
**FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY**

**SEMESTER 1 2024/2025**

**BITU 3923 WORKSHOP 2**

**Group 4**

# FINAL REPORT

**SUPERVISOR: Dr. Syahida Binti Mohtar**

**PROJECT TITLE: FTMK FoodShare System**


| NAME | MATRIC NO |
|---|---|
| NURUL ANIS BINTI MAHADI | B032310042 |
| NUR SHAHIRA ATILIA BINTI ZAINUDDIN | B032210369 |
| NUR SYAFIQAH BINTI SHAMSUDIN | B032210207 |
| MOHAMAD IRFAN BIN NOOR EFFENDY | B032210093 |

# TABLE OF CONTENTS

## EXECUTIVE SUMMARY

The FTMK FoodShare System is an innovative project aimed at addressing food insecurity and reducing food waste within the Faculty of Information and Communication Technology (FTMK) at Universiti Teknikal Malaysia Melaka (UTeM). Thus, by integrating various connected modules, including the inventory module, sponsorship module, volunteer management module, and budgeting module, the suggested system would simplify procedures for food contributions, distributes, and other management tasks. It aligns with the United Nations' SDG 2: Zero Hunger by linking surplus food with hungry students, making food sharing on this network both socially and environmentally sustainable.

A thorough inventory management module is included into the system to monitor food supply, reduce waste, and ensure effective distribution. Sponsor contributions are made easier by the sponsorship module, which also acknowledges their support and promotes sincerity. While the budget module makes sure that financial resources are allocated and monitored appropriately, the volunteer module improves task coordination and participation tracking.

The system's ability to promote collaboration between students, volunteers, sponsors, and administrators thereby establishing a community of support centered on tackling food insecurity is one of its key features. With features like live food availability updates, and comprehensive reporting, the FTMK FoodShare System ensures operational effectiveness and openness.

In addition to solving current issues with food security, this project offers other departments and organizations a sustainable and scalable model. Through the use of technology and community involvement, the FTMK FoodShare System creates a standard for making a significant social effect both on and off campus.

# CHAPTER 1: INTRODUCTION

## 1.1 Introduction

With tons of edible food being thrown away every day and millions of people globally lacking continuous access to enough food, food insecurity and waste continue to be major global concerns. Food waste from grocery shops, restaurants, gatherings, and even on campus is a problem that is close to our university life. FMTK FoodShare system is a project designed specifically for FTMK students to address this problem. It allows to share extra food in community and help out other students who might be in need. Some students have difficulty balancing their studies with part-time jobs, have trouble paying for their education, or lack access to regular meals. This FTMK FoodShare System will bridge the gap between lack of food and a lot and be an essential resource for areas facing food shortages. To make sure that wholesome food is made available, a network of volunteers, sponsors, and organizations will collaborate.The goal of FMTK FoodShare system is to create a supportive environment where organizations, instructors, and those with extra may give and those who need a hand can receive, without judgment. In line with SDG 2: Zero Hunger, FMTK FoodShare system supports the worldwide effort to eliminate hunger, achieve food security, and improve nutrition.

## 1.2 Problem Statements

i. The lack of comprehensive system to manage food sourcing, inventory, and distribution, leading to food wastage and inefficiencies in operations.
ii. The absence of non-centralized funding and financial management processes, making it difficult to track donations and allocate funds, resulting in potential misuse of resources.
iii. The lack of sufficient coordination among volunteers, which hampers effective task assignments and decreases volunteer engagement in foodbank activities.

## 1.3 Objectives

i. To develop an integrated system to streamline food sourcing, inventory tracking, and distribution to reduce waste.
ii. To create a centralized financial management subsystem to track donations and allocate resources transparently.
iii. To implement a volunteer management system to enhance coordination, task assignments, and engagement tracking.

## 1.4 Scope of the Project

i. Module to be developed

a. Inventory – by Mohamad Irfan bin Noor Effendy

- Tracking and monitoring food supply: Develop a system to track food inventory, including incoming donations and current stock.
- System alert of low inventory of foods below minimum threshold to make sure the inventory is always in stock for distributions.
- Inventory category to split the foods between meal, snacks or drinks to ensure healthy and fair food distribution.

b. Volunteer – by Nur Shahira Atilia Binti Zainuddin

- Registers and manages volunteer profiles, ensuring that information such as contact details, availability, and FICTS membership is up to date.
- Manages the roles assigned to volunteers for different activities within the system, ensuring that the right individuals are in the right positions.

- Tracks the assignment of volunteers to specific tasks, ensuring their participation is documented and their contributions are monitored.
- Generate yeaarly reports on registered volunteers based on programme and year of study.

c. Sponsorship – by Nurul Anis Binti Mahadi

i. Sponsors Registration and profile management

- Enable sponsors to register, create profiles, sponsor contribution and manage their information
- Sponsors can contribute foods or funds to the foodShare's needs

ii. Donation Tracking

- Assist sponsors to keep track on their contributions and know the current needs to make informed decision about next donation.
- Assist admin to keep track the contributions, number of sponsors and others.
- Display a live, up-to-date list of the current needs, categorized by items funds and others.
- Highlight high priority needs to encourage support where it's most urgent
- Keep a record of all contributions made by sponsors, including the dates, amounts, and types of food and others.

iii. Funding and contribution

- Provide sponsors with flexible contributions methods to encourage both one-time or recurring contributions
- Provide sponsors with flexible contributions non-monetary resources or funds

- Sponsors can get notification from email for the donation status upon their donation successfully.

d. Budget – by Nur Syafiqah Binti Shamsudin

i. Budget Allocation and Calculation
   - Allows the system administrator to allocate a specific budget for different purposes such as inventory purchases, volunteer events, and sponsorship activities.
   - Keeps track of all budget allocations, ensuring transparency and accurate reporting of funds usage.

ii. Expense Management
   - Ensures that expenses do not exceed the allocated funds by providing real-time validations during expense submissions.

iii. Request Management
   - Tracks and monitors budget requests, including their statuses. For example, Pending, Approved, Rejected.
   - Creates a system for prioritising and reviewing requests based on current budget availability.

1. Target User

a. Administrator

b. Volunteers (FICTS members)

c. Sponsors

## 1.5 Project Significances

1. Improve the food bank program of FTMK
    - By creating a computerized food bank system, it would directly improve the organization and operation of the food bank program and be able to afford assisting more FTMK students who are in need of required food. With FTMK FoodShare System, foods can be distributed fairly and efficiently to the students while also keeping track records of all the operations such as food arrangements, inventory or distribution to avoid inefficiencies and food wastage.

2. Community engagements and partnerships
    - FTMK FoodShare System can collaborate with local businesses, community organizations, and alumni, opening doors for contributions, sponsorships, and volunteer support. These partnerships can strengthen the university's ties with the broader community,contributing to the sustainability of the food bank program and increasing the support for the community and university

3. Scalable model system
    - FTMK FoodShare System can become a lasting resource that helps incoming and future students. It serves as a scalable model for other departments looking to implement a similar system. Establishing a food bank program system today can serve as a foundation for additional programs to further supporting the development of all the students of UTeM.

## 1.6 Conclusion

The FTMK FoodShare System is a comprehensive program that simplifies food sharing, financial administration, and volunteer coordination in order to reduce food insecurity and decrease food waste among FTMK students. By establishing a helpful platform that guarantees equitable food distribution, promotes transparency, and strengthens community partnerships, the system supports SDG 2: Zero Hunger. The system successfully lowers inefficiencies and improves cooperation between everyone involved by incorporating modules for budget distribution, volunteer management, inventory tracking, and sponsorship contributions.

Moreover, the inventory module indicates accurate food supply tracking, cutting waste and preserving equitable distribution. While the sponsorship module enables smooth sponsor contributions and clear donation tracking, the volunteer module enhances task coordination and participation. Financial resources are managed by the budget module, which makes sure money is spent wisely and stays within set parameters.

Lastly, the approach encourages collaborations with local businesses and organisations and fosters collaboration, accountability, and sustainability among administrators, volunteers, and sponsors. The FTMK FoodShare System's scalable architecture makes it a viable model for upcoming initiatives, guaranteeing students' long-term assistance and encouraging a neighborhood-based approach to addressing food insecurity. The approach enhances the university's food bank program and has a long-lasting effect on students' wellbeing by cutting waste and increasing operational effectiveness.

# CHAPTER 2: DATABASE SYSTEM METHODOLOGY AND PLANNING

## 2.1 Introduction

Database system mehodology and planning is the process of gathering accurate requirements, analyzing them, designing the data structure, and determining the system's capabilities. Following this, each operation within the modules is implemented. This process is referred to as the Database Life Cycle (DBLC) approach, which serves as the development methodology for creating a database system. The DBLC ensures that the system is developed thoroughly and effectively. The phases of the DBLC are outlined in Section 2.2, Database Methodology. Each stage of the life cycle is completed before moving on to the next, ensuring a structured and systematic approach to database development.

## 2.2 Database Development Methodology

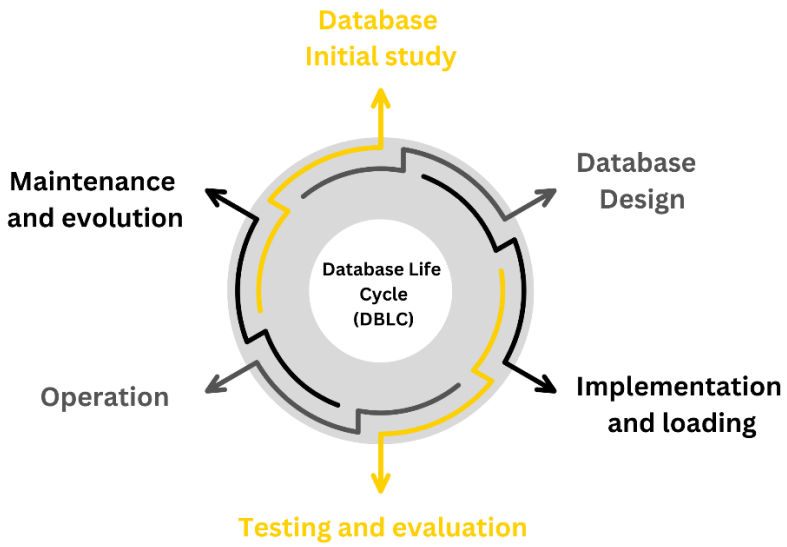



*Figure 2.2. 1 Database Life Cycle (DBLC) Methodology*

i. Database Initial Study

The first stage, Database Initial Study, concerns the need to understand the project requirements and the feasibility regarding database development. This will involve establishing the purpose and objectives of the system, analyzing user needs and data sources, assessing available resources, and preparing a detailed project plan.

ii. Database Design

The second stage, Database Design also covers the elaboration of a blueprint for the database through three sub-phases: conceptual design, during which a high-level data model like an Entity-Relationship Diagram or ERD is produced logical design, where the conceptual model is transformed into a normalized logical schema and physical design, where the logical schema is merged for the specific DBMS, taking into consideration table structures, indexing, and storage.

iii. Implementation and loading

The third stage, Implementation and Loading, focuses on the actual building of the database using the selected DBMS. This includes creating tables, relationships, indexes, and all other objects, including the preliminary population of the database through either data migration or manual input.

iv. Testing and evaluation

The fourth stage, Testing and Evaluation, marks the meeting of performance, security, and accuracy requirements through data integrity checks, testing of queries, load testing, and gathering feedback from its users in order to make refinements.

v. Operation

Once testing is complete, the operation phase, which is the deployment of the database to a live environment. During this stage, users and administrators will be trained on the system, while the running of the system is monitored in an

attempt to provide smooth day to day operations, backup, and support toward availability.

vi. Maintenance and evoluation

Maintenance and Evolution: The final stage is one of operational maintenance and further evolution of the system. Performance monitoring, error correction, or elimination of bottlenecks, updates, data integrity checks, routine backups, and long-term plans for enhancements or integration are part of this evolution process. These stages collectively ensure the database system is reliable, efficient, and aligned with organizational objectives throughout its lifecycle.

## 2.3 Project Management and Requirement

In project management, a requirement is a set of activities or criteria that must be met for the project to be completed effectively. This might refer to product attributes, quality, services, or even procedures. These standards ensure that projects are aligned with the organization's strategic goals.

a. Project Management

1. Project Timeline

- Proposal (Week 1-2): Determine the project's objectives, scope, and feasibility, as well as define the initial development plan.
- Progress 1 (Weeks 3–4): Gather initial requirements, construct a conceptual database design, and validate it with stakeholders.
- Progress 2 (Weeks 5–6): Start logical database design, normalise the schema, and finalise technical requirements.
- Progress 3 (Weeks 7–10): Implement the physical design and begin database building, including tables, relationships, and indexing, while populating the database with sample data.
- Draft Final Report (Week 11-13): Discuss the progress, obstacles, and discoveries from the implementation and initial testing phases.
- Workshop 2 Showcase (Week 14): Demonstrate a functional version of the system, including working modules, to get feedback for final improvements.

- Final Report (Week 15-16): Compile all stages of the project into a complete report that covers the system's approach, design, implementation, and evaluation.

2. Roles and Responsibilities
    - Administrators: In charge of overseeing and maintaining the system, managing database activities, and ensuring that project milestones are met on time.
    - Volunteers: Provide feedback on task management and system usability to ensure that the platform satisfies their requirements.
    - Sponsors: Provide data on cash or food donations to evaluate connectivity with modules such as Budget and Inventory.

3. Tools and Technologies
    - Database tools: DBeaver, MySQL, PostgreSQL, and MariaDB.
    - Development tools: Visual Studio Code, ZeroTier(VPN), and operating systems such as Windows and Linux.
    - Project Management Tools: Draw.io for creating diagrams, such as workflows, database schemas, and system architecture, which aid in project planning and communication.

4. Risk Management
    - Risks include data integrity difficulties, scope creep, and unexpected integration challenges. Mitigation entails doing regular backups, iterative testing, and contingency planning.

b. Requirement
    1. Functional Requirements
        - Inventory Module: Tracks incoming donations, monitors food stock levels, and sends notifications when supplies run short.
        - Volunteer Module: manages volunteer profiles, task assignments,

and participation logs.

- Sponsorship Module: Manages sponsor registrations, tracks donations, and interfaces with the budget to ensure financial transparency.
- Budget Module: Calculates budget allocations, monitors expenses, and connects sponsor and expense data to provide real-time financial tracking.

2. Non-Functional Requirements
    - Performance: the ability to manage several users and requests at the same time.
    - Security: Encrypt and regulate access to sensitive data such as donation details and user profiles.
    - Scalability: The system must be able to handle increasing numbers of users, sponsors, and food donations over time.
    - Availability: Ensure that administrators, sponsors, and volunteers always have access to the system.

3. Hardware Requirements
    - A dedicated server with enough processing power (CPU), memory (RAM), and storage to support the database and application.
    - Network infrastructure to ensure continuous access to the system.

4. Software Requirements
    - Database Management System: MySQL, MariaDB, or PostgreSQL for managing and storing relational data.
    - Development framework such as PHP for backend development. While frontend development uses HTML and CSS to create responsive and user-friendly web interfaces.

5. User Requirements
    - Administrators require tools to manage and monitor all system modules.

- Volunteers require an easy-to-use interface for recording tasks and maintaining participation logs.
- Sponsors want a user-friendly interface for tracking donations and viewing system requirements.

6. Data Requirements
    - Inventory data include food items, expiration dates, and amounts.
    - Financial information (donations, budgetary allocations, and expenses).
    - User profiles (sponsors, volunteers, and administrators).
    - System logs (user activity and transaction history).

7. Legal and Ethical Requirements
    - Compliance with laws that protect data, such as GDPR or PDPA, is required to secure user privacy and security.
    - Ethical considerations include ensuring transparency in donations and financial transactions.

## 2.4 Project Milestone

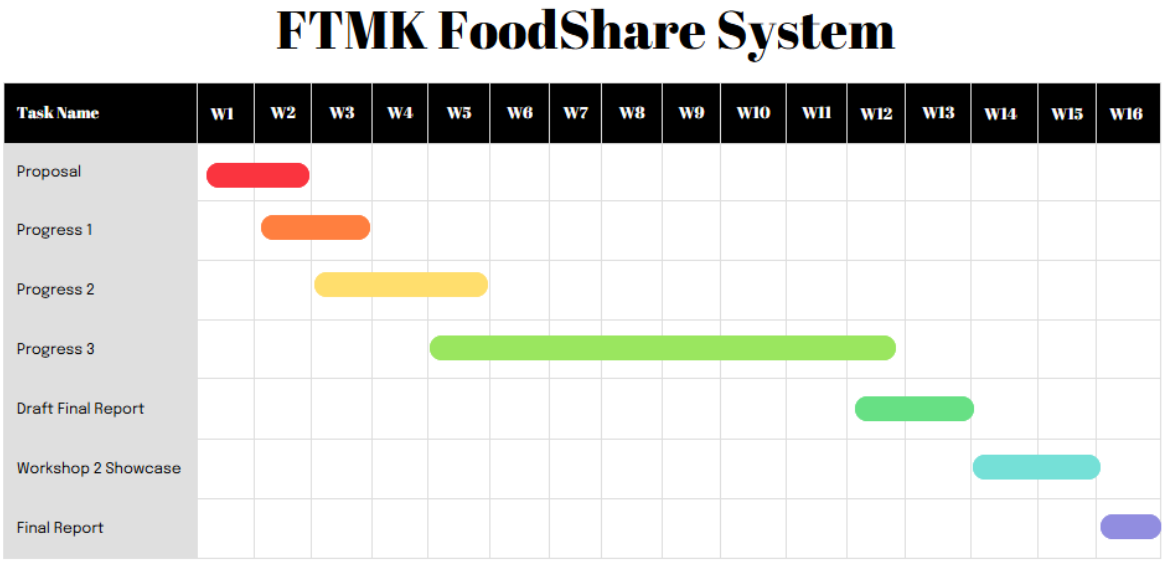



*Figure 2.4. 1 Gantt Chart*

# CHAPTER 3: DATABASE SYSTEM AND ANALYSIS

## 3.1 Introduction

System analysis is a method of understanding and evaluating a system to identify its components, workflows, strengths, and weaknesses. This analysis provides a foundation for improving or developing the database system for the food bank. By breaking down the current manual process, we aim to highlight inefficiencies and areas of improvement. This chapter will present the analysis of the current manual system used in the food bank operations and describe the new system to be developed. Visual aids such as Data Flow Diagrams (DFD) and Context Diagrams (CD) will be utilized to represent the process flow of both the current manual system and the proposed database system.

## 3.2 Current System Analysis

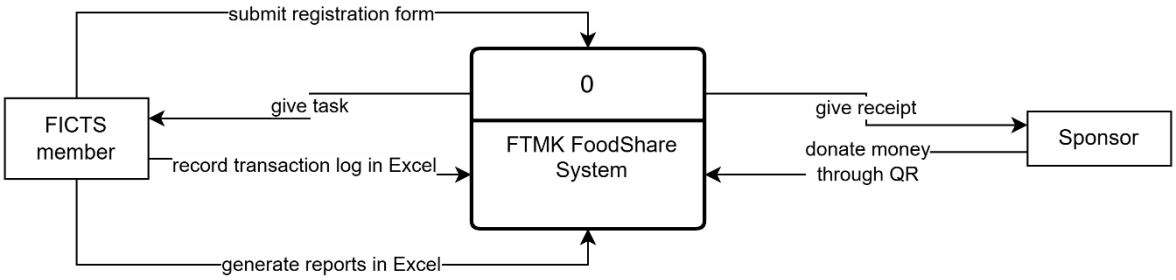



*Figure 3.2.1 Context diagram of manual system*

## 3.3 System To-Be Analysis

### 3.3.1 Context Diagram

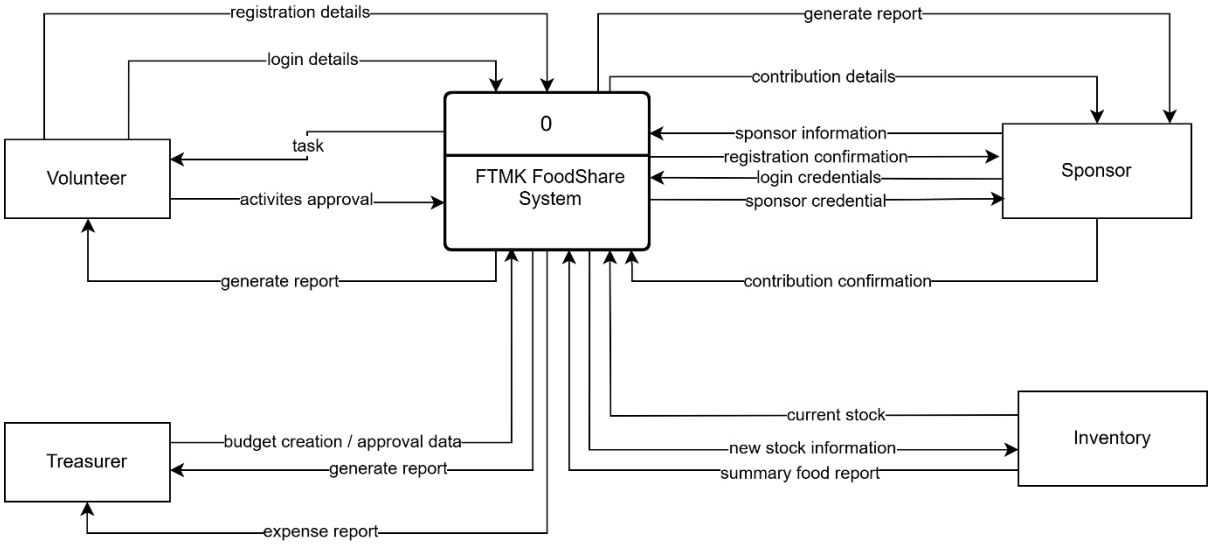



*Figure 3.3.1.1 Context diagram*

### 3.3.2 Data Flow Diagram

#### 1. Data Flow Diagram level -1

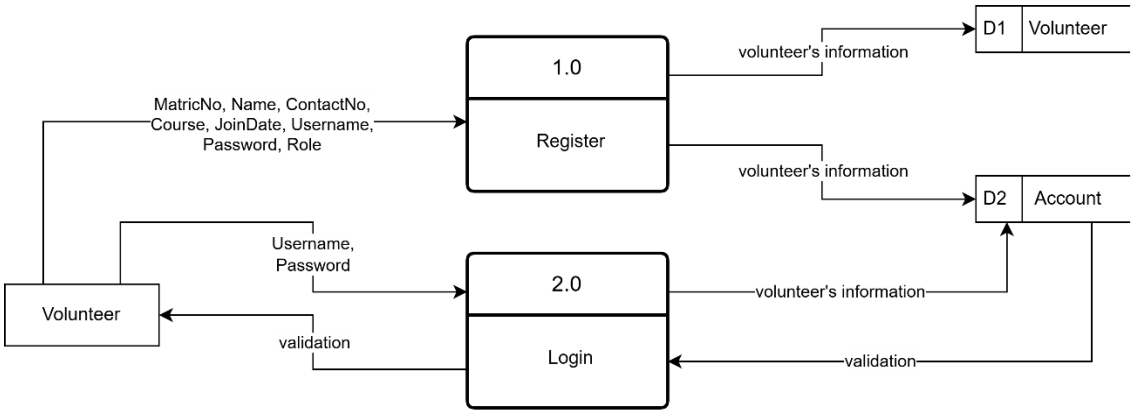



*Figure 3.3.2.1 1 Data flow diagram level-1*

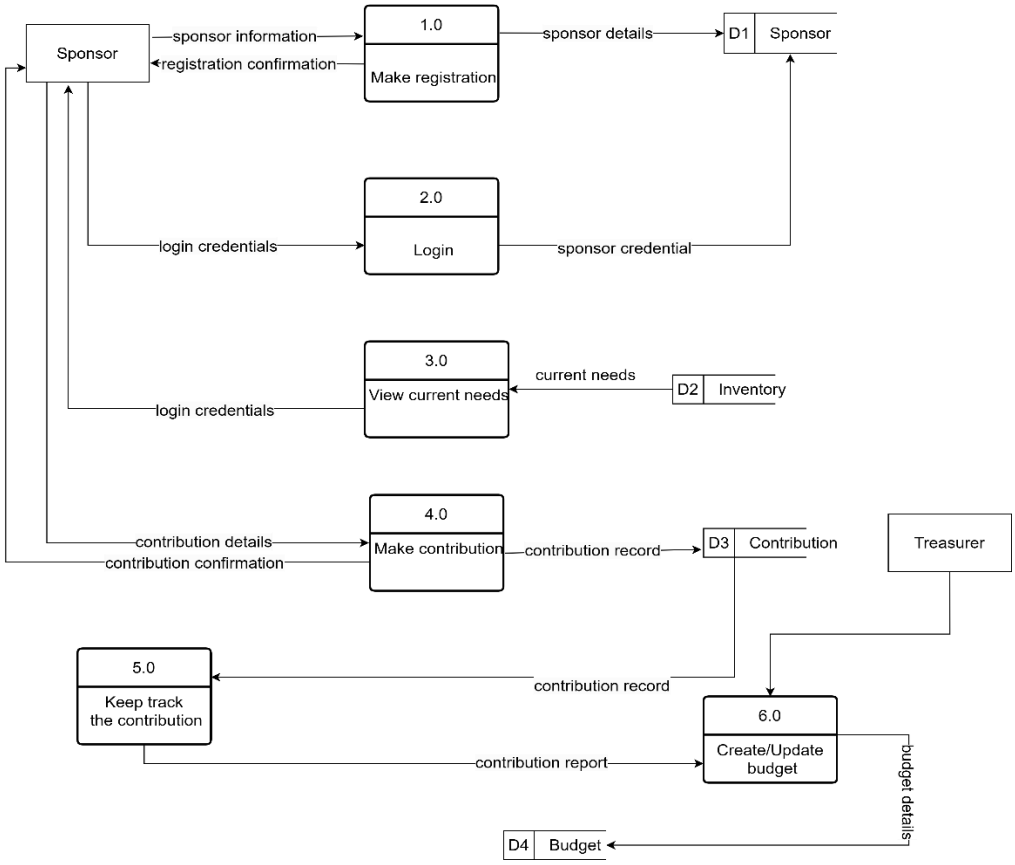



*Figure 3.3.2.1 2 Data Flow Diagram level-1*

## 2. Data Flow Diagram level -2

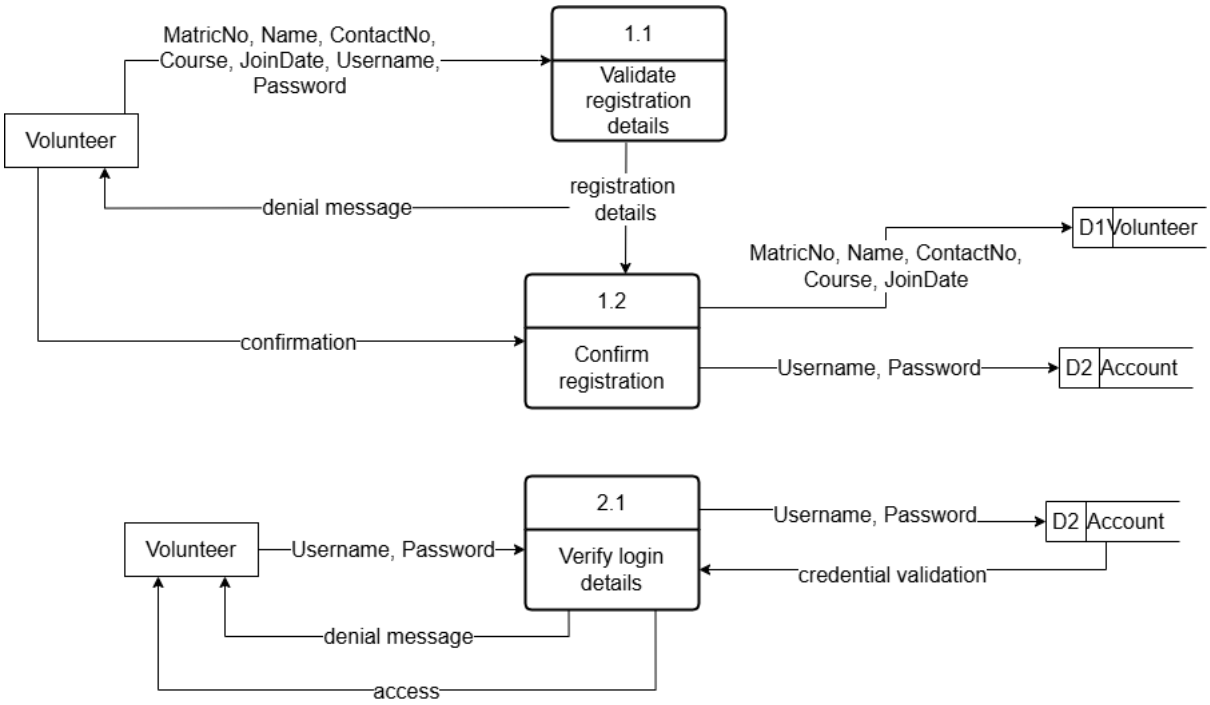



*Figure 3.3.2.2. 1 Data Flow Diagram level-2*

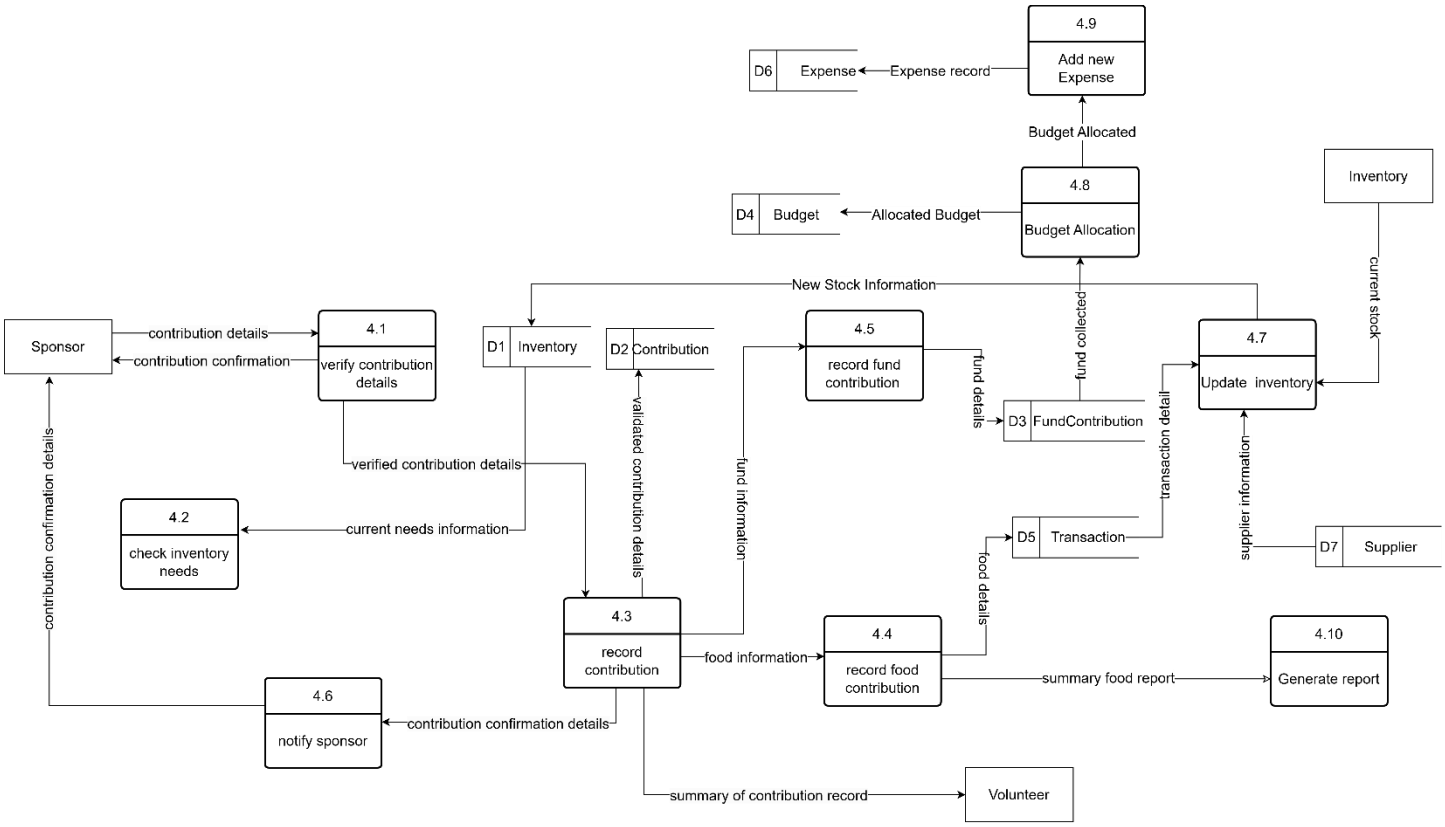



*Figure 3.3.2.2. 2 Data Flow Diagram level-2*

## 3.4 Conclusion

In conclusion, the analysis of the current manual system highlights its limitations, including inefficiencies in data handling, lack of transparency, and potential for errors due to reliance on paper records and Excel sheets. The absence of an integrated system makes it challenging to manage volunteer activities, track donations, and maintain inventory efficiently. Through the proposed computerized system, these challenges will be addressed by introducing a streamlined database that ensures accuracy, improves reporting capabilities, and enhances overall management efficiency. This analysis lays the groundwork for developing a robust solution to meet the foodbank's operational needs effectively.

# CHAPTER 4: DATABASE DESIGN

## 4.1 Introduction

Database design is important in developing the FTMK FoodShare System, ensuring that the system can efficiently store, manage, and retrieve data to support its operations. This section focuses on creating a structured and optimized database architecture, which includes conceptual, logical, and physical designs. By defining data relationships, constraints, and storage mechanisms, the design guarantees data integrity, security, and scalability. The database serves as the backbone of the system, enabling seamless integration across modules such as inventory management, volunteer coordination, sponsorship tracking, and budget allocation.

## 4.2 Conceptual Database Design

Conceptual database design is the initial step in the database design process. The ER-Diagram is created to depict the structure of the database. An entity relationship diagram depicts the relationship between entity sets in a database. It shows the logical structure of a database. It also describes the relationships between database tables. The FTMK FoodShare System includes three key scopes: volunteer, inventory, sponsorship, and budget.

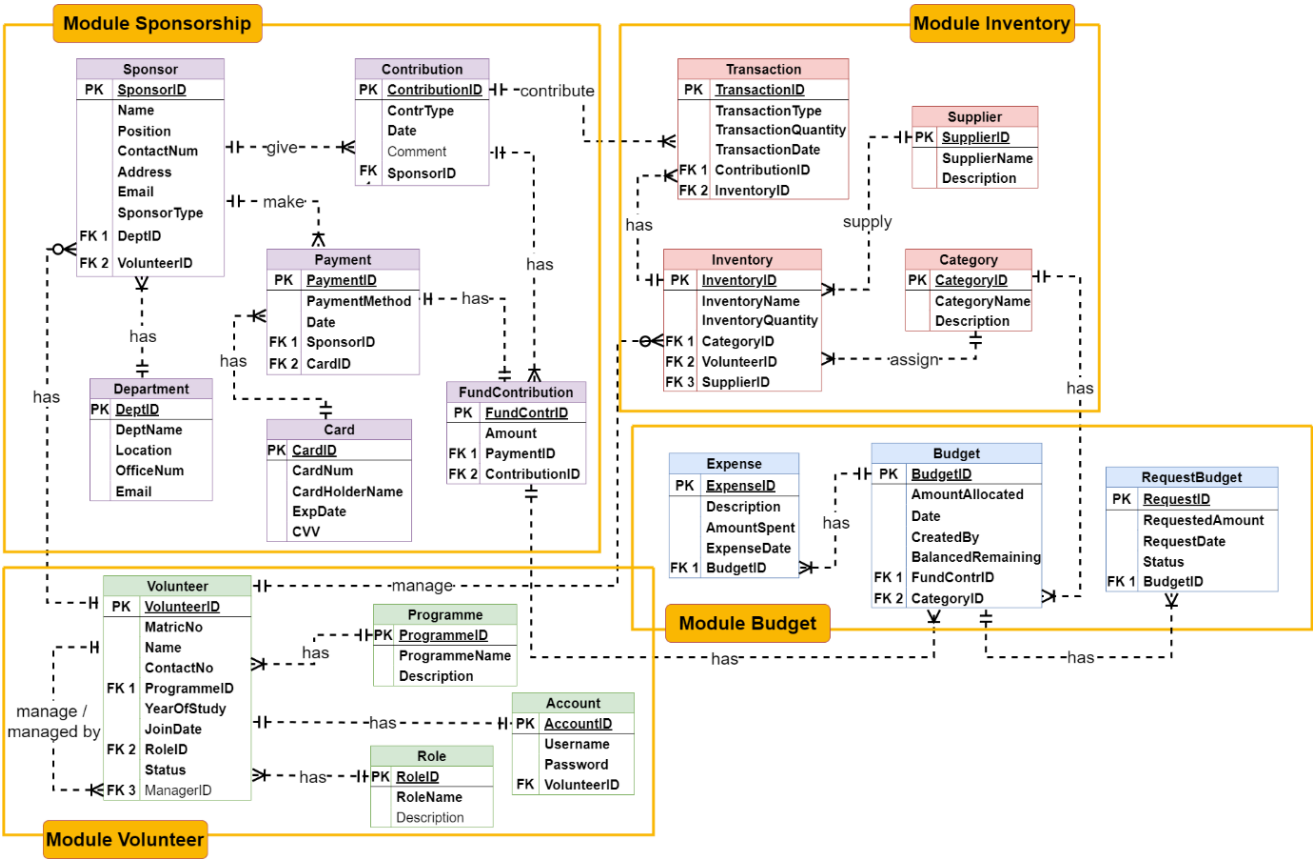



*Figure 4.2. 1 Entity Relatioship Diagram (ERD)*

## 4.3 Logical Database Design

### 4.3.1 Data Dictionary

i. Module: Volunteer

| Role | | | | | |
|---|---|---|---|---|---|
| Column | Data Type | Unique | NULL | Domain Constraint | FK Reference |
| RoleID | Int(11) | YES | NO | PK | |
| RoleName | Varchar(50) | NO | NO | | |
| Description | Varchar(200) | NO | YES | | |

*Table 4.3.1.1 Data dictionary Role*

| Programme | | | | | |
|---|---|---|---|---|---|
| Column | Data Type | Unique | NULL | Domain Constraint | FK Reference |
| ProgrammeID | Int(11) | YES | NO | PK | |
| ProgrammeName | Varchar(4) | YES | NO | | |
| Description | Varchar(50) | NO | NO | | |

*Table 4.3.1.2 Data dictionary Programme*

| Volunteer | | | | | | |
|---|---|---|---|---|---|---|
| Column | Data Type | Unique | NULL | Default Value | Domain Constraint | FK Reference |
| VolunteerID | Int(11) | YES | NO | | PK | |
| MatricNo | Varchar(10) | YES | NO | | | |
| Name | Varchar(100) | NO | NO | | | |
| ContactNo | Varchar(12) | YES | NO | | | |
| ProgrammeID | Int(11) | NO | NO | | FK | Programme |
| YearOfStudy | Int(2) | NO | NO | | | |
| JoinDate | Date | NO | NO | | | |
| RoleID | Int(11) | NO | NO | | FK | Role |
| Status | Varchar(20) | NO | NO | | | |
| ManagerID | Int(11) | NO | YES | | FK | Volunteer |

*Table 4.3.1.3 Data dictionary Volunteer*

| Account | | | | | |
|---|---|---|---|---|---|
| Column | Data Type | Unique | NULL | Domain Constraint | FK Reference |
| AccountID | Int(11) | YES | NO | PK | |
| Username | Varchar2(15) | YES | NO | | |
| Password | Varchar(255) | NO | NO | | |
| VolunteerID | Int(11) | YES | NO | FK | Volunteer |

*Table 4.3.1.4 Data dictionary Account*

ii. Module: Sponsorship

| Department | | | | | |
|---|---|---|---|---|---|
| Column | Data Type | Unique | NULL | Domain Constraint | FK Reference |
| DeptID | Int(11) | YES | NO | PK | |
| DeptName | Varchar(100) | NO | NO | | |
| Location | Varchar(200) | NO | NO | | |
| OfficeNum | Varchar(12) | YES | NO | AK | |
| Email | Varchar(100) | YES | NO | AK | |

*Table 4.3.1. 5 Data Dictionary Department*

| Sponsor | | | | | | |
|---|---|---|---|---|---|---|
| Column | Data Type | Unique | NULL | Default Value | Domain Constraint | FK Reference |
| SponsorID | Int (11) | YES | NO | | PK | |
| Name | Varchar(100) | NO | NO | | | |
| Position | Varchar(100) | NO | NO | | | |
| ContactNum | Varchar(12) | YES | NO | | AK | |
| Address | Varchar(255) | NO | NO | | | |
| Email | Varchar(100) | YES | NO | | AK | |
| SponsorType | Varchar(50) | NO | NO | | | |
| VerificationStatus | Varchar(20) | NO | NO | Pending | | |
| DeptID | Int(11) | YES | NO | | FK | Department |
| VolunteerID | Int(11) | YES | NO | | FK | Volunteer |

*Table 4.3.1. 6 Data Dictionary Sponsor*

| Contribution | | | | | |
|---|---|---|---|---|---|
| Column | Data Type | Unique | NULL | Domain Constraint | FK Reference |
| ContributionID | Int(11) | YES | NO | PK | |
| ContrCategory | Varchar(50) | NO | NO | | |
| Date | Date | NO | NO | | |
| Comment | Varchar(255) | NO | YES | | |
| SponsorID | Int(11) | YES | NO | | Sponsor |

Table 4.3.1.7 Data Dictionary Contribution

| Card | | | | | |
|---|---|---|---|---|---|
| Column | Data Type | Unique | NULL | Domain Constraint | FK Reference |
| CardID | Int(11) | YES | NO | PK | |
| CardNum | Varchar (20) | YES | NO | | |
| CardHolderName | Varchar (50) | NO | NO | | |
| ExpDate | Varchar(5) | NO | NO | | |
| CVV | Varchar(4) | YES | NO | | |

*Table 4.3.1. 8 Data Dictionary Card*

| Payment | | | | | |
|---|---|---|---|---|---|
| Column | Data Type | Unique | NULL | Domain Constraint | FK Reference |
| PaymentID | Int(11) | YES | NO | PK | |
| PaymentMethod | Numeric(9,2) | NO | NO | | |
| Date | Date | NO | NO | | |
| SponsorID | Int(11) | YES | NO | FK | Sponsor |
| CardID | Int(11) | YES | NO | FK | Card |

*Table 4.3.1. 9 Data Dictionary Payment*

| FundContribution | | | | | |
|---|---|---|---|---|---|
| Column | Data Type | Unique | NULL | Domain Constraint | FK Reference |
| FundContrID | Int(11) | YES | NO | PK | |
| Amount | Numeric(9,2) | NO | NO | | |
| PaymentID | Int(11) | YES | NO | FK | Payment |
| ContributionID | Int(11) | YES | NO | FK | Contribution |

*Table 4.3.1. 10 Data Dictionary FundContribution*

iii. Module: Budget

| Budget | | | | | |
|---|---|---|---|---|---|
| Column | Data Type | Unique | NULL | Domain Constraint | FK Reference |
| BudgetID | Int(11) | YES | NO | PK | |
| AmountAllocated | Double | NO | NO | | |
| Date | Date | NO | NO | | |
| CreatedBy | Varchar(100) | NO | NO | | |
| BalancedRemaining | Double | NO | NO | | |
| FundContrID | Int(11) | YES | YES | FK | FundContribution |
| CategoryID | Int(11)\ | YES | YES | FK | Category |

*Table 4.3.1. 11 Data Dictionary Budget*

| Expense | | | | | |
|---|---|---|---|---|---|
| Column | Data Type | Unique | NULL | Domain Constraint | FK Reference |
| ExpenseID | Int(11) | YES | NO | PK | |
| Description | Varchar(100) | NO | NO | | |
| AmountSpent | Double | NO | NO | | |
| ExpenseDate | Date | NO | NO | | |
| BudgetID | Int(11) | YES | NO | FK | Budget |

*Table 4.3.1. 12 Data Dictionary Expense*

| RequestBudget | | | | | |
|---|---|---|---|---|---|
| Column | Data Type | Unique | NULL | Domain Constraint | FK Reference |
| RequestID | Int(11) | YES | NO | PK | |
| RequestedAmount | Double | NO | NO | | |
| RequestedDate | Date | NO | NO | | |
| Status | Varchar(20) | NO | NO | | |
| BudgetID | Int(11) | YES | NO | FK | Budget |

*Table 4.3.1. 13 Data Dictionary RequestBudget*

iv. Module: Inventory

| Inventory | | | | | |
|---|---|---|---|---|---|
| Column | Data Type | Unique | NULL | Domain Constraint | FK Reference |
| inventoryID | Int (11) | YES | NO | PK | |
| inventoryName | Varchar(50) | NO | NO | | |
| inventoryQuantity | Int(11) | NO | NO | | |
| categoryID | Int (11) | NO | NO | FK | Category |
| VolunteerID | Int(11) | YES | NO | FK | Volunteer |
| supplierID | Int(11) | NO | NO | FK | Supplier |

*Table 4.3.1. 14 Data Dictionary Inventory*

| Category | | | | | |
|---|---|---|---|---|---|
| Column | Data Type | Unique | NULL | Domain Constraint | FK Reference |
| categoryID | Int(11) | YES | NO | PK | |
| categoryName | Varchar(50) | NO | NO | | |
| description | Varchar(200) | NO | NO | | |

Table 4.3.1. 15 Data Dictionary category

| Transaction | | | | | |
|---|---|---|---|---|---|
| Column | Data Type | Unique | NULL | Domain Constraint | FK Reference |
| transactionID | Int(11) | YES | NO | PK | |
| transactionType | Varchar(20) | NO | NO | | |
| transactionQuantity | Int(11) | NO | NO | | |
| transactionDate | Date | NO | NO | | |
| ContributionID | Int(11) | YES | NO | FK | Contribution |
| inventoryID | Int(11) | YES | NO | FK | Inventory |

*Table 4.3.1. 16 Data Dictionary Transaction*

| Supplier | | | | | |
|---|---|---|---|---|---|
| Column | Data Type | Unique | NULL | Domain Constraint | FK Reference |
| supplierID | Int(11) | YES | NO | PK | |
| supplierName | Varchar(100) | NO | NO | | |
| description | Varchar(200) | NO | NO | | |

*Table 4.3.1. 17 Data Dictionary Supplier*

## 4.4 Physical Database Design

In this phase, Data Definition Language (DDL) is utilised to translate the conceptual database architecture into physical database structures. It entails establishing tables, indexes, views, and other database structures in order to store and organise data efficiently. DDL specifies each table's structure, including data types, sizes, and constraints (such as primary keys, foreign keys, and unique constraints) to assure data integrity. DDL is also used to establish storage options, partitions, and indexing algorithms that improve query performance and data retrieval. This stage is crucial for building a functional and efficient database system.

### 4.4.1 Data Definition Language

#### 1. Volunteer (MariaDB)

**Table structure for table 'role'**

```
CREATE TABLE ROLE (
   RoleID INT(11) AUTO_INCREMENT PRIMARY KEY,
   RoleName VARCHAR(50) NOT NULL,
   Description VARCHAR(200)
);
```

**Table structure for table 'programme'**

```
CREATE TABLE PROGRAMME (
   ProgrammeID INT(11) AUTO_INCREMENT PRIMARY KEY,
   ProgrammeName VARCHAR(4) NOT NULL UNIQUE,
   Description VARCHAR(100) NOT NULL
);
```

**Table structure for table 'volunteer'**

```
CREATE TABLE VOLUNTEER (
   VolunteerID INT(11) AUTO_INCREMENT PRIMARY KEY,
   MatricNo VARCHAR(10) NOT NULL UNIQUE,
   Name VARCHAR(100) NOT NULL,
```

```
    ContactNo VARCHAR(12) NOT NULL,
    ProgrammeID INT(11) NOT NULL,
    YearOfStudy INT(2) NOT NULL,
    JoinDate DATE NOT NULL,
    RoleID INT(11) NOT NULL,
    Status VARCHAR(20) NOT NULL,
    ManagerID INT(11),
    FOREIGN KEY (ProgrammeID) REFERENCES PROGRAMME(ProgrammeID),
    FOREIGN KEY (RoleID) REFERENCES ROLE(RoleID),
    FOREIGN KEY (ManagerID) REFERENCES VOLUNTEER(VolunteerID)
);
```

**Table structure for table 'account'**

```
CREATE TABLE ACCOUNT (
    AccountID INT(11) AUTO_INCREMENT PRIMARY KEY,
    Username VARCHAR(15) NOT NULL UNIQUE,
    Password VARCHAR(255) NOT NULL,
    VolunteerID INT(11) NOT NULL UNIQUE,
    FOREIGN KEY (VolunteerID) REFERENCES VOLUNTEER(VolunteerID)
);
```

### 2. Sponsorship (Postgresql)

**Table structure for table 'Department'**

```
CREATE TABLE Department (
    DeptID INT NOT NULL PRIMARY KEY,
    DeptName VARCHAR(100) NOT NULL,
    Location VARCHAR(200) NOT NULL,
    OfficeNum VARCHAR(12) NOT NULL,
    Email VARCHAR(100) NOT NULL,
    CONSTRAINT AK_OfficeNum UNIQUE (OfficeNum),
    CONSTRAINT AK_Emails UNIQUE (Email)
);
```

**Table structure for table ‘Sponsor’**

```
CREATE TABLE Sponsor (
   SponsorID SERIAL PRIMARY KEY,
   Name VARCHAR(100) NOT NULL,
   Position VARCHAR(100) NOT NULL,
   Gender VARCHAR(20) NOT NULL,
   DOB DATE NOT NULL,
   ContactNum VARCHAR(12) NOT NULL,
   Address VARCHAR(255) NOT NULL,
   Email VARCHAR(100) NOT NULL,
   SponsorType VARCHAR(50) NOT NULL,
   DeptID INT NOT NULL,
   CONSTRAINT AK_ContactNum UNIQUE (ContactNum),
   CONSTRAINT AK_Email UNIQUE (Email),
   CONSTRAINT FK_Department FOREIGN KEY (DeptID) REFERENCES Department(DeptID)
);
```

**Table structure for table ‘Contribution’**

```
CREATE TABLE Contribution (
   ContributionID SERIAL PRIMARY KEY,
   ContrCategory VARCHAR(50) NOT NULL,
   Date DATE NOT NULL,
   Comment VARCHAR(255) NULL,
   SponsorID INT NOT NULL,
   CONSTRAINT FK_Sponsor FOREIGN KEY (SponsorID) REFERENCES Sponsor(SponsorID)
);
```

**Table structure for table 'Card'**

```
CREATE TABLE Card(
 CardID SERIAL PRIMARY KEY,
 CardNum VARCHAR(20) NOT NULL,
 CardHolderName VARCHAR(50) NOT NULL,
 ExpDate VARCHAR(5) NOT NULL,
 CVV VARCHAR(4) NOT NULL);
```

**Table structure for table 'Payment'**

```
CREATE TABLE Payment(
 PaymentID SERIAL PRIMARY KEY,
 PaymentMethod VARCHAR(20) NOT NULL,
 Date Date NOT NULL,
 SponsorID INT NOT NULL,
 CardID INT NOT NULL,
 CONSTRAINT FK_Sponsor FOREIGN KEY (SponsorID) REFERENCES Sponsor(SponsorID),
 CONSTRAINT FK_Card FOREIGN KEY (CardID) REFERENCES Card(CardID));
```

**Table structure for table 'FundContribution'**

```
CREATE TABLE FundContribution (
   FundContrID SERIAL PRIMARY KEY,
   Amount NUMERIC(9,2) NOT NULL,
   PaymentID INT NOT NULL,
   ContributionID INT NOT NULL,
   CONSTRAINT FK_Payment FOREIGN KEY (PaymentID) REFERENCES Payment(PaymentID),
   CONSTRAINT FK_Contribution FOREIGN KEY (ContributionID)
```

REFERENCES Contribution(ContributionID),

CONSTRAINT chk_amount CHECK (Amount IN (50, 100, 150) OR Amount > 150));

### 3. Budget (MySQL)

**Table structure for table 'Budget'**

```
CREATE TABLE Budget (
   BudgetID INT AUTO_INCREMENT PRIMARY KEY,
   AmountAllocated DECIMAL(10, 2) NOT NULL,
   Date DATE NOT NULL,
   CreatedBy VARCHAR(100) NOT NULL,
   BalancedRemaining DECIMAL(10, 2) NOT NULL,
   FundContrID INT,
   CategoryID INT,
   FOREIGN KEY (FundContrID) REFERENCES FundContribution(FundContrID),
   FOREIGN KEY (CategoryID) REFERENCES Category(CategoryID)
);
```

**Table structure for table 'Expense'**

```
CREATE TABLE Expense (
   ExpenseID INT AUTO_INCREMENT PRIMARY KEY,
   Description VARCHAR(255) NOT NULL,
   AmountSpent DECIMAL(10, 2) NOT NULL,
   ExpenseDate DATE NOT NULL,
   BudgetID INT NOT NULL,
   FOREIGN KEY (BudgetID) REFERENCES Budget(BudgetID)
);
```

**Table structure for table ‘Request Budget’**

```
CREATE TABLE RequestBudget (
   RequestID INT AUTO_INCREMENT PRIMARY KEY,
   RequestedAmount DOUBLE NOT NULL,
   RequestDate DATE NOT NULL,
   Status VARCHAR(20 NOT NULL DEFAULT ‘pending’,
   BudgetID INT NULL,
   CONSTRAINT FK_Budget FOREIGN KEY (BudgetID) REFERENCE Budget(BudgetID)
);
```

### 4. Module Inventory (MySQL)

**Table structure for table ‘Inventory’**

```
CREATE TABLE inventory (
  inventoryID int(4) NOT NULL AUTO_INCREMENT,
  inventoryName varchar(50) NOT NULL,
  inventoryQuantity int(4) NOT NULL,
  categoryID int(4) NOT NULL,
  VolunteerID int(4) NOT NULL,
  supplierID int NOT NULL,
  PRIMARY KEY (inventoryID)
```

**Table structure for table ‘Transaction’**

```
CREATE TABLE `transaction` (
  transactionID int(11) NOT NULL AUTO_INCREMENT,
  transactionType varchar(50) NOT NULL,
  transactionQuantity int(11) NOT NULL,
  transactionDate date NOT NULL,
  transactionStatus varchar(50) NOT NULL DEFAULT 'Pending',
  inventoryID int(11),
  ContributionID int(11),
  PRIMARY KEY (transactionID)
```

**Table structure for table ‘Category’**

```
CREATE TABLE category (
  categoryID int(4) NOT NULL AUTO_INCREMENT,
  categoryName varchar(50) NOT NULL,
  description varchar(200) NOT NULL,
  PRIMARY KEY (categoryID)
```

**Table structure for table ‘Supplier’**

```
CREATE TABLE supplier (
  supplierID int(11) NOT NULL AUTO_INCREMENT,
  supplierName varchar(50) NOT NULL,
  description varchar(200) NOT NULL,
PRIMARY KEY (`supplierID`)
```

## 4.5 Conclusion

The database design phase has laid the foundation for the successful implementation of the FTMK FoodShare System. Through the conceptual, logical, and physical database designs, the system ensures efficient data management, integrity, and scalability. The conceptual design, represented by the ER diagram, effectively captures the relationships among entities, while the logical design normalizes these relationships into a structured schema. The physical design translates these schemas into actionable database structures, ensuring compatibility with the selected DBMS platforms.

# CHAPTER 5: INDIVIDUAL APPLICATION MODULES AND IMPLEMENTATION

## 5.1 Introduction

After gathering requirements and building components to demonstrate system flow, project implementation involves transforming documented data into a real program. This chapter focusses on the programming techniques used to construct the system and error handling. A programming technique describes the development process and the specific language code utilised.

## 5.2 Database Setup and Installation

### 5.2.1 MySQL/MariaDB on Windows

1. Download Windows version of XAMPP that includes MariaDB/MySQL on XAMPP official website. During setup, ensure that MariaDB/MySQL is selected along with PHPMyAdmin.

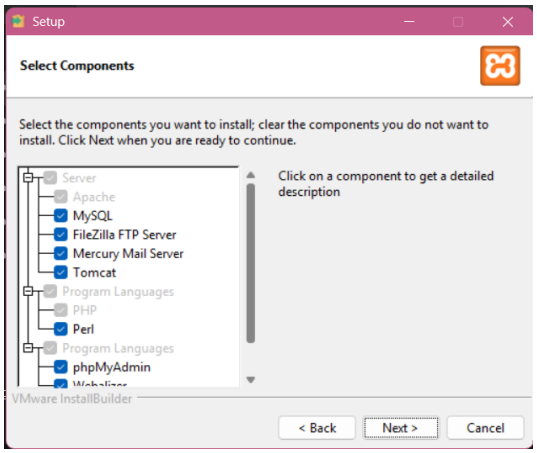



*Figure 5.2.1.1 XAMPP Setup Configuration*

2. Launch the XAMPP Control Panel from the installation. Start the Apache and MySQL modules by clicking the **Start** buttons next to them. Ensure both modules show the status as "Running."

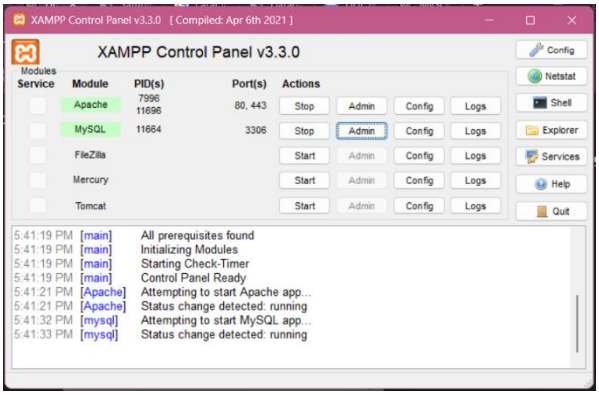



*Figure 5.2.1.2 Starting XAMPP's Apache and MySQL*

3. Open a web browser and navigate to http://localhost/phpmyadmin or select "Admin" from MySQL module in XAMPP Control Panel to access PHPMyAdmin.
4. In PHPMyAdmin, click **Databases** in the top menu. Enter a name for the new database (e.g., fsvolunteer or module_budget) and click **Create**.

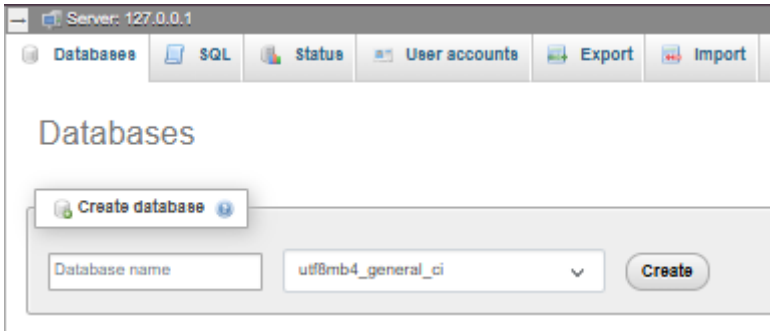



*Figure 5.2.1.3 Creating database in PHPMyAdmin*

5. Create a simple PHP script to test the database connection. Then, save the file (e.g., DBConnection.php) in the htdocs folder of your XAMPP installation directory (e.g., C:\xampp\htdocs).

```php
<?php
$servername = "localhost";
$username = "shira";
$password = "";
$dbname = "FSvolunteer";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
} /*else {
    echo "Database connection successful!";
} */
```

*Figure 5.2.1.4 Testing database connection*

```php
<?php
// Database configuration
$host = 'localhost';       // Hostname (e.g., localhost or 127.0.0.1)
$username = 'root';        // Database username
$password = '';            // Database password (leave empty if no password)
$database = 'module_budget'; // Database name

// Create connection
$conn = new mysqli($host, $username, $password, $database);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
```

*Figure 5.2.1.5 Testing database connection*

### 5.2.2 Postgresql on Windows

1. Download from postgreSQL website through this link https://www.postgresql.org/download/. Then click the windows option.

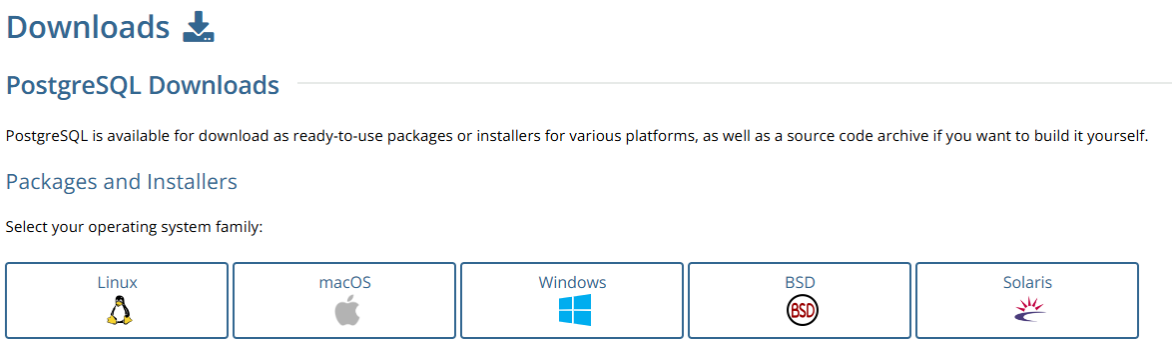



*Figure 5.2.2. 1 PostgreSQl website*

2. Click Download the installer to to get the windows installer

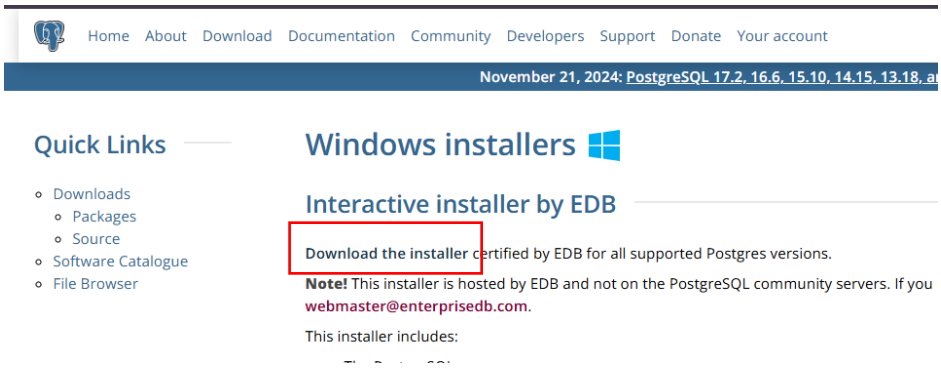



*Figure 5.2.2. 2 PostgreSQL Installer*

3. Choose the latest version and click on download at windows x86-64 sections


*Figure 5.2.2. 3 Download PostgreSQL 17.2*

4. After done download in browser, ready to run the file that already downloads. The feature will appear like this. Then, click Next.

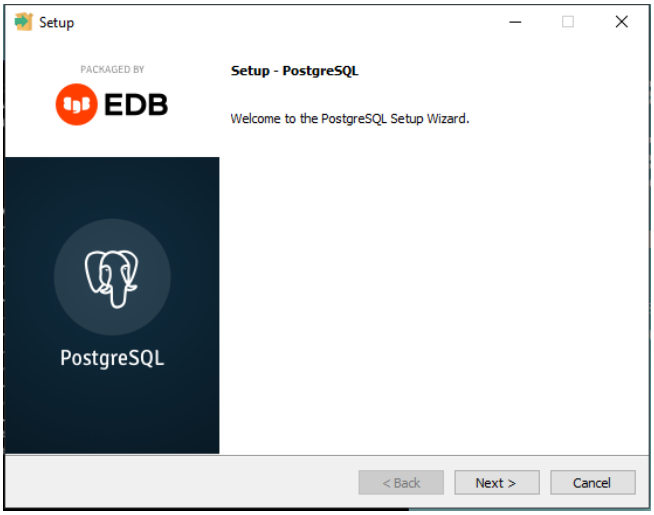



*Figure 5.2.2. 4 PostgreSQL Setup*

5. Choose Directory to be installed, usually installation at C:\Program Files\

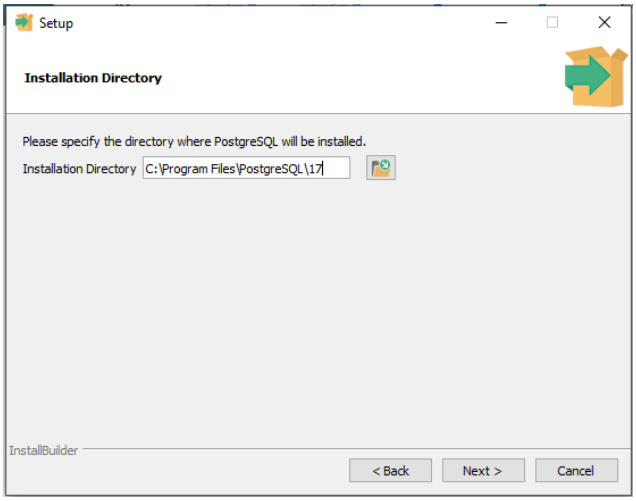



*Figure 5.2.2. 5 Installation Directory*

6. Select the component you want to install and clear the componet you do not want to install. Click next when you ready to continue.

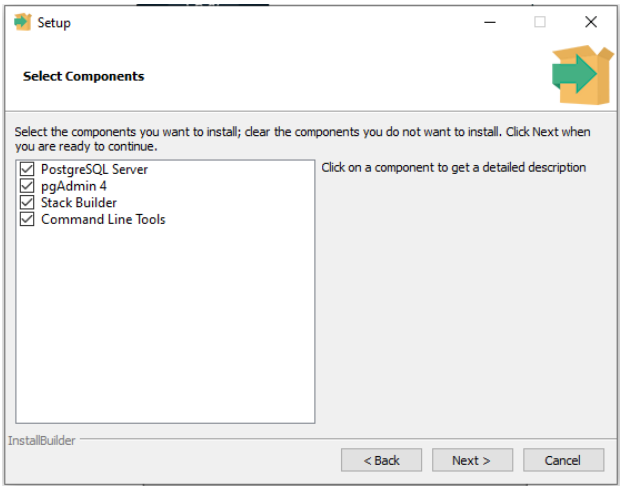



*Figure 5.2.2. 6 Select Component*

7. Select a directory under which to store you data.Then Click next.

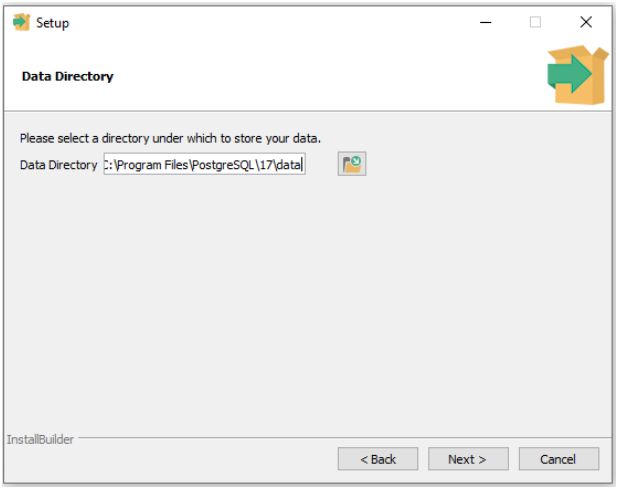



*Figure 5.2.2. 7 Data Directory*

8. Pre Installations Summary. Then, click next.

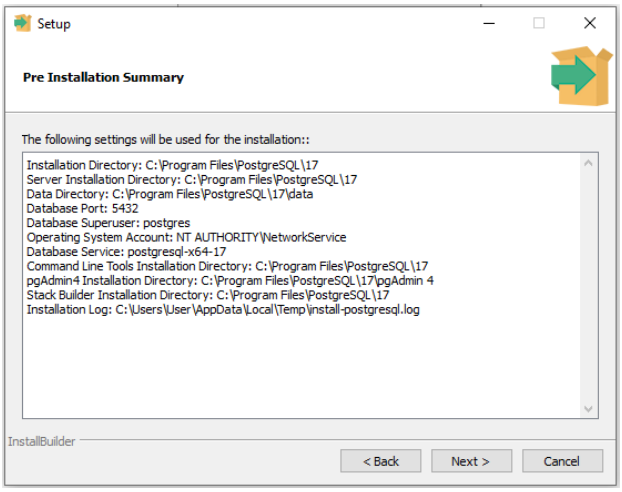



*Figure 5.2.2. 8 Pre Installation Summary*

9. Now Postgresql is ready to install on computer. Click Next to continue.

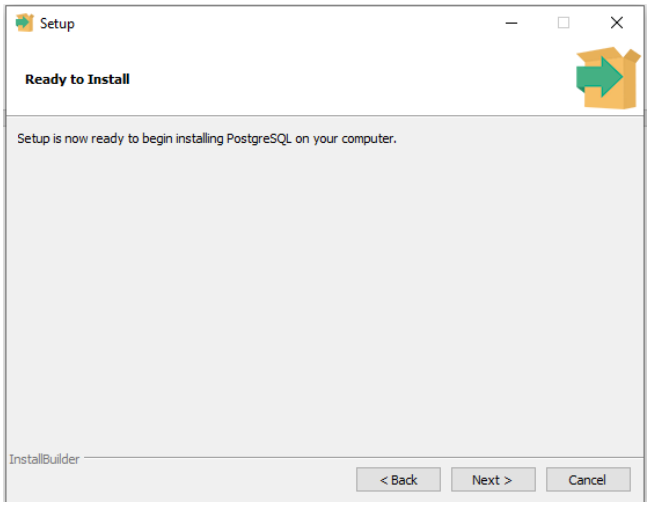



*Figure 5.2.2. 9 Ready to Install*

*2.2. 9 PostgreSQL ready to install*

10. Wait for installation, this may take while.

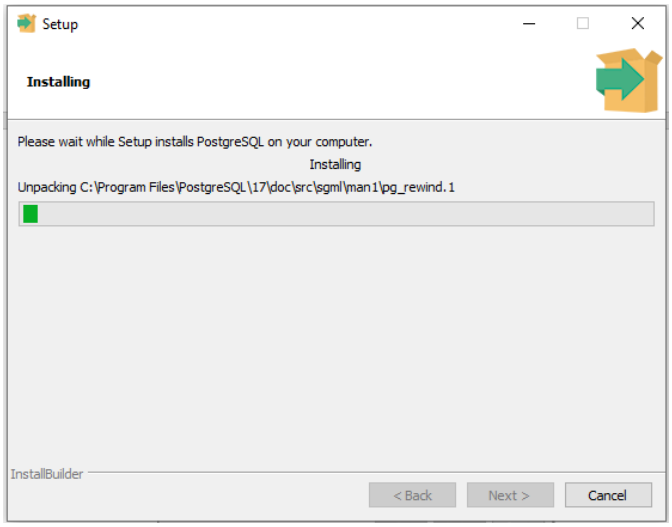



*Figure 5.2.2. 10 Installing PostgreSQL*

11. This is optional, just untick the checkbox, then Finish

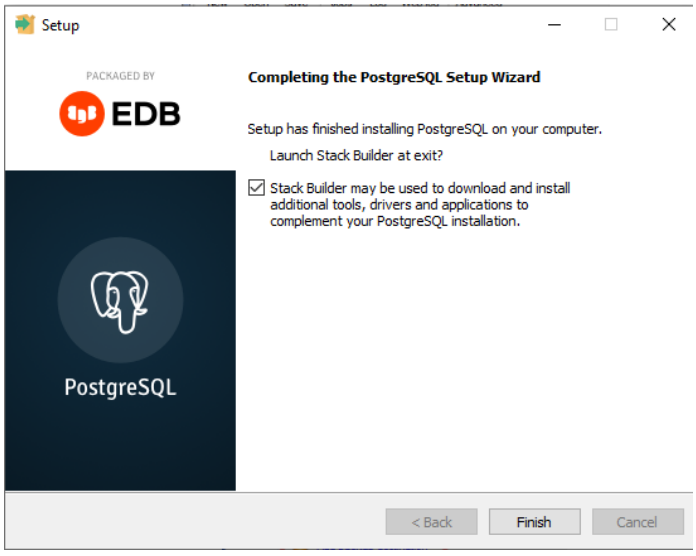



*Figure 5.2.2. 11 Optional Step for Stack Builder*

12. Launch Pg Admin 4 from the desktop.

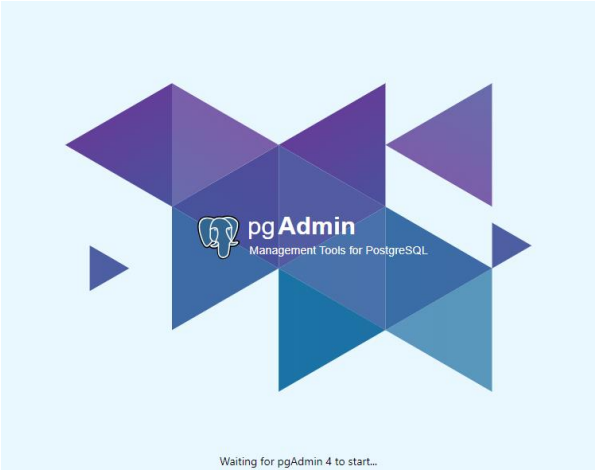



*Figure 5.2.2. 12 pgAdmin 4 launch*

13. Database creation, right click on Posgresql → create → Database

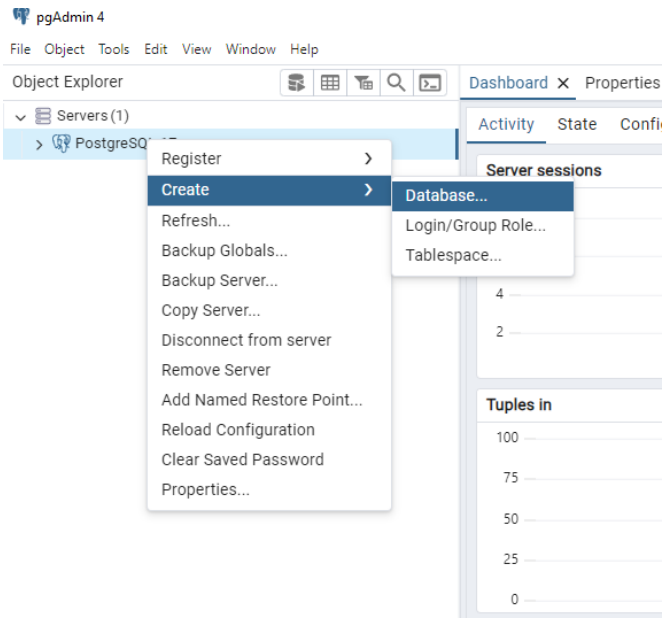



*Figure 5.2.2. 13 Database Creation*

14. Then, create database name for example: FoodShare

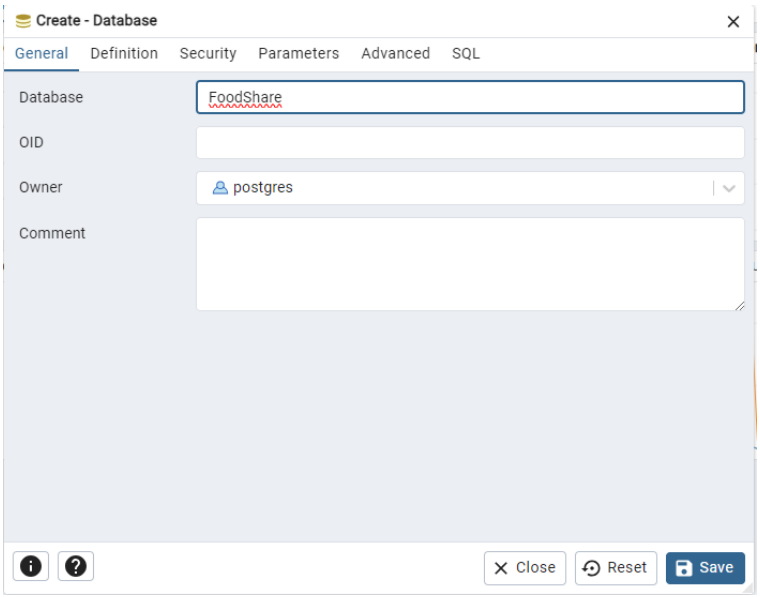



*Figure 5.2.2. 14 Create Database Name*

15. State the host name/address as postgres.For username and password it can be change.Then click save.

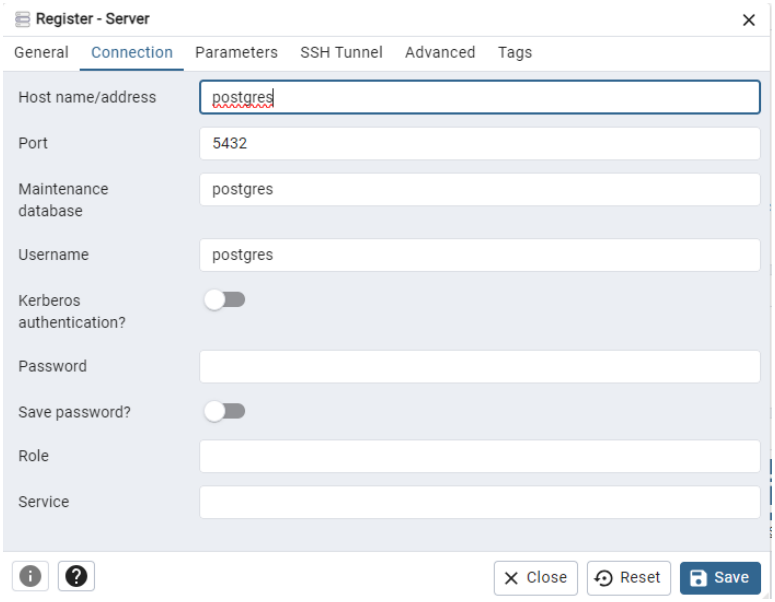



*Figure 5.2.2. 15 Database Connection*

### 5.2.3 MySQL on Linux

After the Linux environment is up and running, in this Ubuntu, use the APT package manager to install MySQL using the command "sudo apt install mysql-server".

```
$ sudo apt install mysql-server
```

Then, start the MySQL to make it running with the command below:

```
$ sudo systemctl start mysql.service
```

Next, we have to configure the MySQL server for security and etc. Run the script below:

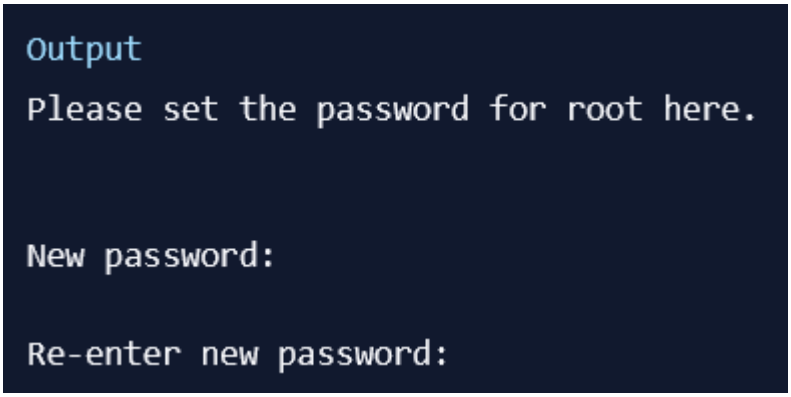
```
$ sudo mysql_secure_installation
```

The script will prompt for password security such as below:

```
Output
Securing the MySQL server deployment.

Connecting to MySQL using a blank password.

VALIDATE PASSWORD COMPONENT can be used to test passwords
and improve security. It checks the strength of password
and allows the users to set only those passwords which are
secure enough. Would you like to setup VALIDATE PASSWORD component?

Press y|Y for Yes, any other key for No: Y

There are three levels of password validation policy:

LOW    Length >= 8
MEDIUM Length >= 8, numeric, mixed case, and special characters
STRONG Length >= 8, numeric, mixed case, special characters and dictionary

Please enter 0 = LOW, 1 = MEDIUM and 2 = STRONG:
 2
```

Choose which option is preferred and remember to follow the policy.

Next, it will ask for root password.




Once completed, the MySQL is properly secured. We can move on to creating users for databases.

## 5.3 Database Implementation

### i. Stored Procedures

**Stored Procedures for Volunteer in MariaDB**

```
CREATE PROCEDURE GetVolunteersCountByProgrammeByYear(IN selectedYear INT)
BEGIN
   SELECT p.ProgrammeName AS Programme, p.Description AS Description,
      COUNT(v.MatricNo) AS Volunteer_Count
   FROM programme p
   LEFT JOIN volunteer v ON v.ProgrammeID = p.ProgrammeID AND YEAR(v.JoinDate) = selectedYear
   GROUP BY p.ProgrammeName, p.Description;
END

CREATE PROCEDURE GetVolunteersCountByYearOfStudyByYear(IN selectedYear INT)
BEGIN
```

```
        SELECT y.YearOfStudy, COUNT(v.VolunteerID) AS Volunteer_Count
        FROM (SELECT 1 AS YearOfStudy UNION SELECT 2 UNION SELECT 3) AS
y
        LEFT JOIN volunteer v ON v.YearOfStudy = y.YearOfStudy AND
YEAR(v.JoinDate) = selectedYear
        GROUP BY y.YearOfStudy;
    END
```

**Stored Procedures  Sponsorship in Postgresql**

```
    CREATE OR REPLACE PROCEDURE get_food_and_money_data()
    AS $$
    BEGIN
      CREATE TEMP TABLE temp_food_and_money_data (
        Month TEXT,
        Total NUMERIC,
        Type TEXT
      );
      INSERT INTO temp_food_and_money_data (Month, Total, Type)
      SELECT
        TO_CHAR("transactionDate", 'YYYY-MM') AS Month,
        SUM("transactionQuantity") AS Total,
        'Food' AS Type
      FROM
        transaction
      GROUP BY
        TO_CHAR("transactionDate", 'YYYY-MM');
      INSERT INTO temp_food_and_money_data (Month, Total, Type)
      SELECT
        TO_CHAR(payment.date, 'YYYY-MM') AS Month,
        SUM(fundcontribution.amount) AS Total,
        'Money' AS Type
      FROM
        fundcontribution
```

```
        JOIN
            payment ON fundcontribution.paymentid = payment.paymentid
        GROUP BY
            TO_CHAR(payment.date, 'YYYY-MM');
    END;
    $$ LANGUAGE plpgsql;
    CALL get_food_and_money_data();
    SELECT * FROM temp_food_and_money_data;
```

```
    CREATE OR REPLACE PROCEDURE
get_user_money_contribution_log(username_param VARCHAR)
    LANGUAGE plpgsql
    AS $$
    BEGIN
        CREATE TEMP TABLE temp_contribution_log (
            payment_date VARCHAR,
            fund_amount NUMERIC,
            contr_comment TEXT
        );

        INSERT INTO temp_contribution_log
        SELECT
            TO_CHAR(p.date, 'DD-MM-YYYY') AS payment_date,
            f.amount AS fund_amount,
            c.comment AS contr_comment
        FROM contribution c
        INNER JOIN fundContribution f ON c.contributionID = f.contributionID
        INNER JOIN payment p ON f.paymentID = p.paymentID
        INNER JOIN sponsor s ON c.sponsorid = s.sponsorid
        WHERE s.username = username_param
        ORDER BY p.date DESC;

    END $$;
    CALL get_user_money_contribution_log('$username');
```

```
SELECT * FROM temp_contribution_log;
```

*Table 5.3. 1 Stored Procedures*

ii. Trigger

**Trigger Volunteer in MariaDB**

```
CREATE TRIGGER BeforeMatricNoInsert
BEFORE INSERT ON volunteer
FOR EACH ROW
BEGIN
   DECLARE managerName VARCHAR(255);
   DECLARE contactNo VARCHAR(255);
   DECLARE errorMessage VARCHAR(512);  -- Declare a variable for the error message

   -- Check if the MatricNo already exists and Status
   IF EXISTS (SELECT 1
         FROM volunteer
         WHERE MatricNo = NEW.MatricNo AND Status IN ('Requesting Resign','Resiqn Approved','Resign Denied')) THEN
      -- Retrieve manager's name and contact number using the ManagerID
      SELECT v2.Name, v2.ContactNo
      INTO managerName, contactNo
      FROM volunteer v1
      JOIN volunteer v2 ON v1.ManagerID = v2.VolunteerID
      WHERE v1.MatricNo = NEW.MatricNo;

      -- Construct the error message
      SET errorMessage = CONCAT('You already resigned. Do consult ', managerName, ' via ', contactNo, ' for any further assistance.');

      -- Raise an error with the constructed message
      SIGNAL SQLSTATE '45000'
```

```
            SET MESSAGE_TEXT = errorMessage;
        END IF;

        -- Check if the MatricNo already exists (without checking Status)
        IF EXISTS (SELECT 1 FROM volunteer WHERE MatricNo = NEW.MatricNo)
THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Matric Number already exists.';
        END IF;
    END;
```

**Trigger Sponsorship in Postgresql**

```
    CREATE OR REPLACE FUNCTION check_age_for_sponsor()
    RETURNS TRIGGER AS $$
    BEGIN
        IF (DATE_PART('year', AGE(NEW.dob)) < 20) THEN
            -- Raise an exception if the user's age is less than 20
            RAISE EXCEPTION 'You must be at least 20 years old to register as a
sponsor.';
        END IF;
        RETURN NEW;
    END;
    $$ LANGUAGE plpgsql;

    CREATE TRIGGER age_check_before_sponsor_registration
    BEFORE INSERT OR UPDATE ON sponsor
    FOR EACH ROW
    EXECUTE FUNCTION check_age_for_sponsor();
```

```
    --check amount in payment--
    CREATE OR REPLACE FUNCTION check_minimum_amount()
    RETURNS TRIGGER AS $$
    BEGIN
        -- Check if the Amount is less than RM 50
```

```
        IF NEW.Amount < 50 THEN
            RAISE EXCEPTION 'Amount must be RM 50 or more, but received RM %',
NEW.Amount;
        END IF;

        RETURN NEW;
    END;
    $$ LANGUAGE plpgsql;

    CREATE TRIGGER check_minimum_amount_trigger
    BEFORE INSERT ON FundContribution
    FOR EACH ROW
    EXECUTE FUNCTION check_minimum_amount();
```

*Table 5.3. 2 Trigger*

**Audit Triggers Budget in MySQL**

```
    CREATE TABLE audit (
        audit_id INT AUTO_INCREMENT PRIMARY KEY,
        table_name VARCHAR(255) NOT NULL,
        operation_type VARCHAR(50) NOT NULL,
        operation_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        old_data JSON,
        new_data JSON
    );

    -- Create trigger for the budget table
    DELIMITER $$
    CREATE TRIGGER budget_insert_audit_trigger
    AFTER INSERT ON budget
    FOR EACH ROW
    BEGIN
```

```
    INSERT INTO audit (table_name, operation_type, new_data)
    VALUES ('budget', 'INSERT', JSON_OBJECT(
        'BudgetID', NEW.BudgetID,
        'AmountAllocated', NEW.AmountAllocated,
        'Date', NEW.Date,
        'CreatedBy', NEW.CreatedBy,
        'BalancedRemaining', NEW.BalancedRemaining,
        'FundContriID', NEW.FundContriID,
        'CategoryID', NEW.CategoryID
    ));
END$$
DELIMITER ;

DELIMITER $$
CREATE TRIGGER budget_update_audit_trigger
AFTER UPDATE ON budget
FOR EACH ROW
BEGIN
    INSERT INTO audit (table_name, operation_type, old_data, new_data)
    VALUES ('budget', 'UPDATE',
        JSON_OBJECT(
            'BudgetID', OLD.BudgetID,
            'AmountAllocated', OLD.AmountAllocated,
            'Date', OLD.Date,
            'CreatedBy', OLD.CreatedBy,
            'BalancedRemaining', OLD.BalancedRemaining,
            'FundContriID', OLD.FundContriID,
            'CategoryID', OLD.CategoryID
        ),
        JSON_OBJECT(
            'BudgetID', NEW.BudgetID,
            'AmountAllocated', NEW.AmountAllocated,
            'Date', NEW.Date,
```

```
            'CreatedBy', NEW.CreatedBy,
            'BalancedRemaining', NEW.BalancedRemaining,
            'FundContriID', NEW.FundContriID,
            'CategoryID', NEW.CategoryID
        )
    );
END$$
DELIMITER ;

DELIMITER $$
CREATE TRIGGER budget_delete_audit_trigger
AFTER DELETE ON budget
FOR EACH ROW
BEGIN
    INSERT INTO audit (table_name, operation_type, old_data)
    VALUES ('budget', 'DELETE', JSON_OBJECT(
        'BudgetID', OLD.BudgetID,
        'AmountAllocated', OLD.AmountAllocated,
        'Date', OLD.Date,
        'CreatedBy', OLD.CreatedBy,
        'BalancedRemaining', OLD.BalancedRemaining,
        'FundContriID', OLD.FundContriID,
        'CategoryID', OLD.CategoryID
    ));
END$$
DELIMITER ;
```

*Table 5.3 3 Audit Trigger Budget*

**Audit Triggers Expense in MySQL**

```
-- Create trigger for the expense table
DELIMITER $$
CREATE TRIGGER expense_insert_audit_trigger
AFTER INSERT ON expense
FOR EACH ROW
BEGIN
   INSERT INTO audit (table_name, operation_type, new_data)
   VALUES ('expense', 'INSERT', JSON_OBJECT(
      'ExpenseID', NEW.ExpenseID,
      'Description', NEW.Description,
      'AmountSpent', NEW.AmountSpent,
      'ExpenseDate', NEW.ExpenseDate,
      'BudgetID', NEW.BudgetID
   ));
END$$
DELIMITER ;

DELIMITER $$
CREATE TRIGGER expense_update_audit_trigger
AFTER UPDATE ON expense
FOR EACH ROW
BEGIN
   INSERT INTO audit (table_name, operation_type, old_data, new_data)
   VALUES ('expense', 'UPDATE',
      JSON_OBJECT(
         'ExpenseID', OLD.ExpenseID,
         'Description', OLD.Description,
         'AmountSpent', OLD.AmountSpent,
         'ExpenseDate', OLD.ExpenseDate,
         'BudgetID', OLD.BudgetID
      ),
```

```
        JSON_OBJECT(
            'ExpenseID', NEW.ExpenseID,
            'Description', NEW.Description,
            'AmountSpent', NEW.AmountSpent,
            'ExpenseDate', NEW.ExpenseDate,
            'BudgetID', NEW.BudgetID
        )
    );
END$$
DELIMITER ;

DELIMITER $$
CREATE TRIGGER expense_delete_audit_trigger
AFTER DELETE ON expense
FOR EACH ROW
BEGIN
    INSERT INTO audit (table_name, operation_type, old_data)
    VALUES ('expense', 'DELETE', JSON_OBJECT(
        'ExpenseID', OLD.ExpenseID,
        'Description', OLD.Description,
        'AmountSpent', OLD.AmountSpent,
        'ExpenseDate', OLD.ExpenseDate,
        'BudgetID', OLD.BudgetID
    ));
END$$
DELIMITER ;
```

*Table 5.3 4 Audit Trigger Expense*

**Audit Triggers RequestBudget in MySQL**

```
-- Create trigger for the requestbudget table
DELIMITER $$
CREATE TRIGGER requestbudget_insert_audit_trigger
AFTER INSERT ON requestbudget
FOR EACH ROW
BEGIN
   INSERT INTO audit (table_name, operation_type, new_data)
   VALUES ('requestbudget', 'INSERT', JSON_OBJECT(
      'RequestID', NEW.RequestID,
      'RequestedAmount', NEW.RequestedAmount,
      'RequestDate', NEW.RequestDate,
      'Status', NEW.Status,
      'BudgetID', NEW.BudgetID
   ));
END$$
DELIMITER ;

DELIMITER $$
CREATE TRIGGER requestbudget_update_audit_trigger
AFTER UPDATE ON requestbudget
FOR EACH ROW
BEGIN
   INSERT INTO audit (table_name, operation_type, old_data, new_data)
   VALUES ('requestbudget', 'UPDATE',
      JSON_OBJECT(
         'RequestID', OLD.RequestID,
         'RequestedAmount', OLD.RequestedAmount,
         'RequestDate', OLD.RequestDate,
         'Status', OLD.Status,
         'BudgetID', OLD.BudgetID
      ),
```

```
        JSON_OBJECT(
            'RequestID', NEW.RequestID,
            'RequestedAmount', NEW.RequestedAmount,
            'RequestDate', NEW.RequestDate,
            'Status', NEW.Status,
            'BudgetID', NEW.BudgetID
        )
    );
END$$
DELIMITER ;

DELIMITER $$
CREATE TRIGGER requestbudget_delete_audit_trigger
AFTER DELETE ON requestbudget
FOR EACH ROW
BEGIN
    INSERT INTO audit (table_name, operation_type, old_data)
    VALUES ('requestbudget', 'DELETE', JSON_OBJECT(
        'RequestID', OLD.RequestID,
        'RequestedAmount', OLD.RequestedAmount,
        'RequestDate', OLD.RequestDate,
        'Status', OLD.Status,
        'BudgetID', OLD.BudgetID
    ));
END$$
DELIMITER ;
```

*Table 5.3 5 Audit Trigger RequestBudget*

## 5.4 Application Modules

### 5.4.1 Module Volunteer

The Volunteer Module facilitates the management and coordination of volunteers in the FTMK FoodShare system. It allows newcomers to register as volunteers, with their registration verified by the Operation Manager. The Volunteer Manager oversees volunteer activities and approvals for departures. The module also enables the Operation Manager to track login activities and perform database maintenance, such as backups and recovery. Additionally, the module generates reports on registered volunteers based on the year, providing valuable insights for monitoring volunteer participation and trends.

### 5.4.2 Module Sponsorship

The sponsorship module is designed to facilitate and manage sponsor contribution effectively.It begins with sponsors registration and login into the system to make contribution either food or money. However, sponsors must first be verified by the Sponsorship Manager before they can log in to the system, ensuring that only verified sponsors can access. Once verified, sponsors can contribute food or money. After that, sponsor can track the current needs that highlight high priority needs to encourage timely support and maintains a detailed record of all contributions, including dates, amounts, and types. The module includes a contribution tracker, which provides sponsors with access to detailed logs of their contributions, categorized into a food contribution log and a money contribution log. This helps sponsors monitor their past contributions and stay informed about their impact. Additionally, sponsor have the option to reschedule the date for food donations, providing flexibility to align with their availability and circumstances. Other than that, sponsorship manager capabilities to track contribution, monitor the number of sponsors, total fund contribution, food contribution and can view through the chart. While, sponsor manager capability to add new departments to support organizational growth and generate detailed sponsorship reports and contribution report for tracking performance. Moreover, the system offers essential maintenance features such as backup and restore to ensuring data security and continuity and audit payment log to maintain a transparent record of financial transactions. With these features, the sponsorship module provides a secure, transparent and efficient framework for managing contributions, support both sponsors and sponsorship managers to achieving FoodShare’s objectives.

### 5.4.3 Module Inventory

The Inventory Module in the FTMK FoodShare system is designed to efficiently manage and track food donations and supplies. It categorizes and monitors food items, ensuring accurate stock levels and timely redistribution. It enables the user to log and update donated items with relevant details such as type and quantity. The module generates comprehensive transaction reports, offering insights into food donations quantity, and distribution needs. With security features for controlled access, the Inventory Module ensures efficient food management and supports effective distribution

### 5.4.4 Module Budget

The budget module in the FTMK FoodShare system is intended to manage financial operations that include the Chief Treasurer and Assistant Treasurer. The Chief Treasurer is in charge of overseeing budget approvals, managing allocations, and ensuring accurate financial planning through connection with the Fund Contribution Table. This job involves creating, modifying, deleting, and viewing budget records. The Assistant Treasurer oversees budget requests and expenditure management, ensuring that all spending is in line with the approved budget. The module automates computations to track budget utilisation, remaining funds, and overall expenses, delivering real-time results. Both roles can conduct CRUD operations to effectively manage financial records. The system offers precise data on budgets, expenses, and residual funds, which promote transparency and accountability. The Budget Module promotes efficient financial management by automating budget requests, approvals, and spending tracking to help organisations achieve their objectives.

## 5.5 User Interface

### 5.5.1 Module Volunteer

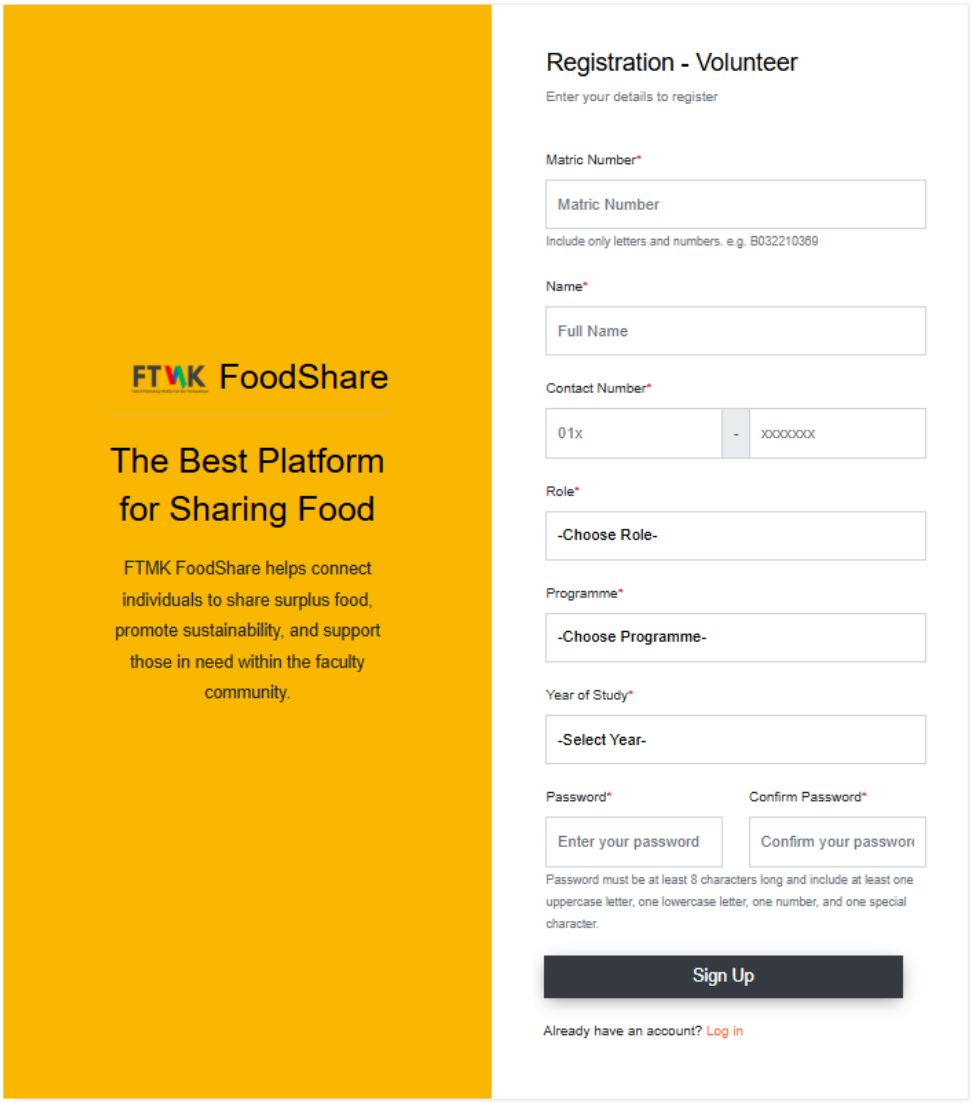



*Figure 5.5.1.1 Registration Page for Volunteer*

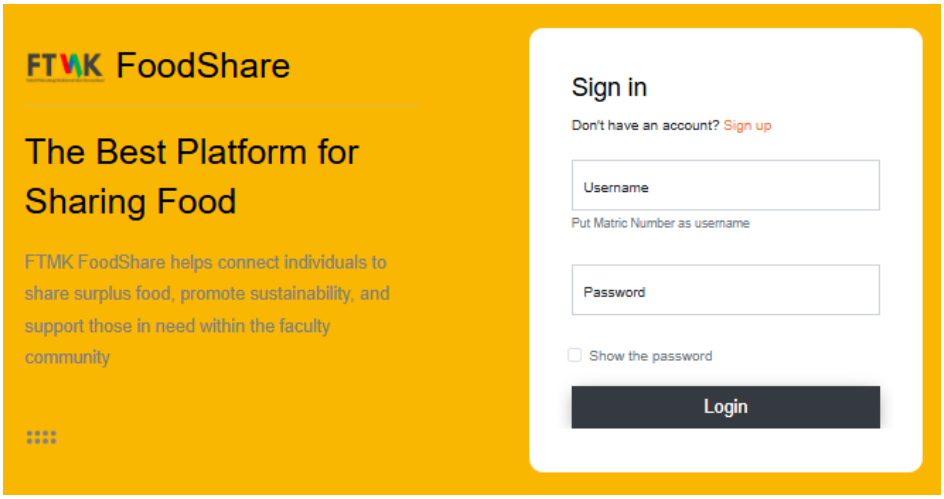



*Figure 5.5.1.2 Login Page for Volunteer*

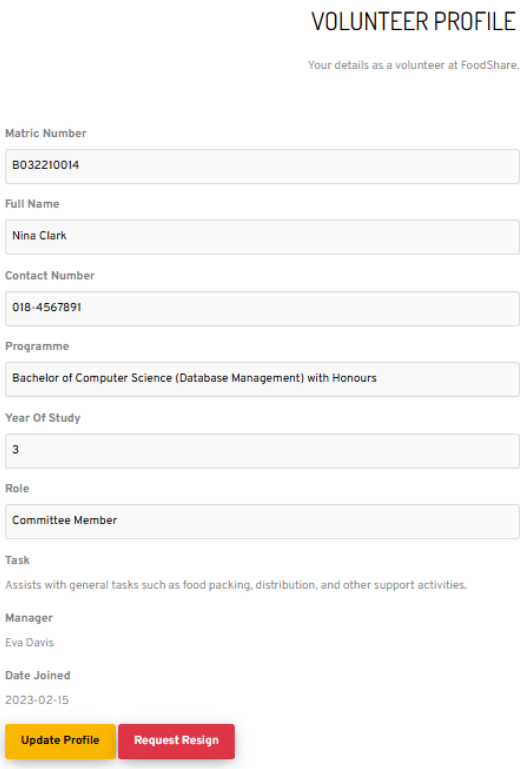



*Figure 5.5.1.4 Volunteer Profile Page*

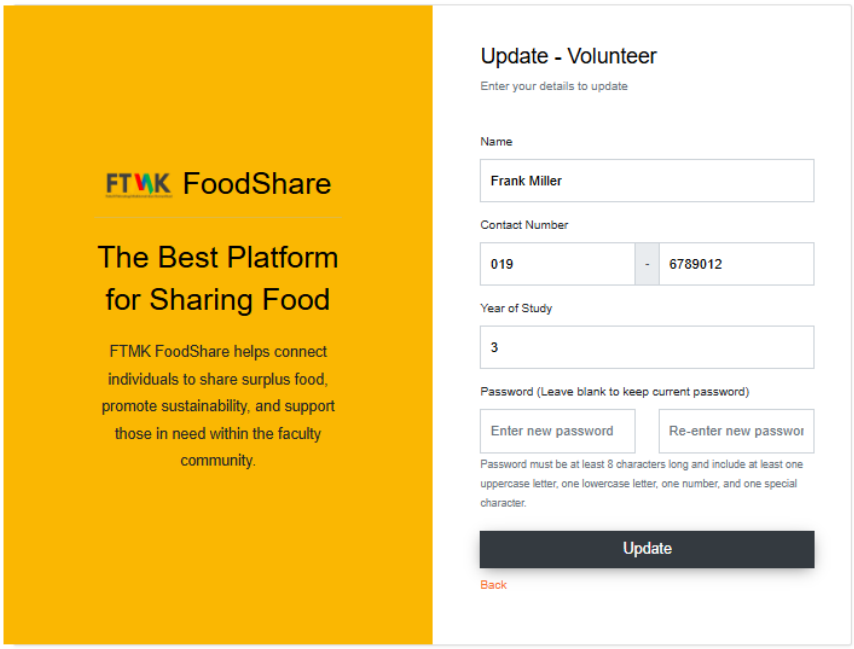



*Figure 5.5.1.5 Update Profile Page for Volunteer*

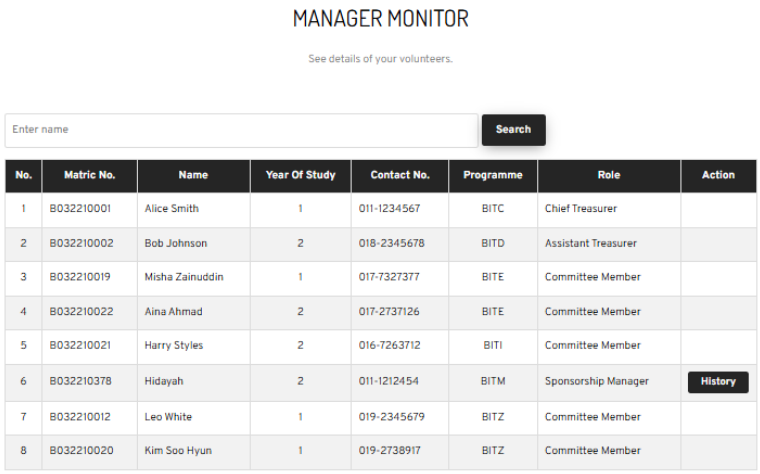



*Figure 5.5.1.6 Manager Monitor Page for Volunteer Manager*

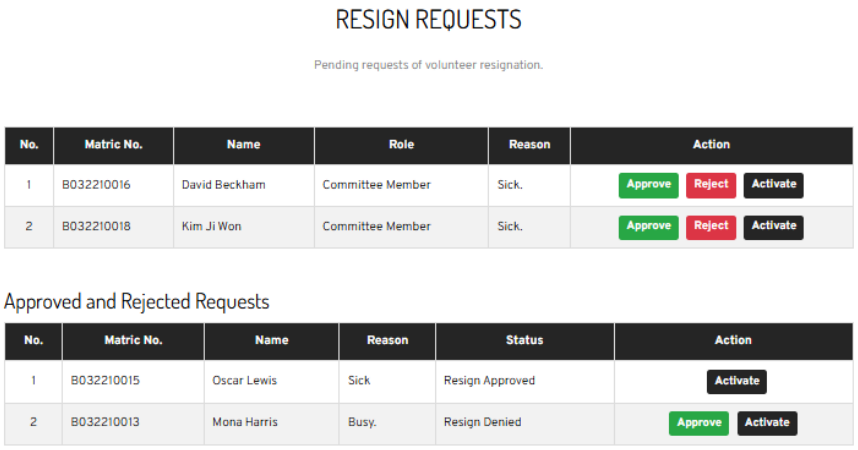
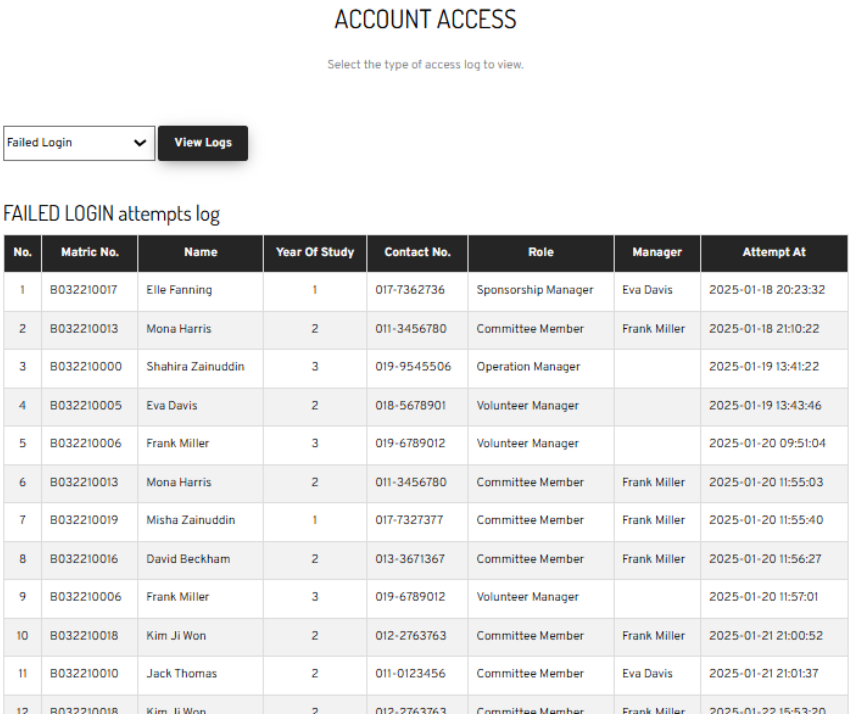



*Figure 5.5.1.7 Resign Requests Page for Volunteer Manager*


*Figure 5.5.1.8 Volunteer Analysis Report Page for Volunteer Manager*




| No. | Matric No. | Name | Year Of Study | Contact No. | Role | Manager | Attempt At |
|---|---|---|---|---|---|---|---|
| 1 | B032210017 | Elle Fanning | 1 | 017-7362736 | Sponsorship Manager | Eva Davis | 2025-01-18 20:23:32 |
| 2 | B032210013 | Mona Harris | 2 | 011-3456780 | Committee Member | Frank Miller | 2025-01-18 21:10:22 |
| 3 | B032210000 | Shahira Zainuddin | 3 | 019-9545506 | Operation Manager | | 2025-01-19 13:41:22 |
| 4 | B032210005 | Eva Davis | 2 | 018-5678901 | Volunteer Manager | | 2025-01-19 13:43:46 |
| 5 | B032210006 | Frank Miller | 3 | 019-6789012 | Volunteer Manager | | 2025-01-20 09:51:04 |
| 6 | B032210013 | Mona Harris | 2 | 011-3456780 | Committee Member | Frank Miller | 2025-01-20 11:55:03 |
| 7 | B032210019 | Misha Zainuddin | 1 | 017-7327377 | Committee Member | Frank Miller | 2025-01-20 11:55:40 |
| 8 | B032210016 | David Beckham | 2 | 013-3671367 | Committee Member | Frank Miller | 2025-01-20 11:56:27 |
| 9 | B032210006 | Frank Miller | 3 | 019-6789012 | Volunteer Manager | | 2025-01-20 11:57:01 |
| 10 | B032210018 | Kim Ji Won | 2 | 012-2763763 | Committee Member | Frank Miller | 2025-01-21 21:00:52 |
| 11 | B032210010 | Jack Thomas | 2 | 011-0123456 | Committee Member | Eva Davis | 2025-01-21 21:01:37 |
| 12 | B032210018 | Kim Ji Won | 2 | 012-2763763 | Committee Member | Frank Miller | 2025-01-22 15:53:20 |

*Figure 5.5.1.9 Account Access Page for Operation Manager*

## 5.5.2 Module Sponsorship

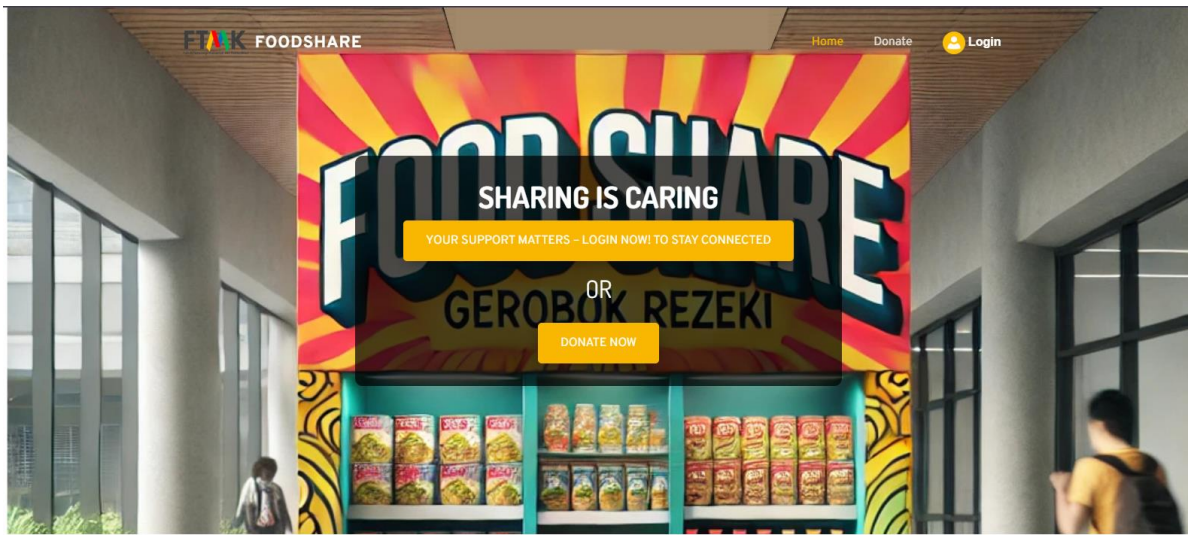



*Figure 5.5.2. 1 Home page*

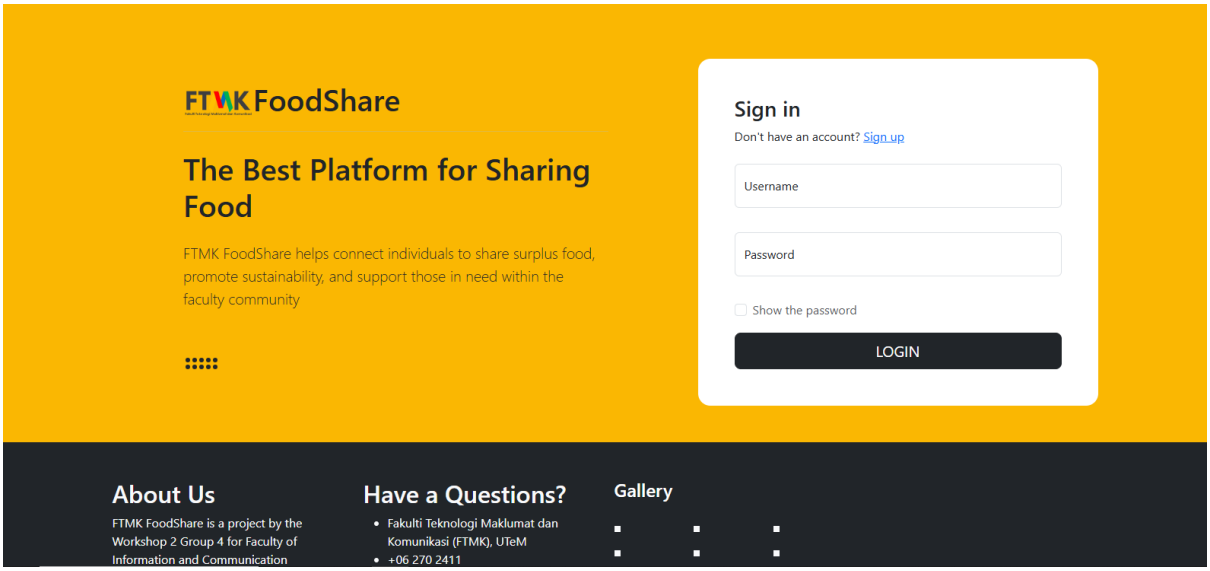



*Figure 5.5.2. 2 Login Page*

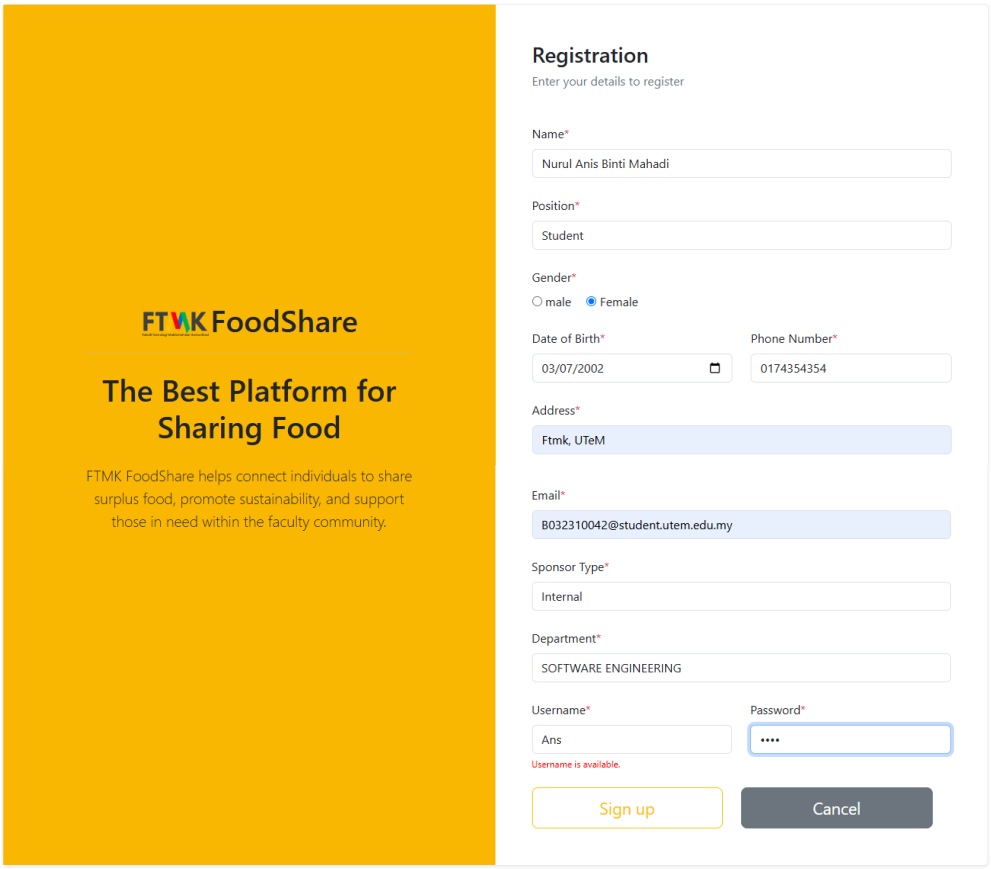



*Figure 5.5.2. 3 Sponsor Registration Page*


*Figure 5.5.2. 4 Sponsor Home Page*

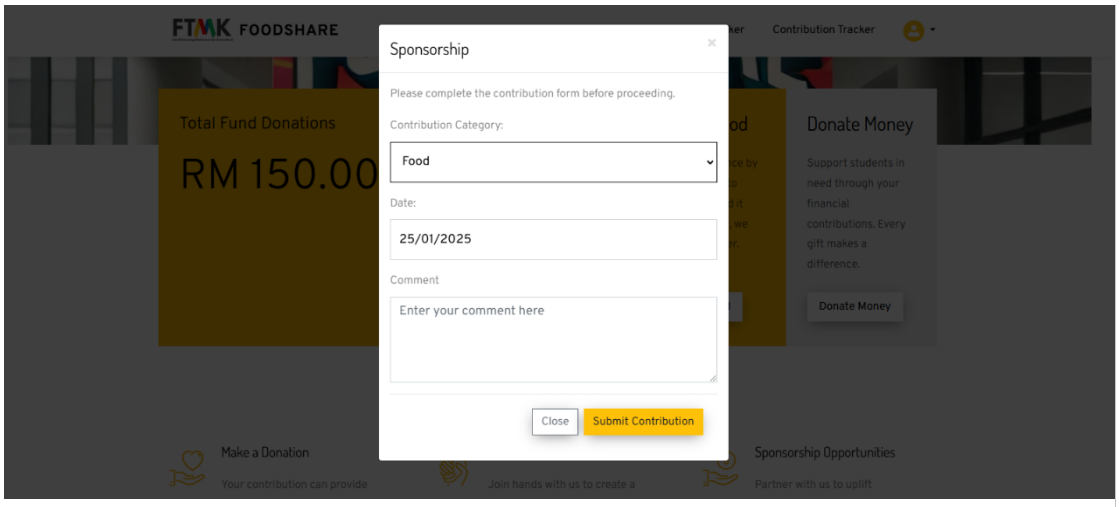



*Figure 5.5.2. 5 Contrinution Form selcted by food*

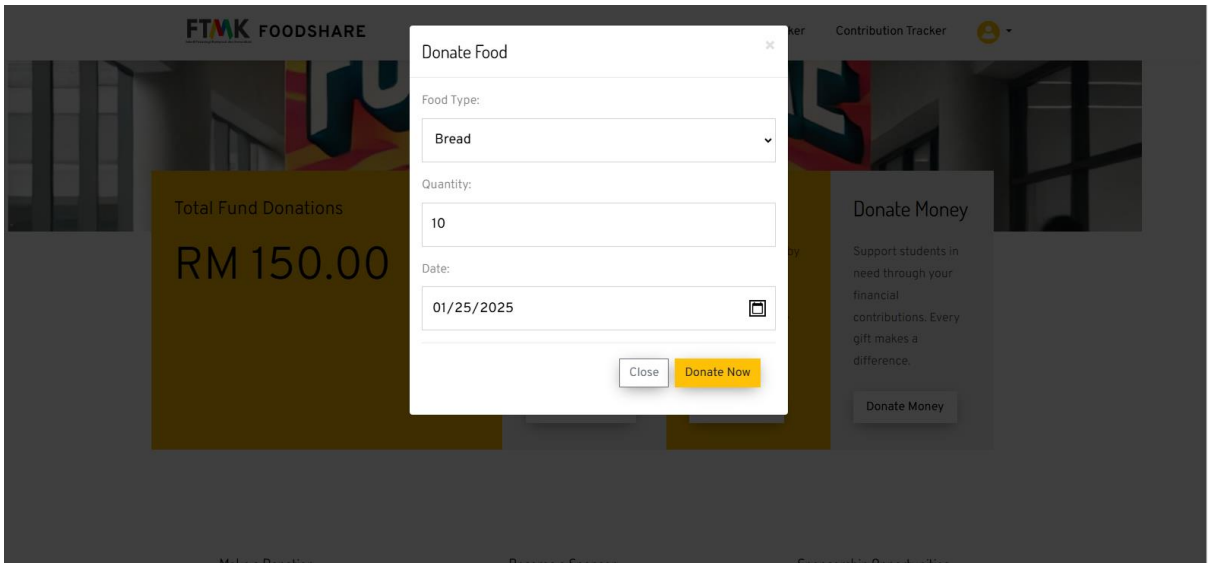



*Figure 5.5.2. 6  Food Contribution Form*

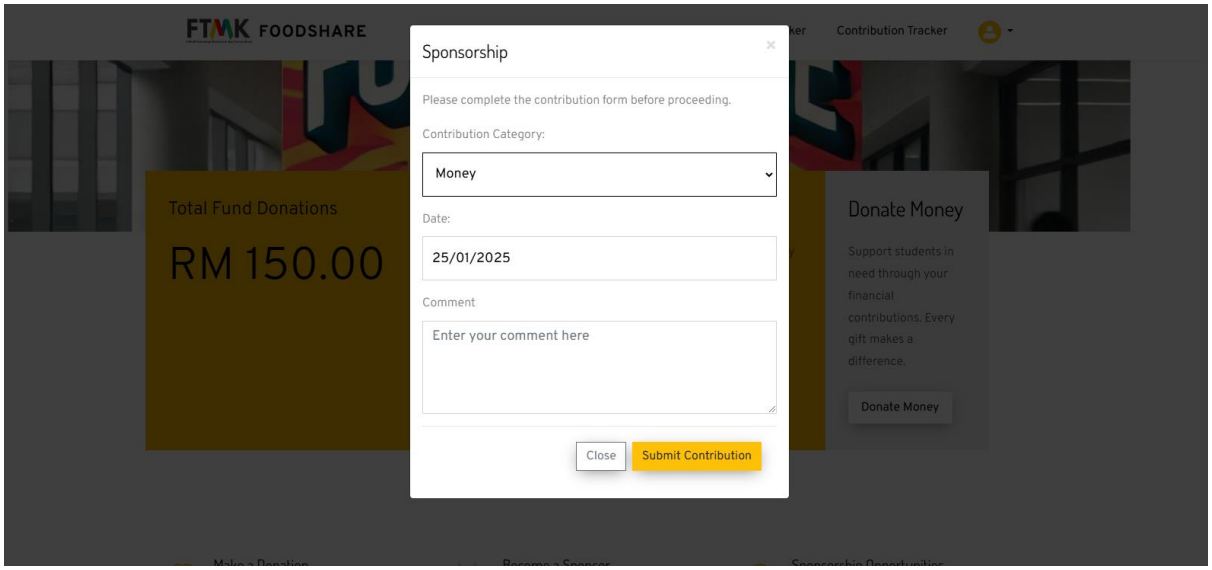



*Figure 5.5.2. 7 Contribution Form selected by money*

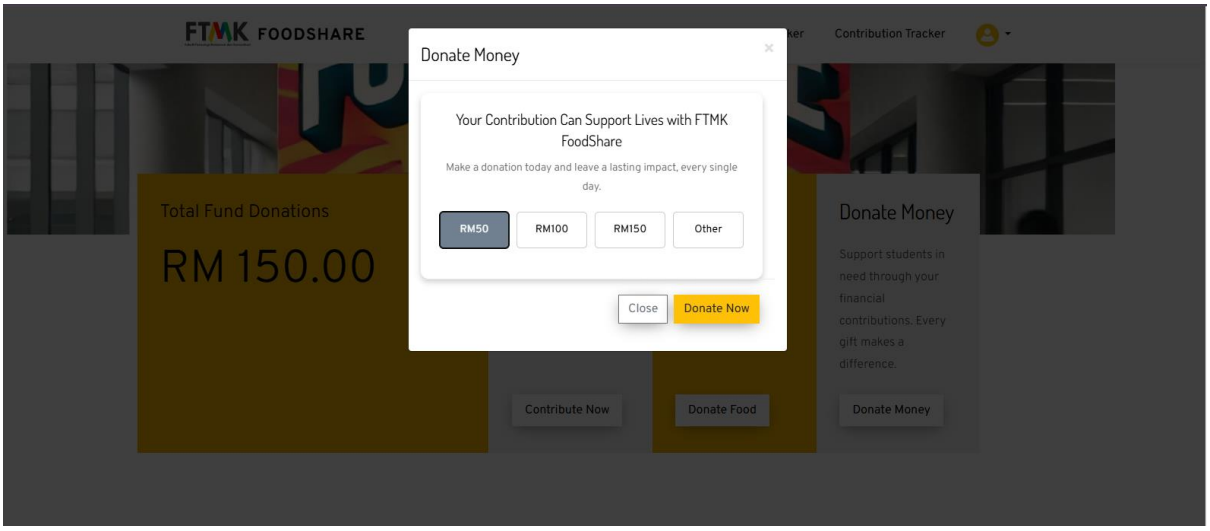



*Figure 5.5.2. 8  Donation money feature*

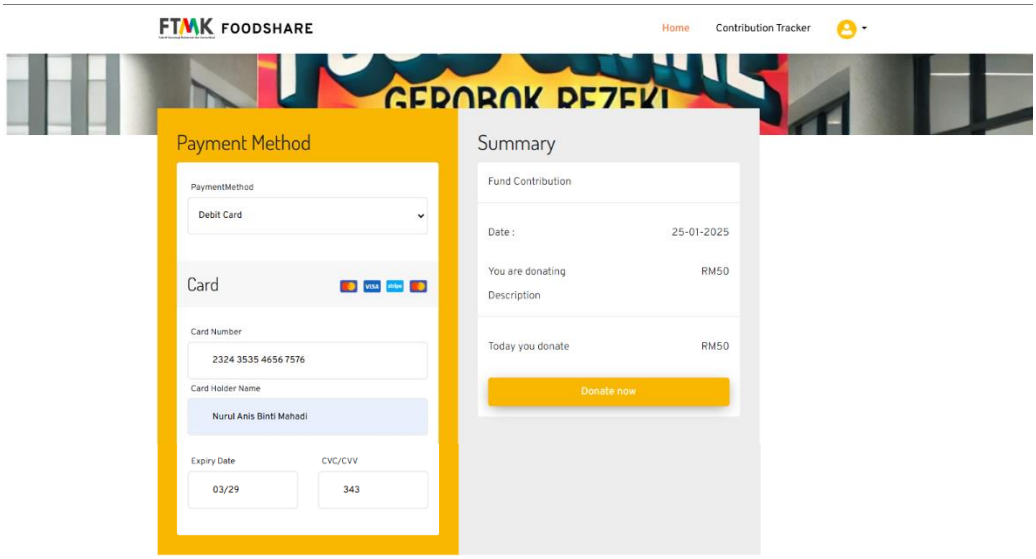



*Figure 5.5.2. 9 Payment form and summary receipt*

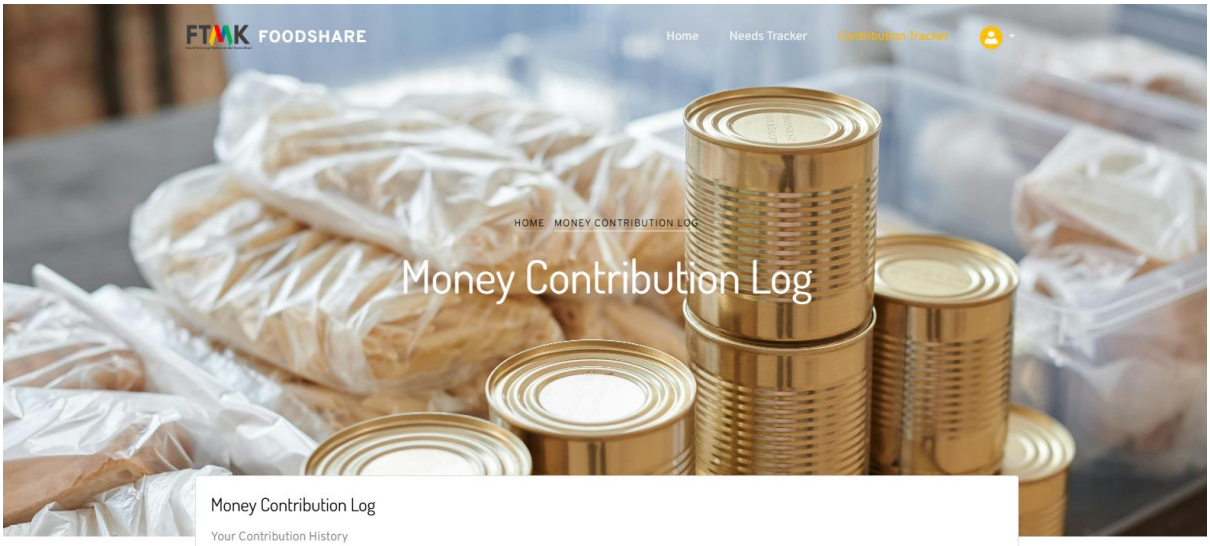



*Figure 5.5.2. 10 Money Contribution Log*

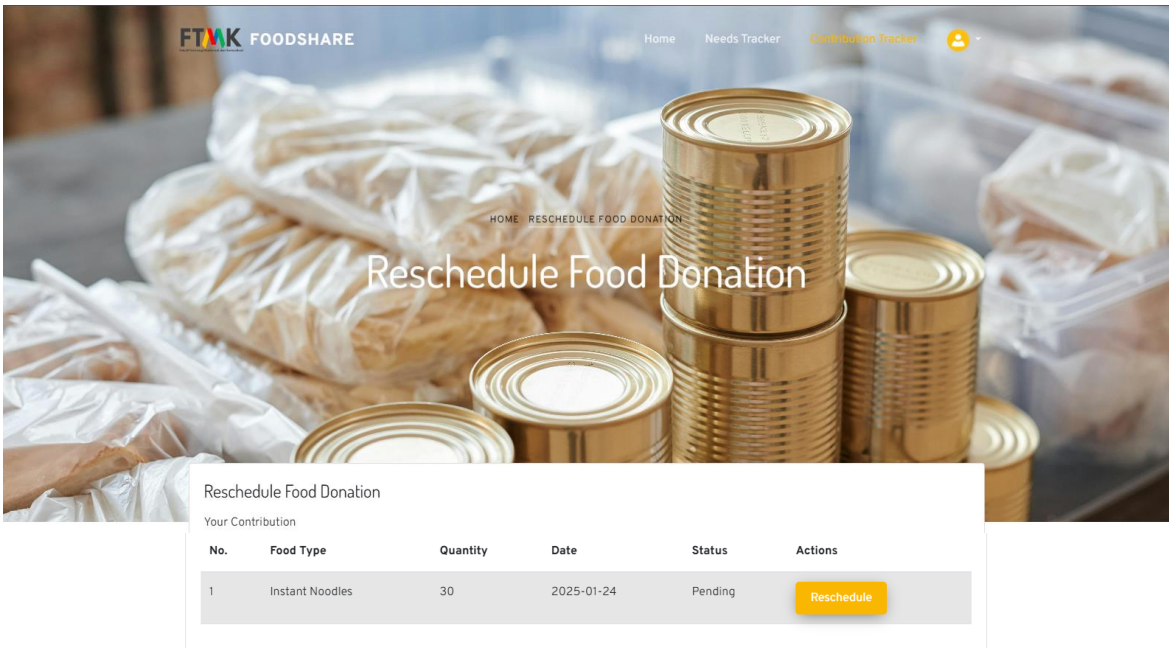



*Figure 5.5.2. 11 Rechedule Food Donation*

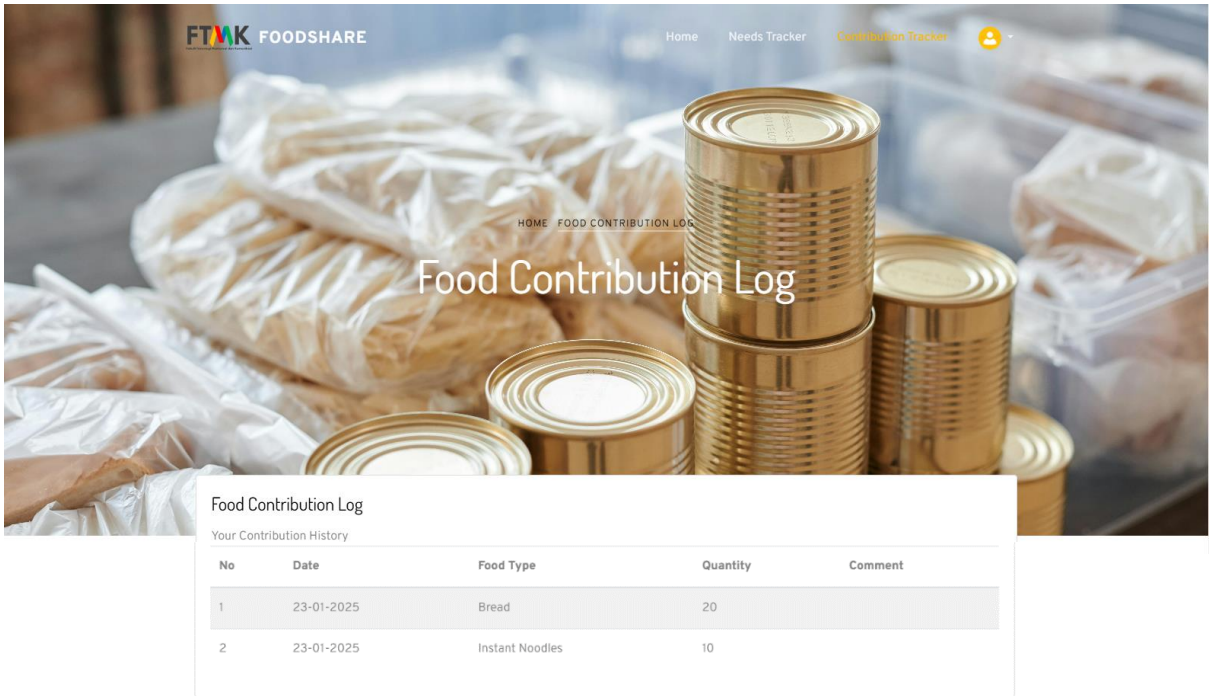



*Figure 5.5.2. 12 Food Contribution Log*

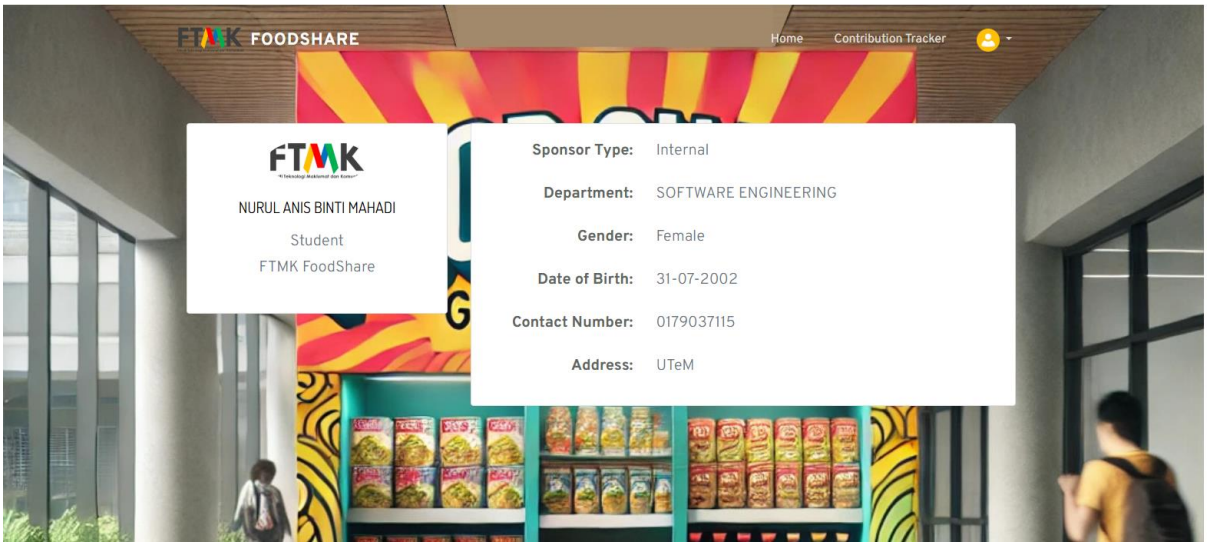



*Figure 5.5.2. 11 Sponsor Profile*

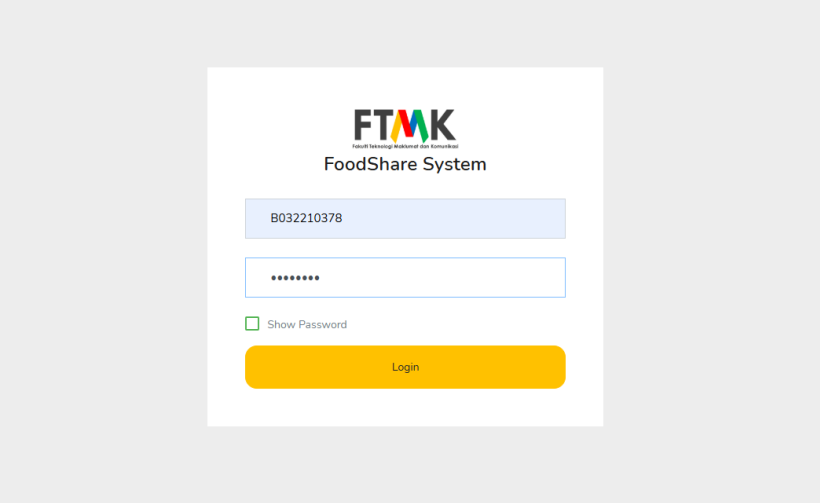



*Figure 5.5.2. 12 Admin Login*

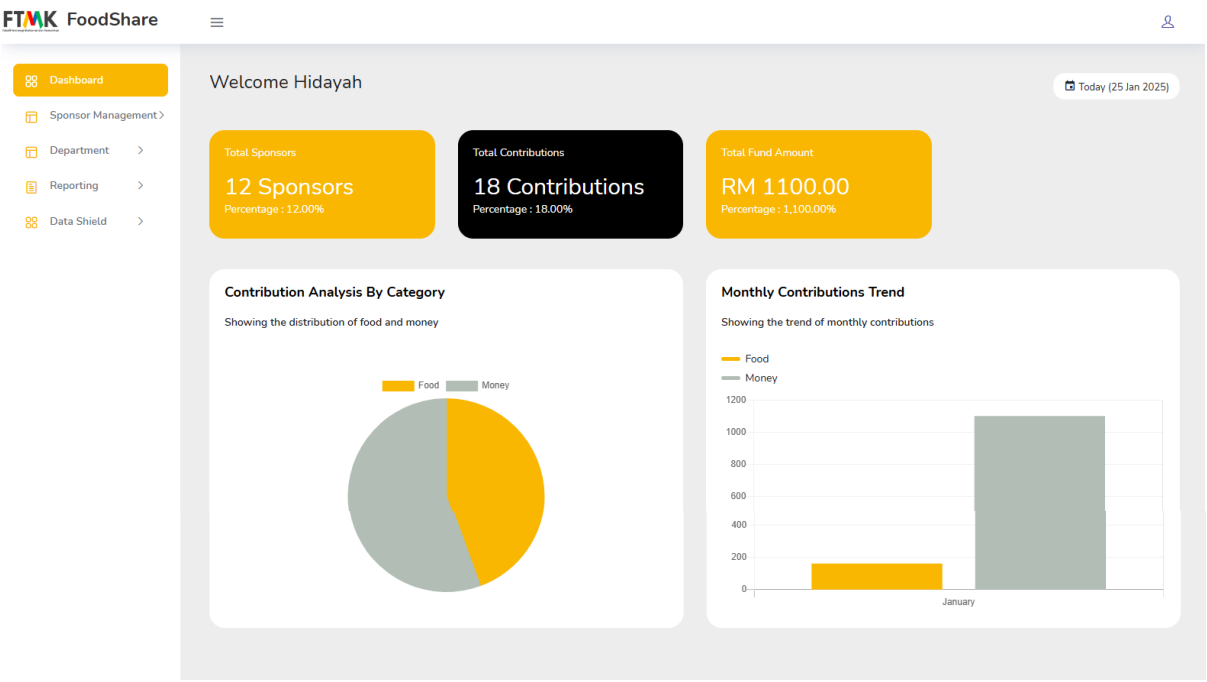



*Figure 5.5.2. 13 Sponsorship Manager Dashboard*

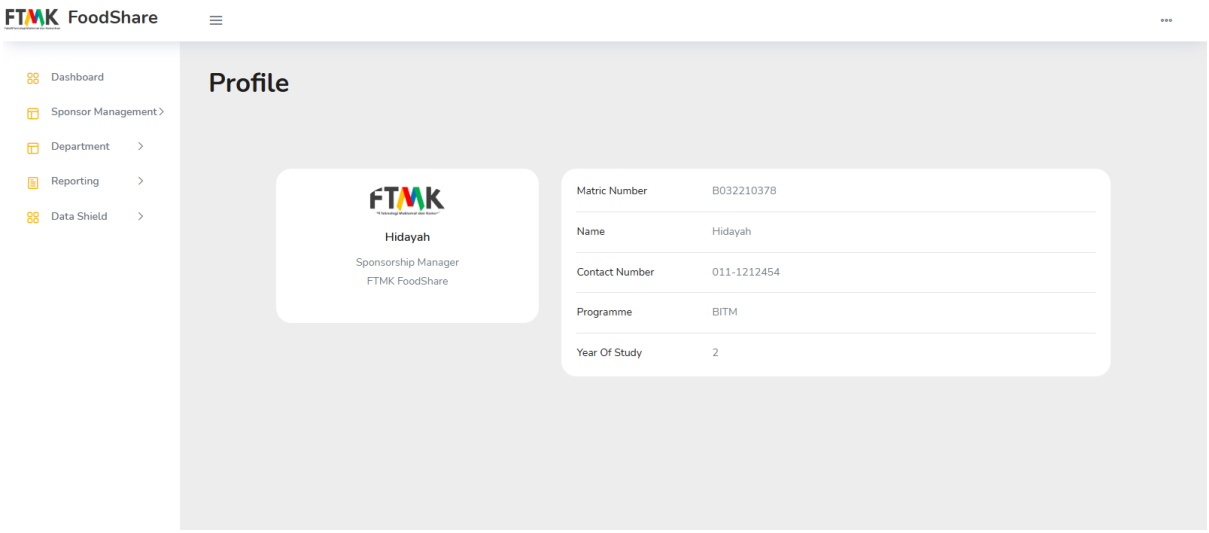



*Figure 5.5.2. 14  Sponsorship Manager Profiles*

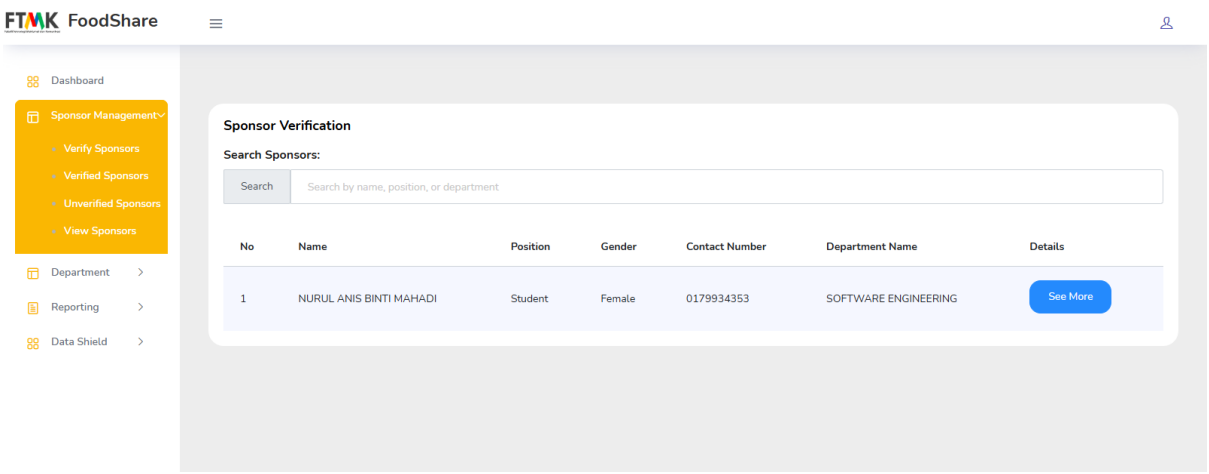



*Figure 5.5.2. 15 Manage Sponsor for verification*

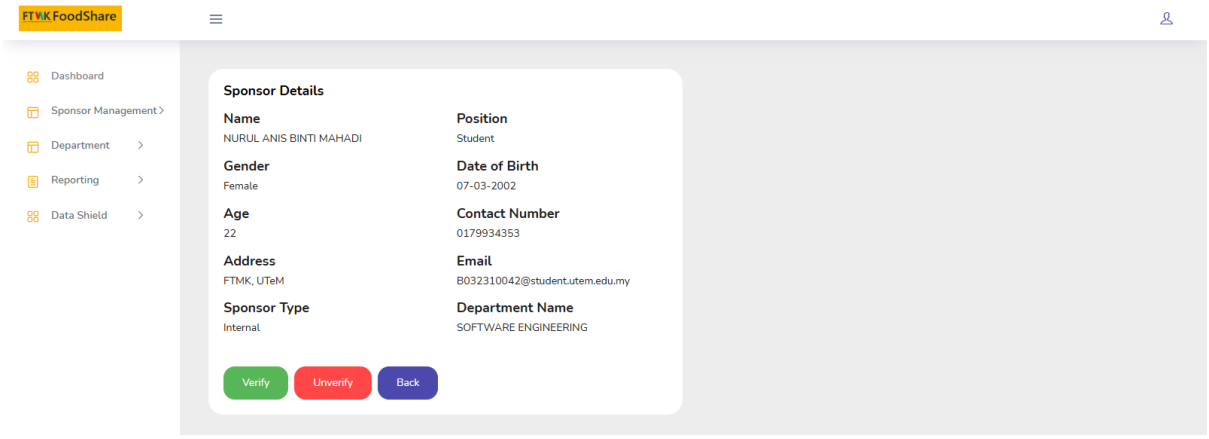



*Figure 5.5.2. 16 Sponsor Detail for verification*

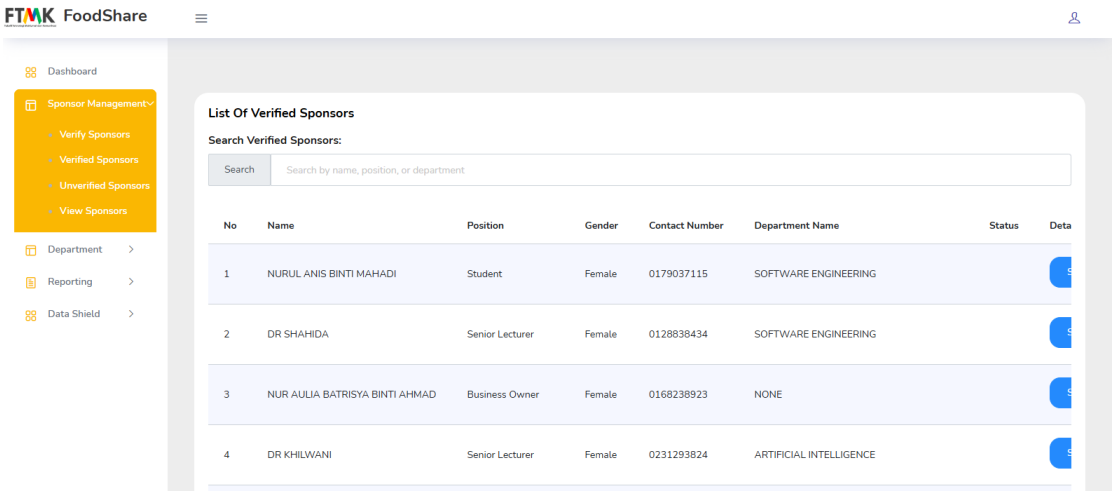



*Figure 5.5.2.17 List of verified Sponsor*

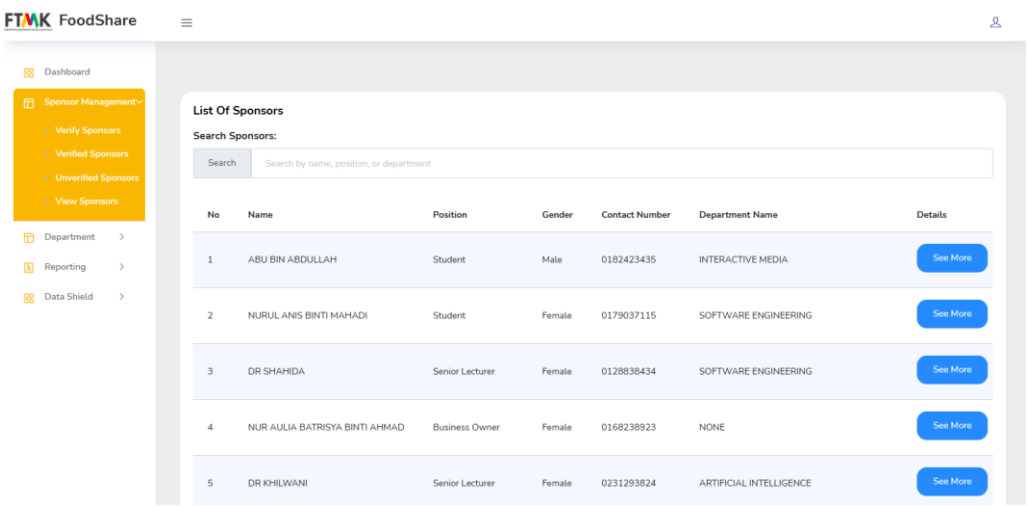



*Figure 5.5.2. 18 List Of  Sponsor*

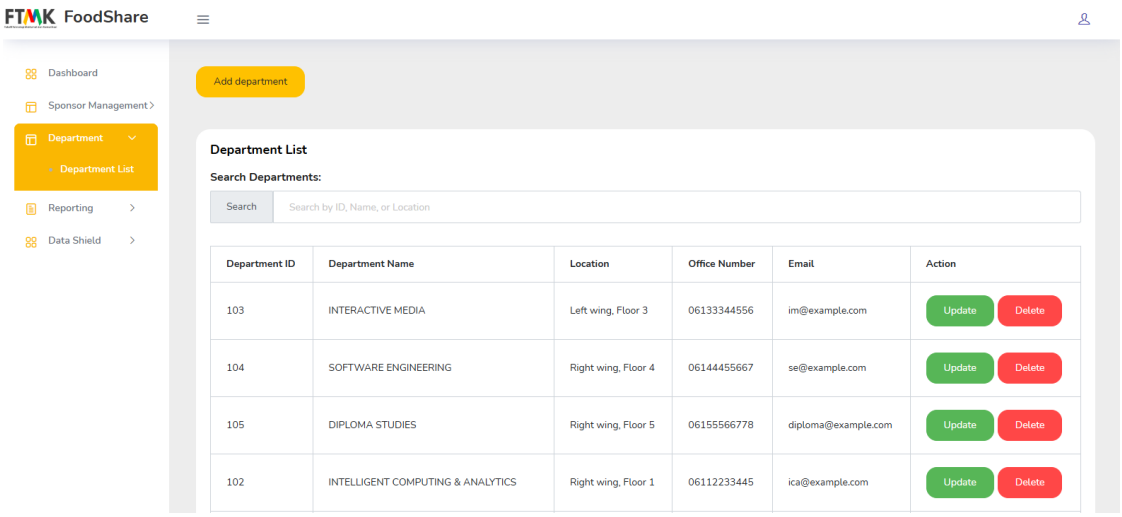



*Figure 5.5.2. 19 Department page*

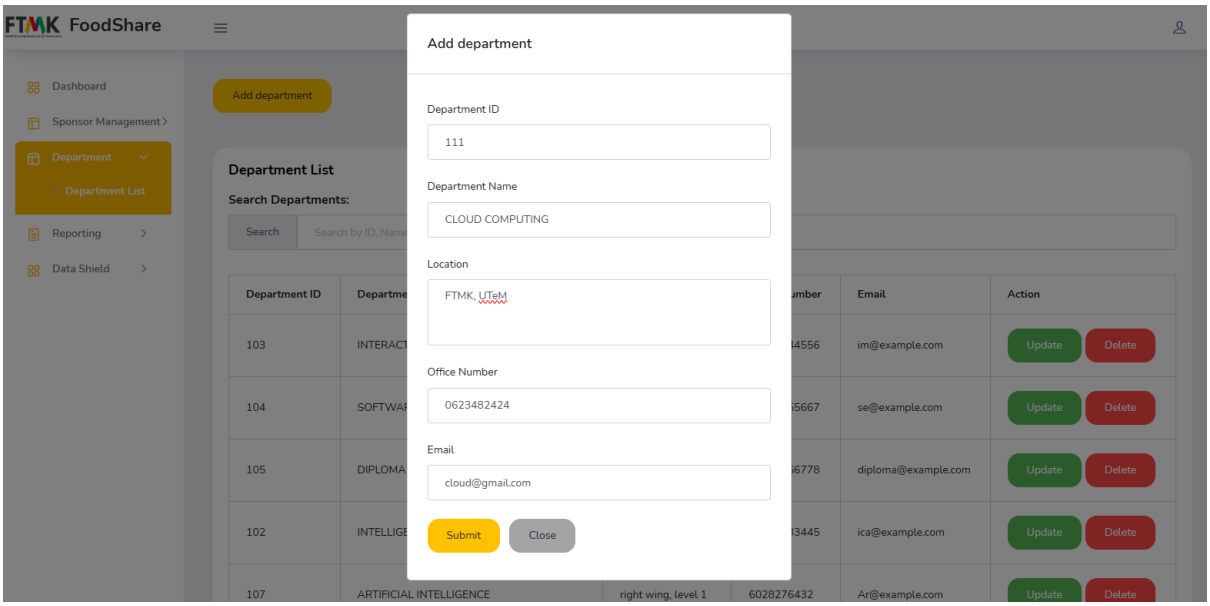



*Figure 5.5.2. 10 Add New Department Form*

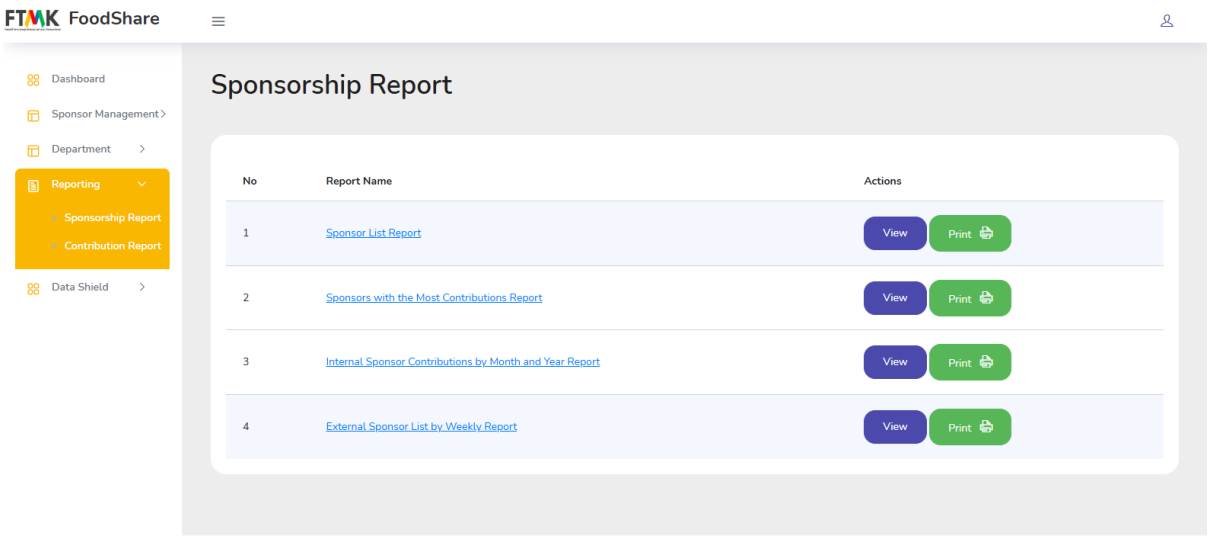



*Figure 5.5.2. 11 Sponsorship Report*

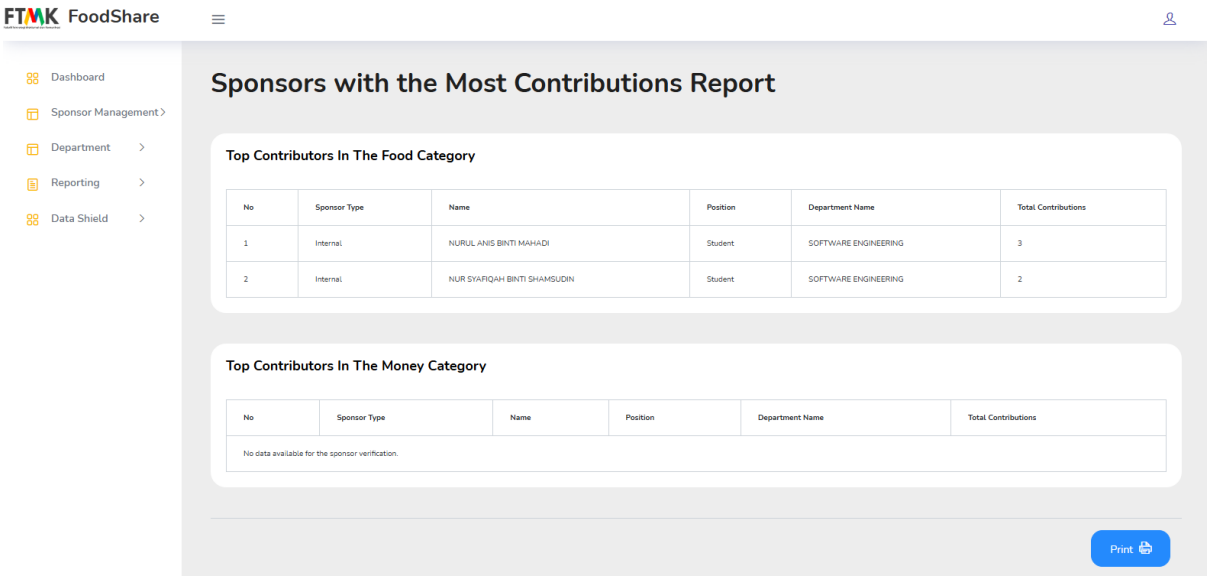



*Figure 5.5.2. 12 Detail of Sponsorship Report*

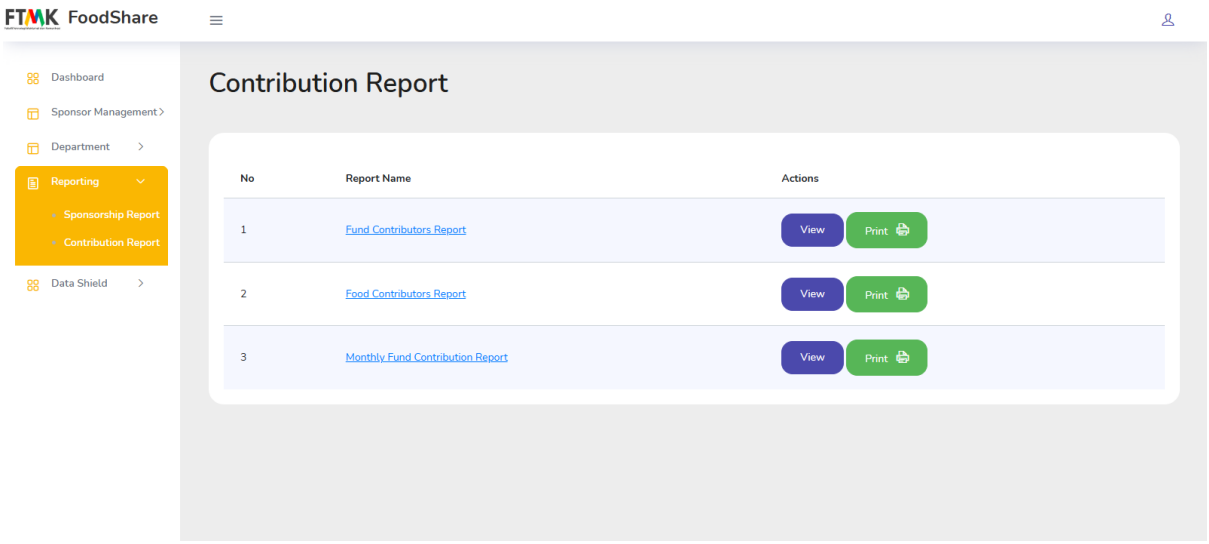



*Figure 5.5.2. 13 Contribution Report*

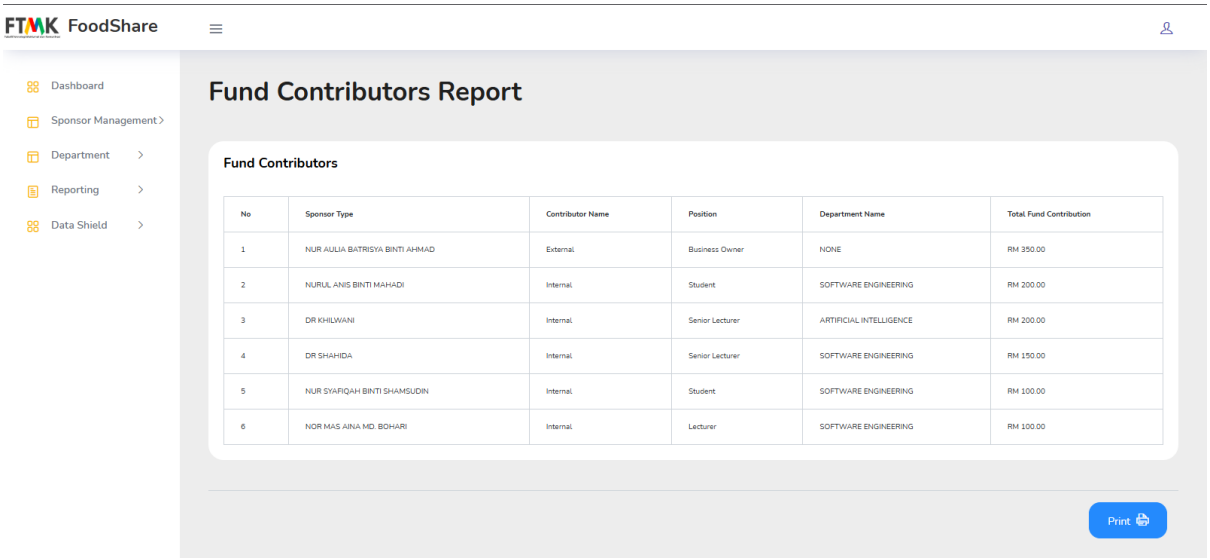



*Figure 5.5.2. 14 Detail of Contribution Report*

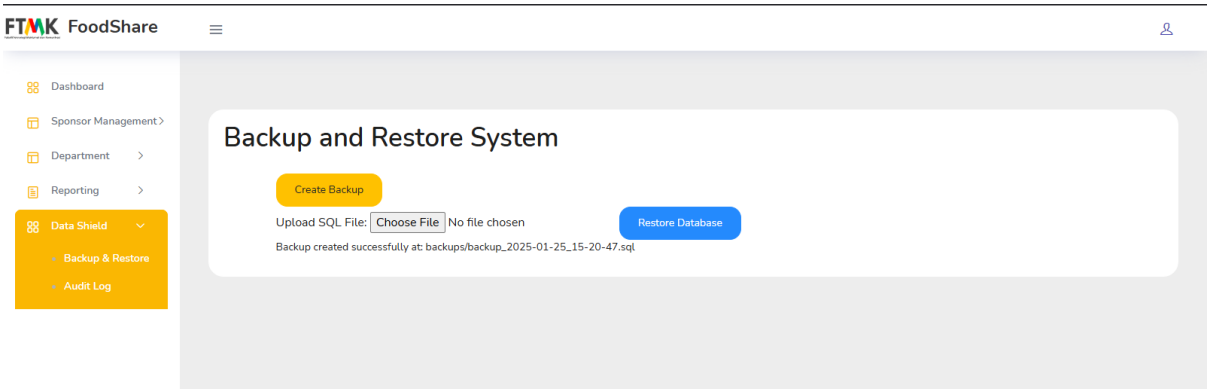
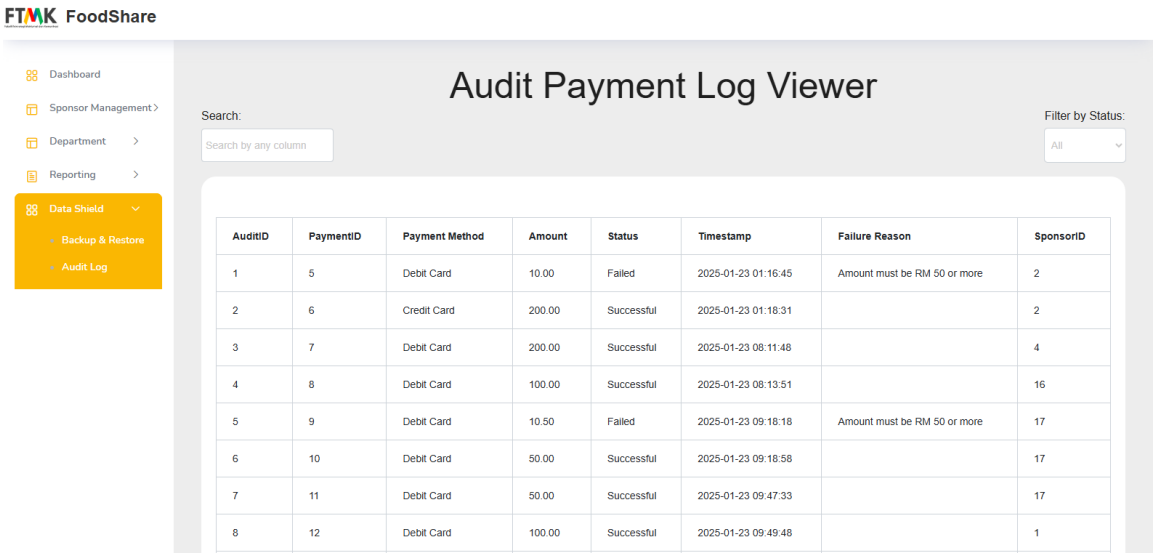



*Figure 5.5.2. 15 Backup and Restore page*




| AuditID | PaymentID | Payment Method | Amount | Status | Timestamp | Failure Reason | SponsorID |
|---|---|---|---|---|---|---|---|
| 1 | 5 | Debit Card | 10.00 | Failed | 2025-01-23 01:16:45 | Amount must be RM 50 or more | 2 |
| 2 | 6 | Credit Card | 200.00 | Successful | 2025-01-23 01:18:31 | | 2 |
| 3 | 7 | Debit Card | 200.00 | Successful | 2025-01-23 08:11:48 | | 4 |
| 4 | 8 | Debit Card | 100.00 | Successful | 2025-01-23 08:13:51 | | 16 |
| 5 | 9 | Debit Card | 10.50 | Failed | 2025-01-23 09:18:18 | Amount must be RM 50 or more | 17 |
| 6 | 10 | Debit Card | 50.00 | Successful | 2025-01-23 09:18:58 | | 17 |
| 7 | 11 | Debit Card | 50.00 | Successful | 2025-01-23 09:47:33 | | 17 |
| 8 | 12 | Debit Card | 100.00 | Successful | 2025-01-23 09:49:48 | | 1 |

*Figure 5.5.2. 16 Audit Payment Log Viewer Log*

### 5.5.3 Module Inventory

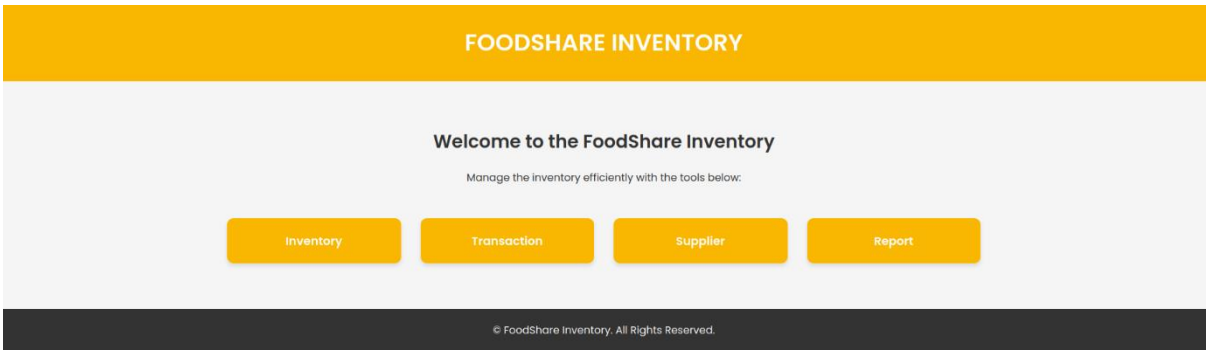



*Figure 5.5.3.1 Main page with navigations*

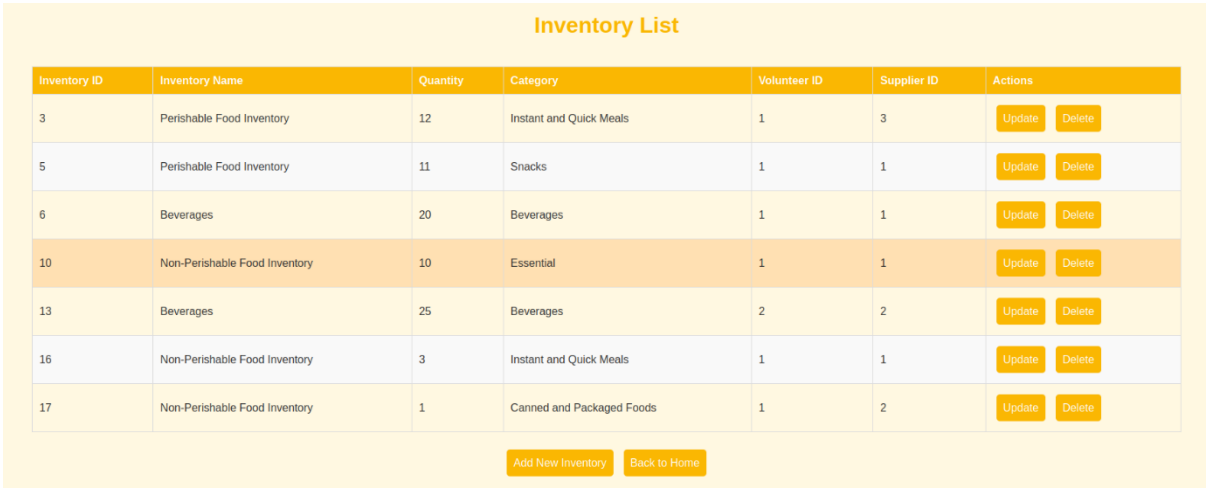



*Figure 5.5.3.2 Inventory page*

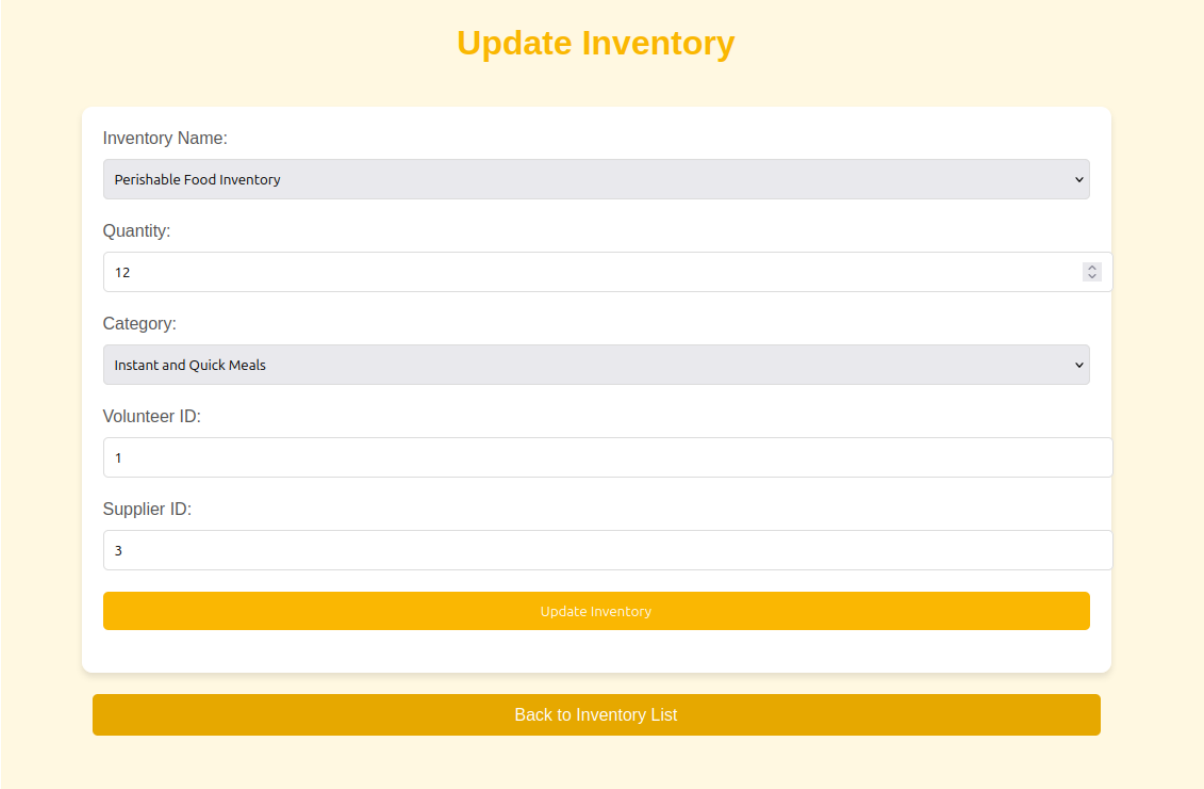



*Figure 5.5.3.3 Inventory update page*

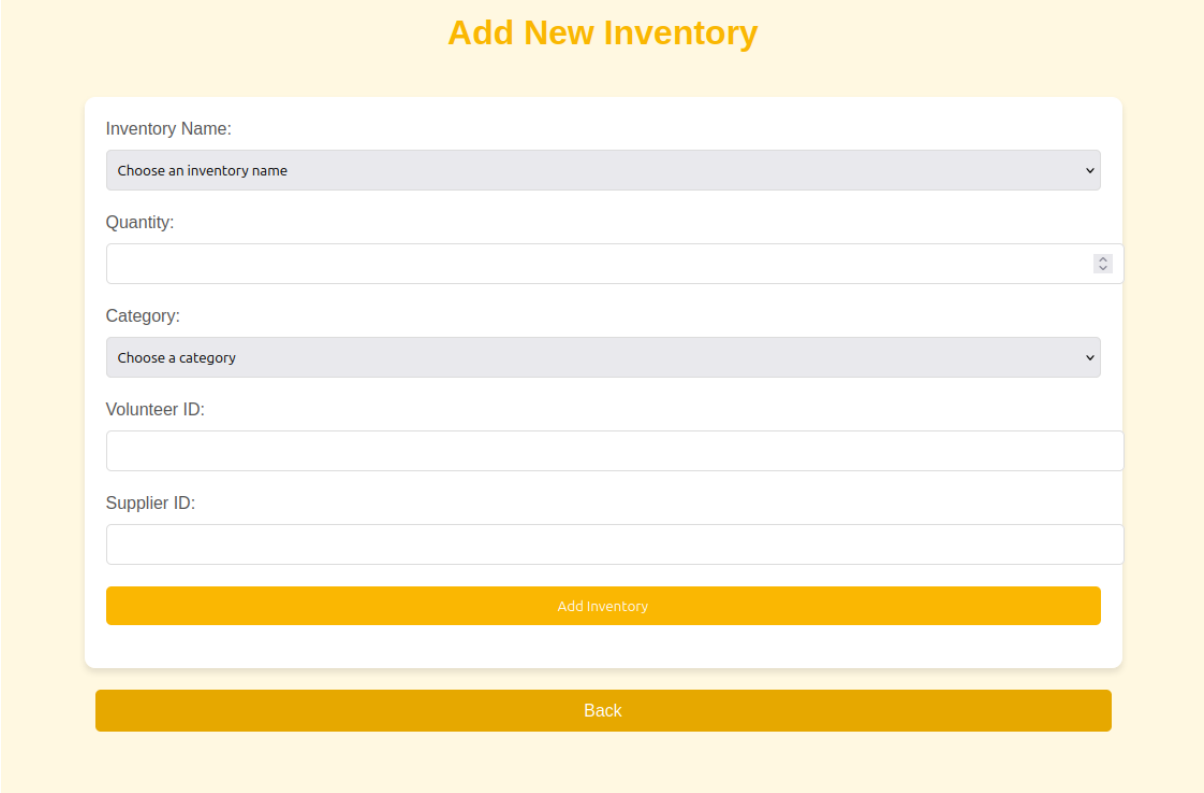



*Figure 5.5.3.4 Add inventory page*

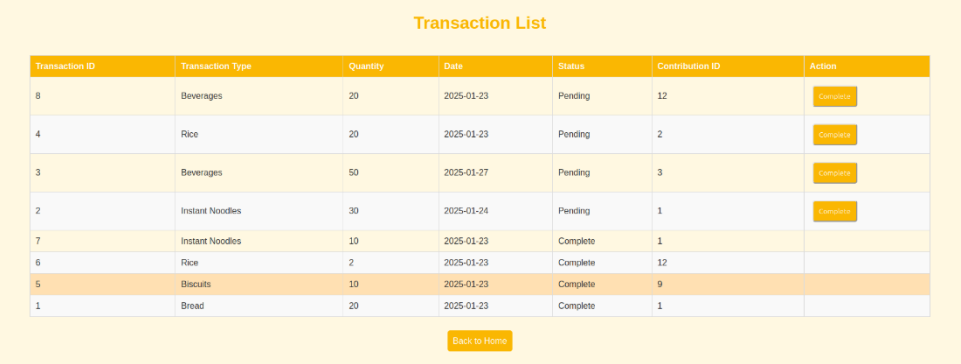



*Figure 5.5.3.5 Transaction page*

## 5.5.4 Module Budget

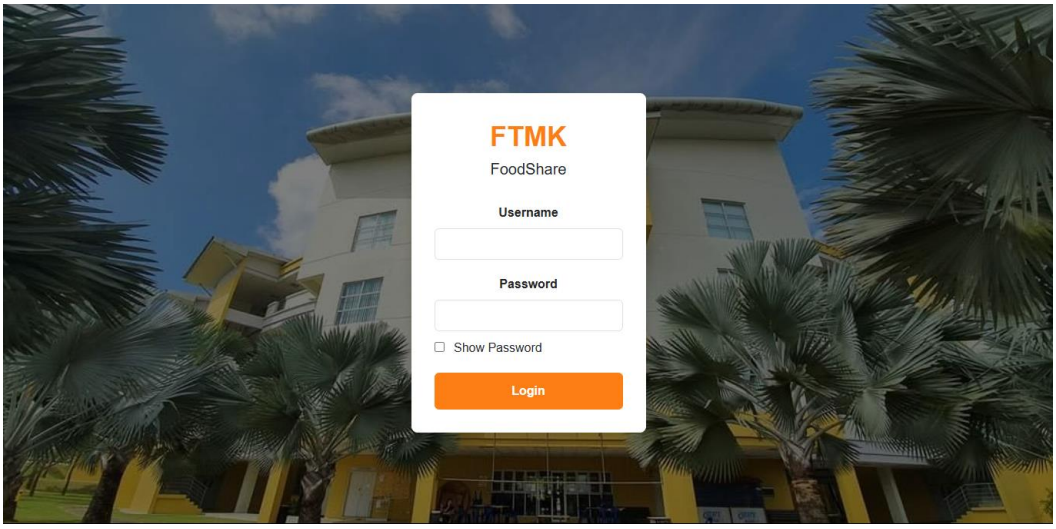



*Figure 5.5.3. 1 Login Page for Chief and Assistant Treasurer*

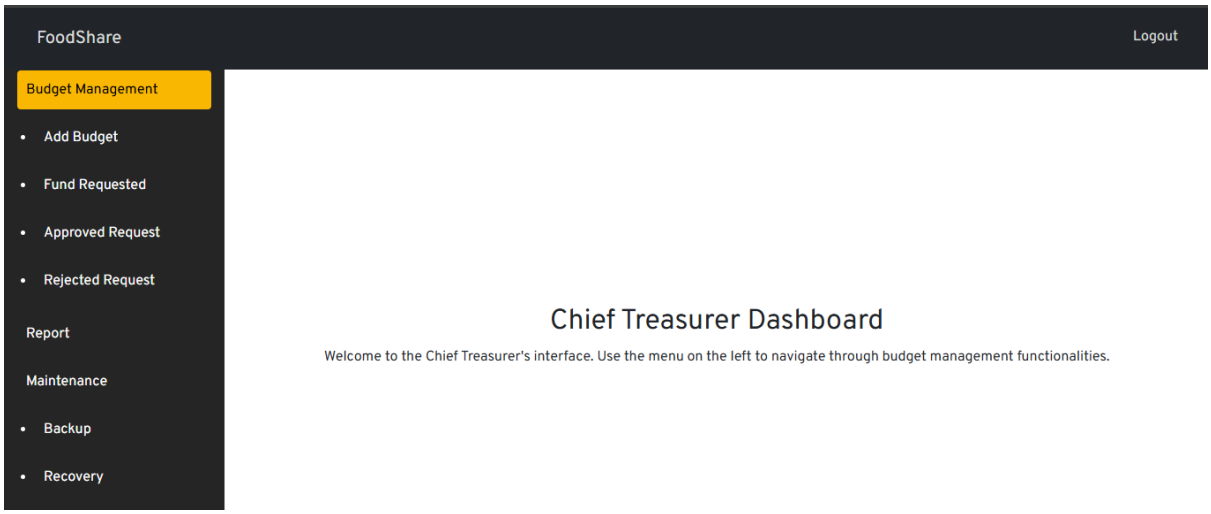



*Figure 5.5.3. 2 Chief Treasurer Dashboard*

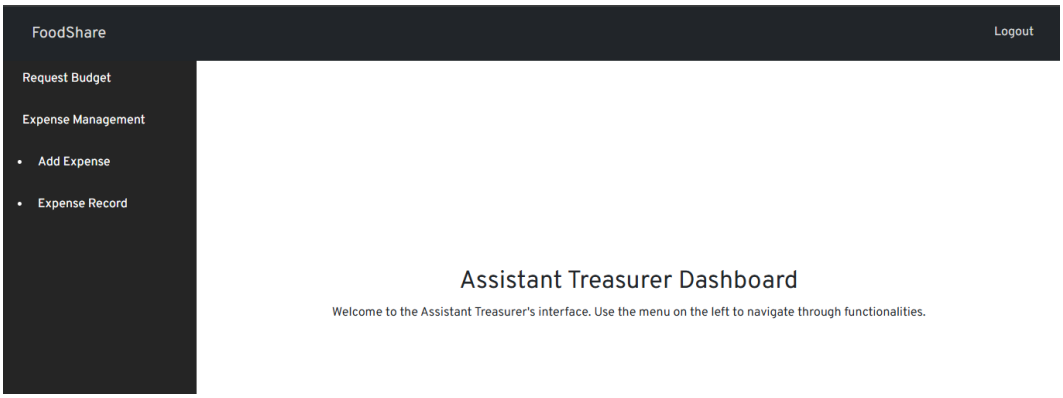



*Figure 5.5.3. 3 Assistant Treasurer Dashboard*

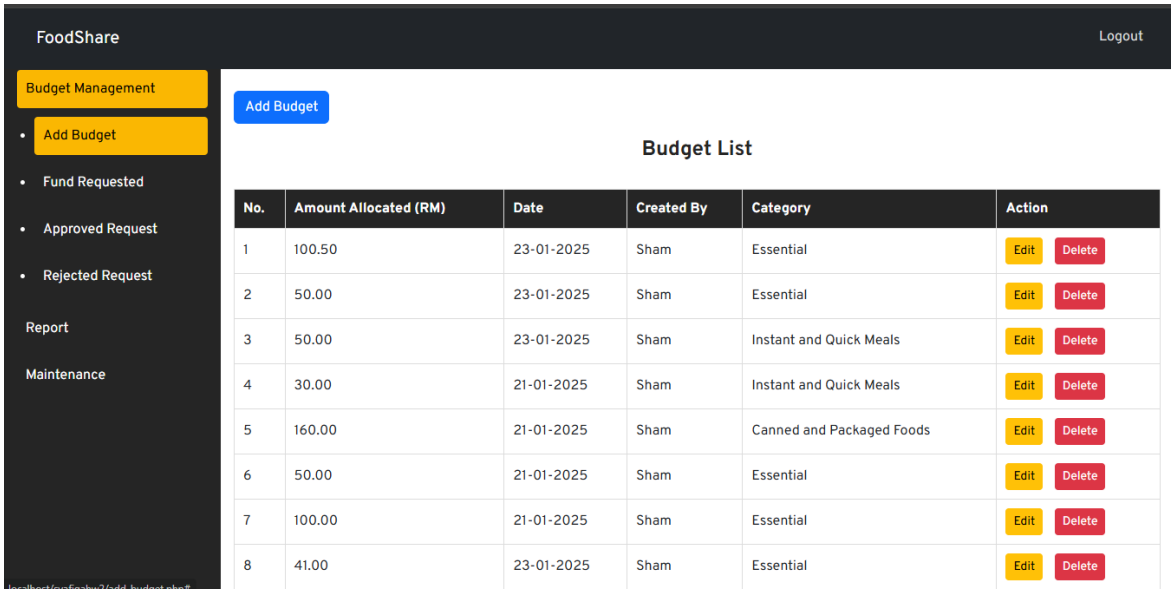



*Figure 5.5.3. 4 Add Budget Page*

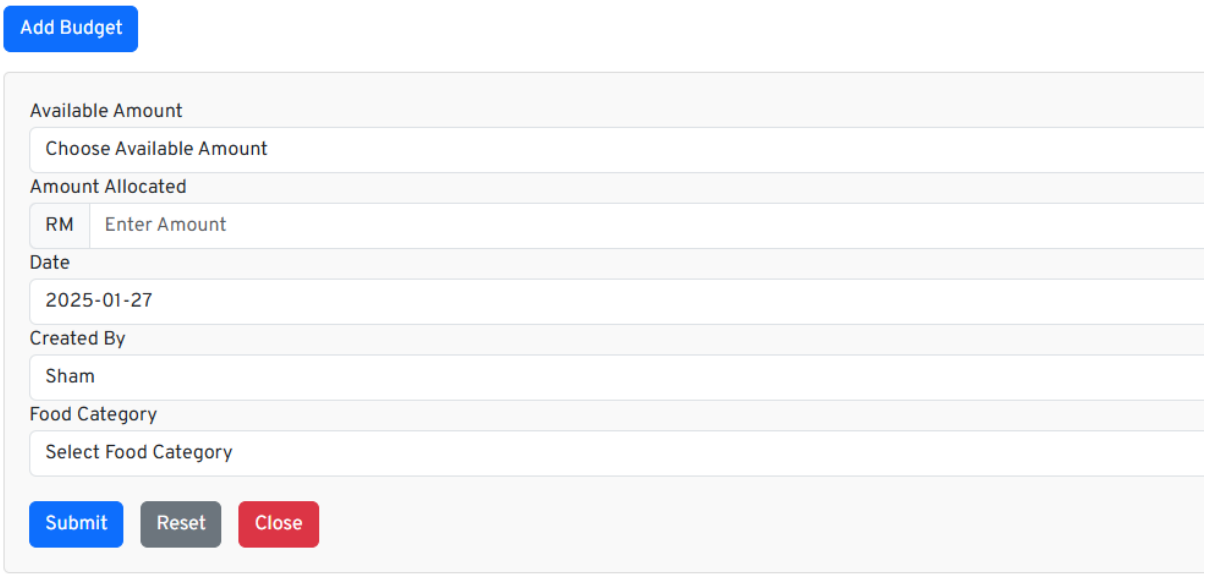



*Figure 5.5.3. 5 Add Budget Form*

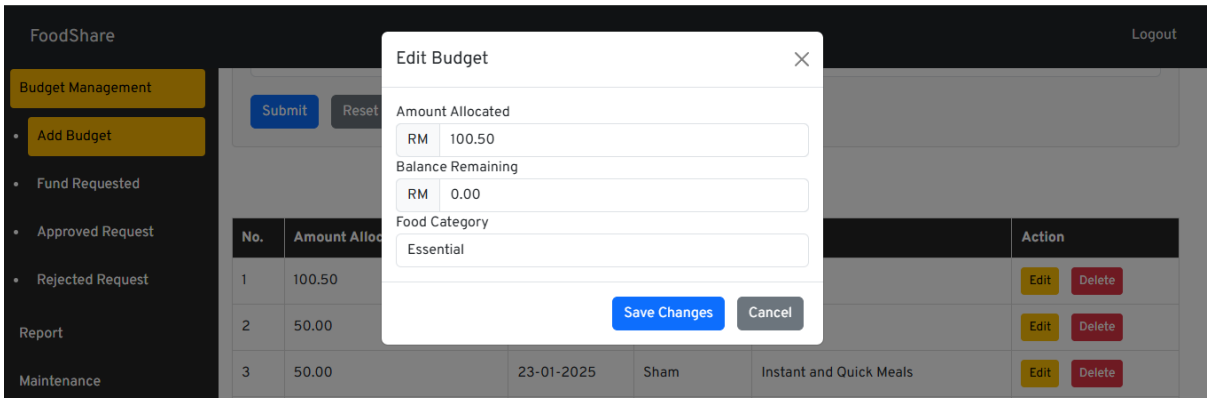



*Figure 5.5.3. 6 Edit Budget Form*

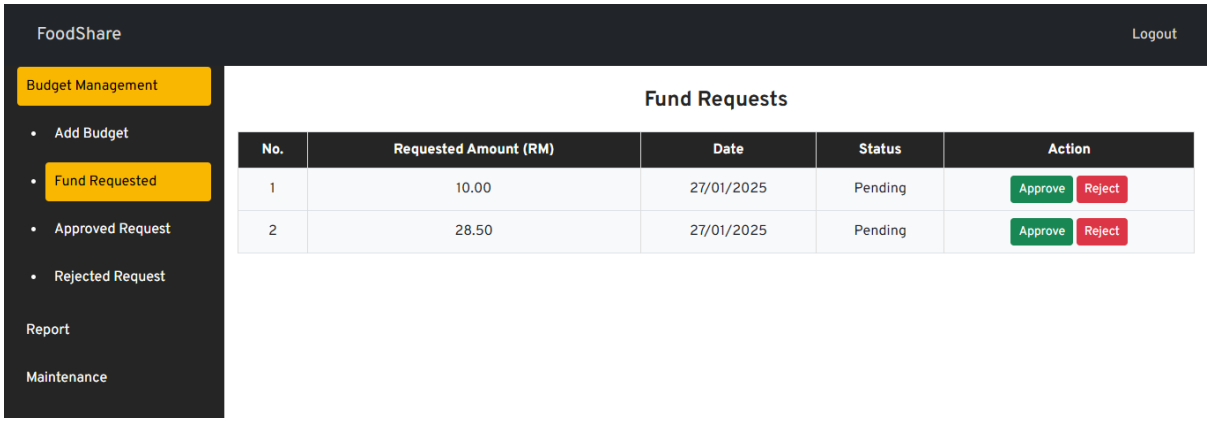



*Figure 5.5.3. 7 Fund Requests Page*

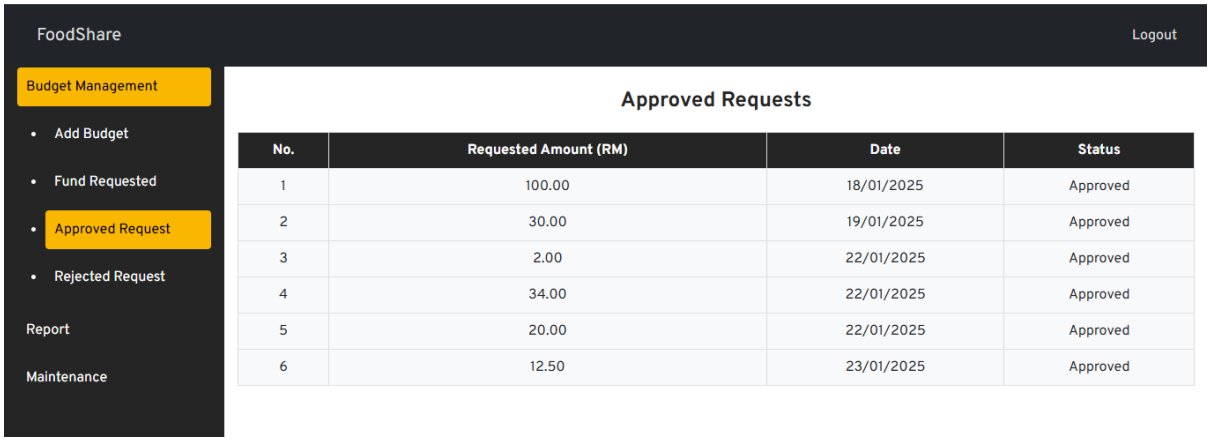



*Figure 5.5.3. 8 Approved Requests Page*

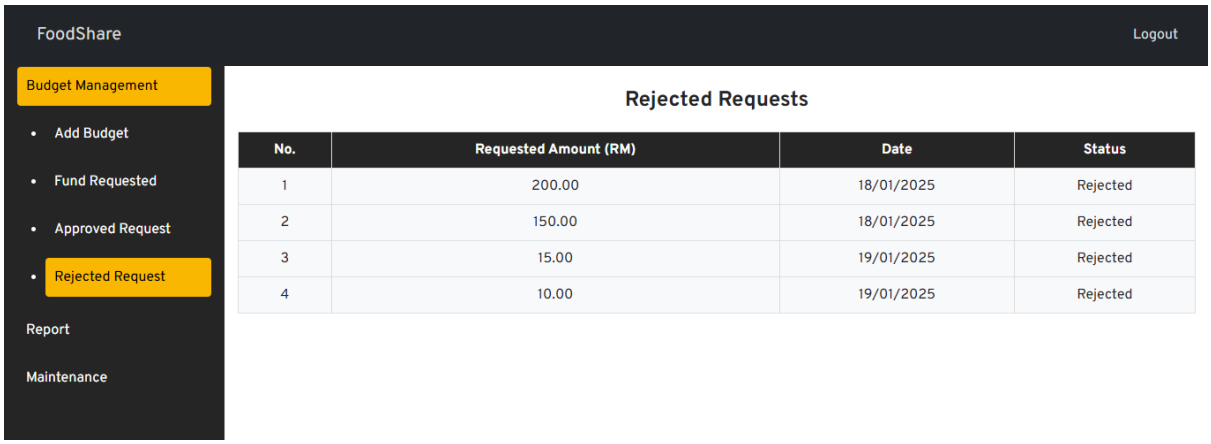



*Figure 5.5.3. 9 Rejected Requests Page*

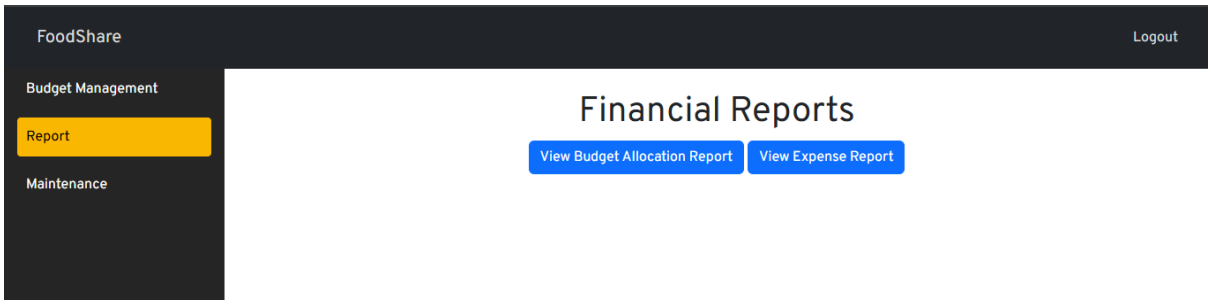



*Figure 5.5.3. 10 Financial Reports Page*

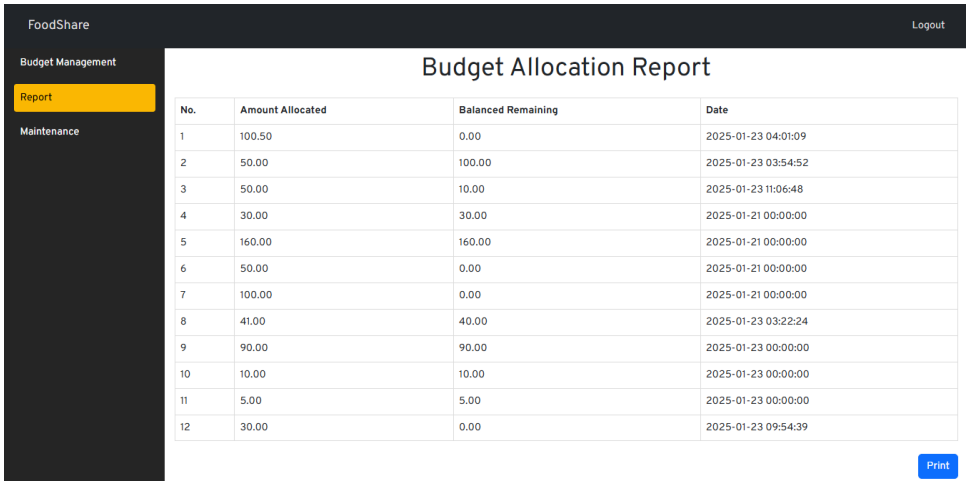



*Figure 5.5.3. 11 Budget Allocation Report*

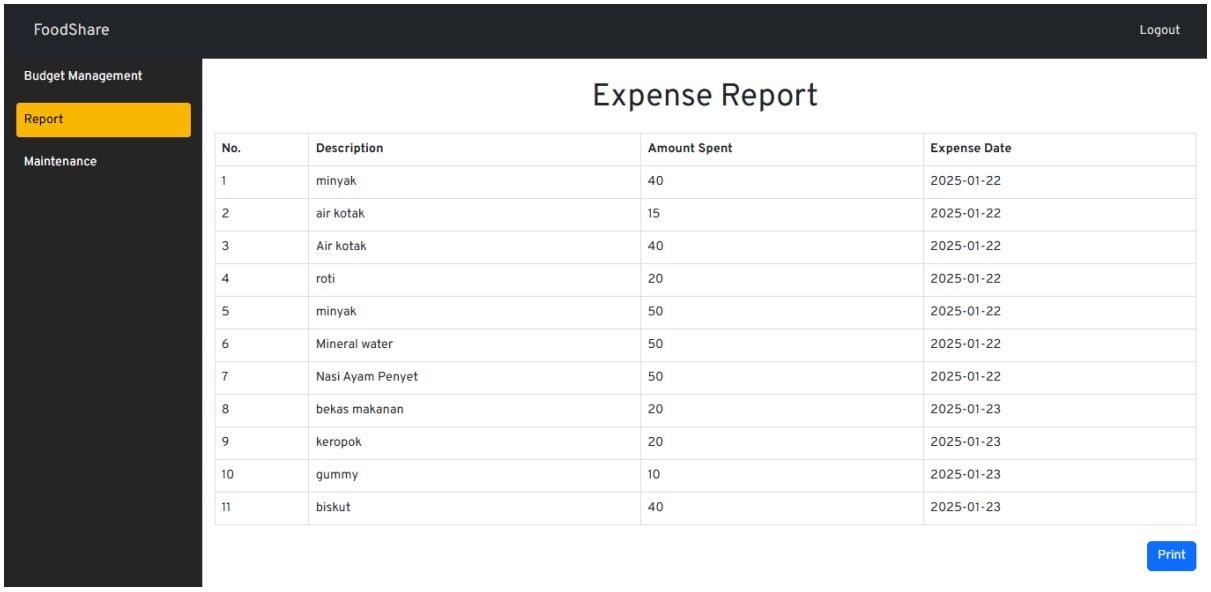



*Figure 5.5.3. 12 Expense Report*

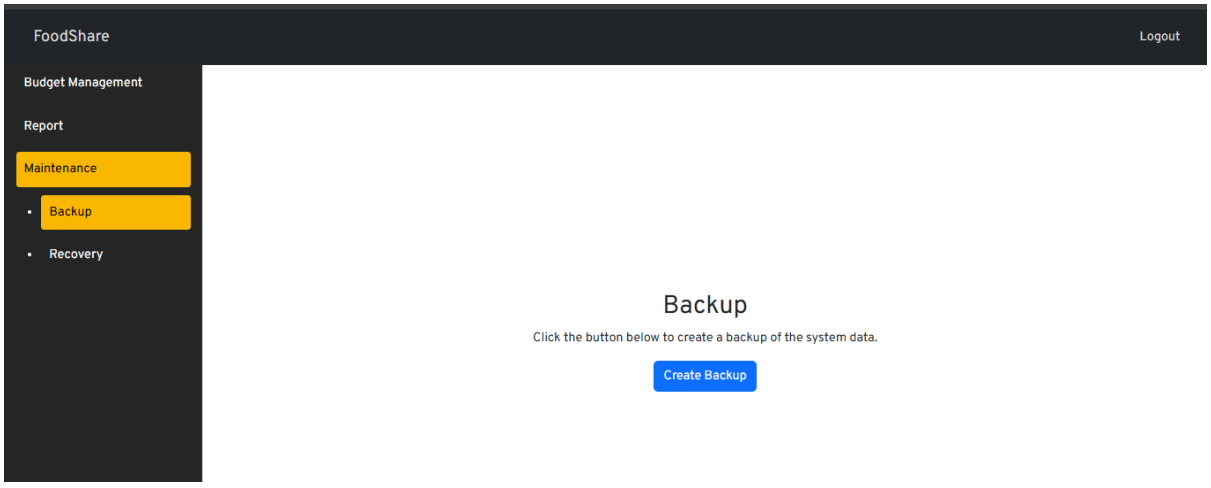



*Figure 5.5.3. 13 Backup Page*

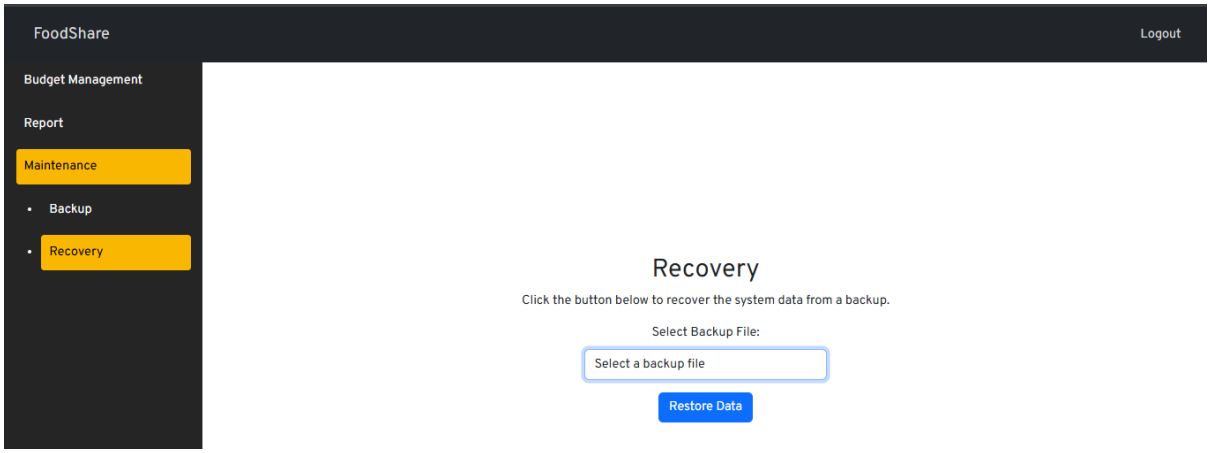



*Figure 5.5.3. 14 Recovery Page*

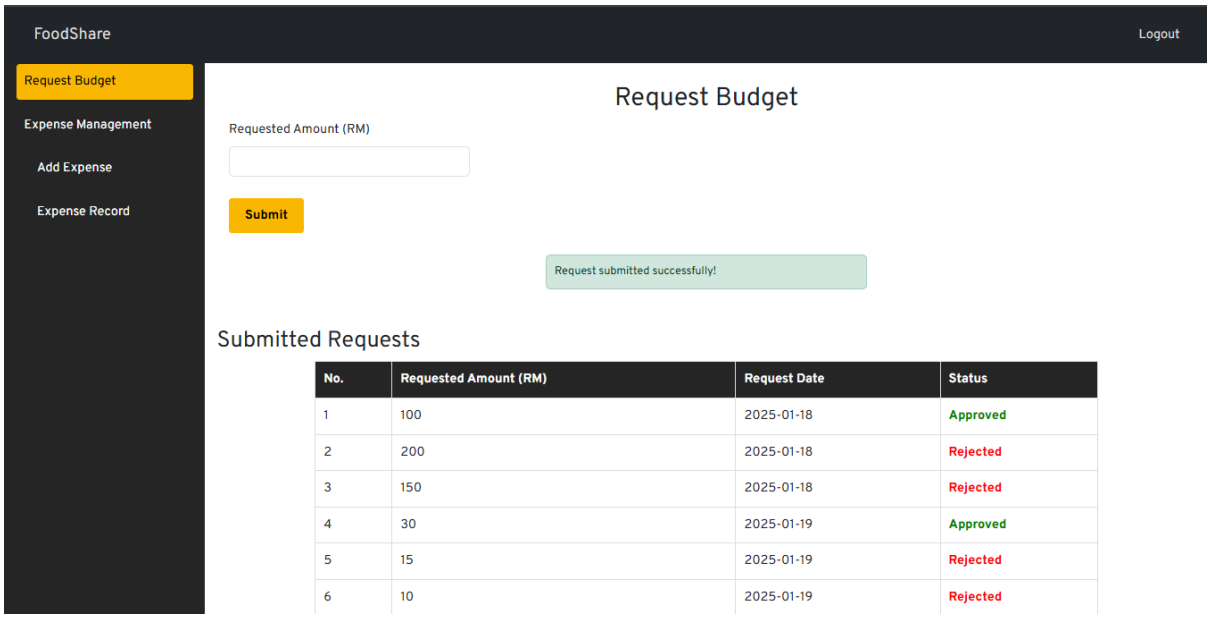



*Figure 5.5.3. 15 Request Budget and Submitted Requests Page*

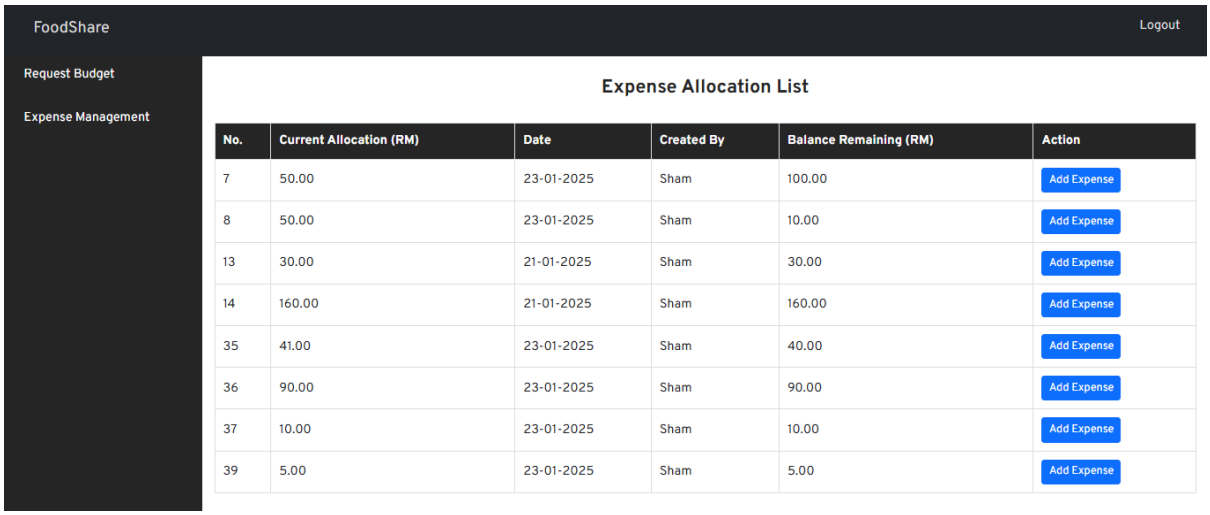


*Figure 5.5.3. 16 Expense Allocation Lists Page*

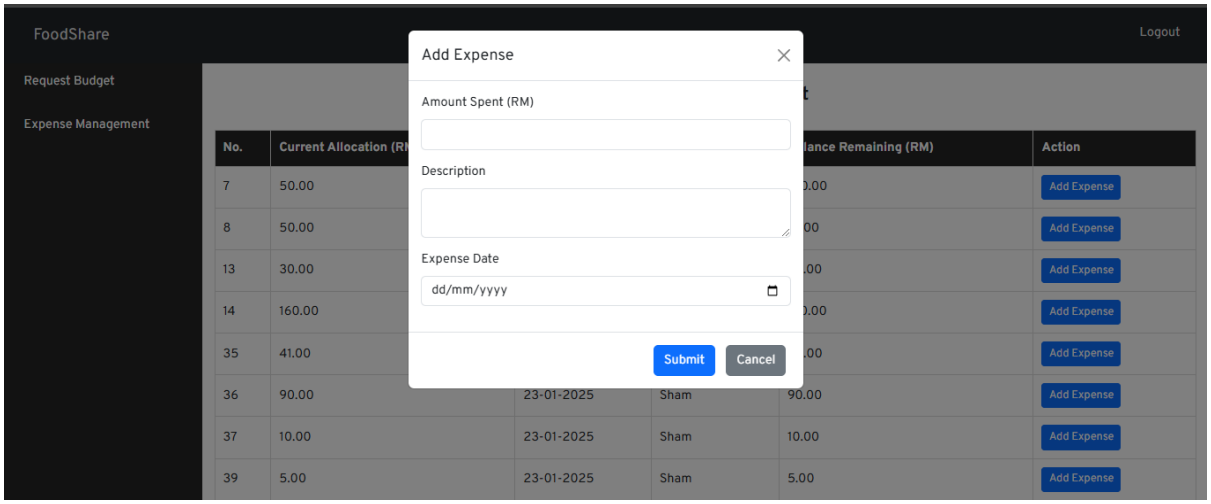



*Figure 5.5.3. 17 Add Expense Form*

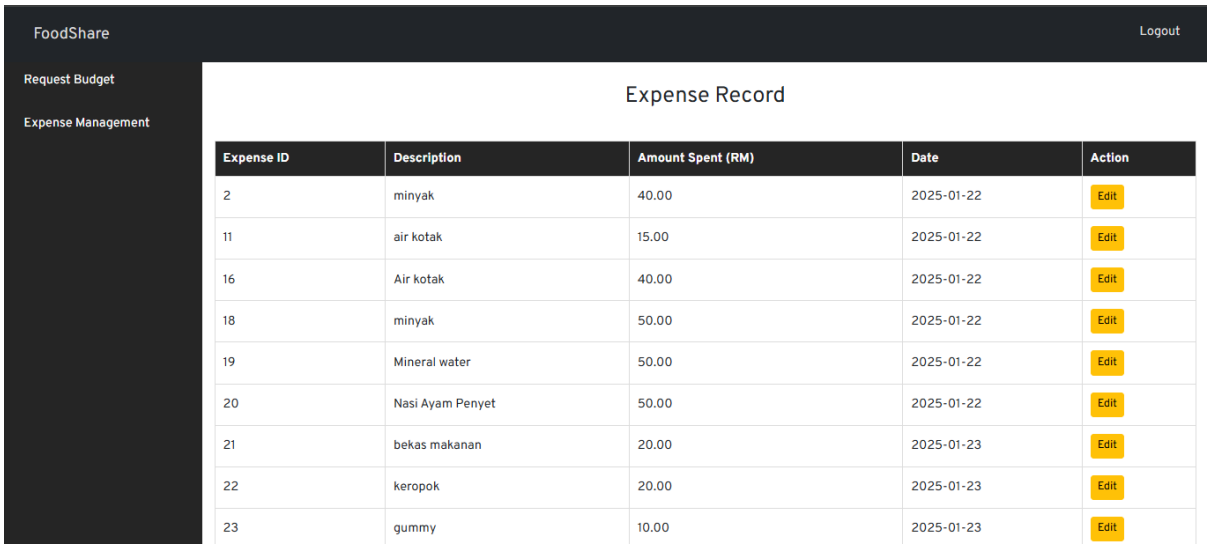



*Figure 5.5.3. 18 Expense Record Page*

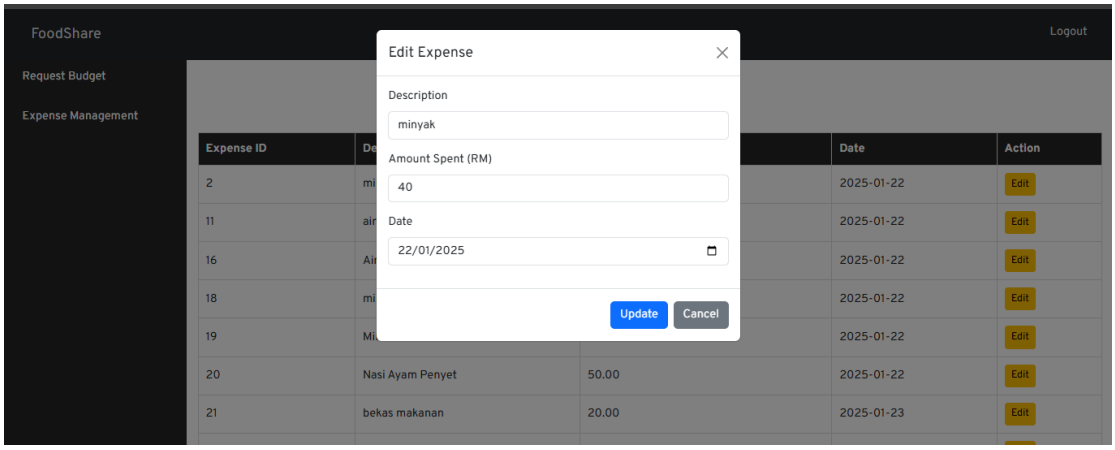



*Figure 5.5.3. 19 Edit Expense Form*

## 5.6 Conclusion

In conclusion, the implementation of the individual application modules within the FTMK FoodShare System has significantly enhanced the overall functionality and efficiency of the food bank operations. Each module Volunteer, Sponsorship, Inventory, and Reporting has been meticulously designed to address specific needs and challenges faced by the food bank. The Volunteer Module streamlines the registration and management of volunteers, ensuring effective coordination and engagement. The Sponsorship Module facilitates the management of sponsor contributions, allowing for transparent tracking and appreciation of donations. The Inventory Module ensures accurate tracking of food supplies, minimizing waste and ensuring that resources are readily available for distribution. Additionally, the Reporting Module utilizes data visualization tools to provide insights into food availability, volunteer participation, and financial contributions, aiding in informed decision-making. Overall, the integration of these modules not only improves operational efficiency but also fosters a collaborative environment that supports the mission of alleviating food insecurity among students. The successful implementation of these modules lays a strong foundation for the FTMK FoodShare System, ensuring its sustainability and effectiveness in the long term.

# CHAPTER 6: DATABASE INTEGRATION AND TESTING

## 6.1 Introduction

Database integration and testing are critical steps to ensuring that a database system operates efficiently and reliably. Integration involves combining individual modules of the system to ensure that they work together seamlessly and that data flows correctly between them. It also includes linking the database with other module systems or applications by enabling smooth interactions. On the other hand, testing is to validates the functionality, security, and performance of the database, identifying and resolving any issues before deployment. This chapter focuses on the techniques used for both integration and testing, aiming to ensure that the database meets the required specifications and functions as intended in real-world conditions.

## 6.2 Database Integration and Testing

This section covers the process of integrating the system with the database and ensuring its functionality through testing. Below are the steps involved:

1. Open web browser and navigate to https://www.zerotier.com/download/. Then, click on download menu.
2. Choose operating system to download zero tier, navigate to windows. Then click MSI Installer (x86/x64) to download the installer

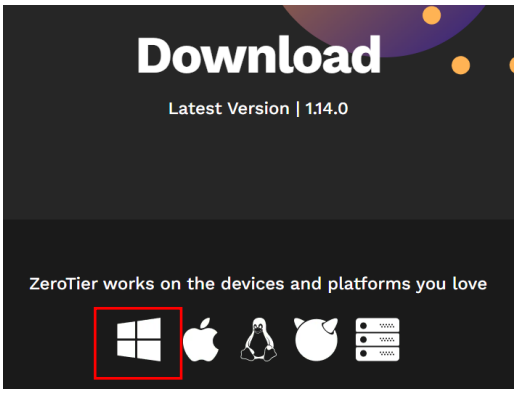



*Figure 6.3. 1 Choose operating System*

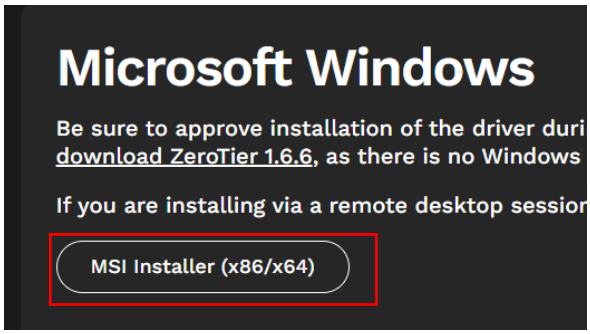



*Figure 6.3. 2 MSI Installer*

3. For Linux by using the command prompt

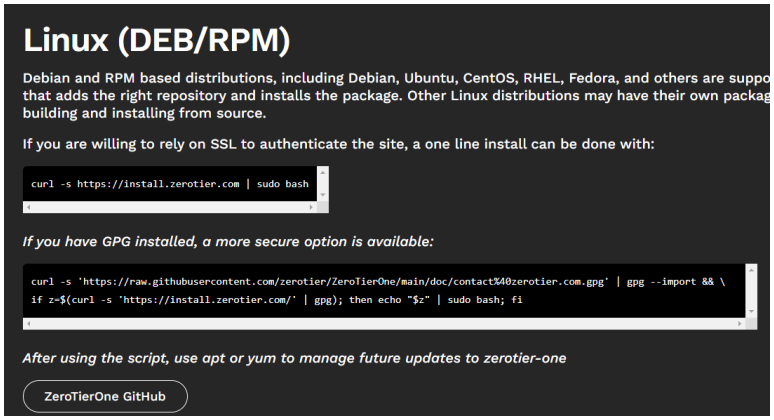



*Figure 6.3. 3 Zero tier Linux installation*

4. After make installation for zero tier, need to login to zero tier to Create A Network and click on Group 4 as the name of network that has been edited.


*Figure 6.3. 4 Create A Network*

5. After navigate to Group 4 Network, copy the Network ID

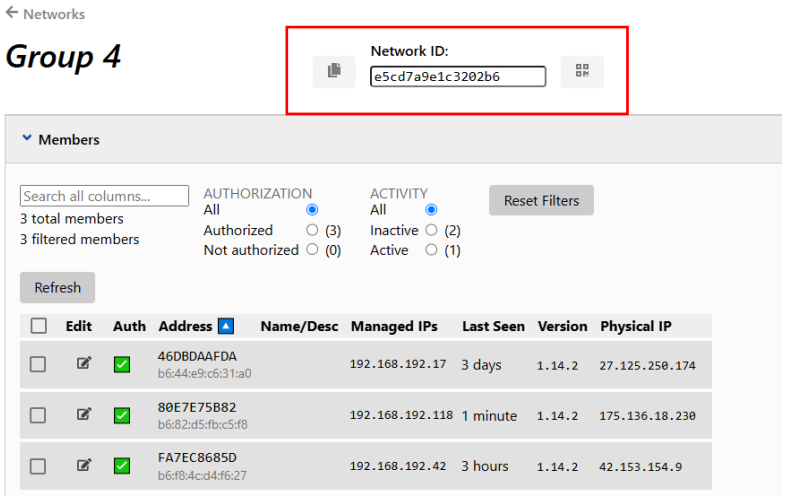



*Figure 6.3. 5 Group 4 Network (Network ID)*

6. Launch the zero tier in desktop, then click on Join New Network and paste the Network ID then click Join.

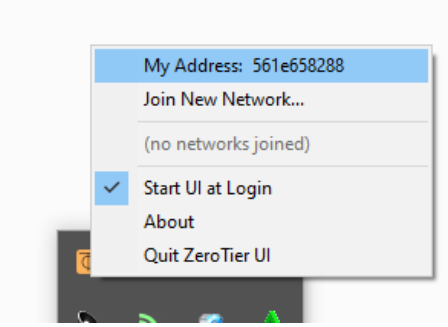



*Figure 6.3. 6 Join New Address*


*Figure 6.3. 7 Network ID to join*

7. Check the checkbox to authorize Address

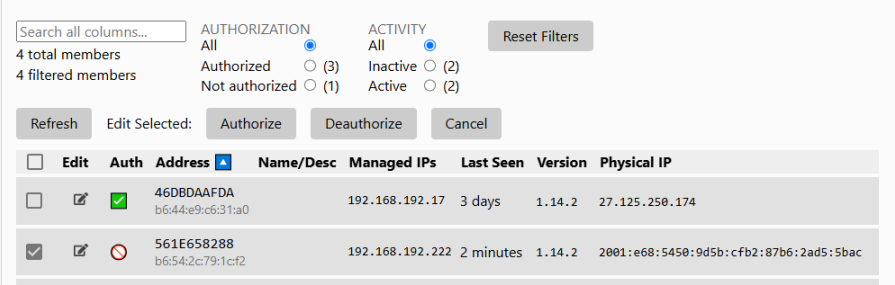



*Figure 6.3. 8 Check Checkbox on IpAddress*

8. Now, 192.168.192.222 IpAddress already authorize

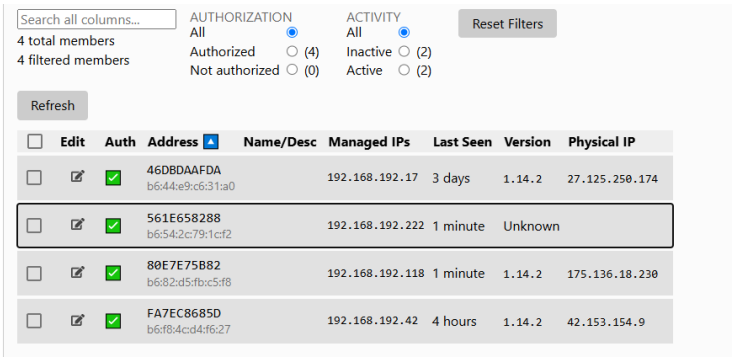



*Figure 6.3. 9 IpAddress Authorized*

9. Press `Win + R`, type `cmd`, and press `Enter` to check the zero tier ipAddress type ipconfing

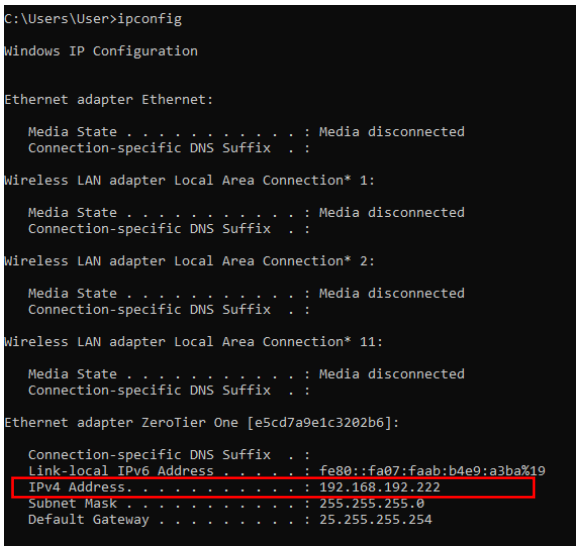



*Figure 6.3. 10 Zero Tier IpAddress*

10. Use the Ping Command to ping each

```
C:\Users\User>ping 192.168.192.118

Pinging 192.168.192.118 with 32 bytes of data:
Reply from 192.168.192.118: bytes=32 time=78ms TTL=128
Reply from 192.168.192.118: bytes=32 time=176ms TTL=128
Reply from 192.168.192.118: bytes=32 time=500ms TTL=128
Reply from 192.168.192.118: bytes=32 time=293ms TTL=128

Ping statistics for 192.168.192.118:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 78ms, Maximum = 500ms, Average = 261ms
```

*Figure 6.3. 11 Ping IpAddress*

11.For windows to disable Telnet, Press Win + R, type control, and press Enter.

i. Go to Programs and Features. In the Control Panel, click on Programs. Click Turn Windows features on or off.


*Figure 6.3. 12 Control Panel*

ii. Enable Telnet. Scroll down to Telnet Client. Check both options. Click OK and wait for the system to apply the changes.

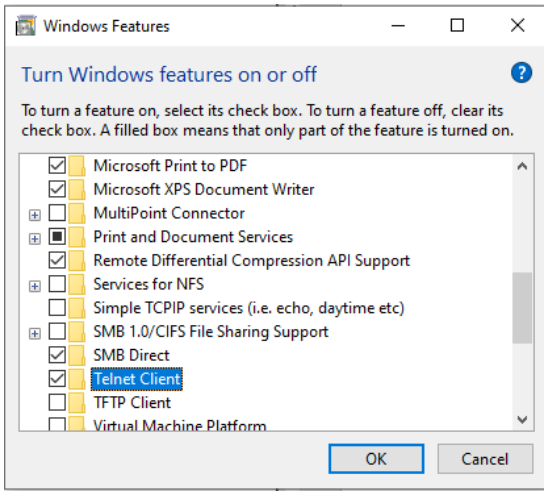



*Figure 6.3. 13 Program and Features*

iii. Before telnet close all firewall then, open cmd type ‘telnet’ and Enter


*Figure 6.3. 14 Telnet command*

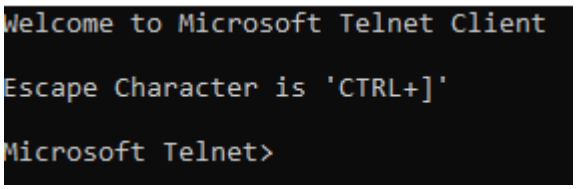



*Figure 6.3. 15 Output telnet*

12. To perform database connection between Windows (PostgreSQL) and Windows (MySQL).
    i. PostgreSQL need to modify the file pg_hba.conf on path C:\Program Files\PostgreSQL\17\data\pg_hba.conf.
    ii. PostgreSQL need to allow all IPAddress to get access the database

```
# Allow connections from a specific IP address
host    all    all    192.168.63.112/32    md5
host    all    all    192.168.192.17/32    md5
host    all    all    192.168.192.42/32    md5
host    all    all    192.168.192.118/32    md5
host    all    all    0.0.0.0/0    md5
host    all    all    192.168.233.160/32    md5
```

*Figure 6.3. 16 pg_hba.conf file*

iii. After allow connectcion save the file.

iv. Press Win + R on your keyboard, type services.msc then click OK. Find posgresql x64-17 then restart the service

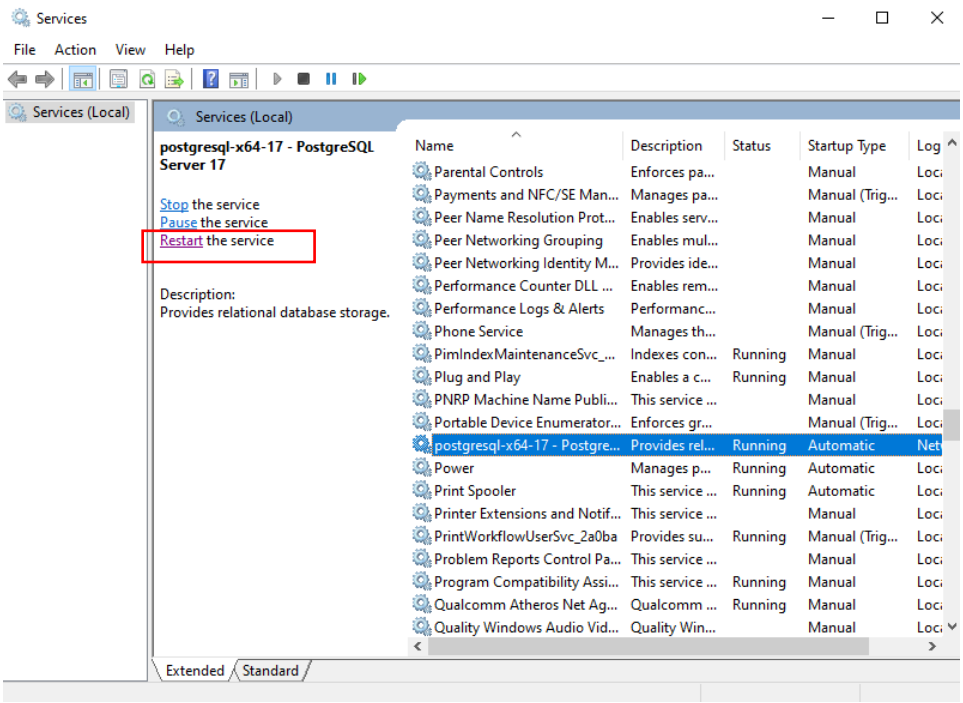



*Figure 6.3. 7 Service.msc*

13. On Windows (MySQL), to allow other databases connection to MySQL, configure the files from XAMPP.

    i. Navigate to XAMPP installation directory, open the ‘mysql’ folder, then the ‘bin’ folder.

    ii. To allow MySQL to accept connections from any IP address, find and open ‘my.ini’ file in a text editor. Search for ‘bind-address’ and change it to:

        bind-address = 0.0.0.0

    iii. To enable PostgreSQL connection, find the ‘php.ini’ file and open in text editor. Look for these lines and uncomment it:

        extension=php_pgsql.dll

        extension=php_pdo_pgsql.dll

    iv. Restart XAMPP.

14. Use ping and telnet to test the connection to other databases.

    i. Close all the enabled firewall settings.

    ii. The ‘ping’ command sends ICMP Echo Request packets to the server to check if it is reachable. Thus, run the command on command prompt:

        ping <host_ip_address>

    ‘host_ip_address’ is the other server/device’s ip address. If receive replies, it indicates that the server is reachable.

    iii. The ‘telnet’ command attempts to connect to the specified IP address and port to verify if the other database port is open and accepting connections. To test the database connection, run these commands on command prompt:

        telnet <postgresql_host_ip> 5432

        telnet <mysql/mariadb_host_ip> 3306

    If the connection is successful, you will see a message indicating that you are connected. If it fails, you may receive a "Connection refused" message, indicating that the port is not open or the PostgreSQL service is not running.

15. Use ping and telnet to test the connection to other databases **(Linux)**.

    i. Firstly, make sure the Linux virtual machine network is using **Bridge Network** to allow the network assign different IP address to guest (VM) instead of same as host (Laptop).

    ii. Disable the firewall for easier connections. Use the command **“sudo ufw disable”.**

    iii. Try ping the devices within ZeroTier VPN to check if they can receive and send packages to the Linux machine. The command would be “ping -c 5 <ip address>”.

    iv. Next, use telnet to do another test for checking the port. The command is “telnet <ip address> 3306” or “telnet <ip address> 5432”. Both for MySQL/MariaDB and PostgresSQL databases respectively.

    v. If connections say they refused then, the connections failed. If it says

otherwise that mean the connections are successful and ready for integration.

vi. Finally, to allow the Linux machine to accept connection from other devices is to change the **bind-address** in the MySQL config file. The path to the file is **/etc/mysql/my.cnf.** The config commands to put is as below:

```
[mysqld]
#port = 3306
bind-address = 0.0.0.0
```

Then, other devices in the VPN network should be able to connect to the Linux machine.

16. Now download Dbeaver as a bridge between database.

i. Open the web browser, go to the official Dbeaver website https://dbeaver.io/download/

ii. Choose DBeaver Community 24.3.3

iii. Navigate to Windows Installer to download

iv. After successful dowload, install the Dbeaver in desktop

v. Now launch Dbeaver application

vi. To add new Database connections use shortcut by press Ctrl + Shift + N on your keyboard

iv. Then, choose database that want to connect for example postgreSQL

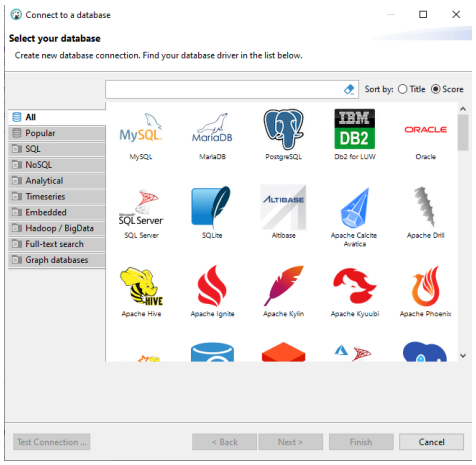



*Figure 6.3. 18 Dbeaver database selection*

17. For the Host field, you can modify it to include the IP address. Additionally, provide the database name, username, and password (if a password is set).Now, test the connection.

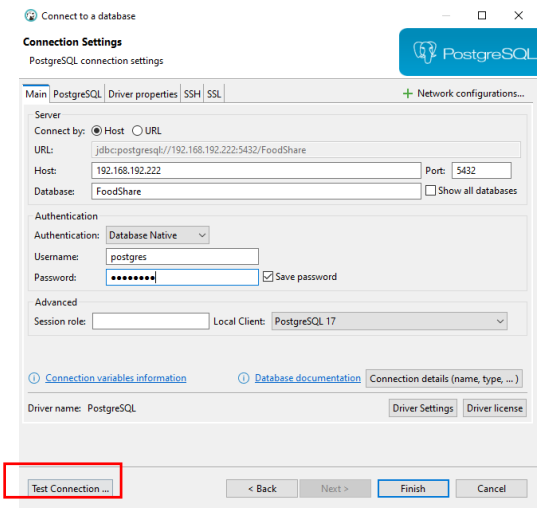



*Figure 6.3. 110 Connection Settings*

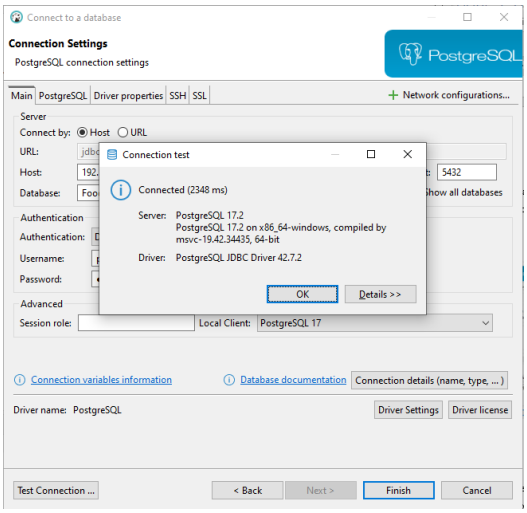


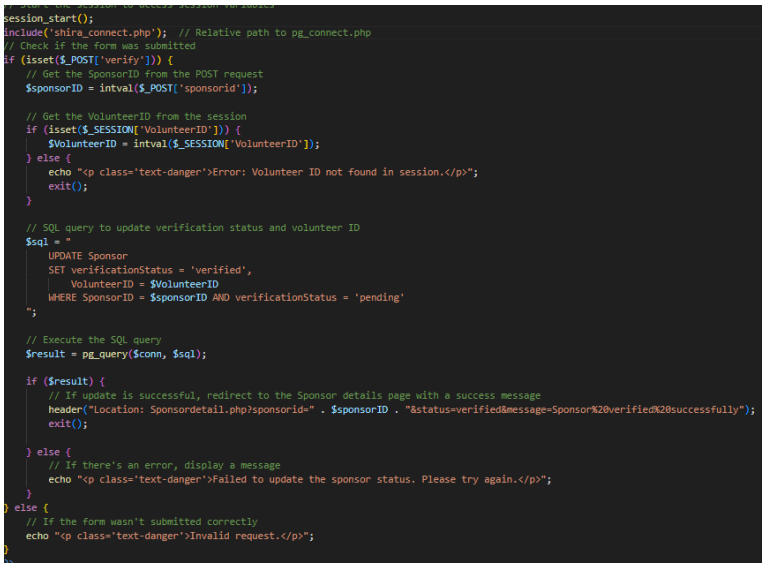

*Figure 6.3. 9 Database Connection Succesful*

18. Now that the databases connection are set up and accessible, connect to it from the application. Create a new php file for database connection including from other servers (e.g. DBConnection.php). Apply these code and modify it:

```php
<?php
// MySQL/MariaDB connection parameters
$mysql_host = 'your_mysql_host'; // e.g., '192.168.1.10'
$mysql_dbname = 'your_mysql_database_name';
$mysql_user = 'your_mysql_username';
$mysql_password = 'your_mysql_password';

// PostgreSQL connection parameters
$pgsql_host = 'your_postgresql_host'; // e.g., '192.168.1.10'
$pgsql_port = '5432'; // Default PostgreSQL port
$pgsql_dbname = 'your_postgresql_database_name';
$pgsql_user = 'your_postgresql_username';
$pgsql_password = 'your_postgresql_password';

// Connect to MySQL/MariaDB
$mysql_conn = new mysqli($mysql_host, $mysql_user, $mysql_password, $mysql_dbname);

// Check MySQL connection
if ($mysql_conn->connect_error) {
    die("MySQL Connection failed: " . $mysql_conn->connect_error);
} else {
    echo "Connected to MySQL/MariaDB successfully!<br>";
}

// Close MySQL connection
$mysql_conn->close();

// Connect to PostgreSQL
$pgsql_conn_string = "host=$pgsql_host port=$pgsql_port dbname=$pgsql_dbname user=$pgsql_user password=
$pgsql_conn = pg_connect($pgsql_conn_string);

// Check PostgreSQL connection
if (!$pgsql_conn) {
    die("PostgreSQL Connection failed: " . pg_last_error());
} else {
    echo "Connected to PostgreSQL successfully!<br>";
}

// Close PostgreSQL connection
pg_close($pgsql_conn);
?>
```

*Figure 6.3. 11 DBConnection.php file*

Once connected, SQL commands can be applied and data can be fetch and modified from other server's database with privileges. Now, can perform standard SQL operations like SELECT, UPDATE, INSERT, and DELETE on the remote server's database. The fundamental operations remain the same, but the way to interact with the remote server can differ slightly, especially when working with databases across different systems or networks.



```php
session_start();
include('shira_connect.php');  // Relative path to pg_connect.php
// Check if the form was submitted
if (isset($_POST['verify'])) {
    // Get the SponsorID from the POST request
    $sponsorID = intval($_POST['sponsorid']);

    // Get the VolunteerID from the session
    if (isset($_SESSION['VolunteerID'])) {
        $VolunteerID = intval($_SESSION['VolunteerID']);
    } else {
        echo "<p class='text-danger'>Error: Volunteer ID not found in session.</p>";
        exit();
    }

    // SQL query to update verification status and volunteer ID
    $sql = "
        UPDATE Sponsor
        SET verificationStatus = 'verified',
            VolunteerID = $VolunteerID
        WHERE SponsorID = $sponsorID AND verificationStatus = 'pending'
    ";

    // Execute the SQL query
    $result = pg_query($conn, $sql);

    if ($result) {
        // If update is successful, redirect to the Sponsor details page with a success message
        header("Location: Sponsordetail.php?sponsorid=" . $sponsorID . "&status=verified&message=Sponsor%20verified%20successfully");
        exit();

    } else {
        // If there's an error, display a message
        echo "<p class='text-danger'>Failed to update the sponsor status. Please try again.</p>";
    }
} else {
    // If the form wasn't submitted correctly
    echo "<p class='text-danger'>Invalid request.</p>";
}
```

*Figure 6.3.1 12 update verificationStatus by using FK in volunteerID (in Windows MySQL)*

## 6.3 Database Management and Administration

### 6.3.1 Backup and Recovery for MySQL/MariaDB on Windows

#### 1. Backup

1. A dedicated folder is created to store the backup files securely. The folder's directory path is noted for reference.
2. A batch file (backup.bat) is created with the following command:
   mysqldump -u [username] -p[password] [database_name] > "[backup_directory_path]\backup_%DATE%.sql"

```
:: Define the backup file name
set backupFile=%backupDir%\backup_file_!formattedDate!_!formattedTime!.sql

:: Execute the mysqldump command without password
"C:\xampp\mysql\bin\mysqldump" -u %dbUser% %dbName% > "!backupFile!"
```

*Figure 6.4.1.1.1 Backup.bat configuration*

3. For daily auto backup, open task scheduler on windows, then create a basic task (e.g. database backup).
    i. In the create task wizard, in trigger tab, choose **Daily** and the time.
    ii. In action tab, select **Start a program**. Then for the **Program/script**, browse and select the previously create 'backup.bat' file.
    iii. Click **Finish**. The new task shall look like this:


4. For semi-automatic backup, a button in the application interface triggers the batch file. When clicked, it executes the backup process, generating a `.sql` file in the specified folder.

```php
// Backup functionality
if (isset($_POST['backup'])) {
    $backupScript = '"C:\\xampp\\htdocs\\MyPHPSite\\workshop 2\\autobackup\\backup.bat"';

    // Execute the batch file
    $output = [];
    $returnVar = 0;
    exec($backupScript, $output, $returnVar); // Use exec to capture output

    if ($returnVar === 0) {
        $message = "<div class='alert alert-success'>Backup successful!</div>";
    } else {
        $message = "<div class='alert alert-danger'>Backup failed. Error: " . implode("\n", $output) . "</div>";
    }
}
```

*Figure 6.4.1.1.2 Interface configuration for semi-backup function*


*Figure 6.4.1.1.3 Interface for semi-backup function*

### 2. Recovery

1. The user selects a `.sql` backup file from the interface.

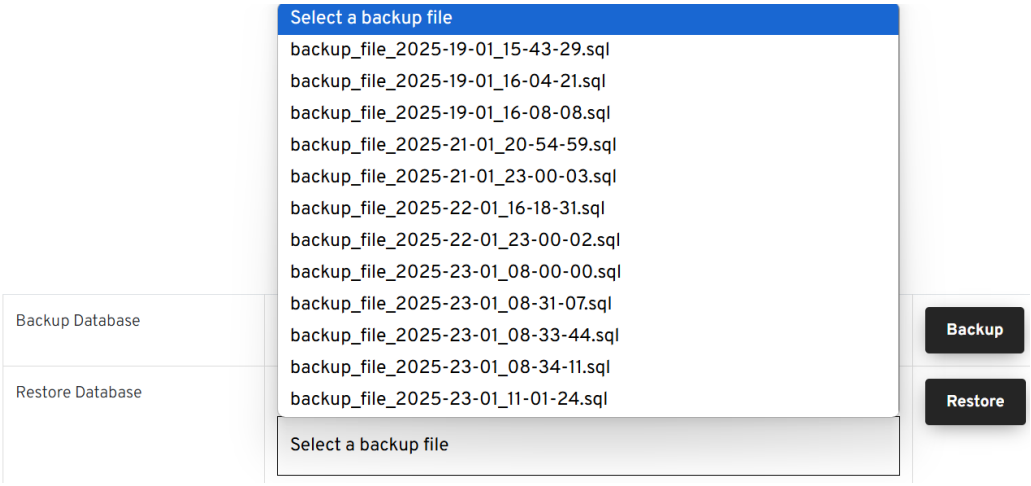



*Figure 6.4.1.2.1 Interface for semi-recover function*

2. When the restore action is triggered, the system runs the following command:
   mysql -u [username] -p[password] [database_name] < "[backup_file_path]"
   The selected file path is passed to the restore process, and the system restores the database content.

```php
// Restore functionality
if (isset($_POST['restore'])) {
    // Get the selected backup file from the dropdown
    $backupFile = $_POST['backupFile'];

    if (!empty($backupFile) && file_exists($backupFile)) {
        $command = "C:\\xampp\\mysql\\bin\\mysql -u $dbUser $dbName < \"$backupFile\"";

        // Execute the command
        $output = [];
        $returnVar = 0;
        exec($command, $output, $returnVar); // Use exec to capture output

        if ($returnVar === 0) {
            $message = "<div class='alert alert-success'>Restore successful!</div>";
        } else {
            $message = "<div class='alert alert-danger'>Restore failed. Error: " . implode("\n", $output) . "</div>";
        }
    } else {
        $message = "<div class='alert alert-danger'>Please select a valid backup file.</div>";
    }
}
```

*Figure 6.4.1.2.2 Interface configuration for semi-recover function*

### 6.3.2 Backup and Restore for PostgreSQL on Windows

#### 1. Backup

The backup system for PostgreSQL on Windows uses a semi-automatic process, allowing users to create backups directly through the interface provided. Below are the detailed steps for performing a backup:

1. On the "Backup and Restore System" page, click the "Create Backup" button.

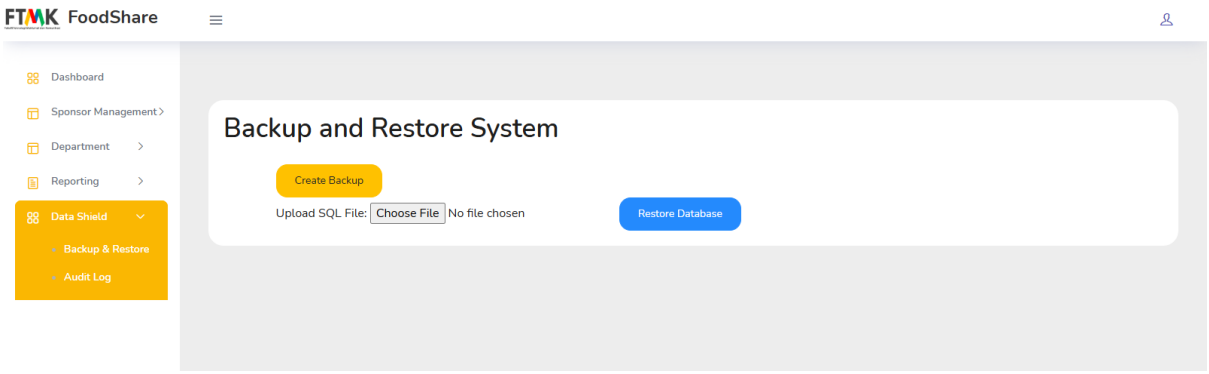



*Figure 6.4.2.1 1 Backup and restore Page*

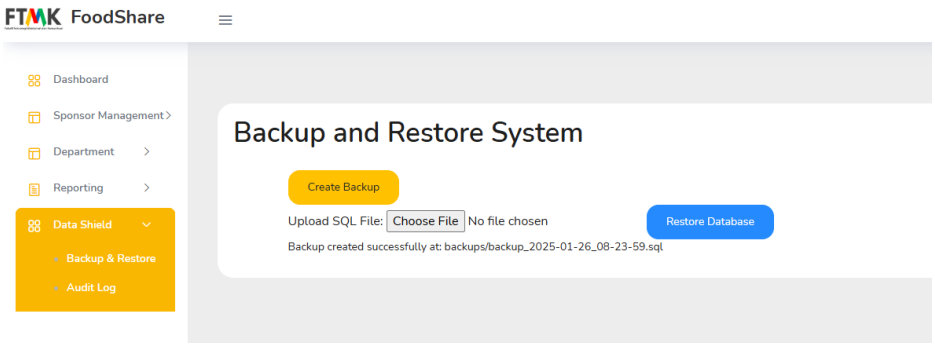



*Figure 6.4.2.1 2 Backup created*

2. When the "Create Backup" button is clicked, the system initiates a PHP script that performs actions checks and Creates Backup Directory:
    i. The script checks if the directory specified (backups/) exists
    ii. If not, it creates the directory with appropriate permissions (0777).
3. The system generates backup file name
    i. The system dynamically generates a unique file name using the

current date and time (e.g., backup_YYYY-MM-DD_HH-MM-SS.sql).

4. Execute pg_dump Command
    i. The script executes the pg_dump command using the database credentials provided in pg_connect.php.
    ii. Environment variables are set for the database password using putenv("PGPASSWORD=$password");.
5. The output of the pg_dump command is saved in the specified directory as a .sql file.

```php
<?php
// Include database credentials
include('../sponsorship/pg_connect.php');  // Relative path to pg_connect.php

$backupDir = 'backups/'; // Directory to save backups
$backupFile = $backupDir . 'backup_' . date('Y-m-d_H-i-s') . '.sql'; // Backup file name

$message = ''; // Message for status updates

// Backup database
if (isset($_POST['backup'])) {
    try {
        // Create backups directory if not exists
        if (!is_dir($backupDir)) {
            mkdir($backupDir, 0777, true);
        }

        // Command to export the database using pg_dump
        $command = "pg_dump --host=$host --username=$user --no-password --file=$backupFile $dbname";

        // Execute the command
        putenv("PGPASSWORD=$password"); // Set the password environment variable
        exec($command, $output, $returnVar);

        if ($returnVar === 0) {
            $message = "Backup created successfully at: $backupFile";
        } else {
            $message = "Error creating backup. Please check permissions or credentials.";
        }
    } catch (Exception $e) {
        $message = "An error occurred during backup: " . $e->getMessage();
    }
}

```

*Figure 6.4.2.1 3 PHP code Backup Database*

6. Successful backup file will save in this path
   C:\xampp\htdocs\FTMK_FoodShare\AdSponsor_Inventory\backups

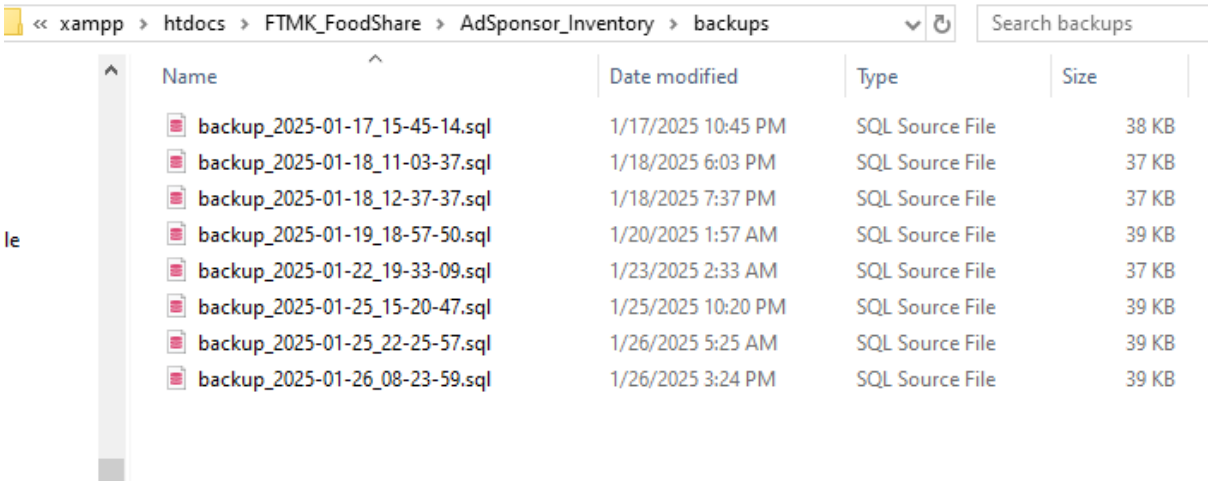



*Figure 6.4.2.1 4 File Path after created backup*

### 2. Recovery

The restore process for PostgreSQL on Windows allows users to upload an SQL file to restore the database. This process is semi-automatic and uses a PHP script to execute PostgreSQL's psql command.

1. Navigate to the "Backup & Restore" section in the "Data Shield" module of the FTMK FoodShare system.
2. Click the "Choose File" button to select the .sql file you want to use for restoration.

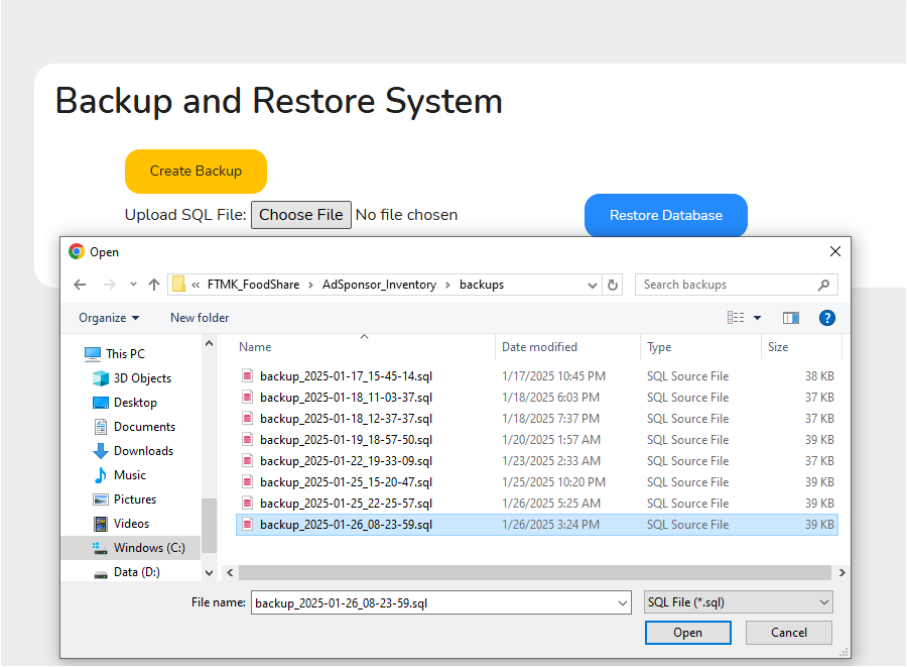



*Figure 6.4.2.2 1 Upload SQL file*

3. Click on the "Restore Database" button. This action initiates the PHP script to process the uploaded file and execute the restoration.
4. The system performs the Validate the Uploaded File and Checks if the uploaded file exists and is not empty.

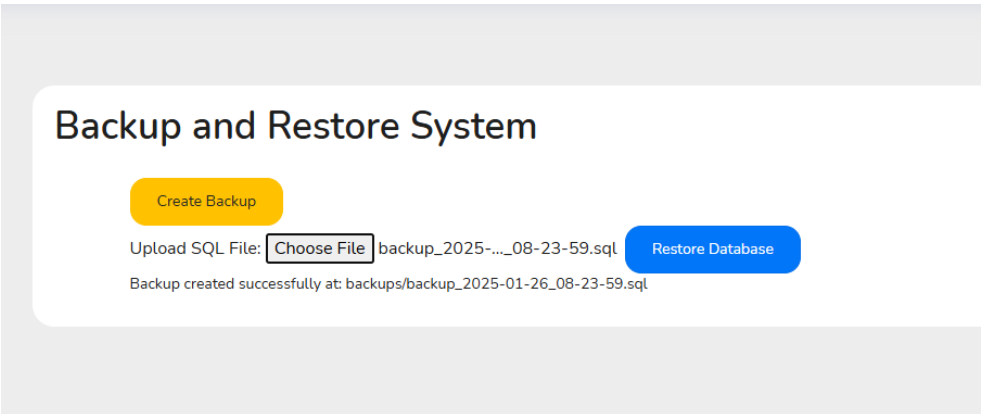



*Figure 6.4.2.2 2 After Upload SQL file*

5. The system checks the return status of the `psql` execution
    i. **Success**: Displays a success message (e.g., "Database restored

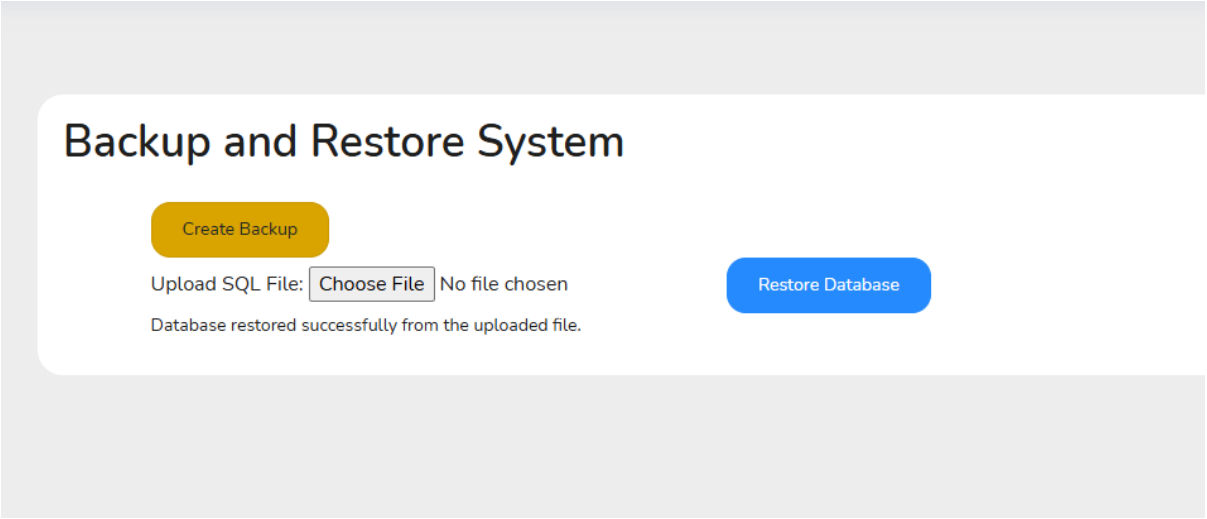
successfully from the uploaded file").

ii. Failure: Displays an error message indicating issues with the file or process (e.g., "Error restoring database. Please check the file and try again.").

```php
// Restore database
if (isset($_POST['restore'])) {
    $uploadedFile = $_FILES['restoreFile']['tmp_name'];

    if (!empty($uploadedFile)) {
        try {
            // Command to restore the database using psql
            $command = "psql --host=$host --username=$user --no-password --dbname=$dbname --file=$uploadedFile";

            // Execute the command
            putenv("PGPASSWORD=$password"); // Set the password environment variable
            exec($command, $output, $returnVar);

            if ($returnVar === 0) {
                $message = "Database restored successfully from the uploaded file.";
            } else {
                $message = "Error restoring database. Please check the file and try again.";
            }
        } catch (Exception $e) {
            $message = "An error occurred during restoration: " . $e->getMessage();
        }
    } else {
        $message = "Please upload a valid SQL file for restoration.";
    }
}
```

*Figure 6.4.2.2 3 PHP code for restore*

6. Database restored successfully.



### 6.3.3 Backup and Recovery for MySQL/MariaDB on Linux

For backup process on MySQL Linux, we will use a simple script for it and make automatic backup every day. The script is as below:

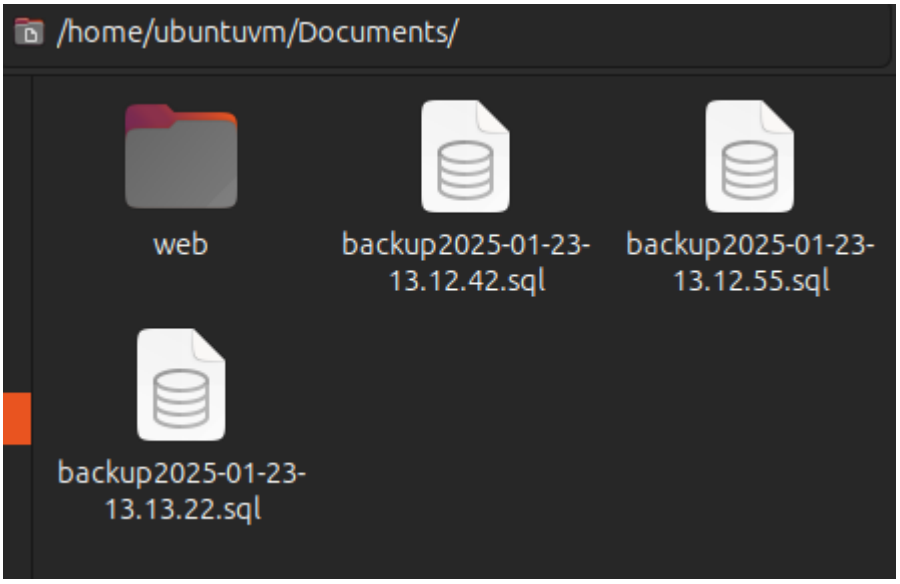
```
ubuntuvm@ubuntuvm:~/Documents$ cat mysqlbackup.sh
/usr/bin/mysqldump --routines -u ubuntuvm -p'ubuntuvm' foodshare > ${HOME}/Documents/backup$(date +%Y-%m-%d-%H.%M.%S).sql
ubuntuvm@ubuntuvm:~/Documents$
```

It uses MySQL backup utility tool **(mysqldump)** to make multiple backups with different times.

For automatic backups, we have to copy the script into crontab to make it a cronjob. Paste the script to crontab by typing **crontab –e** and it will look like this:

```
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command

00 23 * * * /usr/bin/mysqldump --routines -u ubuntuvm -p'ubuntuvm' foodshare > ${HOME}/Documents/backup$(date +%Y-%m-%d-%H.%M.%S).sql
```

Paste the script to the last line. He script will make backups of MySQL every day on 11 p.m. The backups will look like below:



For recovery, a command will be executed to restore the tables from the backup. The command is as below:

```
ubuntuvm@ubuntuvm:~/Documents$ mysql -u ubuntuvm -p fooshare < backup2025-01-23-13.13.22.sql
```

After pressing enter, the database will be restored.

### 6.3.4 Backup and Recovery for MySQL on Windows

1. Backup

i. A separate folder is established to keep the backup files secure. The folder's directory path is noted for future reference.

ii. The command to produce a batch file (backup.bat) is as follows:

mysqldump -u [username] -p[password] [database_name] > "[backup_directory_path]\backup_%DATE%.sql"

```
:: Format the backup file name with date and time
set backupFile=%backupDir%\backup_file_%year%-%month%-%day%_%hour%-%minute%-%second%.sql

:: Run the MySQL dump command
"C:\xampp\mysql\bin\mysqldump" -u %dbUser% %dbName% > %backupFile%
```

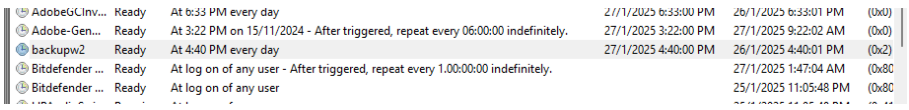
*Figure 6.4.4.1 4 Backup.bat configuration*

iii. Open the Windows task scheduler and create a basic task. For example, backupw2.

- In the create task wizard, in trigger tab, choose **Daily** and the time.

- In action tab, select **Start a program**. Then for the **Program/script**, browse and select the previously create 'backup.bat' file.

- Click **Finish**. The new task shall look like this:




iv. For semi-automatic backup, a button in the program interface initiates the batch file. When clicked, it starts the backup process and creates a.sql file in the chosen folder.

```php
// Handle backup request
if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['action']) && $_POST['action'] === 'backup') {
    $output = [];
    $returnVar = null;

    // Path to the backup.bat file
    $backupBatPath = __DIR__ . DIRECTORY_SEPARATOR . 'backup.bat';

    // Execute the batch file
    exec($backupBatPath, $output, $returnVar);

    if ($returnVar === 0) {
        $message = "Backup created successfully!";
    } else {
        $message = "Failed to create backup. Please check the configuration.";
    }
}
```

*Figure 6.4.4.1 2 Interface Configuration for Semi-Backup Function*


*Figure 6.4.4.1 3 Interface for Semi-Backup Function*

2. Recovery

i. The user chooses a.sql backup file from the interface.

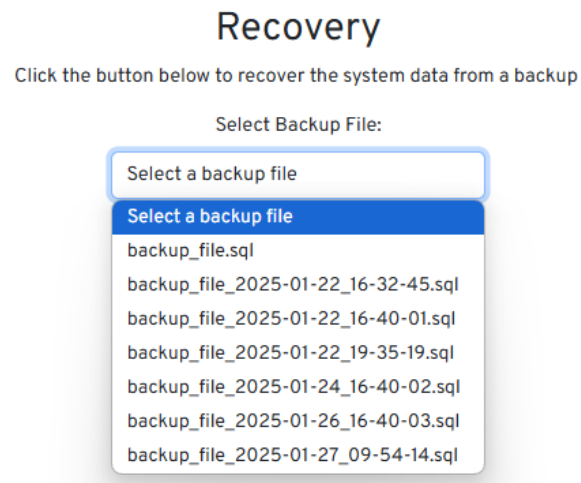



*Figure 6.4.4.2 1 Interface for Semi-Recover Function*

ii. When the restore action is triggered, the system runs the following command: mysql -u [username] -p[password] [database_name] < "[backup_file_path]". The selected file path is passed to the restore process, and the system restores the database content.

```php
$backupFiles = glob($backupDir . DIRECTORY_SEPARATOR . '*.sql'); // Get all .sql files in the directory
$message = ''; // Initialize message variable

// Restore functionality
if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['restore'])) {
    // Get the selected backup file from the dropdown
    $backupFile = $_POST['backupFile'];

    if (!empty($backupFile) && file_exists($backupFile)) {
        $command = sprintf(
            '"C:\\xampp\\mysql\\bin\\mysql" -u root module_budget < "%s"',
            $backupFile
        );

        // Execute the command
        $output = [];
        $returnVar = 0;
        exec($command, $output, $returnVar);

        if ($returnVar === 0) {
            $message = "<div class='alert alert-success'>Restore successful!</div>";
        } else {
            $message = "<div class='alert alert-danger'>Restore failed. Error: " . implode("\n", $output) . "</div>";
        }
    } else {
        $message = "<div class='alert alert-danger'>Please select a valid backup file.</div>";
    }
}
```

*Figure 6.4.4.2 2 Interface Configuration for Semi-Recover Function*

## 6.4 Conclusion

In conclusion, the successful integration and testing of the database system are critical to ensuring seamless data management and operational efficiency within the modules.Connecting multiple parts and making sure data moves easily between modules are all part of database integration, which ensures that every function work properly with the database. The accuracy, performance, and dependability of the database are further established by proper testing in a variety of situations, such as difficult cases and high-load situations.

Additionaly, strong backup and recovery procedures are also essential for preserving data availability and integrity in the situation of unforeseen failures or data loss. While recovery methods reduce downtime and shield the system from possible risks like hardware faults, cyber threats, or user mistake, regular backups ensure that important data is safely saved and readily restoreable.

Prioritizing database integration, thorough testing, and implementing in place dependable backup and recovery plans makes the system more capable of meeting operational needs, providing consistent performance, and protecting important data. The system's long-term sustainability, scalability, and adaptability are all improved by these initiatives taken together.

# CHAPTER 7: CONCLUSION

## 7.1 Introduction

The conclusion will summarize the outcomes of the FTMK FoodShare System development, reflecting on its achievements, challenges, and contributions to addressing food insecurity within the campus community. This section highlights how the system aligns with the goals of Sustainable Development Goal (SDG) 2: Zero Hunger and discusses its broader impact on food redistribution, transparency, and community engagement. It also examines the system's limitations, offering suggestions for future improvements and potential avenues for commercialization.

## 7.2 Achievement and Application in SDG

The FTMK FoodShare system directly contributes to SDG 2: Zero Hunger, targeted to ending hunger, achieving food security, and improving nutrition. It tries to tackle the problem of food insecurity by offering a platform where FTMK students, lecturers, and the surrounding community can share excess food with the needy. By creating an ongoing system for food redistribution within the campus, it will directly impact hunger and help people who cannot catch up with their nutrition by making sure no one is starving because of lack of ability to eat.

The application of SDG 2: Zero Hunger in the FTMK FoodShare System is reflected in several key aspects that contribute to food security and sustainability within the university community First and foremost, the system targets a major problem on campuses where food waste may be substantial by allowing students, lecturers, and others to share excess food rather than throwing it away. This helps to address a serious issue with campus food, where a lot of food might go to waste. For student who are having trouble affording a regular meal or finding one, this food redistribution is a suitable solution.

The system's goal is to prevent those who might have demanding job or educational commitments from going without nutrition because they are unable afford healthy meals. Additionally, the FTMK FoodShare System helps students overcome food insecurity by creating a strong network of organizations, sponsors, and volunteers. This ensures that

everyone has access to food regardless of their financial situation and contributes to the development of an environment of giving and receiving without presumption. In the end, FTMK FoodShare helps humanity move toward sustainable practices for improved nutrition for everyone by achieving SDG 2: food security and zero hunger.

## 7.3 Project Limitation

This project is tailored for managing foodbank activities within FTMK, UTeM, limiting its scalability to larger organizations or other faculties without further customization. It relies on accurate user data entry, making it susceptible to errors that could affect operations and reports. The system also lacks integration with external tools and depends on stable internet connectivity for key functions like donation tracking and database backups, which could disrupt operations during network issues. Additionally, its reporting features are limited to standard templates, and budget and time constraints restrict the inclusion of advanced functionalities, leaving room for future improvements.

## 7.4 Suggestion and Improvement

FTMK FoodShare System can be improved by extending its scalability to other faculties or organisations via modular customisation. Integrating additional technologies like APIs for real-time tracking and payment gateways would improve functionality. Enhanced reporting with customisable templates and graphical dashboards can increase data insights, while offline capabilities guarantees continuous use even during internet failures. To eliminate errors, data validation procedures should be deployed in conjunction with a revised, user-friendly interface that includes accessibility elements. Two-factor authentication and encryption can improve security, while having a mobile app increases accessibility. Gamification features can also increase volunteer and sponsor involvement, while optimisation tools such as route planning and expiration notifications can help reduce food waste even further.

## 7.5 Potential Commercialization

The FTMK FoodShare System holds significant potential for commercialization through various channels. By expanding to other departments, partnering with NGOs and charitable organizations, the platform can address food insecurity on a larger scale. Corporate sponsorships, mobile app development, and data analytics services also provide revenue opportunities. Ultimately, the system's adaptability and alignment with sustainability goals position it for successful commercialization.

## 7.6 Conclusion

In conclusion, the FTMK FoodShare System has proven to be an innovative and impactful solution to address food insecurity within the FTMK community, aligning with the goals of Sustainable Development Goal (SDG) 2: Zero Hunger. Through its focus on food redistribution, sustainability, and community engagement, the system has the potential to significantly reduce food waste and ensure that those in need have access to nutritious meals. While there are some limitations, the system offers a strong foundation for future improvements and growth. With opportunities for commercialization and broader application, the FTMK FoodShare System could expand its positive impact, benefiting the university the community.