UML Class diagram:

We follow a centralised approach: i.e. merging the requirements for each user view into a single set of requirements for the new database system. A data model representing all user views is created during the database design stage. Oue views were made according to the users- Admin, Employee, Student and council.
The tables were defined as per requirement of the users.

- The admin needed to keep track of revenue for which we had to keep track of Payments made to the mess (through QR and cash both), and payments made by the mess (for buying groceries, to pay employees or the utility expenses).
- The employee has to mark their attendance everyday (assuming daily wages) for deciding their payment at the end of each month.
- The student has to view the menu for the day/month and scan the QR code which is generated separately for each student and has to be scanned at every meal he/she has. The student can also provide his/her feedback or send complaints to the admin.
- The Council member (who is also a student) can update the mess Menu and scan QRs while having meals (same as student).

Based on this functionality, We decided to have the following:
Classes:

1. Members (which contains all the information of all the users; has two subclasses -employee and student which are mandatory and disjoint).
2. Employee (sub-class of Members Indicates whether the member is an employee or not and gives functionality to him/her.)
3. Student (subclass of Members and helps in managing the access of the database and functionalities.)
4. Admin (optional sub-class of Employee) gives extra privileges to the admin for the complete management of the system.
5. CouncilMember(optional subclass of student) gives the mess council member the extra access to update the menu.
6. Payments (which keeps a track of the payments made to the mess in the form of cash/UPI or QR scans during each meal session and appends it to the revenue on a daily basis)
7. Vendors (to keep a track of different orders made to different vendors and the status of these orders)
8. Inventory (this entity will help manage the groceries in the mess. It keeps a track of all the orders made by the admin and keeps track of the consumption. If below a particular threshold, an alert is sent to the admin. An alert is also sent if a particular item expires and is no longer usable for the admin to discard the waste to maintain hygiene)
9. Utilities (this entity keeps track of other miscellaneous expenses of the mess like electricity bill, maintenance costs, etc)

10. EmployeeSalary (this entity keeps track of the attendance of the employees to give them salary at the end of every month. It is auto updated when employees mark their attendance.)
11. Revenue (this is a monthly check of the total revenue generated by the mess)
12. Feedback (this stores all the complaints/ feedback provided by students for the admin to see)
13. Mealplan (this is an encapsulation of the four meal sessions of the mess. Has four mandatory disjoint subclasses mentioned below)
14. Breakfast_menu (stores the items in breakfast and the month and day of the week for the meal)
15. Lunch_menu (stores the items in lunch and the month and day of the week for the meal)
16. Snacks_menu (stores the items in snacks and the month and day of the week for the meal)
17. Dinner_menu (stores the items in dinner and the month and day of the week for the meal)

Relationships:

- Member class is a **generalisation** of student and employee, or student and employee are specialisations of the class member.
- Admin is the **specialisation** of employee class and CouncilMember is specialisation of Student class.
- MealPlan is an **aggregation** of Breakfast_menu, Lunch_menu, Snacks_menu and Dinner_menu. Aggregation represents a “is-part-of” relationship between entity types, where one represents the “whole” and the other the “part.” Here MealPlan is the whole.
- The admin has one to many relations UpdatePayments, UpdateVendor, UpdateInventory, UpdateUtilities with the classes Payments, Vendors, Inventory and Utilities as we take the assumption that there is only one admin as of now and hence he/she can make multiple updates to these tables. This relation is a **composition** in all these cases as there is only one admin and he/she creates, updates and deletes entries in these tables. This gives a sense of ownership to the admin class.
- Apart from that, there are **association** relations as follows:
    1. Student provides feedback
    2. Admin reviews/resolves feedback
    3. Inventory sends alert to admin about the stock and waste disposal
    4. The revenue is updated using relationships PaymentToRev, InventoryToRev, UtilitiesToRev, EmployeeSalaryToRev from the classes Payments, Inventory, Utilities, and EmployeeSalary respectively.
    5. The updates relationship between CouncilMember and Mealplan is used to update the mess menu
    6. The updates relationship between employee and Employeesalary takes care of the attendance of mess employees

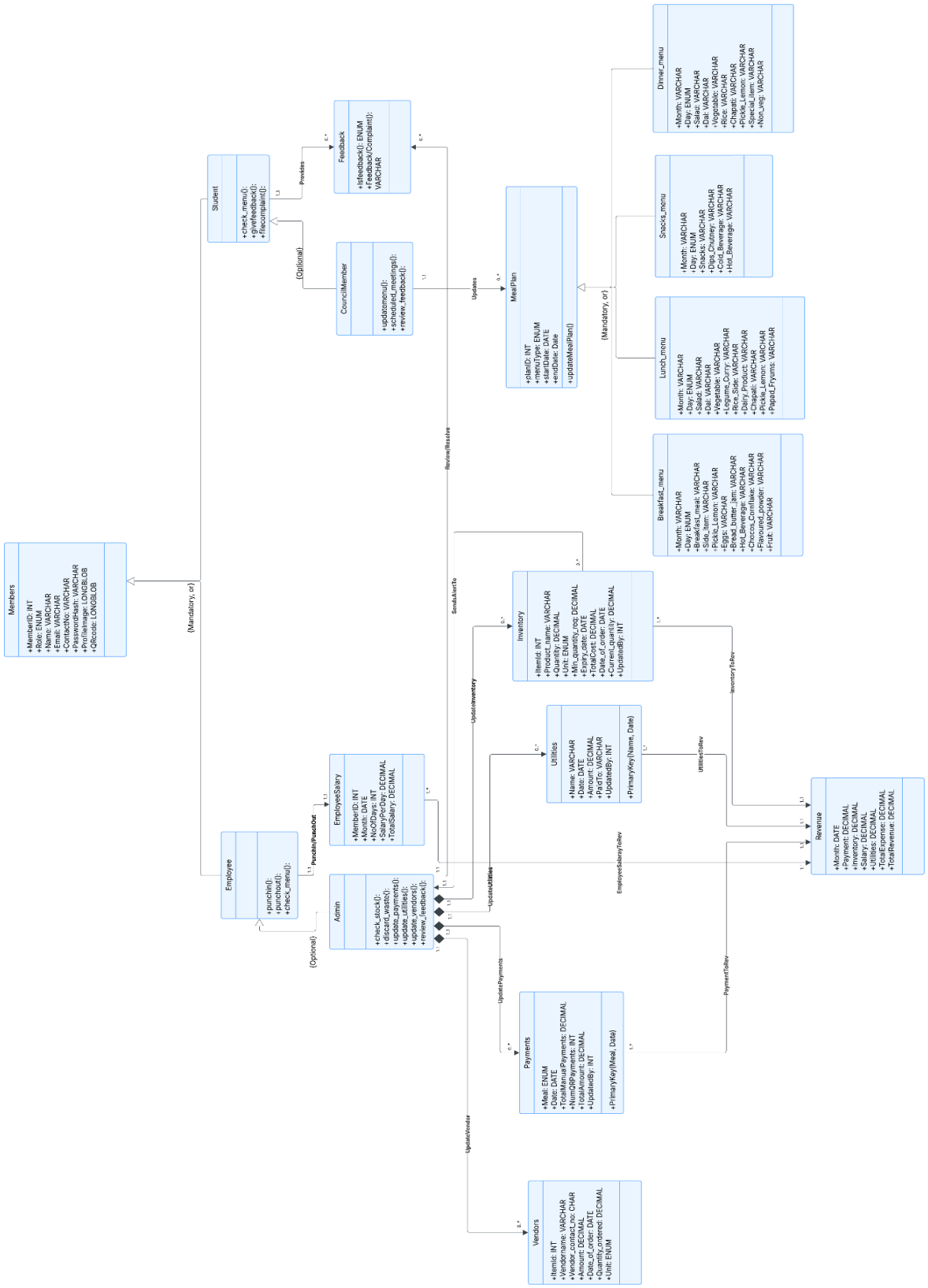
Mapping Cardinalities:

We followed the Pearson cardinality conventions for the UML diagram:

MULTIPLICITY CONSTRAINTS	MEANING
0..1	Zero or one entity occurrence
1..1 (or just 1)	Exactly one entity occurrence
0..* (or just *)	Zero or many entity occurrences
1..*	One or many entity occurrences

**Note: All the attributes are public as of now. We plan to make a more secure system by making some of them private and protected in the future.

UML CLASS DIAGRAM: UML_class_diagram_DINEWELL (3).png



UML Activity Diagram: Activity diagram.png

1. **Users**

   The UML Activity Diagram represents the workflow of a Mess Management System (**Dinewell**) where users are categorized into four distinct roles:

   - Admin
   - Student
   - Employee
   - Mess Council Member

   Each role has specific functionalities, and the diagram illustrates how different users interact with the system from login to logout.

2. **Overview of the Activity Flow**
   - The activity begins with a black circle (initial node), which represents the start of the system.
   - The first action is the login process, which is a decision node that determines the type of user accessing the system.
   - After authentication, the user is directed to their respective functionalities.

3. **Student Activities**

   Once logged in as a Student, the user can perform the following actions:

   - File complaint
   - View Menu
   - View QR Code
   - Provide Feedback

   A decision node checks whether the student wants to continue or logout. If logout is selected, the session ends.

4. **Employee Activities**

   Employees have the following options after login:

   - Punch in/out (for attendance tracking)
   - View Menu
   - Get their QR Code
   - View Salary

   Similar to the Student role, a decision node allows employees to either continue using the system or logout.

5. **Mess Council Member Activities**

Mess Council members are responsible for managing the menu. Their options include:

- View Menu
- Update Menu

A decision node provides the option to logout once tasks are completed.

6. **Admin Activities**

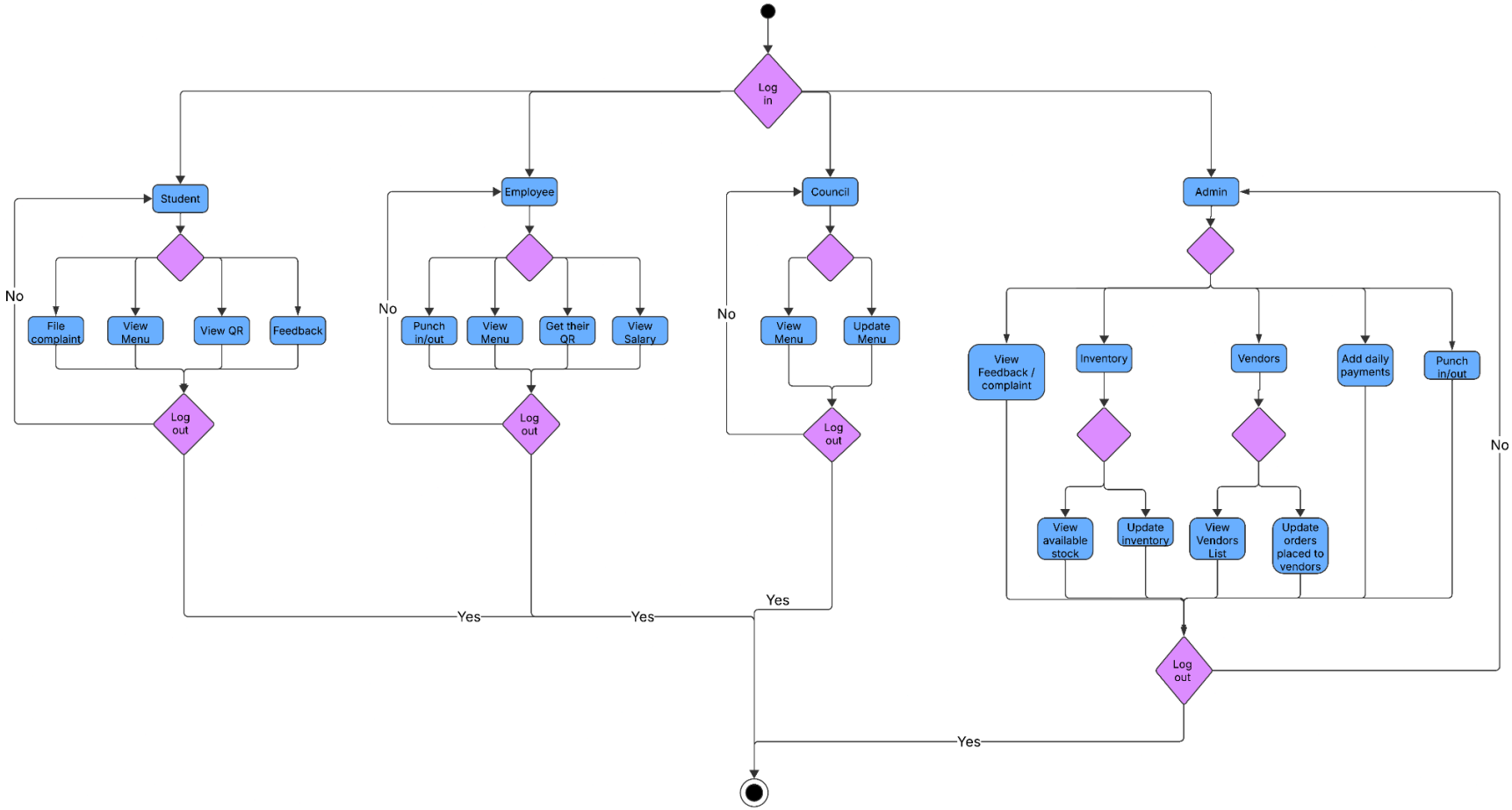
   The Admin role has the most extensive set of functionalities. Admins can:

   - View Feedback/Complaints
   - Manage Inventory
     - View available stock
     - Update inventory
   - Manage Vendors
     - View vendor list
     - Update orders placed to vendors
   - Add Daily Payments
   - Punch in/out

   After performing necessary tasks, the Admin can logout, ending their session.

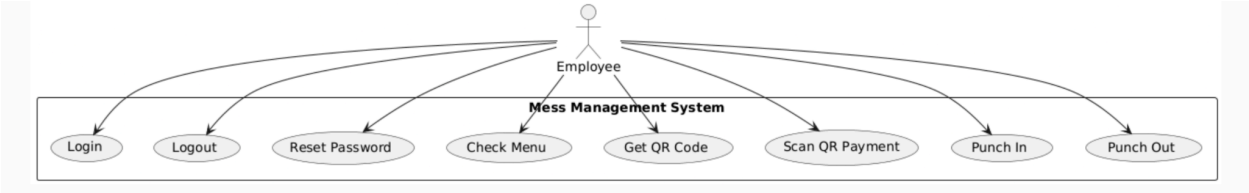
7. **End of Process**

   Once a user completes their tasks and logs out, the process terminates at the **final node** (black circle with an outline).


Log in
Student
File complaint
View Menu
View QR
Feedback
Log out
No
Employee
Punch in/out
View Menu
Get their QR
View Salary
Log out
No
Council
View Menu
Update Menu
Log out
No
Admin
View Feedback / complaint
Inventory
View available stock
Update inventory
Vendors
View Vendors List
Update orders placed to vendors
Add daily payments
Punch in/out
Log out
No
Yes
Yes
Yes
Yes

UML Use-Case Diagram:

1. **Employee Use Case Diagram** Employee_use_case.png

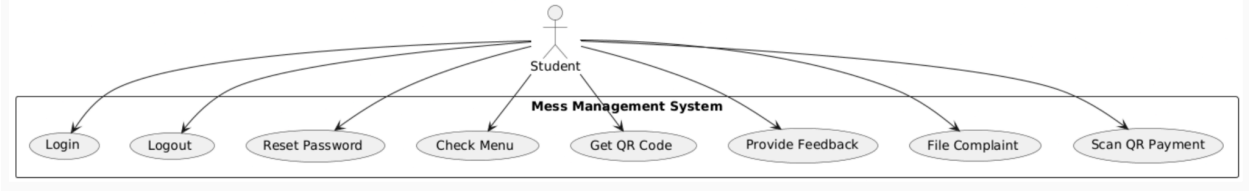


   - Employees can log in, check menus, generate QR codes, scan QR payments, and punch in/out for attendance.
   - Their role is focused on daily operations related to meal access and work hours.

2. **Admin Use Case Diagram** Admin_use_case.png

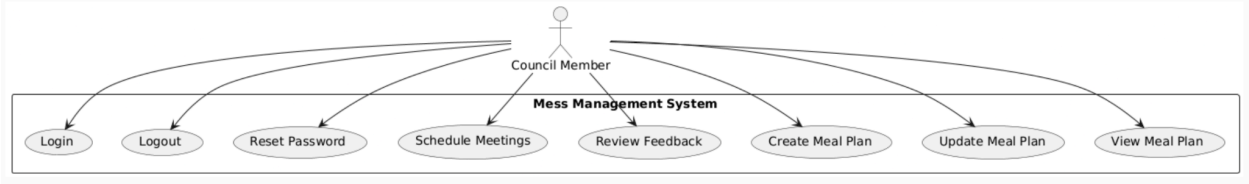
   - Admins have expanded responsibilities beyond employees, including reviewing feedback, managing stock, updating payments, and attendance tracking.
   - Their role ensures smooth mess operations and financial management.

3. **Student Use Case Diagram** Student_use_case.png



   - Students interact with the system to check menus, generate QR codes, provide feedback, and file complaints.
   - Their role is consumer-focused, ensuring they can access meals and report concerns.

4. **Council Member Use Diagram** Council_use_case.png



- Council Members interact with the system to schedule meetings, review feedback, and update meal plans.
- Their role is administrative, ensuring smooth mess operations and addressing student concerns.

UML Deployment diagram Deployment.jpeg

A. Client Devices

This node represents the different users of the system, who interact with the web application through their devices. The users include:

- Admin
- Mess Council Member
- Student
- Employee of the Mess

Each user interacts with the Web Application running on the Server.

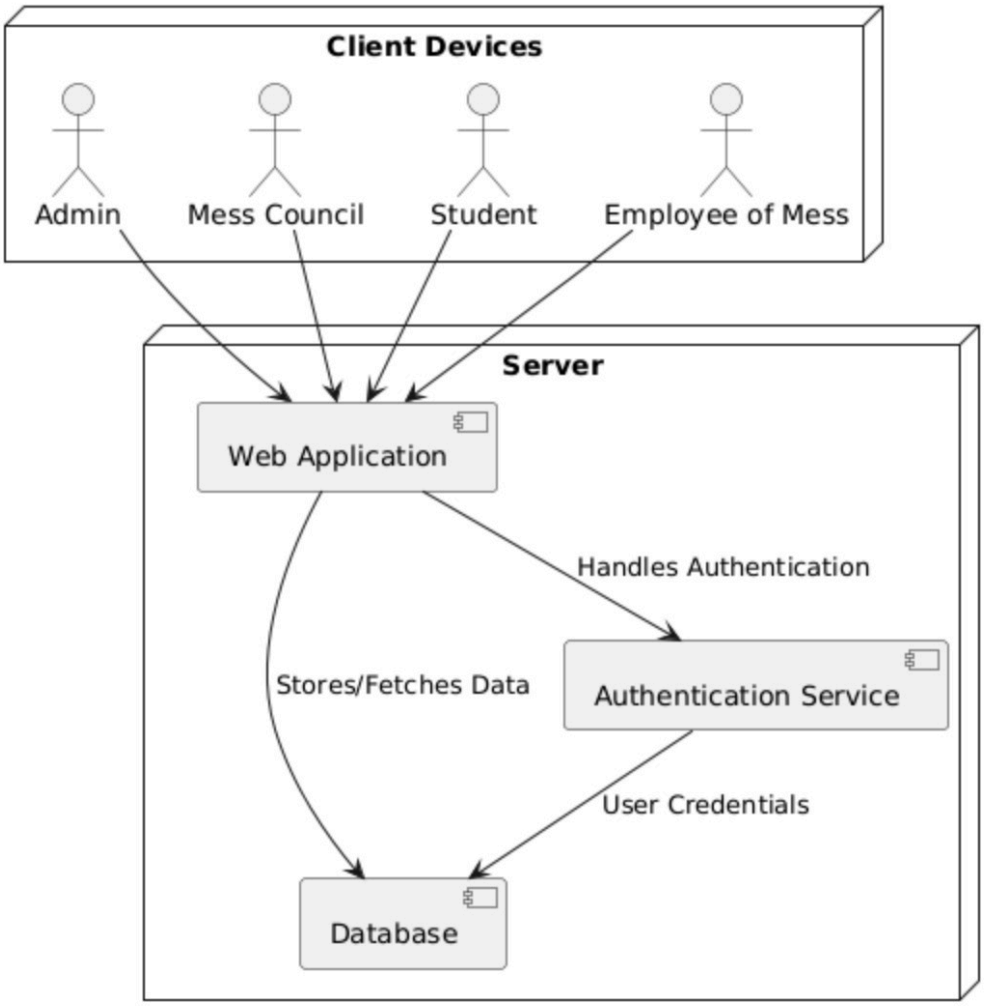
B. Server

The server node hosts the main application and its services. It consists of:

1. Web Application
   - The central component that interacts with all user roles.
   - Handles the main logic of the mess management system.
   - Communicates with the Authentication Service and Database.

2. Authentication Service
   - Responsible for user authentication (login, access control).
   - Verifies credentials stored in the Database.

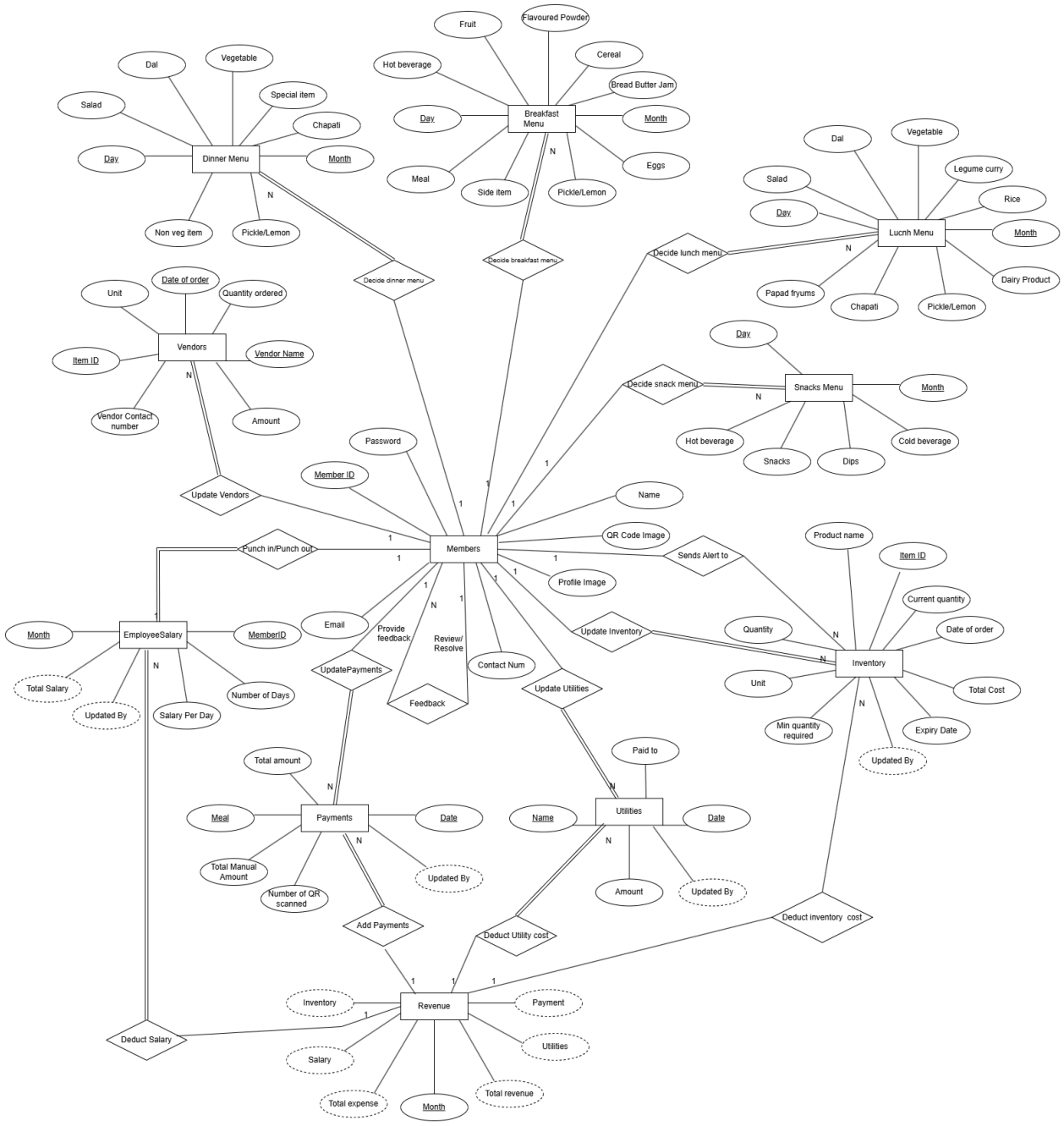
3. Database
   - Stores and retrieves data related to the mess management system.
   - Contains user credentials, inventory data, menu details, employee salaries, and more.
   - Used by both the Web Application (for general data storage) and the Authentication Service (for credentials).


Client Devices
Admin
Mess Council
Student
Employee of Mess
Server
Web Application
Handles Authentication
Stores/Fetches Data
Authentication Service
User Credentials
Database

ER DIAGRAM: Link to diagram

Transition from UML to ER diagram:

1. The classes except the users(Employee, Student, CouncilMember and Admin) were directly converted to Entities.
2. Associations were directly mapped to relationships.
3. The participation and cardinality was finalised through discussions and practical examples.
4. Derived attributes and Primary Keys were discussed by iterating through all attributes and determining the process of making entries to each entity.
5. Recursive relationships were found and added accordingly.
6. MealPlan was excluded from entities as it is an aggregation of 4 other entities(Breakfast_menu, Lunch_menu, Snacks_menu and Dinner_menu), so we decided on skipping MealPlan



This ER diagram was made according to the sample provided in the submission guidelines.

Relationships in the diagram:

One to many relations:

1. UpdatePayments, UpdateVendor, UpdateInventory, UpdateUtilities are one to many as one admin can make multiple additions to the tables to which the relationships map.

2. Admin resolves/reviews the complaints, hence 1 and there may be multiple feedback hence N
3. CouncilMembers can update mealplan. One entry in the mealplan can be made by a single council member hence 1. A particular council member may make multiple entries to the MealPlan, hence N.
4. Add Payment, Deduct inventory cost, Deduct utility cost, Deduct employee salary are one to many as the revenue is updated monthly by deriving the total monthly expense and income from respective entities. Hence each entry to the revenue table is calculated based on many entries in the corresponding tables at the end of every month. So N entries from payments, inventories, utilities, employee salary affect 1 entry in revenue entity.
5. SendAlertTo is many to one as there may be multiple items in the inventory that are out of stock or expired for that day hence N and all these alerts go to admin, hence 1

One to one relations:

1. Employee PunchIn/PunchOut (their attendance) Employeesalary as one employee can mark only his/her attendance, hence it is one to one relation.

Many to many relations:

There are none as of now

Work Distribution:

1. **Sawale Sumeet Shivaji (22110234) -** UML Activity Diagram, Deployment Diagram
2. **Yash Patkar (22110296) -** UML use case diagram
3. **Anura Mantri (22110144) -** UML class diagram
4. **Neerja Kasture (22110165) -** ER Diagram

Everyone made the documentation for their own part.