CS432 – Module 3 Task
Database Integration and Secure API Development

Due Date: 14 April 2025, 11:59 PM

Objective

This assignment focuses on connecting project-specific databases to a central database (CIMS – Centeral Information Management System), using shared tables instead of storing duplicate data, creating secure APIs that check user sessions before making database changes, setting up role-based access control to limit user actions while allowing admin tasks, keeping logs of all changes to detect unauthorized or fake transactions, managing member creation and removal while keeping data accurate, and identifying any unauthorized changes that skip session checks.

Prerequisites

1. Programming with Python, or any programming language to impliment APIs and web UI.
2. SQL Knowledge – Understanding of SQL queries, database security concepts like authentication, session management, and role-based access control.

Assignment Details

1. SQL Server Location: `http://10.0.116.125/phpmyadmin`
2. Authentical APIs Location: `http://10.0.116.125:5000/`
3. Centralized Database Name: `cs432cims`
4. Centralized Tables in CIMS Database:
   (a) `members`: Contains details of all members (e.g., `member_id`, `name`, `email`).
   (b) `members_group_mapping`: Maps members to groups (to maintain integrity).
   (c) `payments`: Contains payment-related information.
   (d) `login`: Contains user credentials (`userid`, `password`) and active sessions (`session`).
5. Project-Specific Databases: Each group has its own database named `cs432g1` to `cs432g17`. These databases are empty, and students must create project-specific tables as required by their project.

Tasks

Task 1: Member Creation – When a new member is created in the centralized `members` table, create a relevant entry in the `login` table for this member with default credentials (`userid`, `password`). This allows the member to log in and obtain a session token using the authentication API (`authUser`).

Task 2: Role-Based Access Control – Implement role-based access control (RBAC):

1. Regular users should not be allowed to perform admin-level actions such as adding/deleting members or accessing any CIMS database directly.
2. Admins should have full access to perform actions like adding/deleting members and accessing admin-level data.

Task 3: Member Deletion – When deleting a member from the centralized `members` table:

1. Check if the member is associated with any other group using the `members_group_mapping` table.
2. If the member is not related to any group, delete the member from both the `members` table and the corresponding entry in the `login` table.

3. If the member is associated with other groups, only remove the specific group mapping from the `members_group_mapping` table.

Task 4: Database Table Creation – Create the required tables in your project-specific database (`CS432_Gx`, where x is your group number). Do not duplicate any table that already exists in the CIMS database (e.g., do not create `members`, `payments`, etc.). Use centralized tables for these purposes.

Task 5: API Development – Develop web APIs locally on your system to perform operations as per your project requirements (e.g., CRUD operations). Ensure that:

1. Each API call validates the session using the provided centralized API (`isValidSession(session)`).
2. Admin-level actions are restricted to admin users only.
3. Unauthorized modifications (i.e., direct database writes without session validation) should be logged and flagged.

Task 6: Logging Changes to CIMS Database Whenever an API call modifies data in the centralized CIMS database (e.g., adding payments or updating member details), ensure logs are printed **locally on your system** and also logged to the server.

1. If any transaction is made without proper session validation, it will not appear in the server logs, revealing unauthorized modifications.

Task 7: Member Portfolio Management – Students must create a portfolio feature for members belonging to their own project. The portfolio should:

1. Display relevant details of members within the project.
2. Restrict access so that profiles of members from other projects remain hidden.
3. Ensure that only authenticated users can view the portfolio.
4. Implement appropriate role-based access (e.g., project admins can edit portfolio details, but regular members can only view their own profiles).

Submission Guidelines

Report – Prepare a report explaining your project implementation, including:

1. How you designed your project-specific tables.
2. How you integrated with the CIMS database.
3. How you implemented session validation prevented data leaks.
4. A short demonstration of your working APIs (share a video with demo of your project)

Code Submission – Submit all source code in a GitHub repository and add its link in your report, including:

1. API code.
2. SQL scripts for creating your project-specific tables.
3. Logs demonstrating session validation for each query.
4. Logs showing changes made to the CIMS database.

Assignment Grading Criteria

Task	Description
Member Creation	Properly integrates new members into the centralized database and login table.
Role-Based Access Control	Implements RBAC correctly to restrict unauthorized actions.
Member Deletion	Ensures proper deletion of members while maintaining integrity in group mappings.
Database Table Creation	Creates project-specific tables without duplicating existing centralized tables.
API Development	Implements secure APIs with session validation and logging.
Logging Changes	Logs all modifications locally and on the server to detect unauthorized access.
Member Portfolio	Ensures project members' profiles are visible only within their project.
Presentation	Clearly explains the implementation and demonstrates the working solution.
Code Submission	Provides complete and well-documented source code, including logs and scripts.
Team members Role	Clearly mention the contributions of each team member

API Documentation

/login (POST)

Description: Authenticates a user and issues a session token.

Arguments (JSON):

- `user` (string): Username.
- `password` (string): User's password.

Returns:

- **200 (JSON)**: {
  `"message": "Login successful",`
  `session_token : 'jwt session token'`
  }
- **401 (JSON)**:
  - {`"error": "Invalid credentials"`}
  - {`"error": "Missing parameters"`}

  on failure.

/isAuth (GET)

Description: Checks if a user's session is valid.

Arguments:

- {
  `session_token` : 'JWT session token'
  }

Returns:

- **200 (JSON)**: {`"message": "User is authenticated", "username": string, "role": string, "expiry": datetime`} on success.
- **401 (JSON)**:
  - {`"error": "No session found"`}
  - {`"error": "Session expired"`}
  - {`"error": "Invalid session token"`}

  on failure.

/ (GET)

Description: Simple welcome endpoint.

Arguments: None.

Returns:

- **200 (JSON)**: {`"message": "Welcome to test apis"`}.