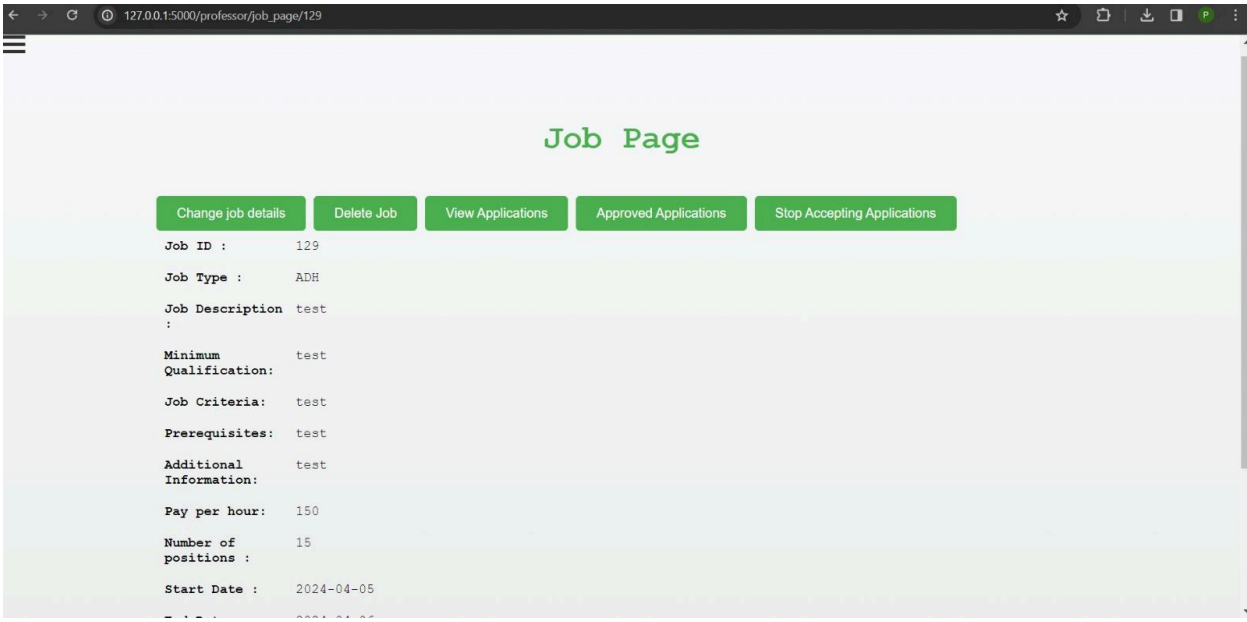
Whenever a professor goes to see their floated job details (based on a particular job_id), then the job_id goes to the URL which then renders the jop details page accordingly.

Example: The professor choose to see the job page of job_id = 129:

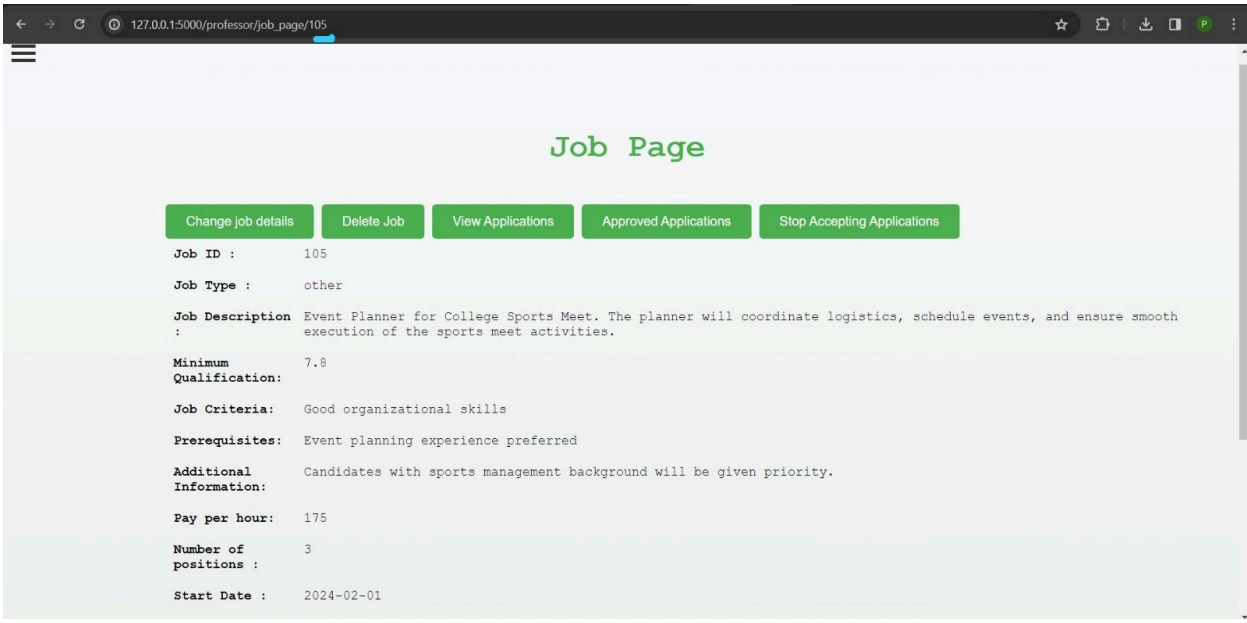


job id	job type	job description	pay per hour	start date	end date	is available	
129	ADH	test	150	2024-04-05	2024-04-06	no	Job Page
130	PAL	English German	150	2024-04-05	2024-04-26	no	Job Page
131	other	tester	12	2024-04-01	2024-04-30	yes	Job Page
132	ADH	tester	12	2024-04-01	2024-04-30	yes	Job Page

The job_id = 129 is entered last in the URL, for which the professor gets the page as:



However, the URL can be manually changed to enter another job_id, say 105. Then, the page renders the details of job_id = 105.



Defense for this attack:
Before running the query, we check if that job_id is floated by that faculty by running another query involving natural join of faculty and job, where we filter using the faculty_id.

IV. URL Interpretation Attack

Similar to above attack, any faculty can edit any other job details by manipulating job_id in the URL bar.
Defense for this attack:
Before running the query, we check if that job_id is floated by that faculty by running another query involving natural join of faculty and job, where we filter using the faculty_id.

2. Relations and their constraints, finalized after the second feedback:

Final tables:

adh:

Query 1 | oceo_management.application_... | oceo_management.adh

Info | Columns | Indexes | Triggers | Foreign keys | Partitions | Grants | DDL

Column	Type	Default Value	Nullable	Character Set	Collation	Privileges	Extra	Comments
course_id	varchar(5)		YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references		
course_name	varchar(45)		NO	utf8mb4	utf8mb4_0900_...	select,insert,update,references		
job_id	int		NO			select,insert,update,references		

application_status:

Query 1 | oceo_management.application_... | oceo_management.adh | oceo_management.application_...

Info | Columns | Indexes | Triggers | Foreign keys | Partitions | Grants | DDL

Column	Type	Default Value	Nullable	Character Set	Collation	Privileges
application_id	int		NO			select,insert,update,references
approval	enum('approved','rejected','pending')		NO	utf8mb4	utf8mb4_0900_ai_ci	select,insert,update,references
dean_approved	tinyint(1)		NO			select,insert,update,references
faculty_approved	tinyint(1)		NO			select,insert,update,references
job_id	int		NO			select,insert,update,references
oceo_coordinator_ap...	tinyint(1)		NO			select,insert,update,references
roll_number	int		NO			select,insert,update,references
SA_approved	tinyint(1)		NO			select,insert,update,references
statement_of_motiva...	text		YES	utf8mb4	utf8mb4_0900_ai_ci	select,insert,update,references

applied_student:

Query 1 | oceo_management.applied_stu...

Info | Columns | Indexes | Triggers | Foreign keys | Partitions | Grants | DDL

Column	Type	Default Value	Nullable	Character Set	Collation	Privileges	Ext
cpi	float		YES			select,insert,update,references	
email_id	varchar(100)		NO	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
first_name	varchar(45)		NO	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
last_name	varchar(45)		YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
last_sem_spi	float		YES			select,insert,update,references	
middle_name	varchar(45)		YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
on_probation	tinyint(1)		YES			select,insert,update,references	
password	varchar(255)		NO	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
person_img	blob		YES			select,insert,update,references	
person_img_caption	text		YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
roll_number	int		NO			select,insert,update,references	

applied_student_phone:

Query 1 | oceo_management.applied_stu... | oceo_management.applied_stu...

Info | Columns | Indexes | Triggers | Foreign keys | Partitions | Grants | DDL

Column	Type	Default Value	Nullable	Character Set	Collation	Privileges	Extra
phone_number	bigint		NO			select,insert,update,references	
roll_number	bigint		NO			select,insert,update,references	

bank_details:

Query 1 | oceo_management.applied_stu... | oceo_management.applied_stu... | oceo_management.bank_details

Info | Columns | Indexes | Triggers | Foreign keys | Partitions | Grants | DDL

Column	Type	Default Value	Nullable	Character Set	Collation	Privileges
account_number	bigint		NO			select,insert,update,references
bank_name	varchar(100)		NO	utf8mb4	utf8mb4_0900_...	select,insert,update,references
IFSC_code	varchar(40)		YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references
roll_number	int		NO			select,insert,update,references

employed_students:

Query 1 | oceo_management.applied_stu... | oceo_management.applied_stu... | oceo_management.bank_details | oceo_management.employed_s ...

Info | Columns | Indexes | Triggers | Foreign keys | Partitions | Grants | DDL

Column	Type	Default Value	Nullable	Character Set	Collation	Privileges	Extra	C
is_employed	tinyint(1)		NO			select,insert,update,references		
job_id	int		NO			select,insert,update,references		
roll_number	int		NO			select,insert,update,references		

faculty:

Query 1 | oceo_management.applied_stu... | oceo_management.applied_stu... | oceo_management.bank_details | oceo_management.employed_s ... | oceo_

Info | Columns | Indexes | Triggers | Foreign keys | Partitions | Grants | DDL

Column	Type	Default Value	Nullable	Character Set	Collation	Privileges	Extra
dept_name	varchar(45)		YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
email_id	varchar(100)		YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
faculty_id	int		NO			select,insert,update,references	
first_name	varchar(45)		NO	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
last_name	varchar(45)		YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
middle_name	varchar(45)		YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
password	varchar(255)		YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
voip_id	int		YES			select,insert,update,references	

job:

Query 1 | oceo_management.job

Info | Columns | Indexes | Triggers | Foreign keys | Partitions | Grants | DDL

Column	Type	Default Value	Nullable	Character Set	Collation	Privileges	Extr
additional_info	text		YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
application_deadline	date		YES			select,insert,update,references	
end_date	date		YES			select,insert,update,references	
faculty_id	int		NO			select,insert,update,references	
is_available	enum('yes','no')		YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
job_criteria	varchar(45)		YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
job_description	text		NO	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
job_id	int		NO			select,insert,update,references	
job_type	varchar(20)		NO	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
min_qualifications	text		YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
no_of_positions	int		YES			select,insert,update,references	
pay_per_hour	int		YES			select,insert,update,references	
prerequisites	text		YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
start_date	date		YES			select,insert,update,references	
tenure	int		YES			select,insert,update,references	

mentees:

Query 1 | oceo_management.mentees

Info | Columns | Indexes | Triggers | Foreign keys | Partitions | Grants | DDL

Column	Type	Default Value	Nullable	Character Set	Collation	Privileges
first_name	varchar(45)		NO	utf8mb4	utf8mb4_0900_...	select,insert,update,references
last_name	varchar(45)		YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references
mentee_roll_number	int		NO			select,insert,update,references
middle_name	varchar(45)		YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references

mentor_mentee:

Query 1 | oceo_management.mentor_men...

Info | Columns | Indexes | Triggers | Foreign keys | Partitions | Grants | DDL

Column	Type	Default Value	Nullable	Character Set	Collation	Privileges
job_id	int		YES			select,insert,update,references
mentee_roll_number	int		NO			select,insert,update,references
roll_number	int		YES			select,insert,update,references

other:

Query 1 | oceo_management.other

Info | Columns | Indexes | Triggers | Foreign keys | Partitions | Grants | DDL

Column	Type	Default Value	Nullable	Character Set	Collation	Privileges
email	varchar(50)		NO	utf8mb4	utf8mb4_0900_...	select,insert,update,references
password	varchar(255)		YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references
user_type	enum('dean','oceo_c...		NO	utf8mb4	utf8mb4_0900_...	select,insert,update,references

others:

Query 1 | oceo_management.other | oceo_management.others

Info | Columns | Indexes | Triggers | Foreign keys | Partitions | Grants | DDL

Column	Type	Default Value	Nullable	Character Set	Collation	Privileges	Extra	Comr
job_id	int		NO			select,insert,update,references		
role_name	varchar(45)	_cp850\'Employee\'	YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references	DEFAULT_GENE...	

sem_other_jobs:

oceo_management.sem_other_j ...

Info | Columns | Indexes | Triggers | Foreign keys | Partitions | Grants | DDL

Column	Type	Default Value	Nullable	Character Set	Collation	Privileges	E:
additional_info	text		YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
end_date	date		YES			select,insert,update,references	
faculty_id	int		NO			select,insert,update,references	
is_available	enum('yes','no')		YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
is_position_open	tinyint(1)		NO			select,insert,update,references	
job_criteria	varchar(45)		YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
job_description	text		NO	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
job_id	int		NO			select,insert,update,references	
job_type	varchar(20)		NO	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
min_qualifications	text		YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
no_of_positions	int		YES			select,insert,update,references	
pay_per_hour	int		YES			select,insert,update,references	
prerequisites	text		YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
start_date	date		YES			select,insert,update,references	
tenure	int		YES			select,insert,update,references	

subjects_under_pal:

oceo_management.subjects_un...

Info | Columns | Indexes | Triggers | Foreign keys | Partitions | Grants | DDL

Column	Type	Default Value	Nullable	Character Set	Collation	Privileges
job_id	int		NO			select,insert,update,references
subjects_to_teach	varchar(45)		NO	utf8mb4	utf8mb4_0900_...	select,insert,update,references

time_card:

oceo_management.time_card

Info | Columns | Indexes | Triggers | Foreign keys | Partitions | Grants | DDL

Column	Type	Default Value	Nullable	Character Set	Collation	Privileges
faculty_approval	enum('approved','rej...		YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references
hours_worked	int		NO			select,insert,update,references
job_id	int		NO			select,insert,update,references
month	varchar(20)		NO	utf8mb4	utf8mb4_0900_...	select,insert,update,references
oceo_coordinator_ap...	enum('approved','pe...	pending	NO	utf8mb4	utf8mb4_0900_...	select,insert,update,references
payment_status	enum('done','pendin...		YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references
roll_number	int		NO			select,insert,update,references
work_description	text		NO	utf8mb4	utf8mb4_0900_...	select,insert,update,references
year	int		YES			select,insert,update,references

transaction_status:

oceo_management.transaction_...

Info | Columns | Indexes | Triggers | Foreign keys | Partitions | Grants | DDL

Column	Type	Default Value	Nullable	Character Set	Collation	Privileges
is_payment_initiated	tinyint(1)		NO			select,insert,update,references
is_payment_received	tinyint(1)		NO			select,insert,update,references
job_id	int		NO			select,insert,update,references
month	varchar(25)		YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references
roll_number	int		NO			select,insert,update,references
time_of_payment	datetime		YES			select,insert,update,references
transaction_id	int		NO			select,insert,update,references

Contributions:

Aditya Gupte:

-

Balgopal Moharana:

- Enhanced backend with concurrency control and locking.
- Optimized table operations for seamless multi-user edits, benefiting Dean SA, SA JS, and OCEO Coordinator.
- Report writing for G2 concurrency question

Darshi Doshi:

- Took feedback from the stakeholder
- Tried doing google authentication

Shah Faisal Khan:

-

Pranjal:

- Modified the web application based on the feedback of stakeholders. (Added backend for time card oCEO coordinator approval.)
- Added Google Authentication to the web application.
- Contributed in finding possibilities for the SQL injection and XSS injection and their defenses.
- Provided feedback on the front-end design and suggested some improvements for better UI.

Rohit Raj:

- Taking 2nd feedback with Animesh.
- Involved in making changes in the webapp and database per the feedback.
- Helped add Google Authentication with Pranjal.
- Found possible attacks through the webapp along with the defenses involved.
- Report writing: G2 (Q2, Q3); G1 and G2 (Q1)
- Added backend code for oceo coordinator time card approval.

Tanish Phopalkar:

- Worked on Concurrency of other users
- Enhanced tables operations for concurrency control

Animesh Tumne:

- Worked on revamping the whole WebApp FrontEnd
- Designed the whole frontend from scratch including tasks like deciding the theme and implementing javascript and css.
- Reworked each link to make sure it is working properly
- Worked on attacks, and how to defend against them

Group Distribution:

G1: Animesh, Darshi

G2: Pranjal, Rohit, Balgopal, Tanish