CS 432: Databases - Prof. Yogesh K Meena

Module 3: Database Integration and Secure API Development

Team-1 (DataCrazies):

Aditya Mehta (22110017) *Jiya Desai (22110107)*
Kishan Ved (22110122) *Nimitt (22110169)*
Pratham Sharda (22110203)

Introduction

This report presents our implementation of the database integration, secure API development and UI creation for module 3 of the course project. So far, we have connected our project-specific database (CampusTrade database) to the Central Information Management System (CIMS); implemented API calls for various actions: member creation and authentication, along with various project-specific actions as described in the upcoming sections; used secure authentication mechanisms, role-based access control, and ensured data integrity across the system while maintaining proper logging of all transactions, both locally and on the server. We have also created a user interface for the system.

Tools Used

For Frontend: ReactJS
For Backend: Python Flask for API Development
Database: MySQL Common Remote Database hosted on the web server at this link
API Testing: Thunder Client (VS Code Extension)

Database Design

The following tables were created in the group-specific database:

- Member (cims members table and another memberExt table for extra attributes)
- category
- product listing
- transaction listing
- credit logs
- wishlist
- reviews ratings
- report analytics
- complaints
- Api-logs

Member Lifecycle and RBAC

- **Member Creation:** Creating records in `members, login, memberExt, and MemberGroupMapping.`
- **Member functionality** - they can directly register as members with three sub-roles (student, staff or faculty) and avail the functionality as mentioned in the video.
- **RBAC:** Role is stored in Login. Admins can delete members; users are limited to personal functions. Also, in our database design, even the members have different roles (student, faculty and staff). This is not to be confused with the admin and member roles as defined in the module.
- **Admin login:** Assuming the admin(s) already have entries in the global members table, they will have an option of logging in as the admin and deleting members, viewing all table records, etc.
- **Member Deletion:** Admin functionality - validates group mapping before deletion

---

API Functionality and Backend Architecture

All endpoints are built with Flask and protected using JWT tokens validated against CIMS's Login table. Here is a small gist of all APIs and their functionality in short:

- `/` (GET): Returns a simple message to confirm the API is running.
- `/isAuth` (GET): Verifies if the provided JWT token is valid and returns the user ID if authenticated.
- `/addMember` (POST): Registers a new member with profile details and creates login credentials.
- `/addAdmin` (POST): Registers a new admin user and creates login credentials.
- `/login` (POST): Authenticates a user (member or admin) and returns a JWT token on successful login.
- `/addToWishlist` (POST): Adds a product to the authenticated user's wishlist.
- `/deletefromwishlist` (DELETE): Removes a product from the authenticated user's wishlist.
- `/getWishlist` (GET): Retrieves all products in the authenticated user's wishlist.
- `/deleteMember` (DELETE): Deletes a member (admin only), including all related records.
- `/addProduct` (POST): Allows an authenticated user to list a new product for sale, including image upload.
- `/getCategories` (GET): Fetches all product categories available in the system.
- `/getProducts` (GET): Retrieves all products not listed by the authenticated user.
- `/getProduct/<product_id>` (GET): Fetches details of a specific product by its ID.
- `/updateProduct/<product_id>` (PUT): Updates details of a specific product (seller only).
- `/addComplaint` (POST): Allows a user to file a complaint with a description.

- `/viewComplaints` (GET): Retrieves all complaints filed by the authenticated user.
- `/toggleComplaintStatus` (POST): Toggles the status (Open/Resolved) of a user's own complaint.
- `/viewGroupMembers` (GET): Lists all group members with non-sensitive details.
- `/getMemberListing` (POST): Retrieves all product listings by a specific member.
- `/buyProduct` (POST): Processes the purchase of a product, updates balances, and records the transaction.
- `/getMyTransactions` (GET): Retrieves all transactions (bought and sold) for the authenticated user.
- `/addReview` (POST): Allows a user to submit a review and rating for another user.
- `/getMyReviews` (GET): Retrieves all reviews written by or about the authenticated user.
- `/myListings` (GET): Lists all products currently listed for sale by the authenticated user.
- `/getMyCreditLogs` (GET): Retrieves the credit log (balance history) for the authenticated user.
- `/profile` (GET): Fetches the authenticated user's profile, including stats and profile image.
- `/getReportAnalytics` (POST): Generates and retrieves analytics reports (e.g., seller sales and revenue summaries).

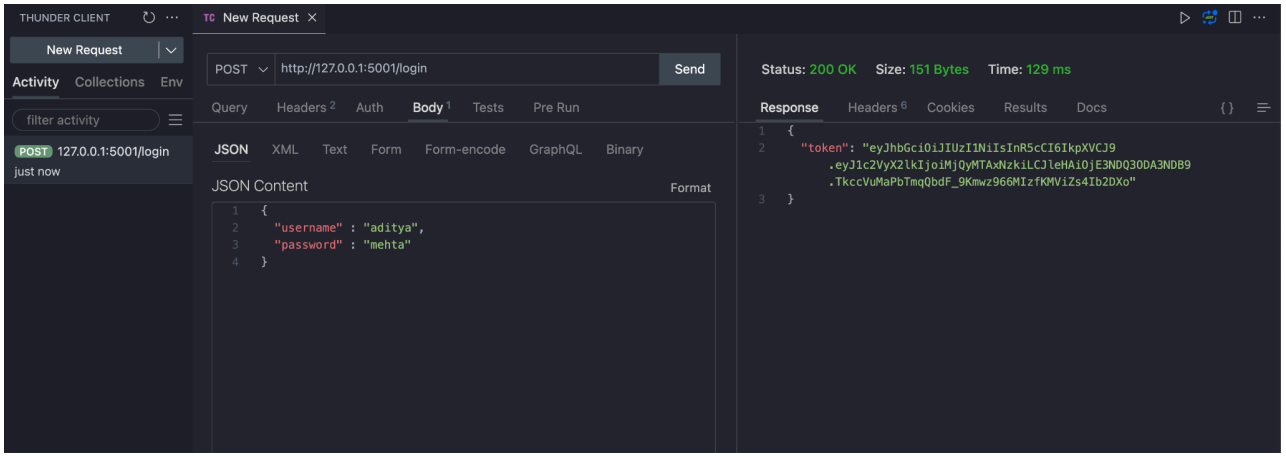
A Sample API Snippet with Backend Code:

```
# Get All Products endpoint
@app.route('/getProducts', methods=['GET'])
@token_required
def get_products():
    try:
        conn2 = get_db_connection(cims=False)
        cursor2 = conn2.cursor(dictionary=True)
        cursor2.execute("SELECT * FROM product_listing")
        products = cursor2.fetchall()

        return jsonify({'products': products}), 200
    except Exception as e:
        return jsonify({'error': str(e)}), 500
    finally:
        if 'cursor2' in locals():
            cursor2.close()
        if 'conn2' in locals():
            conn2.close()
```

Here, the @token required decorator ensures user authentication and session validation for every call of the /getProducts API (which fetches all available listed products form the database)

Testing the Backend with Thunder Client

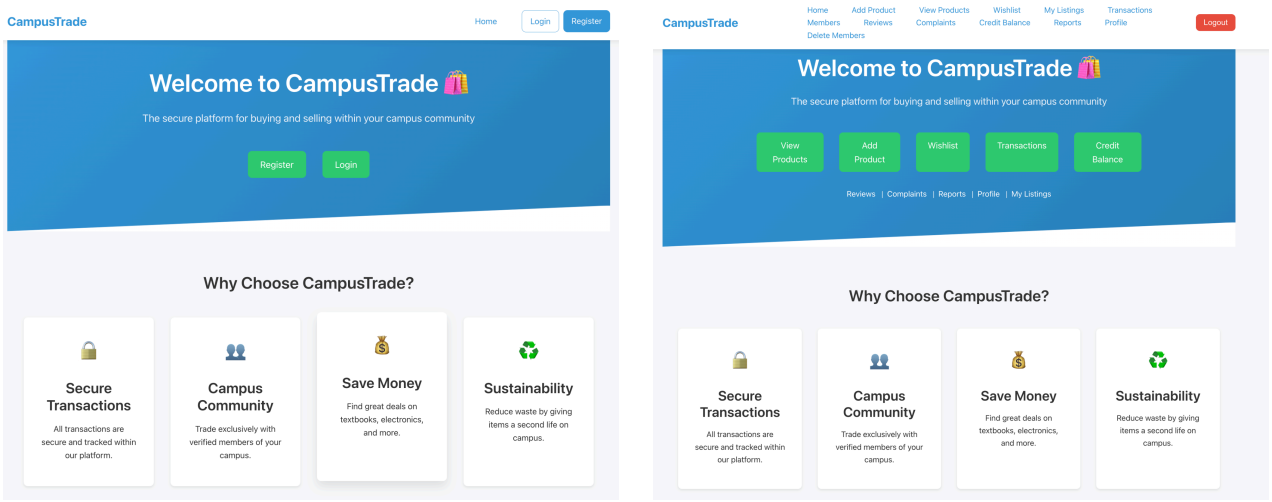


Frontend Overview: React Application

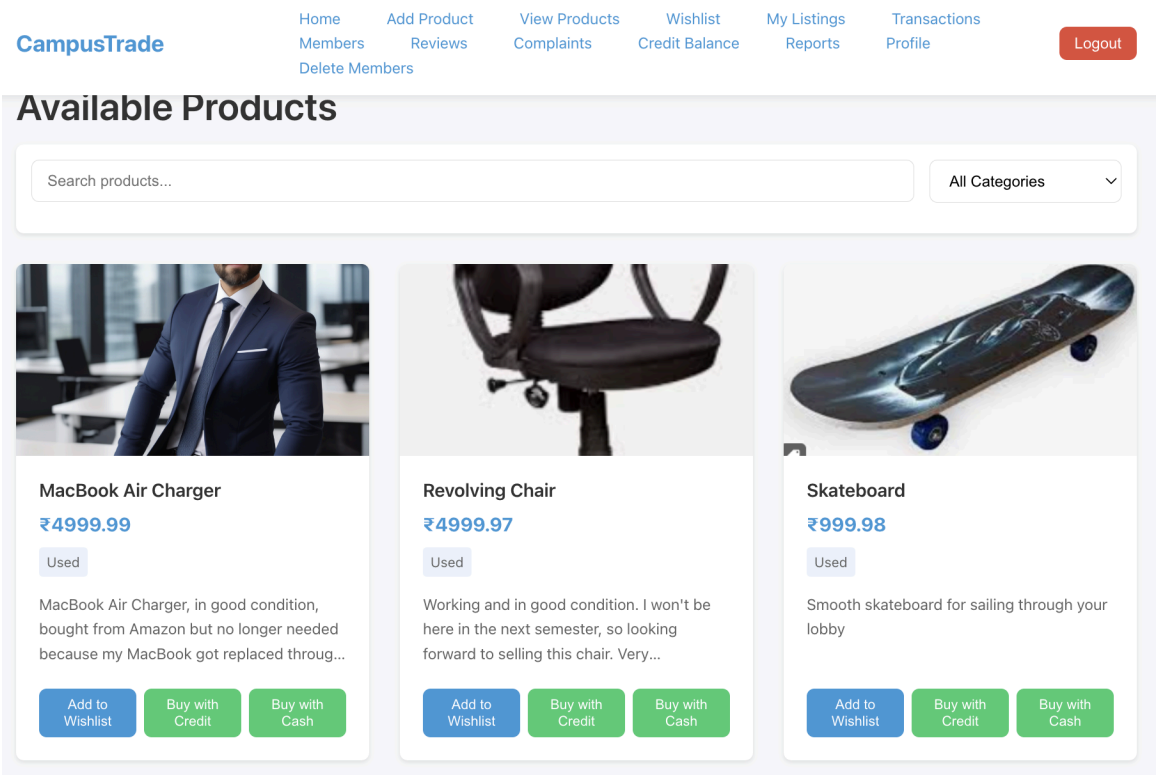
Our UI allows:

- Member Registration
- Secure login for registered members
- Add/view/edit/delete product listings
- Buy products with either cash or credit
- View and manage the wishlist
- File and view complaints
- Submit and see reviews
- See credits and transaction history
- User Profile Dashboard
- View other members in the organization and their listings

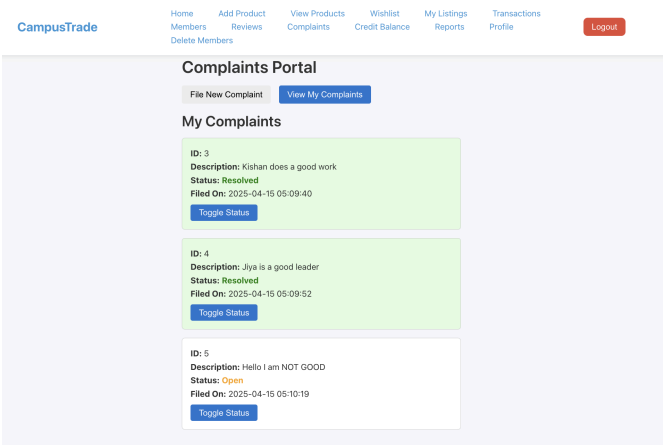
UI Showcase:



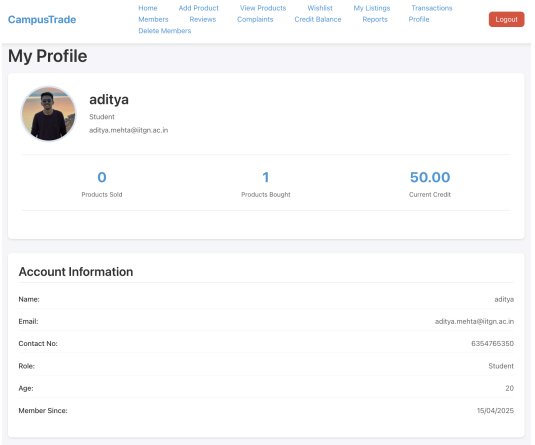
UI for the Frontend, before and after Login



View Products listed by others



View my complaints



View my profile dashboard

There are many more functionalities and pages as can be seen in the top Header.
All of them are explained in detail in the youtube video (link given below)

Security, RBAC and Logging:

Session tokens are stored and validated from the CIMS Login table

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
      .eyJ1c2VyX2lkIjoiMjQyMTAxNzkiLCJleHAiOjE3NDQ3ODA3NDB9
      .TkccVuMaPbTmqQbdF_9Kmwz966MIzfKMViZs4Ib2DXo"
}
```

RBAC is enforced at API level using decorators. Here's an example code snippet

```
def token_required(f):
  @wraps(f)
  def decorated(*args, **kwargs):
    token = request.headers.get('Authorization')
    ...
    return f(*args, **kwargs)
  return decorated
```

Logs capture all write operations and any unauthorized access attempts. Logs are stored locally and on the server in api-logs table (in cs432g1 database)

```
2025-04-14 17:12:43 - [INFO] POST /addProduct - User m452
2025-04-14 17:18:01 - [ERROR] Unauthorized access attempt detected
2025-04-14 17:21:10 - [INFO] Complaint submitted by m231
```

The Delete Member API checks the member ID and using that, the corresponding role of the user. Only users with admin roles are allowed to delete members. Currently, only for testing purposes, delete member option is shown in the header for both the user and the admin, later, we can show it only for the admin.

Using CIMS tables

We have used members, login and G1_report_analytics tables from the CIMS database. Other tables are created in our G1 database.

Password Security

The passwords entered by the user are encrypted using md5 hashing technique. The hashes are stored in the members table on cims database. Additionally, only for testing purposes, we store the unhashed password in memberExt table, that column can easily be dropped before deployment.

Demonstration and Github Repository:

Youtube Video Link: https://youtu.be/HXZwy3-DJIM
A short demo video walkthrough is included with our submission. Covers login, product posting, transaction, complaint and reporting, RBAC, and additional functionalities.

Github Repository Link: https://github.com/Kishan-Ved/CampusTrade/tree/main
Includes complete backend code (Flask), React frontend, SQL table creation scripts, Screenshots, demo video, and logs.

Individual Contributions

- **Aditya Mehta (22110017)**: Designed the Member Listing Functionality, Complaints Portal and User Profile Management (Backend & Frontend) and created the report.
- **Jiya Desai (22110107)**: Designed APIs for Reviews, Logging, Transactions and Analytics part of the Project. Also created the report and the YouTube demonstration video.
- **Kishan Ved (22110122)**: Contributed to Backend API development, Addmember, Addadmin, Login, BuyProduct, Debugging and managing the version workflow of the project via CI-CD framework.
- **Nimitt (22110169)**: Set up API testing and Session Management Framework. Implemented APIs: AddProduct and related APIs. Setup UI framework using Node.
- **Pratham Sharda (22110203)**: Designed the UI and the majority of Frontend of the Project. Also contributed to testing and Debugging.