CS432 – Module 3 Task

Database Integration and Secure API Development

Group 14: MesSQL

Anura Mantri - 22110144

Neerja Kasture- 22110165

Sawale Sumeet Shivaji - 22110234

Yash Patkar - 22110296

This report details the implementation of the Dinewell Mess Management System, a Flask-based web application designed to streamline mess operations for Admin, Employee, Student, and Council Member roles. It integrates a local database (cs432g14) with a centralized database (cs432cims), incorporating robust session validation and secure APIs. The report addresses the project-specific table design, CIMS database integration, session validation for data security, and a demonstration of working APIs, while embedding relevant details from the provided UML and ER documentation to provide a comprehensive view of the system's design and functionality.

1. Design of Project-Specific Tables

The database schema was crafted based on the UML Class Diagram and ER Diagram from the previous assignment. The tables in the cs432g14 database were designed to support the distinct functionalities of Admin, Employee, Student, and Council Member roles, and integrate with the tables in cs432cims database as outlined below:

- **Breakfast_menu**, **Lunch_menu**, **Dinner_menu**, **Snacks_menu**: The primary keys are month and day. They store all the information as per the IITGN mess menu. The mess council updates this.
- **Inventory**: Tracks groceries with ItemId, Current_quantity, Min_quantity_req, Unit, Date_of_order, Expiry_date, and TotalCost. It supports Admin tasks like removing expired items and receiving stock alerts. The admin updates this.
- **Vendors**: The primary key is item id and transaction id. It keeps track of which vendor an item was ordered from, the vendor details, quantity ordered, amount paid etc. Transaction ID is a foreign key from the central payments table.

- **Income** : The income of the mess through payments made by students. The primary key is meal and transaction ID. The admin updates this.

- **EmployeeSalary**: Records salary details with MemberID (foreign key), Salary, and Month, supporting the UML's EmployeeSalary class for attendance-based payroll, auto-updated via employee punch-in/out. The employee updates this.
- **Feedback**: Stores student complaints and feedback, with fields like `FeedbackID`, `MemberID`, and `Description`, as per the UML's `Feedback` class, enabling Admin review and resolution.
- **Utilities**: Tracks miscellaneous expenses (e.g., electricity, maintenance) based on name, transaction ID and amount paid. The admin updates this.
- **G14_AuditLogs**: Whenever an API call modifies data in the centralized CIMS database, the changes are logged server side in this table.

Apart from this we use the following tables in the central CIMS database:

- **Members**: References through internal ID, stores information like username, dob, email of a member.
- **Payments**: Stores the sender, receiver, date and transaction ID.
- **Images**: Stores the profile image of a member
- **Login**: Stores the member ID, password, session, expiry, role. Helps in session validation and role based access control.
- **Member Group mapping**: Stores the member ID and his/her group ID to keep track of the project each member belongs to and avoid conflicts.
- **G14_revenue**: Keeps track of monthly financials including the income through Income table and expenditure through vendors, utilities etc.

2. Integration with the CIMS Database

Our application is tightly integrated with the **centralized CIMS MySQL database**, allowing seamless user management and payment tracking. This integration supports **relational consistency**, **controlled user deletion**, and **modular table updates** while preventing data leaks or orphan records. This is seen at various points.

We have two database configurations for cs432cims and cs432g14 databases and make connections with both to update them.

Key Integration Points:

- **Add / delete members**: Add a member to our group 14 in this table when the admin is adding user. When deleting a user, make sure we do not delete it for any other group. All apis check the group of the member before allowing access using the is_member_of_group() function.

- **Integration with payments :** All the tables which are related to payment (vendors, utilities, income, employeesalary) transactions also update this table in the CIMS database.
- **Login:** The login table in CIMS database is updated with the id, role and token of the member logging in. This also helps in session validation and role based user access.
- **G14_Revenue**: This is the table of the database that provides the final track record of monthly expenses and profits and is updated automatically.

3. Session Validation and Data Leak Prevention

The system implements strong session handling and role validation.

1. The create_token function is used to generate a JWT token when a user successfully logs in. The JWT is created with the following payload:
    a. user_id: The unique identifier of the user.
    b. role: The user's role (e.g., admin, member).
    c. exp: The expiration timestamp of the token (set to 24 hours from the time of creation).

    This is encoded using the HS256 algorithm.

2. Token validation decorator:

    In session validation, the token_required decorator ensures that a valid and authorized session token is provided before allowing access to a specific route or resource

3. Validation API
    a. The code starts by extracting the session token from the request. This token is retrieved from either the URL query parameters or from cookies.
    b. If the token is absent (neither in the URL nor cookies), a response is returned to the client indicating that the token is missing, along with a 400 Bad Request HTTP status.
    c. Once the session token is retrieved, the function queries the database login table to find the session record associated with the token
    d. The session's expiry timestamp is compared to the current time, which is retrieved using datetime.datetime.utcnow().timestamp(). If the session has expired (i.e., the expiry time is earlier than the current timestamp), a 401 Unauthorized response is returned with the error message "Session expired."
    e. If the session token is found to be valid and not expired, the function returns a 200 OK status with a JSON response indicating that the session is valid ({"valid": True}). This confirms to the user that their session is active and can be used for further interactions

Thus we ensure no data leaks by checking the following:

1. **Session Token Integrity**
   The session token is a critical piece of data.By validating the session token in every API request, the application ensures that unauthorized users cannot access sensitive data.

2. **Prevention of Stale Sessions**
   The expiry check ensures that tokens that should no longer be valid due to session timeouts are rejected, which prevents a security hole where an attacker might try to reuse old session tokens. Even if an attacker gains access to a session token, they can only use it within the valid time frame. After the expiration, they are locked out.

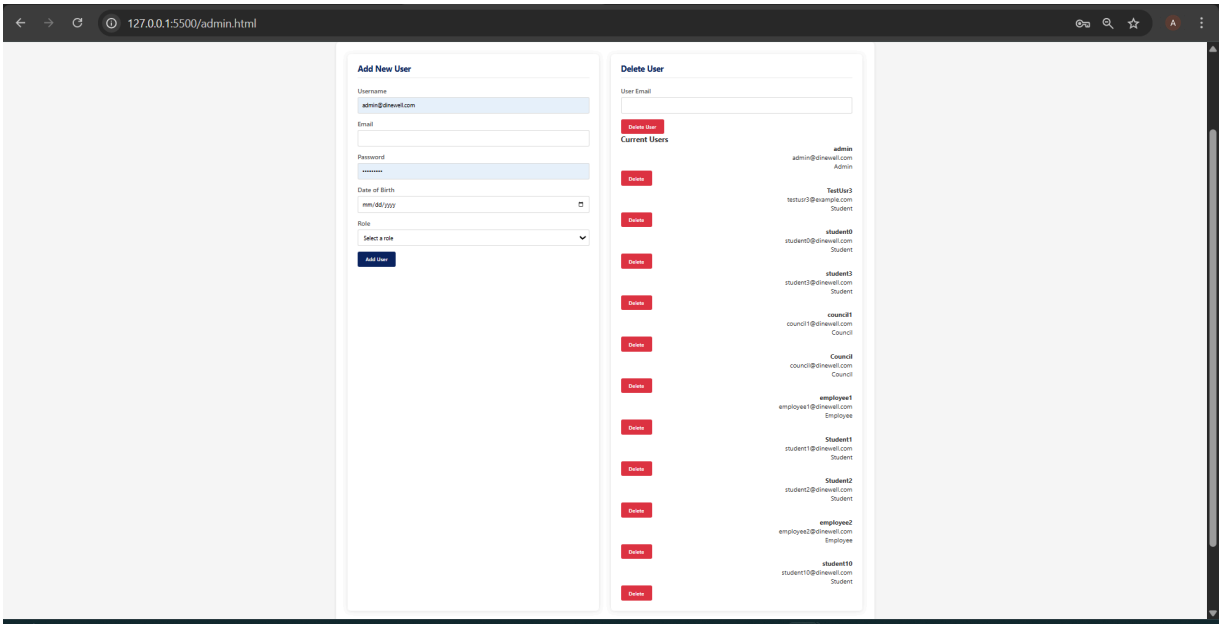
3. **Database Query for Session Lookup**
   By querying the database to validate session details, this ensures that only records stored in the centralized database (and controlled by the app's logic) are accepted. This avoids the risk of relying on potentially insecure or incomplete session data stored elsewhere.

Data Security Measures

- Passwords hashed via MD5
- All APIs require token validation
- Role-based access control is strictly enforced
- Errors logged to app.log as well as the G14_AuditLogs table on server for auditing

4 Portfolio Management

The project specific users are stored into the member group mapping at the time of their creation. The list of the users specific to the project can be seen on the Admin page



5 API descriptions

1. General purpose APIs

API Endpoint	Method	Description
`/login`	POST	Authenticates the user using email and password. Returns a session token upon success.
`/logout`	POST	Clears the session token and logs out the user.
`/ping`	GET	Health check endpoint. Returns `pong` to indicate server availability.
`/isValidSession`	GET	Validates the session token stored in cookies or provided as a query parameter.

2. Admin APIs

API Endpoint	Method	Description
`/admin/users`	GET	Retrieve a list of all users in group 14.

`/admin/addUser`	POST	Adds a new user (Admin, Council, Student, or Employee) and initializes salary if employee.
`/admin/deleteUser`	DELETE	Deletes a user completely or removes them from group 14, depending on their memberships.
`/admin/inventory/remove_expired`	PUT	Sets `Current_quantity = 0` for all expired inventory items.
`/admin/salaries`	GET	Retrieves the salary records of all employees.
`/admin/alerts`	GET	Returns inventory items with quantity below the minimum required threshold.
`/admin/inventory`	GET	Fetches all inventory records.
`/admin/orders`	GET	Returns a list of all vendor orders.
`/admin/place_order`	POST	Records a vendor order, payment transaction, and updates inventory and revenue.
`/admin/inventory/update_quantity/<item_id>`	PUT	Reduces `Current_quantity` for a given item based on expiry order.
`/admin/punch_in/<employee_id>`	POST	Increments the working days of an employee for the current month.
`/admin/add_income`	POST	Adds a student payment to the income table and updates group revenue.
`/admin/add_utility`	POST	Adds a utility expense and updates group revenue.

`/admin/revenue`	GET	Retrieves consolidated revenue data including income, expenses, and salaries.
`/admin/list_utilities`	GET	Lists all recorded utilities.

3. Employee API

API Endpoint	Method	Description
`/employee/salary`	GET	Retrieves the authenticated employee's salary details, including attendance and per-day salary.

4. Council Member APIs

API Endpoint	Method	Description
`/council/menu/update/<meal_type>`	PUT	Updates the menu for a given meal (`breakfast`, `lunch`, `dinner`, or `snacks`).
`/complaint`	GET	Retrieves all complaints submitted by students or users.
`/feedback`	GET	Retrieves all feedback submitted by users.

5. Common APIs

API Endpoint	Method	Description
`/menu/<meal_type>`	GET	Fetches menu items for a specified meal type.
`/feedback`	POST	Allows any user to submit feedback.
`/complaint`	POST	Allows any user to raise a complaint.

6 Testing the APIs

The server is started using **“python3 main.py”** and then the test script is run using **“python3 test_api.py”**

Each API is tested and gives the following response

API Endpoint	HTTP Method	Status Code	Meaning
`/login`	POST	200	Admin login successful
`/ping`	GET	200	Server is online
`/isValidSession`	GET	200	Session is valid
`/admin/users`	GET	200	Admin successfully fetched user list
`/admin/addUser`	POST	201	New user added successfully
`/admin/deleteUser`	DELETE	200	User deleted or removed from group 14
`/admin/inventory/remove_expired`	PUT	200	Expired inventory removed
`/admin/salaries`	GET	200	Employee salaries retrieved
`/admin/add_salary`	POST	201	(Possibly custom or missing from main.py) Salary added successfully
`/admin/inventory`	GET	200	Inventory fetched successfully
`/admin/orders`	GET	200	Vendor orders retrieved
`/admin/place_order`	POST	201	Order placed and inventory updated
`/admin/inventory/update_quantity/1002`	PUT	200	Quantity updated successfully
`/admin/punch_in/2260`	POST	200	Punch-in recorded
`/admin/add_income`	POST	201	Income recorded and linked to transaction
`/menu/lunch`	GET	200	Lunch menu retrieved
`/admin/add_utility`	POST	201	Utility added and revenue updated

/admin/revenue	GET	200	Revenue data fetched
/admin/list_utilities	GET	200	Utility list fetched
/feedback	POST	201	Feedback submitted
/complaint	POST	201	Complaint raised
/admin/alerts	GET	200	Low inventory alerts fetched
/logout	POST	200	Admin logged out
/login	POST	200	Employee login successful
/ping	GET	200	Server responsive
/isValidSession	GET	200	Employee session valid
/employee/salary	GET	200	Salary details fetched
/menu/lunch	GET	200	Menu visible to employee
/logout	POST	200	Employee logged out
/login	POST	200	Council login successful
/ping	GET	200	Server responsive
/isValidSession	GET	200	Council session valid
/council/menu/update/lunch	PUT	201	Menu updated by council
/menu/lunch	GET	200	Council viewed lunch menu
/logout	POST	200	Council logged out
/login	POST	200	Student login successful
/ping	GET	200	Server responsive
/isValidSession	GET	200	Student session valid
/menu/lunch	GET	200	Student viewed lunch menu

`/logout`	POST	200	Student logged out

7 UI Development

The UI was made using html and javascript and hosted for testing and demo on localhost. The link for the UI demo and Github source code are added below:

Github link:https://github.com/NeerjaKasture/Dinewell-Database-API/tree/UI-Development
Video Demonstration of UI: https://youtu.be/oazlcrDmGlU

8 Conclusion

The Dinewell Mess Management System is a robust, secure, and role-driven web platform built upon the structural principles outlined in the UML and ER diagrams. It ensures centralized validation, maintains data integrity, and provides a user friendly interface.

The integration with the centralized CIMS database ensures data consistency and secure role verification. Each user role is isolated and empowered according to its responsibilities, which is validated through access-controlled APIs. Moreover, session security using JWTs and hashed credentials prevent data leaks and unauthorized access.

The successful implementation of these features ensures that Dinewell can serve as a reliable solution in a real-world academic mess environment. The project lays the groundwork for expanding into a fully featured food service management system across campuses, offering potential for real-time QR attendance, inventory analysis, and intelligent revenue tracking modules in the future.

9 Member Contribution:

Sumeet- API development and Testing
Yash- API Testing and Documentation
Neerja- Table creation and Documentation
Anura- UI development