1. **How We Designed Our Project-Specific Tables:**

As part of our **CS432 Placement Management System mini-project**, we were provided credentials to access **phpMyAdmin**, the web-based database management interface. Upon logging in, we observed two available databases:

- **cs432cims** — the main departmental database.
- **cs432g7** — our dedicated group database for project implementation.

Steps Followed in Table Design:

- **Reviewing Module-1:**
  We designed our database by writing SQL queries to create all necessary tables from Module 1, ensuring proper relationships and constraints. These tables manage student records, company profiles, job applications, interviews, placements, and feedback for the placement system.
- **Inserting Tables into the Group Database:**
  After finalizing the queries, we executed them in the **cs432g7** group database. This created all the required tables as per our project design in the dedicated space allocated to our team.
- **Handling Redundant Tables:**
  During this process, we noticed that the **placement_statistics** table already existed in the **cs432cims** main database.

  Since:

  It did not have any foreign key relationship with other tables in our database It was independent and maintained separately in the main database.

We decided **not to include placement_statistics in our cs432g7 group database** to avoid redundancy and unnecessary duplication of data.

2. **How you integrated with the CIMS database:**

   To ensure seamless and secure integration with the CIMS (Campus Information Management System) database, we implemented a structured approach that maintained data consistency while preventing unauthorized access or modifications to the core system.

    - Database Connection & Authentication
      We established a dedicated connection handler (database.py) to interact with the CIMS database (cs432cims) using restricted credentials. This ensured that our application had read-only access where necessary, preventing accidental modifications to critical institutional data. The connection was secured via environment variables (DB_HOST, DB_USER, DB_PASSWORD) to avoid hardcoding sensitive credentials.
    - Credential Verification & JWT Token Generation
      User authentication was performed against the CIMS Login table, where passwords were stored as hashed values. Upon successful validation, our system issued a JWT (JSON Web Token) containing the user's ID, role, and expiration time. This token was then used for session management in subsequent requests.
- Data Flow & Security
    - No Direct Modifications: We avoided altering the CIMS database schema or inserting records directly into it. Instead, our custom database (placement_system_1) stored project-specific data while referencing CIMS identifiers (e.g., student_id, company_id) for consistency.
    - Controlled Access: The integration was designed to pull necessary data (e.g., student records, company profiles) from CIMS into our system without creating dependencies that could disrupt the main database.

Session Validation & Data Leak Prevention:

To protect user sessions and prevent unauthorized data exposure, we implemented a **multi-layered security framework**:

1. **Strict Authentication & Token Validation**
    - Every API request requiring authentication was wrapped in a @token_required **decorator**, which verified the JWT's validity, expiration, and signature against the server's secret key (SECRET_KEY).
    - Invalid or expired tokens triggered **immediate session termination** (HTTP 401), and suspicious activity was logged for monitoring.
2. **Role-Based Access Control (RBAC)**
    - The JWT payload embedded the user's **role** (Admin, Student, or Company), which was validated before granting access to sensitive endpoints (e.g., placement records, interview schedules).

- For example, only **Admin users** could modify job postings, while Students could only view and apply to openings.

3. **Secure Password Storage & Validation**
    - Passwords were **hashed using BCrypt** (with MD5 fallback for legacy compatibility) to prevent plaintext exposure.
    - During login, the system compared input passwords against the hashed values stored in CIMS, ensuring **no raw passwords were transmitted or logged**.
4. **Session Hijacking Prevention**
    - **IP & User-Agent Validation**: Each session was tied to the user's IP and browser fingerprint. Mismatches (e.g., token reuse from a different device) invalidated the session.
    - **Auto-Expiring Tokens**: JWTs had a **1-hour lifespan**, requiring users to reauthenticate periodically.
5. **Comprehensive Logging & Error Handling**
    - All authentication attempts (successful/failed) were logged, including timestamps and IP addresses.
    - Database errors (e.g., connection failures) triggered graceful fallbacks instead of exposing system details.

Github link:

https://github.com/ManoharPulakurthi/CS432

Video:

https://drive.google.com/drive/folders/1VQOxvYQFvBoDmYy_NPHgBrODZCi8Mm1F?usp=sharing

Contributions:

Manohar (23110257): prepared the project report, handled video editing, and assisted in explaining the coding logic and implementation tasks to my teammates.

Raj Kamal (23110319): created the demo video for the project and actively contributed to the coding tasks. Additionally, I collaborated with the team to ensure the successful implementation of the required modules.

Avinash (23110123): focused on writing the project code and played a key role in understanding, analyzing, and clarifying the assignment requirements for the team.