|  |
| --- |
| C:\Users\Tsubaki Yukino\Desktop\FlyAwayPlus Project\Capture.JPG |
| **IVOLUNTEER** |
| **Software Architecture Design** |
|  |
| |  |  |  | | --- | --- | --- | | **iVolunteer** | | | | **Group Registered users** | Nguyễn Minh Đức | SE03053 | | Nguyễn Văn Tâm | SE02495 | | Võ Tôn Phúc | SE03162 | | Nguyễn Thạc Sơn | SE03343 | | Phạm Văn Trọng | SE03292 | | **Supervisor** | Mr. Bui Dinh Chien | | | **Project code** | IVSN | | |
|  |

**- Hanoi, 06/2016 –**

**RECORD OF CHANGE**

\*A-Added; M-Modified; D-Deleted

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Effective Date | Change Items | A\*M, D | Change Description | New Version |
| 02/06/2016 |  | A |  | v1.0 |
|  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

*Table 1: Record of change*

**SIGNATURE PAGE**

|  |  |  |
| --- | --- | --- |
| **ORIGINATOR** | Nguyen Minh Duc |  |
|  | Developer |  |
|  |  |  |
|  |  |  |
| **REVIEWERS** | Nguyen Van Tam |  |
|  | Developer |  |
|  |  |  |
|  |  |  |
| **APPROVAL** | Bui Dinh Chien |  |
|  | Supervisor |  |
|  |  |  |

Table of Contents

[**1.** **INTRODUCTION** 5](#_Toc452942832)

[1.1. Purpose 5](#_Toc452942833)

[1.2. Scope 5](#_Toc452942834)

[1.3. Definitions, Acronyms, Abbreviations 5](#_Toc452942835)

[1.4. References 5](#_Toc452942836)

[1.5. Overview 6](#_Toc452942837)

[**2.** **CHOICE OF ARCHITECTURE DESIGN** 6](#_Toc452942838)

[2.1. MVC Model 6](#_Toc452942839)

[2.1.1. MVC Model overview 6](#_Toc452942840)

[2.1.2. Advantages and disadvantages of MVC model 8](#_Toc452942841)

[2.1.3. The reason of choosing MVC Model 8](#_Toc452942842)

[2.2. .NET Framework 8](#_Toc452942843)

[2.2.1. The reason of choosing .NET Framework 8](#_Toc452942844)

[2.3. MongoDB 9](#_Toc452942845)

[2.3.1. Advantages and disadvantages of MongoDB 9](#_Toc452942846)

[2.4. Microsoft SQL Server 2014 9](#_Toc452942847)

[2.4.1. Advantages and disadvantages of Microsoft SQL Server 2014 9](#_Toc452942848)

[2.4.2. The reason of combining the two MongoDB and Microsoft SQL Server 2014 9](#_Toc452942849)

[3. **ARCHITECTURAL REPRESENTATION** 10](#_Toc452942850)

[**4.** **ARCHITECTURAL GOALS AND CONSTRAINTS** 11](#_Toc452942851)

[**5.** **USE-CASE VIEW** 12](#_Toc452942852)

[5.1. User module 13](#_Toc452942853)

[5.1.1. Common module 13](#_Toc452942854)

[5.1.2. Social module 14](#_Toc452942855)

[5.1.3. Account management 24](#_Toc452942856)

[5.1.4. Administrator management 25](#_Toc452942857)

[**6.** **IMPLEMENTATION VIEW** 26](#_Toc452942858)

[6.1. Overview 26](#_Toc452942859)

[6.2. Architecturally Significant Design Packages 27](#_Toc452942860)

[**7.** **PROCESS VIEW** 30](#_Toc452942861)

[**8.** **DEPLOYMENT VIEW** 37](#_Toc452942862)

[**9.** **QUALITY** 37](#_Toc452942863)

# **INTRODUCTION**

## Purpose

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions that have been made on the system.

## Scope

The scope of this document is to depict the architecture of the IVolunteer Social Network website created by iVolunteer capstone project team.

## Definitions, Acronyms, Abbreviations

|  |  |  |
| --- | --- | --- |
| **Acronym** | **Definition** | **Note** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## References

* IVSN\_ Software Requirements Specification\_v1.0\_EN.docx
* Software Architecture Design Illuminated Book
* Microsoft Developer Network ASP.NET MVC
* A MongoDB White Paper

## Overview

The Software Architecture Document contains the following subsections:

* **Section 1**: Provide an overview of entire Software Architecture Document.
* **Section 2**: Choice of Architecture Design
* **Section 3**: Architectural Representation
* **Section 4**: Architectural Goals and Constraints
* **Section 5**: Use-Case view
* **Section 6**: Logical View
* **Section 7**: Process View
* **Section 8**: Deployment view
* **Section 9**: Quality

# **CHOICE OF ARCHITECTURE DESIGN**

## MVC Model

The purpose of IVSN is developing a volunteer social network where all people who share the willing of helping others can connect to others, build volunteering projects and spread them out to publicity in the IVSN system. The system of IVSN is structured based on MVC combined with layered architecture.

### MVC Model overview

MVC is a software architecture pattern that separates the representation of information from user’s interaction with it. MVC Model separates system into components to make developers can develop, maintain and upgrade some specific parts without making change to whole system. MVC stands for Model-View-Controller where: Model is used to modelize data into objects, View is used to display user interface and Controller is used to implement business logic.

The Model-View-Controller (MVC) design pattern assigns objects in an application one of three roles: model, view, or controller. The pattern defines not only the roles objects play in the application, it defines the way objects communicate with each other. Each of the three types of objects is separated from the others by abstract boundaries and communicates with objects of the other types across those boundaries. The collection of objects of a certain MVC type in an application sometimes referred to as a layer—for example, model layer.

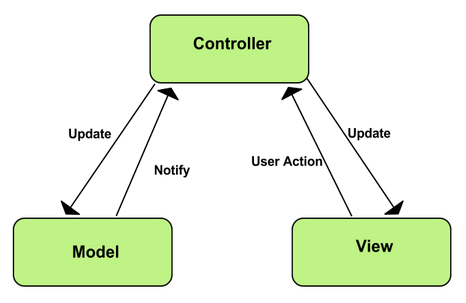


Figure 2.1‑1: MVC Model

In addition to dividing the application into three kinds of components, the MVC design defines the interactions between them:

* **A controller:** can send commands to its associated view to change the view's presentation of the model (e.g., by scrolling through a document). It can also send commands to the model to update the model's state (e.g., editing a document).
* **A model:** notifies its associated views and controllers when there has been a change in its state. This notification allows the views to produce updated output, and the controllers to change the available set of commands. A passive implementation of MVC omits these notifications, because the application does not require them or the software platform does not support them.
* **A view:** requests from the model the information that it needs to generate an output representation to the user.

### Advantages and disadvantages of MVC model

* **Advantages:**
* Allow simultaneous work between developers who are responsible for different components (such as UI layer and core logic).
* Easier to maintain.
* **Disadvantages:**
* Take much time to transfer data between components
* Time consuming to transits data between components.
* Not suitable for agent-oriented applications such as interactive mobile and robotics applications.

### The reason of choosing MVC Model

* MVC separates the concern of storing, displaying, and updating data into three components that can be tested individually and easier.
* Members of team can work individually on separated components of MVC.
* IVSN system is planned to developed into two phases mean that upgrade is indispensable. MVC will help changing one component without huge impact on the whole system.

## .NET Framework

.NET Framework is a software framework developed by Microsoft that runs primarily on Microsoft Windows. It includes a large class library known as Framework Class Library (FCL) and provides language interoperability (each language can use code written in other languages) across several programming languages.

### The reason of choosing .NET Framework

* Team members have got familiar with .NET programming.

## MongoDB

MongoDB is an open-source document database, and leading NoSQL (none-relational) database that provides high performance, high availability, and automatic scaling. A document database is constructed of filed and value pairs which is helpful for modeling unstructured and polymorphic data.

### Advantages and disadvantages of MongoDB

* **Advantages:** 
  + Indexes support faster queries and can include keys from embedded documents and arrays.
  + Rich query language to support read and write operations.
  + Replica set maintain the same dataset, providing redundancy and increasing data availability.
  + Sharding or horizontal scaling supports high throughput and large data sets.
* **Disadvantages:**
  + Data size in MongoDB is typically higher
  + Duplicate data happen frequently.

## Microsoft SQL Server 2014

Microsoft SQL Server 2014 is the next generation of Microsoft’s information platform, with new features that deliver faster performance, expand capabilities in the cloud, and provide powerful business insights.

### Advantages and disadvantages of Microsoft SQL Server 2014

In comparison to NoSQL database, the Microsoft SQL Server have these advantages and disadvantages.

* **Advantages:** 
  + SQL databases are best fit for heavy duty transactional type.
  + SQL is a powerful query language which was designed for managing data in a relational database management system
* **Disadvantages:**
  + Database schema need to be predicted quite exactly at the beginning of the feature development.

### The reason of combining the two MongoDB and Microsoft SQL Server 2014

Considering the advantages that the two Database supports, we have made the decision to combine the two database in developing the IVSN system for these reasons.

* + The IVSN system is planned to develop into two circles of Iterative and incremental developing process. Therefore, project scaling is indispensable. In this case MongoDB is suitable for this.
  + SQL Server support Transaction that guarantees either updates both succeed or both fail.

# **ARCHITECTURAL REPRESENTATION**

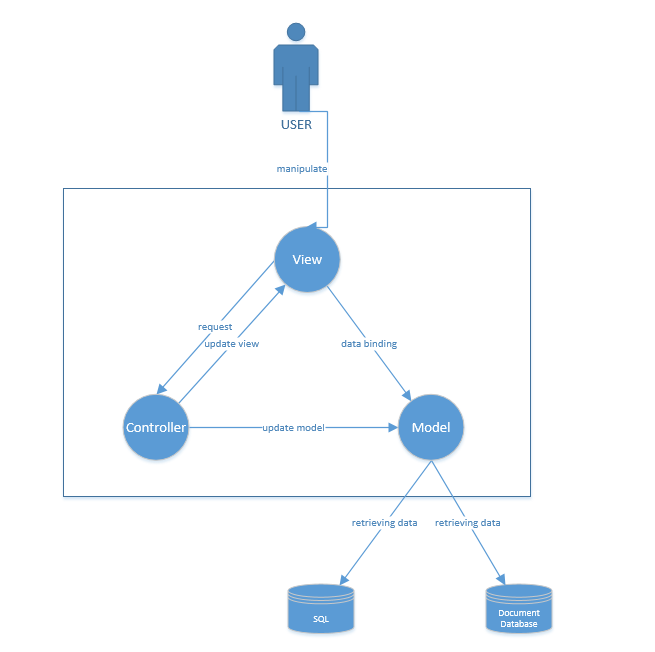


Figure 2.3‑1: System overview

We follow MVC architecture to implement the IVSN Project. MVC offers architectural benefits over standard JavaScript — it helps us write better-organized and therefore more maintainable code.

**Model** is where the application’s data objects are stored. A model object is in charge of encapsulating application state and one object could be related to other objects establishing a one-to-one or one-to-many relationship. The model object does not talk directly to a View, instead is made available to a controller, which accesses it when needed. When a model changes, typically it will notify its observers that a change has occurred. As with any data object it contains instance variables and getter/setter methods.

**View** is what is presented to the users and how users interact with the system. The view is expected to render the model in a meaningful way to the user. In FAP, the view is made with .cshtml file including css, JavaScript or jQuery, it sends user gestures to controller and allows controller to select view.

**Controller** is the decision maker and the glue between the model and view; it handles user actions and gestures, and responds to user events. For example, in CMS, when a user clicks the “Create” button to create a new contract, the controller for that action is invoked.

The controller will then make changes to the contract model. The view will then render the modified contract model to the display so that user can view the new contract he added in the contract list.

# **ARCHITECTURAL GOALS AND CONSTRAINTS**

* **Availability**
* The application must be available 90% of time. Users can access to it everywhere from there .Web browser with internet connection.
* **Maintainability**
* Coding standards and naming conventions
  + - Output of the project must include coding standards and naming conventions documentations. Implementation code must be easy to maintain.
    - All code must be clearly commented, including class, method documentations.
    - If some components are reused, the documentations of those components must also be included.
* Design
  + - The design of the system must be loosely coupled that chances on some module will not affect others.
* Logging
  + - All the errors should be logged, supporting for bug fixing and maintenance.
    - All strange or sensitive situations should also be logged.
* **Usability**
* Usability Requirements support the following from the perspective of its primary users:
  + - Efficiency of use: user can complete each function in less than 12 actions
    - Intuitiveness: all help/error messages are simple to understand; user can know exactly how to do each feature after one time using it.
* **Capacity and scalability**

Throughput, storage and growth requirements.

# **USE-CASE VIEW**

* **This application includes two parts:**

1. The first part is User module. User module includes registered user and guest.
2. Next part is Administrator module. In Administrator side includes Administrator to manage registered user’s account and registered users’ posts.

## User module

### Common module

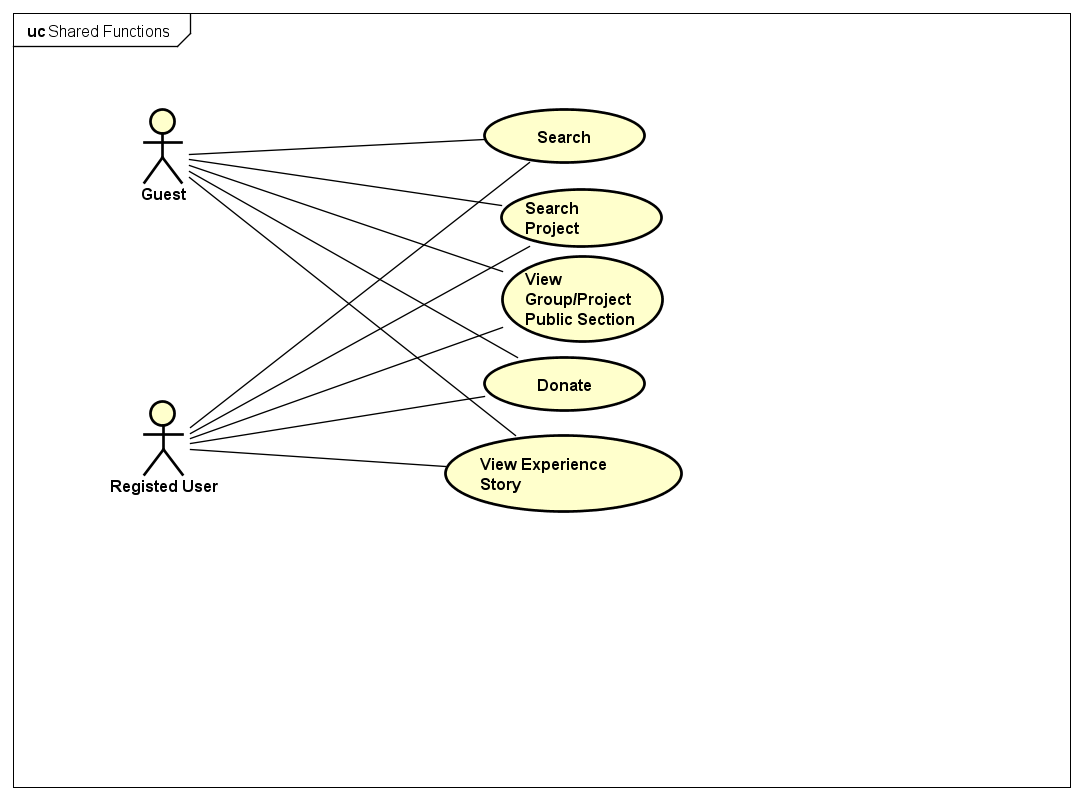


Figure 5.1‑1 Common module

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Use-case name** | **Actor** | **Description** |
|  | View Group/ Project public section | Registered user  Guest | User clicks to be redirected into Group page/ Project page to see more detail information. |
|  | View experience story | Registered user  Guest | User views all the experience stories that are published by Registered User in the system |
|  | Search | Registered user  Guest | User can enter name of Project/ Group/ Registered User as keyword to Search bar to find. |
|  | Advanced Search Project | Registered user  Guest | User can search for Project by entering Project’s date, sector or destination. |
|  | Donate | Registered user  Guest | User can make a donation request to Project Leader. |

### Social module

#### Profile page

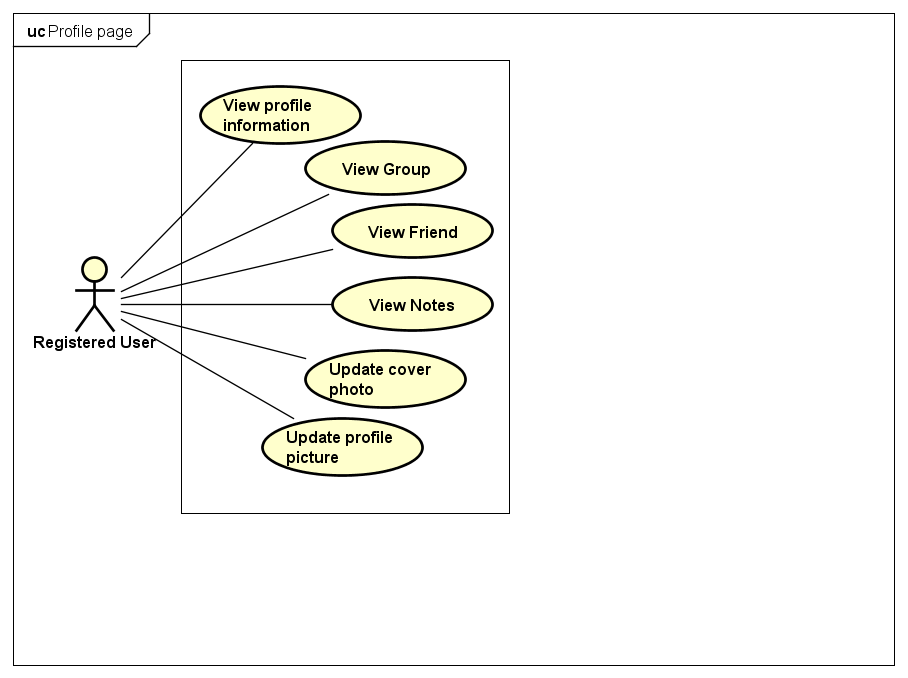


Figure 5.1‑3: Personal Page

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Use-case name** | **Actor** | **Description** |
| 1 | View profile information | Registered use | User views all profile information in Profile information section |
| 2 | View Groups | Registered user | User views all Groups that he has participated in. |
| 3 | View Friends | Registered user | User views all Friends in Friends section. |
| 4 | View Notes | Registered user | User views all Note that he has created in Notes section. |
| 5 | Update cover photo | Registered user | User changes his cover photo |
| 6 | Update profile picture | Registered user | Use changes his profile picture. |

#### Post management

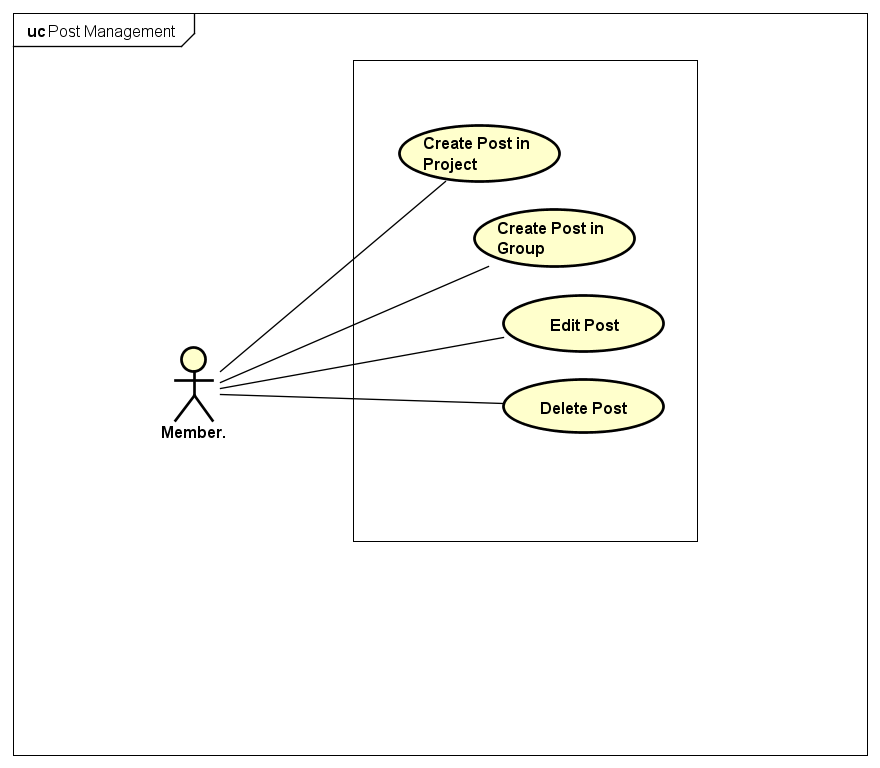


Figure 5.1‑4: Post Management

|  |  |  |  |
| --- | --- | --- | --- |
| No | Use-case name | Actor | Description |
| 1 | Create Post in Group/Project | Member | User creates post in Discussion of Group/ Project to discuss with other Members. |
| 2 | Edit Post | Member | User edits content the post that he created. |
| 3 | Delete Post | Member | User delete content the post that he created. |

#### Comment Management

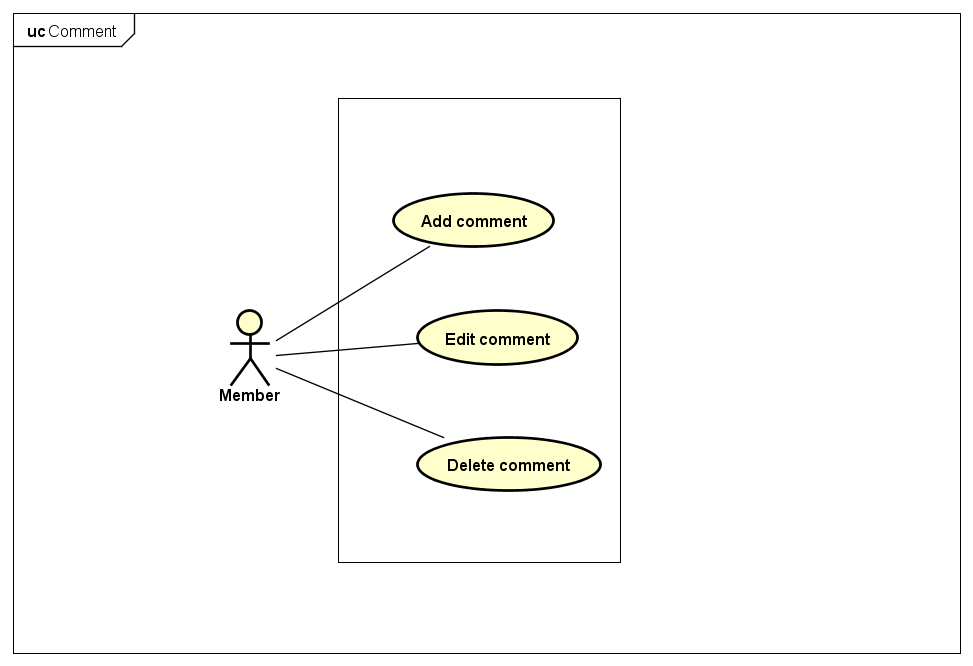


Figure 5.1‑5: Comment Management

|  |  |  |  |
| --- | --- | --- | --- |
| No | Use-case name | Actor | Description |
| 1 | Add comment | Member | Add new comment |
| 2 | Edit comment | Member | Edit existing comment |
| 3 | Delete comment | Member | Delete existing comment |

#### Like management

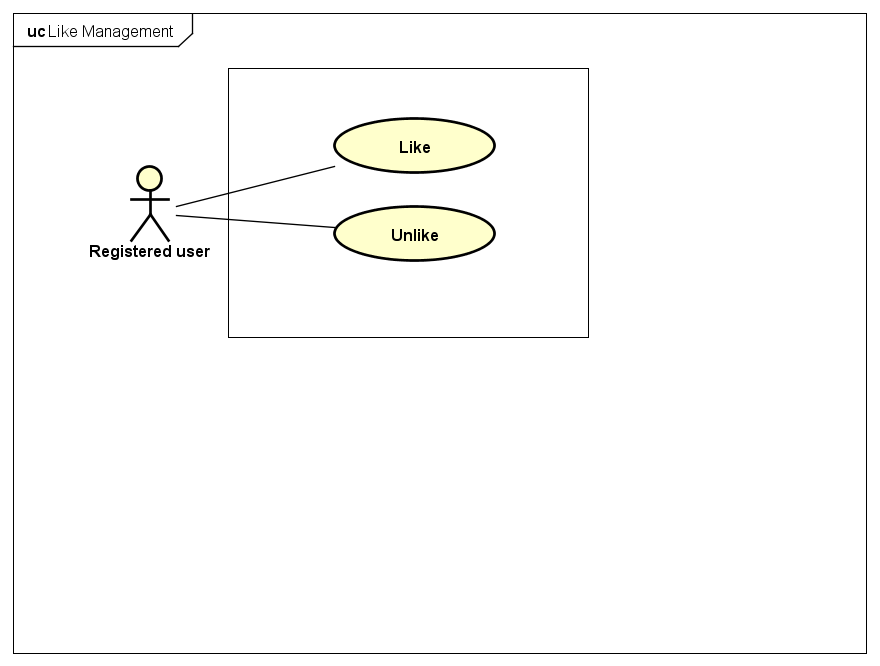


Figure 5.1‑6: Like Management

|  |  |  |  |
| --- | --- | --- | --- |
| No | Use-case name | Actor | Description |
| 1 | Like | Registered user | Like a post/photo |
| 2 | Unlike | Registered user | Unlike a post/photo |

#### Report

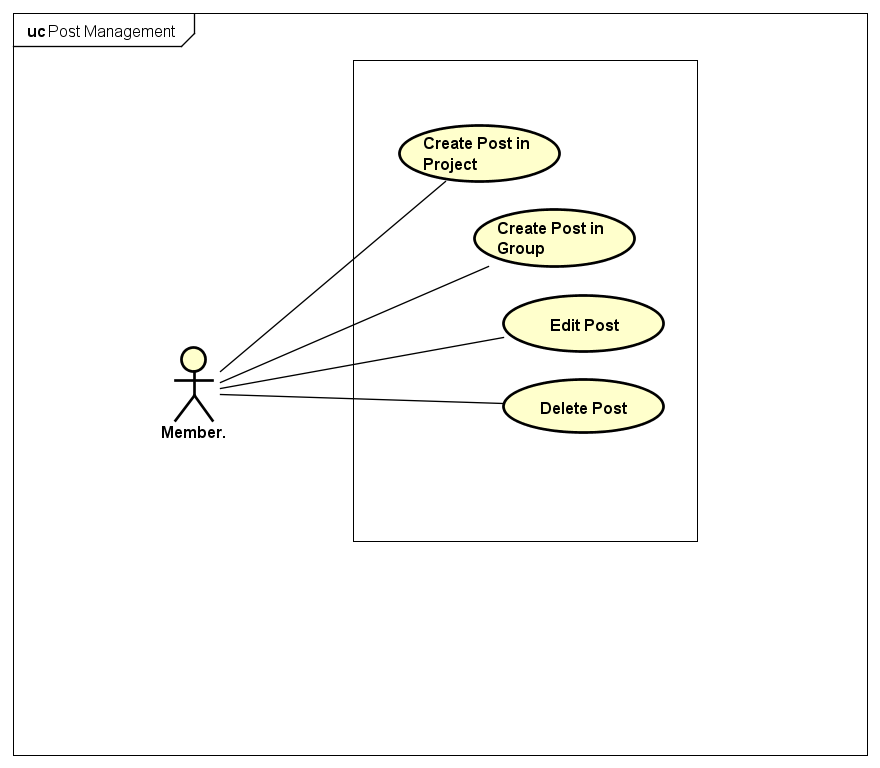


Figure 5.1‑7: Report

|  |  |  |  |
| --- | --- | --- | --- |
| No | Use-case name | Actor | Description |
| 1 | Report a post | Registered user | Report a violated post |
| 2 | Report a user | Registered user | Report a violated user |
| 3 | Report a group | Registered user | Report a violated group |
| 4 | Report a project | Registered user | Report a violated project |

#### Chat

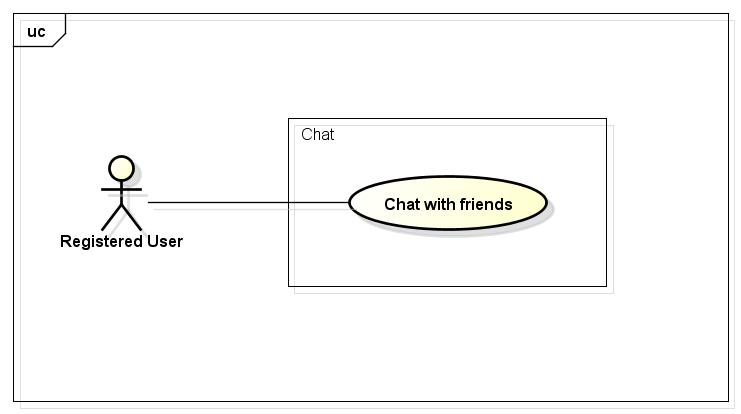


Figure 5.1‑8: Chat

|  |  |  |  |
| --- | --- | --- | --- |
| No | Use-case name | Actor | Description |
| 1 | Chat | Registered user | Chat with other registered user |

#### Friend management

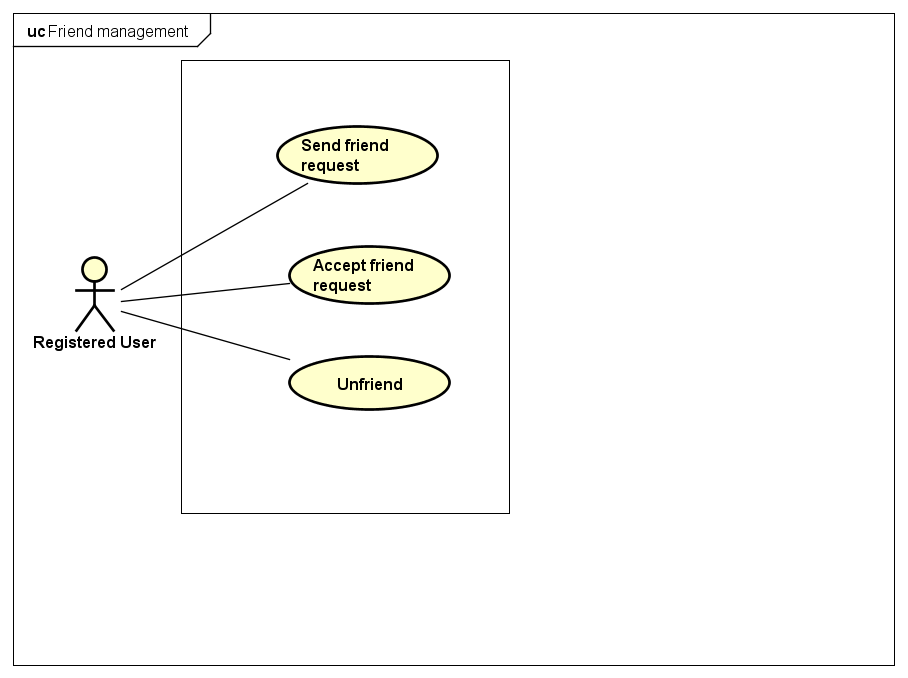


Figure 5.1‑9: Friend Management

|  |  |  |  |
| --- | --- | --- | --- |
| No | Use-case name | Actor | Description |
| 1 | Send friend request | Registered user | Add new registered user as friend |
| 2 | Accept friend request | Registered user | Accept when received friend request |
| 3 | Unfriend | Registered user | Delete a friend from friend list |

#### Plan Management

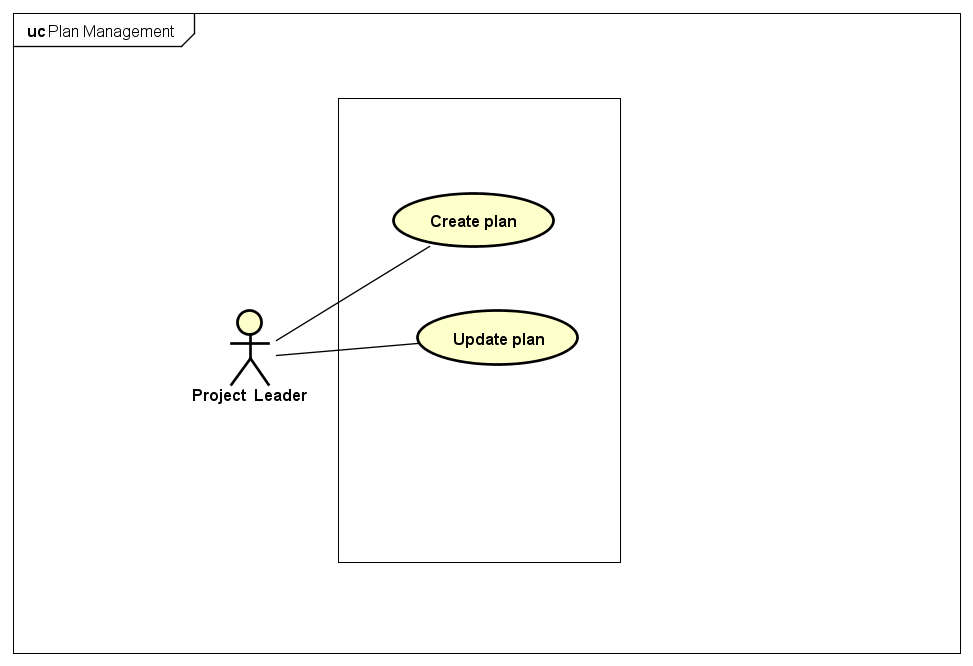


Figure 5.1‑10: Wish List

|  |  |  |  |
| --- | --- | --- | --- |
| No | Use-case name | Actor | Description |
| 1 | Create plan in project | Project Leader | Create a plan in Planning section in Project |
| 2 | Update plan in project | Project Leader | Update plan. |

#### Group management

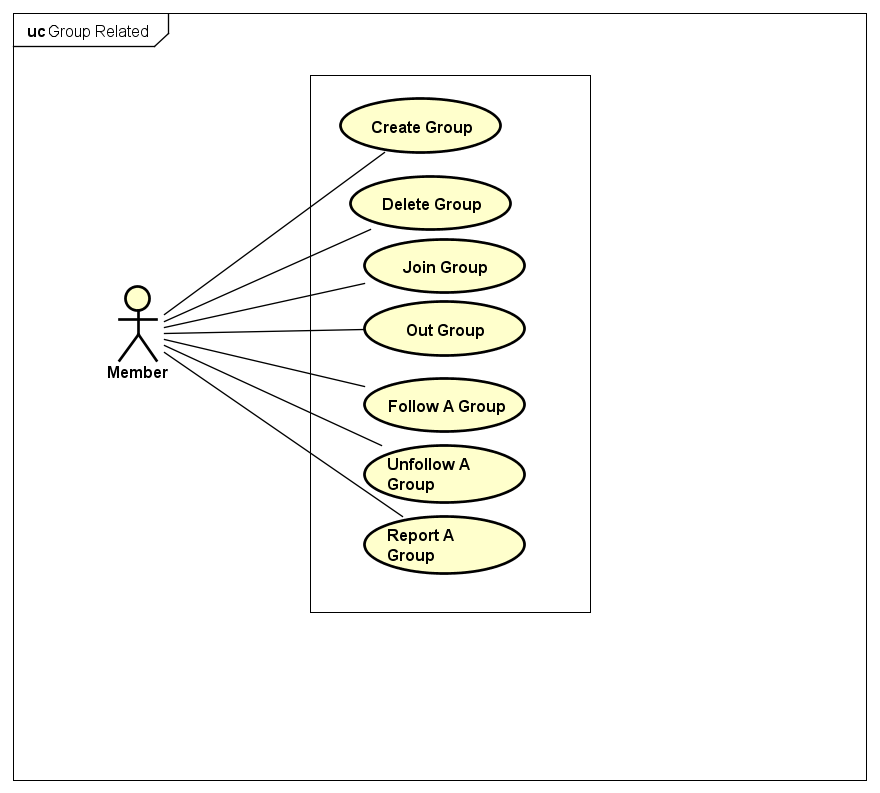


Figure 5.1‑11: Group Management

|  |  |  |  |
| --- | --- | --- | --- |
| No | Use-case name | Actor | Description |
| 1 | Create Group | Registered user | User creates a group in the system for gathering member to do volunteering activities. |
| 2 | Close Group | Group Leader | User closes Group. |
| 3 | Request Join Group | Registered user | User send join request to Group Leader |
| 4 | Leave Group | Member | User leaves Group |
| 5 | Follow a Group | Registered user | User follow a Group |
| 6 | Unfollow a group | Registered user | User unfollow a Group |
| 7 | Report a group | Registered user | User reports a Group |

#### Project management

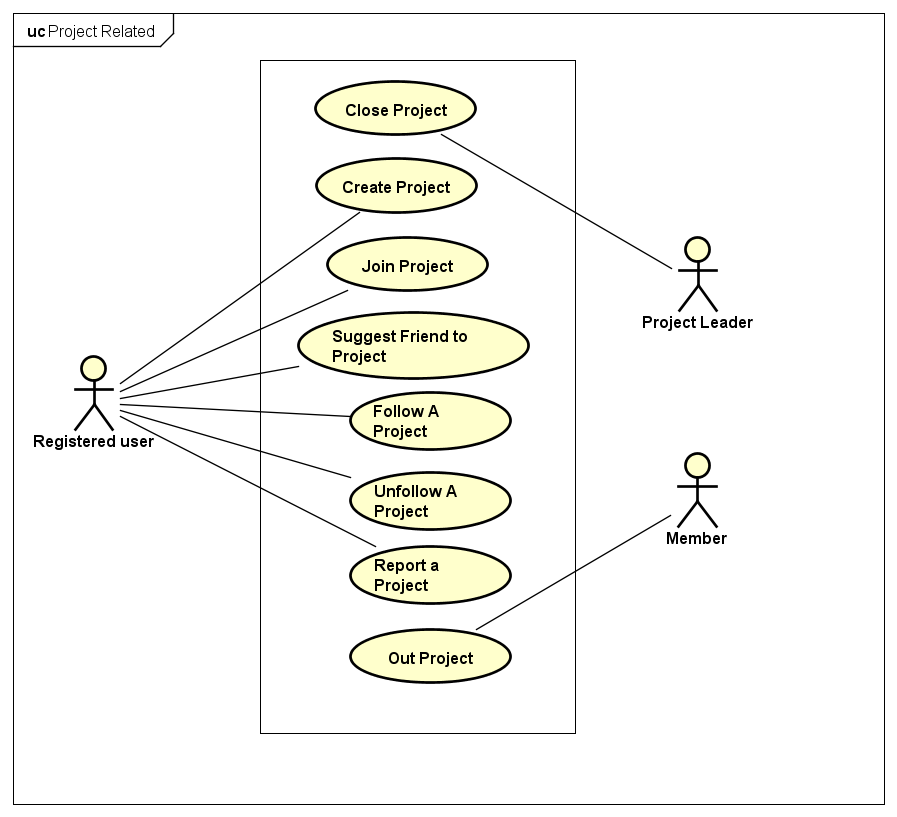
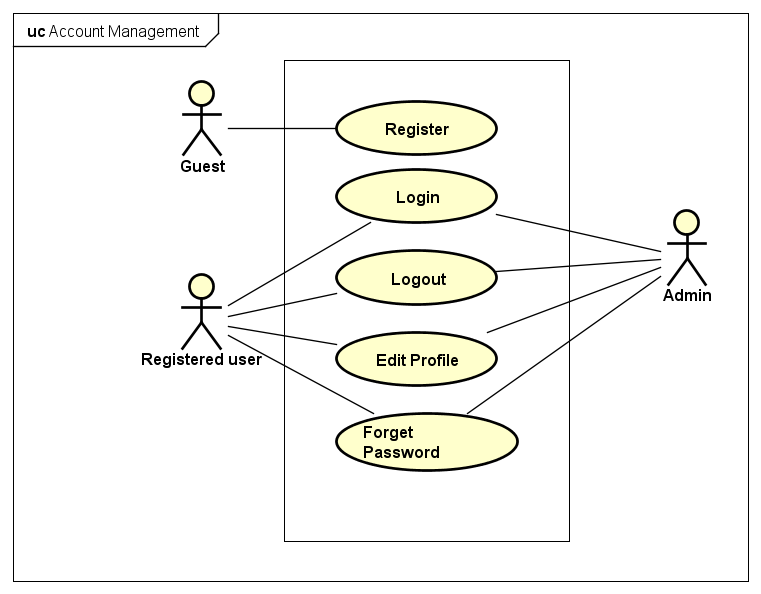


Figure 15: Project Management

|  |  |  |  |
| --- | --- | --- | --- |
| No | Use-case name | Actor | Description |
| 1 | Create Project | Registered user | User creates a volunteering Project in the system. |
| 2 | Close Project | Member | User closes Project. |
| 3 | Request Join Project | Registered user | User send join request to Project Leader. |
| 4 | Leave Project | Member | Member leaves Project. |
| 5 | Follow a Project | Registered user | User follow a Project |
| 6 | Unfollow a Project | Registered user | User unfollow a Project |
| 7 | Report a Project | Registered user | User report an untruthful Project. |
| 8 | Suggest friend to Project | Registered user | User suggest his friend to join Project that he is participating in. |

### Account management



|  |  |  |  |
| --- | --- | --- | --- |
| No | Use-case name | Actor | Description |
| 1 | Login by social network’s account | Registered user | Login to system |
| 2 | Login by registered account | Registered user | Logout |
| 3 | Logout | Registered user | Edit Registered user’s profile |
| 4 | Register | Guest |  |
| 5 | Edit Profile | Registered user | Register account for customers to use more features of the website |
| 6 | Forgot Password | Registered user | Send new password for user |

Figure 5.1‑12: Account Management

### Administrator management

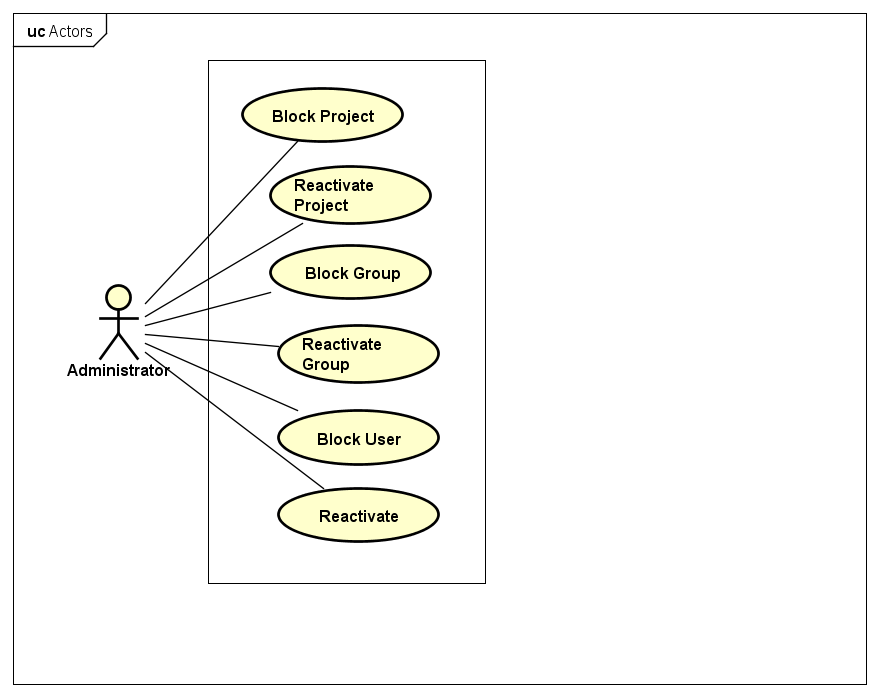


Figure 5.1‑13: Administrator Management

|  |  |  |  |
| --- | --- | --- | --- |
| No | Use-case name | Actor | Description |
| 1 | Block Project | Administrator | Administrator block a untruthful Project |
| 2 | Reactivate Project | Administrator | Administrator reactivate blocked Project |
| 3 | Block Group | Administrator | Administrator block a untruthful Group |
| 4 | Reactivate Group | Administrator | Administrator reactivates blocked Group |
| 5 | Block User | Administrator | Administrator block a untruthful registered User |
| 6 | Reactivate | Administrator | Administrator reactivates blocked user |

# **IMPLEMENTATION VIEW**

## Overview

Implementation View includes Package diagram and Class diagram. Package diagram describes the organization of packages and elements. Class Diagram provides an overview of the target system by describing the objects and classes inside the system and the relationships between them. It provides a wide variety of usages; from modeling the domain-specific data structure to detailed design of the target system

* Controller contain the interface between
* Associated models
* Associated views
* The input devices (e.g., keyboard, pointing device, time).
* Send commands to the model to update the model's state.
* Model is:
* the domain-specific software simulation
* Or implementation of the application's central structure.
* View deal with everything graphical
* Requests data from their model
* Display the data

## Architecturally Significant Design Packages

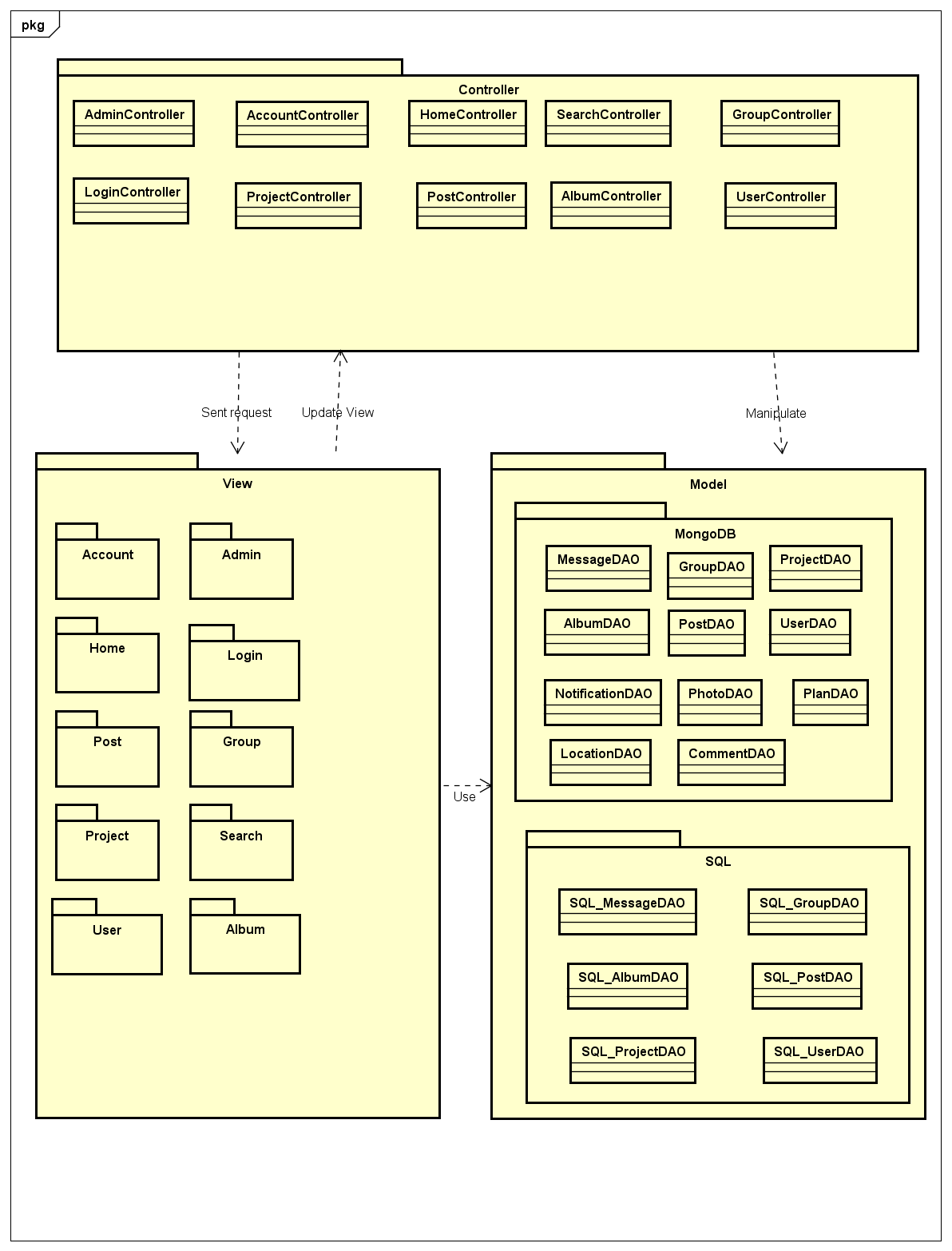


Figure 6.2‑1: Package Diagram

* + - * *Model*

|  |  |  |
| --- | --- | --- |
| No | Controller class | Role |
|  | Comment | Description entity of Comment in database |
|  | Location | Description entity of Location in database |
|  | Notification | Description entity of Notification in database |
|  | Post | Description entity of Post in database |
|  | Message | Description entity of Message in database |
|  | Photo | Description entity of Photo in database |
|  | Project | Description entity of Project in database |
|  | Plan | Description entity of Plan in database |
|  | Group | Description entity of Group in database |
|  | User Profile | Description entity of User Profile in database |
|  | Album | Description entity of Album in database |

* + - * Controller

|  |  |  |
| --- | --- | --- |
| No | Controller class | Role |
|  | AdminController | * Receive request contact’s information from client. * Handle request from client and call method in Admin Model to get data from Database. * Respond data back to Admin View. |
|  | HomeController | * Receive request contact’s information from client. * Handle request from client and call method in Manage View Model to get data from Database. * Respond data back to Home View. |
|  | LoginController | * Receive request contact’s information from client. * Handle request from client and call method in User Model to get data from Database. * Respond data back to Home View. |
|  | GroupController | * Receive request contact’s information from client. * Handle request from client and call method in GroupModel to get data from Database. * Respond data back to Group View. |
|  | ProjectController | * Receive request contact’s information from client. * Handle request from client and call method in ProjectModel to get data from Database. * Respond data back to ProjectView. |
|  | PostController | * Receive request contact’s information from client. * Handle request from client and call method in Post Model to get data from Database. * Respond data back to Post View. |
|  | AccountController | * Receive request contact’s information from client. * Handle request from client and call method in User Model to get data from Database. * Respond data back to Account View. |
|  | SearchController | * Receive request contact’s information from client. * Handle request from client and call method in User Model to get data from Database. * Respond data back to Search View. |
|  | UserController | * Receive request contact’s information from client. * Handle request from client and call method in User Model to get data from Database. * Respond data back to User View. |
|  | AlbumController | * Receive request contact’s information from client. * Handle request from client and call method in Photo Model and Album Model to get data from Database. * Respond data back to Album View. |

* + - * View

Include many .cshtml file

# **PROCESS VIEW**

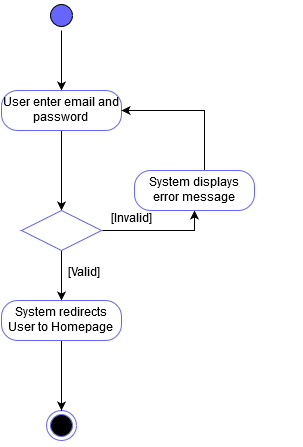


Figure 6.2‑1: Login Activity Diagram

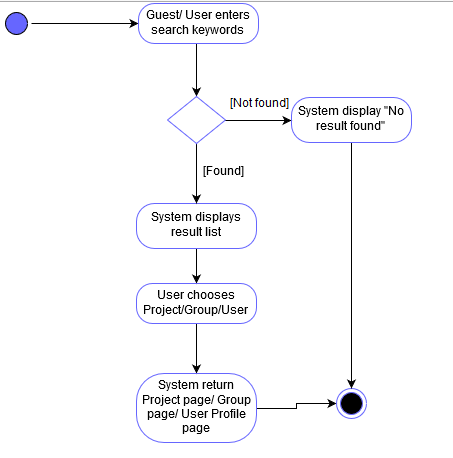


Figure 6.2‑2: Search Activity Diagram

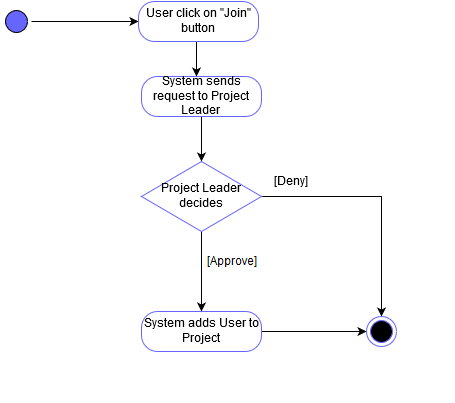


Figure 6.2‑3: Register Activity Diagram

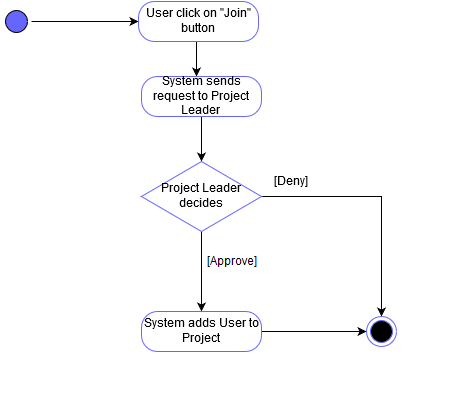


Figure 6.2‑4: Join Project Activity Diagram

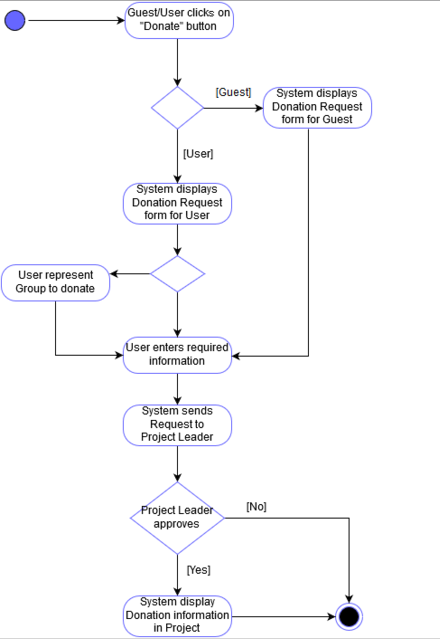


Figure 6.2‑52: Donate Activity Diagram

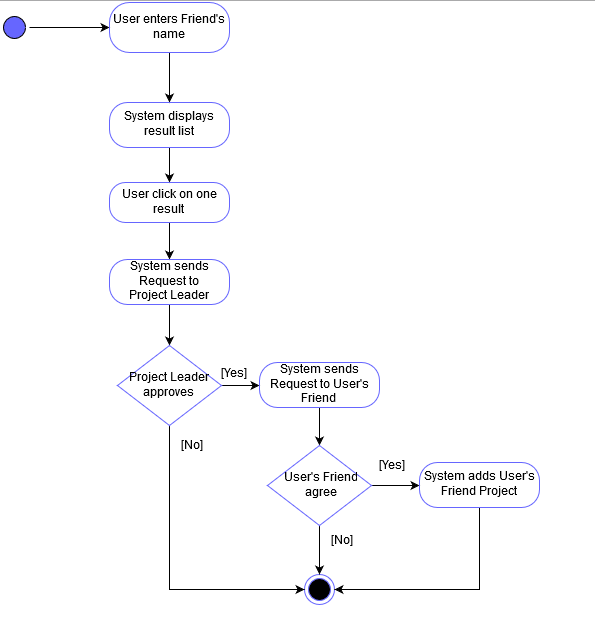


Figure 6.2‑63: Add member Activity Diagram

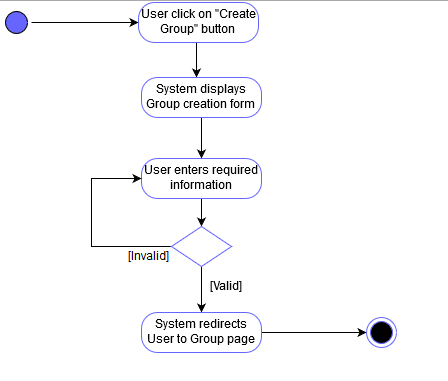


Figure 6.2‑74: Create Group Activity Diagram

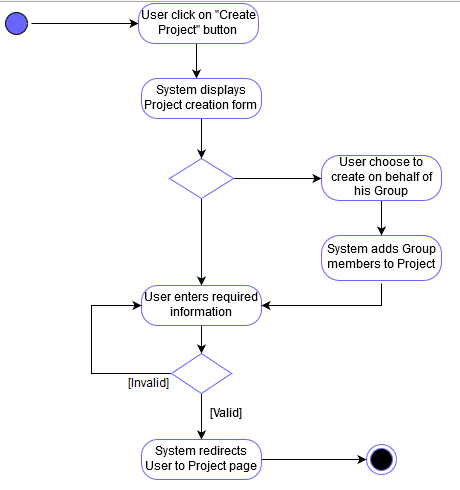


Figure 6.2‑85: Create Project Activity Diagram

# **DEPLOYMENT VIEW**

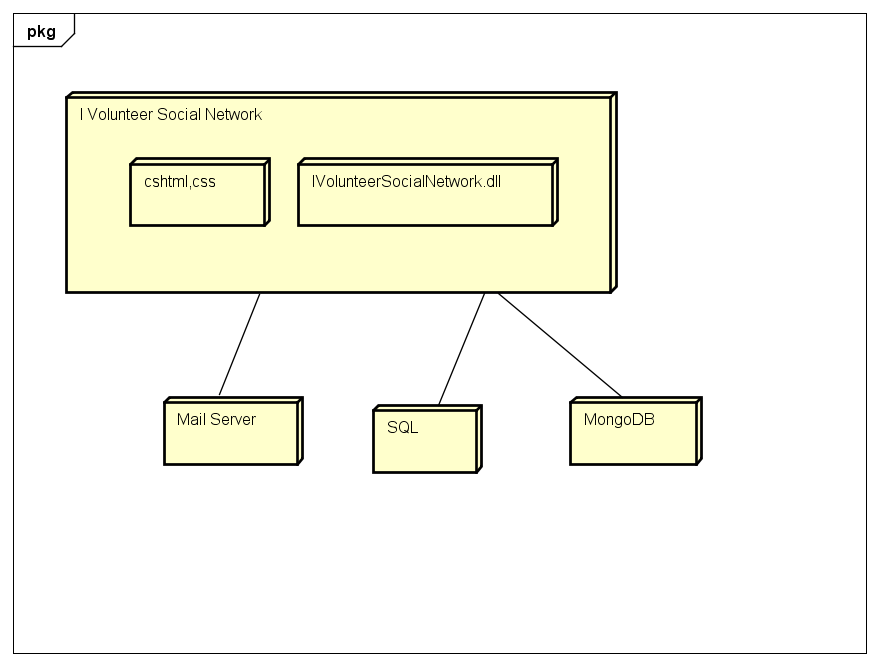


Figure 6.2‑1: Deployment Diagram

# **QUALITY**

Reference to: FAP\_Software requirement specification\_v1.3\_EN