Software Design Document

for

DEVELOPMENT OF PROMAT-D PROJECT MANAGEMENT TRACKING SYSTEM FOR EC NEWENERGIE SOLAR COMPANY WITH DASHBOARD

Version 1.0

Prepared by

MUHAMMAD AZRUL BIN ABDUL RAHIM – 2024545287

EC Newenergie

30th January 2026

Revision History

Name	Date	Reason For Changes	Version

Table of Contents

1. Introduction

1.1 Purpose

The Software Design Document (SDD) serves the purpose of presenting an in-depth description of the system's design and allowing software development by providing an understanding of the intended structure of the design and its developmental process. However, to make it even clearer and more useful, we need to add more details about a few important areas. These include a simple explanation of the different parts of the design and how they work together, a breakdown of the system's overall structure, and a better understanding of how data moves through the system and how different functions are organized.

The product specified in this document is the Katzen Event Management System with Software Design Document (SDD) revision number 1.0 Katzen formerly had some difficulties in administering their data. Plus, Katzen previously conducted the manual registration process for registering their members and volunteers which is through WhatsApp platform. The purpose of this document is to create a website for Katzen Cat Sanctuary that will be an interface for administrators, volunteers, donors, and members. The scope includes:

1. **Member Features**

- Profile Management: Allowing members to manage their profiles.
- Events List: View the events available.
- User Registration: Enabling new users to register and existing users to manage their accounts.
- Merchandise Promotion: View the merchandise promotion that is available for members.

2. **System Administrator Features**

- Admin-Related Tasks: Providing tools and interfaces for administrators to manage the community system, including user management, event management, report generation and

donation management.

3. **Volunteer Features**

- Profile Management: Allowing volunteers to manage their profiles.
- Events Details: View the events and their events schedule.
- User Registration: Enabling new users to register and existing users to manage their accounts.

4. **Donor Features**

- Donation Process: Allowing donors to donate.

1.2 Scope

The Katzen Event Management System is designed to improve interactions between volunteers, members, donors, administrators, and all Katzen communities. The objective of this system is to resolve the data management issue encountered by Katzen where the data were not kept systematically and efficiently. This system will help to record all data consistently and generate a precise report based on the data collected. It will improve the overall efficacy of data management in community-related activities. Katzen Event Management System will replace the manual registration process with a digital registration form for members and volunteers in a system that is more organized and efficient. Plus, the overall operational Katzen's efficiency can be improved with a more well-organized system. This system includes volunteer management that allows volunteers to register, manage their profiles, view events' details, and view event schedules. Member administration includes the ability to register, manage profiles, view events' details, and view merchandise promotions. It also contains donation management that facilitates donors to make donations to Katzen. This allows system administrators to manage the community system, including user administration, event management, donation management, and report generation. The system will also perform other requirements that cover its performance, safety, security, availability, reliability, usability, portability, and correctness.

1.3 Document Structure

- Section 1: Introduction

  The introduction section includes information about the purpose of the document, scope, references and the document structure. It also includes definitions, acronyms, and abbreviations as well the system overview to ensure clarity and understanding throughout the document and system.

- Section 2: System Architecture

  The second section in the document is about the system architecture which includes information about the overall architecture of the system, including its components, modules, interfaces, and gives a general description about the functionality and design of the system. The architectural description provides a high-level overview of the system's structure including its components, relationships, and the overall design and organization of the system. Meanwhile, the design rationale explains the reasoning behind the system design, addressing the reason why specific architectural patterns and components were selected and how they meet the requirements. The decomposition descriptions involve breaking down the system into smaller parts, manageable components and interactions to ensure the development and scalability of the system.

- Section 3: Data Design

  The primary focus of this section is on the design and structure of the data used by the system. It includes the information about the Database description, mapping of the problem domain's objects to relational tables, and data dictionary. The database description provides an overview of the database's structure and organization, including the tables and columns. The mapping of the problem domain objects to relational tables describes how objects are represented in the database. Meanwhile, the data dictionary defines the meaning and application of the data used in the system.

- Section 4: Component Design

This section of the document describes the design of the components that create the system out. This includes information about the details of the classes, packages, their respective functions, dependencies, and interfaces that each one has. A general explanation of each package's purpose, function, dependencies, and components can be found in this section. The class identifier section provides details about specific classes within a package, including their purpose, dependencies, subordinate classes, and functions. This section serves to provide a clear understanding of the structure to make it more visible to achieve our desired outcomes.

- Section 5: Human Interface Design (Screens)

  The Human Interface Design section of this document focuses on the design of the user interface, which includes the screens, objects, and actions for the system. It includes the overview of the user interface, screen images, objects and actions given in the document along with the report. The screen images provide a visual representation of the system including the system's layouts and elements. This section also shows how it interacts with the user and describes the objects and actions available on each screen.

- Section 6: Traceability Requirements Matrix

  This section is a part of the document that provides the information of the Traceability Requirements Matrix tool. It outlines the purpose and the method to ensure that all product requirements are systematically tracked throughout the development process. The purpose of the matrix is to trace the relationships between requirements and elements of the system that implement those requirements. The Traceability Requirements Matrix is an important tool that helps to enhance the overall quality and the correctness of the system.

- Section 7: Resources Estimates

  This section provides a complete overview of the hardware and software resources required to make sure that the application runs smoothly. This includes the hardware requirements specifications, which include both minimum and recommended

configurations as well as the requirements for networking, processor (CPU), RAM, storage and software connections.

- Section 8: Appendices

  Appendices section includes the additional details that support and enhance the primary content of the document. This section refers to the extra materials that offer additional viewpoints or sources that viewers can read into a particular aspect of the software design that may be useful. In this section, we have provided an Entity Relationship Diagram, and Sequence Class Diagram.

1.4 Definitions, Acronyms and Abbreviations

Table 1.5.1 Terms and Definitions

Terms	Definition
Detail Class Diagram	A detailed class diagram is a visual representation of how different parts of a software system are connected. It shows classes, which represent objects or concepts, along with their attributes including the characteristics or properties. Meanwhile methods refer to the actions they can perform. It also includes relationships between classes, such as how they depend on or interact with each other. These relationships can be associations, inheritance when one class takes features from another, or dependencies when one class relies on another. This diagram also helps to plan, organize, and understand how a system works before writing the actual code.
Package Diagram	A package diagram is a one of the diagrams in software design that helps organize and group related parts of a system. It includes the packages that contain related classes or components. These packages help manage complex systems by keeping things organized and reducing dependencies between different

	parts. The diagram also shows relationships between packages, such as dependencies when one package relies on another. This diagram makes it easier to understand how different sections of a system are connected and how they interact with each other.
System Architecture	The architectural description provides a high-level overview of the system's structure including its components, relationships, and the overall design and organization of the system. System architecture also helps to understand the structure of a system and make it easier to plan, build, and maintain the system.

Table 1.5.2 Acronyms and Descriptions

Acronyms	Descriptions
SDD	Software Design Document
UCD	Use Case Description
UCID	USe Case ID
UC	Use Case (Ex: UC-100)
RAM	Random Access Memory

Table 1.5.3 Abbreviations and Descriptions

Abbreviations	Descriptions
Ex	Example
N/A	Not Applicable
IC	Identity Card
HTML	HyperText Markup Language
etc	Et cetera

1.5 System Overview

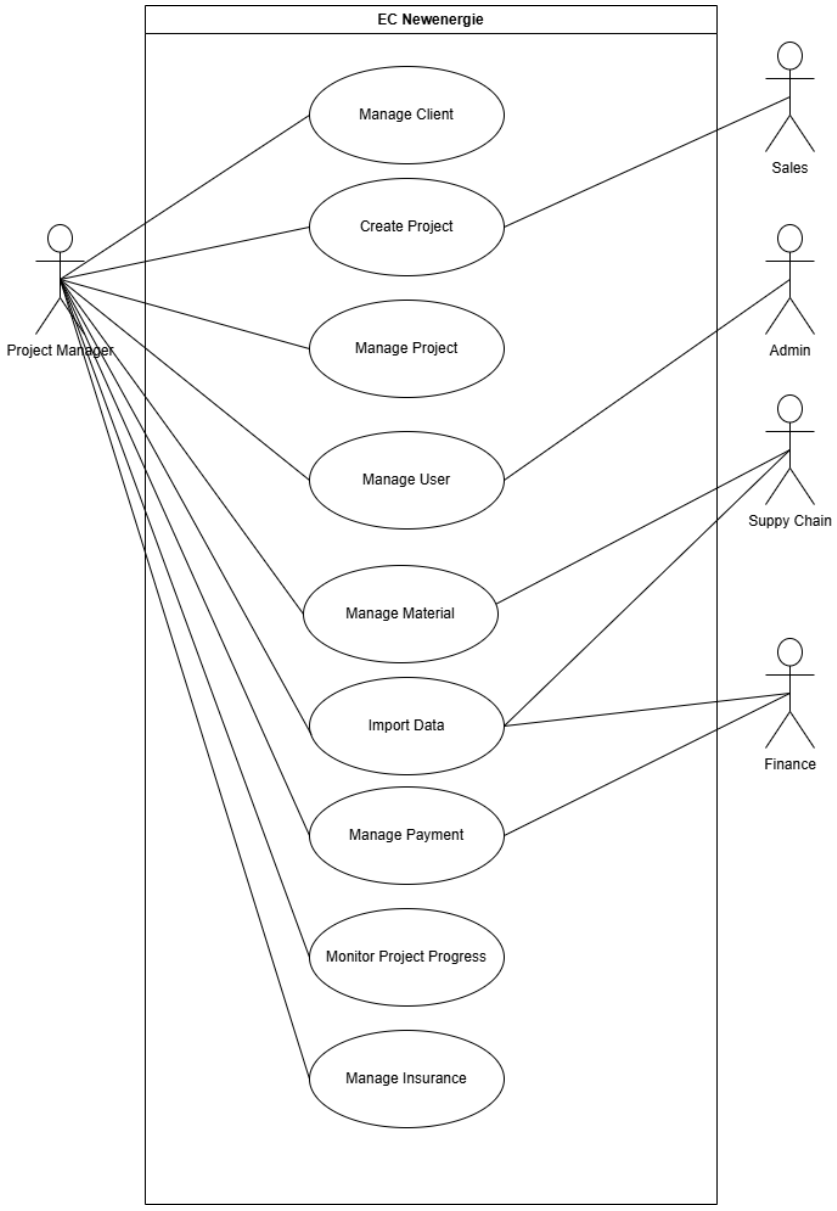


Figure 1.6.1 Use Case Diagram of Promat-D Project Management Tracking System For EC Newenergie Solar Company With Dashboard

Table 1.6.1 Actor, Use Case, UCID, and Descriptions for the Use Case of Promat-D Project Management Tracking System For EC Newenergie Solar Company With Dashboard

Actor(s)	**Use Case**	**UCID**	**Brief Description**
Project Manager, Admin	Manage User	UC-100	Actors create, view, update or delete data for User
Project Manager	Manage Client	UC-200	Actors create, view, update or delete data for client
	Monitor Project Progress	UC-300	Actors view the progress of all the project
	Manage Project	UC-400	Actors create, view, edit or delete data for projects in the company
	Manage Insurance	UC-500	Actors create, view, update or delete data for the project insurance
Project Manager, Supply Chain, Finance	Import Data	UC-600	Actors upload the excel file to insert existing data into database
Project Manager, Sales	Create Project	UC-700	Actors create new project
Project Manager, Supply Chain	Manage Material	UC-800	Actors create, view, update or delete data for the material
Project Manager, Finance	Manage Payment	UC-900	Actors create, view, update or delete data for the payment

2. System Architecture

2.1 Architectural Description

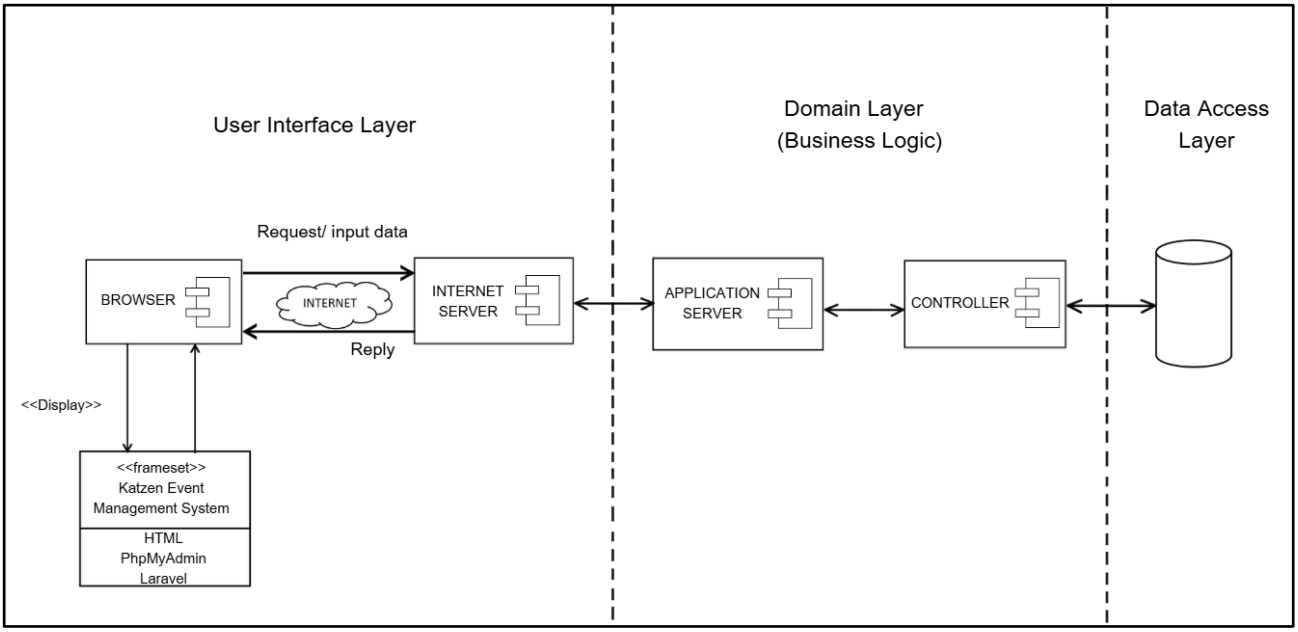


Figure 2.1 UML Component Diagram of Katzen Event Management System

Katzen event management system is designed with a three-layer architecture, which consists of the View Layer, Domain Layer, and Data Access Layer. The View Layer is the interface between the user and the system. It consists of a browser, internet server component, view page, and sessions. The view page is built using HTML, PhpMyAdmin, and Laravel, and is responsible for displaying information and accepting input data from the user. The browser communicates with the internet server through the internet, sending requests and receiving responses.

The Domain Layer acts as the heart of the system, processing the requests received from the View Layer. It consists of an application server, business logic classes, and response pages. The application server processes the requests according to the business logic classes and then sends out a response back to the View Layer using the appropriate response page. This layer is responsible for implementing the business logic and ensuring that the data received from the View Layer is processed correctly.

2.2 Design Rationale

The three-layer architecture described in Section 2.1 was selected for its ability to support scalability, maintainability, and modularity in system design. This architecture divides the system into three distinct layers—User Interface, Domain (Business Logic), and Data Access—each responsible for specific functionalities. This separation of concerns ensures that changes in one layer, such as upgrading the user interface or altering the database, do not affect the others. It also allows developers to work on each layer independently, streamlining the development process and minimizing dependencies.

One of the critical issues considered was the need for a system that can handle high user interaction while maintaining responsiveness. By using a browser-based user interface, the architecture leverages widely supported technologies like HTML, PhpMyAdmin, and Laravel, enabling accessibility across multiple platforms. At the same time, the application server in the Domain Layer ensures that the business logic is processed efficiently, while the Data Access Layer manages database operations securely and effectively. This approach balances usability with performance and security.

Ultimately, the three-layer architecture was chosen as the optimal balance between simplicity and scalability. It provides a structured, organized framework that is easier to maintain and extend over time while avoiding the pitfalls of overly simplistic or excessively complex alternatives. The modular design also positions the system well for future enhancements, whether scaling for increased demand or integrating new technologies.

2.3 Decomposition Description

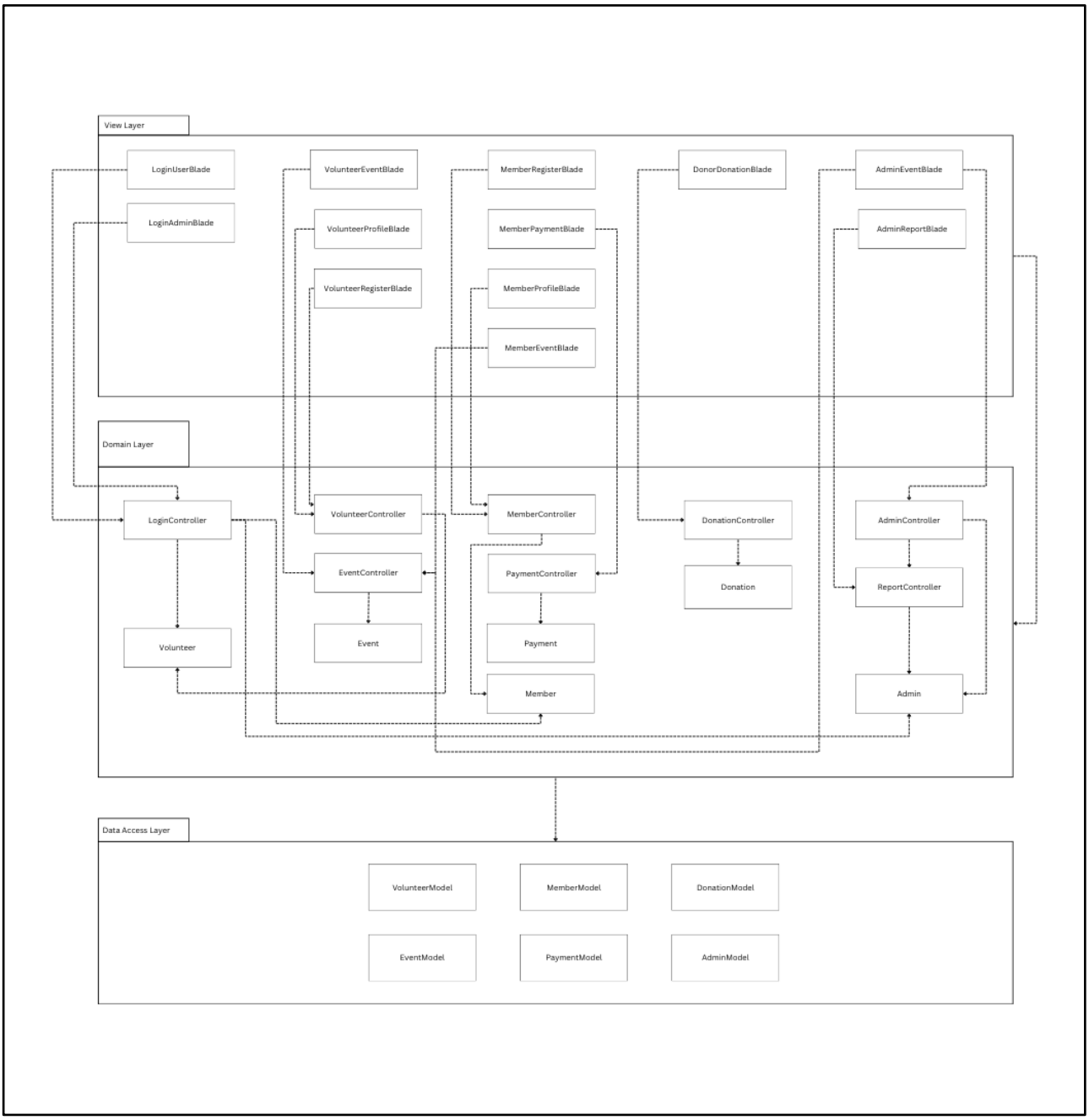


Figure 2.3.1 Package Class Diagram of Katzen Event Management System

The figure above represents a system architecture diagram for the Katzen Event Management System that shows how different parts of the system interact. The system is divided into three main layers: View Layer, Domain Layer, and Data Access Layer. Firstly, the View Layer includes different interface components, such as LoginUserBlade, VolunteerRegisterBlade, MemberProfileBlade, DonorDonationBlade, and AdminReportBlade. These components allow users to interact with the system, including volunteers, members, donors, and admins. Meanwhile, the Domain Layer in the middle contains the core logic and processing components. It includes various controllers such as LoginController, VolunteerController, MemberController, EventController, PaymentController, DonationController, and ReportController. These controllers handle user requests and connect the View Layer with the system's data. They interact with key entities like Volunteer, Member, Event, Payment, Donation, and Admin, which store important system information. Lastly is the Data Access Layer which is responsible for storing and retrieving data. It includes models such as VolunteerModel, MemberModel, EventModel, PaymentModel, DonationModel, and AdminModel. These models interact with the database to ensure that information is stored, updated, and accessed when needed. This helps in managing event registrations, donations, payments, and user profiles efficiently.

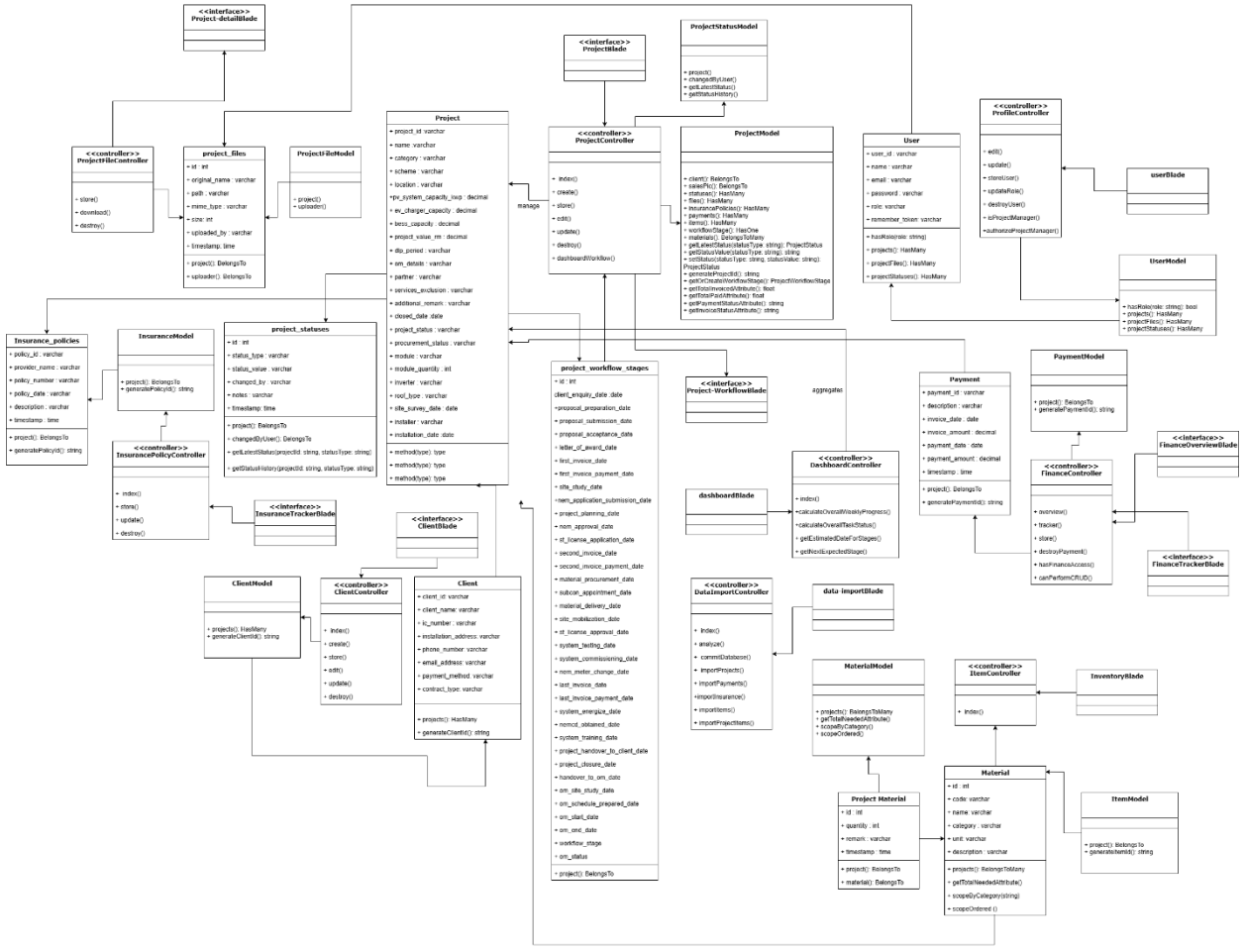


Figure 2.3.2 Detail Design Class Diagram of Promat-D Project Management Tracking System For EC Newenergie Solar Company With Dashboard

The Katzen Event Management System comprises 10 primary domain classes, which form the core of this system. This system incorporates a total of 10 main entities, including Volunteer, Member, Admin, Event, Donor, Donation, Payment, Report, Attendance, and Controllers. Each class has specific attributes such as names, IDs, and dates and methods. The system helps manage volunteer registrations, manage events, donations, payments, and reports by connecting these entities. Different Controllers are used to process actions like registering users, verifying payments and donations, and managing event details etc. These classes are connected through various relationships, such as associations between volunteers and events, members and payments, or donations and donors. The system ensures smooth event management by organizing all these entities and their interactions effectively.

3. Data Design

3.1 Database Description

1. Database Description

   The system utilizes a relational database to manage the key entities and relationships necessary for its operation. The database is designed to support the management of events, members, volunteers, donors, donations, payments, and reports. It ensures data integrity and supports efficient query processing.

2. Key Features of the Database
   - Normalization: The database is normalized to reduce redundancy and ensure efficient data storage.
   - Primary and Foreign Keys: Each table includes primary keys for unique identification and foreign keys to maintain relationships between tables.
   - Scalability: The design supports scalability, enabling future extensions to accommodate more entities or relationships.

Tables and Their Roles:

- Event: Manages information about events, including names, dates, descriptions, and associated admins.
- Volunteer: Stores data about volunteers who contribute to events, linked to registration and admin approvals.
- Register_volunteer: Handles temporary data for volunteers pending admin approval.
- Admin: Contains admin details who oversee operations and approvals.
- Member: Holds member information for individuals participating in the system.
- Donor: Records details about donors contributing to the system.
- Donation: Tracks donations made by donors, including amounts and dates.
- Payment: Manages payments made by members, including methods and statuses.
- Report: Stores reports generated by admins for tracking and analysis purposes.
- Attendance: Tracks attendance of volunteers at events.

3.2 Mapping of Problem Domain Objects to Relational Tables

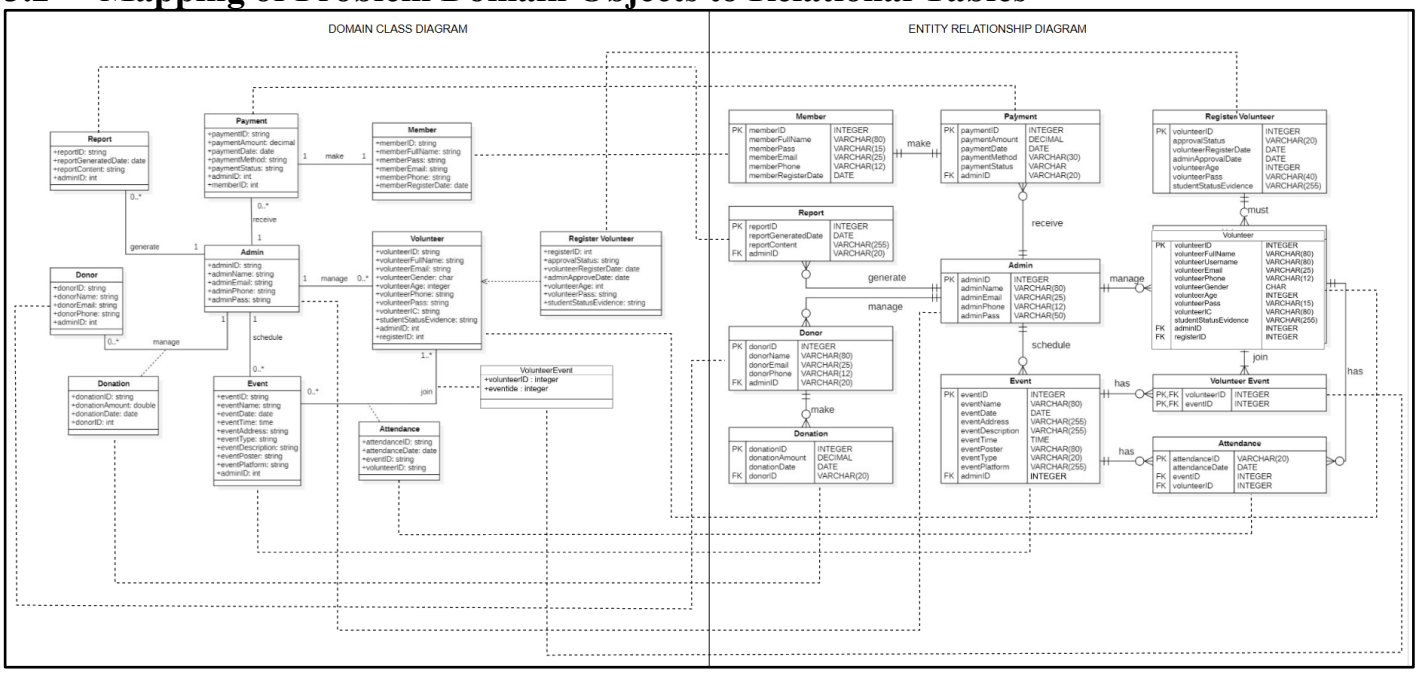


Figure 3.2.1 Mapping of Problem Domain Objects to Relational Tables

The system is structured around several core data tables. The Member table captures details about registered members, such as their full name, contact information, and registration date. Payments made by members or donors are tracked in the Payment table, which includes details like amount, date, method, and status, along with a reference to the associated administrator.

Volunteer registration and management are handled through two tables: Register Volunteer, which stores registration details and approval statuses, and Volunteer, which maintains detailed profiles of approved volunteers. These volunteers can participate in events managed by the Event table, which holds information like event name, date, location, and type.

To link volunteers to events, the Volunteer Event table is used, representing a many-to-many relationship. Attendance records for these events are maintained in the Attendance table. The system also manages donations through the Donor and Donation tables, where

donors' details and their contributions are recorded. Lastly, the Report table stores reports generated by administrators, who oversee the system and are represented by the Admin table.

3.3 Data Dictionary

Table	Member				
No	**Attributes**	**Datatype**	**Constraint**	**Example**	**Description**
1	memberID	Integer	PK,NOT NULL	1	Primary Key
2	memberFullName	Varchar(80)	-	Ahmad Adam bin Ali	-
3	memberPass	Varchar(15)	-	Ahmad@123	-
4	memberEmail	Varchar(25)	-	aadam18@gmail.com	-
5	memberPhone	Varchar(12)	-	01998765797	-
6	memberRegisterDate	Date	-	18-12-2024	-

Table	Payment				
No	**Attributes**	**Datatype**	**Constraint**	**Example**	**Description**
1	paymentID	Integer	PK,NOT NULL	1	Primary Key
2	paymentAmount	Decimal	-	5.00	-
3	paymentDate	Date	-	10-01-2025	-
4	paymentMethod	Varchar(30)	-	Online	-
5	paymentStatus	Varchar(20)	-	Paid	-
6	adminID	Varchar(20)	FK	1	Foreign Key

Table	Admin				
No	**Attributes**	**Datatype**	**Constraint**	**Example**	**Description**

1	adminID	Integer	PK,NOT NULL	1	Primary Key
2	adminName	Varchar(80)	-	Suraya binti Aziz	-
3	adminEmail	Varchar(25)	-	suraya60@gmail.com	-
4	adminPhone	Varchar(12)	-	01355668882	-
5	adminPass	Varchar(50)	-	Suraya12@56	-

Table	Report				
No	**Attributes**	**Datatype**	**Constraint**	**Example**	**Description**
1	reportID	Integer	PK,NOT NULL	1	Primary Key
2	reportGeneratedDate	Date	-	14-01-2025	-
3	reportContent	Varchar(255)	-	Member List	-
4	adminID	Varchar(20)	FK	1	Foreign key

Table	Donor				
No	**Attributes**	**Datatype**	**Constraint**	**Example**	**Description**
1	donorID	Integer	PK,NOT NULL	1	Primary Key
2	donorName	Varchar(80)	-	Amin bin Fauzi	-
3	donorEmail	Varchar(25)	-	aminf@gmail.com	-
4	donorPhone	Varchar(12)	-	01456548979	-
5	adminID	Varchar(20)	FK	1	Foreign Key

Table	Donation				
No	**Attributes**	**Datatype**	**Constraint**	**Example**	**Description**

1	donationID	Integer	PK,NOT NULL	1	Primary Key
2	donationAmount	Decimal	-	35.00	-
3	donationDate	Date	-	02-01-2025	-
4	donorID	Varchar(20)	Foreign Key	1	Foreign Key

Table	Volunteer				
No	**Attributes**	**Datatype**	**Constraint**	**Example**	**Description**
1	volunteerID	Integer	PK,NOT NULL	1	Primary Key
2	volunteerFullName	Varchar(80)	-	Nur Anis bin Mohamad	-
3	volunteerUserName	Varchar(80)			
4	volunteerEmail	Varchar(25)	-	anismohd23@gmail.com	-
5	volunteerPhone	Varchar(12)	-	013245423199	-
6	volunteerGender	Char	-	Female	-
7	volunteerAge	Integer	-	22	-
8	volunteerPass	Varchar(15)	-	Anis@1239	-
9	volunteerIC	Varchar(80)	-	030911033452	-
10	studentStatusEvidence	Varchar(255)	-	studentEvidence.png	-
11	adminID	Varchar(20)	FK	1	Foreign Key
12	registerID	Integer	FK	1	Foreign Key

Table	RegisterVolunteer				
No	**Attributes**	**Datatype**	**Constraint**	**Example**	**Description**
1	volunteerID	Integer	PK, NOT NULL	1	Primary Key
2	approvalStatus	Varchar(20)	-	Approved	-
3	volunteerRegisterDate	Date	-	05-02-2025	-
4	adminApprovalDate	Date	-	07-02-2025	-
5	volunteerAge	Integer	-	22	-
6	volunteerPass	Varchar(40)	-	Anis@1239	-
7	studentStatusEvidence	Varchar(255)	-	studentEvidence.png	-

Table	Event				
No	**Attributes**	**Datatype**	**Constraint**	**Example**	**Description**
1	eventID	Integer	PK,NOT NULL	1	Primary Key
2	eventName	Varchar(80)	-	Meow Fest	-
3	eventDate	Date	-	10-02-2025	-
4	eventAddress	Varchar(255)	-	Cat Paws Sanctuary, Petaling Jaya	-
5	eventDescription	Varchar(255)	-	A festival celebrating cats with various activities.	-
6	eventTime	Time	-	11:00:00	-
7	eventPoster	Varchar(80)	-	poster.png	-
8	eventType	Varchar(20)	-	Physical	-
9	eventPlatform	Varchar(255)	-	Not Related	-
10	adminID	Varchar(20)	-	1	Foreign Key

Table	VolunteerEvent				
No	**Attributes**	**Datatype**	**Constraint**	**Example**	**Description**
1	volunteerID	Integer	PK,FK,NOT NULL	1	Primary Key, Foreign Key
2	eventID	Integer	PK,FK,NOT NULL	1	Primary Key, Foreign Key

Table	Attendance				
No	**Attributes**	**Datatype**	**Constraint**	**Example**	**Description**
1	attendanceID	Integer	PK,NOT NULL	1	Primary Key
2	attendanceDate	Date	-	10-02-2025	-
3	eventID	Integer	FK	1	Foreign Key
4	volunteerID	Integer	FK	1	Foreign Key

4. Component Design

4.1 Package Identifier

Identifier	Package
(SDD_PD_100)	View Layer
(SDD_PD_200)	Domain Access Layer
(SDD_PD_300)	Data Layer

4.1.1 Package Purpose

In our project development, we have identified three important packages: View, Domain, and Data Access. These packages help keep the software well-organized by grouping related classes based on what they do. This makes the system easier to understand, manage, and update as needed. Instead of having all the code mixed together, we separate it into these packages so that each one has a clear purpose and function. To better understand how these packages work together, we use package diagrams, which are part of the Unified Modeling Language (UML). These diagrams give us a big-picture view of the software's structure, showing how different classes are grouped and connected within each package. This visual representation makes it much easier to plan, build, and improve the system. Each package plays a unique role. By organizing the system this way, we create a software design that is easier to develop, maintain, and expand. It helps developers work more efficiently, reduces errors, and makes future improvements simpler. In the end, this structured approach leads to a more reliable and effective software system.

4.1.2 Package Function

Package	Package Function
View Layer (SDD_PD_100)	1. Responsible for displaying everything the user sees and interacts with. It includes screens, forms, buttons, menus, and other visual elements that

	make up the user interface. 2. Handles user actions, such as clicking buttons, typing on a keyboard, or moving a mouse. Whenever a user interacts with the software, this part captures their input and sends it to the next layer, called the controller, which processes the information and determines what happens next.
Domain Layer (SDD_PD_200)	1. Process information according to the rules and logic that define how the software should work. It follows specific business rules and requirements to ensure that the data is handled correctly and that the software behaves as expected. 2. Takes care of managing and organizing the application's data that works closely with the model layer to retrieve existing information, update records, and store new data in a secure and structured way.
Data Access Layer (SDD_PD_300)	1. Handles storing and retrieving data from data storage based on specific queries or criteria. It ensures that only the necessary information is fetched, and to make the data access quick and efficient. 2. Takes care of organizing, updating, and maintaining the stored data. Whether it's adding new records, modifying existing ones, or deleting outdated information, to ensure that all data is accurate, secure, and properly managed.

4.1.3 Package Dependencies

In this software system, all the packages are connected and rely on each other to function properly. The View Layer depends on the Controller Layer to process user input and display the right information to the user. It also relies on the Data Access Layer to fetch stored data from the database when needed. The Domain Layer needs information from the View Layer to process data and apply business rules, ensuring the system works correctly. Additionally, the Domain Layer depends on the Data Access Layer to store and retrieve processed data from the database. This connection between layers ensures smooth communication and efficient data flow, making the system work as expected.

4.1.4 Package Components

Package	Classes
View Layer (SDD_PD_100)	LoginUserBlade LoginAdminBlade VolunteerEventBlade VolunteerProfileBlade VolunteerRegisterBlade MemberRegisterBlade MemberProfileBlade MemberEventBlade DonorDonationBlade DonorEventBlade AdminEventBlade AdminReportBlade
Domain Layer (SDD_PD_200)	LoginController VolunteerController EventController MemberController

	PaymentController DonorController AdminController ReportController Admin Volunteer Member Event Donation Payment
Data Access Layer (SDD_PD_300)	VolunteerModel EventModel MemberModel PaymentModel DonationModel AdminModel

4.1.4.1 Class Identifier

Package	Classes
View Layer (SDD_PD_100)	LoginUserBlade (SDD_PD_101) LoginAdminBlade (SDD_PD_102) VolunteerEventBlade (SDD_PD_103) VolunteerProfileBlade (SDD_PD_104) VolunteerRegisterBlade (SDD_PD_105) MemberRegisterBlade (SDD_PD_106) MemberProfileBlade (SDD_PD_107) MemberEventBlade (SDD_PD_108)

	DonorDonationBlade (SDD_PD_109) DonorEventBlade (SDD_PD_110) AdminEventBlade (SDD_PD_111) AdminReportBlade (SDD_PD_112)
Domain Layer (SDD_PD_200)	LoginController (SDD_PD_201) VolunteerController (SDD_PD_202) EventController (SDD_PD_203) MemberController (SDD_PD_204) PaymentController (SDD_PD_205) DonorController (SDD_PD_206) AdminController (SDD_PD_207) ReportController (SDD_PD_208) Admin (SDD_PD_209) Volunteer (SDD_PD_210) Member (SDD_PD_211) Event (SDD_PD_212) Donation (SDD_PD_213) Payment (SDD_PD_214)
Data Access Layer (SDD_PD_300)	VolunteerModel (SDD_PD_301) EventModel (SDD_PD_302) MemberModel (SDD_PD_303) PaymentModel (SDD_PD_304) DonationModel (SDD_PD_305) AdminModel (SDD_PD_306)

4.1.4.2 Class Purpose

Each class is placed in a specific package based on its functionalities. This helps keep the software well-organized and easy to manage. The purpose of class in the view layer is to receive input from the user and display the output for the user. In the domain layer, classes

oversee the flow of business processes and direct methods to handle business logic. Meanwhile, the data access layer takes care of storing and retrieving data. It connects with the database to save new information, update existing records, find needed data, and remove unnecessary entries. This layer ensures that all data is well-organized and easily accessible when needed. Together, these layers work smoothly to make the software efficient, reliable, and easy to maintain.

Class	Purpose
LoginUserBlade (SDD_PD_101)	This class will be use to receive input from user for login process
LoginAdminBlade (SDD_PD_102)	This class will be use to receive input from admin for login process
VolunteerEventBlade (SDD_PD_103)	This class will be use to receive input and display data for volunteer to see the event
VolunteerProfileBlade (SDD_PD_104)	This class will be use to receive input and display volunteer's profile
VolunteerRegisterBlade (SDD_PD_105)	This class will be use to input and submit new volunteer details that want to register
MemberRegisterBlade (SDD_PD_106)	This class will be use to input and submit new member details that want to register
MemberProfileBlade (SDD_PD_107)	This class will be use to receive input and display member's profile
MemberEventBlade (SDD_PD_108)	This class will be used to receive input and display data for member to see the event
DonorDonationBlade (SDD_PD_109)	This class will be use to input donor's details and payment for donation

AdminEventBlade (SDD_PD_110)	This class will be use to receive input and display data for admin to see the event
AdminReportBlade (SDD_PD_111)	This class will be use to receive input and display data for admin to generate report
LoginController (SDD_PD_201)	This class provides the authentication, access control, and acts as a switchboard to forward them to admin, volunteer, and member to log in to the system
VolunteerController (SDD_PD_202)	This class will be use to manage the input from VolunteerProfileBlade and VolunteerRegisterBlade and act as a switchboard to forward them to Volunteer
EventController (SDD_PD_203)	This class will be use to manage the input from VolunteerEventBlade, MemberEventBlade, and AdminEventBlade and act as a switchboard to forward them to Event
MemberController (SDD_PD_204)	This class will be use to manage the input from MemberProfileBlade and MemberRegisterBlade and act as a switchboard to forward them to Member
PaymentController (SDD_PD_205)	This class will be use to manage the input from MemberPaymentBlade and act as a switchboard to forward them to Payment
DonationController (SDD_PD_206)	This class will be use to manage the input from DonorDonationBlade and act as a switchboard to forward them to Donation and Donor
AdminController (SDD_PD_207)	This class will be use to manage the input from AdminEventBlade and act as a switchboard to forward

	them to Admin
ReportController (SDD_PD_208)	This class will be use to manage the input from AdminReportBlade and AdminController and act as a switchboard to forward them to Admin
Admin (SDD_PD_209)	This class will represent the volunteer that will store admin attributes that will be needed in managing and view events, manage volunteers, and members.
Volunteer (SDD_PD_210)	This class will represent the volunteer that will store volunteer attributes that will be needed in registration, manage profile, and view event process.
Member (SDD_PD_211)	This class will represent the volunteer that will store member attributes that will be needed in registration, manage profile, and view event process.
Event (SDD_PD_212)	This class will represent the volunteer that will store events attributes that will be needed in view events and manage the event process.
Donation (SDD_PD_213)	This class will represent the volunteer that will store donation attributes that will be needed in the donation process.
Payment (SDD_PD_214)	This class will represent the volunteer that will store payment attributes that will be needed in the payment process.
VolunteerModel (SDD_PD_301)	This class will be use to establish connection with volunteer table in database and to store data about volunteer

EventModel (SDD_PD_302)	This class will be use to establish connection with event table in database and to store data about event
MemberModel (SDD_PD_303)	This class will be use to establish connection with member table in database and to store data about member
PaymentModel (SDD_PD_304)	This class will be use to establish connection with payment table in database and to store data about payment
DonationModel (SDD_PD_305)	This class will be use to establish connection with donation table in database and to store data about donation
AdminModel (SDD_PD_306)	This class will be use to establish connection with admin table in database and to store data about admin

4.1.4.3 Class function

View Layer		
Class	**Pseudocode**	**Method**
LoginUserBlade (SDD_PD_101)	BEGIN 1. Display LoginUserBlade END	
LoginAdminBlade (SDD_PD_102)	BEGIN 1. Display LoginAdminBlade END	
VolunteerEventBlade	BEGIN	

(SDD_PD_103)	1. Display VolunteerEventBlade END	
VolunteerProfileBlade (SDD_PD_104)	BEGIN 1. Display VolunteerProfileBlade 2. Update volunteer's profile END	
VolunteerRegisterBlade (SDD_PD_105)	BEGIN 1. Display make registration form 2. Submit inserted account details END	
MemberRegisterBlade (SDD_PD_106)	BEGIN 1. Display make registration form 2. Submit inserted account details END	
MemberProfileBlade (SDD_PD_107)	BEGIN 1. Display MemberProfileBlade 2. Update member's profile END	
MemberEventBlade (SDD_PD_108)	BEGIN 1. Display MemberEventBlade END	
DonorDonationBlade (SDD_PD_109)	BEGIN 1. Display DonorDonationBlade END	
AdminEventBlade (SDD_PD_110)	BEGIN 1. Display AdminEventBlade END	
AdminReportBlade (SDD_PD_111)	BEGIN 1. Display AdminReportBlade END	

Domain Layer

Class	Pseudocode	Method
LoginController (SDD_PD_201)	BEGIN 1. Verify username and password END	login () getAdminID () getVolunteerUsername () getMemberUsername ()
VolunteerController (SDD_PD_202)	BEGIN If value method is create volunteer account 1. Display registration form. 2. Create a volunteer account. 3. Store volunteer details. 4. Submit registration form. If value method is update volunteer account 1. Display volunteer profile. 2. Update volunteer profile. 3. Store updated details. END	createVolunteer () updateVolunteer () viewVolunteerDetails () getVolunteerFullName () getVolunteerPhone () getVolunteerEmail ()
EventController (SDD_PD_203)	BEGIN If value method is create new event 1. Create event data 2. Store new event data in the Event table 3. Return eventlist with upcomingEvents and pastEvents If value method is view event 1. list all event with its details If value method is edit event 1. retrieve input from html form 2. Update event data 3. Store new event data in Event	createEvent() updateEvent() deleteEvent() viewEventDetails()

	table 4. Return eventlist with upcomingEvents and pastEvents with success message If value method is delete event 1. retrieve input from html form 2. Delete event and its data 3. Return eventlist with upcomingEvents and pastEvents with success message END	
MemberController (SDD_PD_204)	BEGIN If value method is registerMember 1. Retrieve input from page 2. Show memberbfrregister 3. Retrieve input from page 4. Create member data access object 5. Check for username and password validity 6. Store member data in member table If value method is paymentMember 1. Check if memberID is exist 2. Redirect to payment page If value method is memberafterlogin 1. Check if payment is done 2. Redirect to login page END	createMember () updateMember () viewMemberDetails ()
PaymentController (SDD_PD_205)	BEGIN If value method is create bill 1. Find memberID in member table 2. Set payment options 3. send request to Toyyibpay API	verifyAdmin() verifyMember() savePaymentDetails()

	4. Retrieve bill code from Toyyibpay If value method is paymentStatus 1. Retrieve payment details from ToyyibPay response 2. Set default payment amount and method 3. Store payment details in database 4. Redirect to success message If value method is paymentCallback 1. Set payment status to Paid 2. Store payment details in database 3. Update member record with transaction ID 4. Redirect to login page with success message "Payment successful!" END	
DonationController (SDD_PD_206)	BEGIN If value method is donate 1. Redirect to donate page If value method is save 1. Validate request inputs 2. Create new donor record in database 3. Create donation record in database If value method is createBill 1. FIND donor by donorID 2. Retrieve donationAmount from request 3. Set payment options 4. Send request to ToyyibPay API 5. Redirect to Toyyibpay payment page	

	If value method is paymentStatus 1. Retrieve payment details from ToyyibPay response 2. Set default payment amount and method 3. Store payment details in database 4. Redirect to success message If value method is paymentCallback 1. Set payment status to Paid 2. Store donation details in database 3. Update donor record with transaction ID 4. Redirect to login page with success message "Donation successful!" END	
AdminController (SDD_PD_207)	BEGIN If value method is create admin account 1. Display registration form. 2. Create a admin account. 3. Store admin details. 4. Submit registration form. If value method is update admin account 1. Display admin profile. 2. Update admin profile. 3. Store updated details. END	createAdmin () updateAdmin () viewAdminDetails ()
ReportController (SDD_PD_208)	BEGIN If value method is generateReport 1. Retrieve input from html 2. Extract type, month, year,	generateReport() getAdmin()

	and current timestamp 3. Return printable report view with generated data If value method is download 1. Fetch report from database 2. Return printable report view with stored data END	
Admin (SDD_PD_209)	BEGIN AdminModel USE LaravelAuthenticatable, Notifiable DEFINE TABLE admin DEFINE PRIMARY KEY adminID DISABLE timestamps DEFINE FILLABLE ATTRIBUTES: - adminName - adminEmail - adminPhone - adminPass DEFINE HIDDEN ATTRIBUTES: - adminPass END AdminModel	admin()
Volunteer (SDD_PD_210)	BEGIN VolunteerModel USE LaravelModel, HasFactory DEFINE TABLE volunteer DEFINE PRIMARY KEY volunteerID ENABLE timestamps DEFINE FILLABLE ATTRIBUTES:	volunteer()

	- volunteerFullName - volunteerUsername - volunteerEmail - volunteerPhone - volunteerGender - volunteerAge - volunteerPass - volunteerIC - studentStatusEvidence - adminID - registerID FUNCTION registerVolunteer() RETURN One-to-One Relationship with RegisterVolunteer using volunteerID END VolunteerModel	
Member (SDD_PD_211)	BEGIN MemberModel USE LaravelModel, HasFactory DEFINE TABLE member DEFINE PRIMARY KEY memberID DISABLE timestamps DEFINE FILLABLE ATTRIBUTES: - memberFullName - memberUsername - memberEmail - memberPhone - memberRegisterDate - paymentID FUNCTION payment() RETURN Many-to-One Relationship with Payment using paymentID END MemberModel	member()

Event (SDD_PD_212)	BEGIN EventModel DEFINE TABLE event DEFINE PRIMARY KEY eventID DEFINE FILLABLE ATTRIBUTES: - eventName - eventDate - eventTime - eventType - eventPlatform - eventDescription - eventPoster - eventAddress FUNCTION volunteers() RETURN Many-to-Many Relationship with Volunteer through volunteer_event TABLE ENABLE timestamps FUNCTION attendance() RETURN One-to-Many Relationship with VolunteerEvent using eventID END EventModel	event()
Donation (SDD_PD_213)	BEGIN DonationModel USE LaravelModel, HasFactory DEFINE TABLE donation DEFINE PRIMARY KEY donationID ENABLE timestamps DEFINE FILLABLE ATTRIBUTES: - donationAmount - donationDate - donorID	donation()

	FUNCTION donor() RETURN Many-to-One Relationship with Donor using donorID END DonationModel	
Payment (SDD_PD_214)	BEGIN PaymentModel USE LaravelModel, HasFactory DEFINE TABLE payment DEFINE PRIMARY KEY paymentID DEFINE FILLABLE ATTRIBUTES: - paymentAmount - paymentDate - paymentMethod - paymentStatus - adminID - memberID FUNCTION member() RETURN Many-to-One Relationship with Member using memberID	payment()

Data Access Layer		
Class	**Pseudocode**	**Method**
VolunteerModel (SDD_PD_301)	BEGIN VolunteerModel USE LaravelModel, HasFactory DEFINE TABLE volunteer DEFINE PRIMARY KEY volunteerID	admin()

	ENABLE timestamps DEFINE FILLABLE ATTRIBUTES: - volunteerFullName - volunteerUsername - volunteerEmail - volunteerPhone - volunteerGender - volunteerAge - volunteerPass - volunteerIC - studentStatusEvidence - adminID - registerID FUNCTION registerVolunteer() RETURN One-to-One Relationship with RegisterVolunteer using volunteerID END VolunteerModel	
EventModel (SDD_PD_302)	BEGIN EventModel DEFINE TABLE event DEFINE PRIMARY KEY eventID DEFINE FILLABLE ATTRIBUTES: - eventName - eventDate - eventTime - eventType - eventPlatform - eventDescription - eventPoster - eventAddress FUNCTION volunteers() RETURN Many-to-Many Relationship with Volunteer	volunteers() attendance() getAuthPassword()

	through volunteer_event TABLE ENABLE timestamps FUNCTION attendance() RETURN One-to-Many Relationship with VolunteerEvent using eventID END EventModel	
MemberModel (SDD_PD_303)	BEGIN MemberModel USE LaravelModel, HasFactory DEFINE TABLE member DEFINE PRIMARY KEY memberID DISABLE timestamps DEFINE FILLABLE ATTRIBUTES: - memberFullName - memberUsername - memberEmail - memberPhone - memberRegisterDate - paymentID FUNCTION payment() RETURN Many-to-One Relationship with Payment using paymentID END MemberModel	payments() getAuthPassword()
PaymentModel (SDD_PD_304)	BEGIN PaymentModel USE LaravelModel, HasFactory DEFINE TABLE payment DEFINE PRIMARY KEY paymentID	member()

	DEFINE FILLABLE ATTRIBUTES: - paymentAmount - paymentDate - paymentMethod - paymentStatus - adminID - memberID FUNCTION member() RETURN Many-to-One Relationship with Member using memberID END PaymentModel	
DonationModel (SDD_PD_305)	BEGIN DonationModel USE LaravelModel, HasFactory DEFINE TABLE donation DEFINE PRIMARY KEY donationID ENABLE timestamps DEFINE FILLABLE ATTRIBUTES: - donationAmount - donationDate - donorID FUNCTION donor() RETURN Many-to-One Relationship with Donor using donorID END DonationModel	donor()
AdminModel (SDD_PD_306)	BEGIN AdminModel USE LaravelAuthenticatable, Notifiable	

	DEFINE TABLE admin DEFINE PRIMARY KEY adminID DISABLE timestamps DEFINE FILLABLE ATTRIBUTES: - adminName - adminEmail - adminPhone - adminPass DEFINE HIDDEN ATTRIBUTES: - adminPass END AdminModel	

4.1.4.4 Class subordinates

No subordinates in this design.

4.1.4.5 Class Dependencies

The Design Class Diagram helps show how different classes are connected, whether they are in the same package or in different ones. It clearly displays the relationships between classes in the controller and model layers, as well as how the view layer depends on the controller. This diagram gives a visual representation of how different parts of the system work together, making it easier to understand how classes interact and rely on each other for the software to function smoothly.

4.1.4.6 Interfaces

The control and data flow from the controller to the class can be referred to in the Design Class Diagram in Chapter 2, Subchapter 2.3 Decomposition Description.

4.1.4.7 Data

All the data that is being used can be referred to in the Data Dictionary in Chapter 3, Subchapter 3.3 Data Dictionary.

5. Human Interface Design (Screens)

5.1 Overview of the User Interface

The Katzen Event Management System is designed to serve four main types of users: volunteers, members, donors, and the system administrator. Each of these stakeholders has a specific role within the system. The user interface is designed to be simple, clear, and efficient, making it easy for each user type to navigate and perform their tasks without confusion. By tailoring functionalities to each role, the system ensures a smooth experience for all users.

From the user's perspective, firstly, they will be redirected to the main page. On this page, there will be the navigation menu that allows users to view the About Us page and Events page. For donors, they can make their donation simply by clicking the Donate Now button. They also can click on the donate menu on the navigation bar, then it will redirect them to the donation page. On the donation page, the donors will need to choose the donation amount and enter their details including their full name, phone number, and email address. After they click on the continue button, it will be redirected to the payment gateway allowing them to choose on the payment method. For other users who are members and volunteers, they will need to click on the register menu on the navigation bar and choose between member and volunteer registration. For members, they will need to read up on the requirements before they can register to be a member. After that on the registration page, they need their details including fullname, username, email, phone number, and their password. They will then need to pay a RM5 registration one-time fee.

For volunteers, they will also need to read up on the requirements before they can register to be a volunteer. After that on the registration page, they need their details including fullname, username, email, phone number, gender, IC, a question that will ask whether they are a student or not and their password. If the user clicks on yes, they will need to give the student evidence whether they are truly a student or not. Once every field has been filled, they need to click on the register button to

create an account. In the case where there is no input error, the system will be redirected to the login page. However, if there is/are input error(s) done by the users, an error message(s) will be displayed respectively to the error that has been done. Alternatively, if they already have an account, they can simply log in to the system by entering their username and password. If the customers enter valid input, the system will redirect them to the main page of the user which is for members and volunteers. However, if the input is invalid, an error message will be displayed instead and users need to re-enter the valid input for their account.

Secondly, the users can view the upcoming events and past events of Katzen Event including the event name, poster, date, time, location, and description by simply clicking on the our events menu. For the members, they can also view merchandise as they can get promotion for the katzen's merchandise for having the membership. For the volunteer, they can view their schedule based on the assigned event. They will just need to simply click on the view schedule menu on the navigation bar. Then, they will be able to click on the attend button if they have attended the event. However, if the volunteer did not click on the attend button, the button will be unavailable as it has expired.

From the system administrator's perspective, the administrator will need to login first to the system by entering their username and password. Then, the system will display the administrator dashboard. The admin has access to various features that allow them to monitor users, organize events, track donations, and generate reports. Their responsibilities include managing members, volunteers, donors, and events while ensuring that all data is accurate and up-to-date. The system keeps a record of all registered members. The admin can view and review member details, ensuring that all information is maintained. Other than that, the administrator will also be able to handle volunteers. The admin can view the list of registered volunteers and has the ability to assign them to different events based on availability.

Additionally, The admin is also responsible for tracking donations, which are important for funding activities. The system provides a clear summary of all donations made by users, allowing the admin to monitor the total amount collected and observe donation trends. Other than that, the admin can view donor details such as name, email, and phone number. This makes it easier to keep track of contributors, and communicate with them when needed.

Next, the admin will be able to create new events by entering relevant details such as the event name, date, time, address, type, description, poster, and platform. Once an event is created, it becomes visible to both the admin and the users. Additionally, the admin has the ability to edit event details to ensure that all information remains accurate and up-to-date. If an event is no longer needed, the admin can remove it from the system by deleting it. all the changes made on the admin's page on the event list will be updated on the user's page so the volunteers can keep track of the upcoming events.

Beyond managing users, donations, and events, the administrator can also generate reports based on different aspects of the system. Whether it is tracking new membership and volunteers, analyzing donation of the month, or reviewing event participation, the admin can choose which data to include in the reports. After choosing which data to be generated, the system will display a summary of the chosen data so the administrator can view and keep the record of the reports.

5.2 Screen Images

User

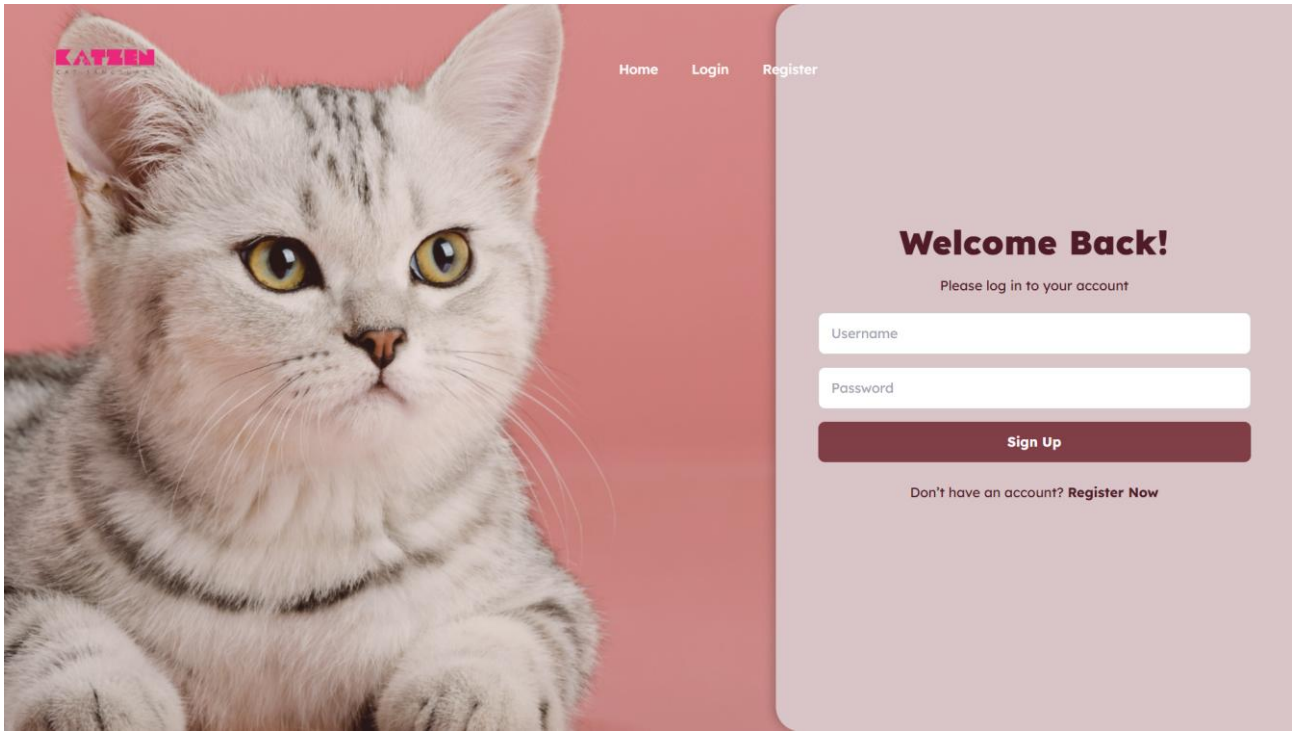


Figure 5.2.1 Login Page

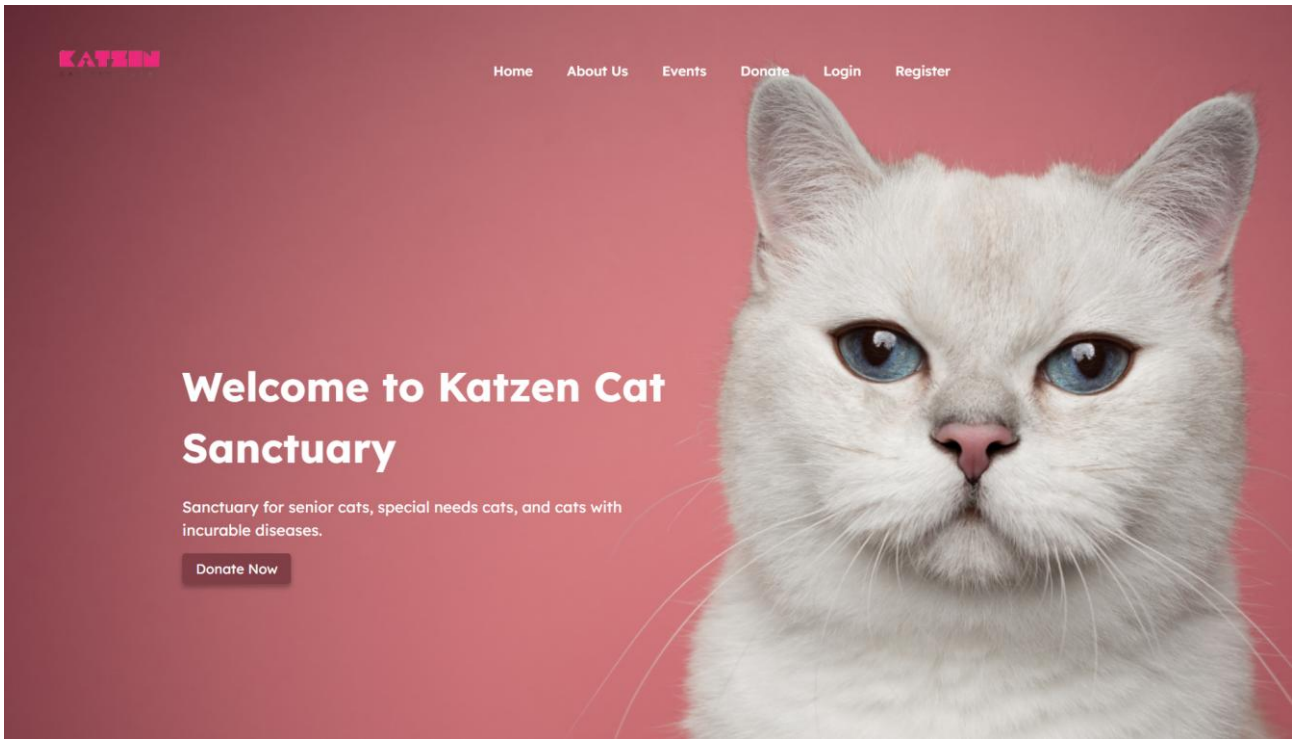


Figure 5.2.2 Home Page

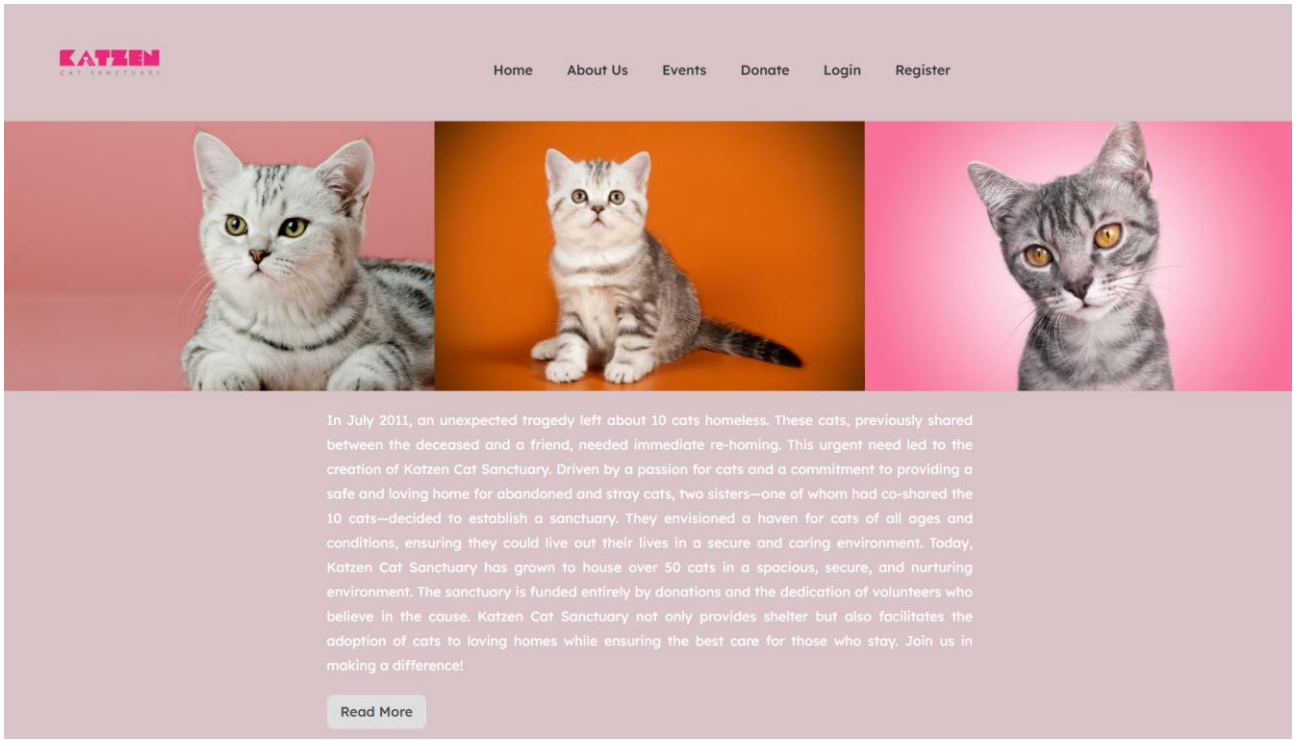
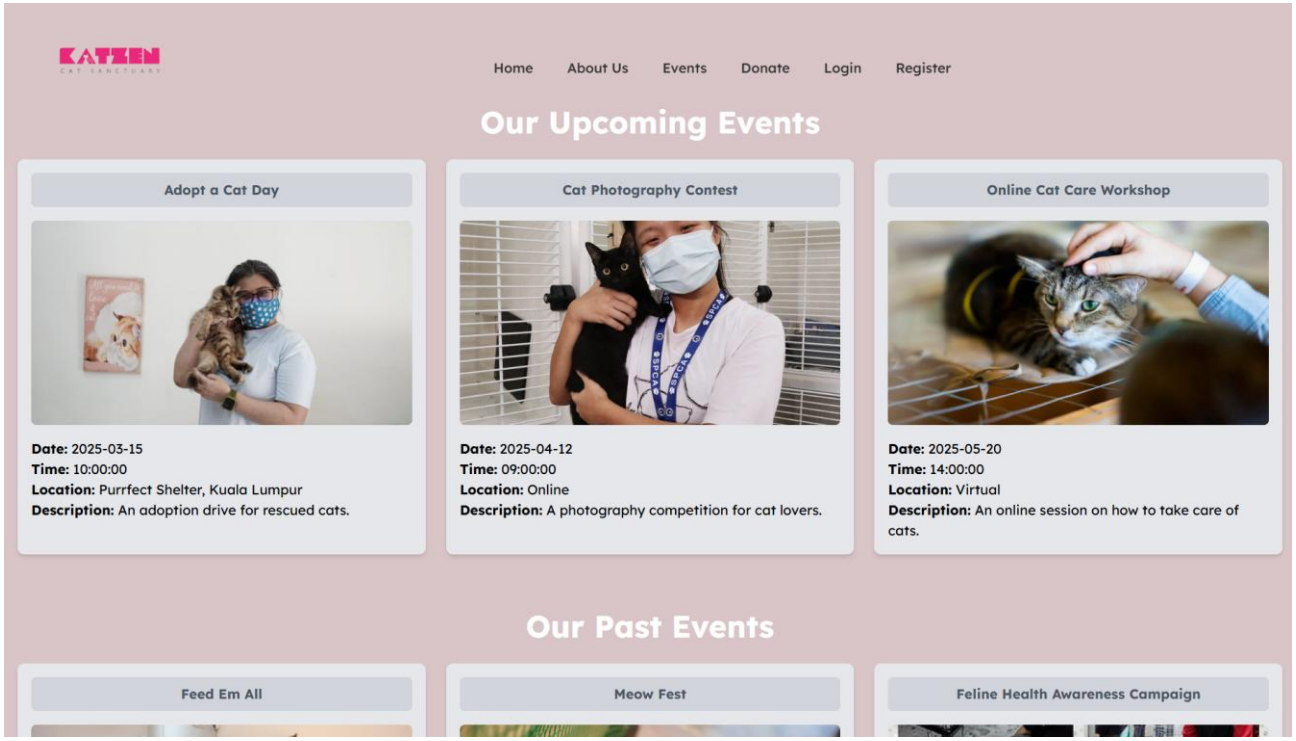


Figure 5.2.3 About Us Page



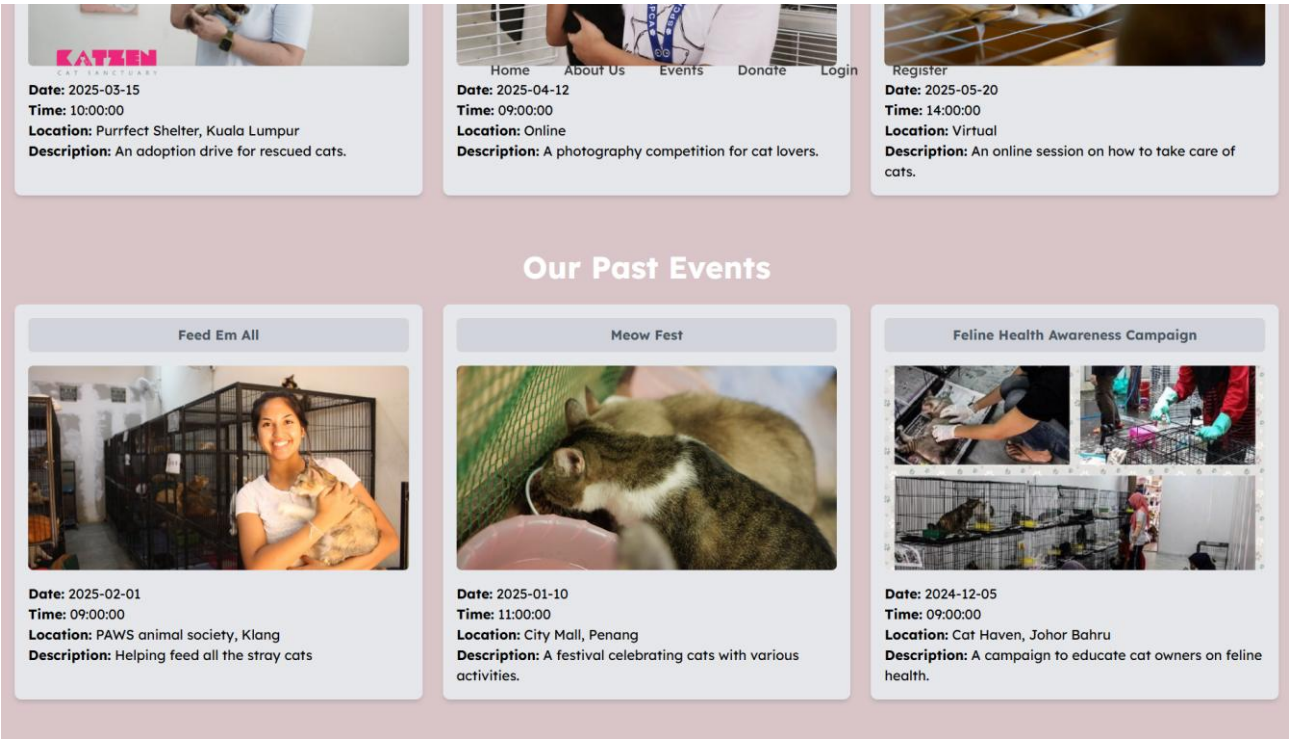
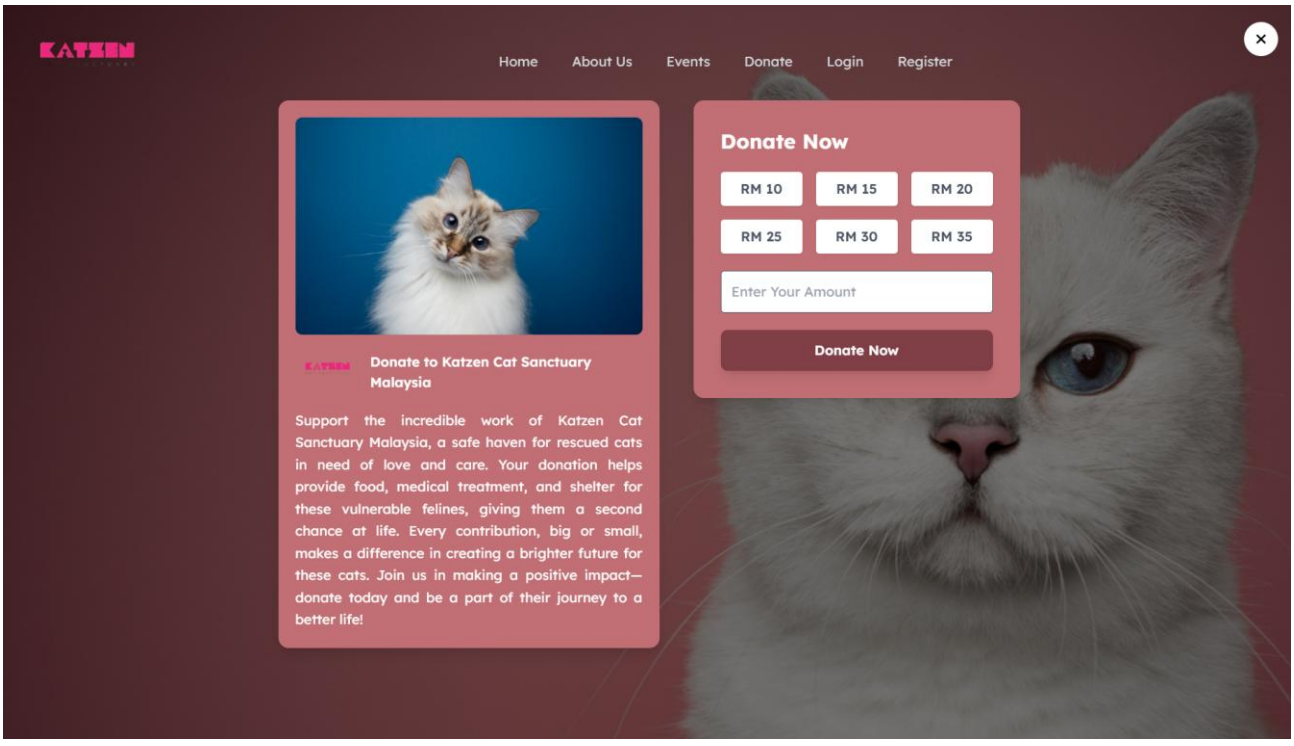


Figure 5.2.4 Our Events Page



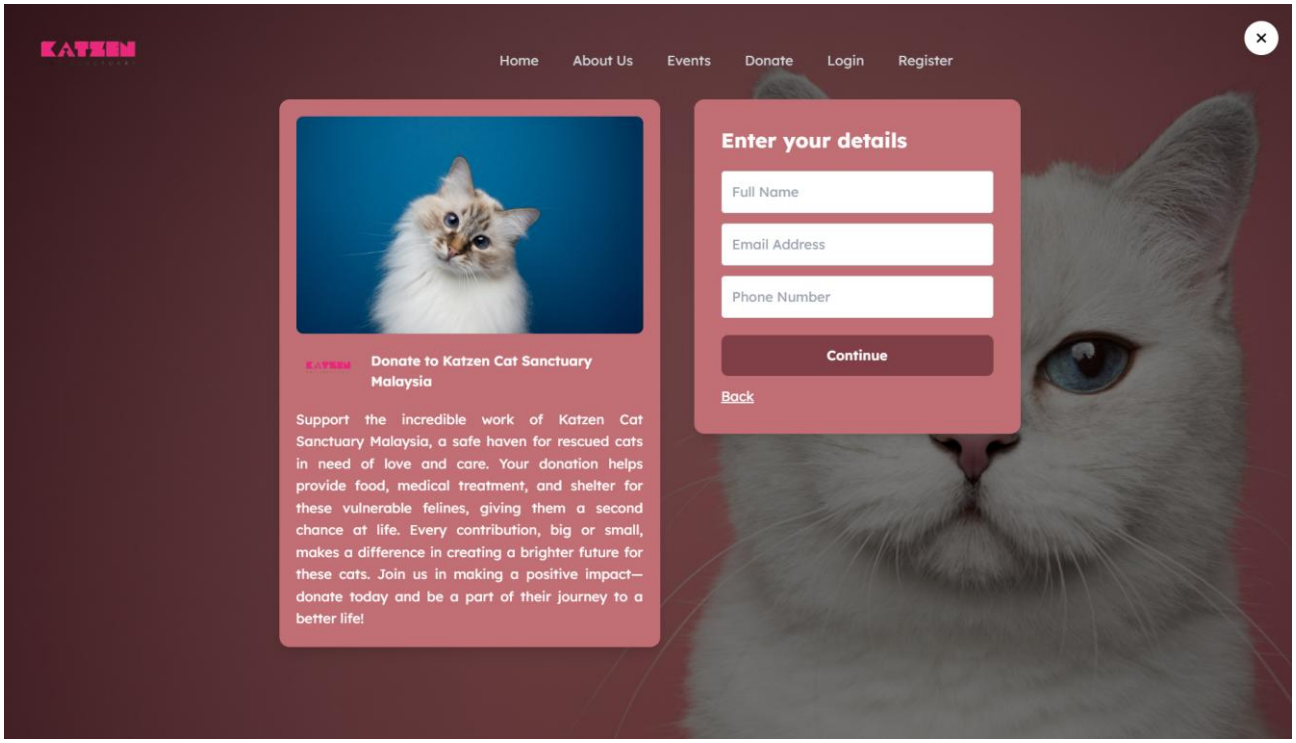


Figure 5.2.5 Donate Page

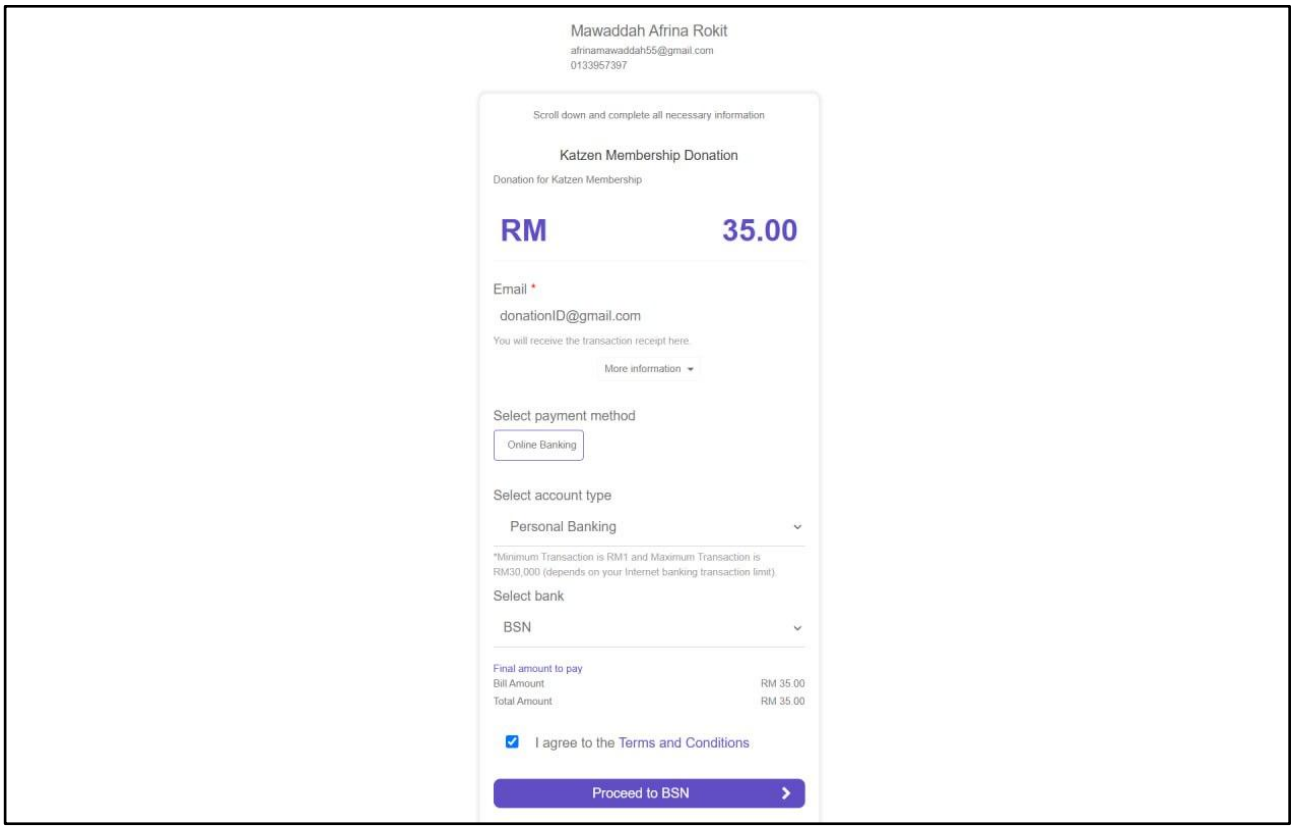


Figure 5.2.6 Payment Method Page

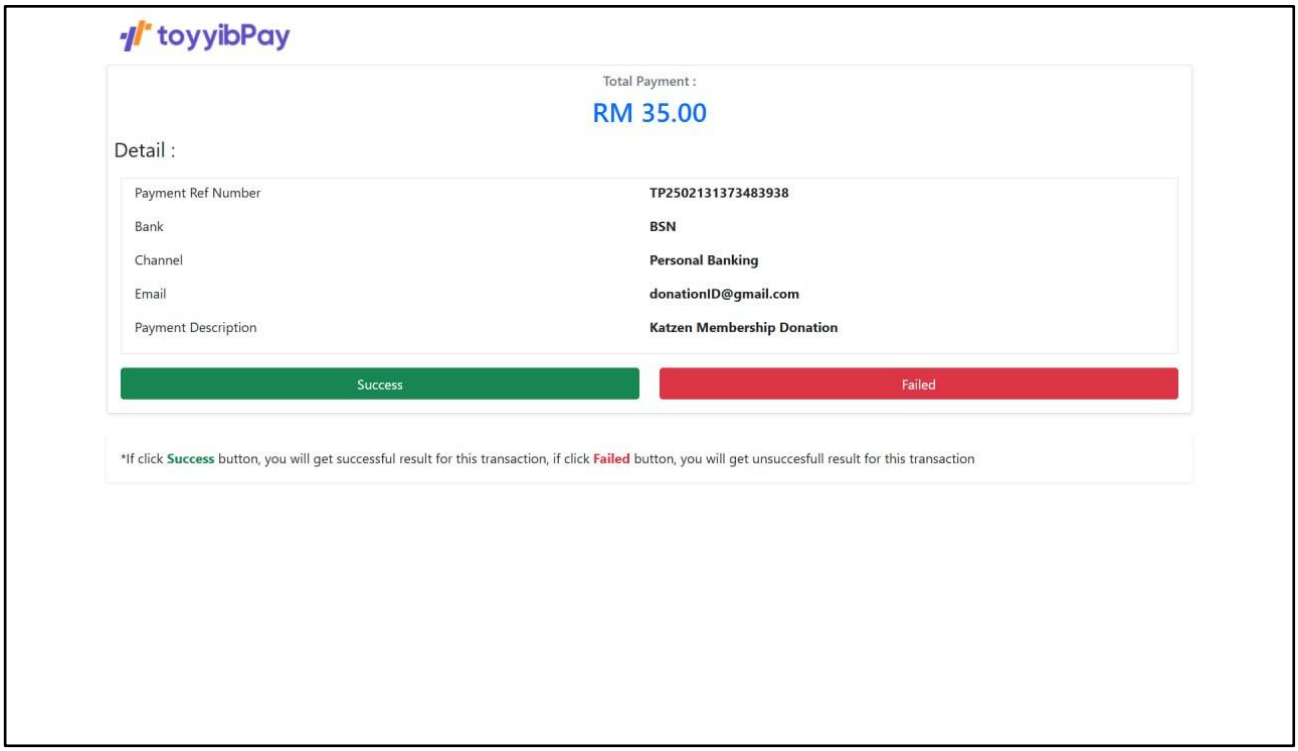


Figure 5.2.7 Payment Detail Page

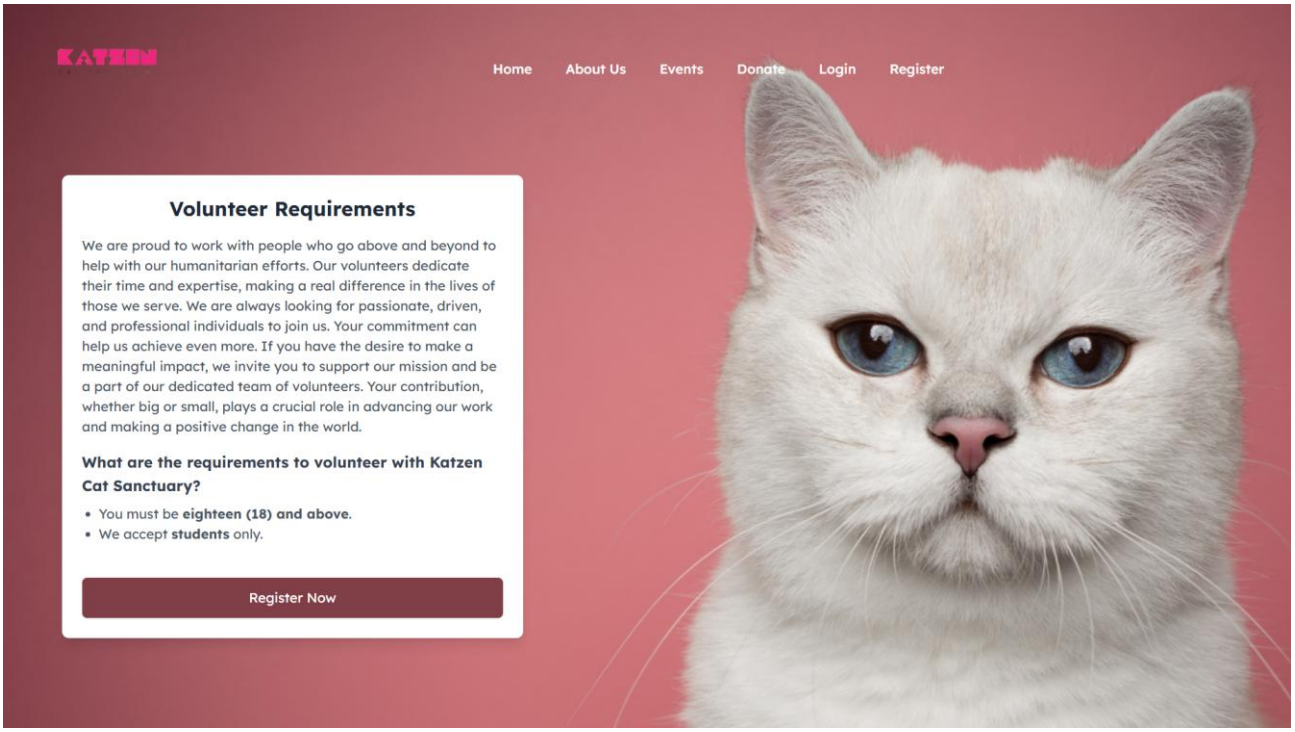


Figure 5.2.8 Volunteer Requirement Page

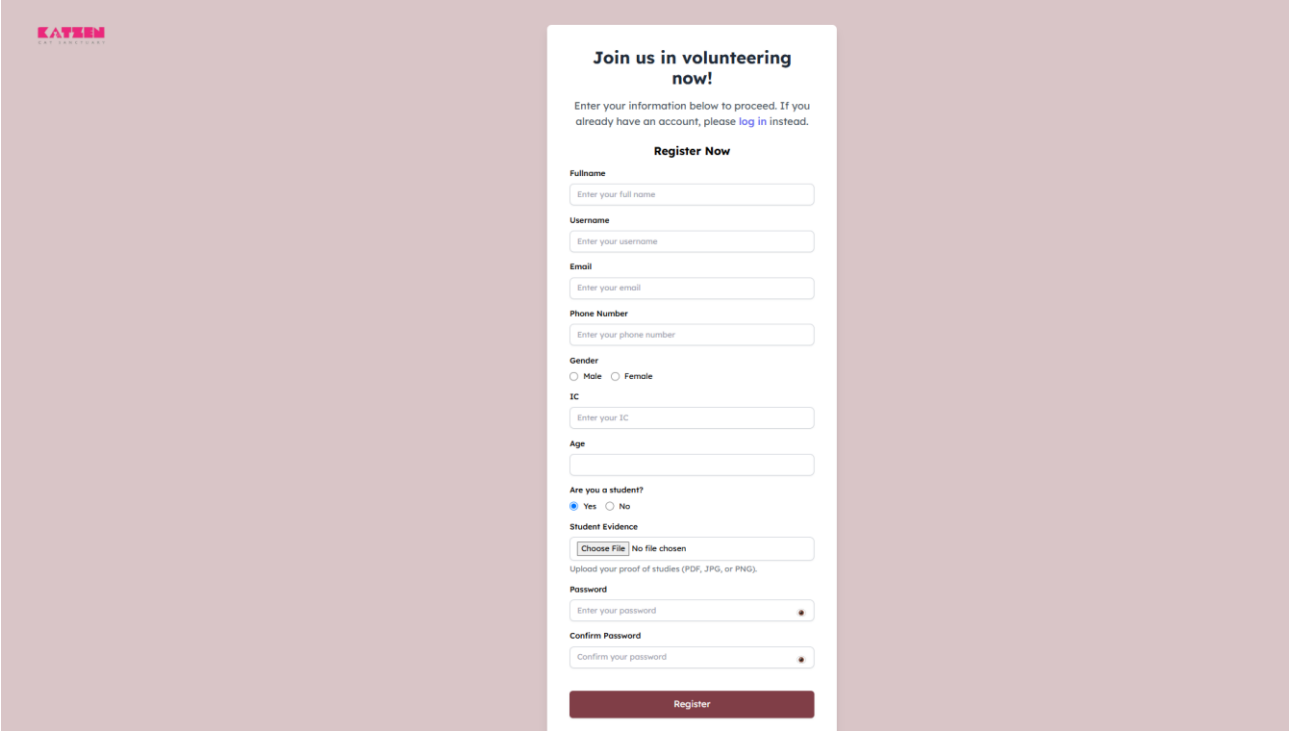


Figure 5.2.9 Volunteer Registration Form Page

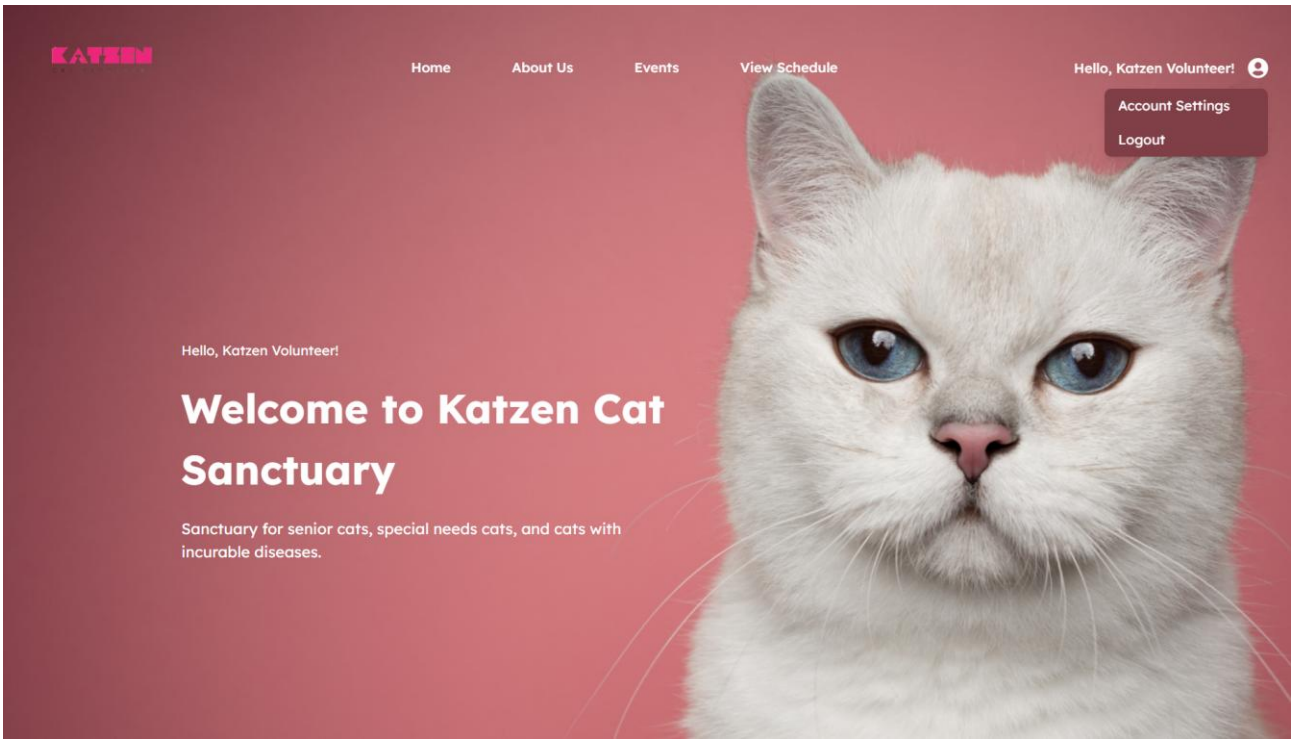


Figure 5.2.10 Volunteer Home Page

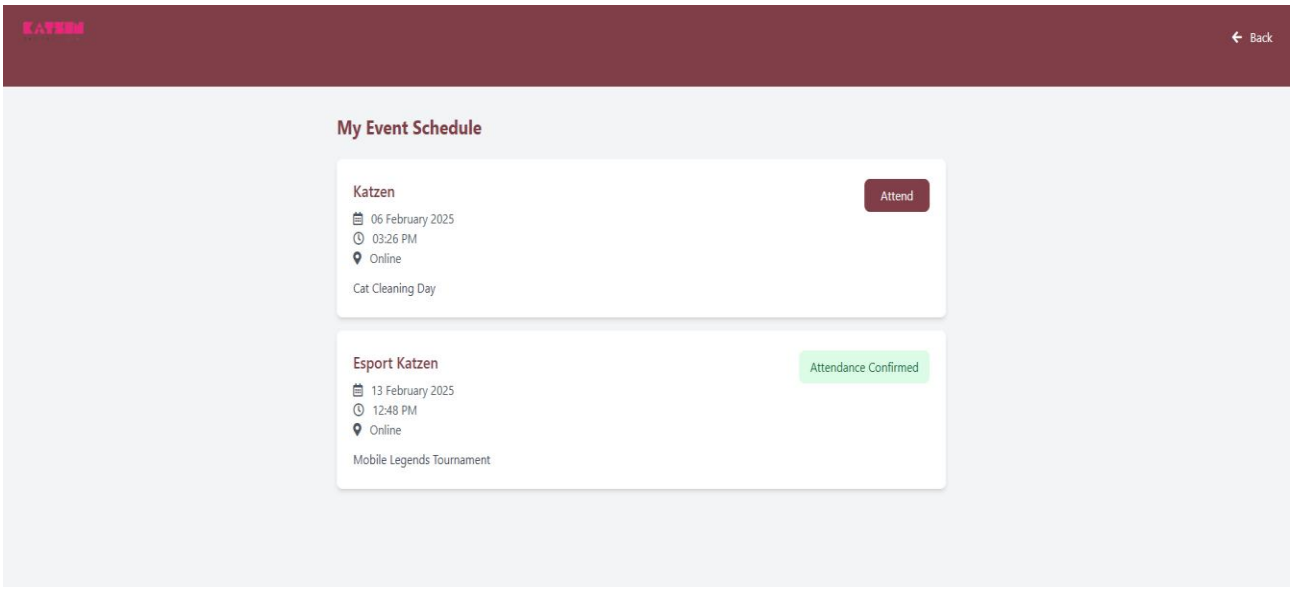


Figure 5.2.11 View Schedule Page

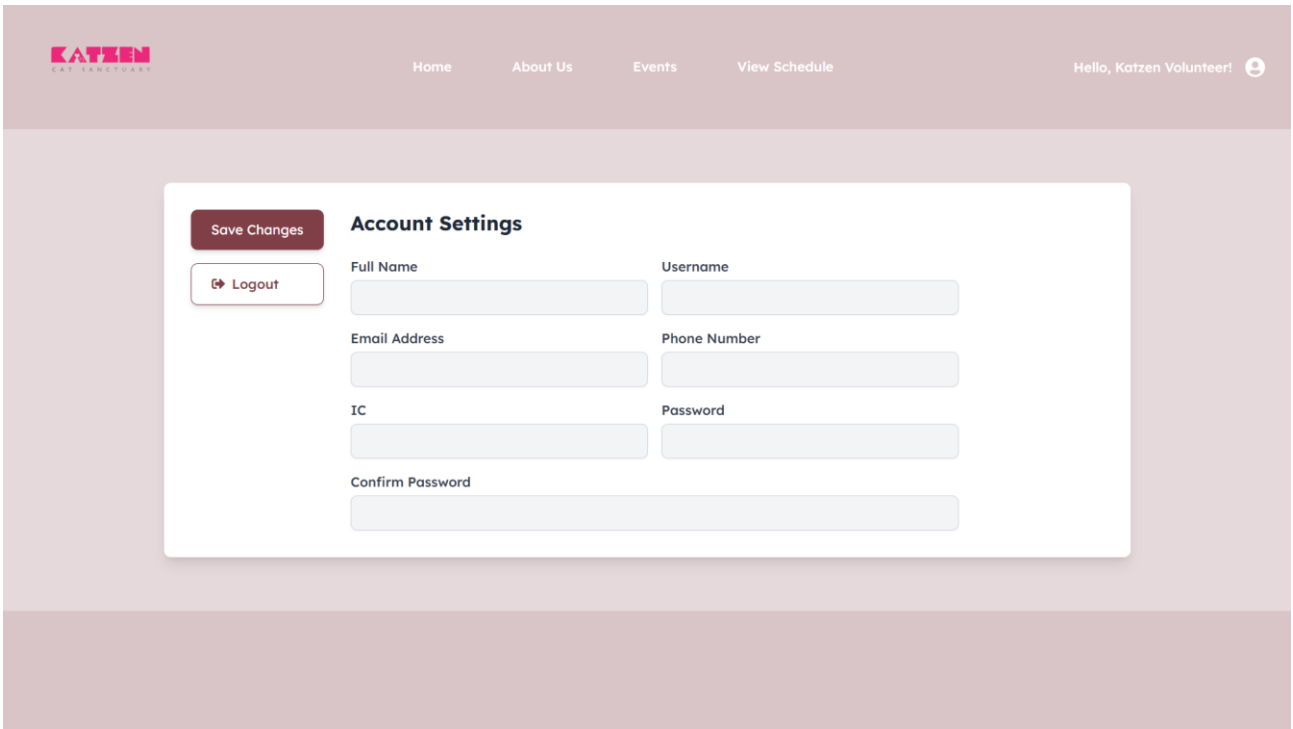


Figure 5.2.12 Edit Profile Page

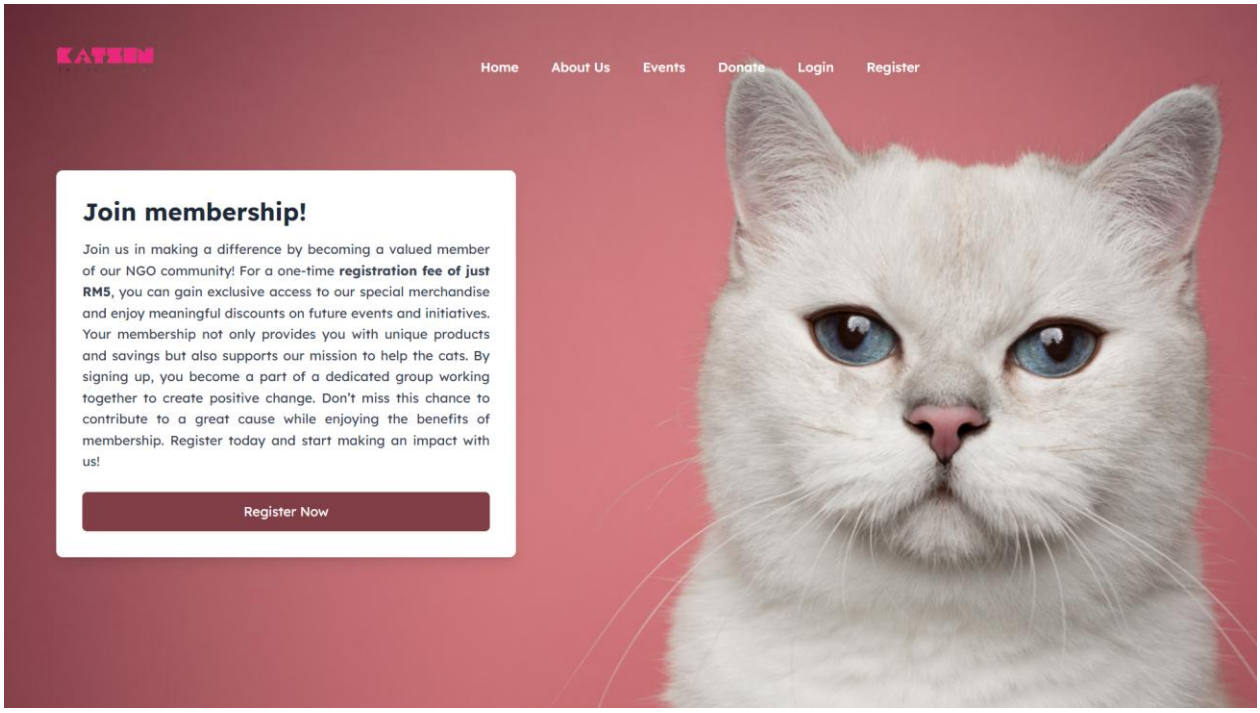


Figure 5.2.13 Member Requirements Page

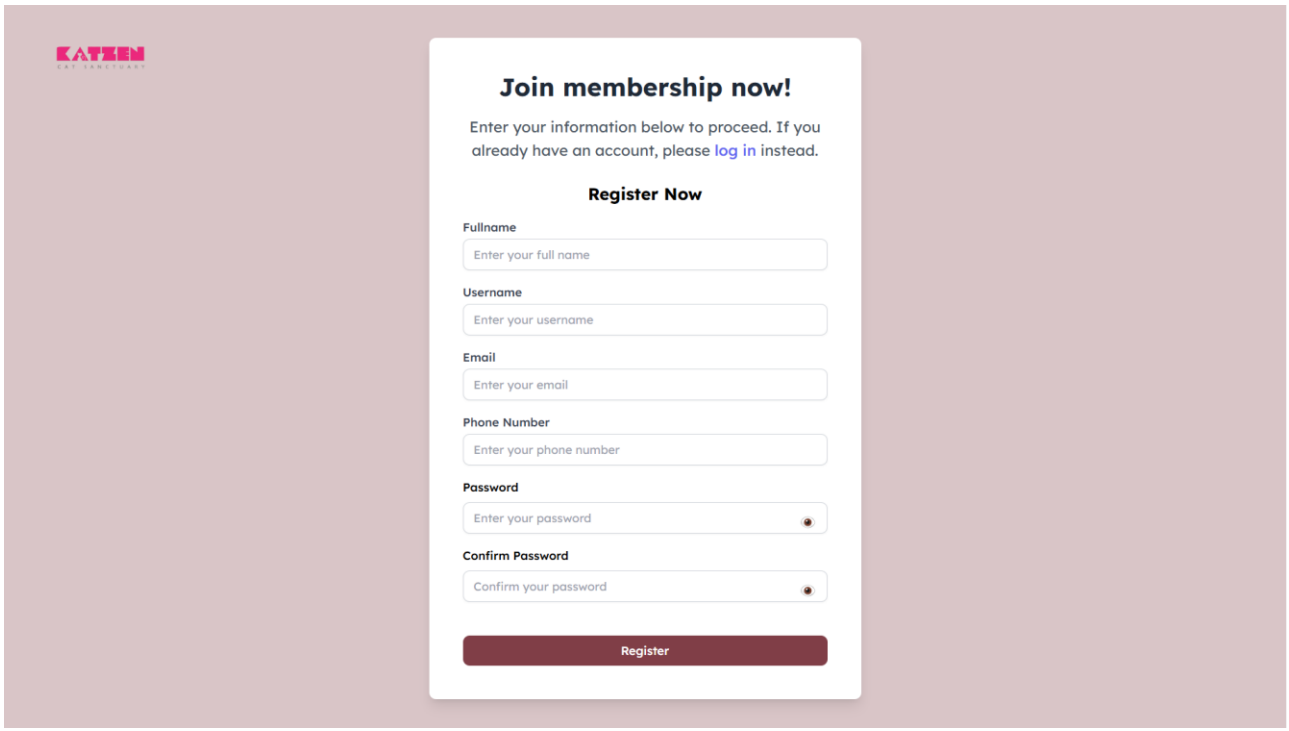


Figure 5.2.14 Member Registration Form Page

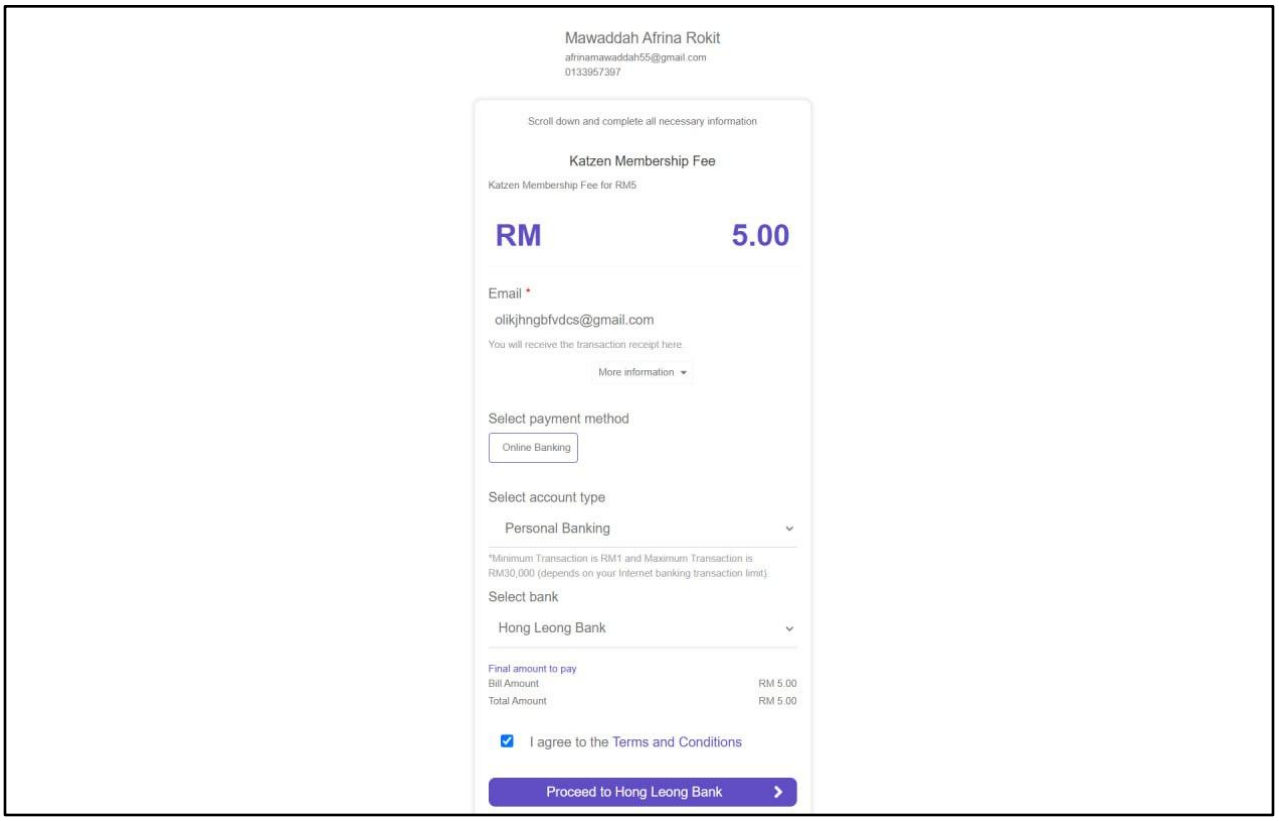


Figure 5.2.15 Payment Method Page

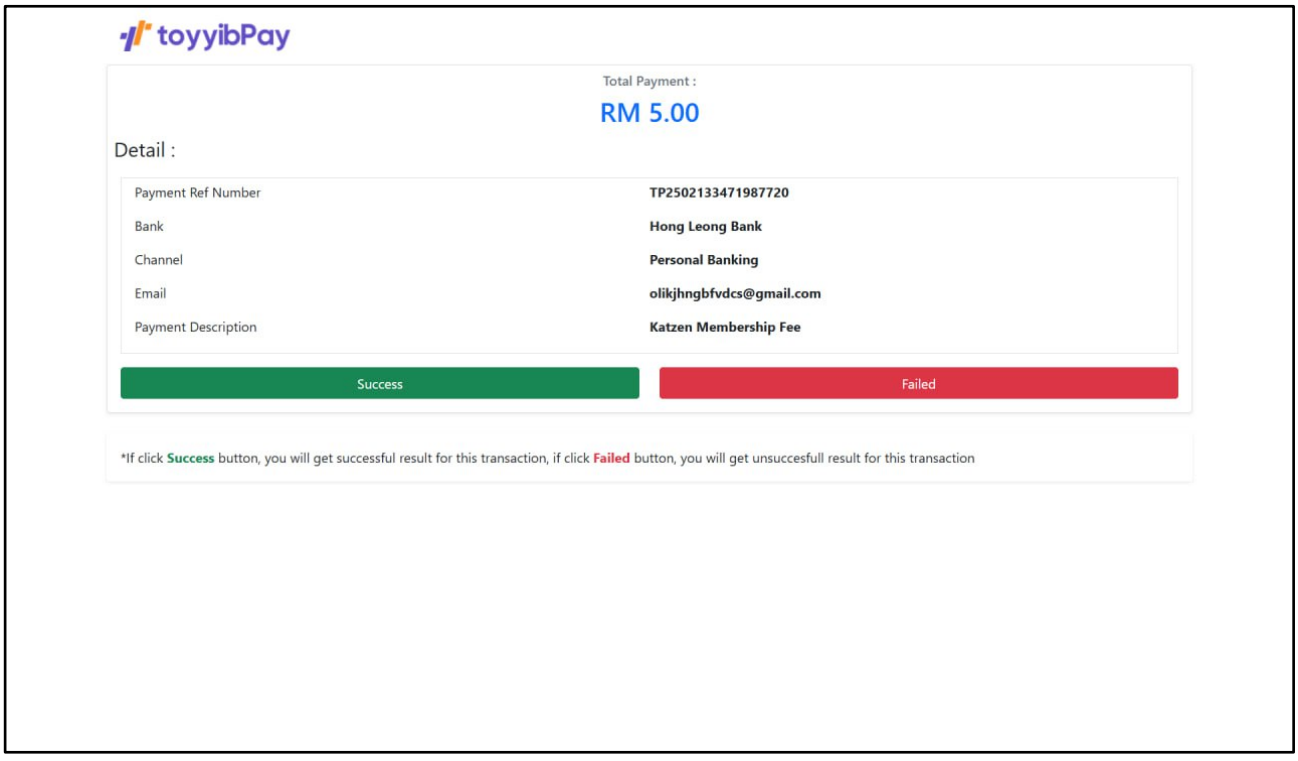


Figure 5.2.16 Payment Detail Page

Figure 5.2.17 Member Home Page

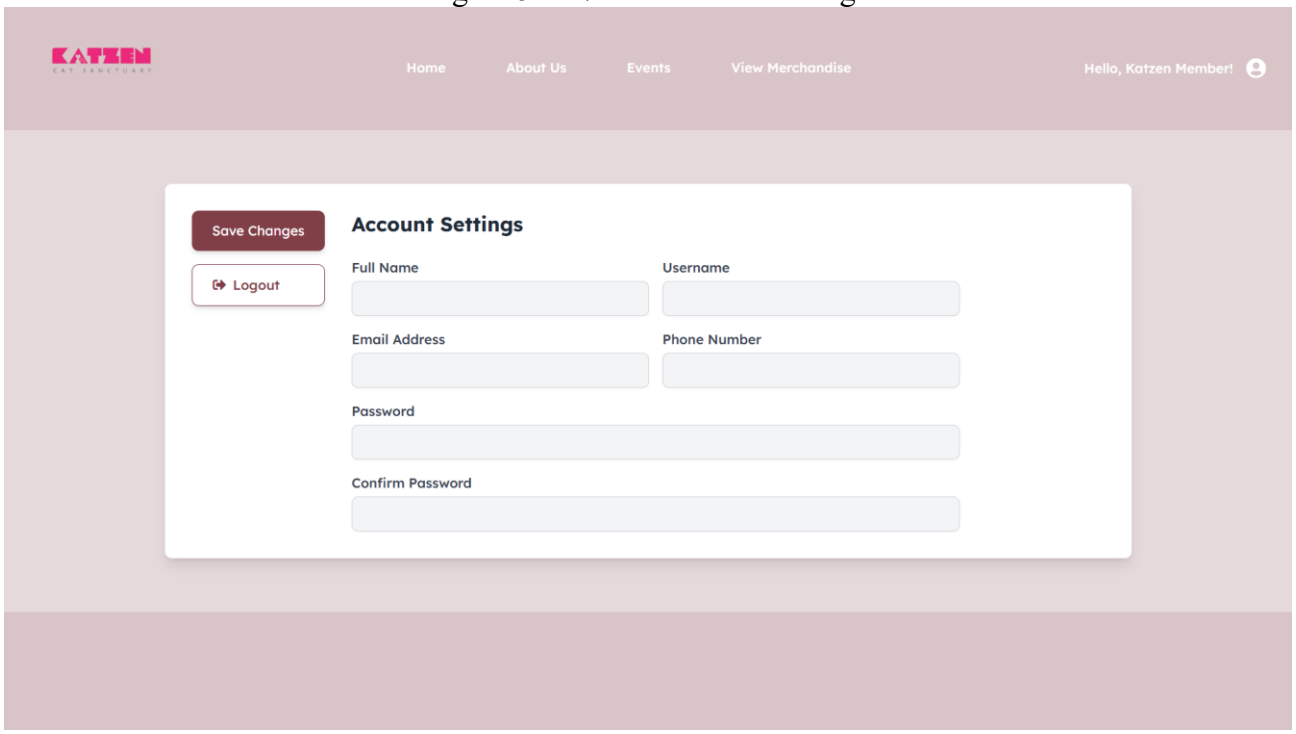


Figure 5.2.18 Edit Profile Page

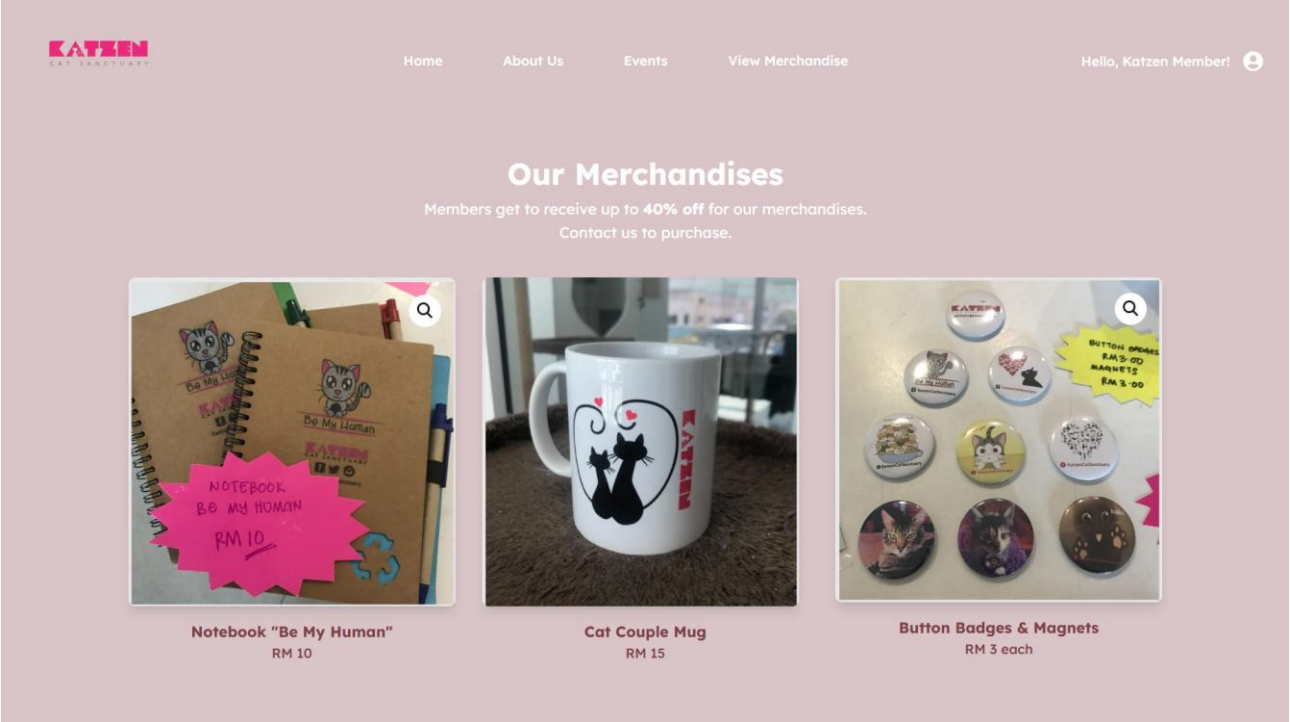


Figure 5.2.19 View Schedule Page

Admin

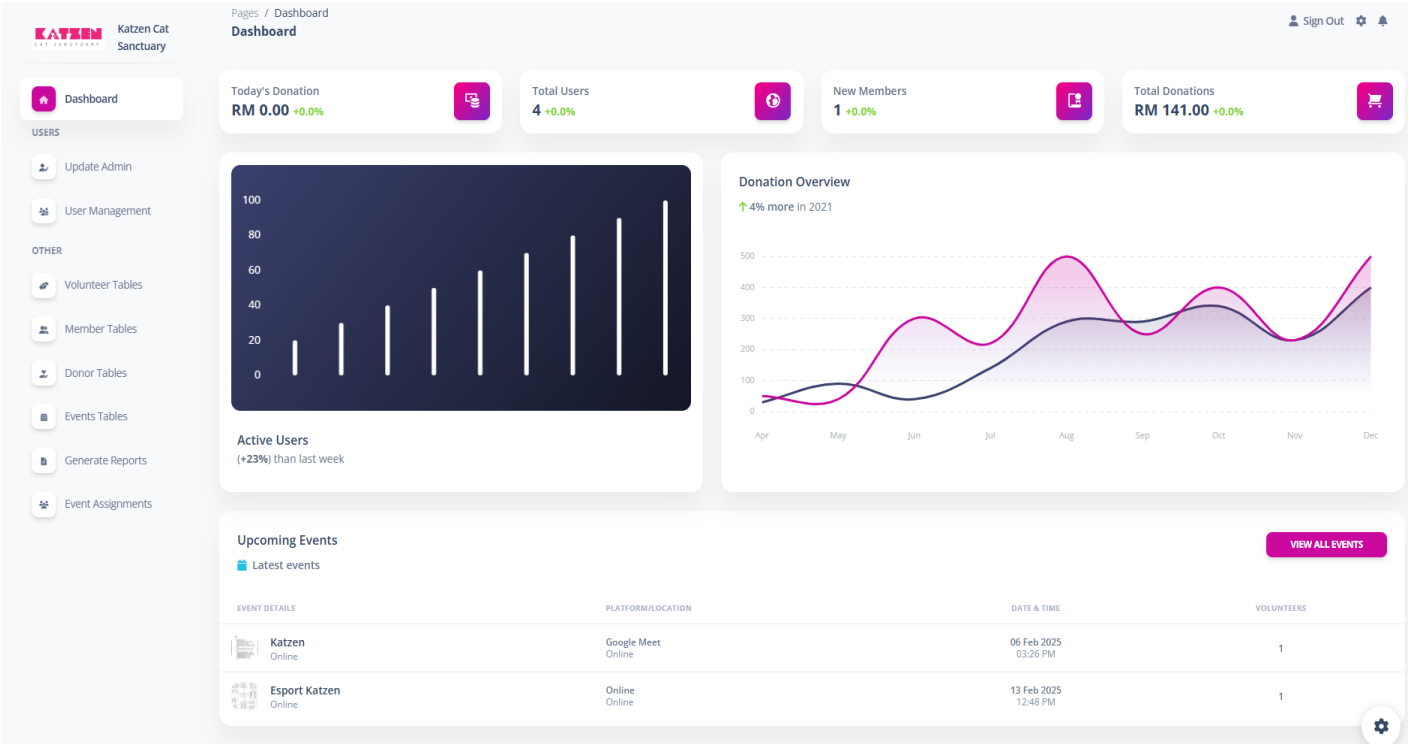


Figure 5.2.20 Admin Dashboard Page

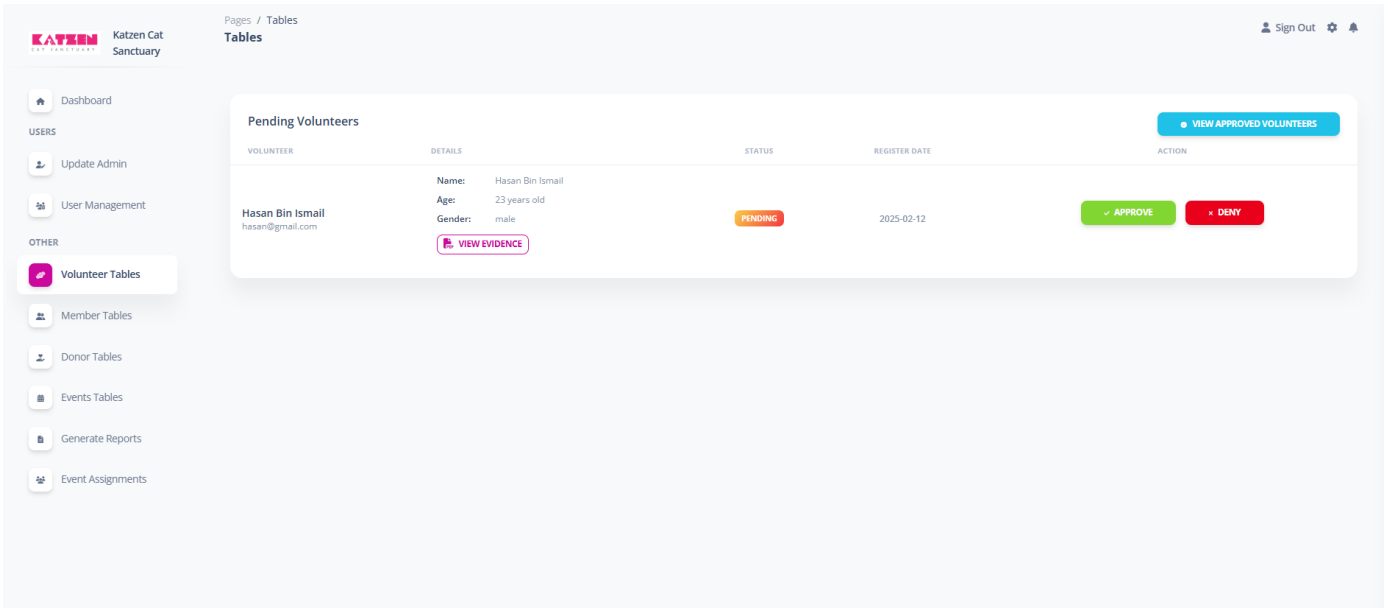


Figure 5.2.21 Pending Volunteers

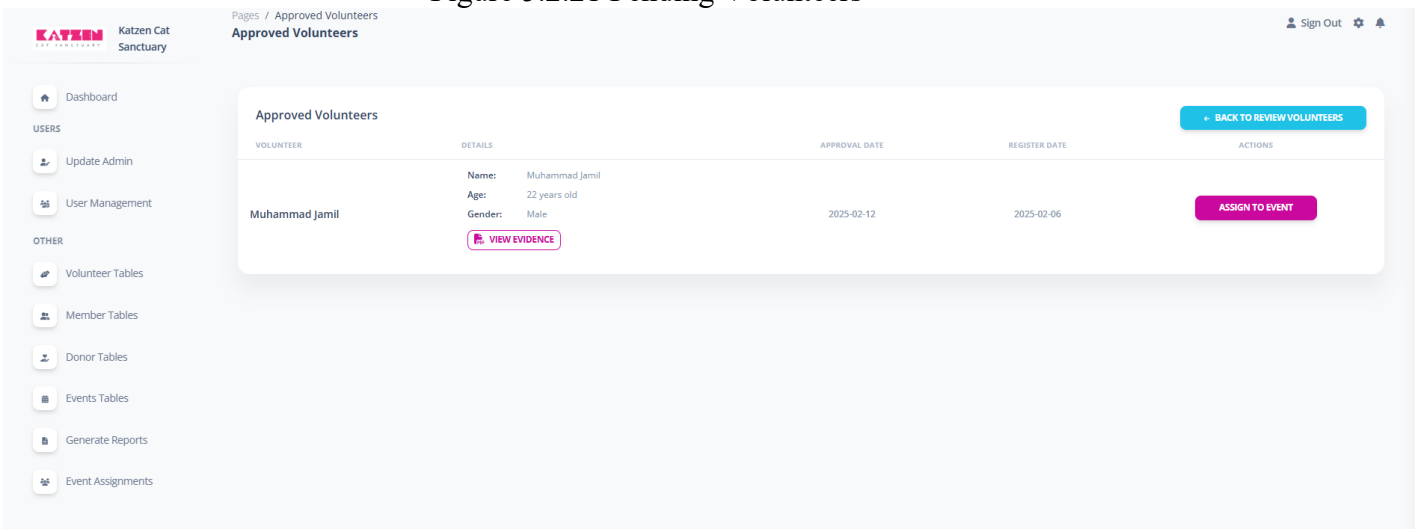


Figure 5.2.22 Approval Page

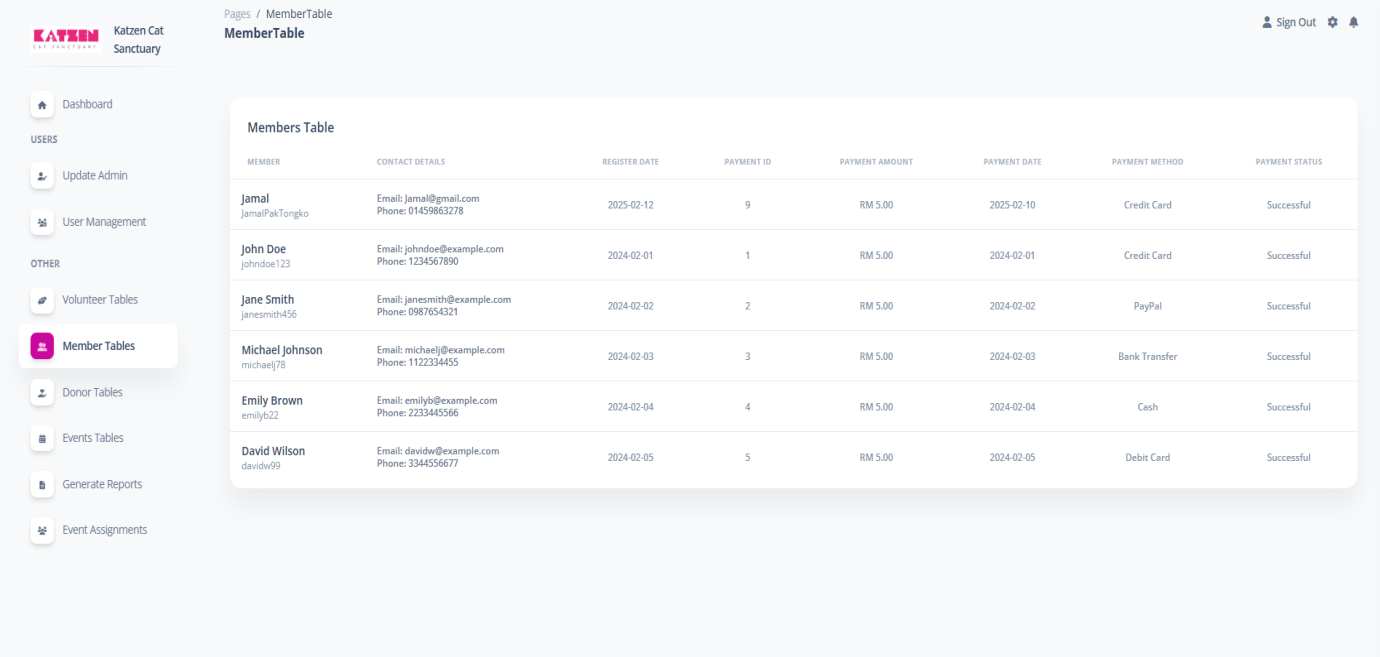


Figure 5.2.23 Member List Page

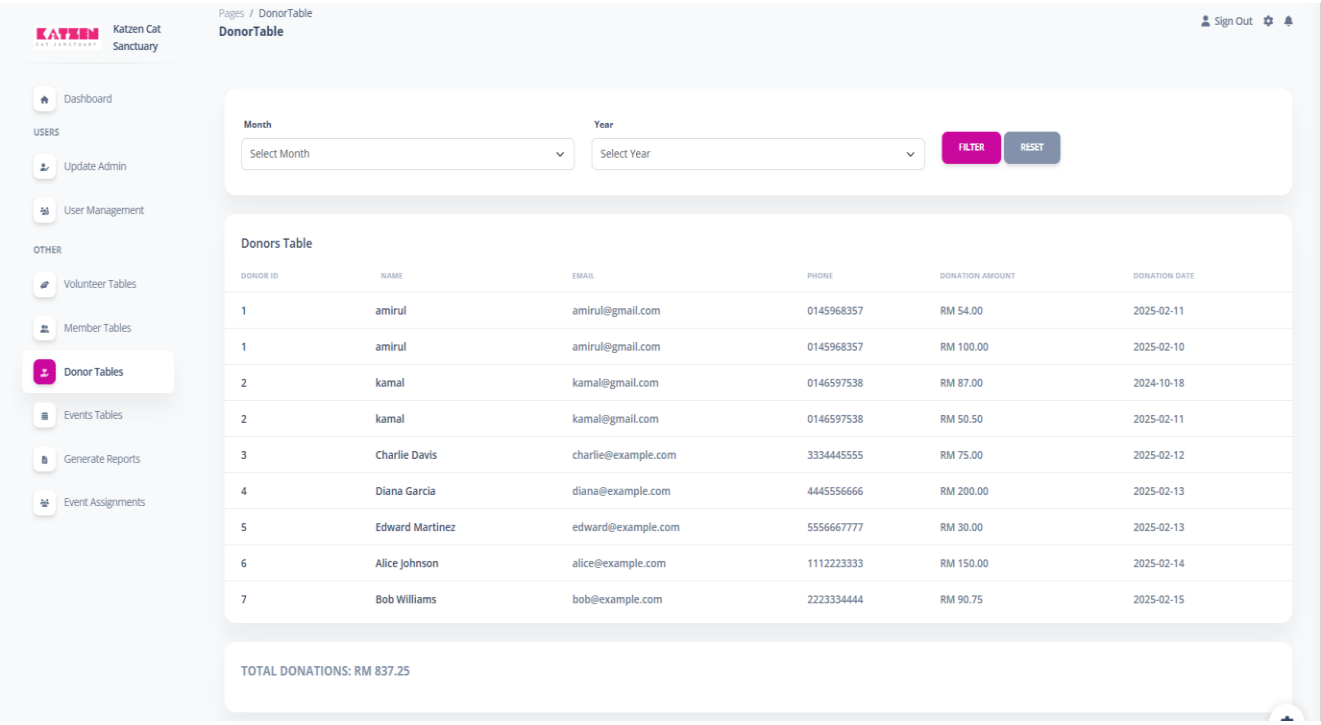


Figure 5.2.24 Donor List Page

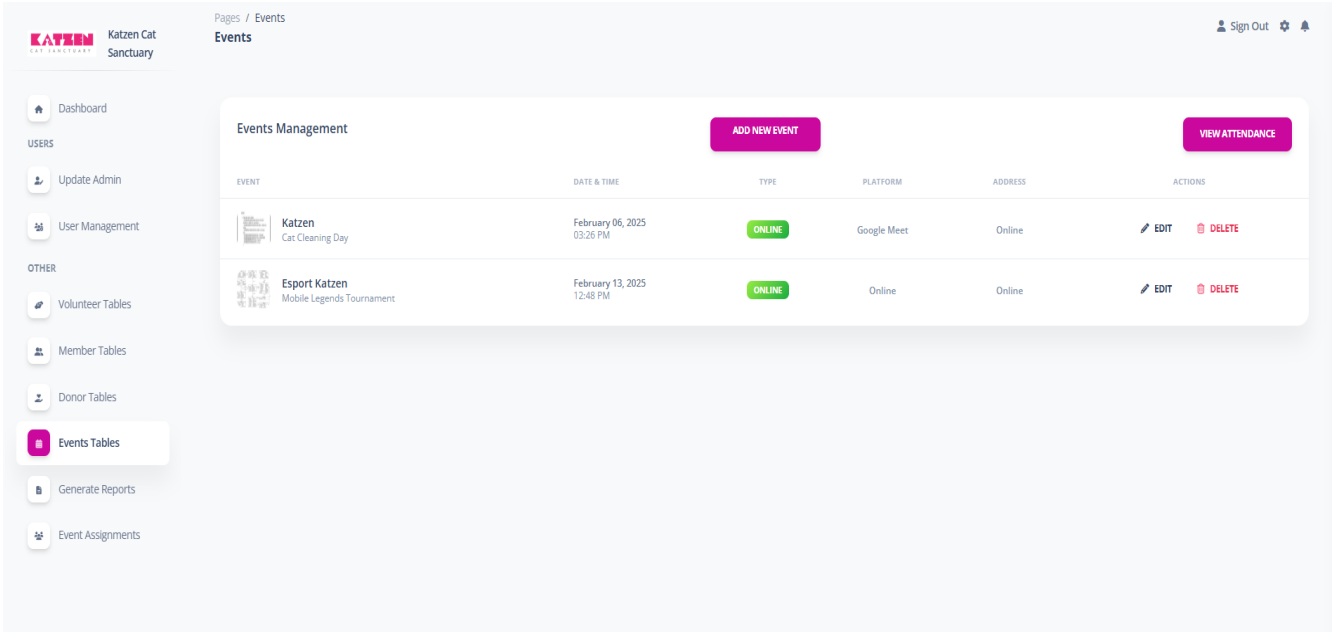


Figure 5.2.25 Event List Page

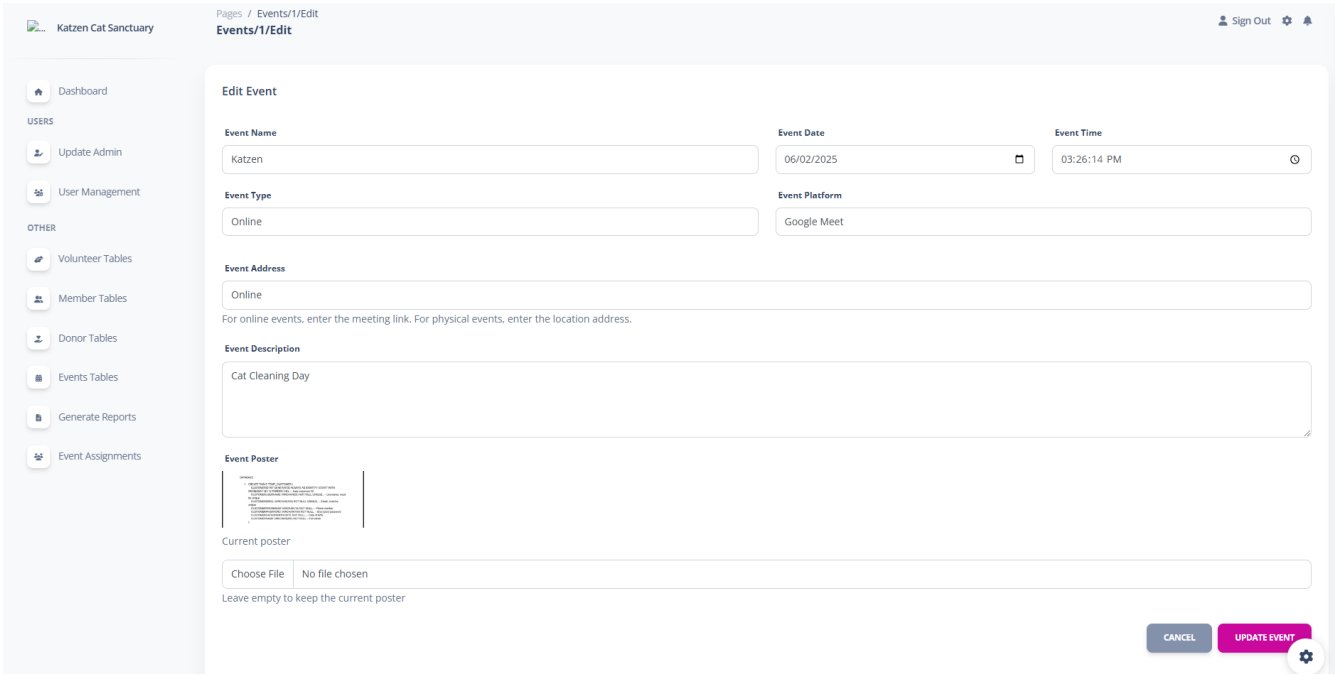


Figure 5.2.26 Edit Event Page

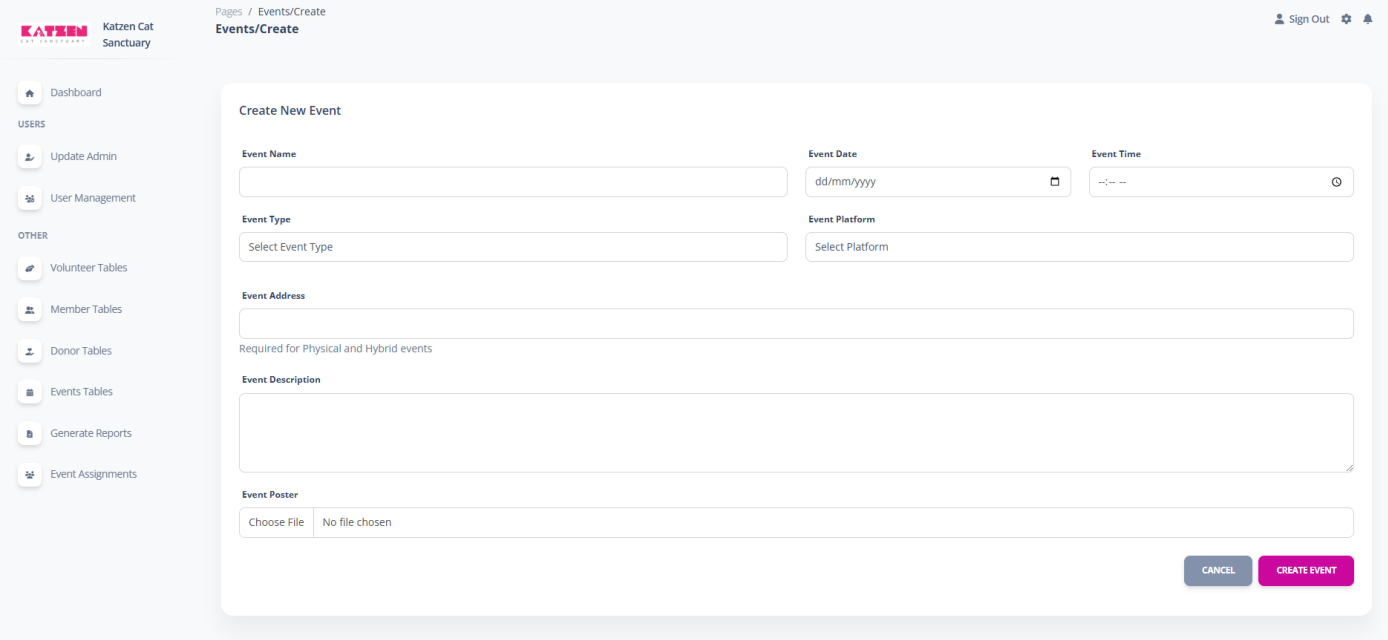


Figure 5.2.27 Create Event Page

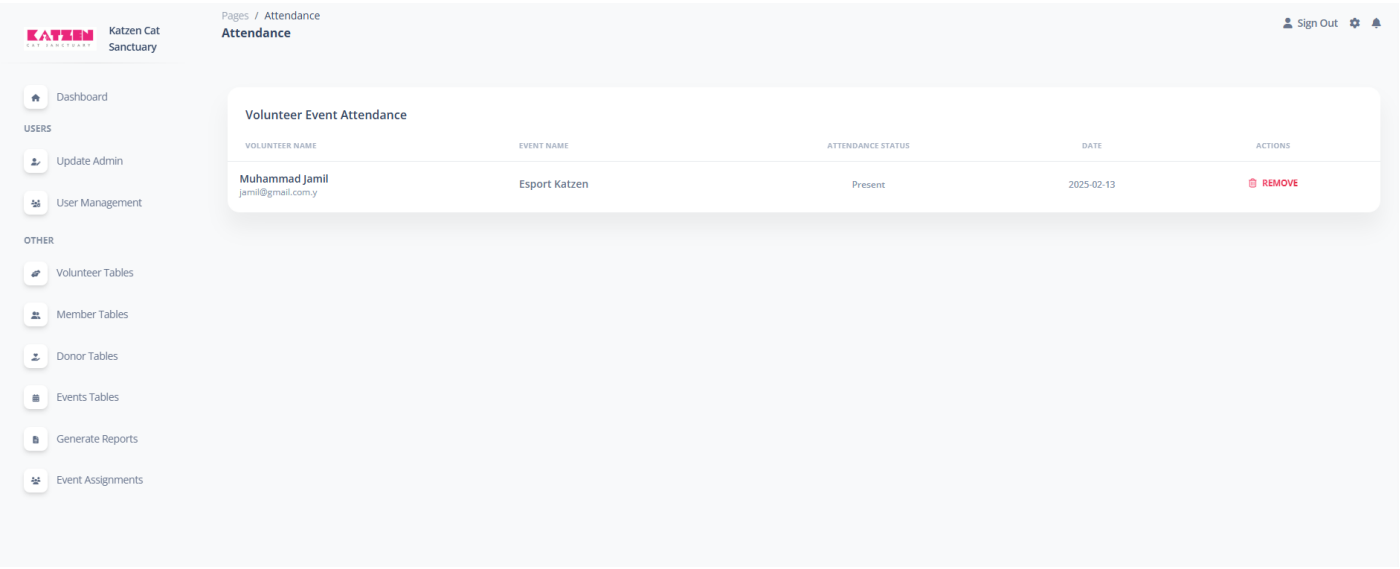


Figure 5.2.28 Attendance List Page

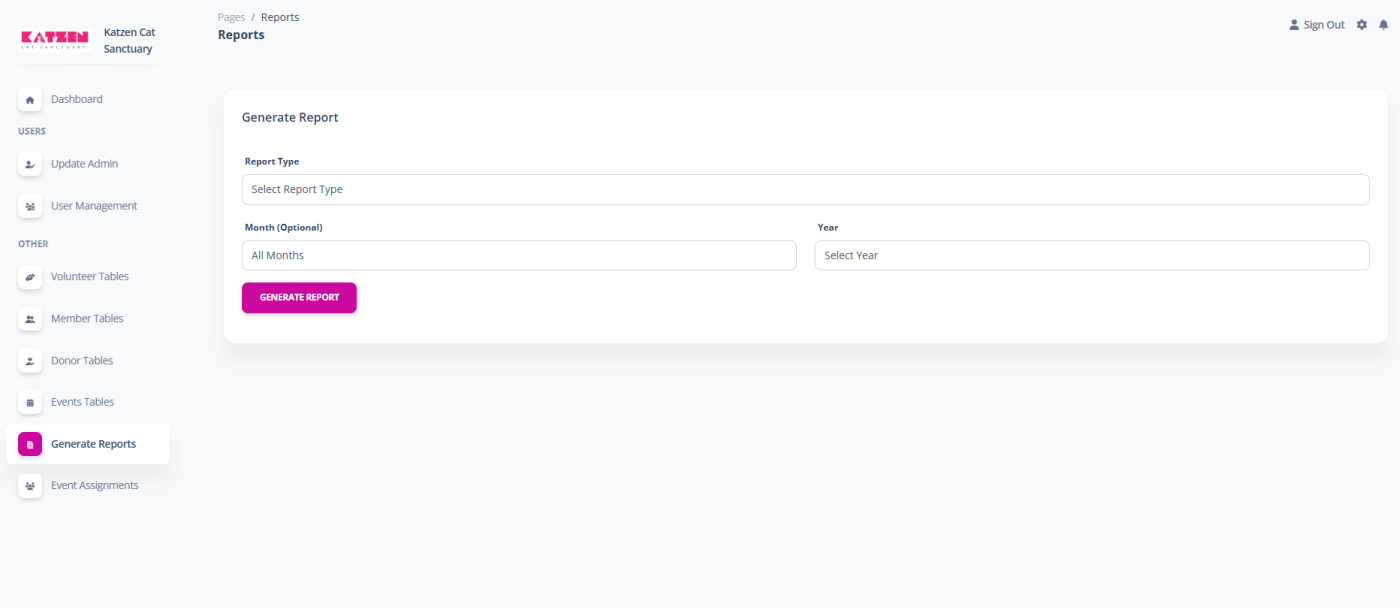


Figure 5.2.29 Generate Report Page

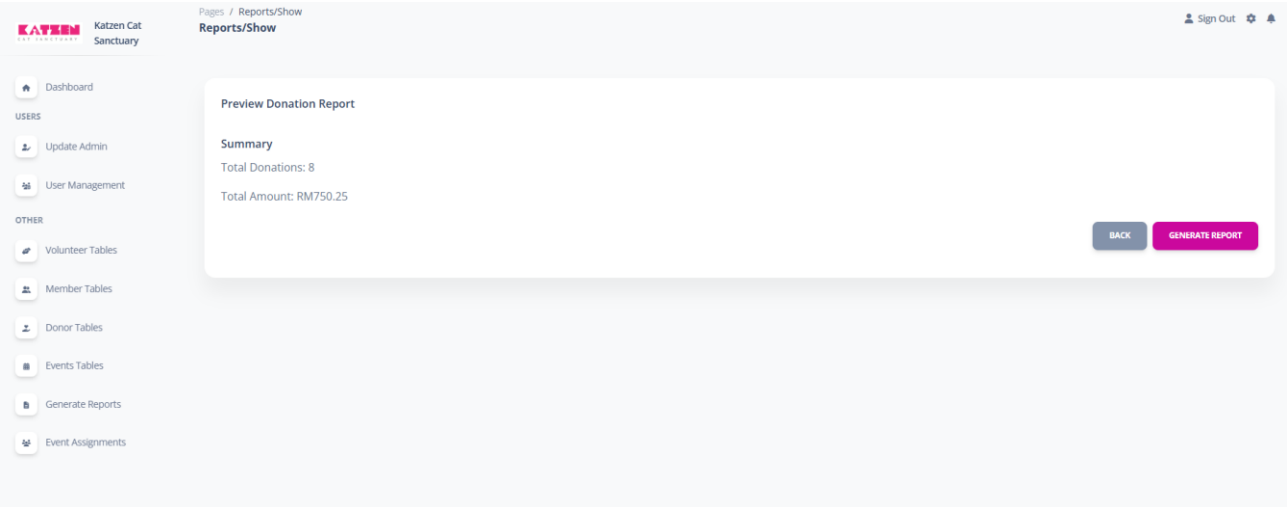


Figure 5.2.30 Show Report Page

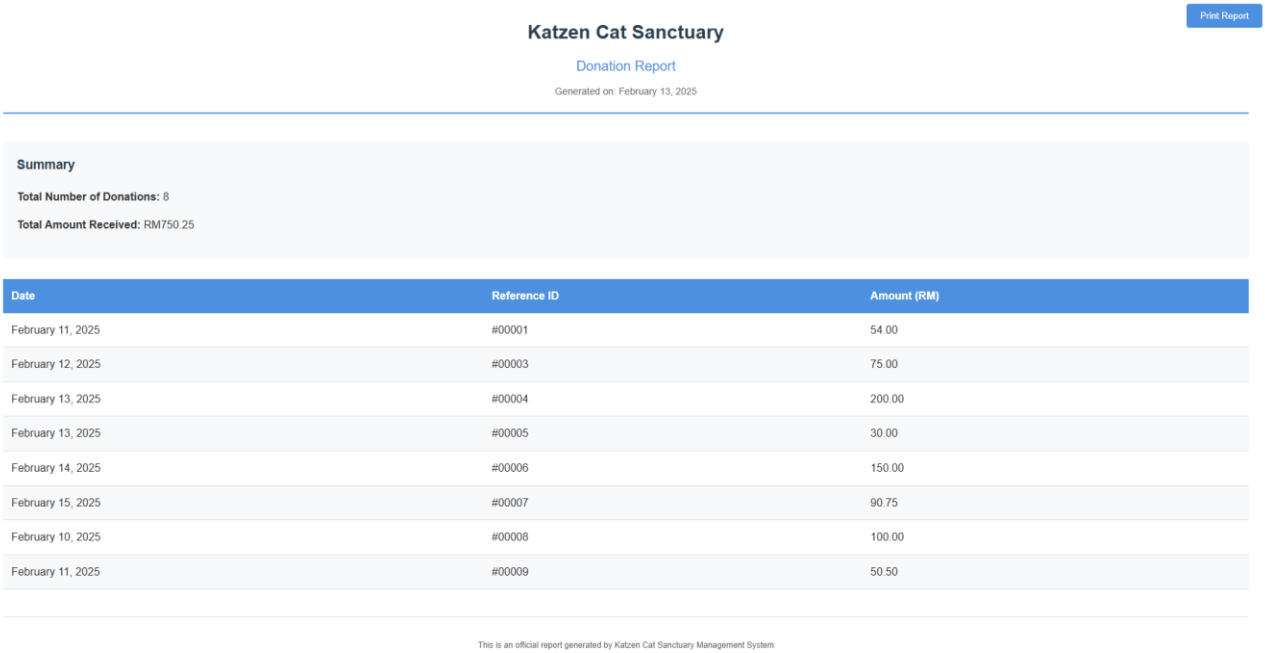


Figure 5.2.31 Report Page

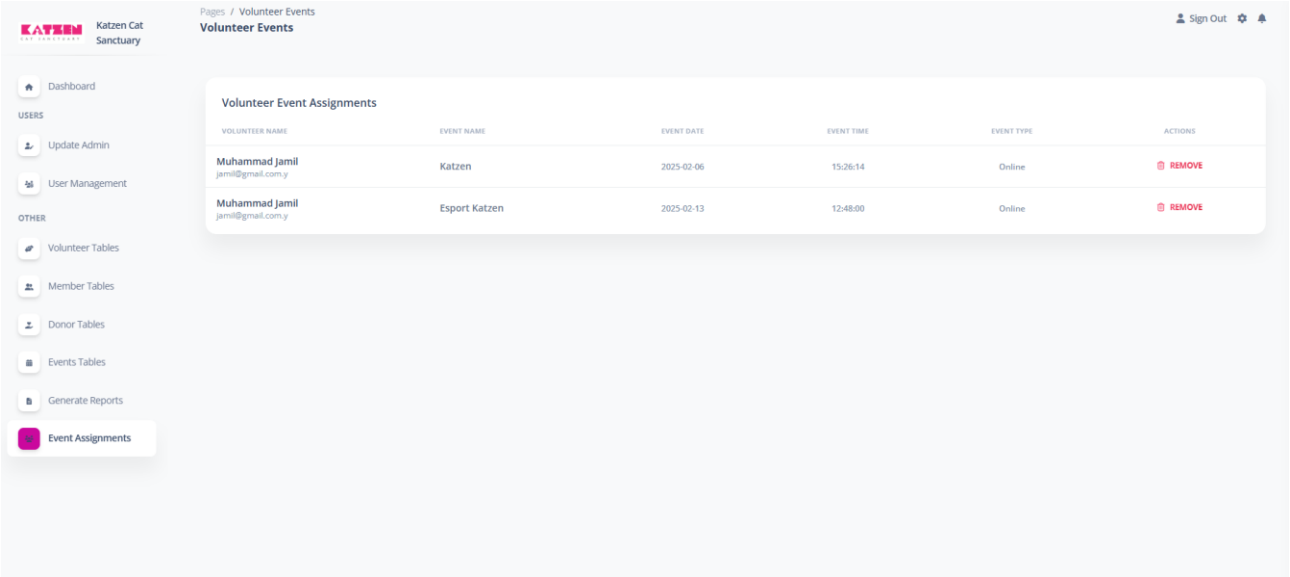


Figure 5.2.32 Volunteer Event Page

5.3 Screen Objects and Actions

Not applicable.

5.4 Report

Not applicable.

6. Traceability Requirements Matrix

	Package	(SDD_PKG_100)											(SDD_PKG_200)							(SDD_PKG_300)					
	Class	SDD_PKG_101	SDD_PKG_102	SDD_PKG_103	SDD_PKG_104	SDD_PKG_105	SDD_PKG_106	SDD_PKG_107	SDD_PKG_108	SDD_PKG_109	SDD_PKG_110	SDD_PKG_111	SDD_PKG_201	SDD_PKG_202	SDD_PKG_203	SDD_PKG_204	SDD_PKG_205	SDD_PKG_206	SDD_PKG_207	SDD_PKG_301	SDD_PKG_302	SDD_PKG_303	SDD_PKG_304	SDD_PKG_305	SDD_PKG_306
UC-100 - Make payment						/										/							/		
UC-200 - Manage profile			/				/						/		/					/		/			
UC-300 - Make registration				/	/								/		/			/		/		/			
UC-400 - View event		/						/		/				/				/		/	/	/			/
UC-500 - Manage event											/			/				/			/				/
UC-600 - Generate report												/						/	/							/
UC-700 - Make donation									/								/							/	

Figure 6.1 Traceability Requirements Matrix of Katzen Event Management System

7. Resources Estimates

1. System requirement:
    - Operating System: Windows 10
    - Processor: Intel Core i5 or equivalent
    - RAM: 4 GB or higher
    - Storage 256GB SSD or higher

2. Development Environment
    - Laravel Framework
    - Visual Studio Code (VSCode)
    - phpMyAdmin

3. Database Management
    - XAMPP
        - Version 3.3.0

4. Testing Environment
    - Browsers
        - Latest version of Chrome and Microsoft Edge.

8. Appendices

1. **Entity Relationship Diagram of Katzen Event Management System**

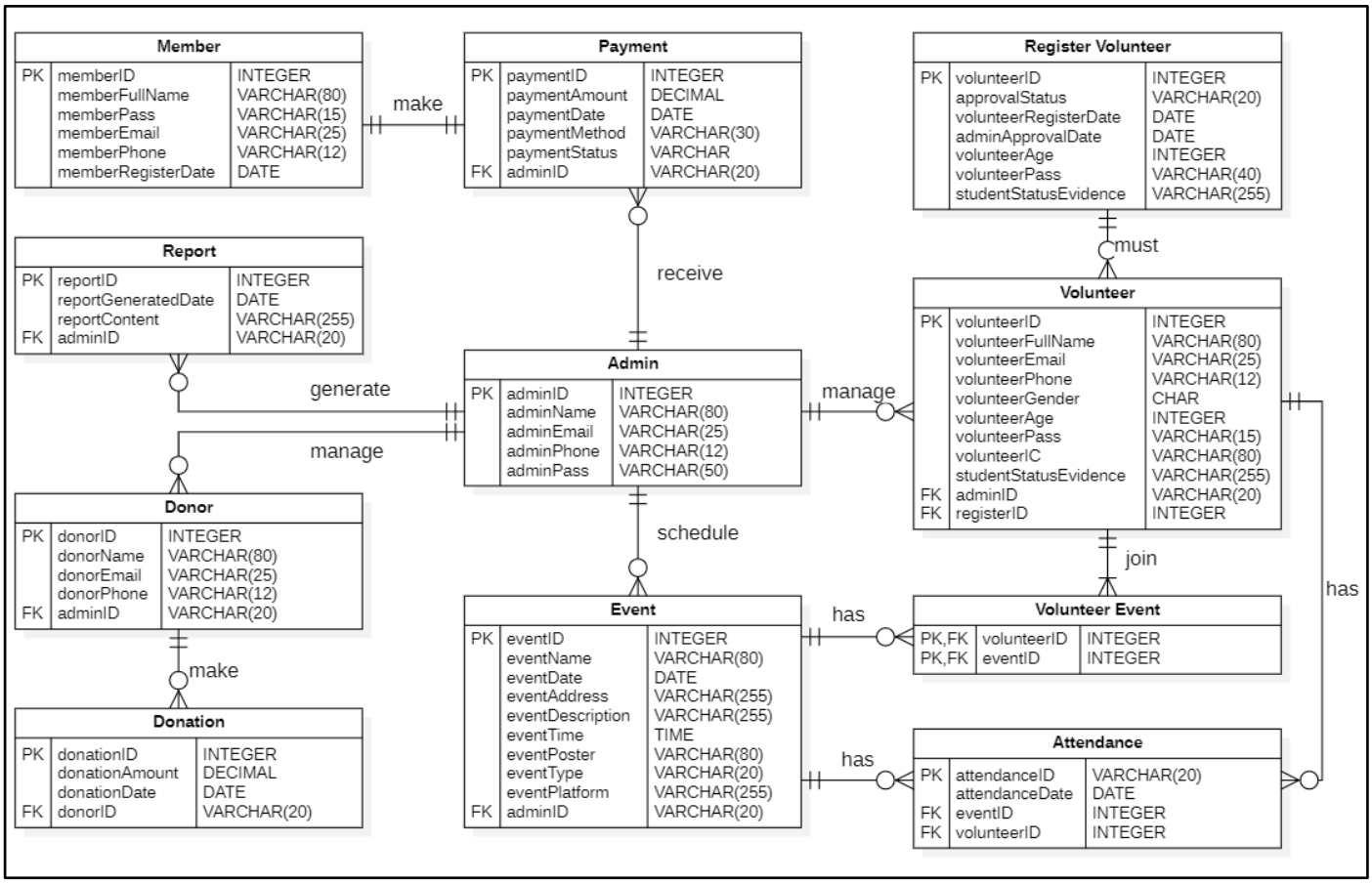
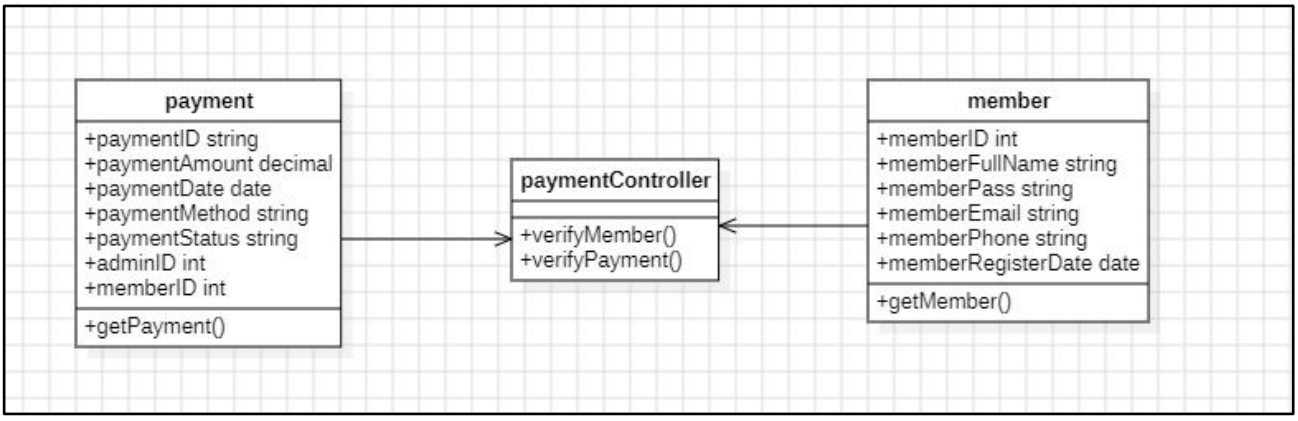


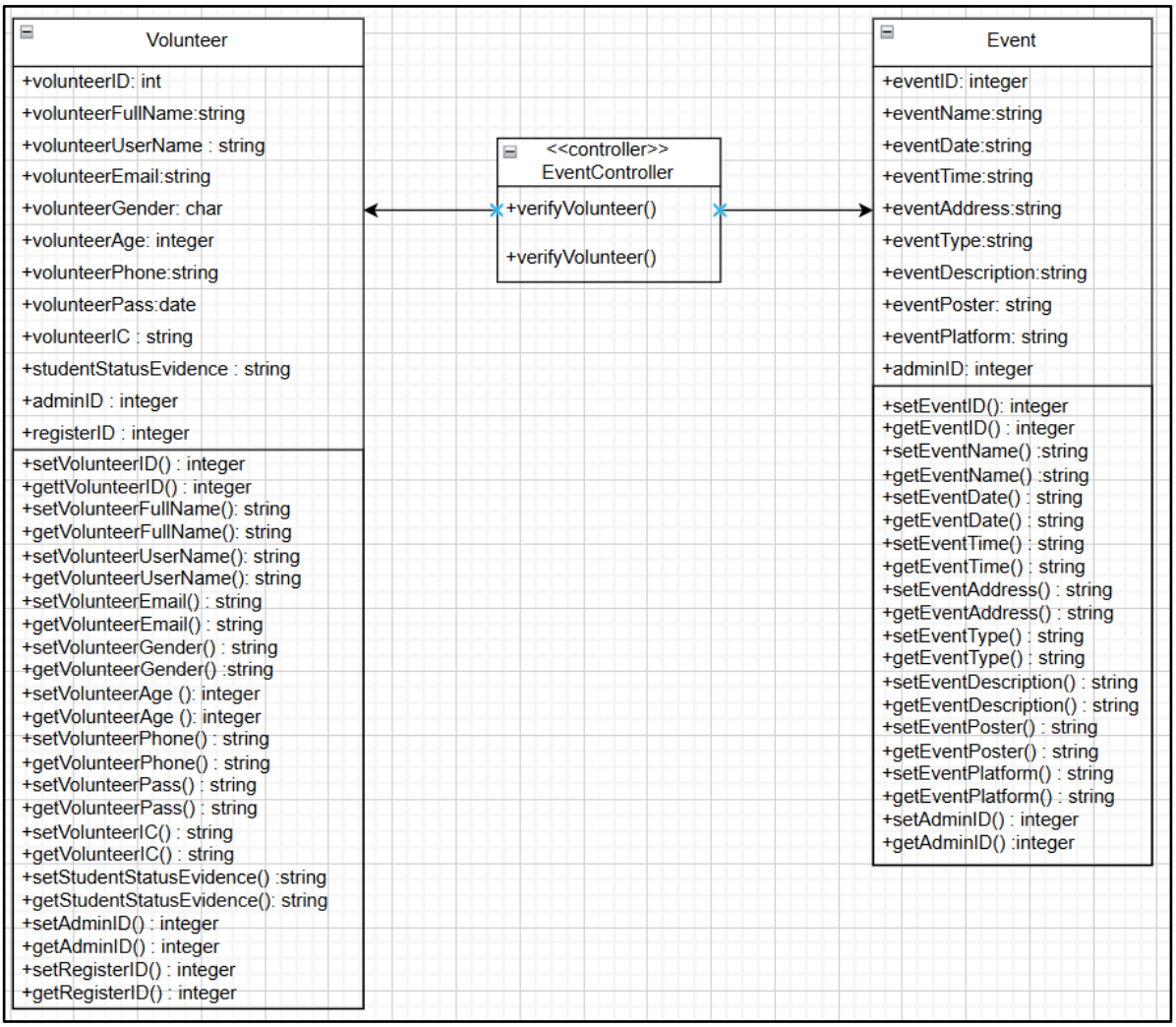
Figure 8.1 Entity Relationship Diagram of Katzen Event Management System

2. First Cut Design Class Diagram

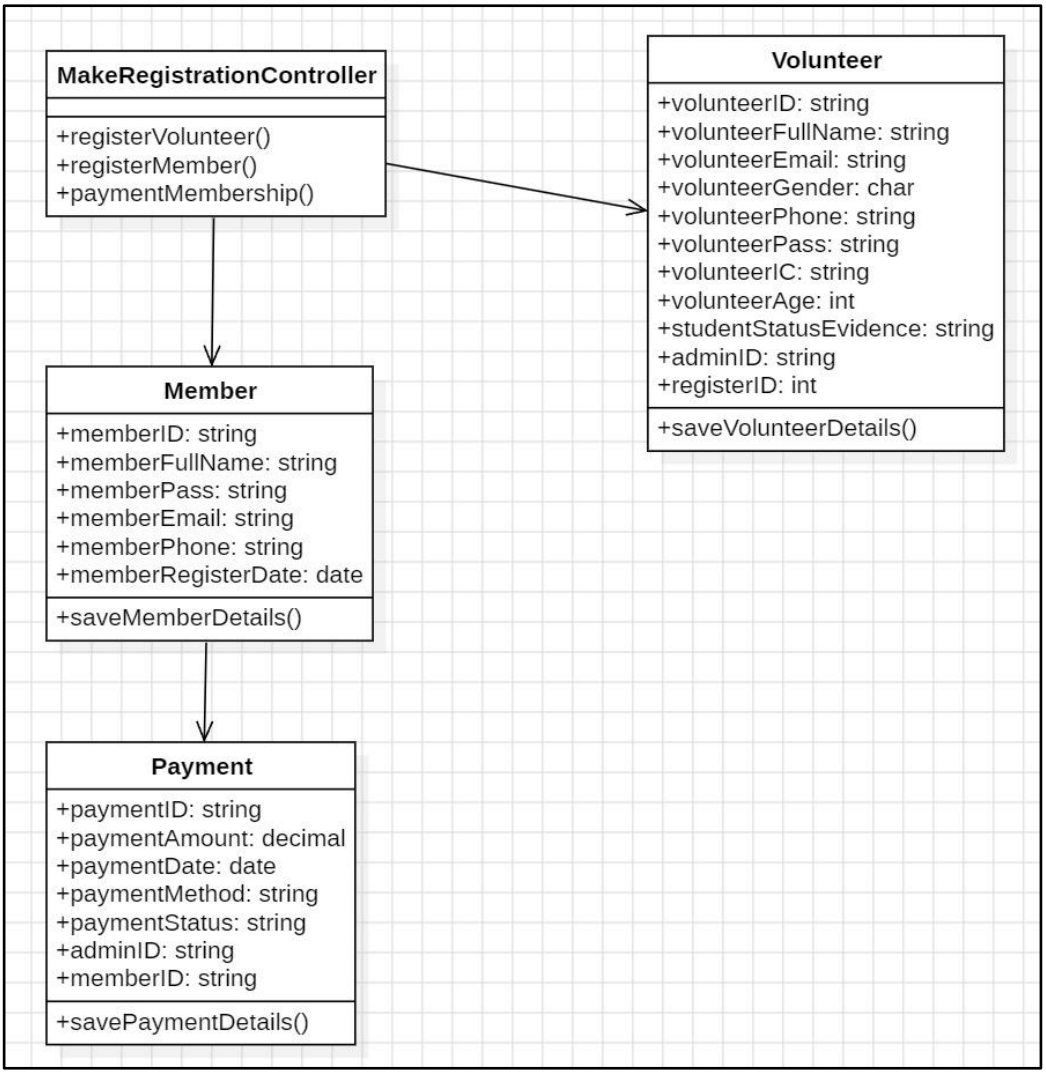
UCD-100 Make Payment



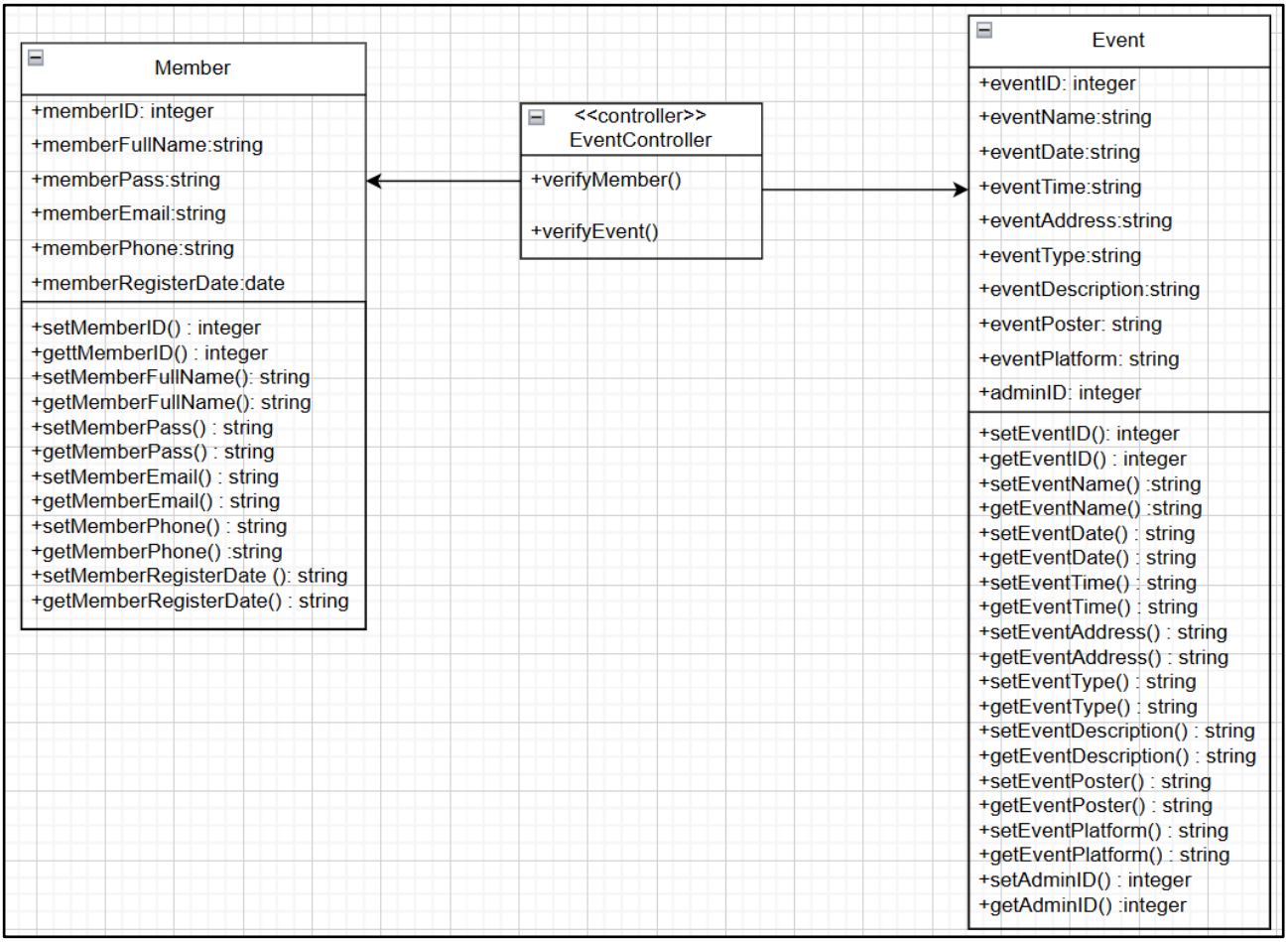
UCD-200 Manage Profile

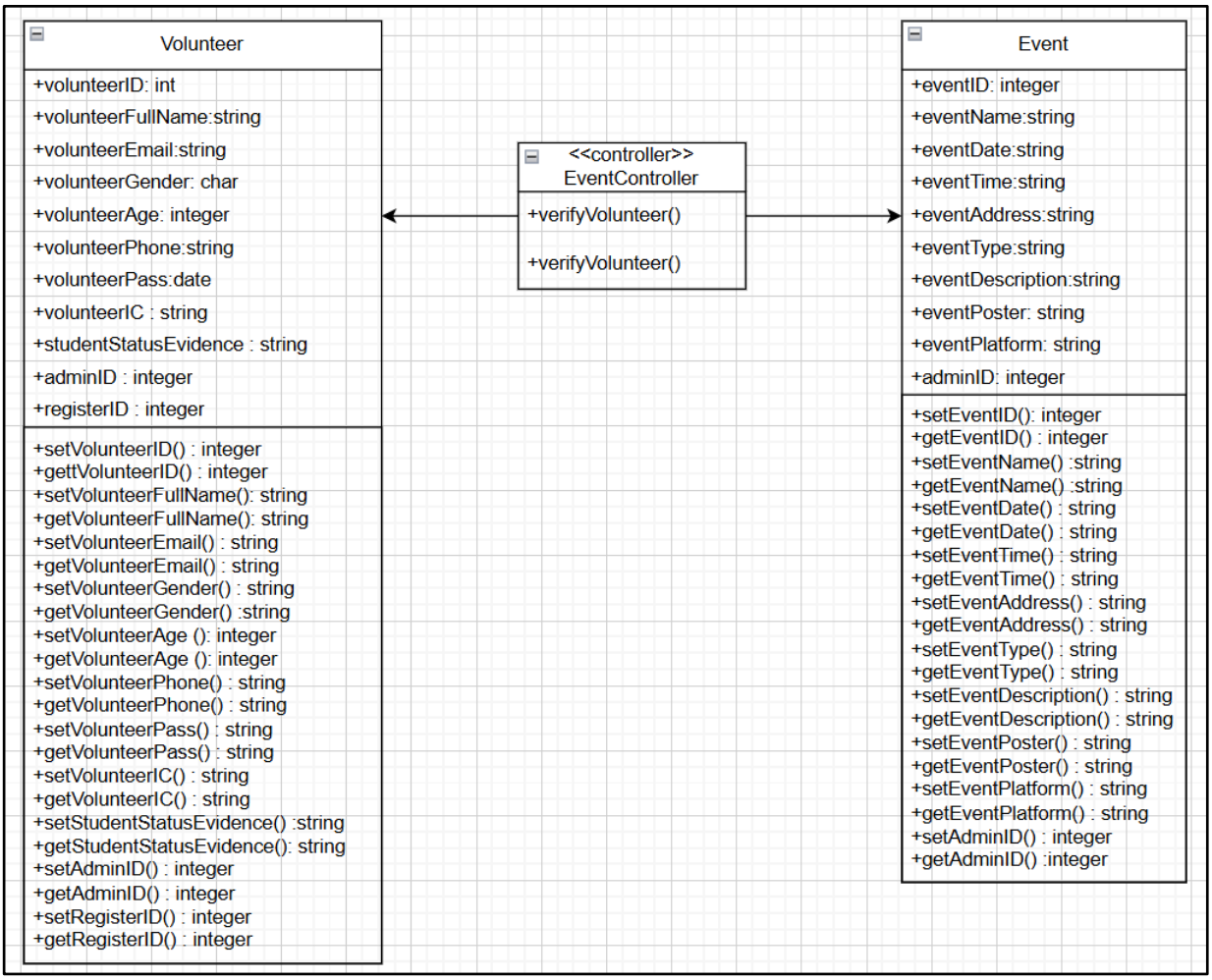


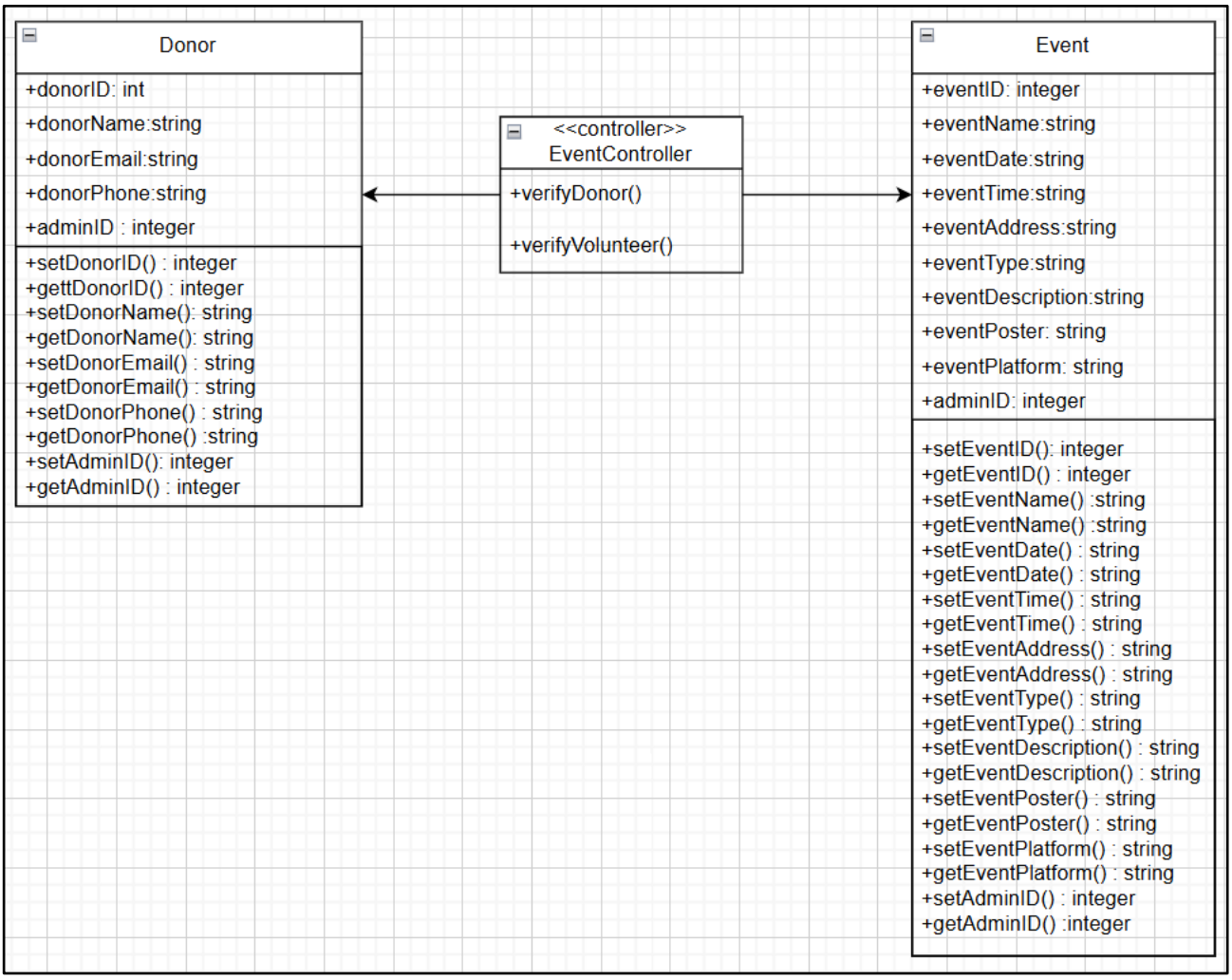
UCD-300 Make Registration

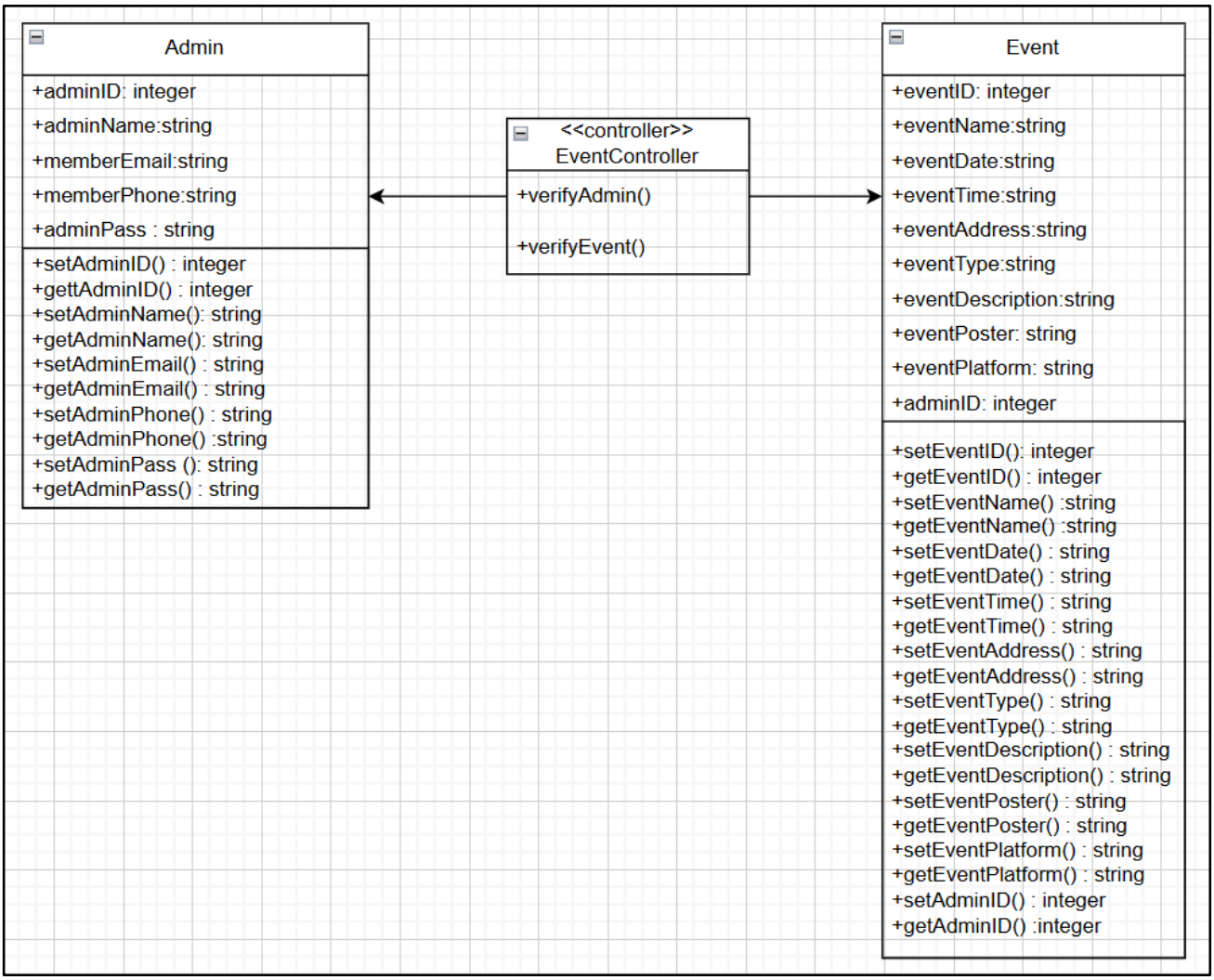


UCD-400 View Event

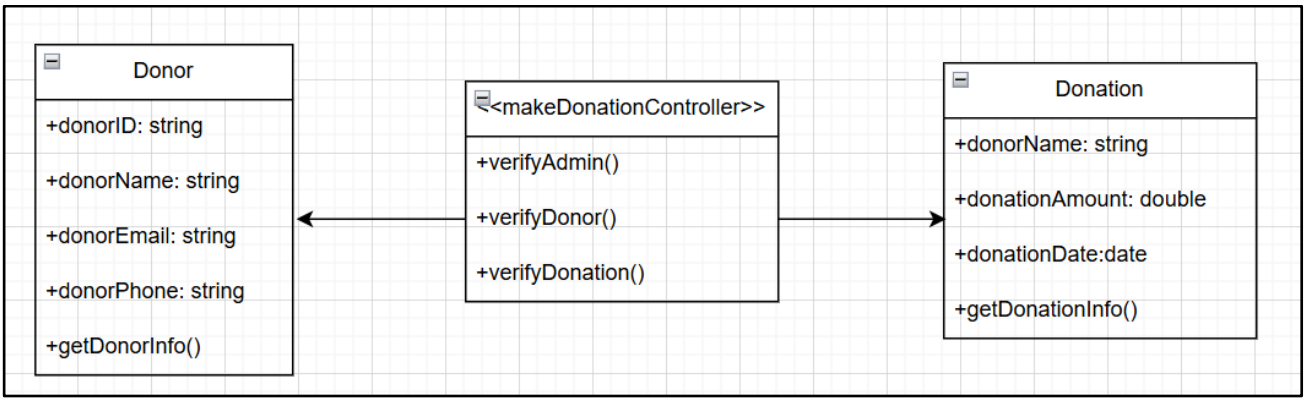



Volunteer
+volunteerID: int
+volunteerFullName:string
+volunteerEmail:string
+volunteerGender: char
+volunteerAge: integer
+volunteerPhone:string
+volunteerPass:date
+volunteerIC : string
+studentStatusEvidence : string
+adminID : integer
+registerID : integer
+setVolunteerID() : integer
+gettVolunteerID() : integer
+setVolunteerFullName(): string
+getVolunteerFullName(): string
+setVolunteerEmail() : string
+getVolunteerEmail() : string
+setVolunteerGender() : string
+getVolunteerGender() :string
+setVolunteerAge (): integer
+getVolunteerAge (): integer
+setVolunteerPhone() : string
+getVolunteerPhone() : string
+setVolunteerPass() : string
+getVolunteerPass() : string
+setVolunteerIC() : string
+getVolunteerIC() : string
+setStudentStatusEvidence() :string
+getStudentStatusEvidence(): string
+setAdminID() : integer
+getAdminID() : integer
+setRegisterID() : integer
+getRegisterID() : integer
<<controller>>
EventController
+verifyVolunteer()
+verifyVolunteer()
Event
+eventID: integer
+eventName:string
+eventDate:string
+eventTime:string
+eventAddress:string
+eventType:string
+eventDescription:string
+eventPoster: string
+eventPlatform: string
+adminID: integer
+setEventID(): integer
+getEventID() : integer
+setEventName() :string
+getEventName() :string
+setEventDate() : string
+getEventDate() : string
+setEventTime() : string
+getEventTime() : string
+setEventAddress() : string
+getEventAddress() : string
+setEventType() : string
+getEventType() : string
+setEventDescription() : string
+getEventDescription() : string
+setEventPoster() : string
+getEventPoster() : string
+setEventPlatform() : string
+getEventPlatform() : string
+setAdminID() : integer
+getAdminID() :integer


Donor
+donorID: int
+donorName:string
+donorEmail:string
+donorPhone:string
+adminID : integer
+setDonorID() : integer
+gettDonorID() : integer
+setDonorName(): string
+getDonorName(): string
+setDonorEmail() : string
+getDonorEmail() : string
+setDonorPhone() : string
+getDonorPhone() :string
+setAdminID(): integer
+getAdminID() : integer
<<controller>>
EventController
+verifyDonor()
+verifyVolunteer()
Event
+eventID: integer
+eventName:string
+eventDate:string
+eventTime:string
+eventAddress:string
+eventType:string
+eventDescription:string
+eventPoster: string
+eventPlatform: string
+adminID: integer
+setEventID(): integer
+getEventID() : integer
+setEventName() :string
+getEventName() :string
+setEventDate() : string
+getEventDate() : string
+setEventTime() : string
+getEventTime() : string
+setEventAddress() : string
+getEventAddress() : string
+setEventType() : string
+getEventType() : string
+setEventDescription() : string
+getEventDescription() : string
+setEventPoster() : string
+getEventPoster() : string
+setEventPlatform() : string
+getEventPlatform() : string
+setAdminID() : integer
+getAdminID() :integer

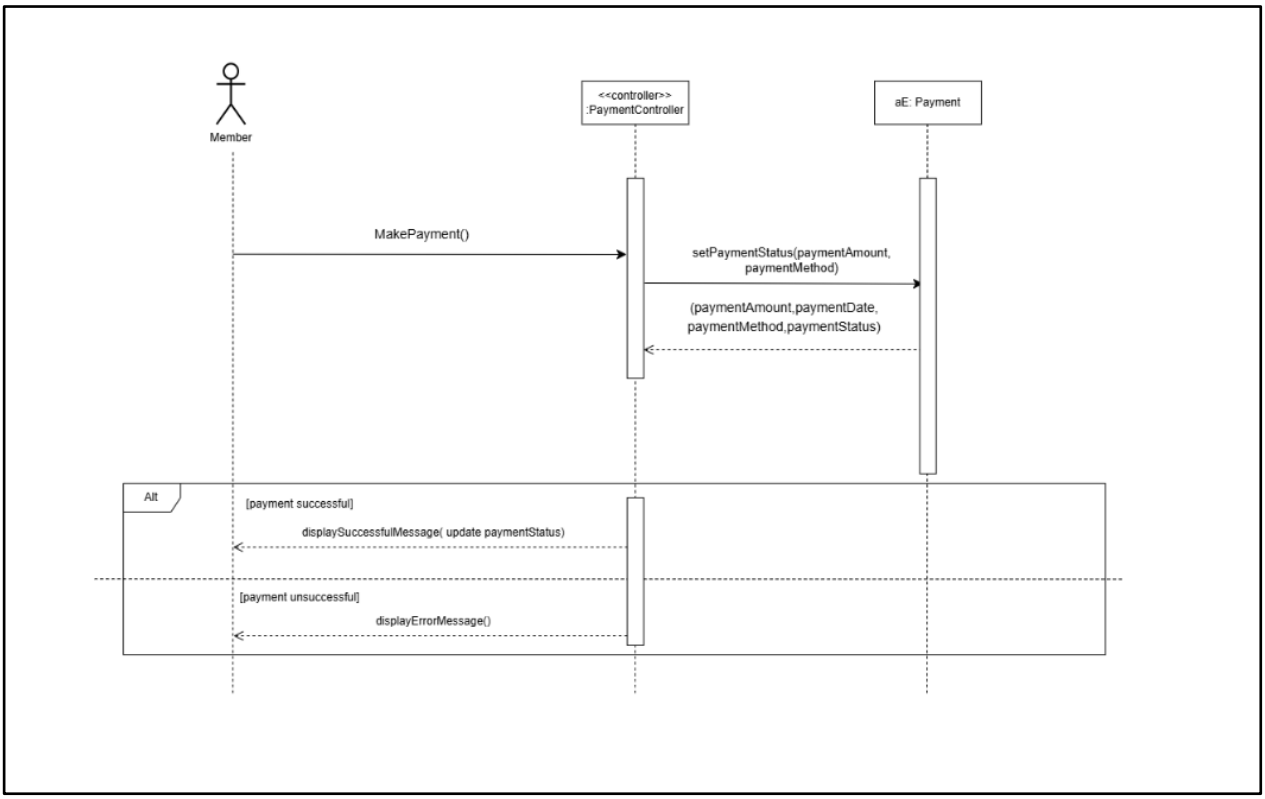


UCD-500 Manage Event

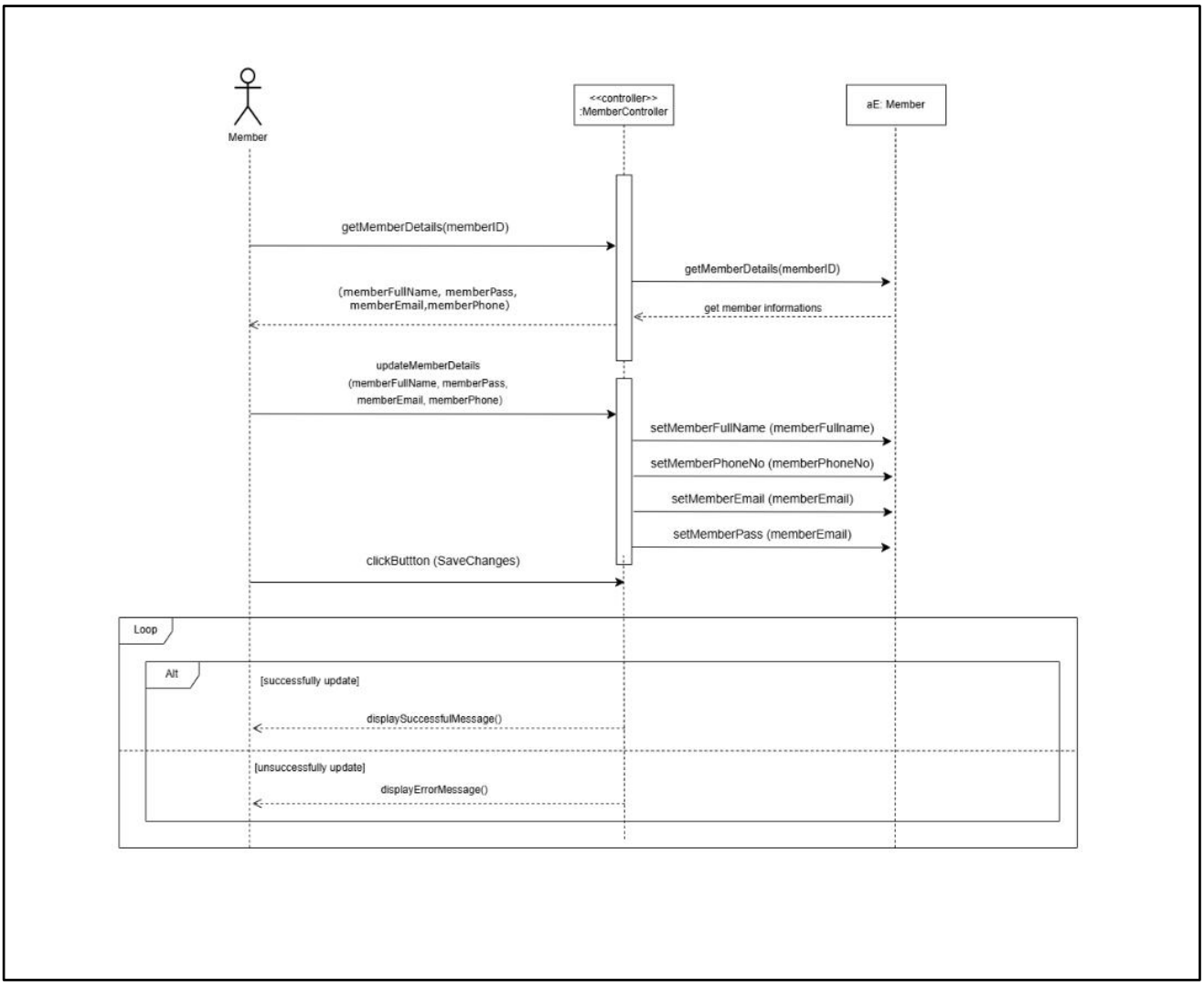


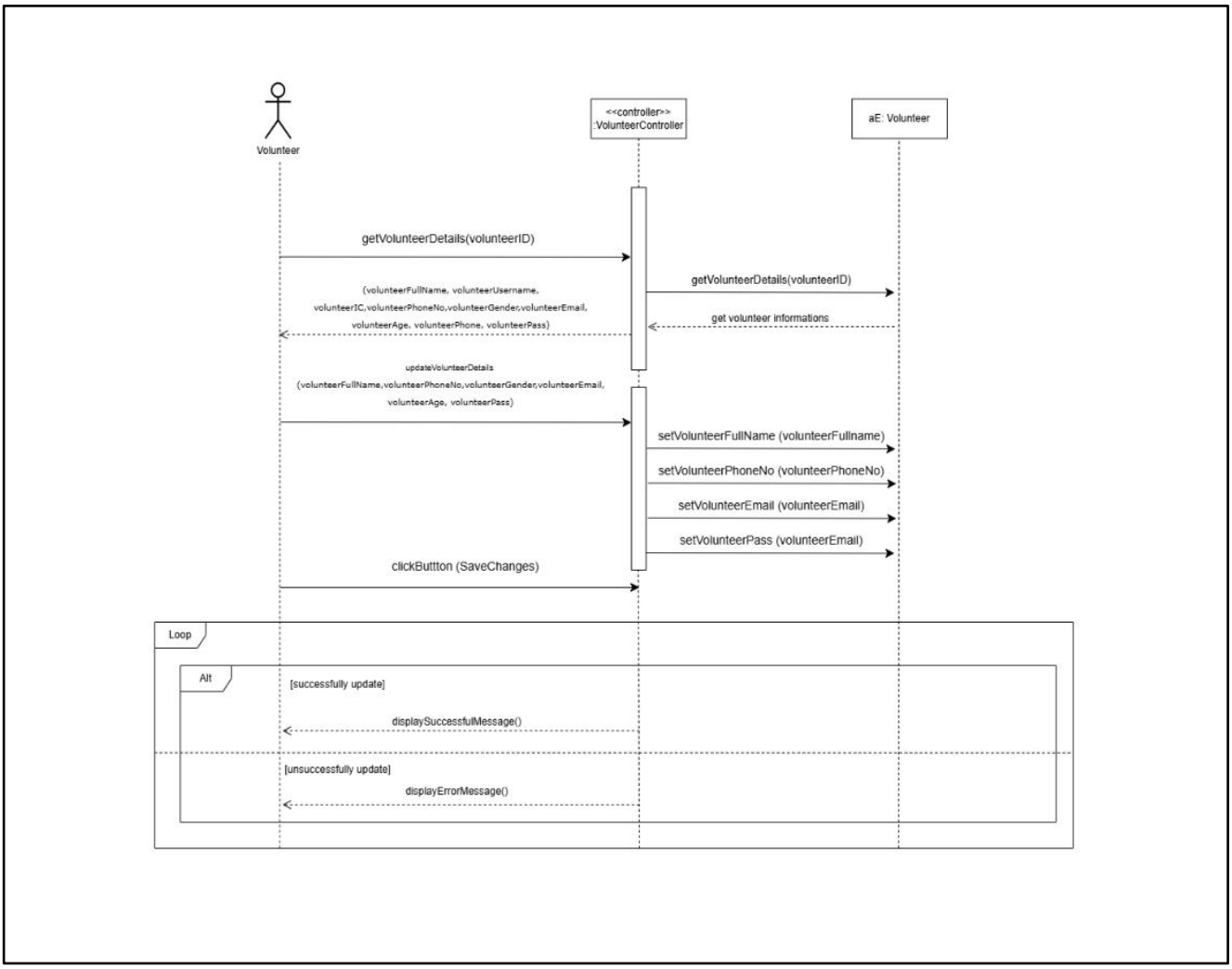
3. First Cut Sequence Diagram

UCD-100 Make Payment

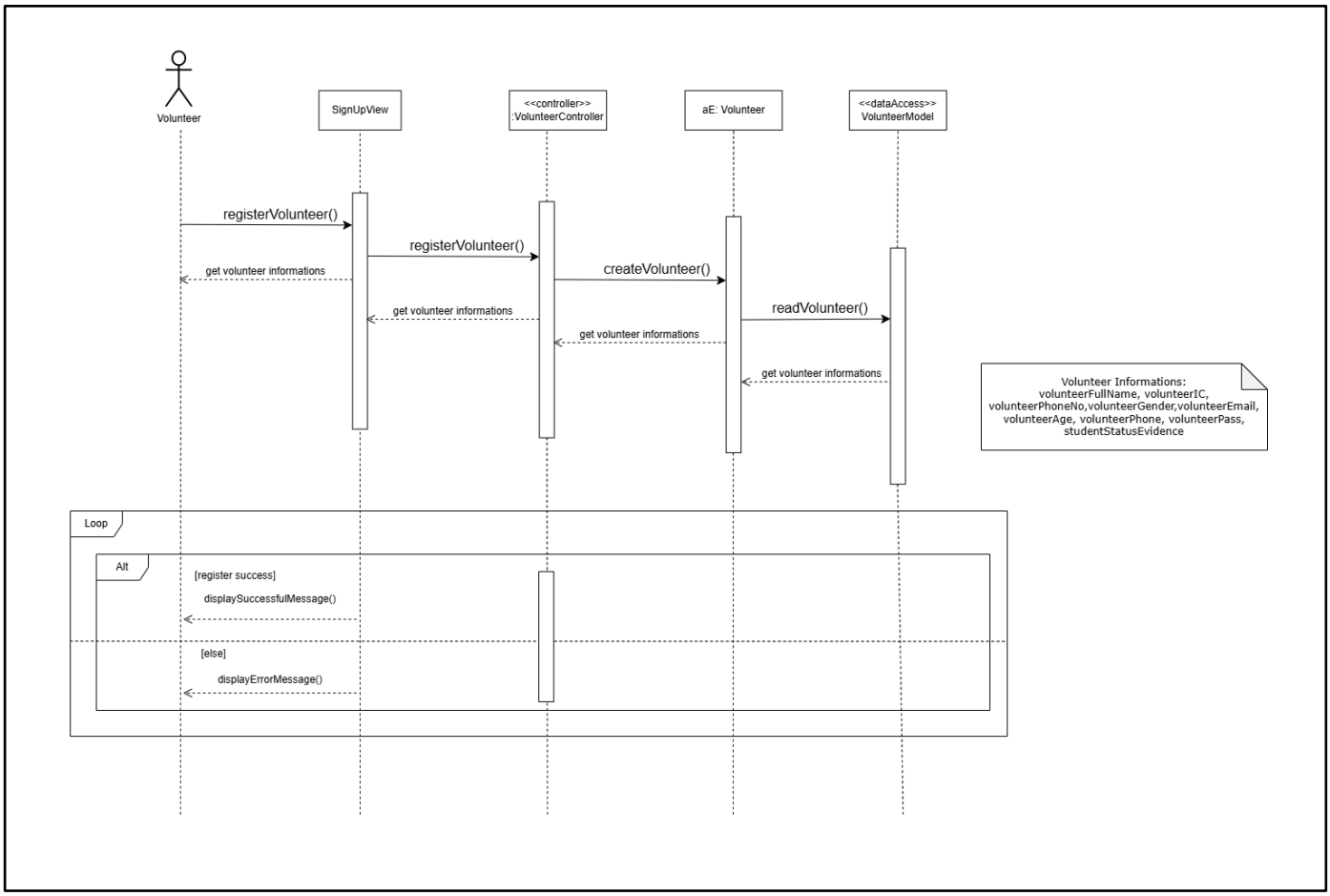


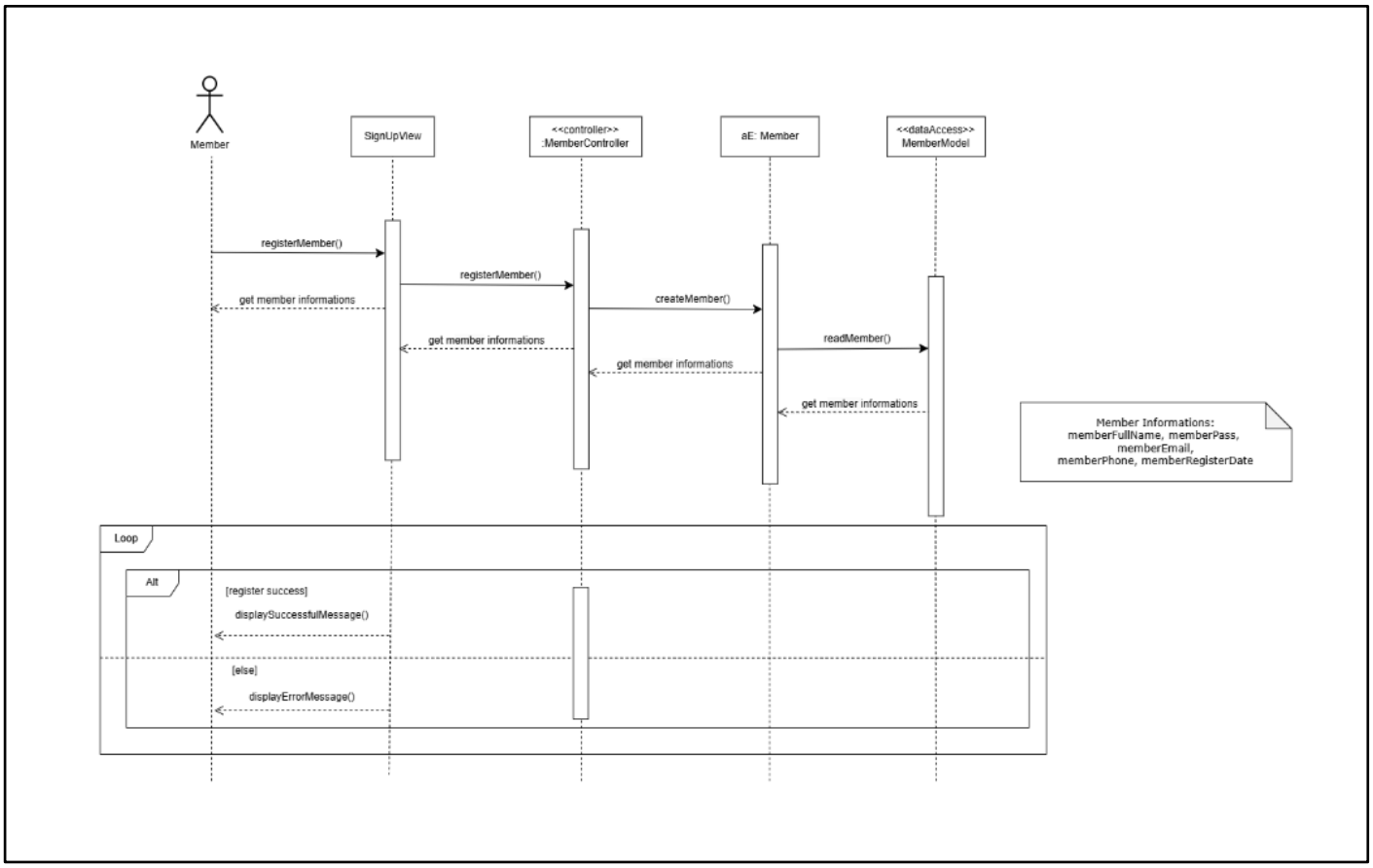
UCD-200 Manage Profile



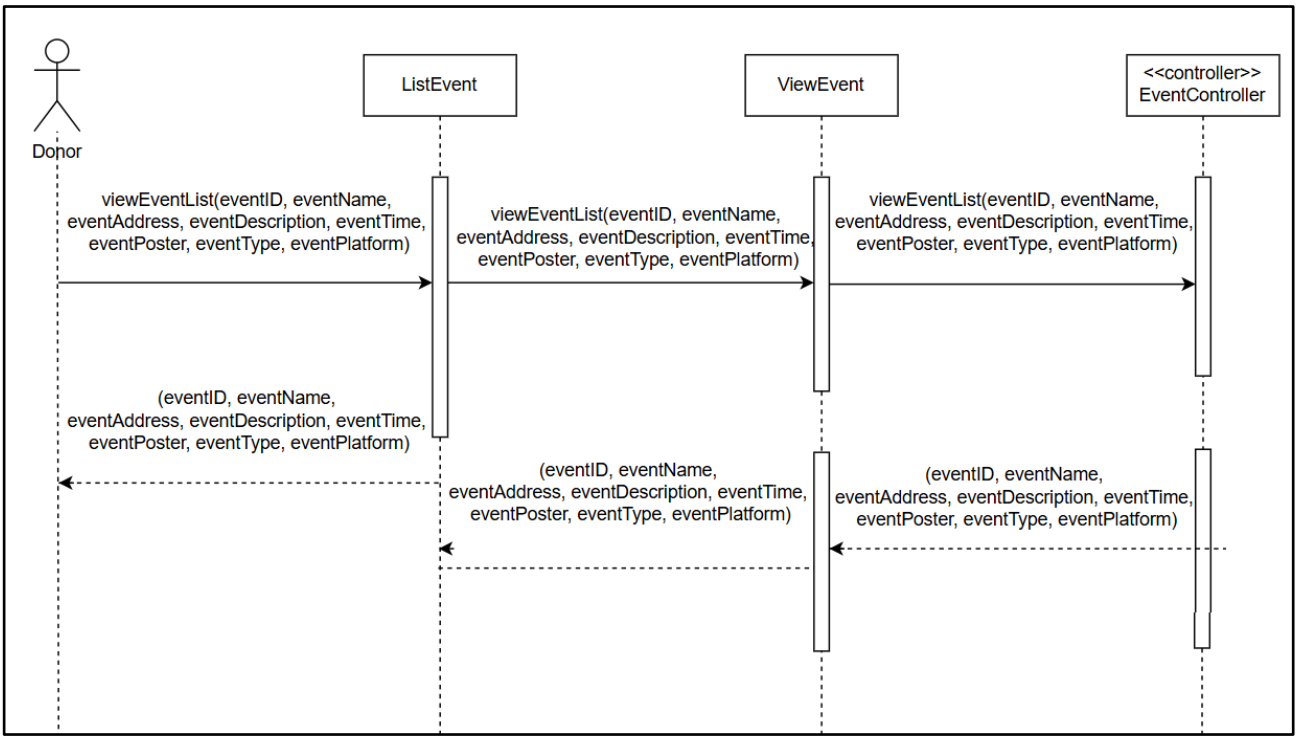


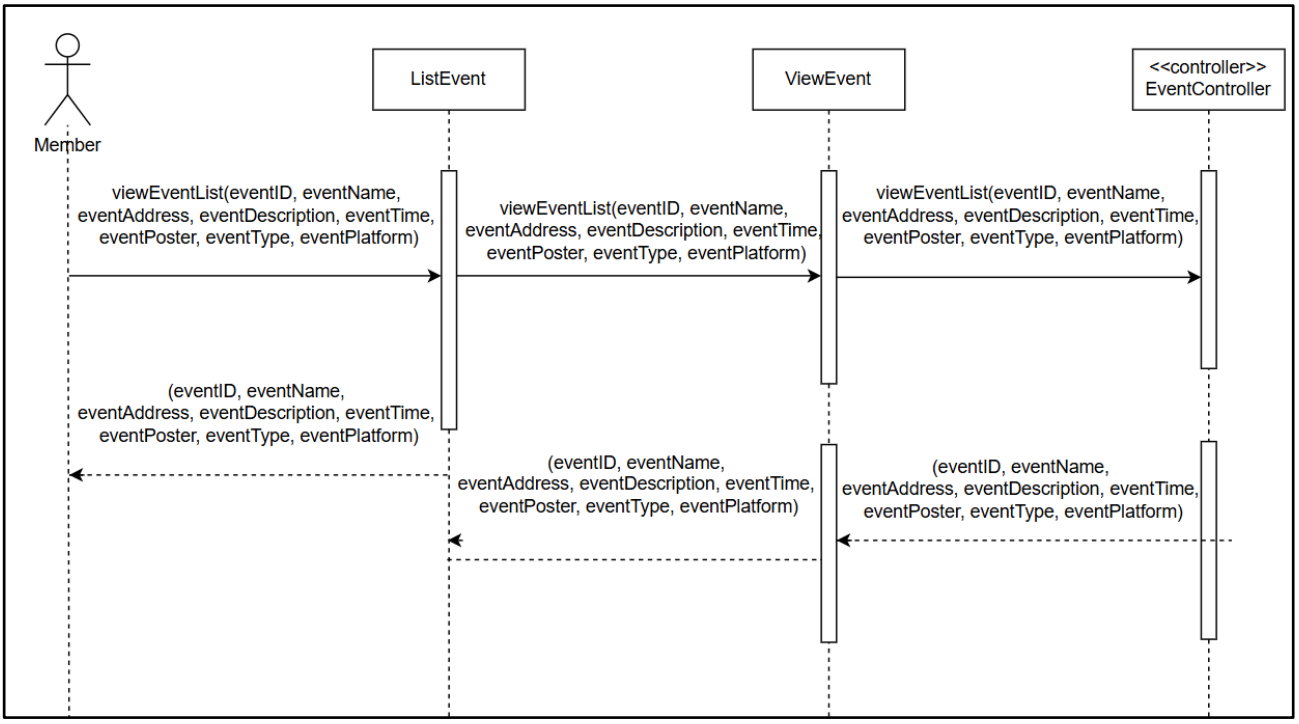
UCD-300 Make Registration

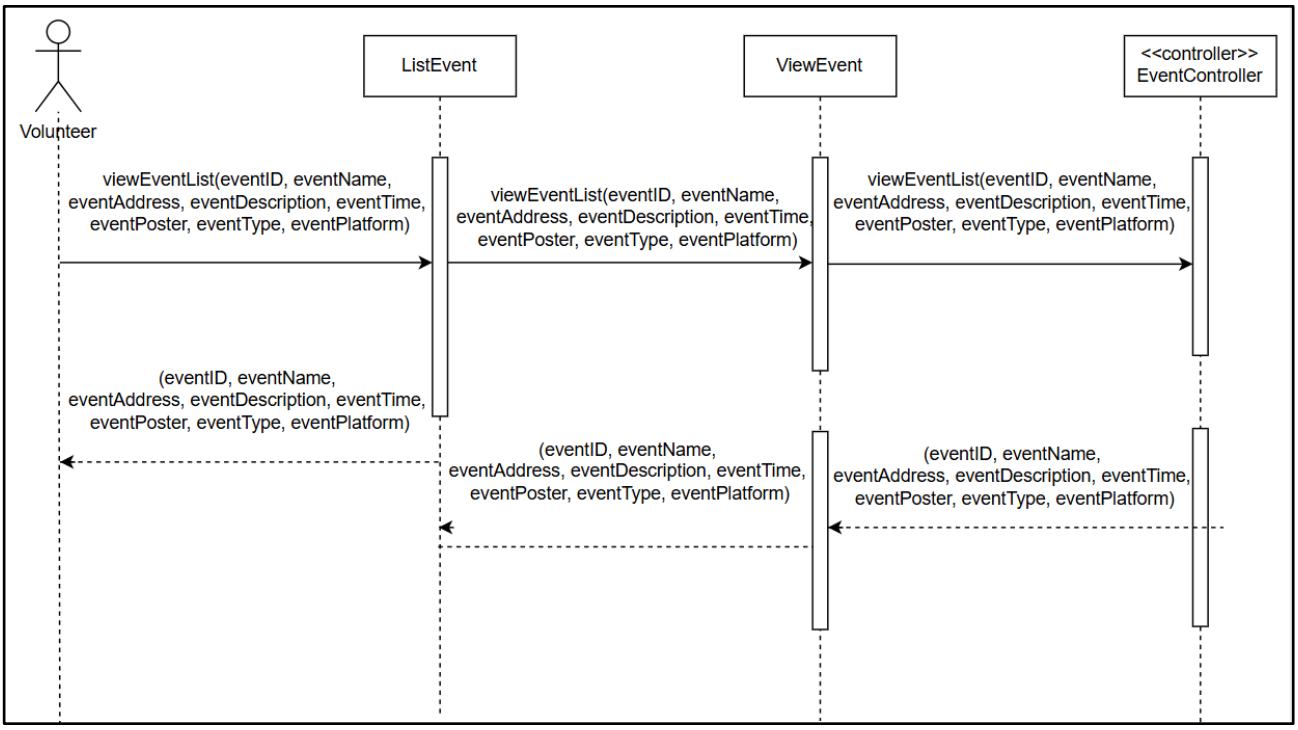

Volunteer
SignUpView
<<controller>>
:VolunteerController
aE: Volunteer
<<dataAccess>>
VolunteerModel
registerVolunteer()
registerVolunteer()
createVolunteer()
readVolunteer()
get volunteer informations
get volunteer informations
get volunteer informations
get volunteer informations
Volunteer Informations:
volunteerFullName, volunteerIC,
volunteerPhoneNo,volunteerGender,volunteerEmail,
volunteerAge, volunteerPhone, volunteerPass,
studentStatusEvidence
Loop
Alt
[register success]
displaySuccessfulMessage()
[else]
displayErrorMessage()

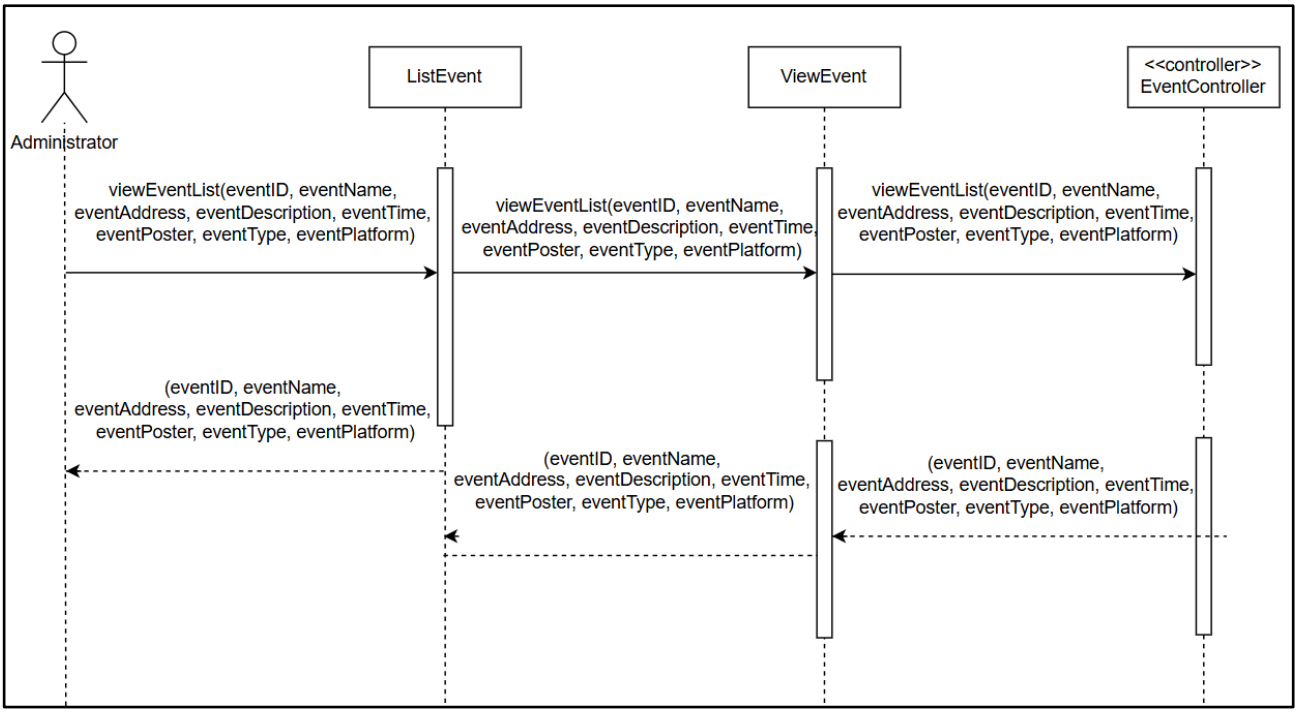

Member
SignUpView
<<controller>>
:MemberController
aE: Member
<<dataAccess>>
MemberModel
registerMember()
registerMember()
createMember()
readMember()
get member informations
get member informations
get member informations
get member informations
Member Informations:
memberFullName, memberPass,
memberEmail,
memberPhone, memberRegisterDate
Loop
Alt
[register success]
displaySuccessfulMessage()
[else]
displayErrorMessage()

UCD-400 View Event

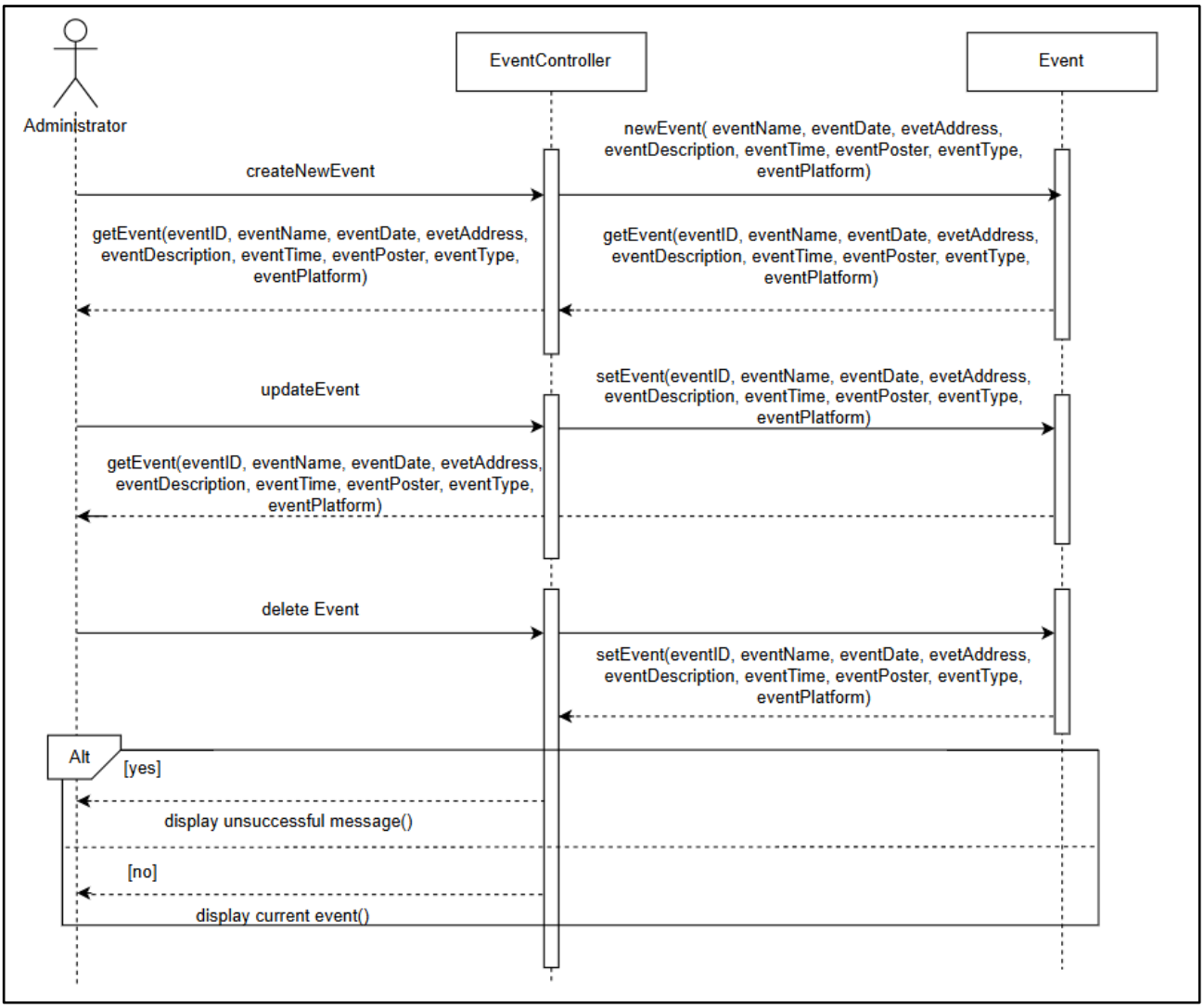




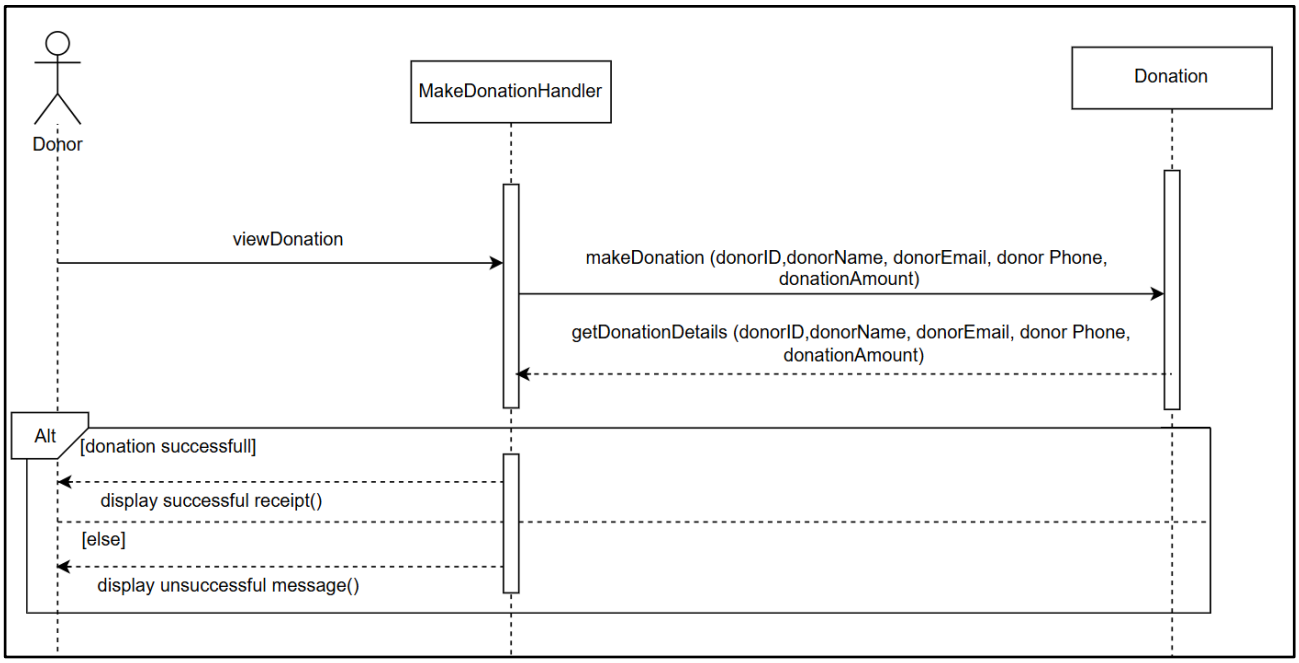

Volunteer
ListEvent
ViewEvent
<<controller>>
EventController
viewEventList(eventID, eventName, eventAddress, eventDescription, eventTime, eventPoster, eventType, eventPlatform)
viewEventList(eventID, eventName, eventAddress, eventDescription, eventTime, eventPoster, eventType, eventPlatform)
viewEventList(eventID, eventName, eventAddress, eventDescription, eventTime, eventPoster, eventType, eventPlatform)
(eventID, eventName, eventAddress, eventDescription, eventTime, eventPoster, eventType, eventPlatform)
(eventID, eventName, eventAddress, eventDescription, eventTime, eventPoster, eventType, eventPlatform)
(eventID, eventName, eventAddress, eventDescription, eventTime, eventPoster, eventType, eventPlatform)


Administrator
ListEvent
ViewEvent
<<controller>>
EventController
viewEventList(eventID, eventName, eventAddress, eventDescription, eventTime, eventPoster, eventType, eventPlatform)
viewEventList(eventID, eventName, eventAddress, eventDescription, eventTime, eventPoster, eventType, eventPlatform)
viewEventList(eventID, eventName, eventAddress, eventDescription, eventTime, eventPoster, eventType, eventPlatform)
(eventID, eventName, eventAddress, eventDescription, eventTime, eventPoster, eventType, eventPlatform)
(eventID, eventName, eventAddress, eventDescription, eventTime, eventPoster, eventType, eventPlatform)
(eventID, eventName, eventAddress, eventDescription, eventTime, eventPoster, eventType, eventPlatform)

UCD-500 Manage Event



UCD-700 Make Donation


Donor
MakeDonationHandler
Donation
viewDonation
makeDonation (donorID,donorName, donorEmail, donor Phone, donationAmount)
getDonationDetails (donorID,donorName, donorEmail, donor Phone, donationAmount)
Alt
[donation successfull]
display successful receipt()
[else]
display unsuccessful message()