# Software Design Document

**for**

# Module 1: MyKhairat Application Project

**Version 1.0 approved**

**Prepared by**

**Muhammad Lukman Aqil Mohamed Laile**

**Muhammad Nur Addin Abdul Hamid**

**Muhammad Akmal Hazim Mohd Zaki**

# Contents

# 1 Introduction

## 1.1. Purpose

This document describes and sets forward detailed design, database design, User-Interface (UI), and the architecture of MyKhairat Application. This document will illustrate and provide the introduction, purpose, usage, and detail of the development concepts of the user module. This document is specifically focused on user profiles and their dependent information.

This design document has an accompanying SRS document of MyKhairat Application. It illustrates and provides the overall description, purpose, usage, and detail of the development concepts of the module.

## 1.2. Scope

This document will explain in detail about the design of Sign Up, Complete Profile, Login, Validate User, Claim Khairat, and Approve Khairat process. These functions will have 2 users that are user and Masjid Committee.

## 1.3 Definition and Acronym

| Terms | Definitions |
|---|---|
| UI | User Interface |
| SRS | Software Requirements Specification |

| | |
|---|---|
| SDD | Software Design Document |
| IEEE | Institute of Electrical and Electronics Engineers |

## 1.4 References


I. SRS of MyKhairat Application Project

II. "IEEE Recommended Practice for Software Design Descriptions," in IEEE Std 1016-1998, vol., no., pp.1-23, 4 Dec. 1998, doi: 10.1109/IEEESTD.1998.88828.

III. "IEEE Standard for Information Technology--Systems Design--Software Design Descriptions," in IEEE STD 1016-2009, vol., no., pp.1-35, 20 July 2009, doi: 10.1109/IEEESTD.2009.5167255.

# 2 Design considerations

## 2.1 Assumptions and Dependencies

A1- We assume that the users use a stable internet connection in order to access the application and its features.

A2- We assume that the users have basic knowledge of technologies and know how to use MyKhairat without a user manual.

A3- We depend on the servers to be running and functioning always.

## 2.2 Constraints

C1- The mobile application should be finished, fulfilling all client's requirements, within the 14 weeks given.

C2- The mobile application is displayed only in Malay language.

C3 - Most developers for MyKhairat never have experience building the application using Flutter with Laravel backend.

## 2.3 System Environment

OE1 - The mobile application is developed using Flutter with Laravel backend

OE2 - The database used for the application is MySQL

## 2.4 Design Methodology

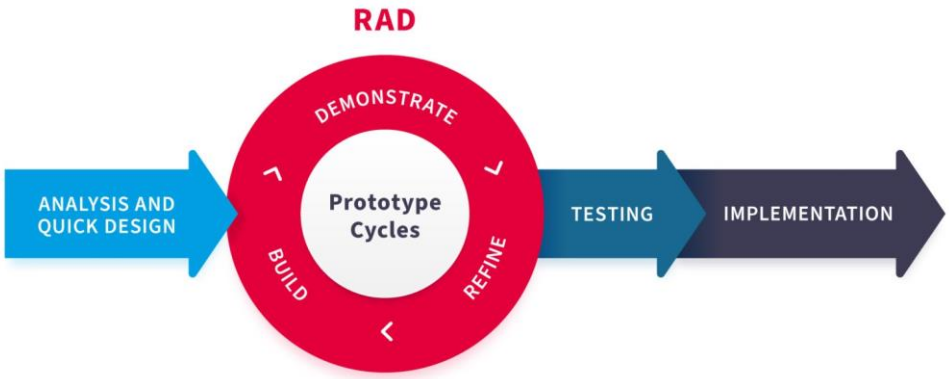



Diagram 2.4.1 Rapid Application Development

The module is built based on the methodology of Agile software development by using the Rapid Application Development (RAD) approach. From the requirement planning stage we will begin building a prototype. After that we will test and refine the prototype to get the feedback. After we gain the positive feedback, we will proceed to testing and implementation stages.

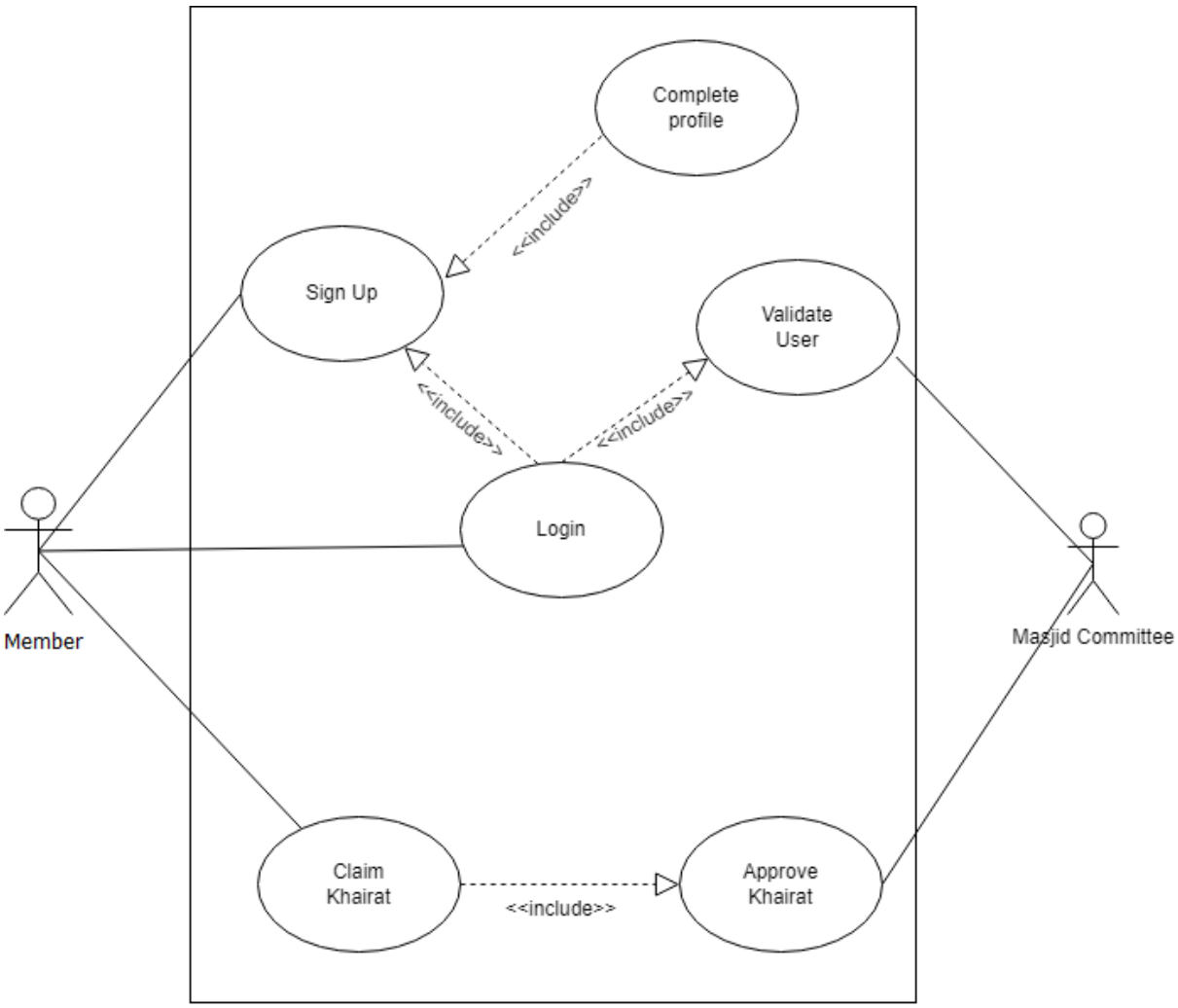
## 2.5 Risk and Volatile Areas

None.

# 3 Detailed Design

## 3.1 Use Case Diagram

Diagram 3.1.1 is the use cases for module 1. For our module we will focus on the Sign Up (REQ 1), Complete Profile (REQ 2), Login (REQ 3), Validate User (REQ 4), Claim Khairat (REQ 17) and Approve Khairat (REQ 18). For this module we will have 2 users that are Member and Masjid Committee.




*3.1.1 Use Case Diagram*

## 3.2 Class Diagram

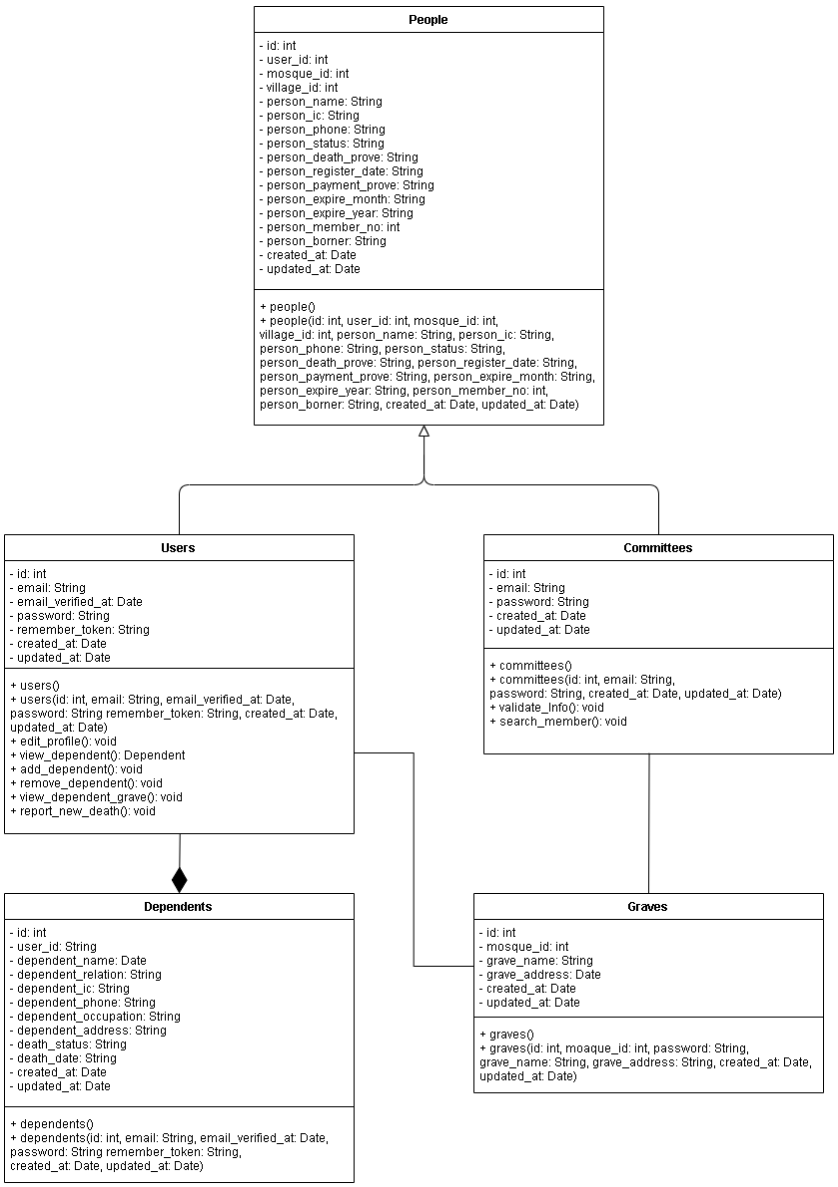



*Diagram 3.2.1 - Class Diagram*

### 3.2.1 People Class

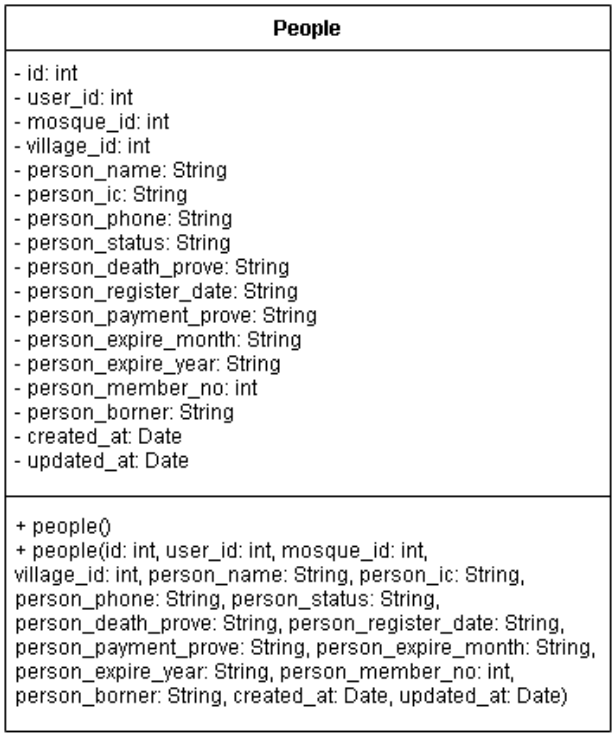



*Diagram 3.2.1.1 People Class*

**a. Classification**

The class component consists of attributes that include private modifiers, attribute names, data types, and a list of methods.

**b. Definition**

The people class is a class that contains many attributes based on the requirements indicated in Diagram 3.2.1.1 above. This class has both no-arguments constructor and constructor with parameters to initiate the attributes inside it.

**c. Responsibilities**

Diagram 3.2.1.1 shows the People class. As this is an entity class, it is responsible for creating people and storing the people's information.

#### 3.2.1.1 Operation Specification

| | |
|---|---|
| Operation Specification: | people(id: int, user_id: int, mosque_id: int,<br>village_id: int, person_name: String, person_ic: String,<br>person_phone: String, person_status: String,<br>person_death_prove: String,<br>person_register_date: String,<br>person_payment_prove: String,<br>person_expire_month: String,<br>person_expire_year: String,<br>person_member_no: int,<br>person_borner: String, created_at: Date, updated_at: Date) |
| Operation intent: | Creates a new people by entering and submitting training details parameters |
| Operation signature: | people:: people(id: int, user_id: int, mosque_id: int,<br>village_id: int, person_name: String, person_ic: String,<br>person_phone: String, person_status: String,<br>person_death_prove: String,<br>person_register_date: String,<br>person_payment_prove: String,<br>person_expire_month: String,<br>person_expire_year: String,<br>person_member_no: int,<br>person_borner: String, created_at: Date, updated_at: Date) |

| | |
|---|---|
| Logic description<br><br>(Pre-Post Condition): | Context: people<br>Pre:<br>Post: display message “people successfully created” |
| Other operations called: | + people |
| Events transmitted to other objects: | None |
| Attribute set: | None |
| Response to exceptions: | None |
| Non-functional requirements: | Performance: This operation should take place in less than 2 minutes. |

### 3.2.2 Committee Class

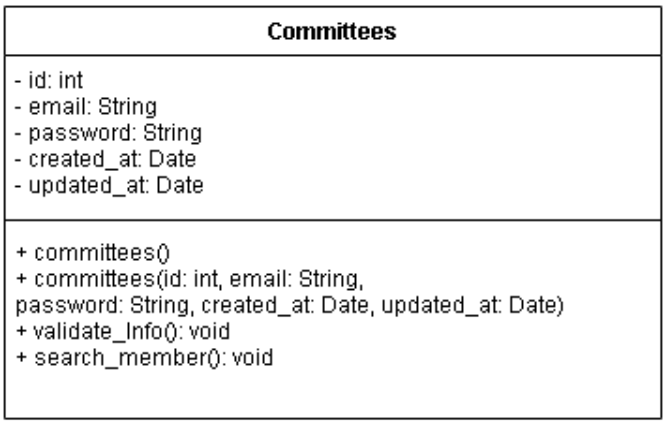



*Diagram 3.2.2.1 - Committee Class*

**a. Classification**

The class component consists of attributes that include private modifiers, attribute names, data types, and a list of methods.

**b. Definition**

The committee class is a class that contains many attributes based on the requirements indicated in Diagram 3.2.2.1 above. This class has a method where the committee can validate the dependent information and to search members of the mosque. Lastly, this class also has both no-arguments constructor and also constructor with parameters to initiate the attributes inside it.

**c. Responsibilities**

Diagram 3.2.3.1 shows the Committee class. As this is an entity class, it is responsible for creating committees, validating dependent information, searching members and storing the committees' information.

### 3.2.2.1 Operation Specification

| | |
|---|---|
| Operation Specification: | committees(id: int, email: String, password: String, created_at: Date, updated_at: Date) |
| Operation intent: | Creates a new committee by entering and submitting committee details parameters |
| Operation signature: | committees:: committees(id: int, email: String, password: String, created_at: Date, updated_at: Date) |
| Logic description<br><br>(Pre-Post Condition): | Context: committee |

| | |
|---|---|
| Other operations called: | + committee()<br>+ validate_info(): void<br>+ search_member() member |
| Events transmitted to other objects: | None |
| Attribute set: | None |
| Response to exceptions: | None |
| Non-functional requirements: | Performance: This operation should take place in less than 2 minutes. |

### 3.2.3 Dependent Class

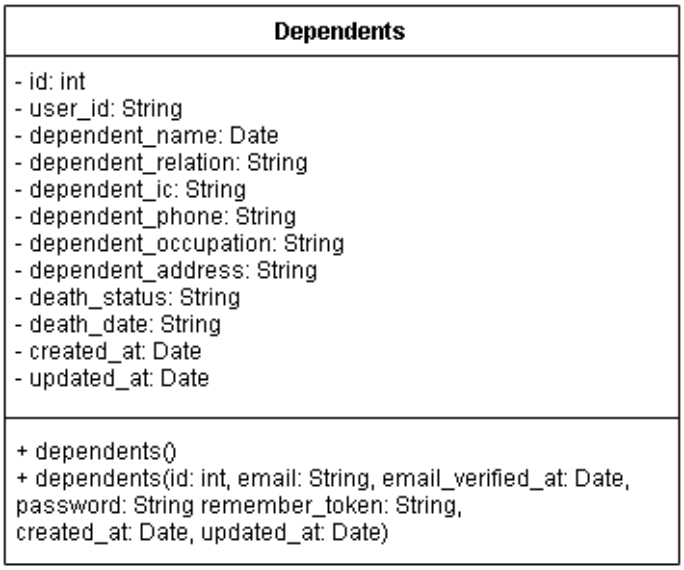



*Diagram 3.2.3.1 - Dependent Class*

### a. Classification

The class component consists of attributes that include private modifiers, attribute names, data types, and a list of methods.

### b. Definition

The Dependent class is a class containing numerous attributes based on the requirements specified in Diagram 3.2.3.1. There are some methods in the Dependent Class that explain the behavior of this class. This class has both no-args constructor and constructor with parameters to initiate the attributes inside it.

### c. Responsibilities

Diagram 3.2.3.1 shows the Dependent class. As this is an entity class, it is responsible for creating dependents, editing dependents and storing the dependent's information.

**3.2.3.1 Operation Specification**

| | |
|---|---|
| Operation Specification: | Dependent(id: bigint, user_id: bigint, dependent_name: varchar, dependent_relation: varchar, dependent_ic: varchar, dependent_phone: varchar, depedent_occupation: varchar, dependent_date: varchar, death_status: varchar, death_date: varchar, created_at: timestamp, updated_at: timestamp) |
| Operation intent: | Creates a new dependent by inputting and storing dependent details parameters |

| | |
|---|---|
| Operation signature: | Dependent:: Dependent(id: bigint, user_id: bigint, dependent_name: varchar, dependent_relation: varchar, dependent_ic: varchar, dependent_phone: varchar, depedent_occupation: varchar, dependent_date: varchar, death_status: varchar, death_date: varchar, created_at: timestamp, updated_at: timestamp) |
| Logic description<br><br>(Pre-Post Condition): | **Context:** Dependent<br><br>**Pre:** None<br><br>**Post:** None |
| Other operations called: | **context:** +Dependent() |
| Events transmitted to other objects: | None |
| Attribute set: | None |
| Response to exceptions: | None |
| Non-functional requirements: | Performance: This operation should take place in less than 1 minute. |

# 4 Database Design

This section describes the database design for the MyKhairat mobile application.

## 4.1 Data Model

- Describe the transformation/mapping process from classes to tables for relational database design
- Present the data model

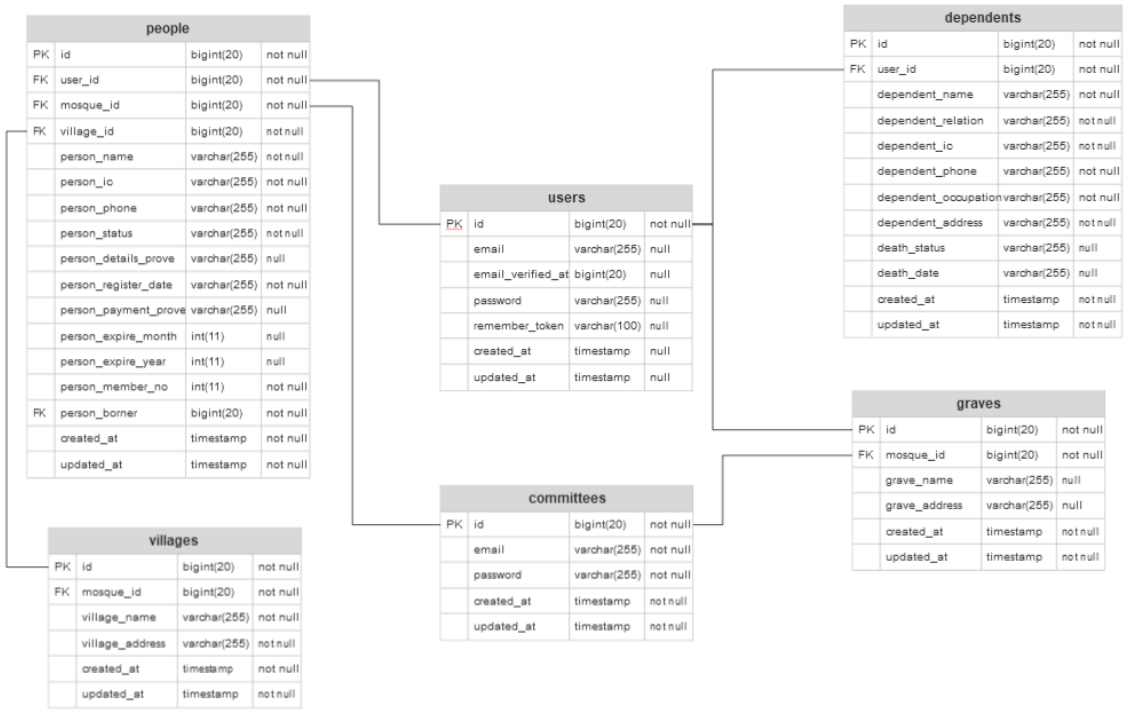



*Diagram 4.1.1 - Relational Table Mapping*

In Diagram 4.1.1 shows the data involved within this module. The relational table is based on the class diagram and details the attributes. The database is already in 3rd Normal Form (3NF). It is listed with the name of the attributes - (e.g id), the data type (e.g. bigint), primary key (PK) and foreign key (FK), and the constraint of the attribute (either null or not).

## 4.2 Data Dictionary

Lists all tables for the relational database.

### 4.2.1 Dependents

| Attribute's Name | Data Type | Constraint | Description |
|---|---|---|---|
| id | bigint(20) | PRIMARY KEY(NOT NULL) | Unique ID for the dependent |
| user_id | bigint(20) | FOREIGN KEY(NOT NULL) | Unique ID if it is a user |
| dependent_name | varchar(255) | NOT NULL | Name of the dependent |
| dependent_relation | varchar(255) | NOT NULL | Relationship of the dependent to a user |
| dependent_ic | varchar(255) | NOT NULL | IC number of the dependent |
| dependent_phone | varchar(255) | NOT NULL | Phone number of the dependent |
| depedent_occupation | varchar(255) | NOT NULL | Occupation of the dependent |
| dependent_address | varchar(255) | NOT NULL | Address of the dependent |
| death_status | varchar(255) | NULL | Status of the dependent's wellbeing |

| | | | |
|---|---|---|---|
| death_date | varchar(255) | NULL | Date of the dependent's death |
| created_at | timestamp | NOT NULL | Time at which it is created |
| updated_at | timestamp | NOT NULL | Time at which it is updated |

*Table 4.2.1 - Relation Table for Dependents*

### 4.2.2 People

| **Attribute's Name** | **Data Type** | **Constraint** | **Description** |
|---|---|---|---|
| id | bigint(20) | PRIMARY KEY(NOT NULL) | Unique ID for the user |
| user_id | bigint(20) | FOREIGN KEY(NOT NULL) | Unique ID if it is a user |
| mosque_id | bigint(20) | FOREIGN KEY(NOT NULL) | Unique ID if it is the user's mosque |
| village_id | bigint(20) | FOREIGN KEY(NOT NULL) | Unique ID if it is the user's village |
| person_name | varchar(255) | NOT NULL | Name of the person |
| person_ic | varchar(255) | NOT NULL | IC number of the person |
| person_phone | varchar(255) | NOT NULL | Phone number of the person |
| person_occupation | varchar(255) | NOT NULL | Occupation of the person |

| | | | |
|---|---|---|---|
| person_status | varchar(255) | NOT NULL | The status of the person |
| person_details_prove | varchar(255) | NULL | Proof of the person’s details |
| person_register_date | varchar(255) | NOT NULL | Date of the person’s registration |
| person_payment_prove | varchar(255) | NULL | Proof of the person’s payment |
| person_expire_month | int(11) | NULL | Month of expiration |
| person_expire_year | int(11) | NULL | Year of expiration |
| person_member_no | int(11) | NOT NULL | Number of the member |
| person_borner | bigint(20) | NOT NULL | |
| created_at | timestamp | NOT NULL | The time at which it is created |
| updated_at | timestamp | NOT NULL | The time at which it is updated |

*Table 4.2.2 - Relation Table for Dependents*

### 4.2.3 Committee

| Attribute's Name | Data Type | Constraint | Description |
|---|---|---|---|
| id | bigint(20) | PRIMARY KEY(NOT NULL) | Unique ID for the user |
| email | varchar(255) | NOT NULL | Email of the committee member |
| password | varchar(255) | NOT NULL | Password of the committee member |
| created_at | timestamp | NOT NULL | Time at which it is created |
| updated_at | timestamp | NOT NULL | Time at which it is updated |

*Table 4.2.3 - Relation Table for Committee*

### 4.2.4 Graves

| Attribute's Name | Data Type | Constraint | Description |
|---|---|---|---|
| id | bigint(20) | PRIMARY KEY(NOT NULL) | Grave ID |
| mosque_id | bigint(20) | FOREIGN KEY(NOT NULL) | Unique ID if it is in a mosque |
| grave_name | varchar(255) | NULL | Name of the grave |
| grave_address | varchar(255) | NULL | Address of the dependent's grave |
| created_at | timestamp | NOT NULL | Time at which it is created |

| | | | |
|---|---|---|---|
| updated_at | timestamp | NOT NULL | Time at which it is updated |

*Table 4.2.4 - Relation Table for Graves*

### 4.2.5 Village

| **Attribute's Name** | **Data Type** | **Constraint** | **Description** |
|---|---|---|---|
| id | bigint(20) | PRIMARY KEY(NOT NULL) | Village ID |
| mosque_id | bigint(20) | FOREIGN KEY(NOT NULL) | Unique ID if it has a mosque |
| village_name | varchar(255) | NOT NULL | Name of the village |
| village_address | varchar(255) | NOT NULL | Address of the village |
| created_at | timestamp | NOT NULL | Time at which it is created |
| updated_at | timestamp | NOT NULL | Time at which it is updated |

*Table 4.2.5 - Relation Table for Village*

# 5 User Interface Design

## 5.1 Use case: Sign Up

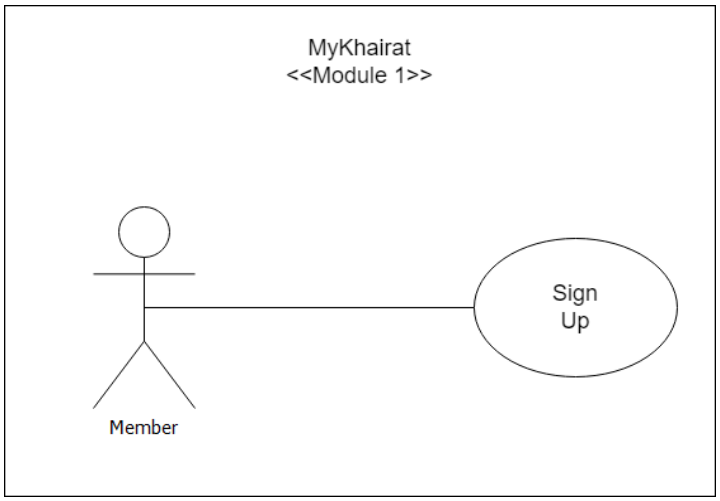



*Diagram 5.1.1 Sign Up Use Case*

In diagram 5.1.1 the Sign Up use case will involve the user and allow them to sign up for registration.

### 5.1.1 Profile Page Prototype

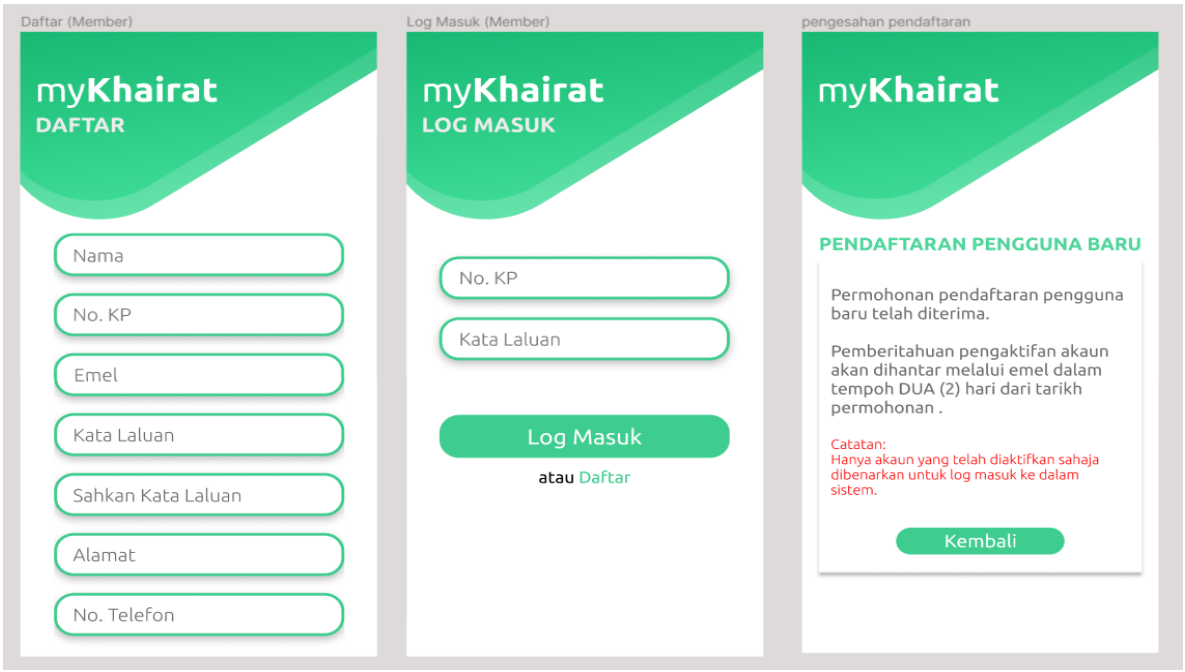



*Diagram 5.1.1.1 Profile Page*

In diagram 5.1.1.1 it shows the sign up steps for user registration. First the user needs to insert the details such as Nama, No. KP and others. After registration, if the user tries to login and the Masjid Committee did not accept the registration yet, the registration request interface will appear.

## 5.2 Use case: Complete Profile

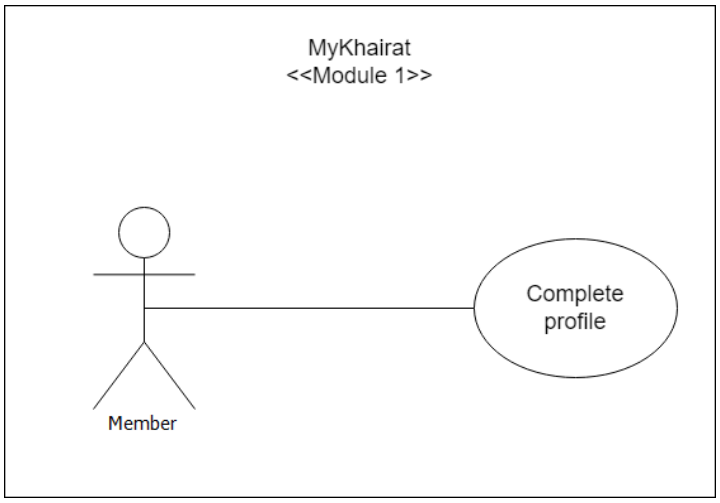



*Diagram 5.2.1 Complete Profile Use Case*

In diagram 5.2.1, the user will complete the details of their profile.

### 5.2.1 Complete Profile Prototype

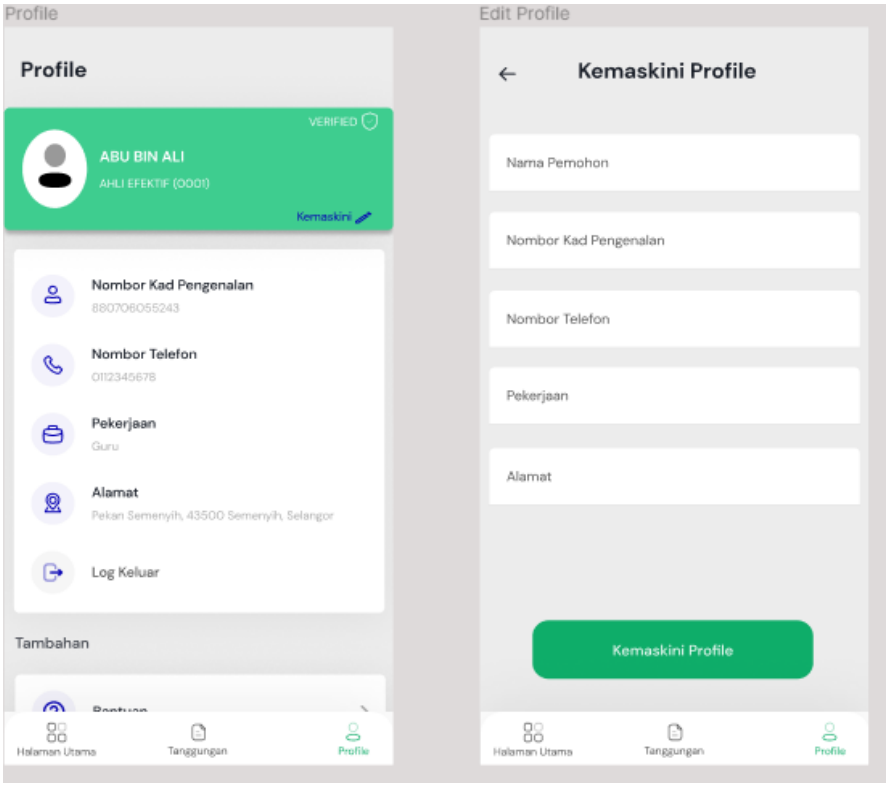



*Diagram 5.2.1.1 Complete profile*

In diagram 5.2.1.1 the member needs to click the profile icon in the bottom right corner. After the interface of the profile appears the member needs to click the kemaskini word in blue colour. After the user clicks the kemaskini, the kemaskini profile will appear and the user needs to complete the profile and click the kemaskini profile button.

## 5.3 Use case: Login

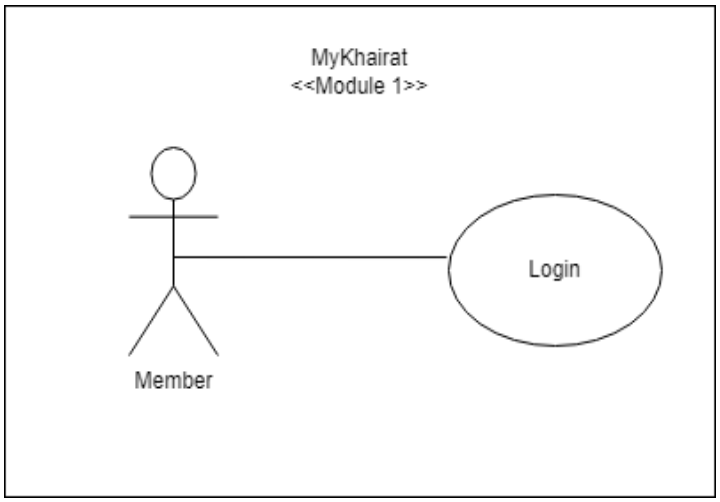



*Diagram 5.3.1 Login Use Case*

For login use case member need to fill up all the details to gain access to the system.

### 5.3.1 Login Prototype

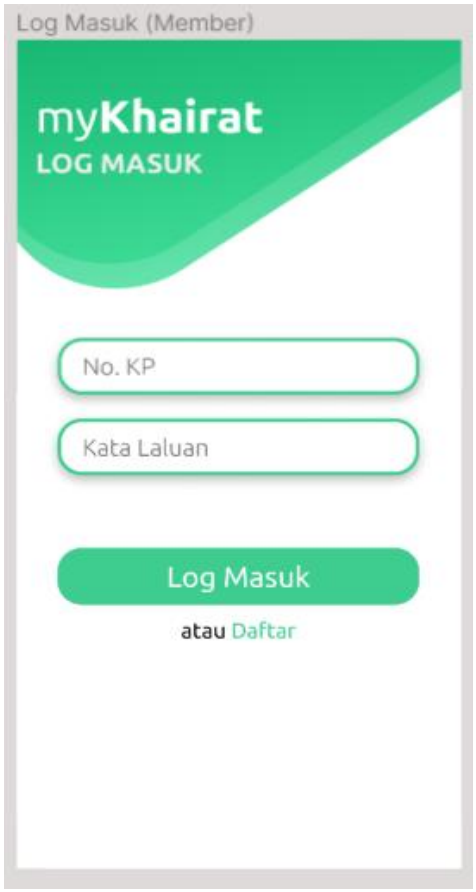



*Diagram 5.3.1.1 Login Page*

In diagram 5.3.1.1 the member needs to login to the page by entering the No. KP and password.

## 5.4 Use case: Validate User

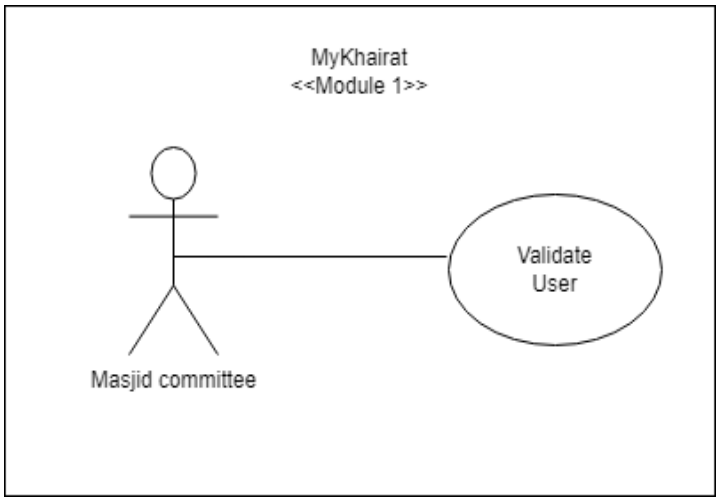



*Diagram 5.4.1 Validate User Use Case*

In diagram 5.4.1 the Validate User use case will involve masjid committee. The Masjid committee will validate the user to accept or reject the registration.

### 5.4.1 Validate User Prototype

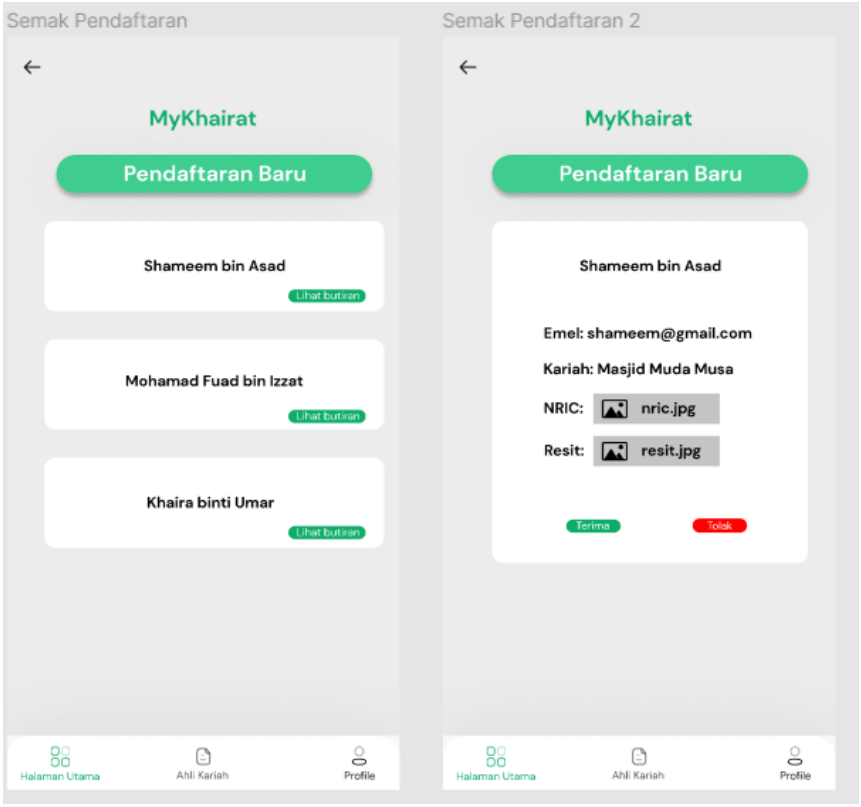



*Diagram 5.4.1.1 Validate User*

In diagram 5.4.1.1 masjid committee can check new registration on homepage. The Masjid committee will click the name box of the new registration and the new page will be shown. Masjid committee will click the terima for accept registration or the tolak button for reject the registration.

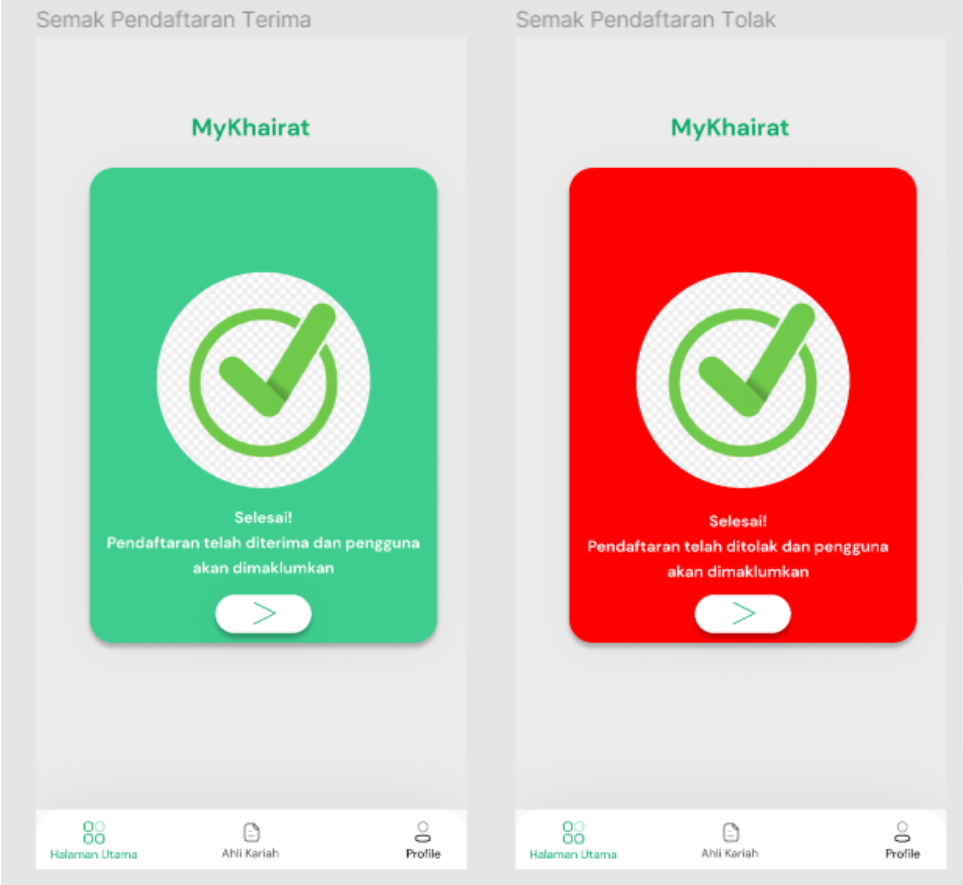



*Diagram 5.4.1.2 Validate User*

In diagram 5.4.1.2 above, these are the interfaces for the masjid committee. The green colour will appear if the masjid committee accepts the registration, but it will show red colour if the masjid committee rejects the registration.

## 5.5 Use case: Claim Khairat

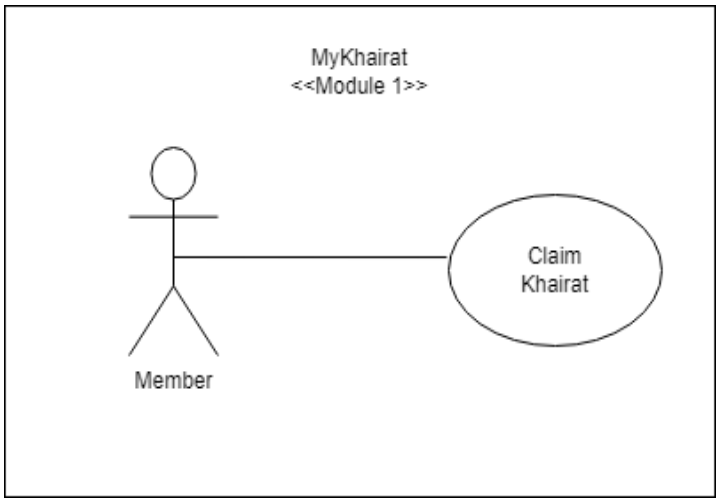



*Diagram 5.5.1 Claim Khairat Use Case*

In diagram 5.5.1 member can claim khairat if one or more of their dependencies has passed away.

### 5.5.1 Claim Khairat Prototype

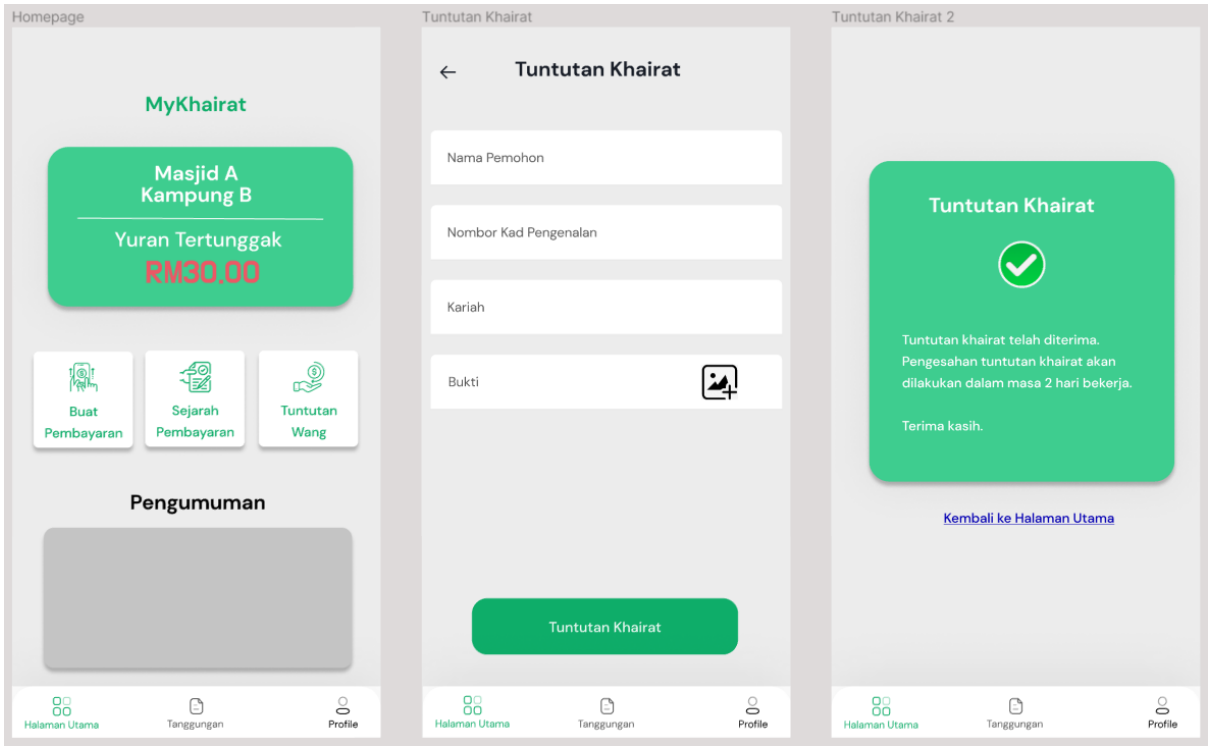



*Diagram 5.5.1.1*

In diagram 5.5.1.1 to claim khairat the user needs to go to halaman utama first and then click the tuntutan wang button. After that, it will show the tuntutan khairat page and the member needs to insert all of the information and also upload the picture and finally click the tuntutan khairat button. After all the information, insert and click the button a page that notified members will be shown.

## 5.6 Use case: Approve Khairat

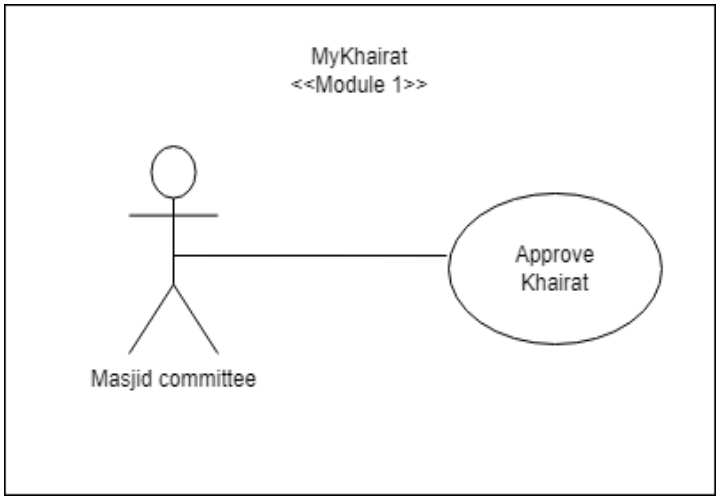



*Diagram 5.6.1 Approve Khairat*

In diagram 5.6.1 the approved khairat process will be done by the masjid committee. The Masjid committee will prove or reject the claim.

### 5.6.1 Approve Khairat Prototype

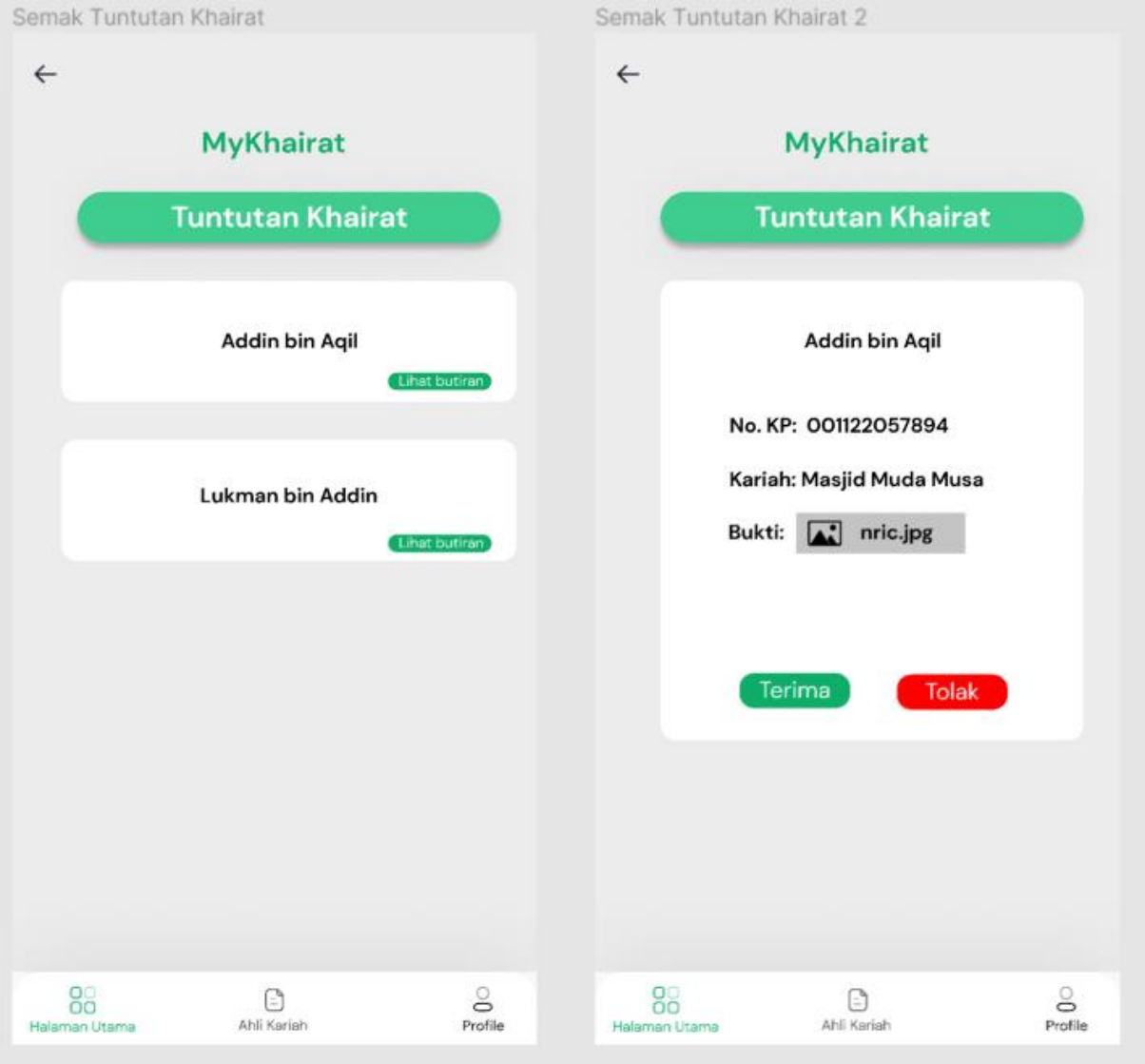



*Diagram 5.6.1.1 Approve Khairat Page*

In diagram 5.6.1.1 shows the prototype pages for the masjid committee to approve or reject claim khairat by members. The first interface will show the list of requests and the masjid committee needs to click the name box and the page will change to the second interface. On the second interface, the masjid committee needs to click terima to accept and click tolak to reject the claim khairat by member.

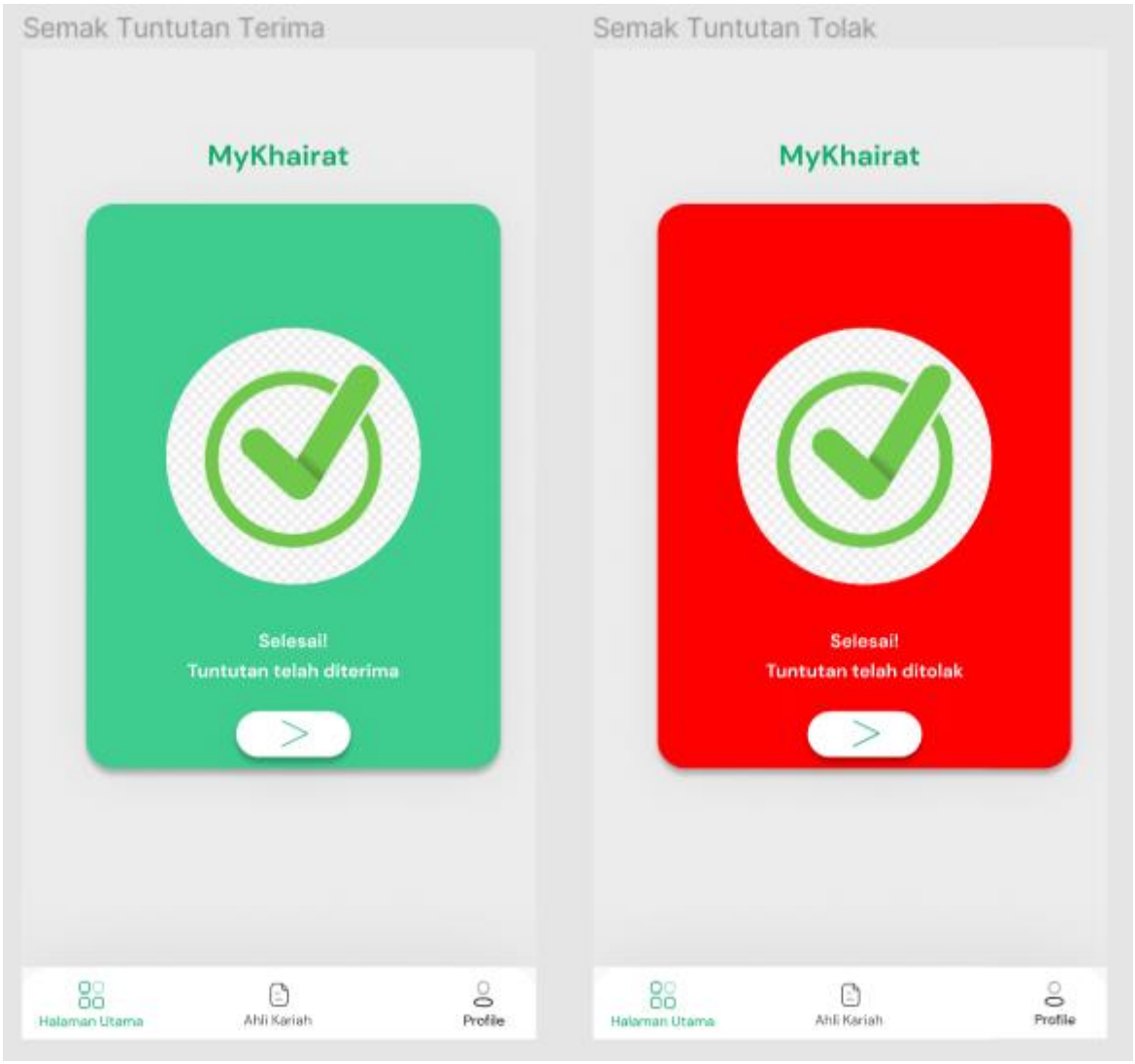



*Diagram 5.6.1.2 Approve Khairat Page*

In diagram 5.6.1.2 are the pages to notify the masjid committee. The green colour is to notify the masjid committee that the claim khairat already approves and the red colour is to notify the masjid committee that the claim khairat already rejects.

# 6 Architecture

## 6.1 Overview

Laravel is used as the framework for MyKhairat mobile application. Laravel is an open-source PHP framework that follows MVC architectural patterns. MVC is an abbreviation for “Model-View-Controller”. This architectural pattern divides the application into three logical parts; the model, the view, and the controller.

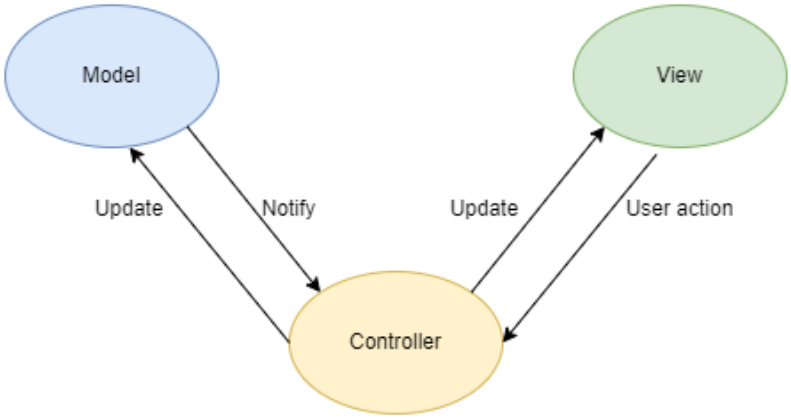



*Figure 6.1.1: MVC Architecture*

The components of MVC architecture are:

- Model: Handles data logic. It is responsible for maintaining data. Adding or retrieving data is done in the model component as it is connected to the database.
- View: Displays the information from the model to the user. It is responsible for data representation. It actually generates UI or user interface for the user.
- Controller: Acts as mediator between Model and View. It controls the data flow into a model object and updates the view whenever data changes.

# 7 Glossary

An ordered list of defined terms and concepts used throughout the document.

| Terms | Definitions |
| --- | --- |
| Android | Android is a mobile operating system based on a modified version of the Linux kernel and other open source software, designed primarily for touchscreen mobile devices such as smartphones and tablets. |
| Flutter | An open source framework by Google to create beautiful, natively compiled applications for mobile, web, and desktop from a single codebase. |
| Laravel | A PHP web application framework with expressive, elegant syntax. It eases common tasks (authentication, routing, etc.) used in web projects. |
| MySQL | An open source relational database management system. It creates a database for storing and manipulating data, defining the relationship of each table. |
| Agile | Agile methodology is a type of project management process, mainly used for software development, where demands and solutions evolve through the collaborative effort of self-organising and cross-functional teams and their customers. |

# Software Design Document

**for**

# Module 2 : MyKhairat Application Project

**Version 1.0 approved**

**Prepared by**

**Nor Syafiqa binti Abd Rahman (202422)**

**Safraa Khairunnisa binti Rahim (202375)**

**Nurul Aina binti Ariffin (204309)**

**Muhammad Aiman bin Mohd Rahimi (204235)**

**Mohamad Ramzuzzhini bin Mohamad Ramlee (201542)**

**Faculty of Computer Science And Information Technology**

**Universiti Putra Malaysia**

# Table of Contents

# Revision History

| Name | Date | Reason For Changes | Version |
|---|---|---|---|
| | | | |
| | | | |

# 1. Introduction

## 1.1. Purpose

This document describes and sets forward detailed design, database design, User-Interface(UI), and also the architecture of MyKhairat Application. This document will illustrate and provide the introduction, purpose, usage, and detail of the development concepts of the user module. This document is specifically focused on masjid committee functionalities, member profiles and their dependent information.

This design document has an accompanying SRS document of MyKhairat Application. It illustrates and provides the overall description, purpose, usage, and detail of the development concepts of the module.

## 1.2. Scope

This module of MyKhairat Application aims to facilitate the process of handling Kariah members' personal information and their dependents which include editing profile, viewing their dependent record along with grave location and reporting new death when one of the dependents died. The information provided by the Kariah members will be used for the mailing process and for distribution for the death benefit. Apart from that, the masjid committee can view the recent death record list of Kariah members' dependents and also make a quick search for Kariah members.

These functions are important as it will help the masjid committee to keep on track with the new reported death in the real-time and help the Kariah members to know their dependents' grave location easily. This application will enhance the existing management system making it more efficient, more organised and time effective.

## 1.3. Definition and Acronym

| Terms | Definitions |
|---|---|
| UI | User Interface |
| SRS | Software Requirements Specification |
| SDD | Software Design Document |

### 1.4. References

1. SRS of MyKhairat Application Project
2. "IEEE Recommended Practice for Software Design Descriptions," in IEEE Std 1016-1998 , vol., no., pp.1-23, 4 Dec. 1998, doi: 10.1109/IEEESTD.1998.88828.

# 2. Design Considerations

### 2.1. Assumptions and Dependencies

A1 - We assume that the users use a stable internet connection in order to access the application and its features.
A2 - We assume that the users have basic knowledge of technologies and know how to use MyKhairat without a user manual.
D1 - We depend on the servers to be running and functioning always.

### 2.2. Constraints

C1 - The mobile application should be finished, fulfilling all client's requirements, within the 14 weeks given.
C2 - The mobile application is displayed only in Malay language.
C3 - Most developers for MyKhairat never have experience building the application using Flutter with Laravel backend.

### 2.3. System Environment

OE1 - The mobile application is developed using Flutter with Laravel backend
OE2 - The database used for the application is MySQL

### 2.4. Design Methodology

The module is built based on the methodology of the Agile process model. In this model, there are few phases, for instance, Concept, Inception, Iteration, Release, Maintenance, and Retirement. This approach is suitable for our project since it can help to build an application in 2 or 3 months.

### 2.5. Risk and Volatile Areas

None.

# 3. Detailed Design

## 3.1. Use Case Diagram

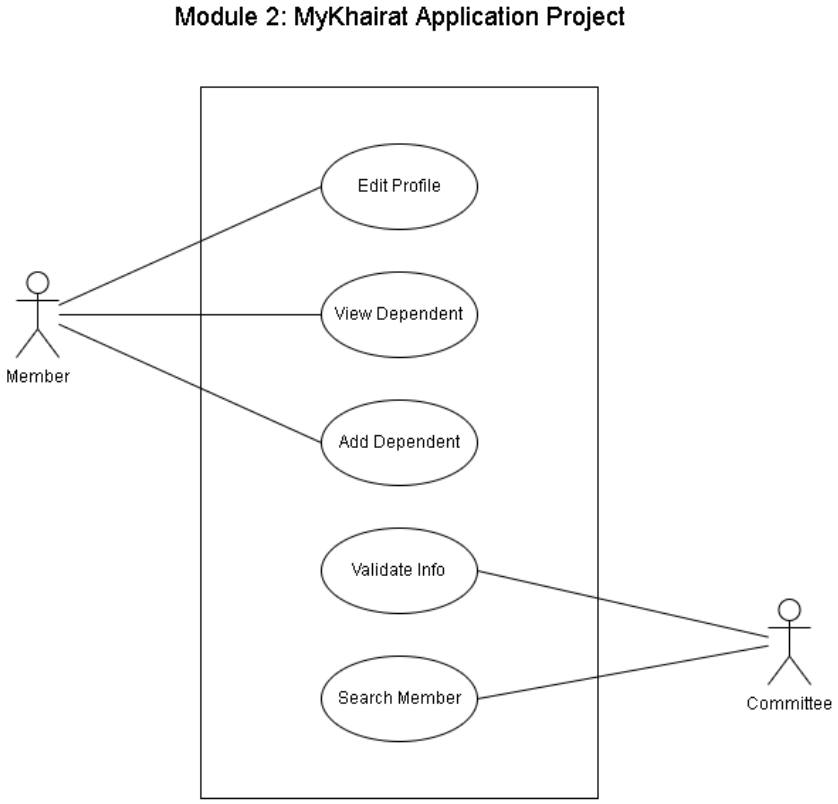



Diagram 3.1.1 - Use Case Diagram

## 3.2. Class Diagram

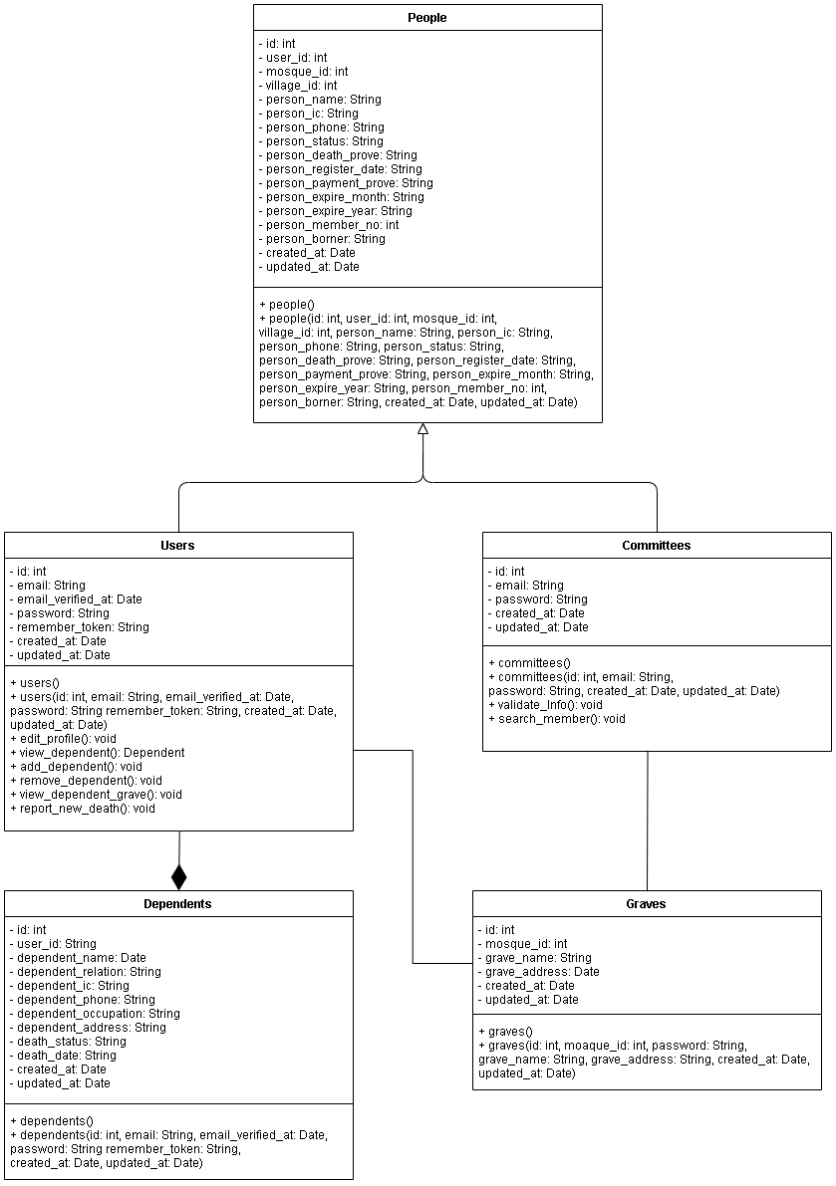



Diagram 3.2.1 - Class Diagram

### 3.2.1. People Class

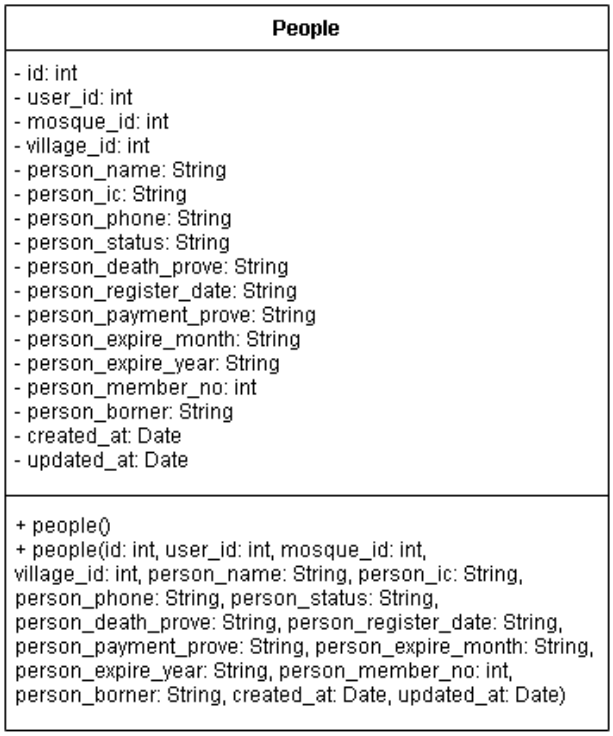



Diagram 3.2.1.1 - People Class

**a. Classification**

The class component consists of attributes that include private modifiers, attribute names, data types, and a list of methods.

**b. Definition**

The people class is a class that contains many attributes based on the requirements indicated in Diagram 3.2.1.1 above. This class has both no-arguments constructor and also constructor with parameters to initiate the attributes inside it.

**c. Responsibilities**

Diagram 3.2.1.1 shows the People class. As this is an entity class, it is responsible for creating people and storing the people's information.

3.2.1.1 Operation Specification

| | |
|---|---|
| Operation Specification: | people(id: int, user_id: int, mosque_id: int,<br>village_id: int, person_name: String,<br>person_ic: String,<br>person_phone: String, person_status: String, |

| | |
|---|---|
| | person_death_prove: String,<br>person_register_date: String,<br>person_payment_prove: String,<br>person_expire_month: String,<br>person_expire_year: String,<br>person_member_no: int,<br>person_borner: String, created_at: Date,<br>updated_at: Date) |
| Operation intent: | Creates a new people by entering and submitting training details parameters |
| Operation signature: | people:: people(id: int, user_id: int,<br>mosque_id: int,<br>village_id: int, person_name: String,<br>person_ic: String,<br>person_phone: String, person_status: String,<br>person_death_prove: String,<br>person_register_date: String,<br>person_payment_prove: String,<br>person_expire_month: String,<br>person_expire_year: String,<br>person_member_no: int,<br>person_borner: String, created_at: Date,<br>updated_at: Date) |
| Logic description<br><br>(Pre-Post Condition): | Context: people<br>Pre:<br>Post: display message “people successfully created” |
| Other operations called: | + people |
| Events transmitted to other objects: | None |

| | |
|---|---|
| Attribute set: | None |
| Response to exceptions: | None |
| Non-functional requirements: | Performance: This operation should take place in less than 2 minutes. |

### 3.2.2. Committee Class

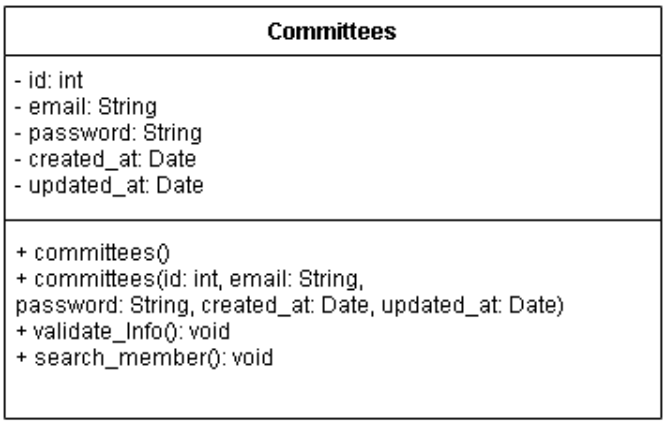



Diagram 3.2.2.1 - Committee Class

**a. Classification**

The class component consists of attributes that include private modifiers, attribute names, data types, and a list of methods.

**b. Definition**

The committee class is a class that contains many attributes based on the requirements indicated in Diagram 3.2.2.1 above. This class has a method where the committee can validate the dependent information and to search members of the mosque. Lastly, this class also has both no-arguments constructor and also constructor with parameters to initiate the attributes inside it.

**c. Responsibilities**

Diagram 3.2.3.1 shows the Committee class. As this is an entity class, it is responsible for creating committees, validating dependent information, searching members and storing the committees' information.

3.2.2.1 Operation Specification

| | |
|---|---|
| Operation Specification: | committees(id: int, email: String, password: String, created_at: Date, updated_at: Date) |
| Operation intent: | Creates a new committee by entering and submitting committee details parameters |
| Operation signature: | committees:: committees(id: int, email: String, password: String, created_at: Date, updated_at: Date) |
| Logic description<br><br>(Pre-Post Condition): | Context: committee |
| Other operations called: | + committee()<br>+ validate_info(): void<br>+ search_member() member |
| Events transmitted to other objects: | None |
| Attribute set: | None |
| Response to exceptions: | None |
| Non-functional requirements: | Performance: This operation should take place in less than 2 minutes. |

### 3.2.3. Dependent Class

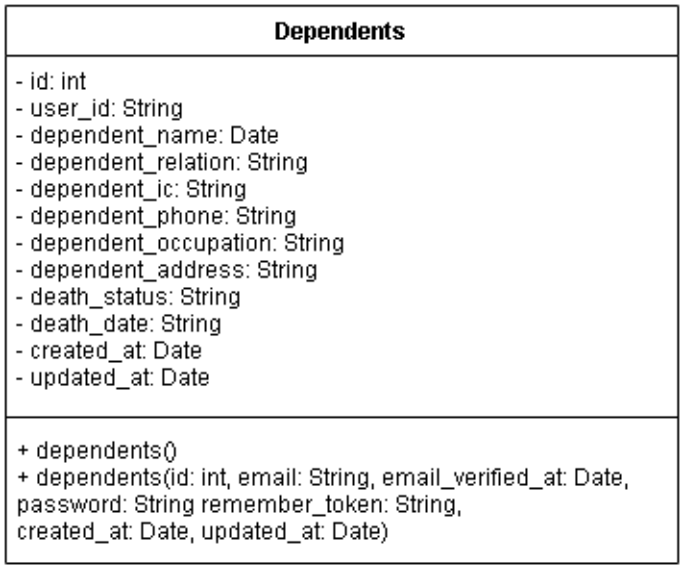



*Diagram 3.2.3.1 - Dependent Class*

#### a. Classification

The class component consists of attributes that include private modifiers, attribute names, data types, and a list of methods.

#### b. Definition

The Dependent class is a class containing numerous attributes based on the requirements specified in Diagram 3.2.3.1. There are some methods in the Dependent Class that explain the behaviour of this class. This class has both no-args constructor and constructor with parameters to initiate the attributes inside it.

#### c. Responsibilities

Diagram 3.2.3.1 shows the Dependent class. As this is an entity class, it is responsible for creating dependents, editing dependents and storing the dependent's information.

3.2.3.1 Operation Specification

| | |
|---|---|
| Operation Specification: | Dependent(id: bigint, user_id: bigint, dependent_name: varchar, |

| | |
|---|---|
| | dependent_relation: varchar, dependent_ic: varchar, dependent_phone: varchar, depedent_occupation: varchar, dependent_date: varchar, death_status: varchar, death_date: varchar, created_at: timestamp, updated_at: timestamp) |
| Operation intent: | Creates a new dependent by inputting and storing dependent details parameters |
| Operation signature: | Dependent:: Dependent(id: bigint, user_id: bigint, dependent_name: varchar, dependent_relation: varchar, dependent_ic: varchar, dependent_phone: varchar, depedent_occupation: varchar, dependent_date: varchar, death_status: varchar, death_date: varchar, created_at: timestamp, updated_at: timestamp) |
| Logic description<br><br>(Pre-Post Condition): | **Context:** Dependent<br><br>**Pre:** None<br><br>**Post:** None |
| Other operations called: | **context:** +Dependent() |
| Events transmitted to other objects: | None |
| Attribute set: | None |
| Response to exceptions: | None |

| | |
|---|---|
| Non-functional requirements: | Performance: This operation should take place in less than 1 minute. |

### 3.2.4. Graves Class

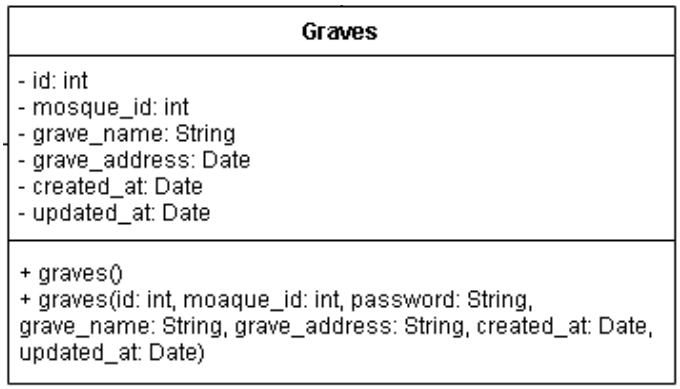



*Diagram 3.2.4.1 - Grave Class*

**a. Classification**

The class component consists of attributes that include private modifiers, attribute names, data types, and a list of methods.

**b. Definition**

The Graves class is a class containing numerous attributes based on the requirements specified in Diagram 3.2.4.1. There are some methods in the Graves Class that explain the behaviour of this class. This class has both no-args constructor and constructor with parameters to initiate the attributes inside it.

**c. Responsibilities**

Diagram 3.2.3.1 shows the Graves class. As this is an entity class, it is responsible for storing the grave's information.

3.2.4.1 Operation Specification

| | |
|---|---|
| Operation Specification: | Graves(id: int, mosque_id: int, password: String, grave_name: String, grave_address: String, created_at: Date, updated_at: Date) |
| Operation intent: | Creates a new grave object by inputting and storing graves details parameters |
| Operation signature: | Graves:: Graves(id: int, mosque_id: int, password: String, grave_name: String, grave_address: String, created_at: Date, updated_at: Date) |
| Logic description<br><br>(Pre-Post Condition): | **Context:** Graves<br><br>**Pre:** None<br><br>**Post:** None |
| Other operations called: | None |
| Events transmitted to other objects: | None |
| Attribute set: | None |
| Response to exceptions: | None |
| Non-functional requirements: | Performance: This operation should take place in less than 1 minute. |

# 4. Database Design

This section describes the database design for the MyKhairat mobile application.

## 4.1. Data Model

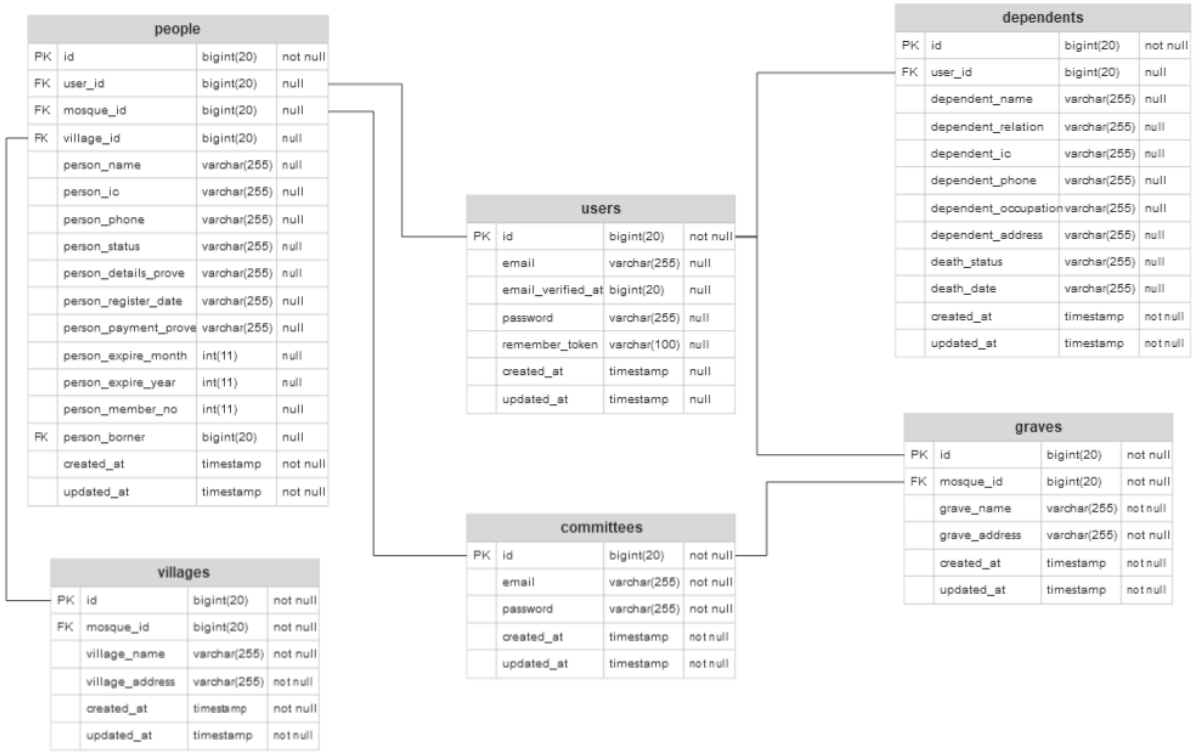



*Diagram 4.1.1 - Relational Table Mapping*

In Diagram 4.1.1 shows the data involved within this module. The relational table is based on the class diagram and details the attributes. The database is already in 3rd Normal Form (3NF). It is listed with the name of the attributes - (e.g id), the data type (e.g. bigint), primary key (PK) and foreign key (FK), and the constraint of the attribute (either null or not).

## 4.2. Data Dictionary

Lists all tables for the relational database.

### 4.2.1 Dependents

| Attribute's Name | Data Type | Constraint | Description |
|---|---|---|---|

| | | | |
|---|---|---|---|
| id | bigint(20) | PRIMARY KEY(NOT NULL) | Unique ID for the dependent |
| user_id | bigint(20) | FOREIGN KEY(NULL) | Unique ID of member used as reference to determine that dependent are under specific member |
| dependent_name | varchar(255) | NULL | Name of the dependent |
| dependent_relation | varchar(255) | NULL | Relationship of the dependent to a member |
| dependent_ic | varchar(255) | NULL | IC number of the dependent |
| dependent_phone | varchar(255) | NULL | Phone number of the dependent |
| depedent_occupation | varchar(255) | NULL | Occupation of the dependent |
| dependent_address | varchar(255) | NULL | Address of the dependent |
| death_status | varchar(255) | NULL | Status of the dependent's wellbeing |
| death_date | varchar(255) | NULL | Date of the dependent's death |
| created_at | timestamp | NOT NULL | Time at which it is created |
| updated_at | timestamp | NOT NULL | Time at which it is updated |

Table 4.2.1 - Relation Table for Dependents

### 4.2.2 People

| **Attribute's Name** | **Data Type** | **Constraint** | **Description** |
|---|---|---|---|
| id | bigint(20) | PRIMARY KEY(NOT NULL) | Unique ID that is auto-increment to avoid error |
| user_id | bigint(20) | FOREIGN KEY(NULL) | Unique ID if it is a member |
| mosque_id | bigint(20) | FOREIGN KEY(NULL) | Unique ID if it is the member's mosque |
| village_id | bigint(20) | FOREIGN KEY(NULL) | Unique ID if it is the member's village |
| person_name | varchar(255) | NULL | Name of the person |
| person_ic | varchar(255) | NULL | IC number of the person |
| person_phone | varchar(255) | NULL | Phone number of the person |
| person_occupation | varchar(255) | NULL | Occupation of the person |
| person_status | varchar(255) | NULL | The status of the person |
| person_details_prove | varchar(255) | NULL | Proof of the person's details |
| person_register_date | varchar(255) | NULL | Date of the person's registration |

| | | | |
|---|---|---|---|
| person_payment_prove | varchar(255) | NULL | Proof of the person's payment |
| person_expire_month | int(11) | NULL | Month of expiration |
| person_expire_year | int(11) | NULL | Year of expiration |
| person_member_no | int(11) | NULL | Number of the member |
| person_borner | bigint(20) | NULL | Number for join table |
| created_at | timestamp | NOT NULL | The time at which it is created |
| updated_at | timestamp | NOT NULL | The time at which it is updated |

Table 4.2.2 - Relation Table for Dependents

### 4.2.3 Committee

| **Attribute's Name** | **Data Type** | **Constraint** | **Description** |
|---|---|---|---|
| id | bigint(20) | PRIMARY KEY(NOT NULL) | Unique ID for the mosque committee |
| email | varchar(255) | NOT NULL | Email of the committee member |
| password | varchar(255) | NOT NULL | Password of the committee member |
| created_at | timestamp | NOT NULL | Time at which it is created |
| updated_at | timestamp | NOT NULL | Time at which it is updated |

Table 4.2.3 - Relation Table for Committee

### 4.2.4 Graves

| **Attribute's Name** | **Data Type** | **Constraint** | **Description** |
|---|---|---|---|
| id | bigint(20) | PRIMARY KEY(NOT NULL) | Grave ID |
| mosque_id | bigint(20) | FOREIGN KEY(NOT NULL) | Unique ID if it is in a mosque |
| grave_name | varchar(255) | NOT NULL | Name of the grave |
| grave_address | varchar(255) | NOT NULL | Address of the dependent's grave |
| created_at | timestamp | NOT NULL | Time at which it is created |
| updated_at | timestamp | NOT NULL | Time at which it is updated |

Table 4.2.4 - Relation Table for Graves

### 4.2.5 Village

| **Attribute's Name** | **Data Type** | **Constraint** | **Description** |
|---|---|---|---|
| id | bigint(20) | PRIMARY KEY(NOT NULL) | Village ID |
| mosque_id | bigint(20) | FOREIGN KEY(NOT NULL) | Unique ID if it has a mosque |
| village_name | varchar(255) | NOT NULL | Name of the village |
| village_address | varchar(255) | NOT NULL | Address of the village |

| | | | |
|---|---|---|---|
| created_at | timestamp | NOT NULL | Time at which it is created |
| updated_at | timestamp | NOT NULL | Time at which it is updated |

Table 4.2.5 - Relation Table for Village

# 5. User Interface Design

## 5.1. Use case: Edit Profile

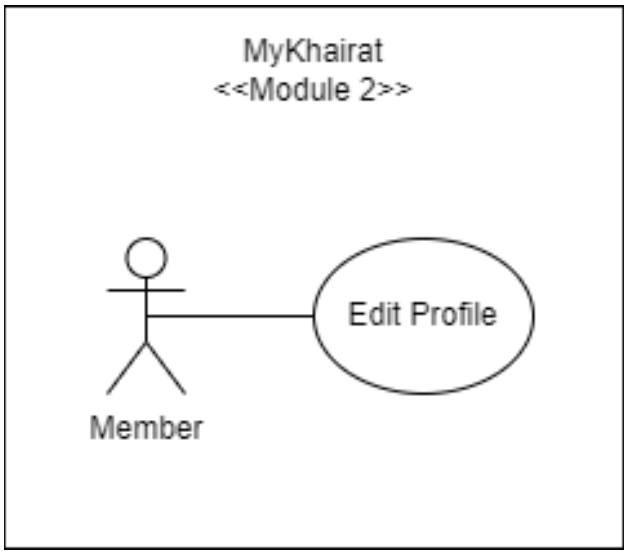



*Diagram 5.1.1 - Edit Profile Use Case*

In Diagram 5.1.1, the Edit Profile use case will involve the member and allow them to edit their profile details if they wish to change anything.

### 5.1.1. Sequence Diagram Entity Approach

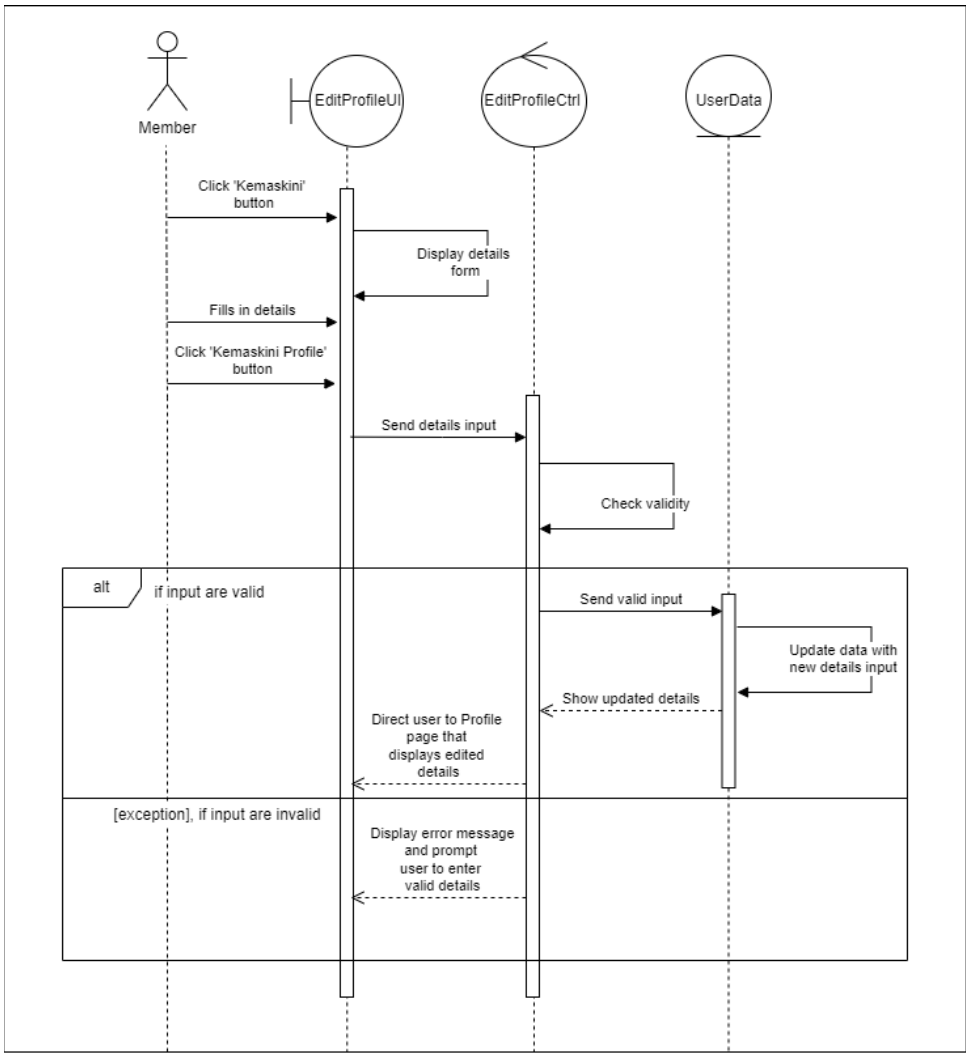



*Diagram 5.1.1.1 - Sequence Diagram for Edit Profile*

### 5.1.2. User Interface Prototype

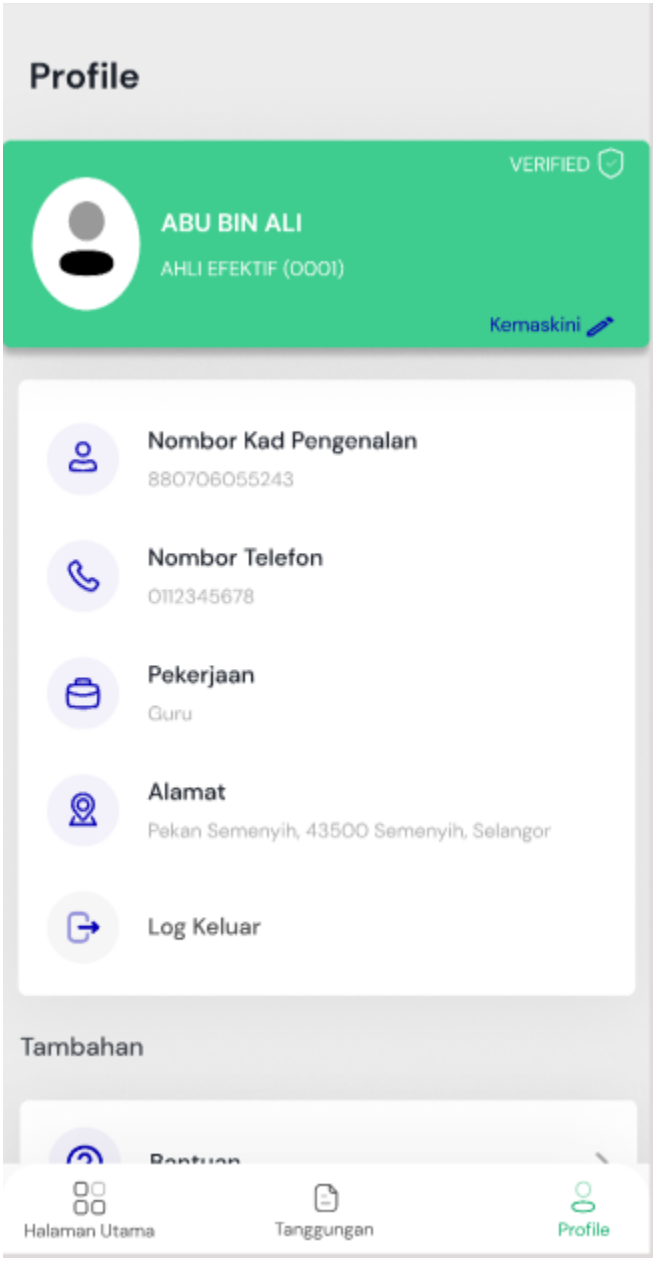



*Diagram 5.1.2.1 - Profile Page (Member)*

In Diagram 5.1.2.1, it shows the user interface of all the member's profile details. The button 'Kemaskini' is for the member to edit/update their profile information.

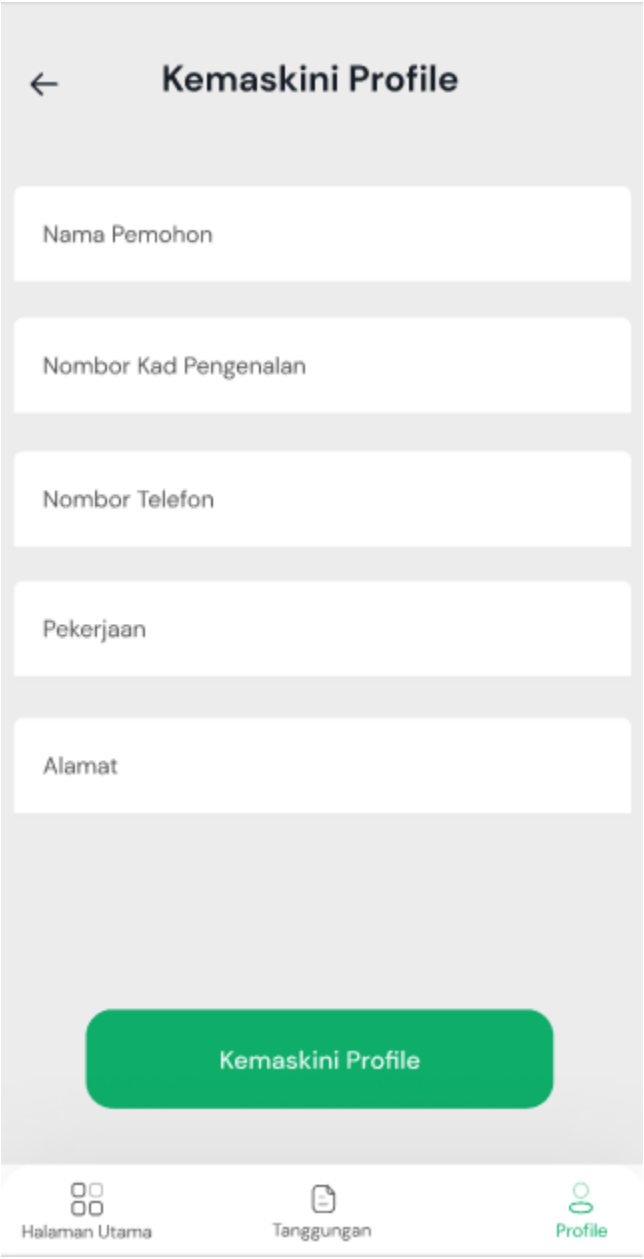



*Diagram 5.1.2.2 - Edit Profile Page*

In Diagram 5.1.2.2, it will show a page where the member can input information about themselves. Once they are done editing, they can save by pressing the 'Kemaskini Profile' button.

## 5.2. Use case: View Dependent Record

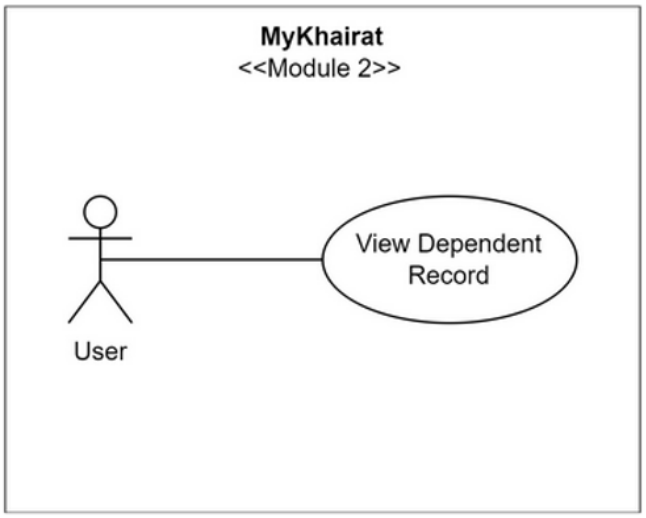



*Diagram 5.2.1 - View Dependent Record Use Case*

In Diagram 5.2.1, the member can view their dependent's record, from a list of dependent records, and read the details.

### 5.2.1. Sequence Diagram Entity Approach

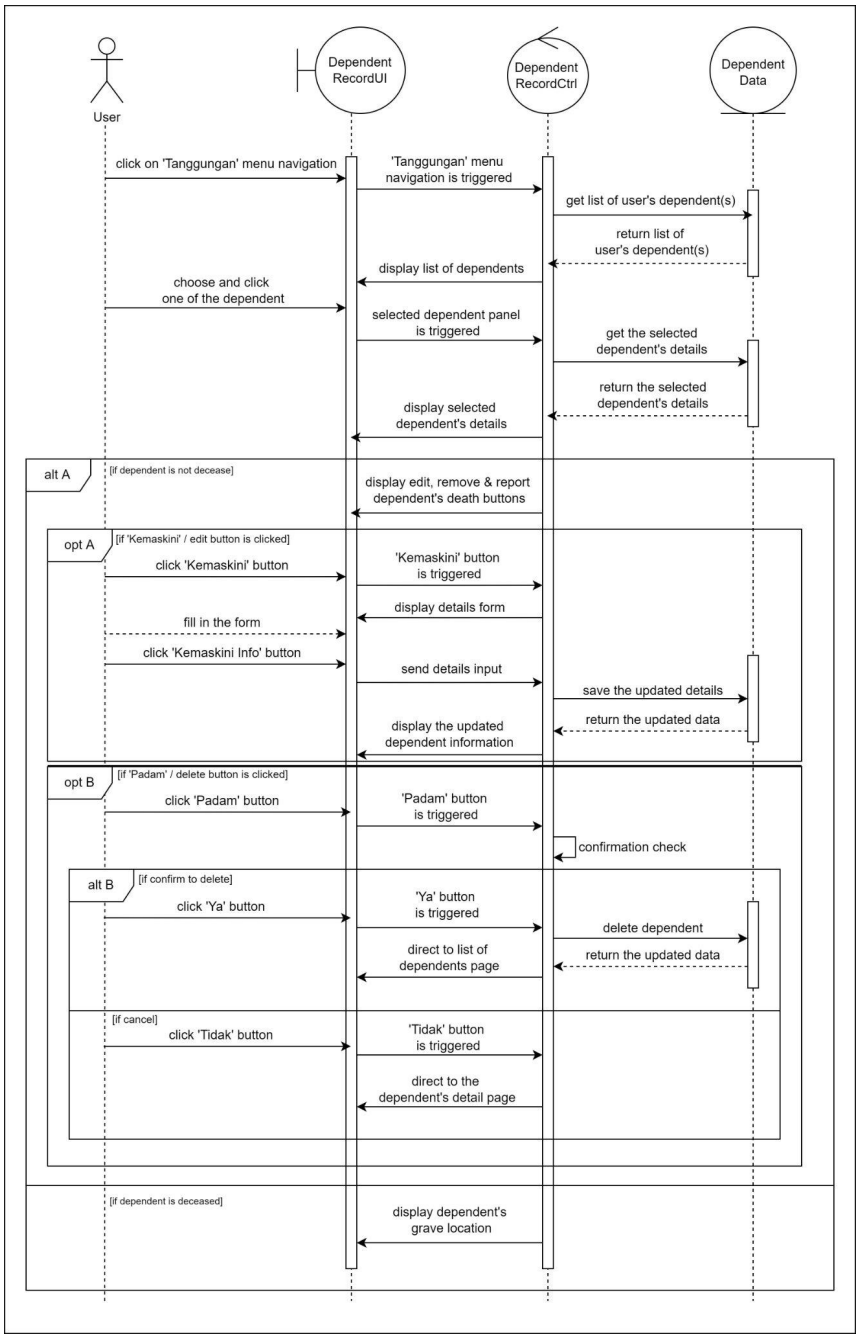



*Diagram 5.2.1.1 - Sequence Diagram for View Dependent Record*

### 5.2.2. User Interface Prototype

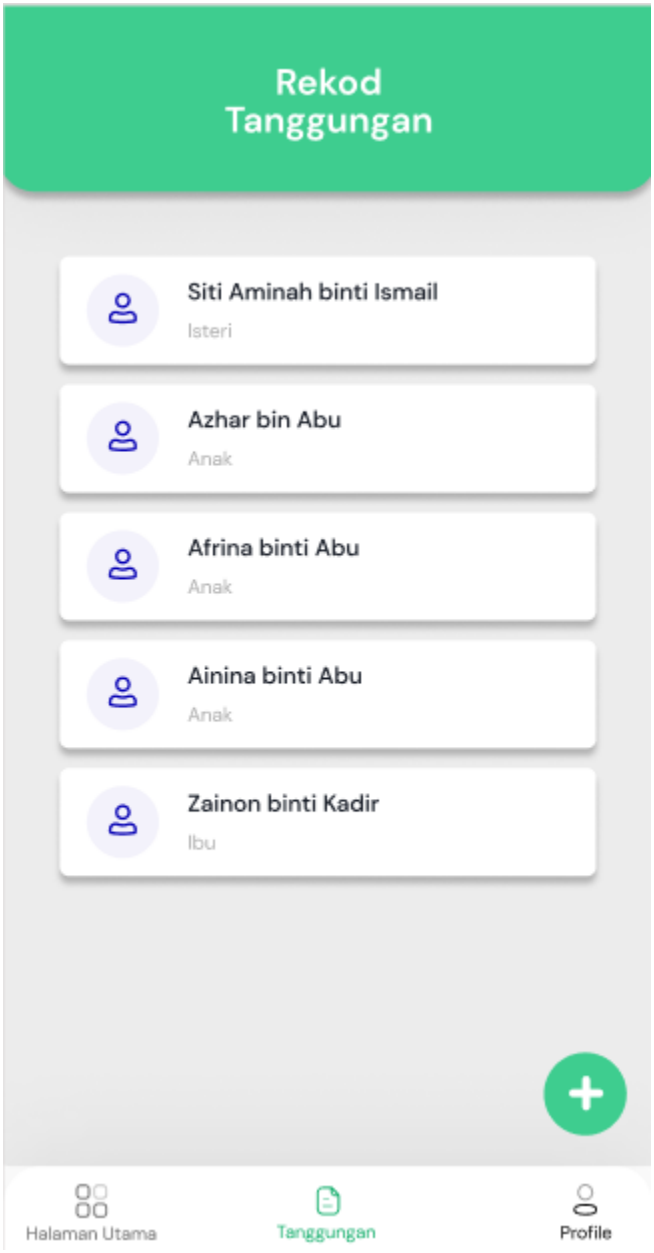



*Diagram 5.2.2.1 - Dependent Records*

In Diagram 5.2.2.1, the interface will show all the records of the dependents that the member has added and submitted. To view the dependent's record, the member has to press on their record.

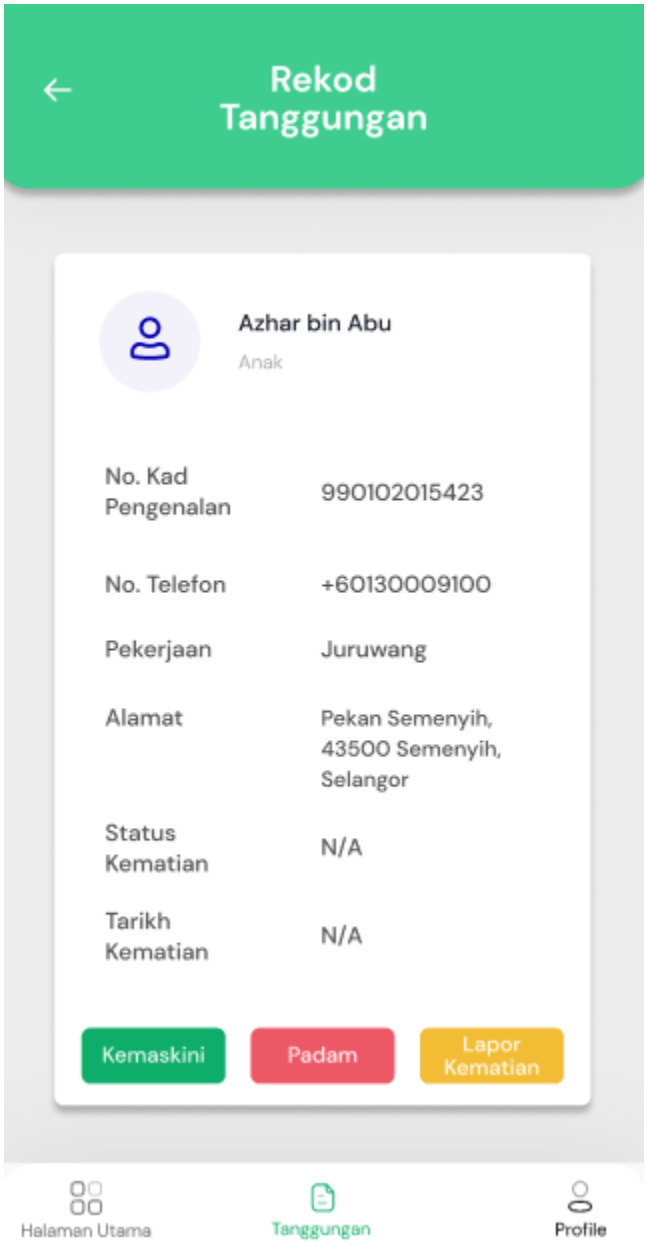



*Diagram 5.2.2.2 - Living Dependent's Record*

In Diagram 5.2.2.2, it shows the record of a dependent who is still living. Options for the record are below it, either to edit and delete the record as well as reporting the death of the dependent.

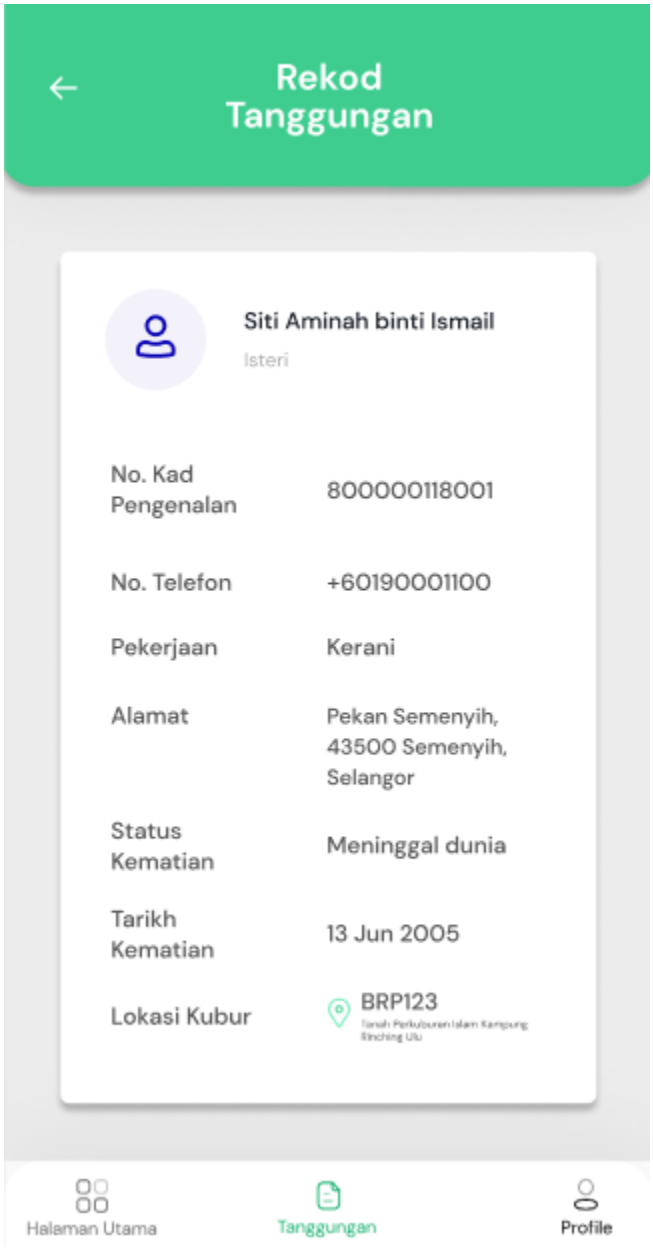



*Diagram 5.2.2.3 - Deceased Dependent's Record*

In Diagram 5.2.2.3, it shows the record of a dependent who has passed away. All their details are shown, including the grave location which can be shown if pressed on.

## 5.3. Use case: Add Dependent

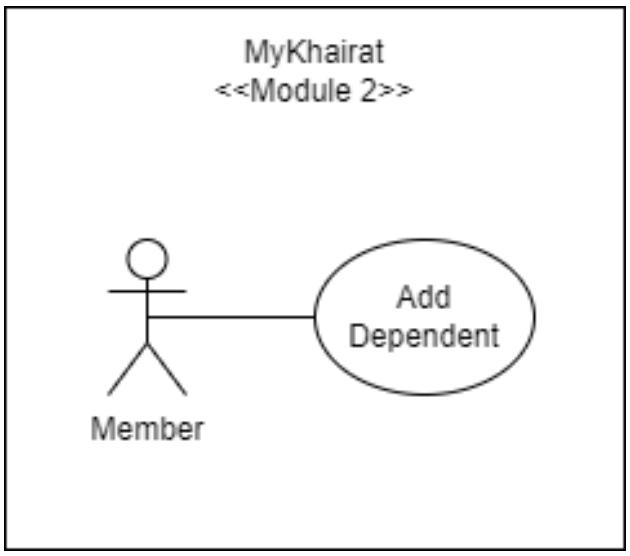



*Diagram 5.3.1 - Add Dependent Use Case*

In Diagram 5.3.1, the member can add a new dependent to their account. It will need to wait to be validated by the committee before confirming it.

### 5.3.1. Sequence Diagram Entity Approach

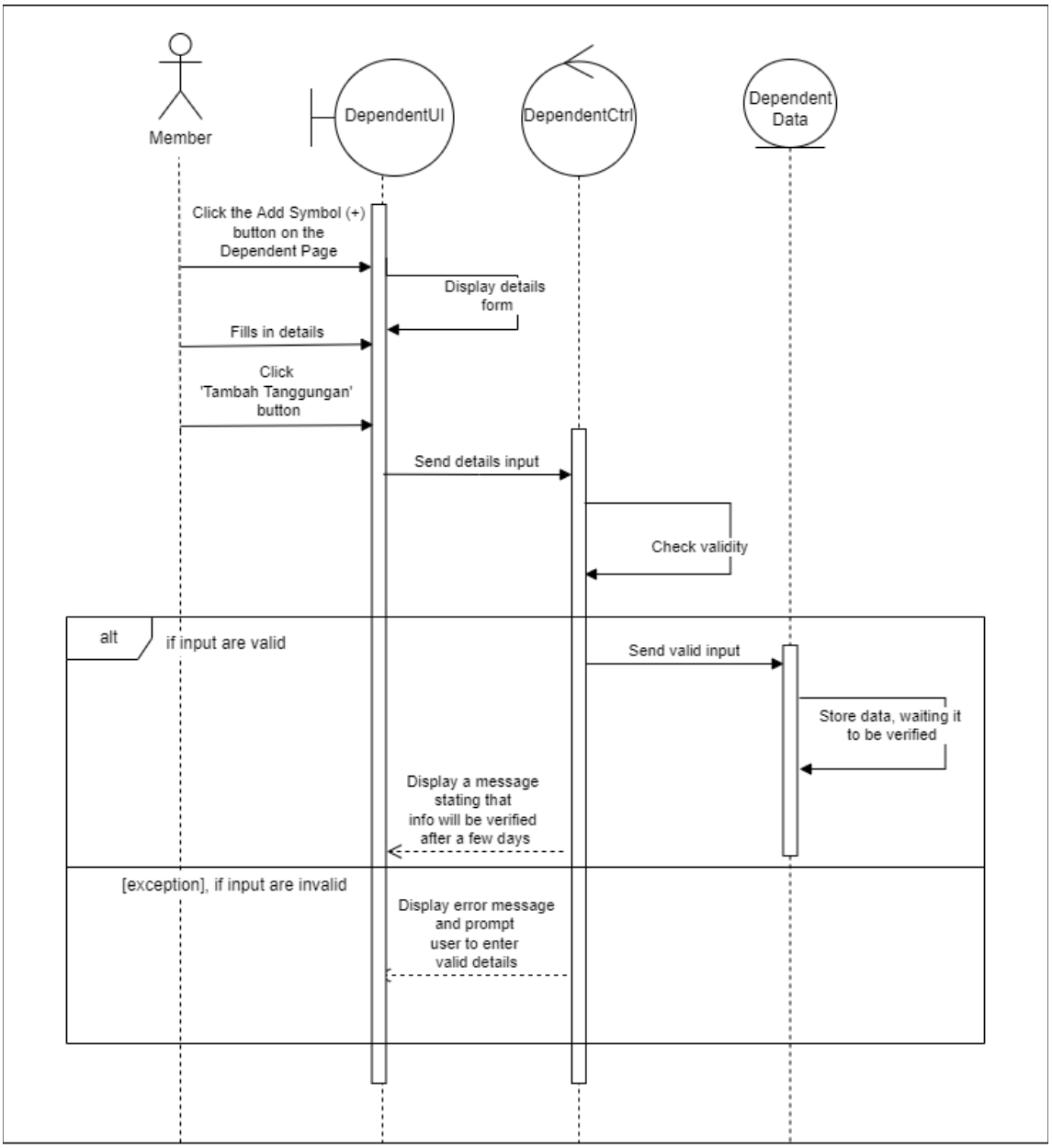



*Diagram 5.3.1.1 - Sequence Diagram for Add Dependent*

### 5.3.2. User Interface Prototype

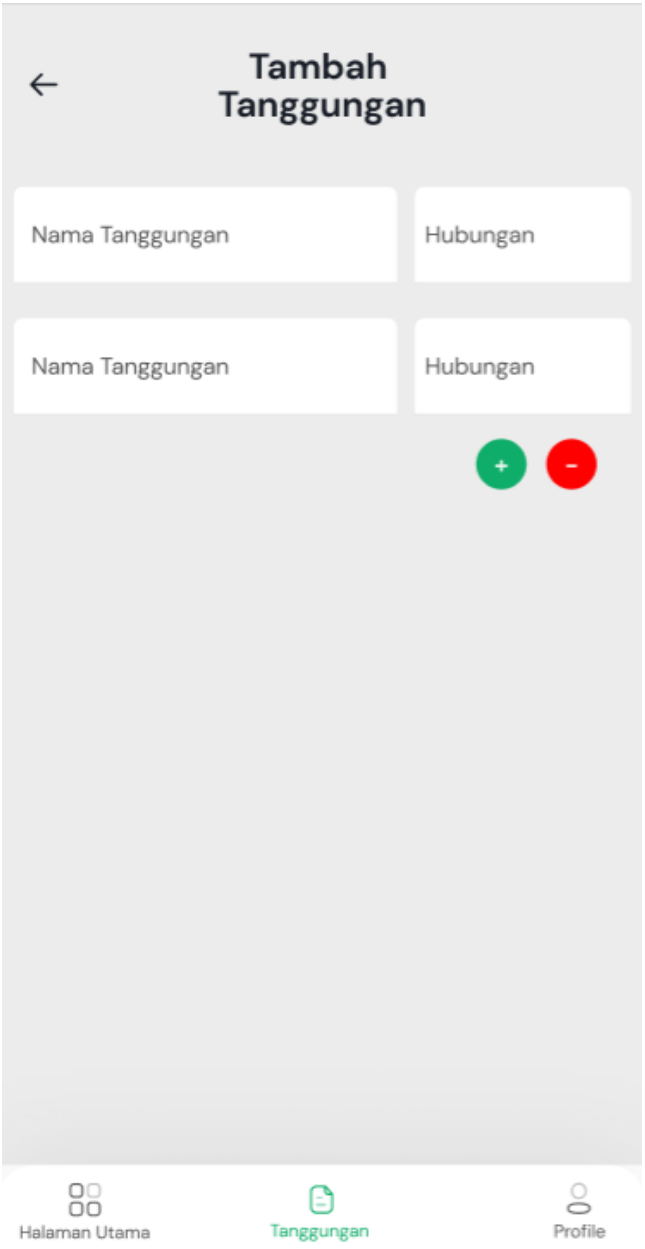



*Diagram 5.3.2.1 - Add Dependent*

In Diagram 5.3.2.1, the interface shows where new dependents can be added in by pressing the green '+' button.

## 5.4. Use case: Validate Info

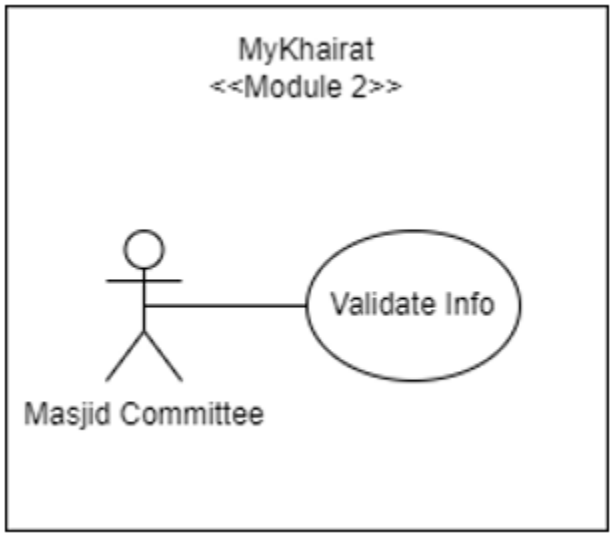



*Diagram 5.4.1 - Validate Info Use Case*

In Diagram 5.4.1, a committee member can view a list of new dependents, submitted by members, and can either accept the dependent before being added into the database or reject it if it doesn't seem suitable.

### 5.4.1. Sequence Diagram Entity Approach

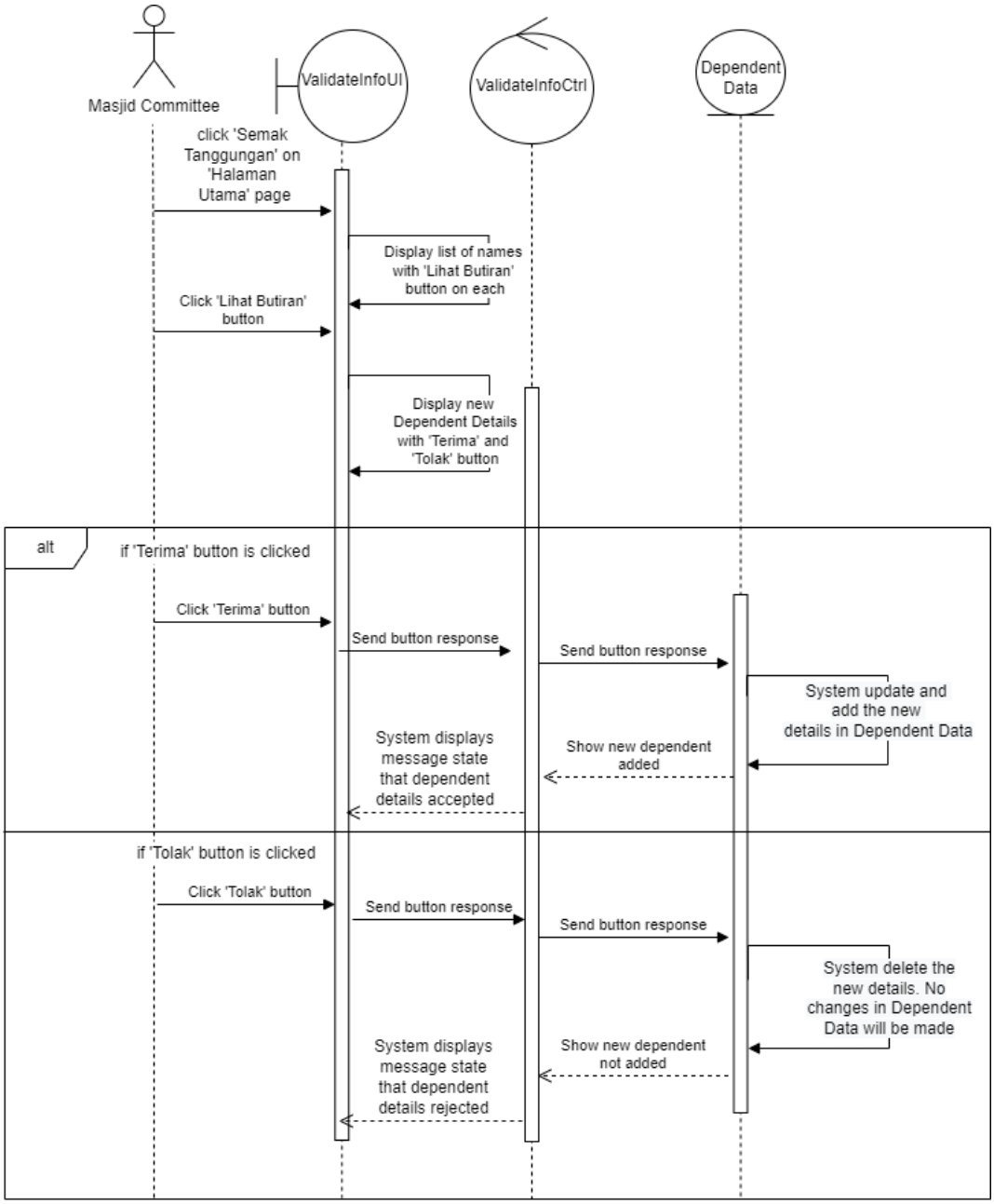



*Diagram 5.4.1.1 - Sequence Diagram for Validate Info*

### 5.4.2. User Interface Prototype

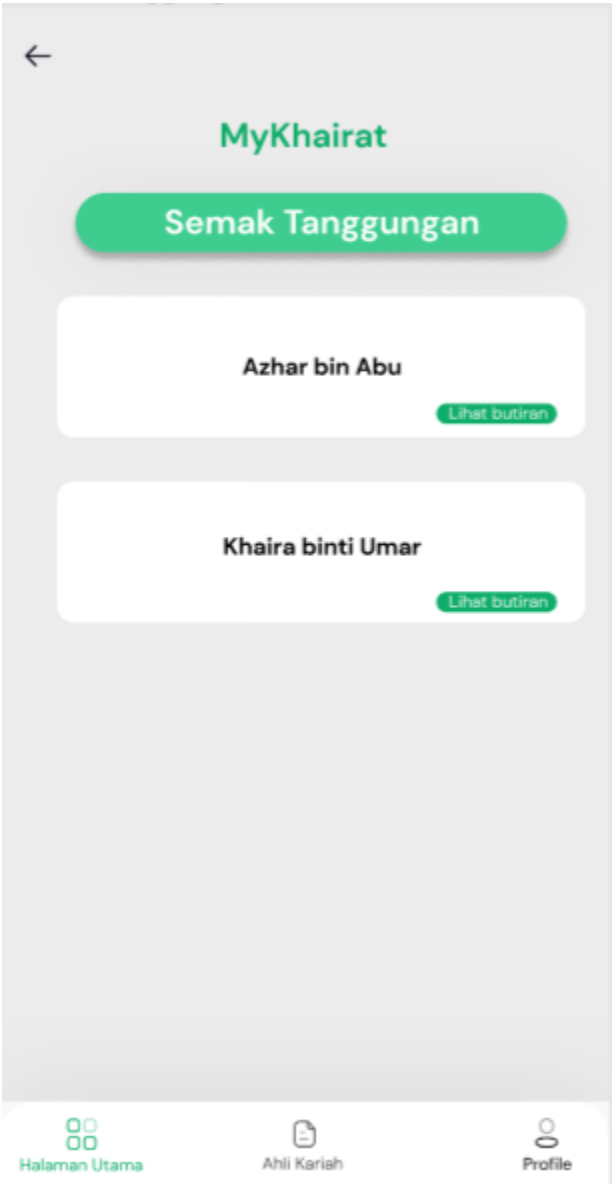



*Diagram 5.4.2.1 - Validate Dependents List*

In Diagram 5.4.2.1, the list of dependents that needs validation are shown. Pressing the 'Lihat Butiran' button will show the dependent's details.

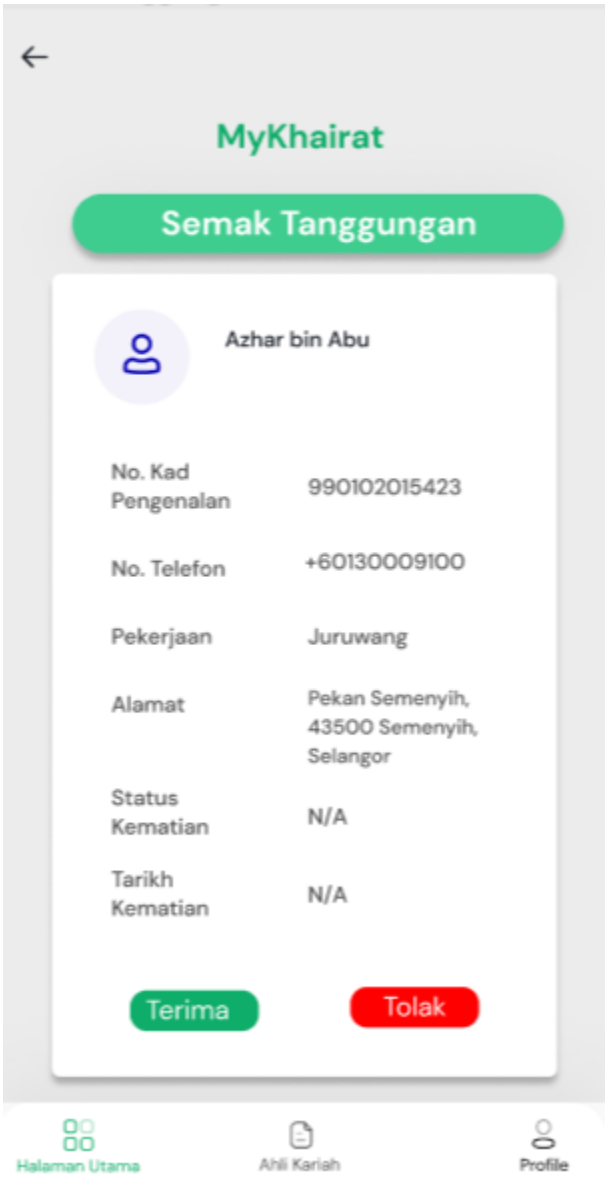



*Diagram 5.4.2.2 - New Dependent's Validation*

In Diagram 5.4.2.2, it shows the dependent's record details. There are two buttons to either accept the dependent (green) or reject the dependent (red).

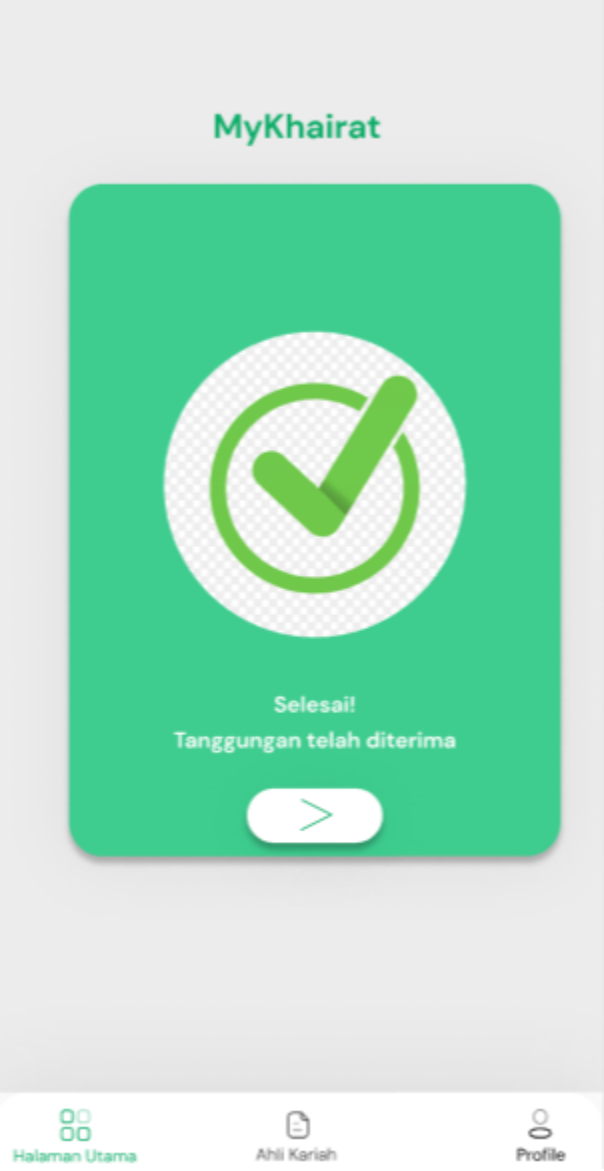



*Diagram 5.4.2.3 - Dependent Accepted*

In Diagram 5.4.2.3, the interface will show this green notification after accepting the dependent's record. Pressing the button below will lead back to the list again.

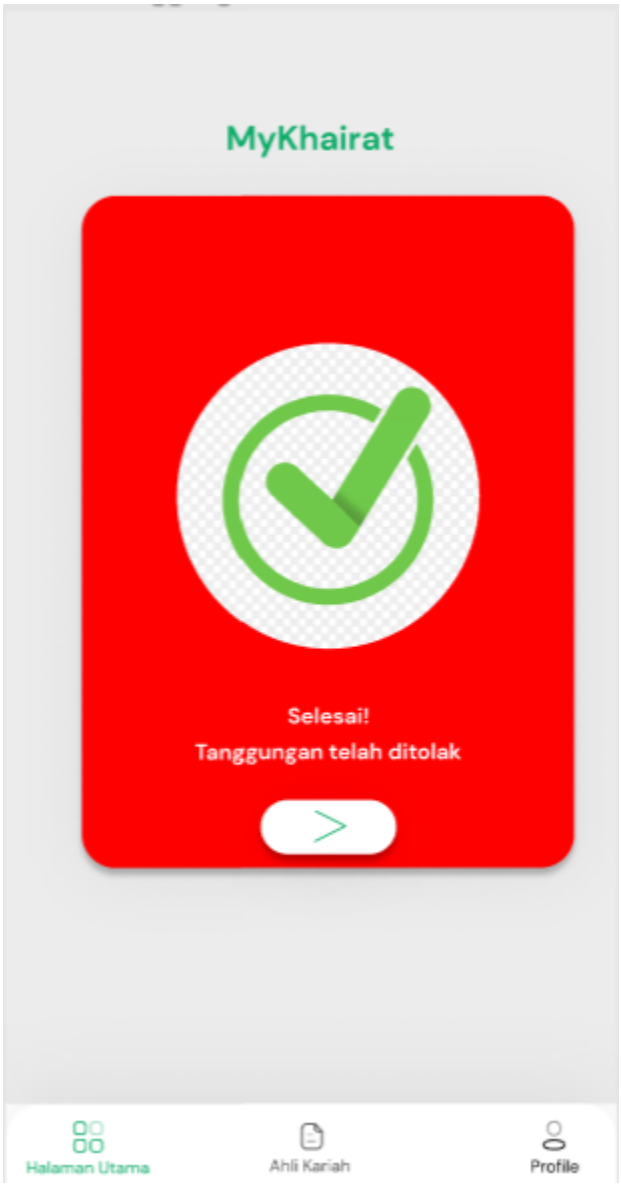



*Diagram 5.4.2.4 - Dependent Rejected*

In Diagram 5.4.2.4, the interface will show this red notification after rejecting the dependent's record. Pressing the button below will lead back to the list again.

## 5.5. Use case: Search Member

### 5.5.1. Sequence Diagram Entity Approach

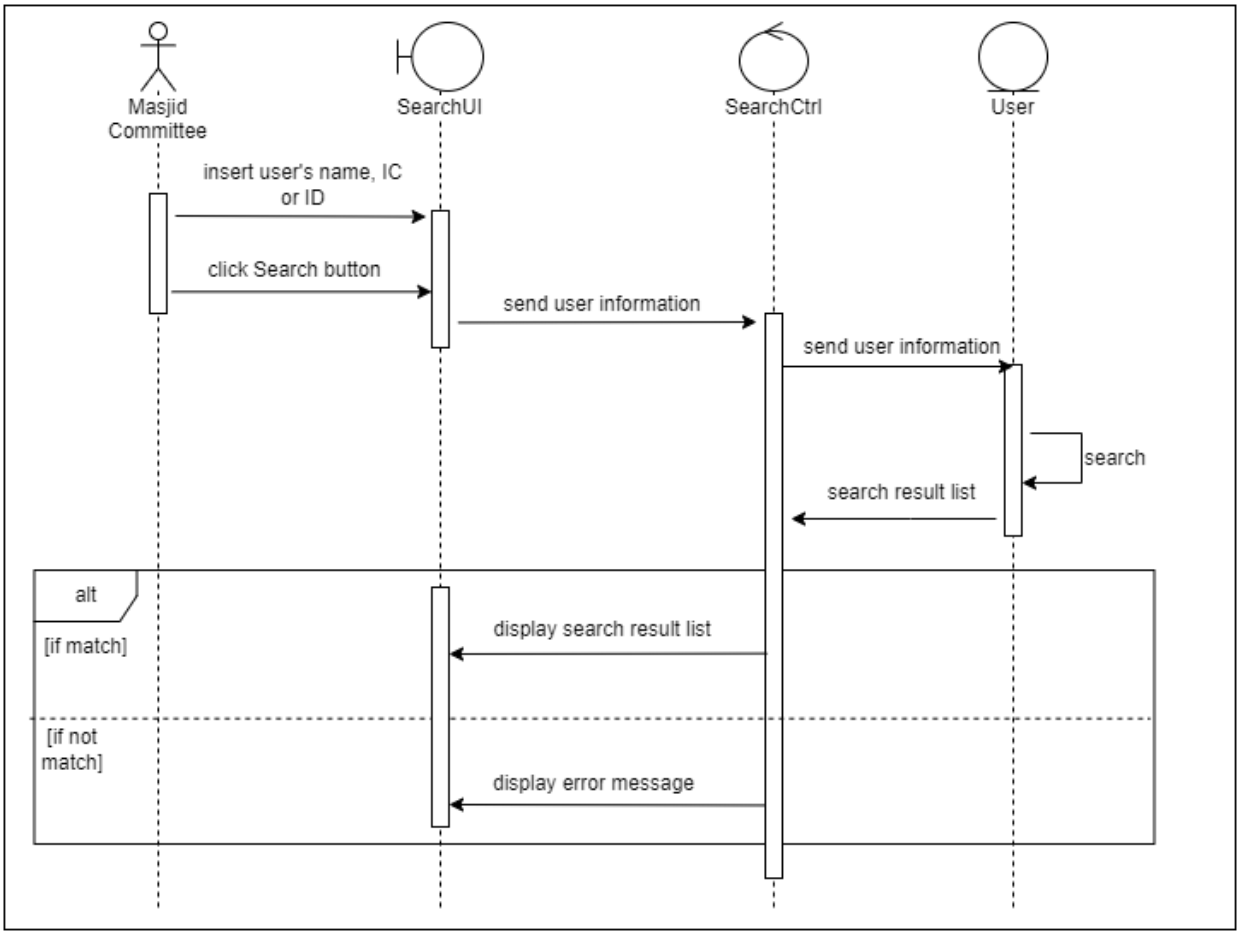



*Diagram 5.5.1.1 - Sequence Diagram for Search Members*

### 5.5.2. User Interface Prototype

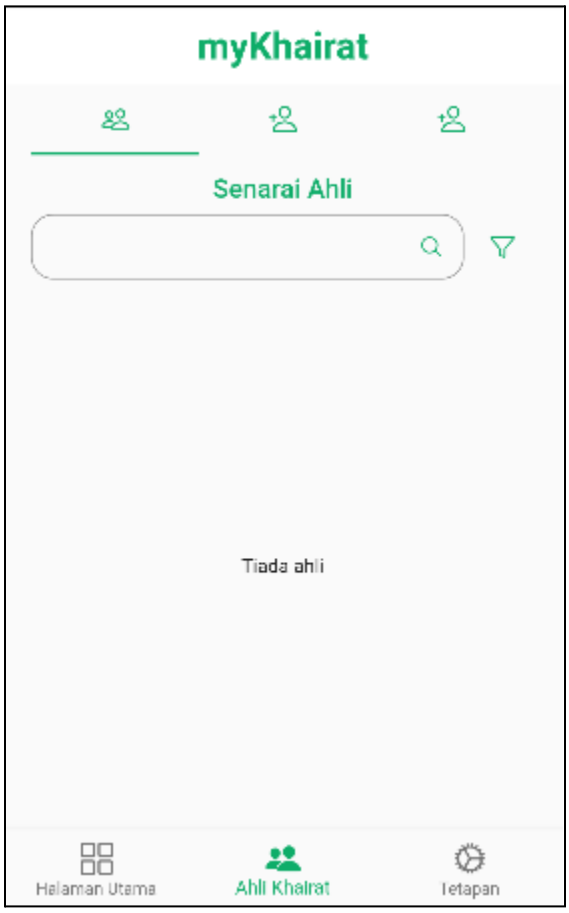



*Diagram 5.5.2.1. - Search Member*

In Diagram 5.5.2.1, it shows the Khairat Member lists (if exists) and with a search bar. It also has a filter icon button to filter the Khairat members with expired dates.

# 6. Architecture

## 6.1. Overview

Laravel is used as the framework for MyKhairat mobile application. Laravel is an open-source PHP framework that follows MVC architectural patterns. MVC is an abbreviation for "Model-View-Controller". This architectural pattern divides the application into three logical parts; the model, the view, and the controller.

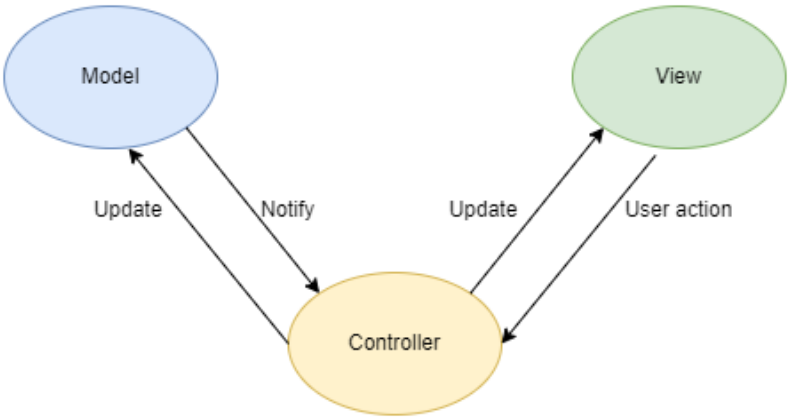



*Diagram 6.1.1: MVC Architecture*

The components of MVC architecture are:

- Model: Handles data logic. It is responsible for maintaining data. Adding or retrieving data is done in the model component as it is connected to the database.
- View: Displays the information from the model to the user. It is responsible for data representation. It actually generates UI or user interface for the user.
- Controller: Acts as mediator between Model and View. It controls the data flow into a model object and updates the view whenever data changes.

## 6.2. Deployment Diagram

This section details the execution architecture design of Module 2 using a deployment diagram.

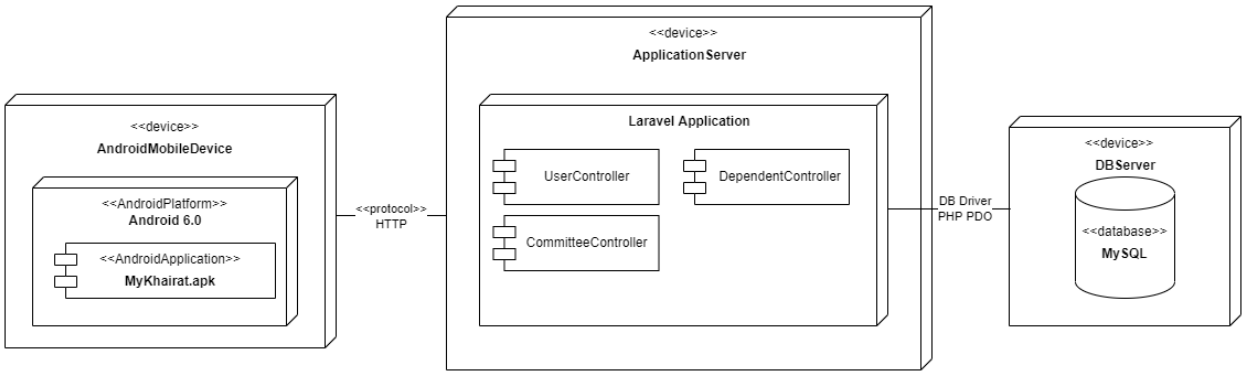



*Diagram 6.2.1 Deployment diagram for MyKhairat mobile application (Module 2)*

Diagram 6.2.1 represents the deployment diagram of MyKhairat. Starting from the left side of the diagram, the first node depicts the client's Android mobile device. The second node is the application server that handles user requests to which they send the outcome the user wants. The data in the database will be retrieved by the application server through the database drivers in order to send the users the suitable and relevant data they need.

## 6.3. Package Diagram

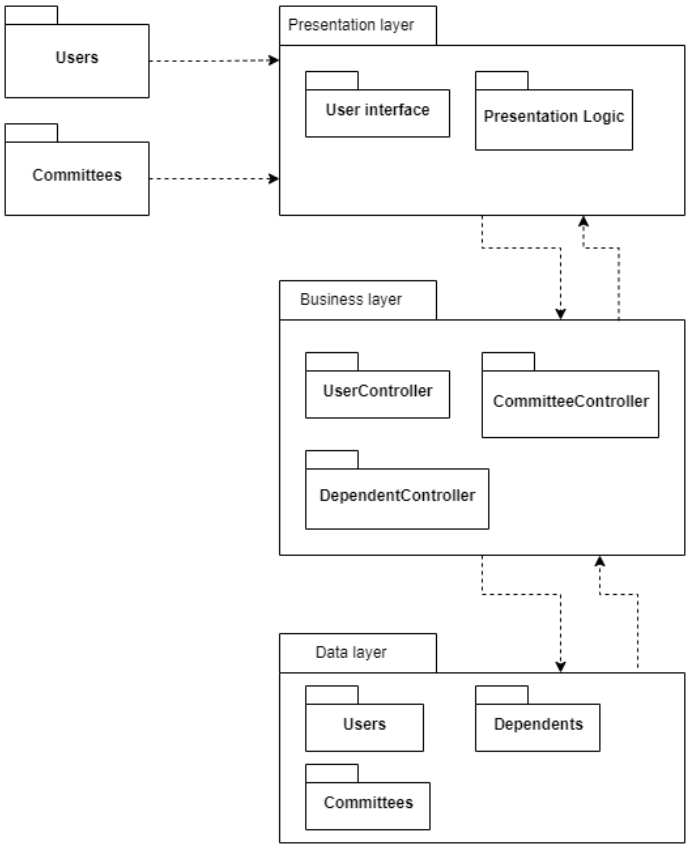



*Diagram 6.3.1: Package Diagram for Module 2 of MyKhairat*

Diagram 6.3.1 shows the package diagram for module 2 of MyKhairat mobile application. This package model illustrates the interaction between the presentation layer, business layer and data layer of the application implying to client-server architecture. Presentation layer displays the user interface and facilitates user interaction with the system. Business layer handles the functions of the application. The workflows by which the data and requests travel through the back end are encoded in a business layer. The data layer manages the storage and retrieval of application's data.

## 6.4. Strategy

Design justification:

a) MVC architectural pattern

The main advantage of MVC is separation of concern. In computer science, separation of concerns is a design principle for separating a computer program into distinct sections such that each section addresses a separate concern, in this case - Model, View, and Controller. The maintenance of applications can be made easier by separation of concerns due to the lack of duplication and singularity of purpose of the individual components. The minimal overlapping

between the functions also eases the development and testing process as it increases code reusability and readability.

b) Client-server architecture

Client-server architecture is used because of its efficiency in delivering resources to the client and also requires low-cost maintenance. It also can enhance the security as servers have better control and access of resources to ensure that only authorised clients can access or manipulate the application's data. Since client-server architecture is a distributed model representing dispersed responsibilities of distinct functions, it is an advantage in terms of maintenance. The encapsulation makes it easy to replace, repair, upgrade, and relocate a server without affecting the clients.

# 7. Glossary

An ordered list of defined terms and concepts used throughout the document.

| Terms | Definitions |
|---|---|
| Android | Android is a mobile operating system based on a modified version of the Linux kernel and other open source software, designed primarily for touchscreen mobile devices such as smartphones and tablets. |
| Flutter | An open source framework by Google to create beautiful, natively compiled applications for mobile, web, and desktop from a single codebase. |
| Laravel | A PHP web application framework with expressive, elegant syntax. It eases common tasks (authentication, routing, etc.) used in web projects. |
| MySQL | An open source relational database management system. It creates a database for storing and manipulating data, defining the relationship of each table. |
| Agile | Agile methodology is a type of project management process, mainly used for software development, where demands and solutions evolve through the collaborative effort of self-organising and cross-functional teams and their customers. |

# Software Design Document

**for**

# Module 3 : MyKhairat Application Project

**Version 1.0 approved**

**Prepared by**

**Muhammad Aiman Hakim bin Azahari (200903)**

**Haris Lukman bin Muhammad Isror (202810)**

**Nur Suraiya binti Mustapha (204233)**

**Fadi Fuad Radman Abdulrab (199967)**

**Al-Sharafi Abdulwahid Taleb Gubran (205063)**

**Faculty of Computer Science And Information Technology**

**Universiti Putra Malaysia**

# Table of Contents

# Revision History

| Name | Date | Reason For Changes | Version |
|---|---|---|---|
| | | | |
| | | | |

# 1. Introduction

## 1.1. Purpose

This document describes and sets forward detailed design, database design, User-Interface(UI), and also the architecture of MyKhairat Application. This document will illustrate and provide the introduction, purpose, usage, and detail of the development concepts of the user module. This document is specifically focused on user profiles and their dependent information.

This design document has an accompanying SRS document of MyKhairat Application. It illustrates and provides the overall description, purpose, usage, and detail of the development concepts of the module.

## 1.2. Scope

This module of MyKhairat Application aims to facilitate the process of handling Kariah members' personal information and their dependents which include editing profile, viewing their dependent record along with grave location and reporting new death when one of the dependents died. The information provided by the Kariah members will be used for the mailing process and for distribution for the death benefit. Apart from that, the masjid committee can view the recent death record list of Kariah members' dependents and also make a quick search for Kariah members.

These functions are important as it will help the masjid committee to keep on track with the new reported death in the real-time and help the Kariah members to know their dependents' grave location easily. This application will enhance the existing management system making it more efficient, more organised and time effective.

### 1.3. Definition and Acronym

| Terms | Definitions |
|---|---|
| UI | User Interface |
| SRS | Software Requirements Specification |
| SDD | Software Design Document |

### 1.4. References


1. SRS of MyKhairat Application Project
2. "IEEE Recommended Practice for Software Design Descriptions," in IEEE Std 1016-1998 , vol., no., pp.1-23, 4 Dec. 1998, doi: 10.1109/IEEESTD.1998.88828.


# 2. Design Considerations

### 2.1. Assumptions and Dependencies

A1 - We assume that the users use a stable internet connection in order to access the application and its features.
A2 - We assume that the users have basic knowledge of technologies and know how to use MyKhairat without a user manual.
A3 - We assume that the users and masjid committee have access to Android devices with Android 7 and above.
D1 - We depend on the servers to be running and functioning always.

## 2.2. Constraints

C1 - The mobile application should be finished, fulfilling all client's requirements, within the 14 weeks given.
C2 - The mobile application is displayed only in Malay language.
C3 - Most developers for MyKhairat never have experience building the application using Flutter with Laravel backend.

## 2.3. System Environment

OE1 - The mobile application is developed using Flutter with Laravel backend
OE2 - The database used for the application is MySQL

## 2.4. Design Methodology

The module is built based on the methodology of the Agile process model. In this model, there are few phases, for instance, Concept, Inception, Iteration, Release, Maintenance, and Retirement. This approach is suitable for our project since it can help to build an application in 2 or 3 month.

## 2.5. Risk and Volatile Areas

None.

# 3. Detailed Design

## 3.1. Use Case Diagram

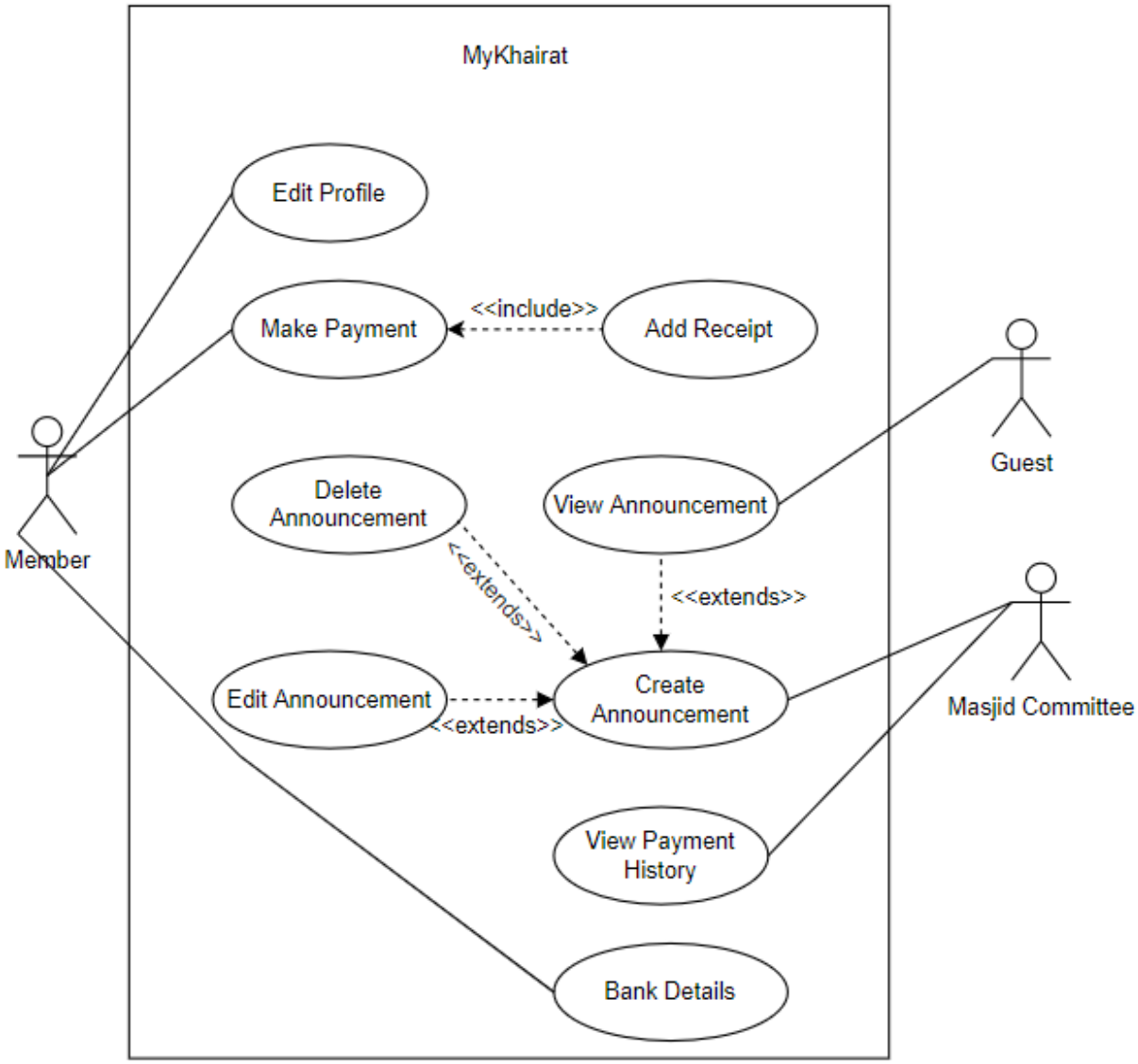



Diagram 3.1.1 - Use Case Diagram

## 3.2. Class Diagram

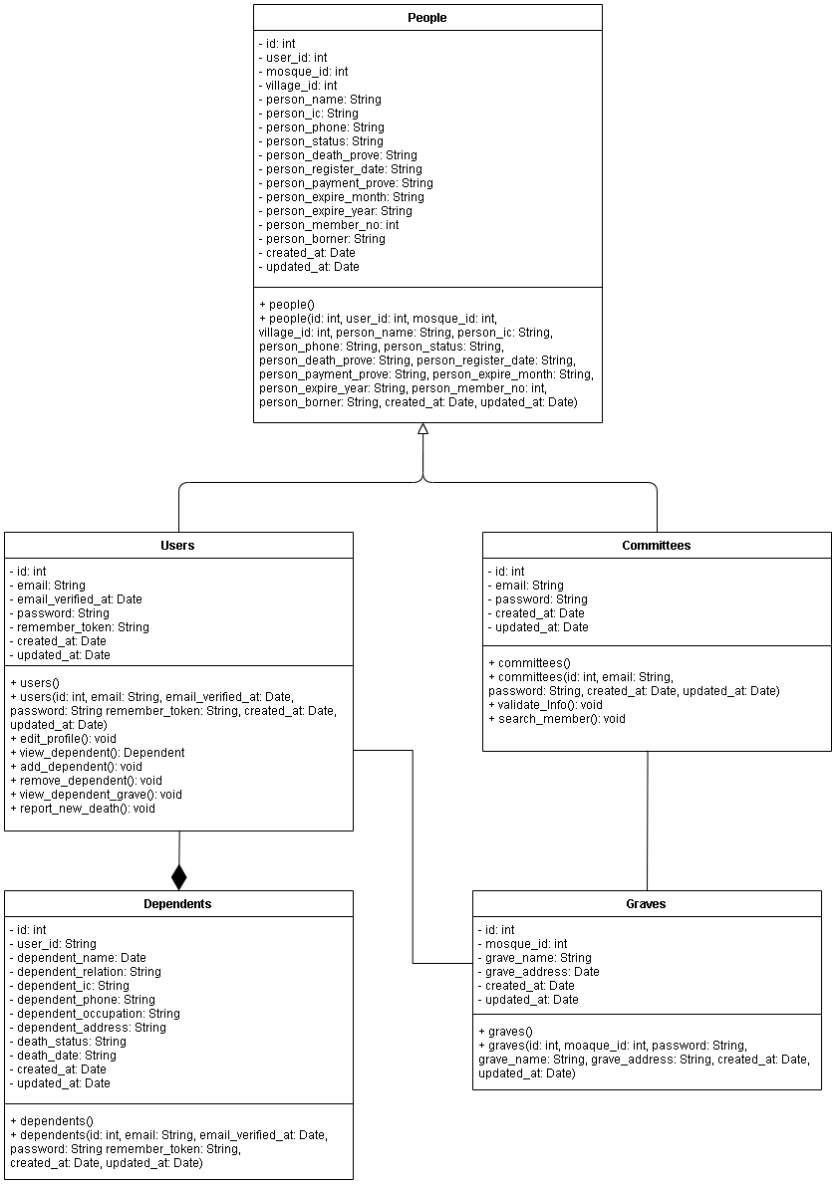



Diagram 3.2.1 - Class Diagram

### 3.2.1. People Class

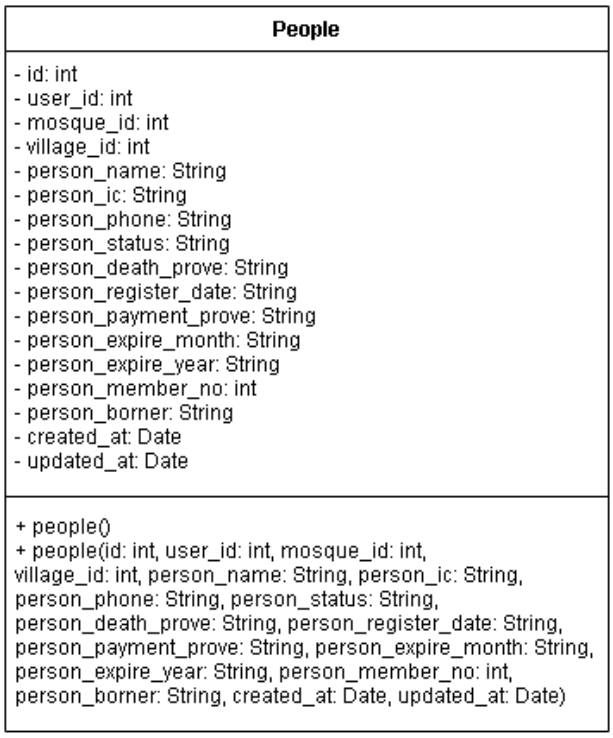



Diagram 3.2.1.1 - People Class

**a. Classification**

The class component consists of attributes that include private modifiers, attribute names, data types, and a list of methods.

**b. Definition**

The people class is a class that contains many attributes based on the requirements indicated in Diagram 3.2.1.1 above. This class has both no-arguments constructor and also constructor with parameters to initiate the attributes inside it.

**c. Responsibilities**

Diagram 3.2.1.1 shows the People class. As this is an entity class, it is responsible for creating people and storing the people's information.

3.2.1.1 Operation Specification

| | |
|---|---|
| Operation Specification: | people(id: int, user_id: int, mosque_id: int,<br>village_id: int, person_name: String,<br>person_ic: String,<br>person_phone: String, person_status: String,<br>person_death_prove: String,<br>person_register_date: String,<br>person_payment_prove: String,<br>person_expire_month: String,<br>person_expire_year: String,<br>person_member_no: int,<br>person_borner: String, created_at: Date, updated_at: Date) |
| Operation intent: | Creates a new people by entering and submitting training details parameters |
| Operation signature: | people:: people(id: int, user_id: int, mosque_id: int,<br>village_id: int, person_name: String,<br>person_ic: String,<br>person_phone: String, person_status: String,<br>person_death_prove: String,<br>person_register_date: String,<br>person_payment_prove: String,<br>person_expire_month: String,<br>person_expire_year: String,<br>person_member_no: int, |

| | |
|---|---|
| | person_borner: String, created_at: Date, updated_at: Date) |
| Logic description<br><br>(Pre-Post Condition): | Context: people<br>Pre:<br>Post: display message “people successfully created” |
| Other operations called: | + people |
| Events transmitted to other objects: | None |
| Attribute set: | None |
| Response to exceptions: | None |
| Non-functional requirements: | Performance: This operation should take place in less than 2 minutes. |

### 3.2.2. Committee Class

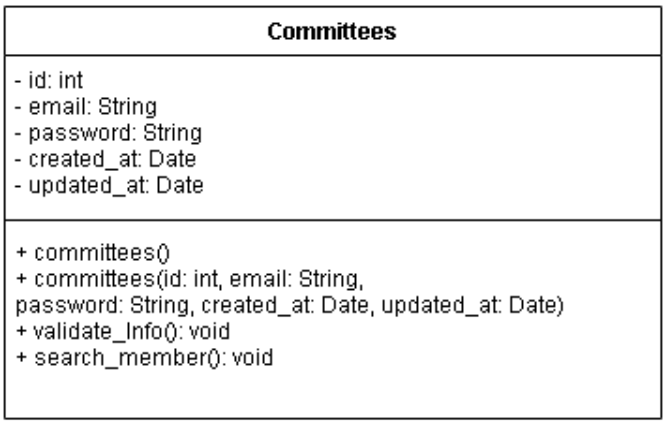



Diagram 3.2.2.1 - Committee Class

**a. Classification**

The class component consists of attributes that include private modifiers, attribute names, data types, and a list of methods.

**b. Definition**

The committee class is a class that contains many attributes based on the requirements indicated in Diagram 3.2.2.1 above. This class has a method where the committee can validate the dependent information and to search members of the mosque. Lastly, this class also has both no-arguments constructor and also constructor with parameters to initiate the attributes inside it.

**c. Responsibilities**

Diagram 3.2.3.1 shows the Committee class. As this is an entity class, it is responsible for creating committees, validating dependent information, searching members and storing the committees' information.

3.2.2.1 Operation Specification

| | |
|---|---|
| Operation Specification: | committees(id: int, email: String, |

| | |
|---|---|
| | password: String, created_at: Date, updated_at: Date) |
| Operation intent: | Creates a new committee by entering and submitting committee details parameters |
| Operation signature: | committees:: committees(id: int, email: String,<br>password: String, created_at: Date, updated_at: Date) |
| Logic description<br><br>(Pre-Post Condition): | Context: committee |
| Other operations called: | + committee()<br>+ validate_info(): void<br>+ search_member() member |
| Events transmitted to other objects: | None |
| Attribute set: | None |
| Response to exceptions: | None |
| Non-functional requirements: | Performance: This operation should take place in less than 2 minutes. |

### 3.2.3. Dependent Class

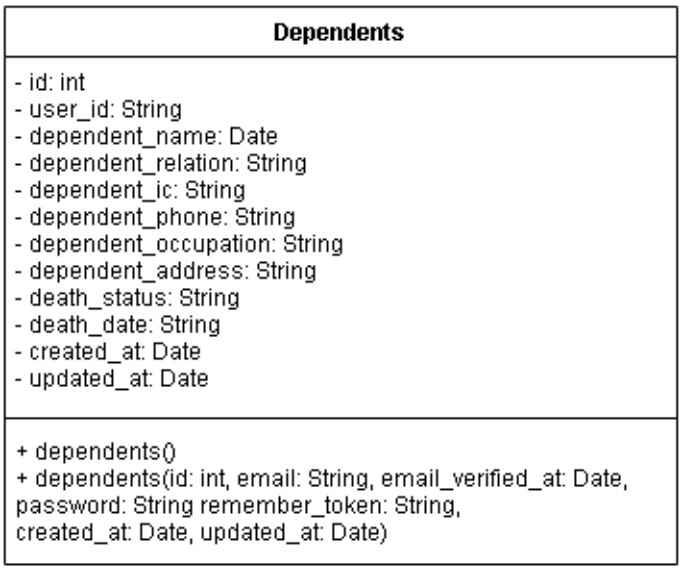



*Diagram 3.2.3.1 - Dependent Class*

**a. Classification**

The class component consists of attributes that include private modifiers, attribute names, data types, and a list of methods.

**b. Definition**

The Dependent class is a class containing numerous attributes based on the requirements specified in Diagram 3.2.3.1. There are some methods in the Dependent Class that explain the behavior of this class. This class has both no-args constructor and constructor with parameters to initiate the attributes inside it.

**c. Responsibilities**

Diagram 3.2.3.1 shows the Dependent class. As this is an entity class, it is responsible for creating dependents, editing dependents and storing the dependent's information.

3.2.3.1 Operation Specification

| | |
|---|---|
| Operation Specification: | Dependent(id: bigint, user_id: bigint, dependent_name: varchar, dependent_relation: varchar, dependent_ic: varchar, dependent_phone: varchar, depedent_occupation: varchar, dependent_date: varchar, death_status: varchar, death_date: varchar, created_at: timestamp, updated_at: timestamp) |
| Operation intent: | Creates a new dependent by inputting and storing dependent details parameters |
| Operation signature: | Dependent:: Dependent(id: bigint, user_id: bigint, dependent_name: varchar, dependent_relation: varchar, dependent_ic: varchar, dependent_phone: varchar, depedent_occupation: varchar, dependent_date: varchar, death_status: varchar, death_date: varchar, created_at: timestamp, updated_at: timestamp) |
| Logic description | **Context:** Dependent |

| | |
|---|---|
| (Pre-Post Condition): | **Pre:** None<br><br>**Post:** None |
| Other operations called: | **context:** +Dependent() |
| Events transmitted to other objects: | None |
| Attribute set: | None |
| Response to exceptions: | None |
| Non-functional requirements: | Performance: This operation should take place in less than 1 minute. |

### 3.2.4. Death Class

Detailed discussion of the kind of component of the system; such as a subsystem, module, class, package, function, file, etc. ....
E.g. Detailing each classes:

a. Classification

The kind of component, such as a subsystem, module, class, package, function, file, etc. ....

b. Definition

The specific purpose and semantic meaning of the component. This may need to refer back to the requirements specification.

c. Responsibilities

The primary responsibilities and/or behavior of this component. What does this component accomplish? What roles does it play? What kinds of services does it provide to its clients? For some components, this may need to refer back to the requirements specification.

(Refer Chapter 4 – Operation Specification and Control Specification)

d. Constraints

Any relevant assumptions, limitations, or constraints for this component. This should include constraints on timing, storage, or component state, and might include rules for interacting with this component (encompassing preconditions, postconditions, invariants, other constraints on input or output values and local or global values, data formats and data access, synchronization, exceptions, etc.)

…

# 4. Database Design

This section describes the database design for the MyKhairat mobile application.

## 4.1 Data Model

- Describe the transformation/mapping process from classes to tables for relational database design
- Present the data model

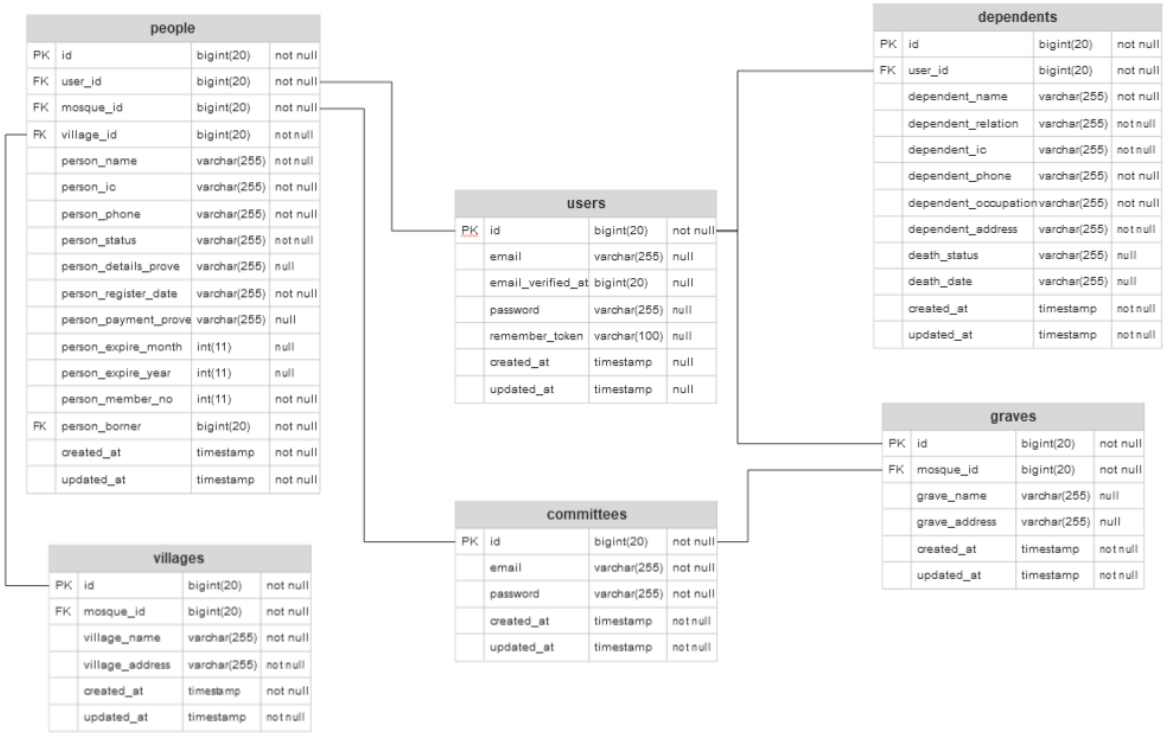



*Diagram 4.1.1 - Relational Table Mapping*

In Diagram 4.1.1 shows the data involved within this module. The relational table is based on the class diagram and details the attributes. The database is already in 3rd Normal Form (3NF). It is listed with the name of the attributes - (e.g id), the data type (e.g. bigint), primary key (PK) and foreign key (FK), and the constraint of the attribute (either null or not).

## 4.2 Data Dictionary

Lists all tables for the relational database.

### 4.2.1 Dependents

| Attribute's Name | Data Type | Constraint | Description |
|---|---|---|---|

| | | | |
|---|---|---|---|
| id | bigint(20) | PRIMARY KEY(NOT NULL) | Unique ID for the dependent |
| user_id | bigint(20) | FOREIGN KEY(NOT NULL) | Unique ID if it is a user |
| dependent_name | varchar(255) | NOT NULL | Name of the dependent |
| dependent_relation | varchar(255) | NOT NULL | Relationship of the dependent to a user |
| dependent_ic | varchar(255) | NOT NULL | IC number of the dependent |
| dependent_phone | varchar(255) | NOT NULL | Phone number of the dependent |
| depedent_occupation | varchar(255) | NOT NULL | Occupation of the dependent |
| dependent_address | varchar(255) | NOT NULL | Address of the dependent |
| death_status | varchar(255) | NULL | Status of the dependent's wellbeing |
| death_date | varchar(255) | NULL | Date of the dependent's death |
| created_at | timestamp | NOT NULL | Time at which it is created |

| | | | |
|---|---|---|---|
| updated_at | timestamp | NOT NULL | Time at which it is updated |

Table 4.2.1 - Relation Table for Dependents

### 4.2.2 People

| **Attribute's Name** | **Data Type** | **Constraint** | **Description** |
|---|---|---|---|
| id | bigint(20) | PRIMARY KEY(NOT NULL) | Unique ID for the user |
| user_id | bigint(20) | FOREIGN KEY(NOT NULL) | Unique ID if it is a user |
| mosque_id | bigint(20) | FOREIGN KEY(NOT NULL) | Unique ID if it is the user's mosque |
| village_id | bigint(20) | FOREIGN KEY(NOT NULL) | Unique ID if it is the user's village |
| person_name | varchar(255) | NOT NULL | Name of the person |
| person_ic | varchar(255) | NOT NULL | IC number of the person |
| person_phone | varchar(255) | NOT NULL | Phone number of the person |
| person_occupation | varchar(255) | NOT NULL | Occupation of the person |
| person_status | varchar(255) | NOT NULL | The status of the person |

| | | | |
|---|---|---|---|
| person_details_prove | varchar(255) | NULL | Proof of the person's details |
| person_register_date | varchar(255) | NOT NULL | Date of the person's registration |
| person_payment_prove | varchar(255) | NULL | Proof of the person's payment |
| person_expire_month | int(11) | NULL | Month of expiration |
| person_expire_year | int(11) | NULL | Year of expiration |
| person_member_no | int(11) | NOT NULL | Number of the member |
| person_borner | bigint(20) | NOT NULL | |
| created_at | timestamp | NOT NULL | The time at which it is created |
| updated_at | timestamp | NOT NULL | The time at which it is updated |

Table 4.2.2 - Relation Table for Dependents

### 4.2.3 Committee

| **Attribute's Name** | **Data Type** | **Constraint** | **Description** |
|---|---|---|---|
| id | bigint(20) | PRIMARY KEY(NOT NULL) | Unique ID for the user |

| | | | |
|---|---|---|---|
| email | varchar(255) | NOT NULL | Email of the committee member |
| password | varchar(255) | NOT NULL | Password of the committee member |
| created_at | timestamp | NOT NULL | Time at which it is created |
| updated_at | timestamp | NOT NULL | Time at which it is updated |

Table 4.2.3 - Relation Table for Committee

**4.2.4 Graves**

| **Attribute's Name** | **Data Type** | **Constraint** | **Description** |
|---|---|---|---|
| id | bigint(20) | PRIMARY KEY(NOT NULL) | Grave ID |
| mosque_id | bigint(20) | FOREIGN KEY(NOT NULL) | Unique ID if it is in a mosque |
| grave_name | varchar(255) | NULL | Name of the grave |
| grave_address | varchar(255) | NULL | Address of the dependent's grave |
| created_at | timestamp | NOT NULL | Time at which it is created |
| updated_at | timestamp | NOT NULL | Time at which it is updated |

Table 4.2.4 - Relation Table for Graves

### 4.2.5 Village

| Attribute's Name | Data Type | Constraint | Description |
|---|---|---|---|
| id | bigint(20) | PRIMARY KEY(NOT NULL) | Village ID |
| mosque_id | bigint(20) | FOREIGN KEY(NOT NULL) | Unique ID if it has a mosque |
| village_name | varchar(255) | NOT NULL | Name of the village |
| village_address | varchar(255) | NOT NULL | Address of the village |
| created_at | timestamp | NOT NULL | Time at which it is created |
| updated_at | timestamp | NOT NULL | Time at which it is updated |

Table 4.2.5 - Relation Table for Village

# 5. User Interface Design

(Explain and describe using Communication diagrams and Boundary classes, include the prototype design)

(Refer Chapter 8)

…

## 5.1 Use case: Edit Profile

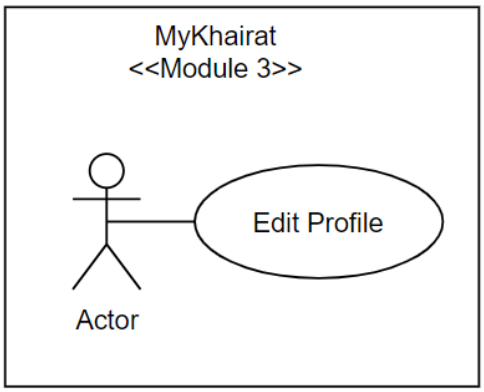



*Diagram 5.1.1 - Edit Profile Use Case*

In Diagram 5.1.1, the Edit Profile use case will involve the member and allow them to edit their profile details if they wish to change anything.

## 5.1.1 Sequence Diagram Entity Approach

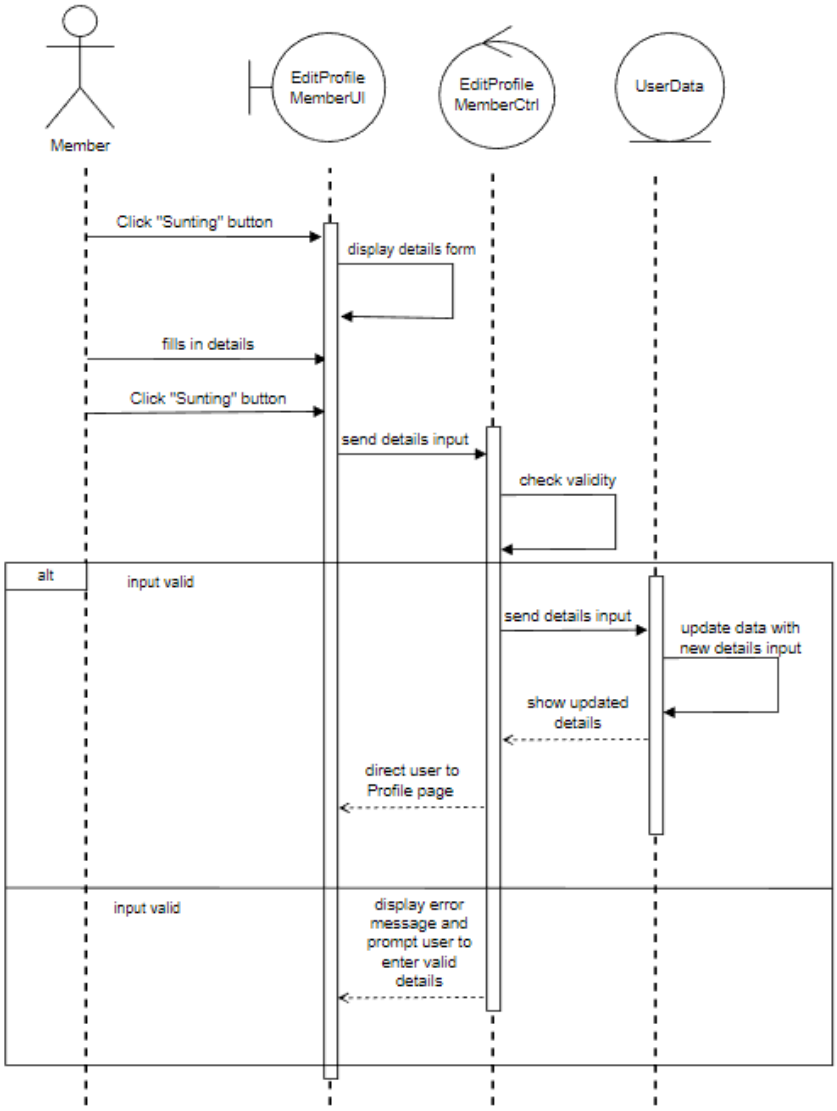



*Diagram 5.1.1.1 - Sequence Diagram for Edit Profile*

### 5.1.2 User Interface Prototype

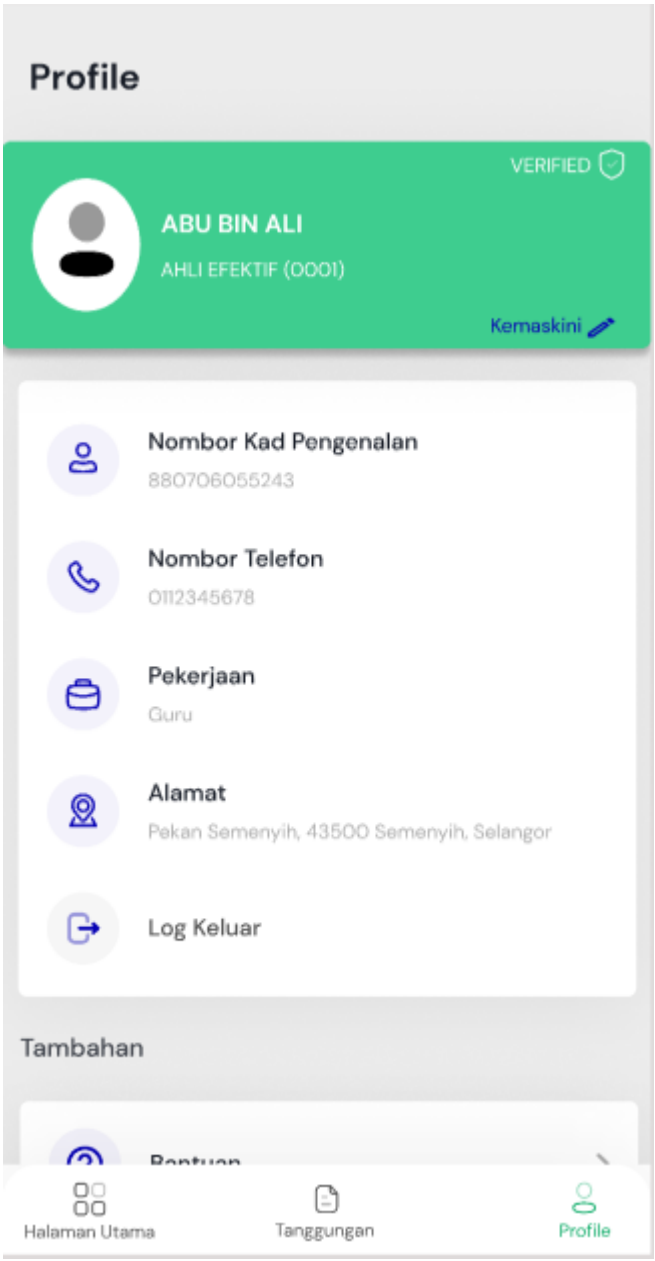



*Diagram 5.1.2.1 - Profile Page (Member)*

In Diagram 5.1.2.1, it shows the user interface of all the member's profile details. The button 'Kemaskini' is for the member to edit/update their profile information.

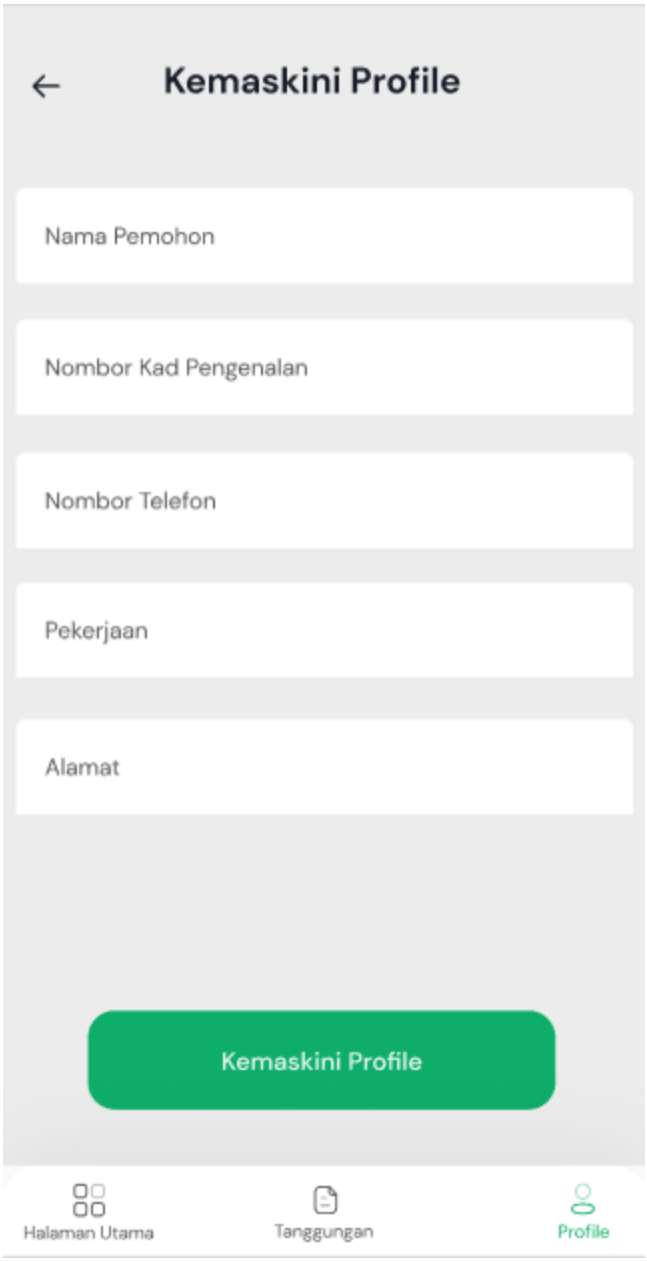



*Diagram 5.1.2.2 - Edit Profile Page*

In Diagram 5.1.2.2, it will show a page where the user can input information about themselves. Once they are done editing, they can save by pressing the ‘Kemaskini Profile’ button.

## 5.2 Use case: Make Payment

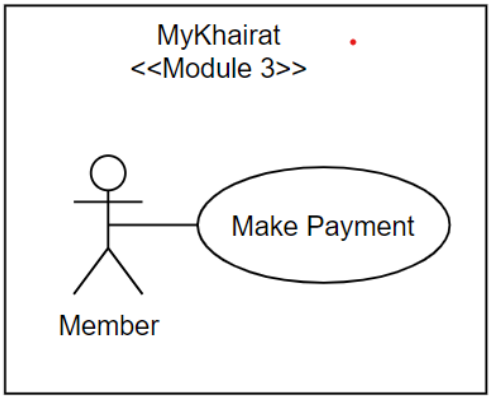



*Diagram 5.2.1 - Make Payment Use Case*

In Diagram 5.2.1, member can make their payment to pay the membership fee.

## 5.2.1 Sequence Diagram Entity Approach

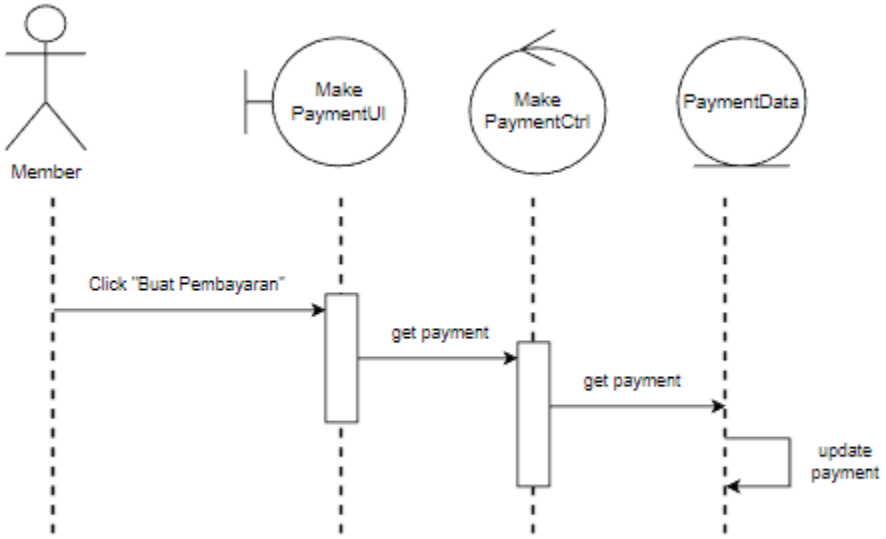



*Diagram 5.2.1.1 - Sequence Diagram for Make Payment*

### 5.2.2 User Interface Prototype


*Diagram 5.2.2.1 - Make Payment*

In Diagram 5.2.2.1, the interface will show the payment homepage. Member can click on the ‘Buat Pembayaran’ to make the payment.

## 5.3 Use case: Add Receipt

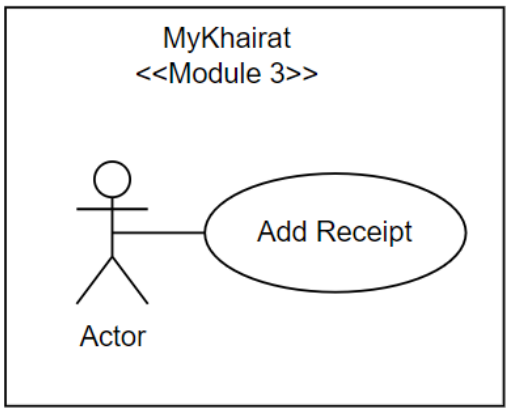



*Diagram 5.3.1 - Add Receipt*

In Diagram 5.3.1, the member can proceed with the payment by uploading the receipt. Member need to upload the receipt manually.

### 5.3.1 Sequence Diagram Entity Approach

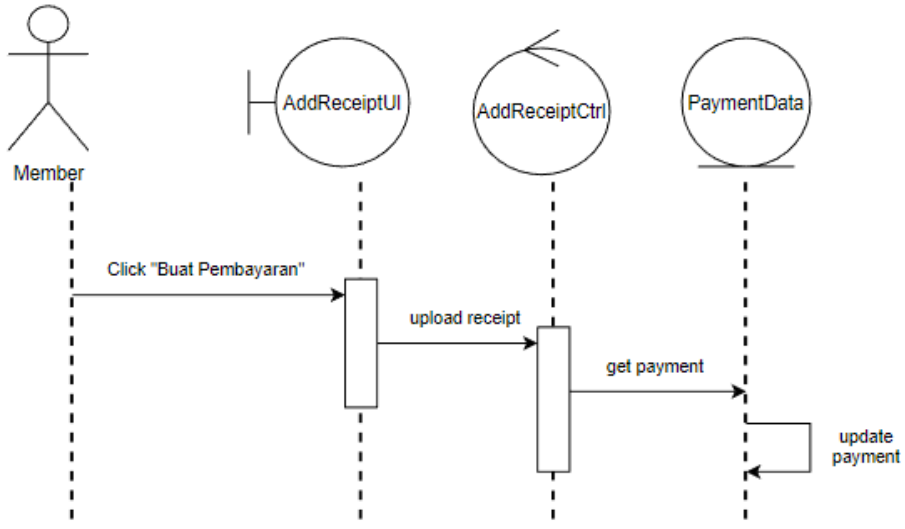



*Diagram 5.3.1.1 - Sequence Diagram for Add Receipt*

### 5.3.2 User Interface Prototype

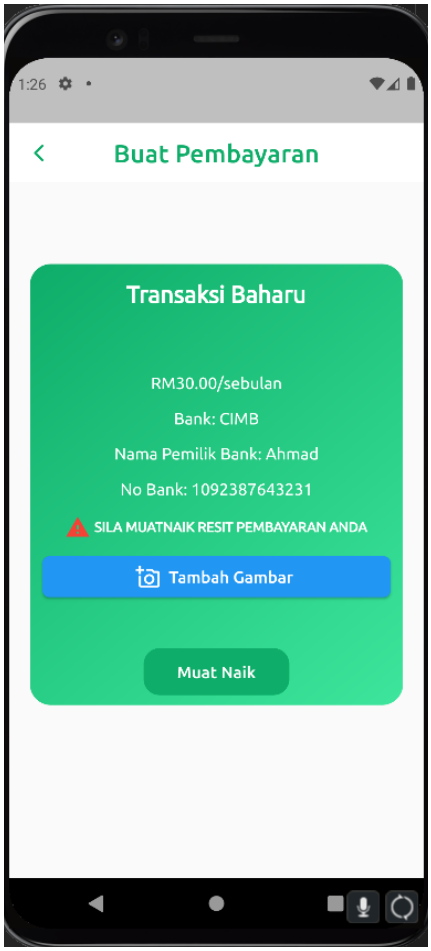



*Diagram 5.3.2.1 - Add Receipt*

In Diagram 5.3.2.1, the interface shows where the member can upload the receipt by clicking on the ‘Muat Naik’ button.

## 5.4 Use case: Create Announcement

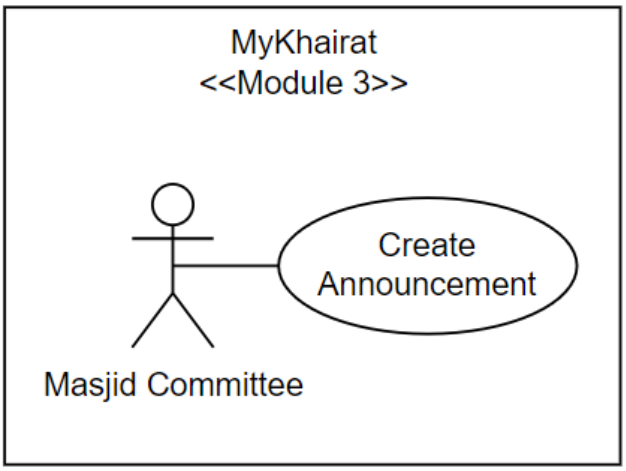



*Diagram 5.4.1 - Create Announcement Use Case*

In Diagram 5.4.1, Masjid Committee can create announcement for member if there is any.

## 5.4.1 Sequence Diagram Entity Approach

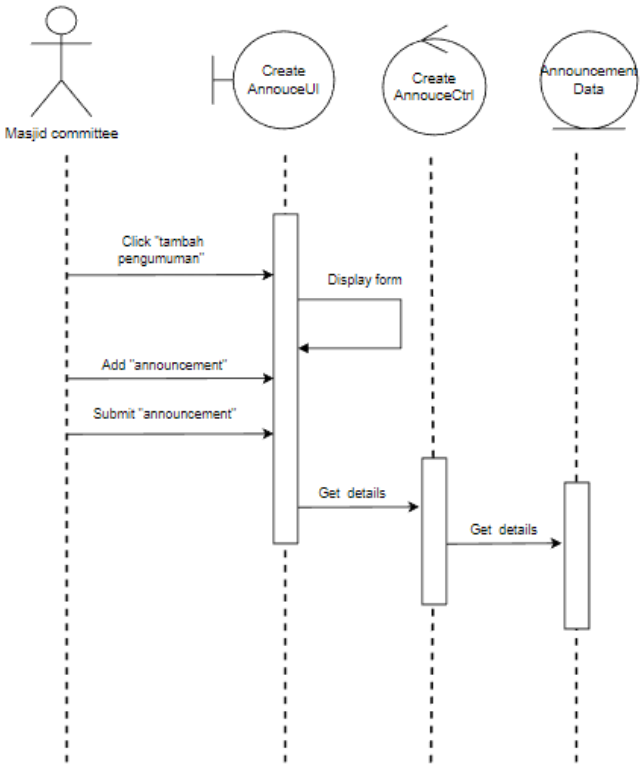



*Diagram 5.4.1.1 - Sequence Diagram for Create Announcement*

### 5.4.2 User Interface Prototype

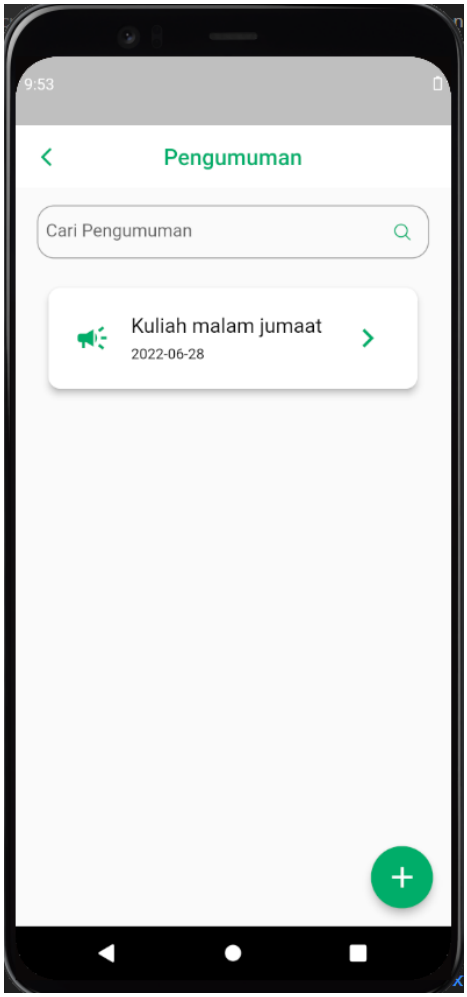



*Diagram 5.4.2.1 - Create Announcement*

In Diagram 5.4.2.1, the list of announcement will be display on this user interface. If Masjid Committee has any announcement to create, they can click '+' icon to create the announcement.

## 5.5 Use Case : View Announcement

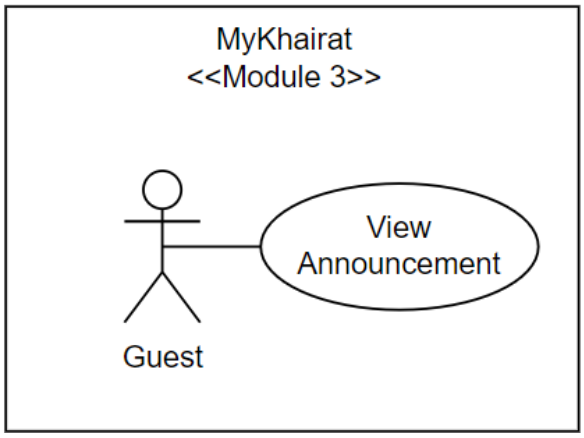



*Diagram 5.5.1 - View Announcement Use Case*

In Diagram 5.5.1, guest who is unregistered member can also view the announcement posted by the masjid committee on the homepage.

## 5.5.1 Sequence Diagram Entity Approach

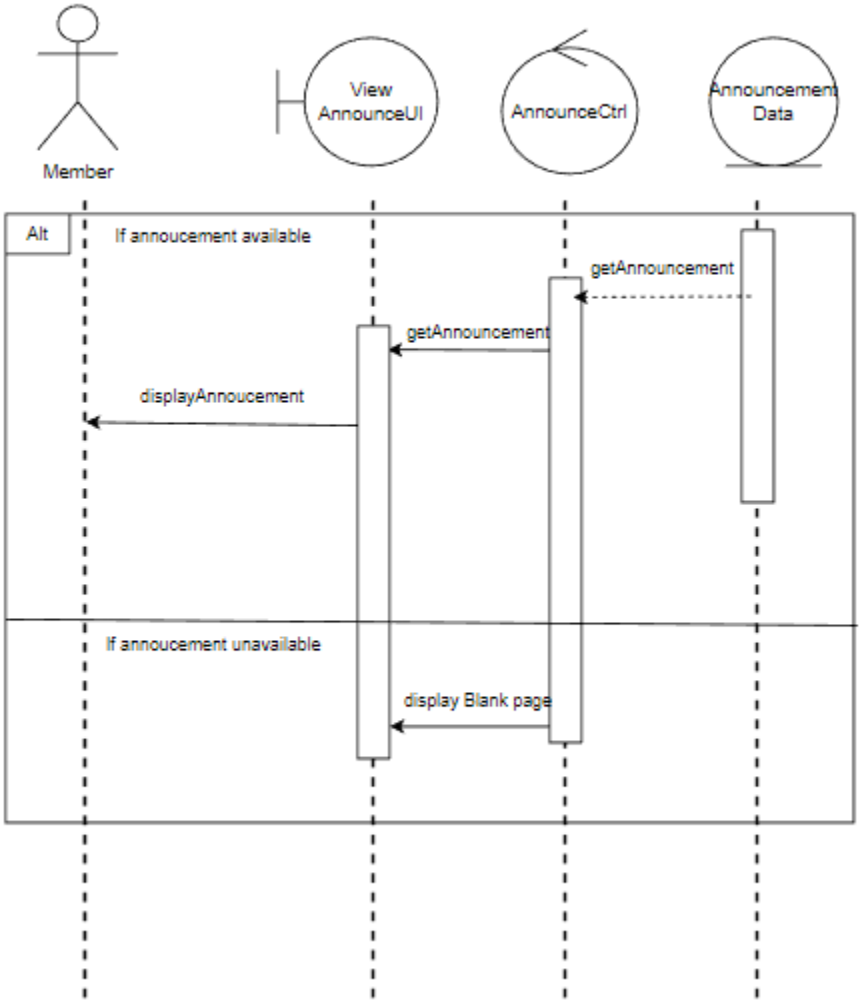



*Diagram 5.5.1.1 - Sequence Diagram for View Announcement*

### 5.5.2 User Interface Prototype


*Diagram 5.5.1.1 - View Announcement*

Diagram 5.5.1.1 shows the user interface of view announcement. Member and guest can view the announcement posted by the Masjid Committee in this page.

## 5.6 Use Case : Edit Announcement

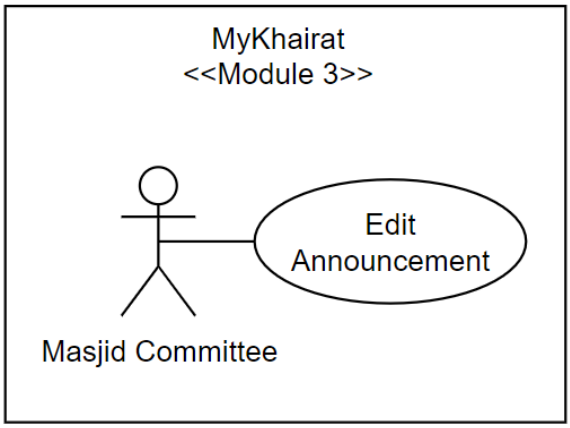



*Diagram 5.6.1 - Edit Announcement Use Case*

Diagram 5.6.1 shows that Masjid Committee can edit the previous or existed announcement. They can edit either the content or the title itself.

## 5.6.1 Sequence Diagram Entity Approach

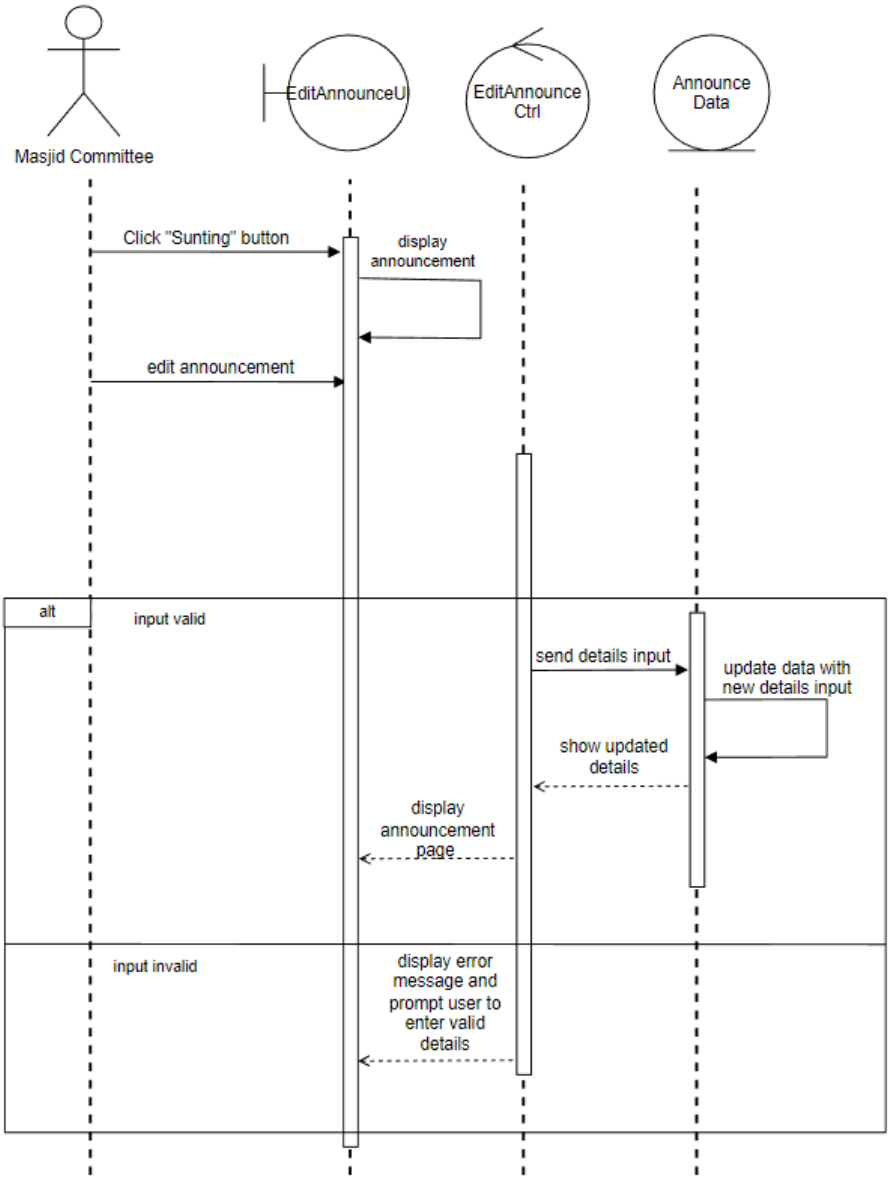



*Diagram 5.6.1.1 - Sequence Diagram for Edit Announcement*

## 5.6.2 User Interface Prototype

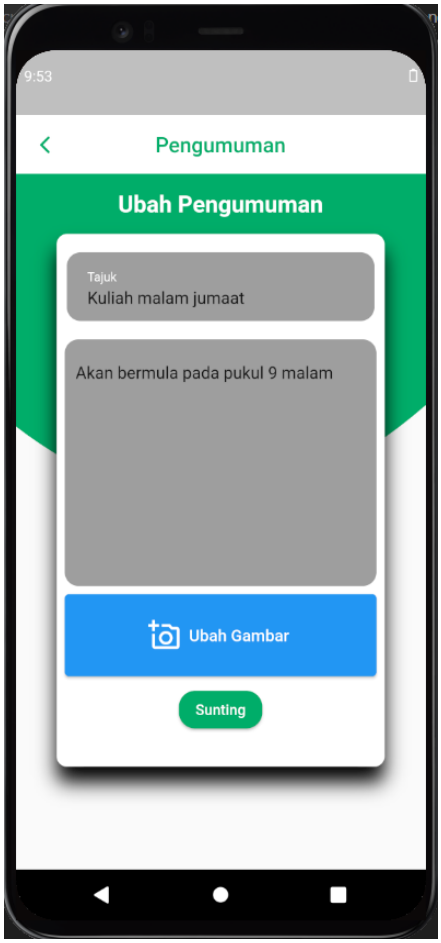



*Diagram 5.6.1.1*

Diagram 5.6.1.1 shows the user interface for the edit annoucement. Masjid Committee can edit any part of the announcement either its title or its body or even changing the pictures,

## 5.7 Use Case : Delete Announcement

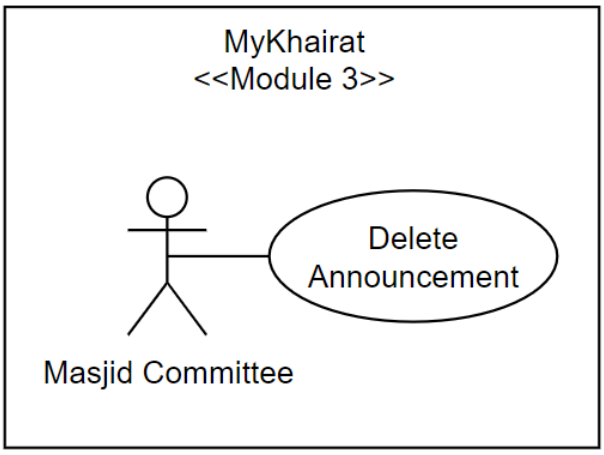



*Diagram 5.7.1 - Delete Announcement Use Case*

The diagram above shows that Masjid Committee can also delete the announcement if the announcement is not needed anymore

## 5.7.1 Sequence Diagram Entity Approach

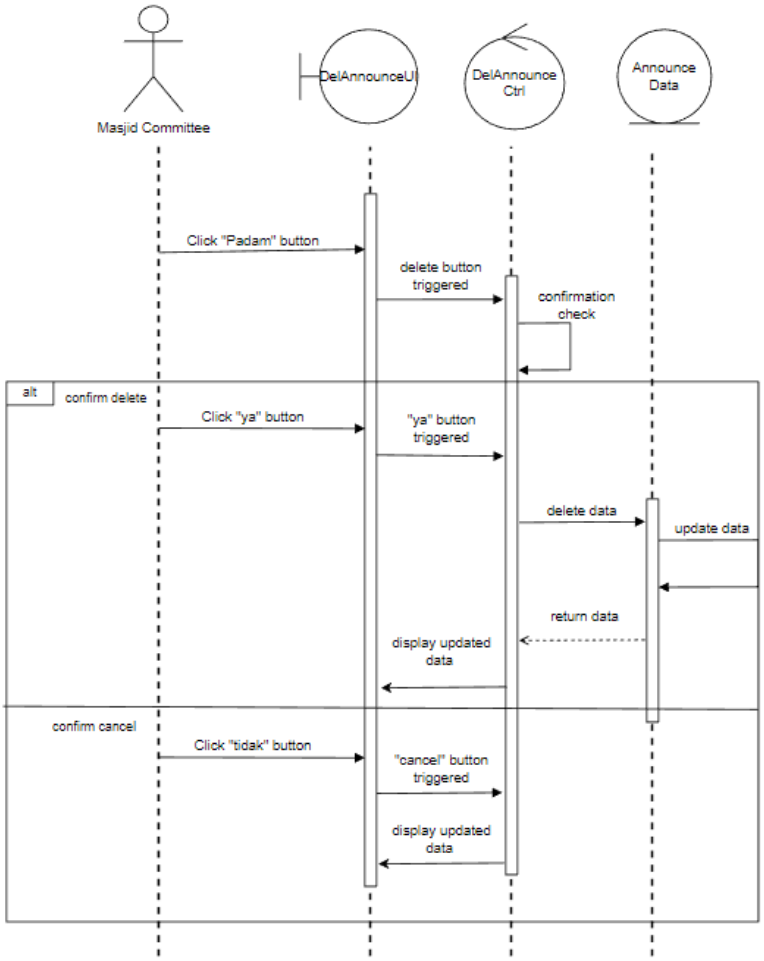



*Diagram 5.7.1.1 - Sequence Diagram for Delete Announcement*

### 5.7.2 User Interface Prototype

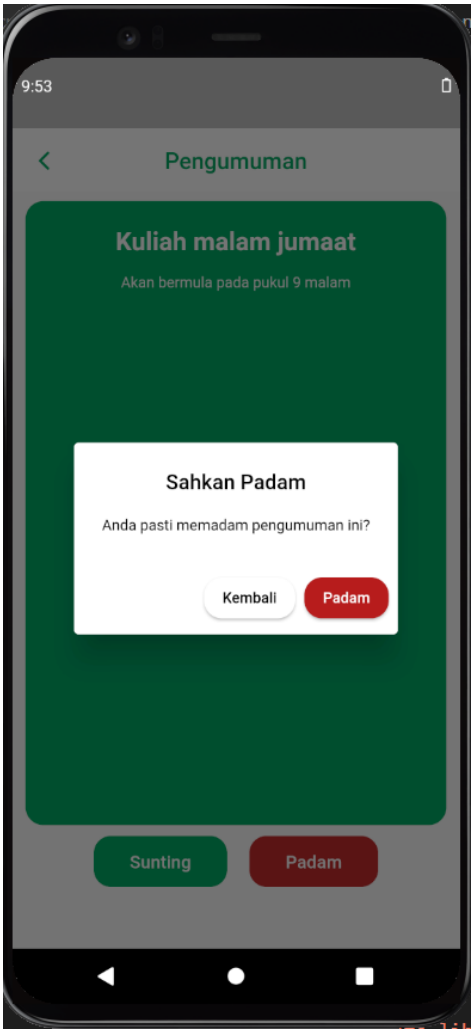



*Diagram 5.7.1.1 - Delete Announcement.*

Diagram 5.7.1.1 shows that whenever the Masjid Committee want to delete any announcement, they can just simply click on ‘padam’ button and validate the deleting announcement process.

## 5.8 Use Case : View Payment History

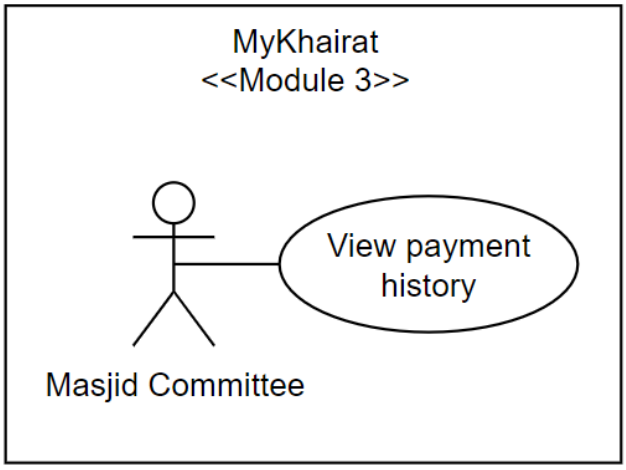



*Diagram 5.8.1 - View Payment History Use Case*

The diagram above shows that Masjid Committee can also view the payment history of the members.

## 5.8.1 Sequence Diagram Entity Approach

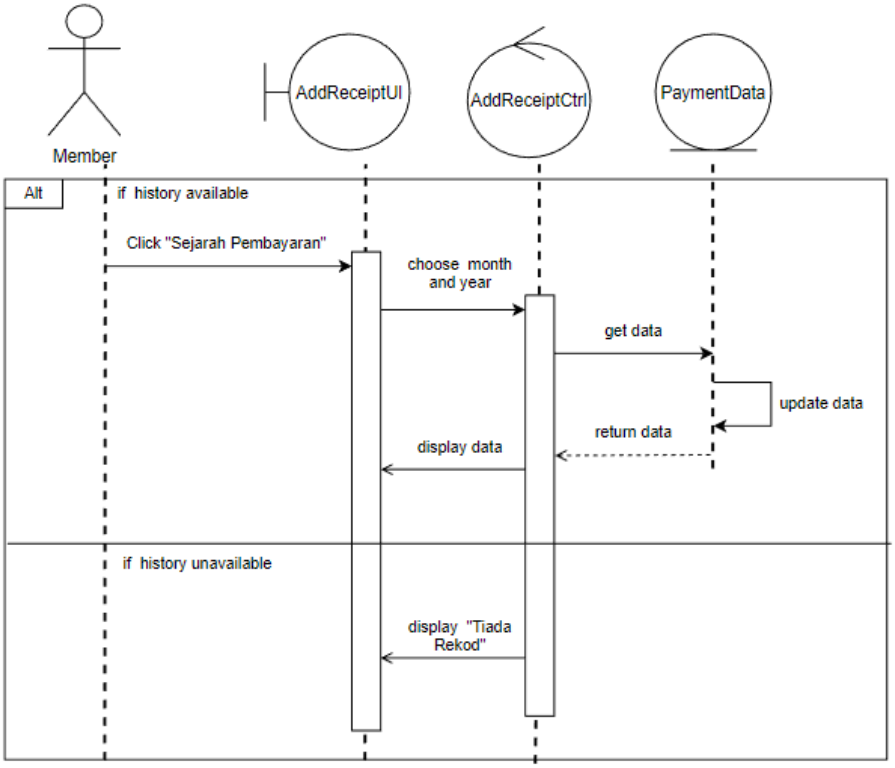



*Diagram 5.8.1.1 - Sequence Diagram for Create Announcement*

### 5.8.2 User Interface Prototype

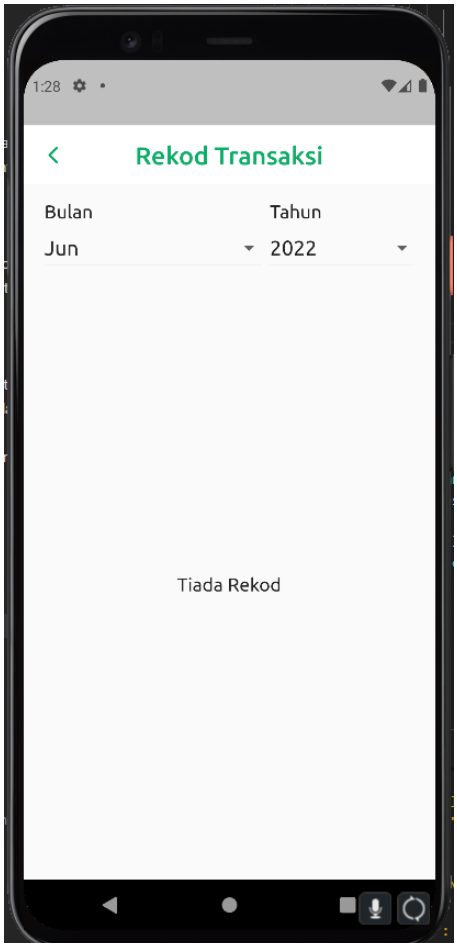



*Diagram 5.8.1.1 - View Payment History*

Diagram 5.8.1.1 shows that Masjid Committee can view the payment history. The list of successful payment will be displayed in the ‘Rekod Transaksi’

## 5.9 Use Case : Bank Details

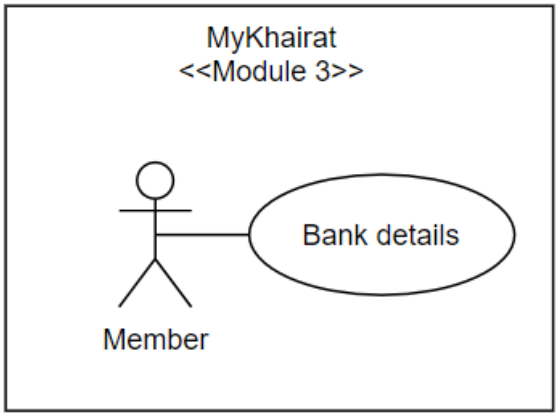



*Diagram 5.9.1 - Bank details*

The diagram 5.9.1 shows the bank details will be displayed on the upload receipt page.

## 5.9.1 Sequence Diagram Entity Approach

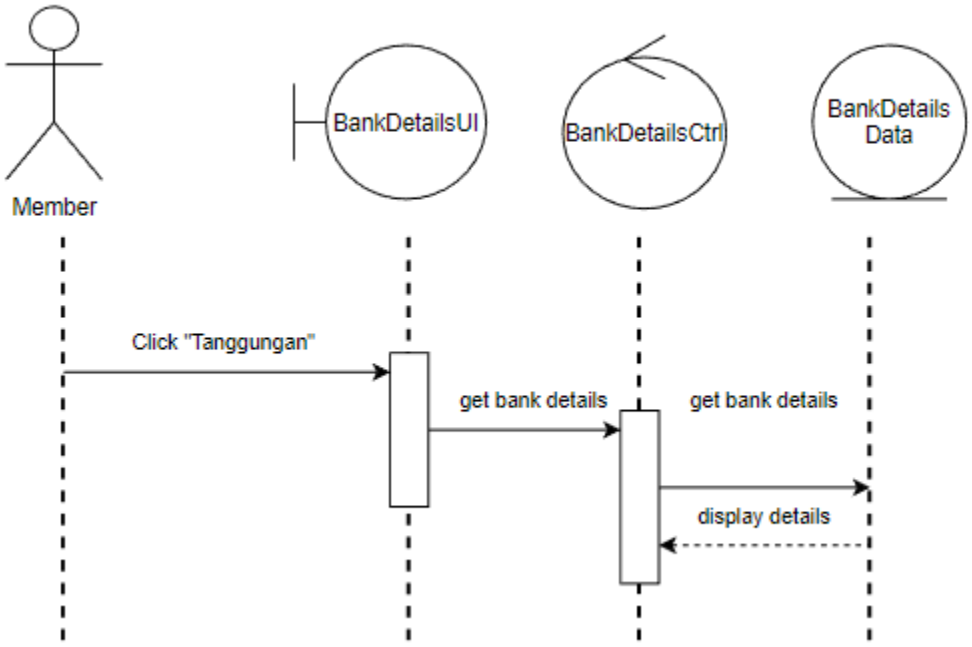



*Diagram 5.9.1.1 - Sequence Diagram for Bank Details*

### 5.9.2 User Interface Prototype

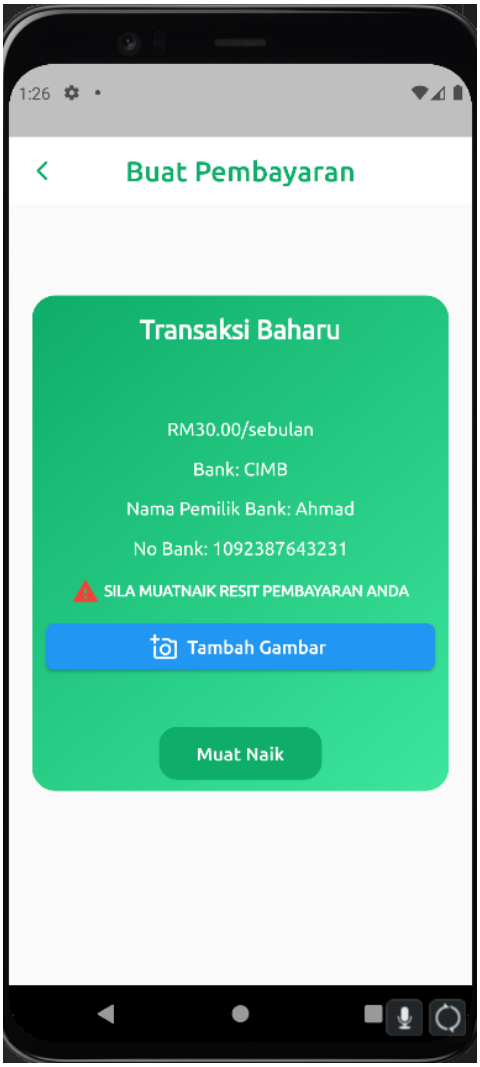



**Diagram 5.9.1.1 - Bank Details**

Diagram above shows that the bank details displayed on the add receipt  page. This is too ease the member to make a payment.

# 6. Architecture

## 6.1. Overview

Laravel is used as the framework for MyKhairat mobile application. Laravel is an open-source PHP framework that follows MVC architectural patterns. MVC is an abbreviation for “Model-View-Controller”. This architectural pattern divides the application into three logical parts; the model, the view, and the controller.

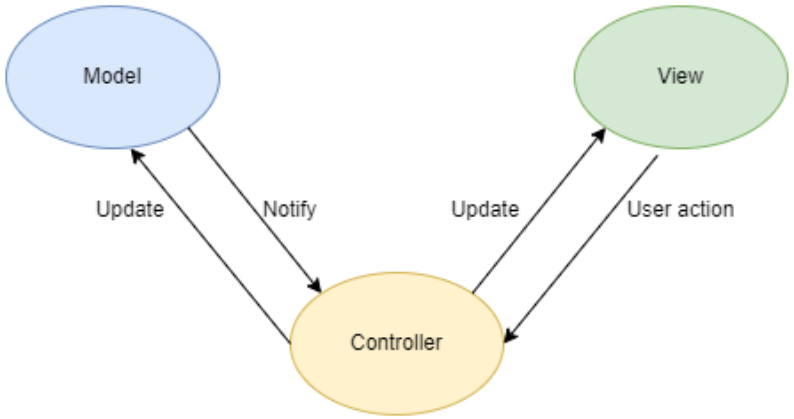



*Figure 6.1.1: MVC Architecture*

The components of MVC architecture are:

7. Model: Handles data logic. It is responsible for maintaining data. Adding or retrieving data is done in the model component as it is connected to the database.
8. View: Displays the information from the model to the user. It is responsible for data representation. It actually generates UI or user interface for the user.
9. Controller: Acts as mediator between Model and View. It controls the data flow into a model object and updates the view whenever data changes.

## 9.1. Deployment Diagram

This section details the execution architecture design of Module 3 using a deployment diagram.

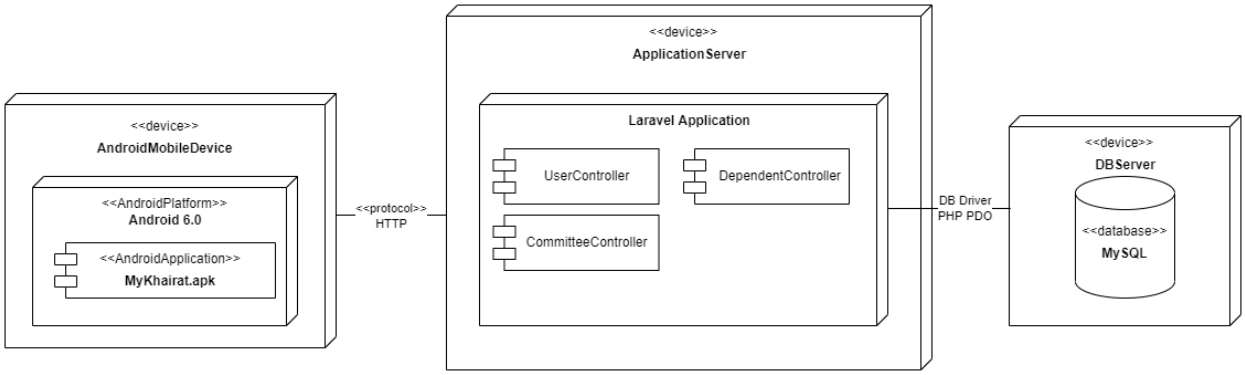



*Diagram 6.2.1 module 3 deployment diagram*

Diagram 6.2.1 represents the architecture of module 3 of MyKhairat.

Diagram 6.2.1 represents the deployment diagram of MyKhairat. Starting from the left side of the diagram, the first node depicts the client's Android mobile device. The second node is the application server that handles user requests to which they send the outcome the user wants. The data in the database will be retrieved by the application server through the database drivers in order to send the users the suitable and relevant data they need.

## 9.2. Package Diagram

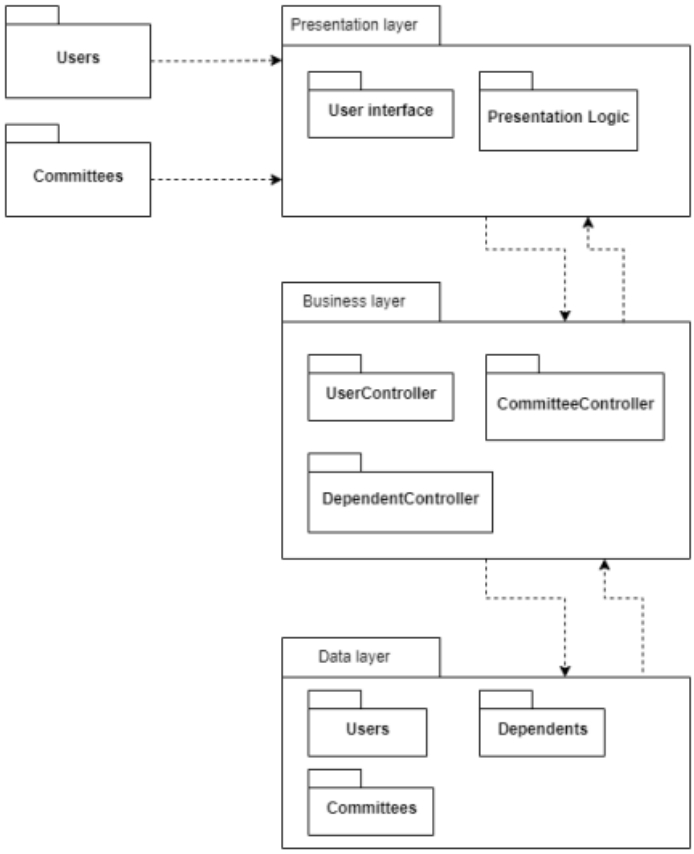



Diagram 6.3.1: Package Diagram for Module 3 of MyKhairat

Diagram 6.3.1 shows the package diagram for module 3 of MyKhairat mobile application. With reference to client-server architecture, this package model shows how the application's display layer, business layer, and data layer interact. The presentation layer makes the user interface visible and makes it easier for users to engage with the system. The business layer is in charge of the application's features. A business layer contains the procedures that control how data and requests go through the back end. The data layer controls how application data is stored and retrieved.

### 9.3. Strategy

a) MVC architectural pattern

Separation of concerns is MVC's major benefit. Separation of concerns is a design idea used in computer science to divide a computer programme into portions, in this example, Model, View, and Controller, each of which addresses a different worry. Due to the lack of duplication and the specific goal of each component, the maintenance of applications may be facilitated by the separation of concerns. The minimum overlap between the functions also makes the testing and development processes easier because it makes the code more reusable and readable.

b) Client-Server architecture

Client-server architecture is popular because it is efficient in providing resources to clients and needs little maintenance. It can also improve security by giving servers better control and access to resources, ensuring that only authorised clients can access or change the application's data. Client-server architecture is advantageous in terms of maintenance since it is a distributed model expressing scattered duties of diverse tasks. Because of the encapsulation, it is simple to replace, repair, upgrade, and relocate a server without disrupting the clients.

## 10. Glossary

An ordered list of defined terms and concepts used throughout the document.

| Terms | Definitions |
|---|---|
| Android | Android is a mobile operating system based on a modified version of the Linux kernel and other open source software, designed primarily for touchscreen mobile devices such as smartphones and tablets. |
| Flutter | An open source framework by Google to create beautiful, |

| | |
|---|---|
| | natively compiled applications for mobile, web, and desktop from a single codebase. |
| Laravel | A PHP web application framework with expressive, elegant syntax. It eases common tasks (authentication, routing, etc.) used in web projects. |
| MySQL | An open source relational database management system. It creates a database for storing and manipulating data, defining the relationship of each table. |
| Agile | Agile methodology is a type of project management process, mainly used for software development, where demands and solutions evolve through the collaborative effort of self-organising and cross-functional teams and their customers. |