CS432 Databases
ScholarEase- Scholarship/Stipend Management System
Module Task 3

23110103@iitgn.ac.in; 23110140@iitgn.ac.in; 23110192@iitgn.ac.in; 23110353@iitgn.ac.in

Q1) How you designed your project-specific table

We created a project-specific MySQL database (`cs432g16`) to store data relevant only to the scholarship management functionality. This included a total of 15 tables with the major ones being:

- **Scholarship Table:** Stores various scholarship opportunities with its ID, Name, Amount and deadline to apply. Students can use this table to view the available scholarships.
- **College Details Table**: This is a table which contains details of all the students' in the college like name,age,program,department,gender,contact,annual_income,cpi and photo. This can be accessed by only the admin.
- **Eligibility Table:** This table contains the eligibility criteria (like minimum CPI or maximum annual_income) for each scholarship. This is used to check the eligibility and define the status (approved/rejected) in the status table.
- **Application table:** This table contains application data of students that apply for a scholarship/financial aid.
- **Bank Details Table** : Stores bank account information linked to applications.
- **Change_Log:** This table is used to log all changes made to the centralized CIMS database via API calls. It captures details like the action performed, timestamp, member ID, and session token to ensure traceability.
- **Admin Table:** This table includes details of the 4 admins that are handling the scholarships/stipends.

These tables were normalized for efficient querying and flexibility. Foreign keys (if present) were used to ensure relational integrity across tables.

Q2) How you integrated with the CIMS database

To ensure centralized authentication and member identity management, we integrated our system with the `cs432cims` database. We created a dual-database architecture in our Flask backend. Host : (`10.0.116.125`) , and

We did this using the common members and login table already created in the centralized database. We added new members in the members table which were simultaneously added in the login table and images after authorization. A session token was also generated. Using this, we allowed users to login into our database portal. When a user logged in with an ID and password, the session token authenticates whether the user is an admin or a student. Based on that the corresponding functionalities are displayed with some controls for students. Like only admins can update/delete data, while a student can view all the scholarships, apply for one, check its status etc.

Session token was generated using the following:

```
 # Generate new session token and expiry timestamp (as INT)

        session_token = str(uuid.uuid4())

        expiry_time = int((datetime.now() +
timedelta(hours=1)).timestamp())
```

Any member from the cims database will be able to login in our portal, and work with the scholarship database, as a student or an admin, which an integration between the cims and g16 database has been done.
All major changes like adding, deleteing and updating are recorded in the change_log table in g16 databse.

Q3) How you implemented session validation prevented data leaks

Security was enforced through authorization:

Session Tokens: Upon login, users receive a UUID (Universally Unique Identifier 4) -based session token stored in the `Login` table with an expiry time (stored as a UNIX timestamp). The token is returned in the login response and must be included in the `Authorization` header for all subsequent requests.

Example of a UUID 4 session token from our database:

c807721b-e8b0-466b-af8d-27c7329a7442

Session Validation: All admin-only endpoints (like `admin_add_member`, `admin_delete_member`, etc.) check the session token using `is_admin_authorized()` Only is the authorization is successful , the user is allowed to access admin functions.

Role-Based Access Control (RBAC):
Regular users can only access general data (like scholarship details and apply for them).

Admin users (validated via the `Role` field in `Login`) can perform CRUD operations on member data and view restricted datasets (like bank_details and college_details).

Data Leak Prevention: To prevent data leaks the following was done:
No sensitive member information is exposed without proper authorization.
All API responses are validated.
Admin endpoints explicitly check headers for an Authorization token.
A logging system using Python's `logging` module was implemented to record activity which aids in monitoring and debugging.

All the sensitive routes were protected by validating it with the following api:

```
def is_admin_authorized(session_token):

    try:

        conn = get_db_connection(cims=True)

        cursor = conn.cursor(dictionary=True)

        cursor.execute("""

            SELECT Role FROM Login

            WHERE Session = %s AND Expiry > %s

        """, (session_token, int(datetime.now().timestamp())))

        result = cursor.fetchone()

        return result and result['Role'] == 'admin'

    except mysql.connector.Error as e:

        logging.error(f"Authorization check failed: {e}")

        return False

    finally:

        cursor.close()

        conn.close()
```

Link to the Git Repository: https://github.com/Diyanandan/G16_Task3_CS432.git

Link to the video: Module3_G16_final video.mp4

Work Contribution

Name	Roll Number	Contributions
Diya Nandan	23110103	All member functionalities
Ishika Paresh Patel	23110140	All member functionalities
Maitri Chaudhari	23110192	All admin functionalities
Vardayini Agrawal	23110353	All admin functionalities

All members have equally contributed to the planning, working, and integration of this project. All of us have worked together in making and debugging the codes, and also in the making of the report.