|  |
| --- |
| C:\Users\Tsubaki Yukino\Desktop\FlyAwayPlus Project\Capture.JPG |
| **FLY AWAY PLUS** |
| **Software Architecture Design** |
|  |
| |  |  |  | | --- | --- | --- | | **Fly Away Plus** | | | | **Group Registered users** | Hoàng Nghĩa Đức | SE02882 | | Dương Thanh Hải | SE02856 | | Nguyễn Minh Hoàng | SE02819 | | Trần Mạnh Hiếu | SE02778 | | Phan Tiến Lực | SE02923 | | Lê Minh Thúy | SE02881 | | **Supervisor** | Mr. Nguyễn Văn Sang | | | **Project code** | FAP | | |
|  |

**- Hanoi, 05/2015 –**

**RECORD OF CHANGE**

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Effective Date | Change Items | A\*M, D | Change Description | New Version |
| 27/06/2015 | Use-case view | M | Add use-case | v1.1 |
| Use-case view | M | Add diagram |  |
| 15/07/2015 | Sequence diagram | M | Modify sequence diagram | v1.2 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

*Table 1: Record of change*

**SIGNATURE PAGE**

|  |  |  |
| --- | --- | --- |
| **ORIGINATOR** | Lê Minh Thúy |  |
|  | Developer |  |
|  |  |  |
|  |  |  |
| **REVIEWERS** | Hoàng Nghĩa Đức |  |
|  | Developer |  |
|  |  |  |
|  |  |  |
| **APPROVAL** | Nguyễn Văn Sang |  |
|  | Supervisor |  |
|  |  |  |

Table of Contents

[**1.** **INTRODUCTION** 5](#_Toc420530771)

[1.1. Purpose 5](#_Toc420530772)

[1.2. Scope 5](#_Toc420530773)

[1.3. Definitions, Acronyms, Abbreviations 5](#_Toc420530774)

[1.4. References 5](#_Toc420530775)

[1.5. Overview 5](#_Toc420530776)

[**2.** **CHOICE OF ARCHITECTURE DESIGN** 6](#_Toc420530777)

[2.1. MVC Model 6](#_Toc420530778)

[2.1.1. MVC Model overview 6](#_Toc420530779)

[2.1.2. Advantages and disadvantages of MVC model 7](#_Toc420530780)

[2.1.3. The reason of choosing MVC Model 7](#_Toc420530781)

[2.2. Codeigniter 7](#_Toc420530782)

[2.2.1. Advantages and disadvantages of Codeigniter 7](#_Toc420530783)

[2.2.2. The reason of choosing Codeigniter 8](#_Toc420530784)

[**3.** **ARCHITECTURAL REPRESENTATION** 8](#_Toc420530785)

[**4.** **ARCHITECTURAL GOALS AND CONSTRAINTS** 9](#_Toc420530786)

[**5.** **USE-CASE VIEW** 9](#_Toc420530787)

[5.1. User module 9](#_Toc420530788)

[5.1.1. Common module 9](#_Toc420530789)

[5.1.2. Suggest module 10](#_Toc420530790)

[5.1.3. Social module 10](#_Toc420530791)

[5.1.4. Account management 12](#_Toc420530792)

[5.1.5. Administrator management 12](#_Toc420530793)

[5.1.6. Help center 12](#_Toc420530794)

[**6.** **LOGICAL VIEW** 13](#_Toc420530795)

[6.1. Overview 13](#_Toc420530796)

# **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 Fly Away Plus website created by FAP capstone project team.

## Definitions, Acronyms, Abbreviations

|  |  |  |
| --- | --- | --- |
| **Acronym** | **Definition** | **Note** |
| **FAP** | Fly Away Project |  |
| **MVC** | Model view control |  |
| **IDE** | Integrated development environment |  |
| **Q&A** | Question and answer |  |
| **GUI** | Graphic user interface |  |

## References

* FAP\_ Software Requirements Specification\_v1.0\_EN.docx
* FAP \_Data Design\_v1.0\_EN.docx
* Software Architecture Design Illuminated Book

## 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 FAP is developing as a travelling social network, to make people with same interest can find each other. The system of FAP 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 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.
* MVC in FAP system:
* The Model is the part of the application that handles the logic for the application data.  
  Often model objects retrieve data (and store data) from a database.
* The View is the parts of the application that handles the display of the data.  
  Most often the views are created from the model data.
* The Controller is the part of the application that handles user interaction.  
  Typically controllers read data from a view, control user input, and send input data to the model.

### Advantages and disadvantages of MVC model

* **Advantages:**
* MVC separates system into components, which can be developed, maintained and upgraded individually without pausing system.
* Develop tools is useful and easy to use.
* Large of documentary sources.
* **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 makes parts of system can be developed individually and simultaneously to reduce developing time.
* Better support for test-driven development.
* Tools is useful and documentary source is large makes MVC is easy to develop.
* FAP is a social network, needs to upgrade as user feedback, so we needs to upgrade part of system without affect to 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

* Consistent with FAP system
* Many team member can use

## Graph Database

A graph database is a database that uses graph structures for semantic queries with nodes, edges and properties to represent and store data. A graph database is any storage system that provides index-free adjacency. This means that every element contains a direct pointer to its adjacent elements and no index lookups are necessary. General graph databases that can store any graph are distinct from specialized graph databases such as triple stores and network databases.

Users have friends, friends have posts, posts have comments and likes, each comment has one commenter and each like has one liker. If we store this data in relational database, with the data fully normalize, it would be a seven-table join to get everything out. Seven table joins – with the complexity of O(log(n)\*m)^2 for each join in the best case (all tables are indexed) – would be very slow when the size of the datasets is considerably large. So far, there has been several work-around for querying newsfeed on relational database – the most popular ones are: caching and de-normalizing the tables. However, they all have trade-offs such as lower consistency, anti-updating patterns…etc.

**In FAP system we are using Neo4j, an open-source graph database for storing data.** In Neo4j, everything is stored in form of either an edge, a node or an attribute. Each node and edge can have any number of attributes. Both the nodes and edges can be labelled. Labeling is useful, because you can narrow down your searching area using the labels.

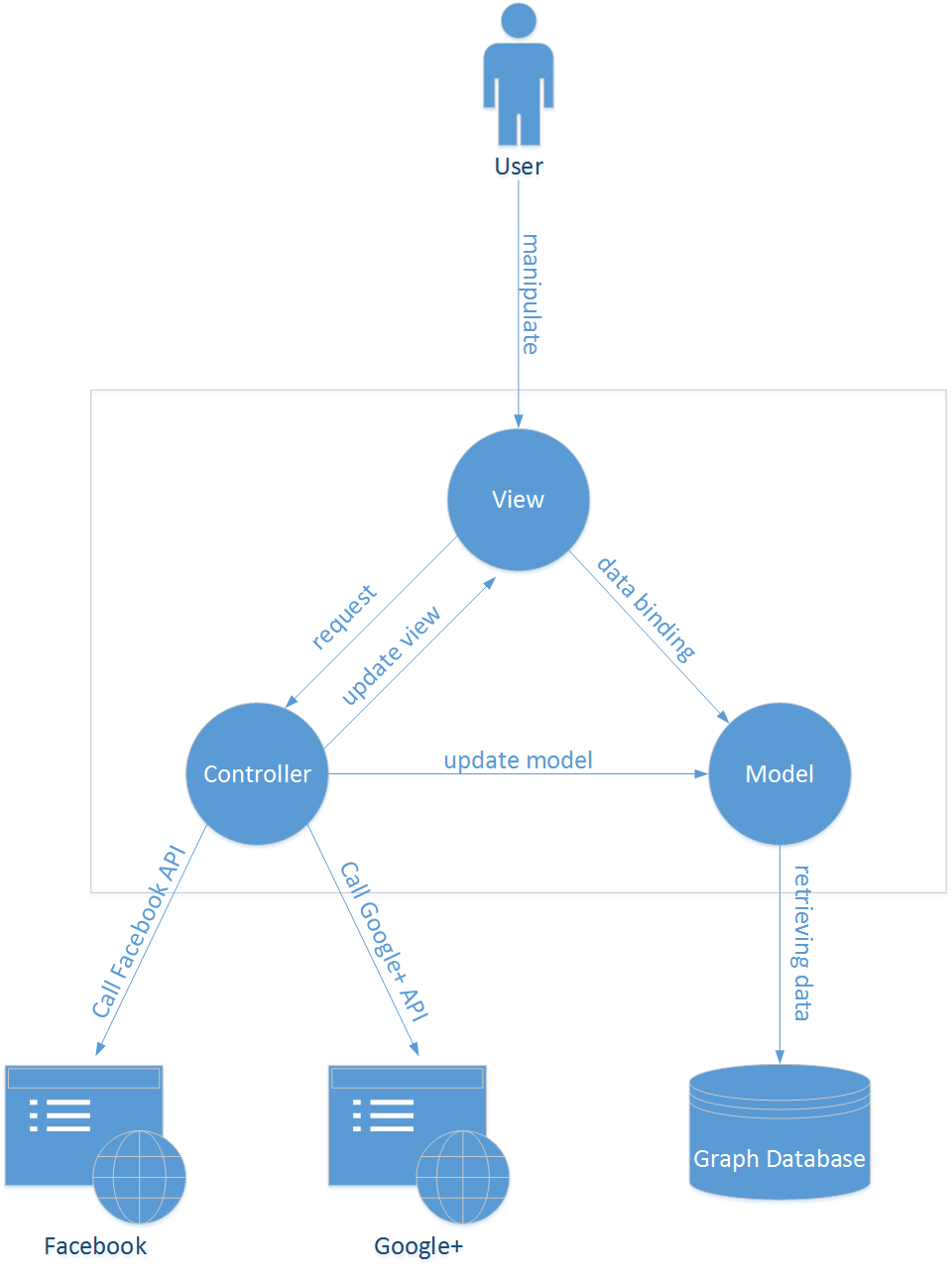
### Advantages and disadvantages of Graph Database

* **Advantages:**
* Faster for associative data sets and map more directly to the structure of object-oriented applications
* Scale more naturally to large data sets as they do not typically require expensive join operations
* Optimized for connection, more suitable to manage ad-hoc and changing data
* Optimized for traversing connected data
* Optimized for seeing the forest, trees, branches and trunks
* Suitable for managing a social network
* **Disadvantages:**
* Sharding (lot of people working on this)
* Team members have to study new techniques

### The reason of choosing Graph Database

* Suitable for managing a social network
* Consistent with FAP system

# **ARCHITECTURAL REPRESENTATION**



**Figure 2**: System overview

We follow MVC architecture to implement the FAP 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 95% 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.

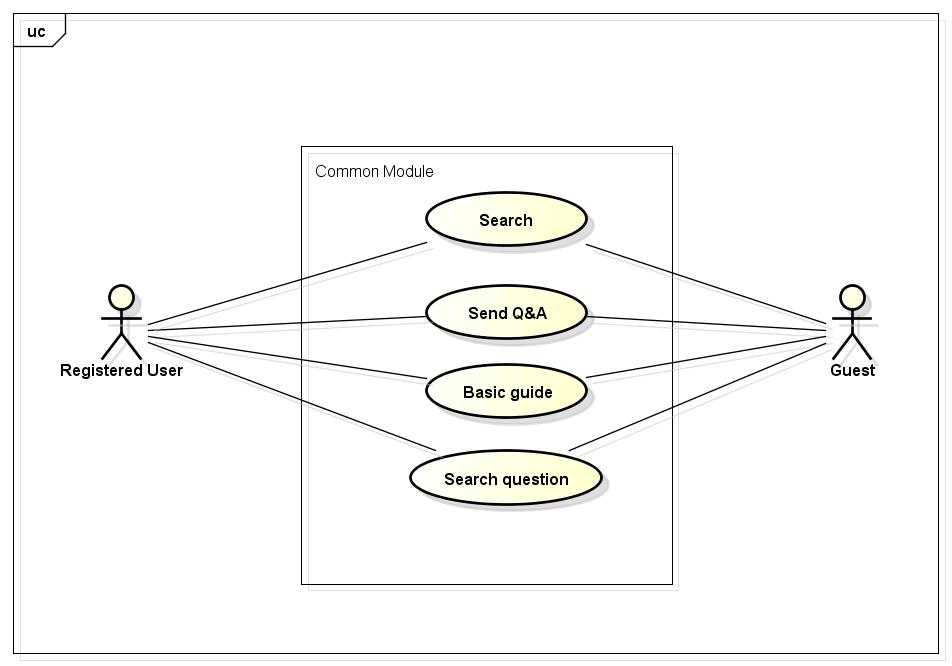
# **USE-CASE VIEW**

* **This application includes two parts:**

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

## User module

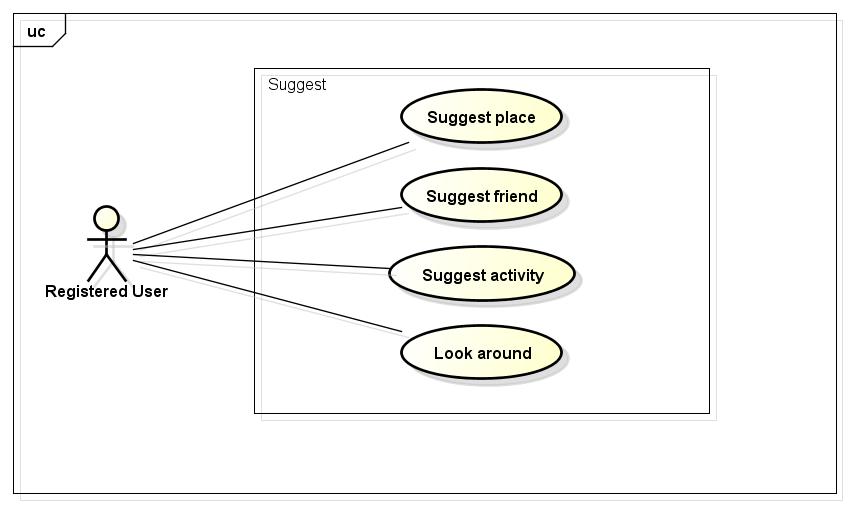
### Common module



**Figure 3**: Common module

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Use-case name** | **Actor** | **Description** |
| 1 | Search by name | Guest, Registered user | Search existing information on FAP system |
| 2 | Send question | Guest, Registered user | Send any questions to admin |
| 3 | Basic guide | Guest, Registered user | Basic guides of using system for guest and registered user |
| 4 | Search question | Guest, Registered user | Search an answer by enter a question. |

### Suggest module

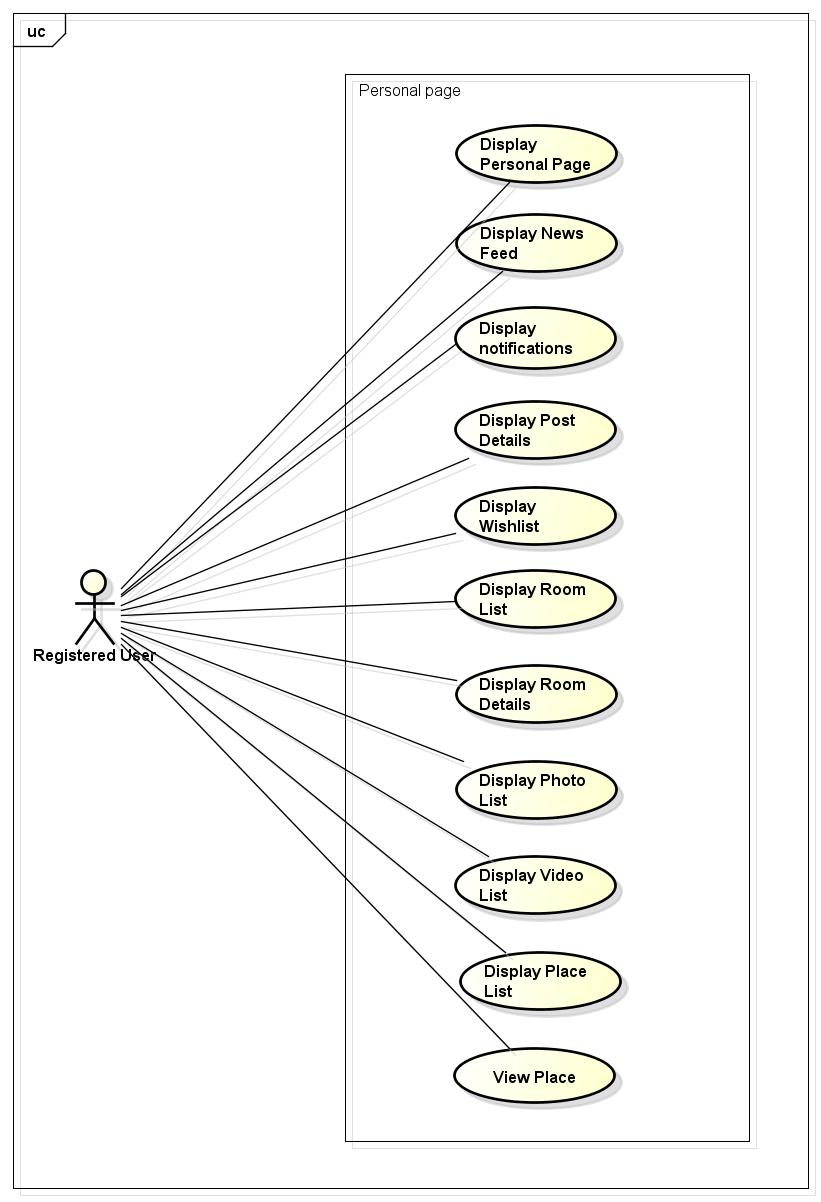


**Figure 4**: Suggest module

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Use-case name** | **Actor** | **Description** |
| 1 | Suggest place | Registered user | Suggest trendy places for user |
| 2 | Suggest friend | Registered user | Suggest new friends for user |
| 3 | Suggest activity | Registered user | Suggest trendy activities for user |
| 4 | Look around | Registered user | Suggest hottest post around current positive of user |

### Social module

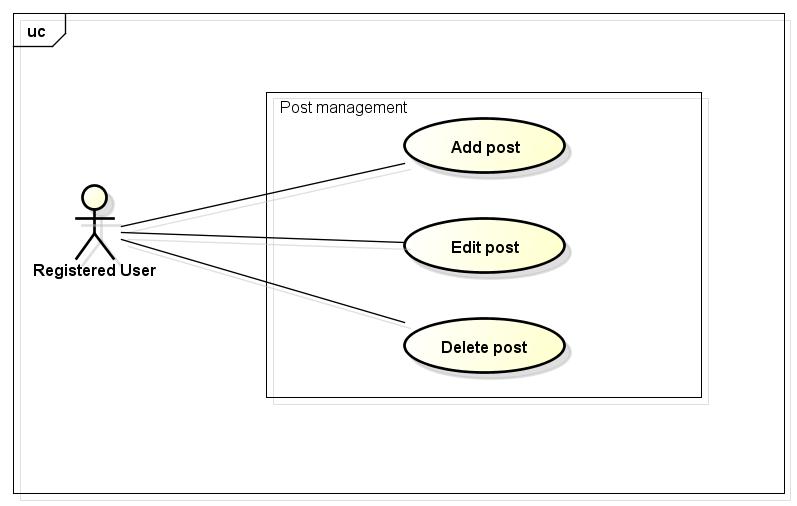
* + - 1. Personal page



**Figure 5:** Personal page

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Use-case name** | **Actor** | **Description** |
| 1 | Display personal page | Registered user | Display posts and photos of users |
| 2 | Display news feed | Registered user | Display user’s friends’ posts and suggestion |
| 3 | Display notifications | Registered user | Display user’s notifications |
| 4 | Display post details | Registered user | Display details of a single post |
| 5 | Display Wish list | Registered user | Display wish list of an user |
| 6 | Display Room list | Registered user | Display list of all Room |
| 7 | Display Room Details | Registered user | Display a details of a Room |
| 8 | Display Photo list | Registered user | Display all photos of an user |
| 9 | Display Video list | Registered user | Display all videos of an user |
| 10 | Display Place list | Registered user | Display all places of an user |
| 11 | View places | Registered user | Display all post related to a place |

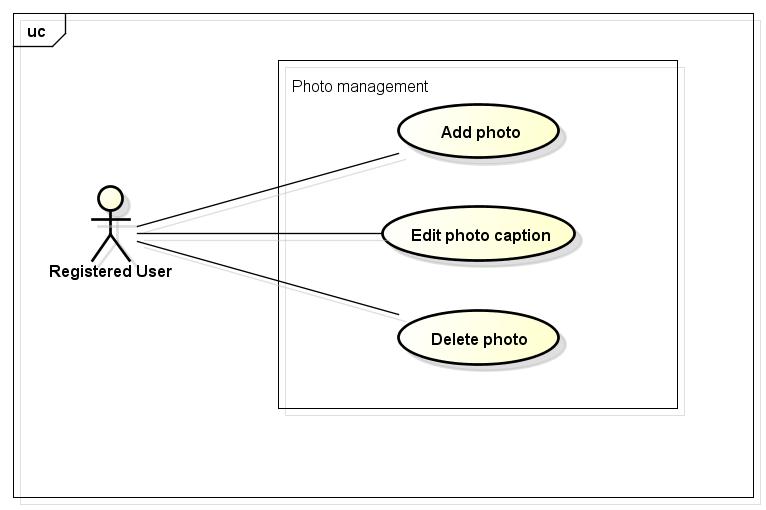
* + - 1. Post management



**Figure 6:** Post management

|  |  |  |  |
| --- | --- | --- | --- |
| No | Use-case name | Actor | Description |
| 1 | Add post | Registered user | Add new post |
| 2 | Edit post | Registered user | Edit existing post |
| 3 | Delete post | Registered user | Delete existing post |

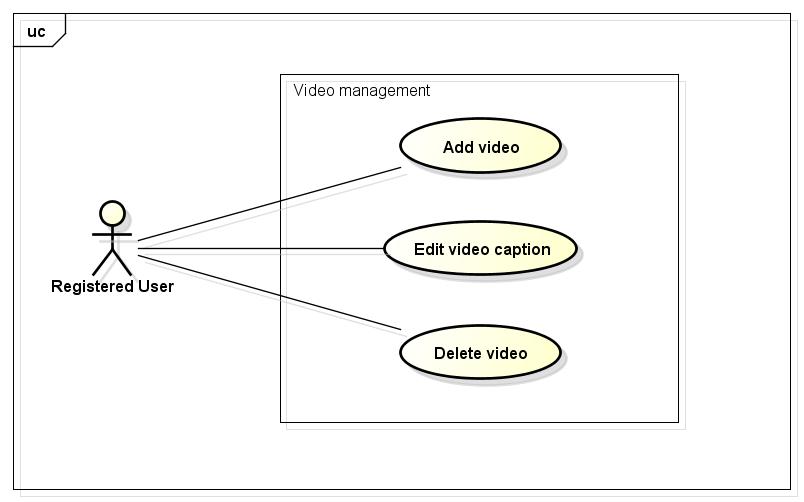
* + - 1. Photo management



**Figure 7:** Photo management

|  |  |  |  |
| --- | --- | --- | --- |
| No | Use-case name | Actor | Description |
| 1 | Add photo | Registered user | Add new photos |
| 2 | Edit photo | Registered user | Edit existing photos |
| 3 | Delete photo | Registered user | Delete existing photos |

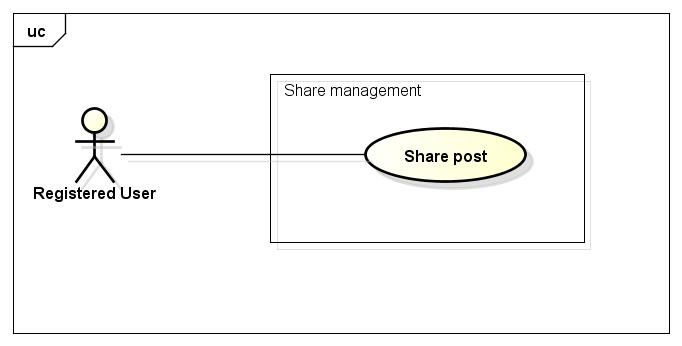
* + - 1. Video management



**Figure 8:** Video management

|  |  |  |  |
| --- | --- | --- | --- |
| No | Use-case name | Actor | Description |
| 1 | Add video | Registered user | Add new video |
| 2 | Edit video | Registered user | Edit existing video |
| 3 | Delete video | Registered user | Delete existing video |

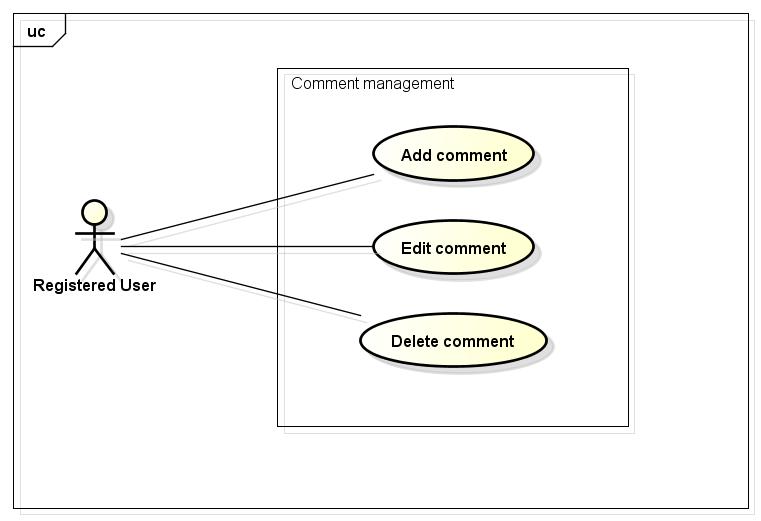
* + - 1. Share



*Figure 9: Share management*

|  |  |  |  |
| --- | --- | --- | --- |
| No | Use-case name | Actor | Description |
| 1 | Share a post/photo/video | Registered user | Share another’s post |

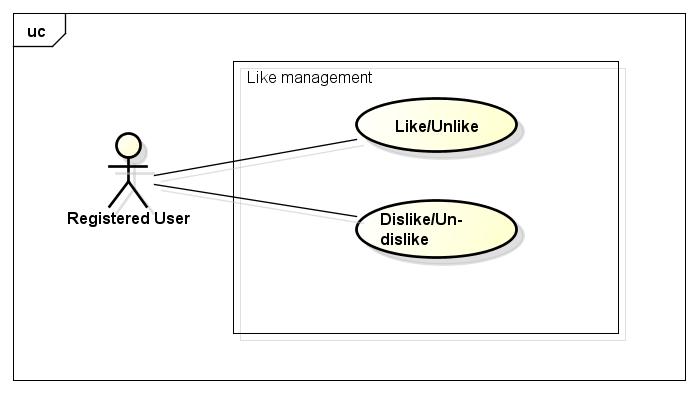
* + - 1. Comment



**Figure 10:** Comment management

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

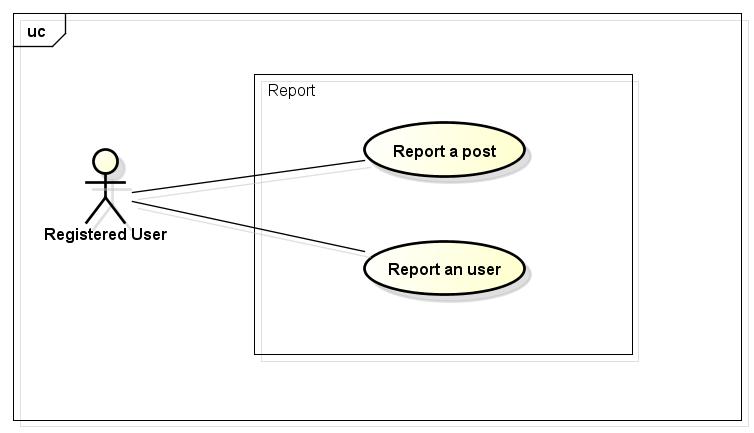
* + - 1. Like management



**Figure 11:** Like management

|  |  |  |  |
| --- | --- | --- | --- |
| No | Use-case name | Actor | Description |
| 1 | Like/Unlike | Registered user | Like/Unlike a post/photo/video/comment |
| 3 | Dislike | Registered user | Dislike/ Un-dislike a post/photo/video/comment |

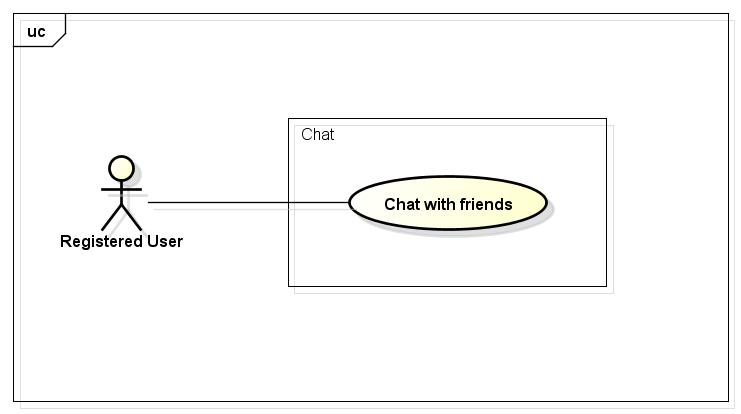
* + - 1. Report



*Figure 12: 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 |

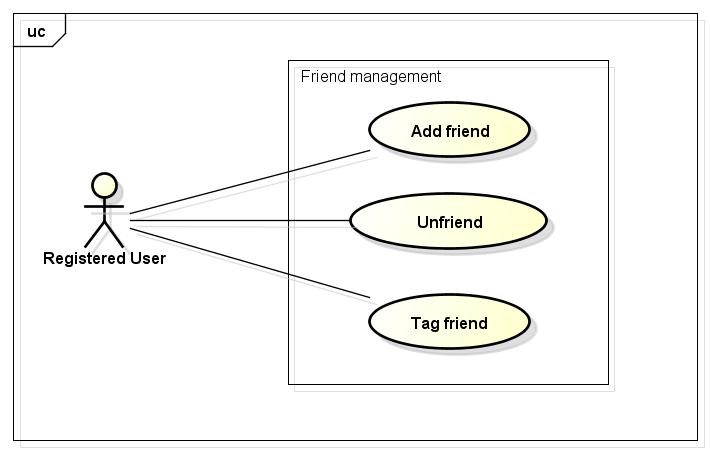
* + - 1. Chat



**Figure 12:** Chat

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

* + - 1. Friend management



**Figure 13:** Friend management

|  |  |  |  |
| --- | --- | --- | --- |
| No | Use-case name | Actor | Description |
| 1 | Add friend | Registered user | Add new registered user as friend |
| 2 | Unfriend | Registered user | Delete a friend from friend list |
| 3 | Tag friend | Registered user | Tag friends on their own post |

* + - 1. Room management



**Figure 14:** Room management

|  |  |  |  |
| --- | --- | --- | --- |
| No | Use-case name | Actor | Description |
| 1 | Join room | Registered user | Join an existing room |
| 2 | Leave room | Registered user | Leave a joined room |
| 3 | Create room | Registered user | Create a new room |
| 4 | Edit room’s information | Registered user | Edit their own room’s information |
| 5 | Create plan/schedule | Registered user | Create plan/schedule for their own room |
| 6 | Edit plan/schedule | Registered user | Edit plan/schedule of their own room |
| 7 | Add member | Registered user | Add member to their own room |
| 8 | Remove member | Registered user | Remove member from their own room |
| 9 | Update schedule status | Registered user | Update schedule status of their room |
| 10 | Chat | Registered user | Chat with other members on room |

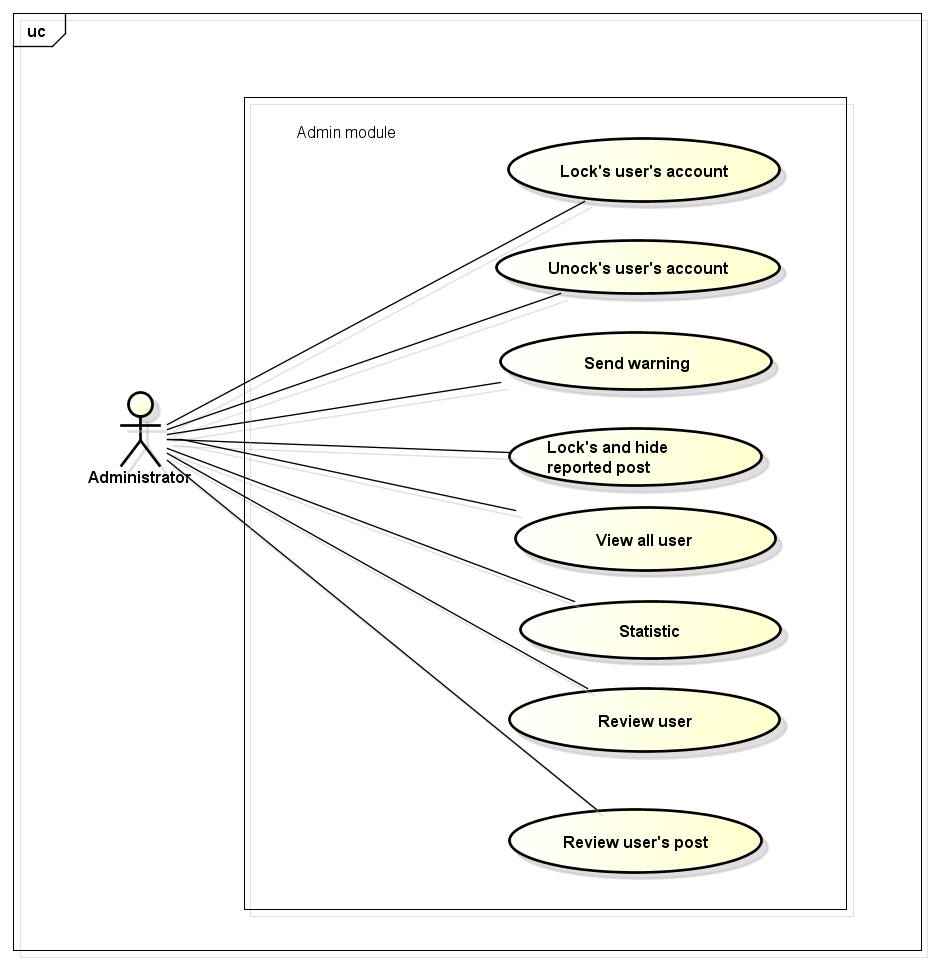
### Account management



*Figure 15: 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 | Registered user | Forgot password |
| 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 |

### Administrator management



**Figure 16:** Admin management

|  |  |  |  |
| --- | --- | --- | --- |
| No | Use-case name | Actor | Description |
| 1 | View all user | Administrator | List all user of FAP |
| 2 | Lock user’s account | Administrator | Review reported user |
| 3 | Unlock user’s account | Administrator | Review reported post |
| 4 | Send warning | Administrator | Reply user/guest’s question |
| 5 | Review user | Administrator | Lock user’s account |
| 6 | Review post | Administrator | Unlock user’s account |
| 7 | Lock and Hide post | Administrator | Lock and hide user’s violated post |
| 8 | Statistics | Administrator | Statistics information of website |

# **LOGICAL VIEW**

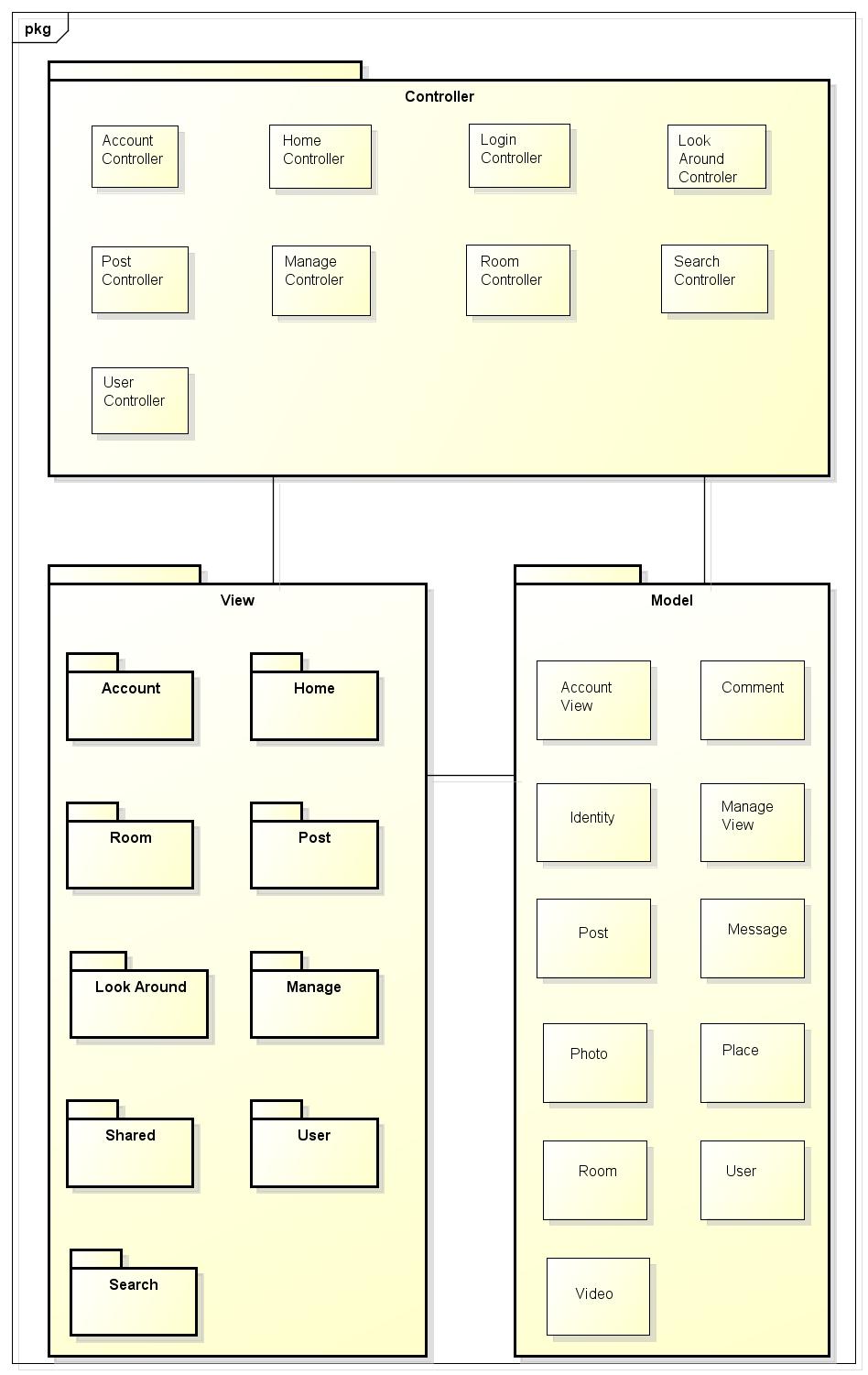
# **IMPLEMENTATION VIEW**

## Overview

Logical 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 17:** Package Diagram

* **Model**

|  |  |  |
| --- | --- | --- |
| No | Controller class | Role |
| 1 | Account | Description entity of Account in database |
| 2 | Home | Description entity of Home in database |
| 3 | Identity | Description entity of Identity in database |
| 4 | Manage View | Description entity of Manage View in database |
| 5 | Post | Description entity of Post in database |
| 6 | Message | Description entity of Message in database |
| 7 | Photo | Description entity of Photo in database |
| 8 | Place | Description entity of Place in database |
| 9 | Room | Description entity of Room in database |
| 10 | User | Description entity of User in database |
| 11 | Video | Description entity of Video in database |

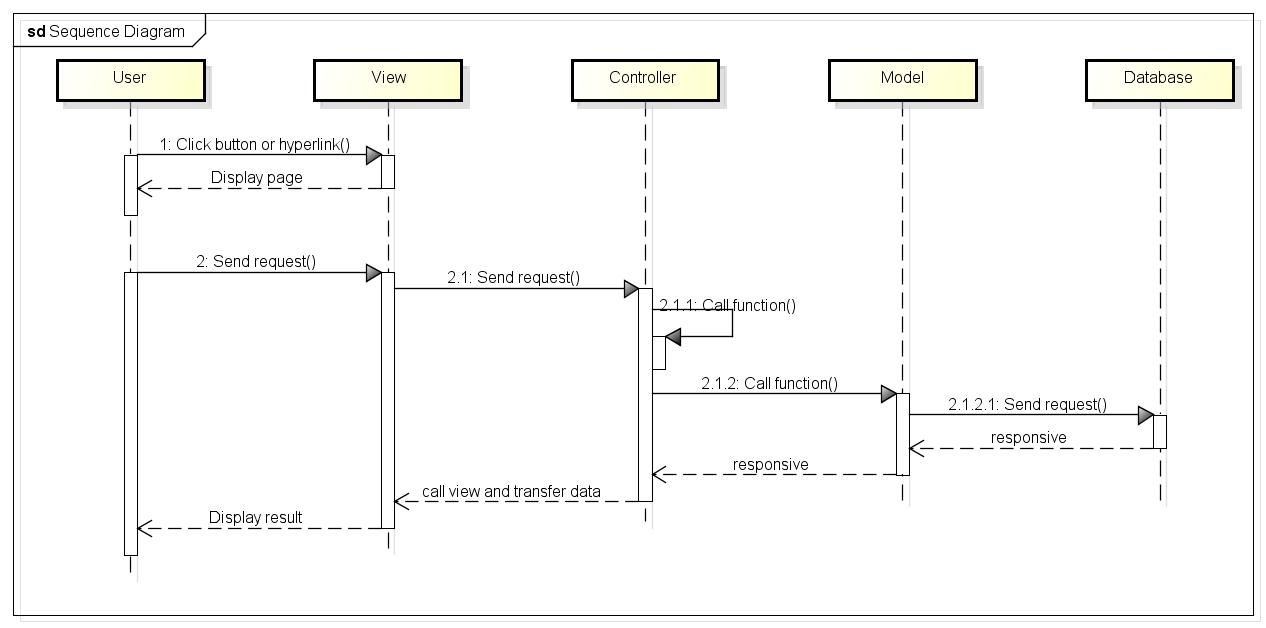
* **Controller**

|  |  |  |
| --- | --- | --- |
| No | Controller class | Role |
| 1 | AccountController | * Receive request contact’s information from client. * Handle request from client and call method in Account Model to get data from Database. * Respond data back to Account View. |
| 2 | 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. |
| 3 | LoginController | * Receive request contact’s information from client. * Handle request from client and call method in Contact Model to get data from Database. * Respond data back to Contact View. |
| 4 | RoomController | * Receive request contact’s information from client. * Handle request from client and call method in Room Model to get data from Database. * Respond data back to Room View. |
| 5 | 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. |
| 6 | LookAroundController | * Receive request contact’s information from client. * Handle request from client and call method in Suggest Model to get data from Database. * Respond data back to Suggest View. |
| 7 | ManageController | * 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 Manage. |
| 8 | SearchController | * 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 Search View. |
| 9 | 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. |
| 10 | AccountController | * Receive request contact’s information from client. * Handle request from client and call method in Account Model to get data from Database. * Respond data back to Account View. |
| 11 | 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. |

* **View**

Include .cshtml files

# **PROCESS VIEW**



**Figure 18**: Sequence diagram

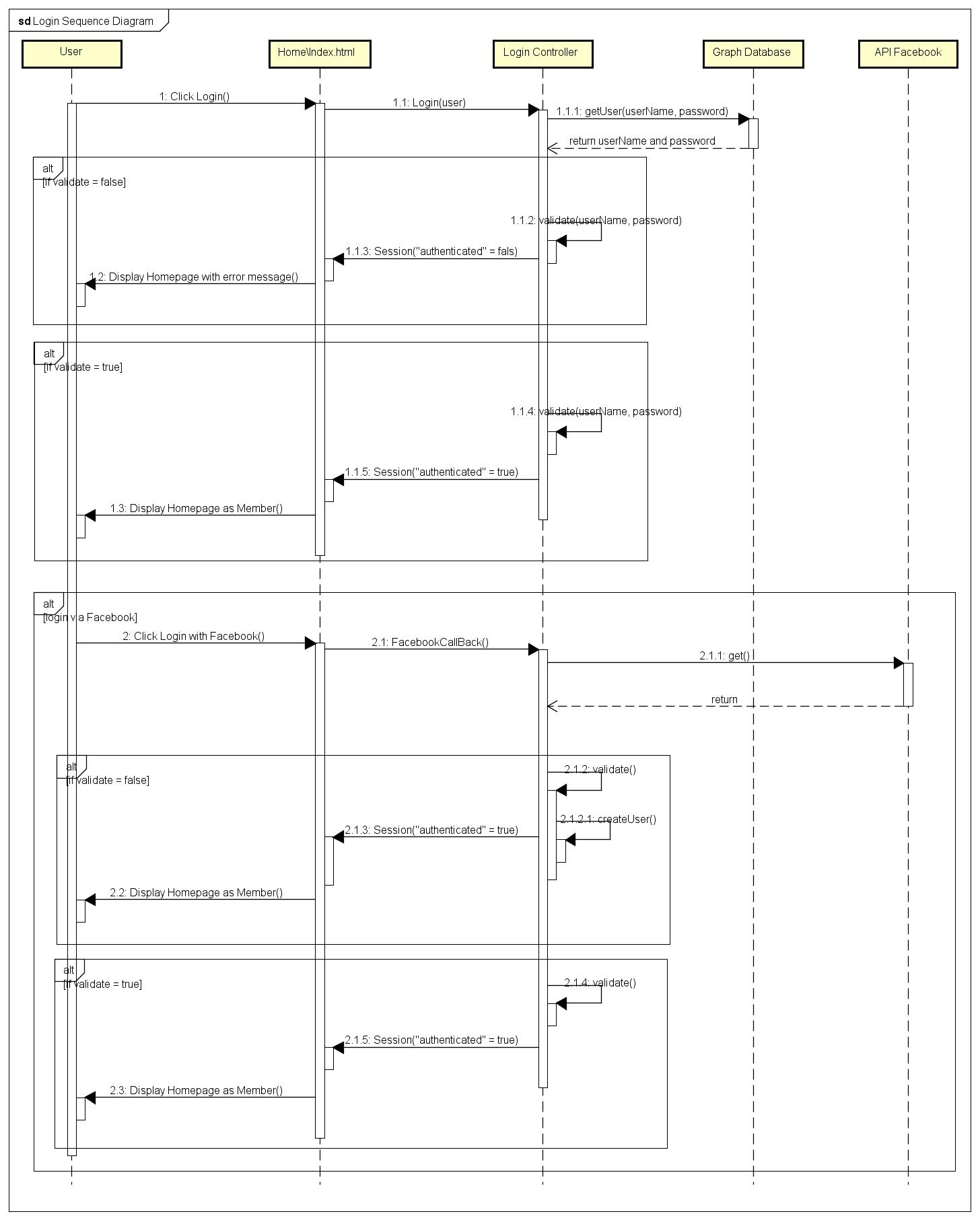
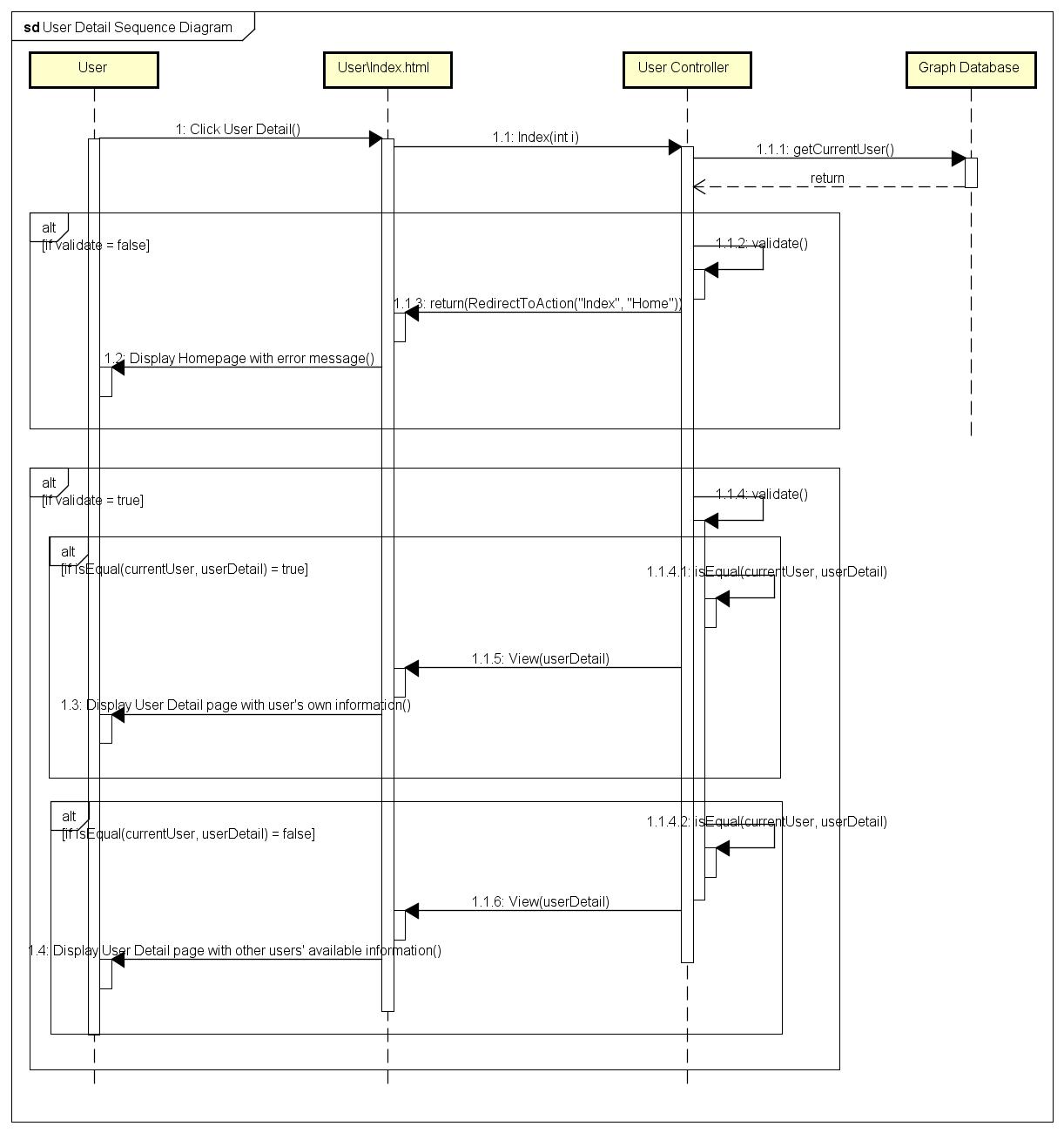


Figure 19: Login diagram

* User input data(username, password) and click Login, a login request is sent to Login Controller via function Login(user)
* Function Login(user) call function GetUser(username, password) to get user name and password from Graph Database and return user name and password.
* User name and password are validated by function Validate(userName, password). If user name and password are blank because of non-existence or do not match what user has entered, this function will return false, else true.
* Depend on what Login Controller return, Homepage will be displayed as Member or be displayed with error message.
* User can also Login via Facebook by clicking Login with Facebook. A login via Facebook is sent to Login Controller via function FacebookCallBack().
* Function FacebookCallBack() send a request to Facebook API, and receive data.
* Function Validate() will check if this user have been logged into system or not. If not, function CreateUser() is called to register new user then Homepage is displayed as Member, else display Homepage instead.



**Figure 20**: User Detail Sequence Diagram

* User can view user’s detail by clicking User Detail from Header or User Menu, a request is sent to User Controller via function Index(int i).
* Function GetCurrentUser() is called to get ID of current user, and return an integer.
* User Controller call function Validate() to check if user with received ID is available (not deleted). If not available, Homepage with error message is displayed.
* If available, function IsEqual() is called to check if user request to view own detail. If right, user’s own detail page is displayed with full information.
* If not, other user’s detail page is displayed with available information.

# **DEPLOYMENT VIEW**

# **QUALITY**

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