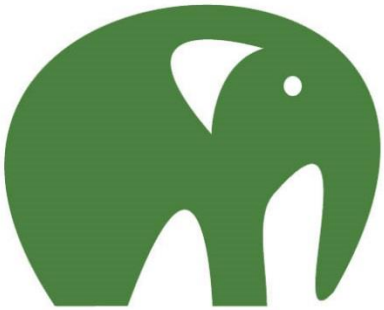


HEC mitigation
And
Information sharing System

ETH PAURA

Software Design Specification

Rajarata University of Sri Lanka

Faculty of Applied Sciences

Information & Communication Technology

3.8.3.10. Add Video
3.8.3.11. Remove video
3.8.3.12. Add posts
3.8.3.13. Edit posts
3.8.3.14. Delete Posts
3.8.3.15. Upload Document
3.8.3.16. Delete Document
3.8.3.17. Ask Questions
3.8.3.18. Delete Questions
3.8.3.19. Edit Questions
3.8.3.20. Answer Questions
3.8.3.21. Edit Answer
3.8.3.22. Download
3.8.3.23. Remove User
3.8.3.24. Add User
3.8.3.25. Change Priviledge levels by admin
3.8.3.26. Remove posts by admin
3.8.3.27. Edit Comments By admin

1. INTRODUCTION

1.1. PURPOSE

This is the Software Design Specification version 1.0 of the third year project, “” also called the “ETHPAURA”. This document covers the all the design aspects of the project starting from the high level components of the system to the class level design. It also includes the sequence diagrams and necessary UI mockups of the project.

This document is intended to be referred frequently in the implementation phase, since this document forms the concrete basis for the implementation.

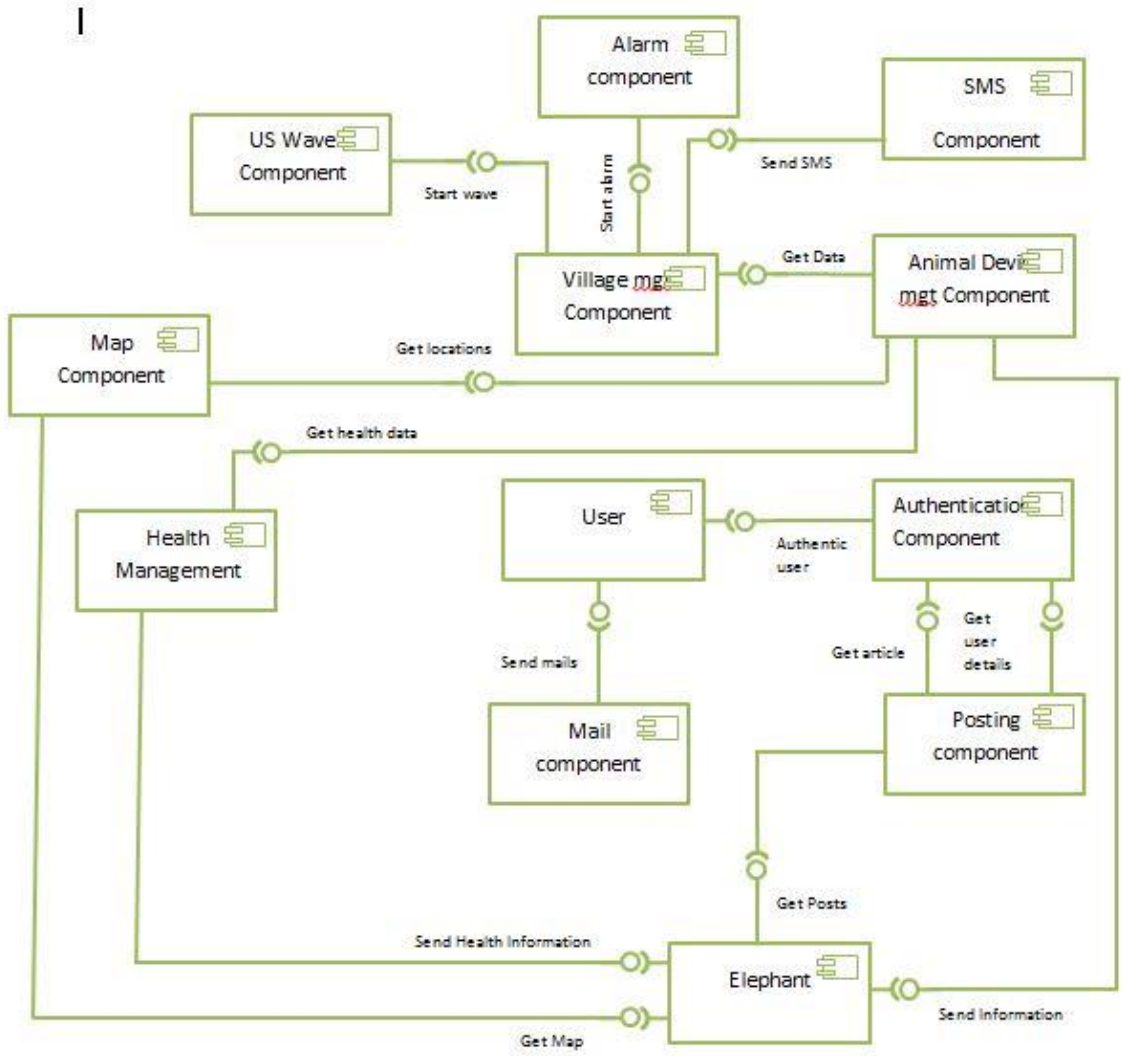
1.2. DOCUMENT CONVENTIONS

Heading Style	Font Size (pt.)	Font Type	Font Color
Master Headings	13	Times New Roman	White
Sub Headings	12	Times New Roman	Black
Other Headings	11	Times New Roman	Black
Body	10	Times New Roman	Black

2. ARCHITECTURAL DESIGN

2.1. HIGH LEVEL COMPONENTS AND THEIR INTERACTIONS

2.1.1 Component (Sub-System) Design



2.1.2 Components

1. Map component

This component has internal functionality of handling maps. Here we use algorithms to simplify and optimize the use of map component. This component in that sense is responsible for handling all map related functions.

2. SMS component

SMS component will be called when there is a need for sending bulk or single SMS s. We hope to use two hosted SMS API from a service provider to implement this feature.

3. Health component

Basically this component is responsible for keeping and handling health related data. Will get the service from Animal Device Manager Component and will mainly provide service to elephant component.

4. User Authentication component

This component can be introduced as one of the key component in the system which bears the burden of security. And group based action restrictions. In that sense this will work as a gateway and will route users along their authorized parts of the system.

5. Mail component

Emailing component is responsible for single or bulk emails when needed. Other components can request the service from this component when needed. All the web registered users are supposed to have an email entered.

6. Village component

This can be identified as a critical component. In the whole system responsible for handling data related to the villages, including whom to notify, when a threat what to notify, when to start other components as necessary. Will serve mainly to elephant device.

7. User component

The main subsystem that has the internal functionality of handling and managing student related data. This component will maintain necessary categorization regarding users. Also this component is responsible for giving services to many other components as well.

8. Posting component.

This component is responsible for publishing posts. Also this is capable of handling multiuser posts. Some of the posts are automatically published by other components such as elephant component. And some of the posts published by users.

9. Elephant component

Main component responsible for all the underline processes in web part. This component is responsible for handling and managing all the tasks related to elephants. Take the use of other components such as mail component etc. and give its service to other components such as post component.

10. Elephant Device Component

Main task of this component is to interpret the data between outside and the system. Will control the data formats and data set sending to the inner system. This is the main controlling body of the system.

2.1.3 Interfaces

Since this system implemented as tight MVC style there will be well defined interfaces between models, controllers and views. But for the sake of modularity, we'll define some interfaces to breakdown the system structure. So below defined interfaces will use all the controllers within particular component as listed under each component.

Map Component

getMap

Will serve elephant maps particular to the selected elephant

Controllers involved

- buildmap controller

SmsComponent

SendSms

Will act as a messenger component andwill send sms through an hostedAPI

Controllers involved

- SmsController

HealthComponent

SendHealthdata

Hold and send raw health data to the elephant component as received

SendProcessedInformation

After analyzing received raw data send it to the elephant component

Controllers involved

- SendHealthDataController
- SendPredictionsController

UserAuthentication

authenticateUser

Do a gatekeeper's duty by authenticating the users and routing them with necessary control list priviledges.

Controllers Involved

- SystemUsersController

Mail Component

SendMail

Serve as a postman. Another component can initiate the service asking to do the job.

Controllers Involved

- EmailsController

Village Component

ThreatDetectorInterpreter

Analyze the village ID and threat level by interpreting the message sent by the village device.

getVillagerContacts(ID,…….)

Get necessary villagers' contact details and make necessary duties to make the environment to send alerts.

SendMessage

Act as an intermediary who sends details necessary to send alert messages to the villagers and if necessary to wildlife officers.

Controllers:-

- villagerController
- threatDetector Controller
- messageprocessController

User Component

getNormalUser

All the services related to non-logged, normal user can be obtained through this interface.

get RegisteredUser

Services related to logged Registered user can be obtained through this interface.

getAdmin

Services related to admin can be obtained through this interface.

Controllers:-

- AdminController
- RegisteredUserController
- NormalUserController

PostingComponent

getPost

This interface is basically for the creation and notification of posts

Controllers:-

- postController
- feedbackController

ElephantComponent

getElephant

All the services related to the elephant is provided through this interface. This is one of the busiest interfaces in the system.

Controllers:-

- ElephantController
- AttacksController
- PredictionsController
- PositionsController

ElephantDeviceManagerComponent

InterpretMessage

The communication among Elephant device and other inner components in the system by doing interpretations.

Controllers:-

- dataManagementController

2.2. ARCHITECTURAL STYLES / PATTERNS

Development of project will using Spring Web MVC framework. The Spring Web model-view-controller (MVC) framework is designed around a DispatcherServlet. In a Spring MVC application, **there can be any number of DispatcherServlets** for various purposes and **each DispatcherServlet has its own WebApplicationContexts**. Furthermore the implementation of this project will do via the Object-Oriented architectural style. That is because all the Models and Controllers in the layered architectural pattern will write as classes and instantiated as objects in the runtime.

Explanation of used architectural styles

2.2.1 Layered Architectural Style

Layered architecture focuses on the grouping of related functionality within an application into distinct layers that are stacked vertically on top of each other. Functionality within each layer is related by a common role or responsibility. Communication between layers is explicit and loosely coupled. Layering our application appropriately helps to support a strong separation of concerns that, in turn, supports flexibility and maintainability.

The reason for the decision:-

Since layered architectural style is one of the best suitable architectural styles for a web application, we tend to use it in our project. As we need to separate our presentation logic from, business logic and our data access logic, the raw mechanism done within the models, controllers and views are explained below.

Model

The Model layer represents the part of our application that implements the business logic. It is responsible for retrieving data and converting it into meaningful concepts for our application. This includes processing, validating, associating or other tasks related to handling data.

View

View that returned from the controller class to a physical view page or JSP page. The View renders a presentation of modeled data. Being separated from the Model objects, it is responsible for using the information it has available to produce any presentational interface our application might need.

Controller

Controller class to handle the web request. The Controller layer handles requests from users. It's responsible for rendering back a response with the aid of both the Model and the View Layer. Controllers can be seen as managers taking care that all needed resources for completing a task are delegated to the correct workers.

Benefits of Spring Web MVC architecture

In Spring MVC can use any object as command or form-backing object, do not need to implement the framework-Specific interface or base class. And also **spring** data binding is highly flexible validation errors can be evaluated by the application not by the System. A Controller is typically responsible for preparing a model Map with data and selecting a view name but it can also write directly to the response stream and complete the request.

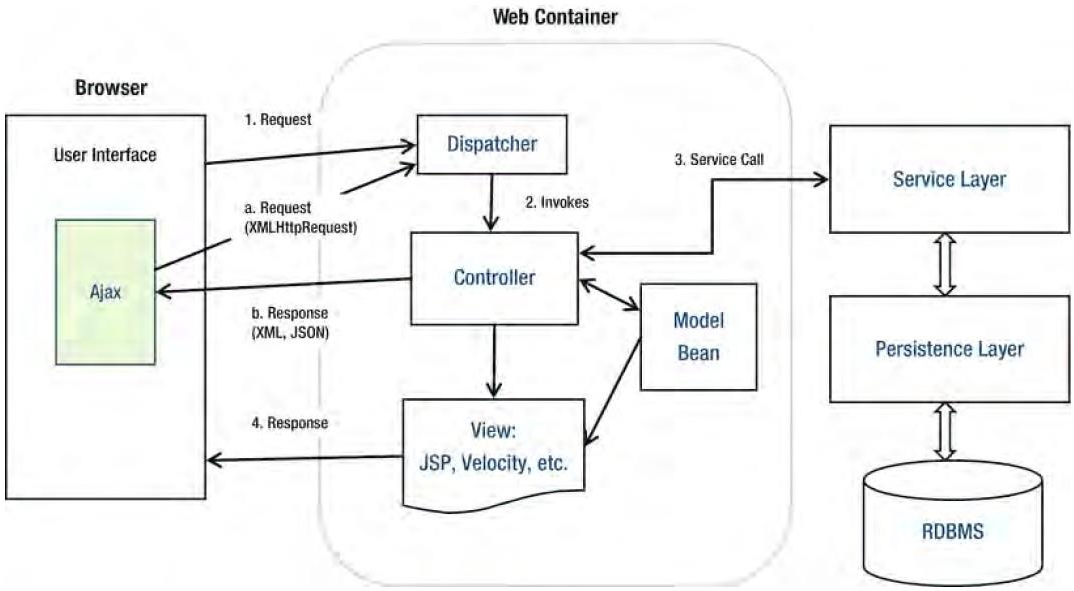


Figure 17-1. The MVC pattern

The illustration of how our system Spring Web MVC architecture works.

Object-Oriented Architectural Style

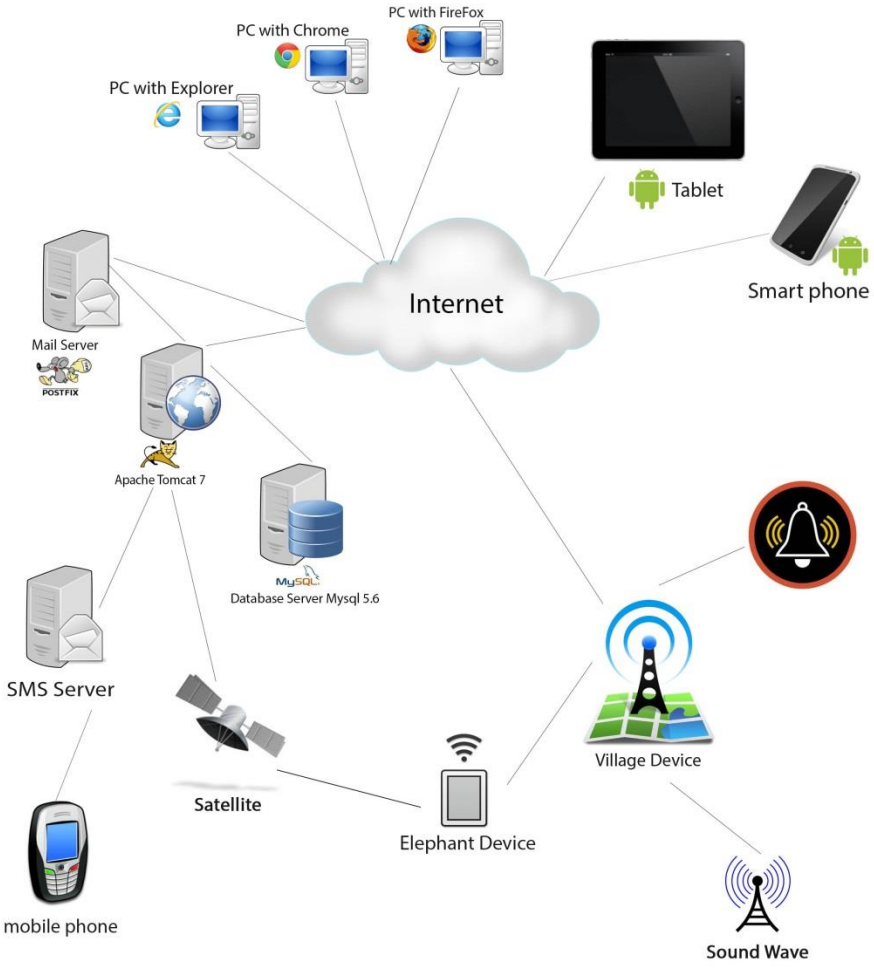
Object-oriented architecture is a design paradigm based on the division of responsibilities for an application or system into individual reusable and self-sufficient objects, each containing the data and the behavior relevant to the object. An object-oriented design views a system as a series of cooperating objects, instead of a set of routines or procedural instructions. Objects are discrete, independent, and loosely coupled; they communicate through interfaces, by calling methods or accessing properties in other objects, and by sending and receiving messages.

The reason for the decision:-Understandable. It maps the application more closely to the real world objects, making it more understandable. Reusable. It provides for reusability through polymorphism and abstraction.

Benefits of using object-oriented architectural style

Testable. It provides for improved testability through encapsulation. Extensible. Encapsulation, polymorphism, and abstraction ensure that a change in the representation of data does not affect the interfaces that the object exposes, which would limit the capability to communicate and interact with other objects. Highly Cohesive. By locating only related methods and features in an object, and using different objects for different sets of features, we can achieve a high level of cohesion.

2.3. PHYSICAL ARRANGEMENTS OF DEVICES



2.4. DESIGN DECISIONS

2.4.1. Spring Web MVC development style is been used

Reason: spring web MVC can be taken as for a popular and easy to handle web application development framework that has the feature of separating the Presentation, Business and Intermediate logics. So using the MVC style, it'll ease the coding and provide well defined interfaces within every logic.

2.4.2. Object Oriented Software development method is used

Reason: - Since this system is used mostly within a specific set of users, we're not expecting millions of hits in to the application within few milliseconds. Rather than we're focusing on the extensibility of the system which will be really helpful for further developments according to requirements. So Object Oriented style is slightly behind from the perspective of performance, it'll satisfy the main target described above.

2.4.3. AJAX (Asynchronous JavaScript and XML) is used

Reason: - AJAX allows web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes. So it provides better interactivity to their users. This is due to the fact that implementing AJAX on a website does not require a page to be reloaded for dynamic content on web pages.

3. COMPONENT AND DETAIL DESIGN

3.1. DESIGN PATTERNS

3.1.1. Prototype Pattern

Declare an abstract base class that specifies a pure virtual "clone" method, and, maintains a dictionary of all "cloneable" concrete derived classes. Any class that needs a "polymorphic constructor" capability: derives itself from the abstract base class, registers its prototypical instance, and implements the clone() operation.

Reason for using Prototype pattern:

When applying access control list enforcements to the user, there will be an abstract model class that break in to two concrete model classes name ARO and ACO. These classes can get in to work by making clone() from the abstract model class.

3.1.2. Observation Pattern

Basically this pattern defines an object that is the "keeper" of the data model or business logic (the Subject). Delegate all "view" functionality to decoupled and distinct Observer objects. Observers register themselves with the Subject as they are created. Whenever the Subject changes, it broadcasts

to all registered Observers that it has changed, and each Observer queries the Subject for that subset of the Subject's state that it is responsible for monitoring.

Reason for using Observer pattern:

In our projects' MVC architectural pattern, The separation of Models and Views done through the decoupling method. So that enhance to ability of increasing flexibility and reusing of code in a way that it reduce the redundant coding. In that fact, we identified that it is very efficient to decouple objects so that changes to one can affect any number of others without requiring the changed object to know details of the others, simply This allows the number and "type" of "view" objects to be configured dynamically, instead of being statically specified.

3.1.2. Composite Design Pattern

The Composite is known as a structural pattern, as it's used to form large object structures across many disparate objects. This pattern allows us to set up a tree structure and ask each element in the tree structure to perform a task. After the tree structure is established, we can then ask each element, to perform a common operation.

Reason for using Composite pattern:

CakePHP has the native support of view elements that can be added separately in to a view which performs a specific view task. For an example we can have specifically implemented set of buttons under an "index" view, where we need navigation to different pages. It clearly specifies that the view "index" is nesting various types of button elements.

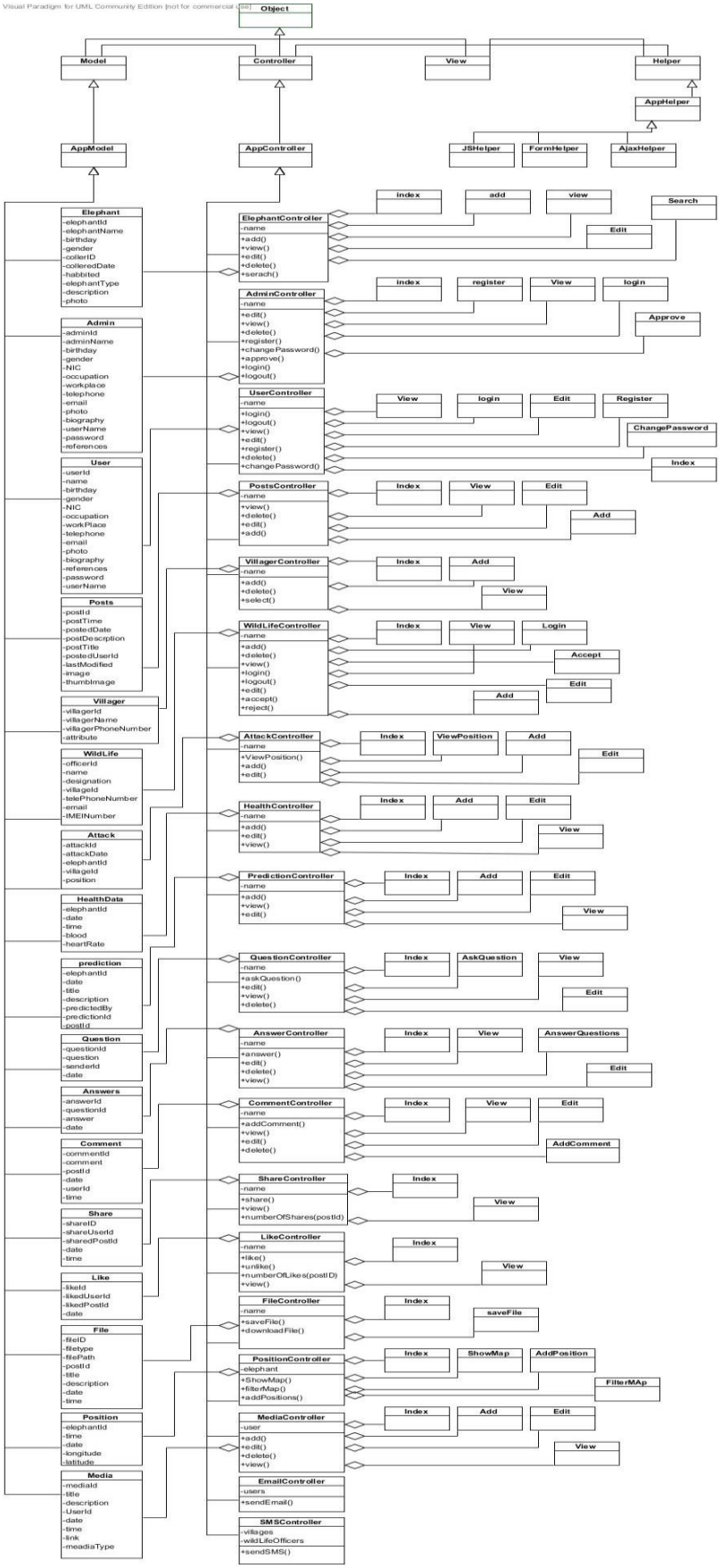
3.1.2. Strategy Design Pattern

This design pattern is very useful when n it comes to define a family of algorithms, encapsulate each one and make them interchangeable. It allows us to change the algorithm independently with out changing the client using it. It converts the generalization of the template method to composition or aggregation.

Reason for using Strategy pattern:

In our planned development in MVC pattern, the View-Controller relationship can be maintained by burying the complex algorithms inside the controllers and let the views to render only the content passed by the controllers. It's useful when we want to replace the algorithm either statically or dynamically, when we have a lot of variants of the algorithm for example the filtering scenario of students using diffident algorithmic factors, or when the algorithm has complex data structures that we want to encapsulate.

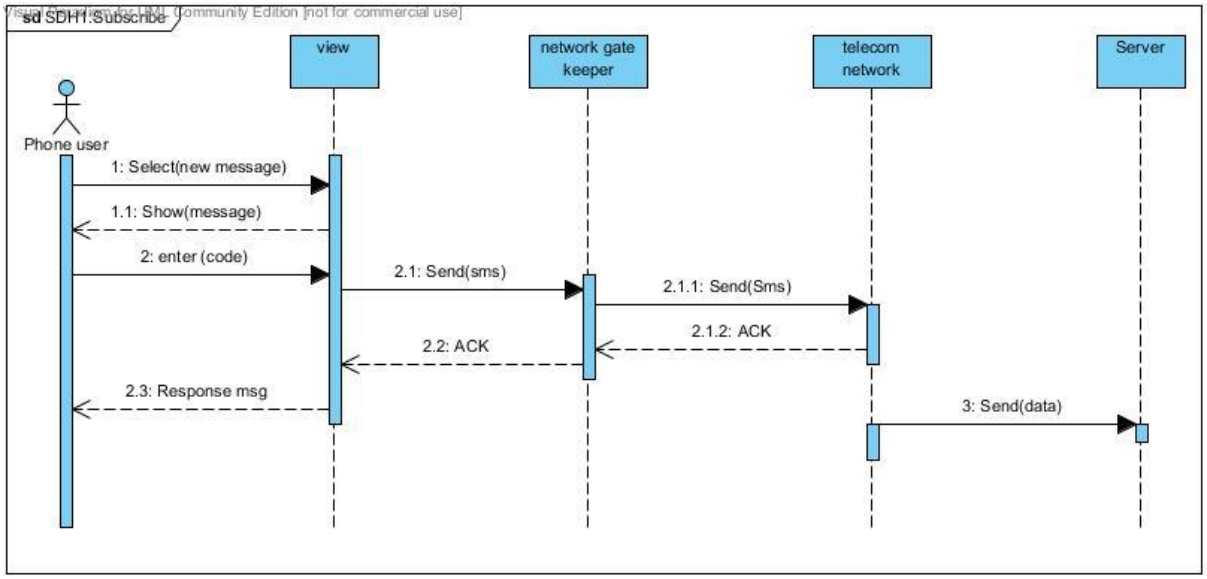
3.2. CLASS DIAGRAM


Visual Paradigm for UML Community Edition [not for commercial use]
Object
Model
Controller
View
Helper
AppHelper
AppModel
AppController
JSHelper
FormHelper
AjaxHelper
Elephant
-elephantId
-elephantName
-birthday
-gender
-collerID
-collerestDate
-habbited
-elephantType
-description
-photo
Admin
-adminId
-adminName
-birthday
-gender
-NIC
-occupation
-workplace
-telephone
-email
-photo
-biography
-userName
-password
-references
User
-userId
-name
-birthday
-gender
-NIC
-occupation
-workPlace
-telephone
-email
-photo
-biography
-references
-password
-userName
Posts
-postId
-postTime
-postedDate
-postDescrption
-postTitle
-postedUserId
-lastModified
-image
-thumbImage
Villager
-villagerId
-villagerName
-villagerPhoneNumber
-attribute
WildLife
-officerId
-name
-designation
-villageId
-telePhoneNumber
-email
-IMEINumber
Attack
-attackId
-attackDate
-elephantId
-villageId
-position
HealthData
-elephantId
-date
-time
-blood
-heartRate
prediction
-elephantId
-date
-title
-description
-predictedBy
-predictionId
-postId
Question
-questionId
-question
-senderId
-date
Answers
-answerId
-questionId
-answer
-date
Comment
-commentId
-comment
-postId
-date
-userId
-time
Share
-shareID
-shareUserId
-sharedPostId
-date
-time
Like
-likeId
-likedUserId
-likedPostId
-date
File
-fileID
-filetype
-filePath
-postId
-title
-description
-date
-time
Position
-elephantId
-time
-date
-longitude
-latitude
Media
-mediaId
-title
-description
-UserId
-date
-time
-link
-meadiaType
ElephantController
-name
+add()
+view()
+edit()
+delete()
+serach()
index
add
view
Search
Edit
AdminController
-name
+edit()
+view()
+delete()
+register()
+changePassword()
+approve()
+login()
+logout()
index
register
View
login
Approve
UserController
-name
+login()
+logout()
+view()
+edit()
+register()
+delete()
+changePassword()
View
login
Edit
Register
ChangePassword
Index
PostsController
-name
+view()
+delete()
+edit()
+add()
Index
View
Edit
Add
VillagerController
-name
+add()
+delete()
+select()
Index
Add
View
WildLifeController
-name
+add()
+delete()
+view()
+login()
+logout()
+edit()
+accept()
+reject()
Index
View
LogIn
Accept
Edit
Add
AttackController
-name
+ViewPosition()
+add()
+edit()
Index
ViewPosition
Add
Edit
HealthController
-name
+add()
+edit()
+view()
Index
Add
Edit
View
PredictionController
-name
+add()
+view()
+edit()
Index
Add
Edit
View
QuestionController
-name
+askQuestion()
+edit()
+view()
+delete()
Index
AskQuestion
View
Edit
AnswerController
-name
+answer()
+edit()
+delete()
+view()
Index
View
AnswerQuestions
Edit
CommentController
-name
+addComment()
+view()
+edit()
+delete()
Index
View
Edit
AddComment
ShareController
-name
+share()
+view()
+numberOfShares(postId)
Index
View
LikeController
-name
+like()
+unlike()
+numberOfLikes(postID)
+view()
Index
View
FileController
-name
+saveFile()
+downloadFile()
Index
saveFile
PositionController
-elephant
+ShowMap()
+filterMap()
+addPositions()
Index
ShowMap
AddPosition
FilterMAp
MediaController
-user
+add()
+edit()
+delete()
+view()
Index
Add
Edit
View
EmailController
-users
+sendEmail()
SMSController
-villages
-wildLifeOfficers
+sendSMS()

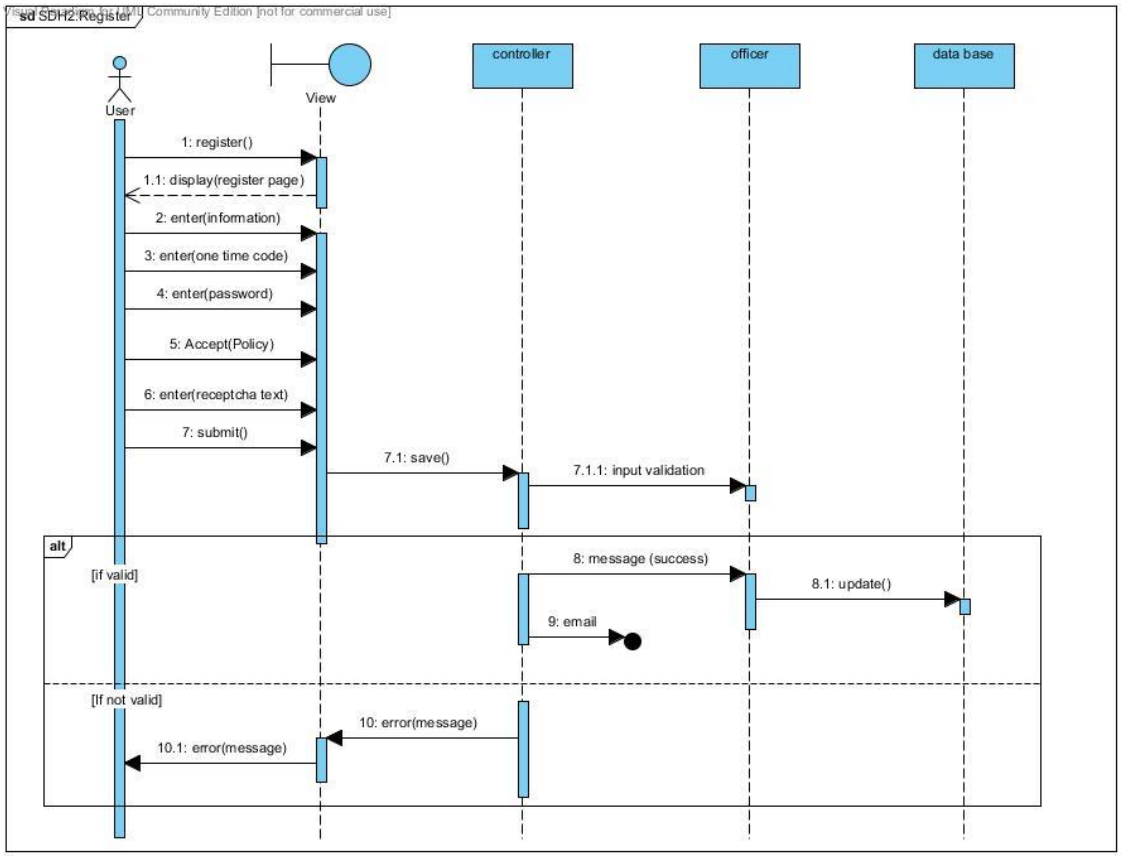
3.3. SEQUENCE DIAGRAMS

3.3.1. HEC Sequence Diagrams

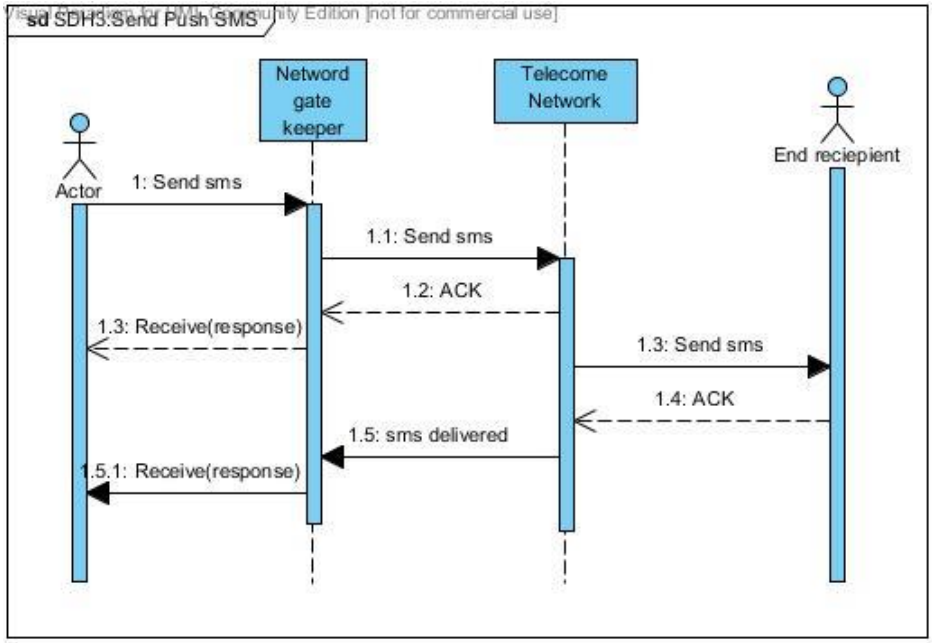
1. Subscribe



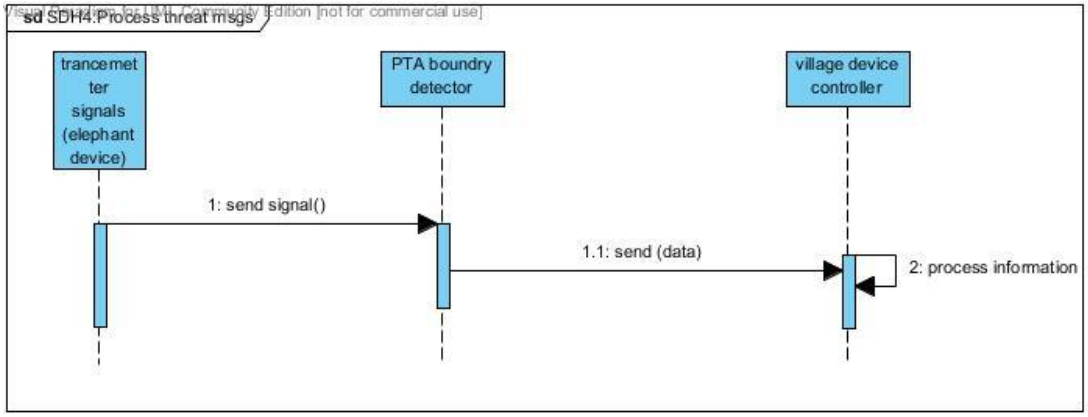
2. Register



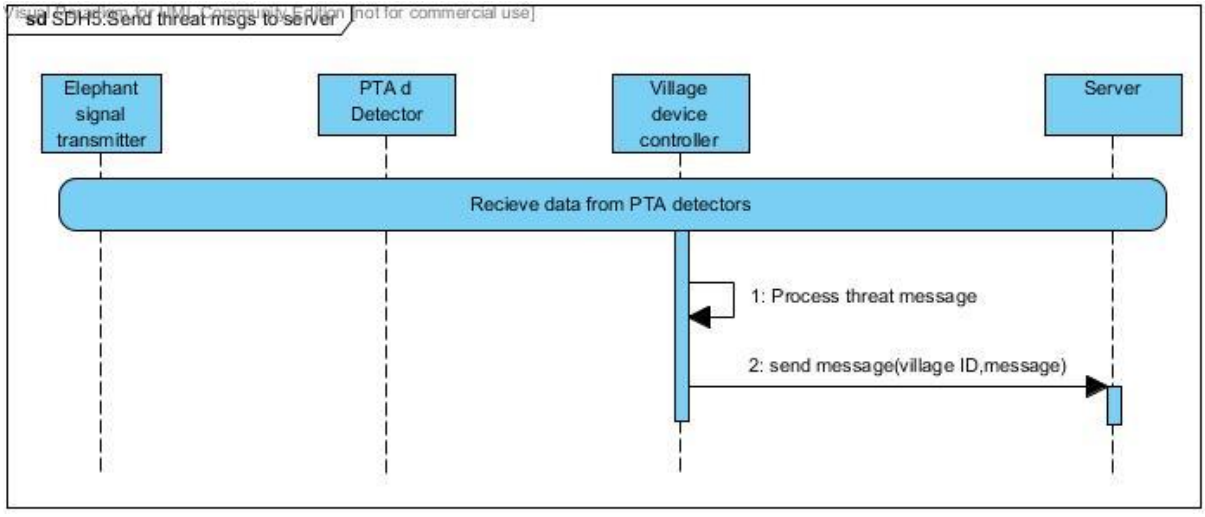
3. Send Push SMS



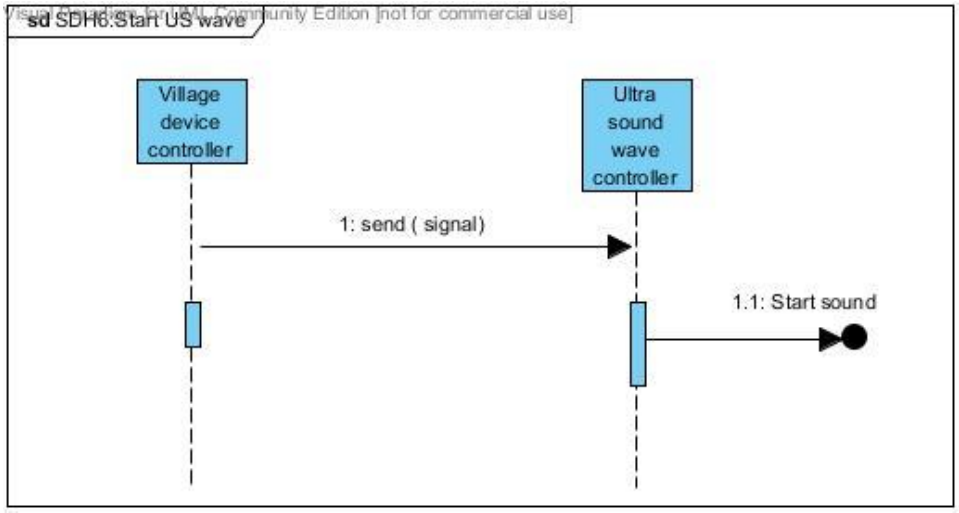
4. Detect Threat and Threat msg Processing



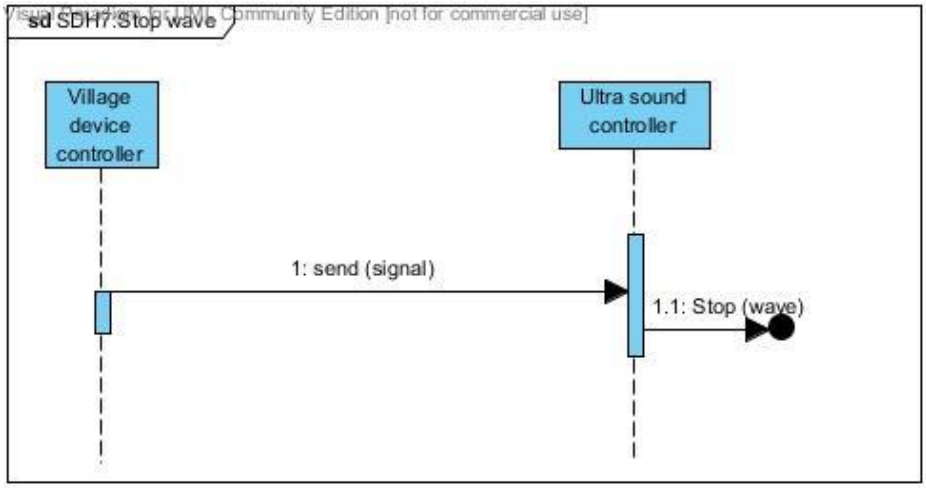
5. Send threat msgs to server



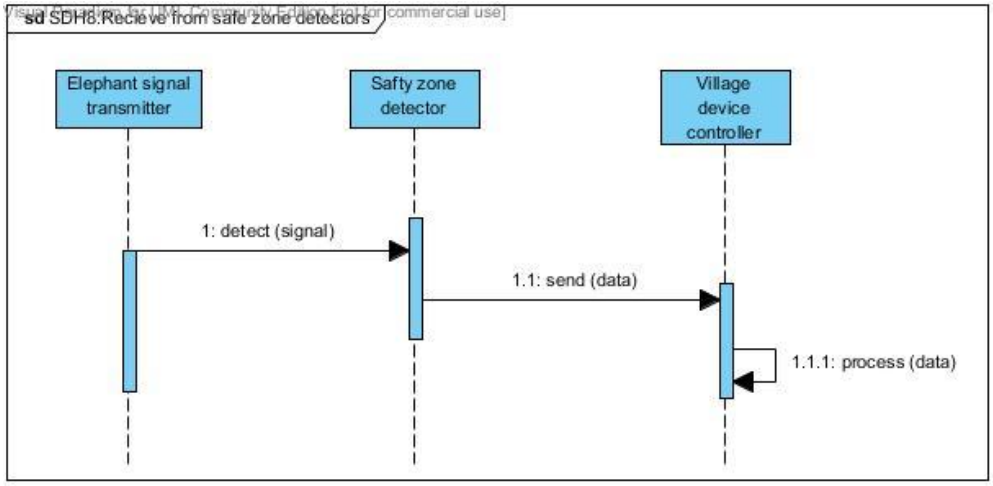
6. Start Ultra sound wave



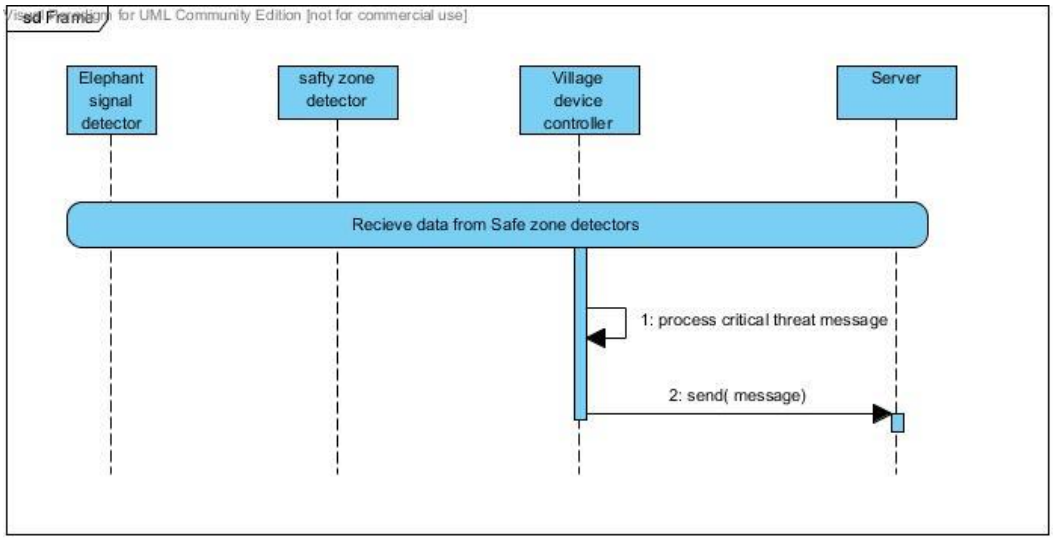
7. Stop Ultra sound wave



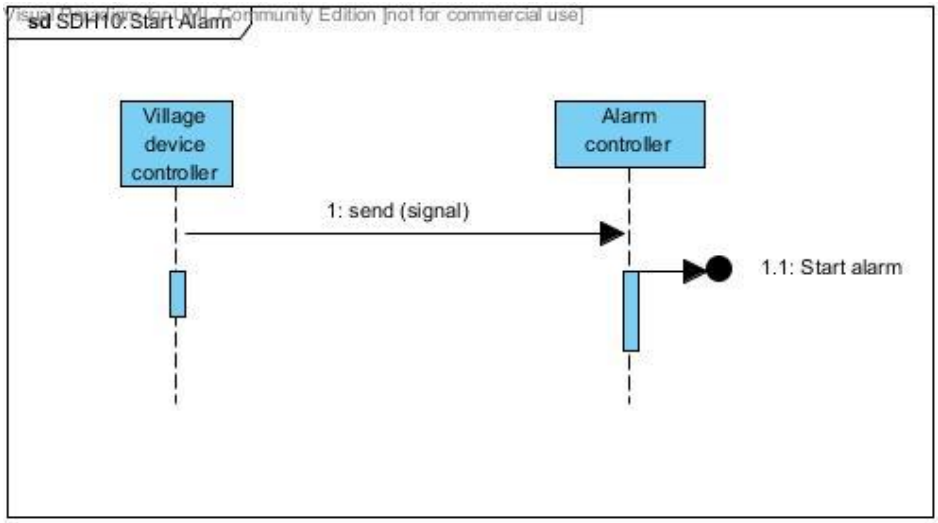
8. Receive data from safe zone detectors



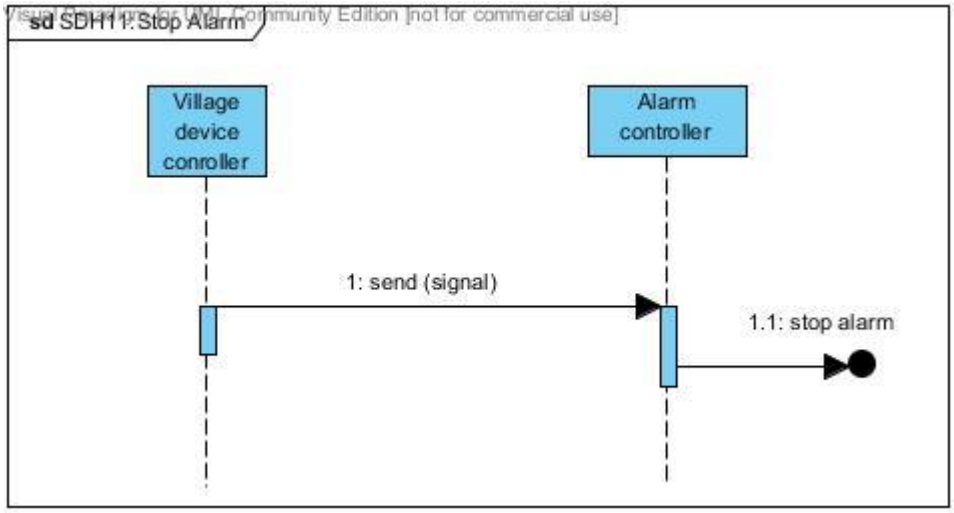
9. Process critical treat msgs



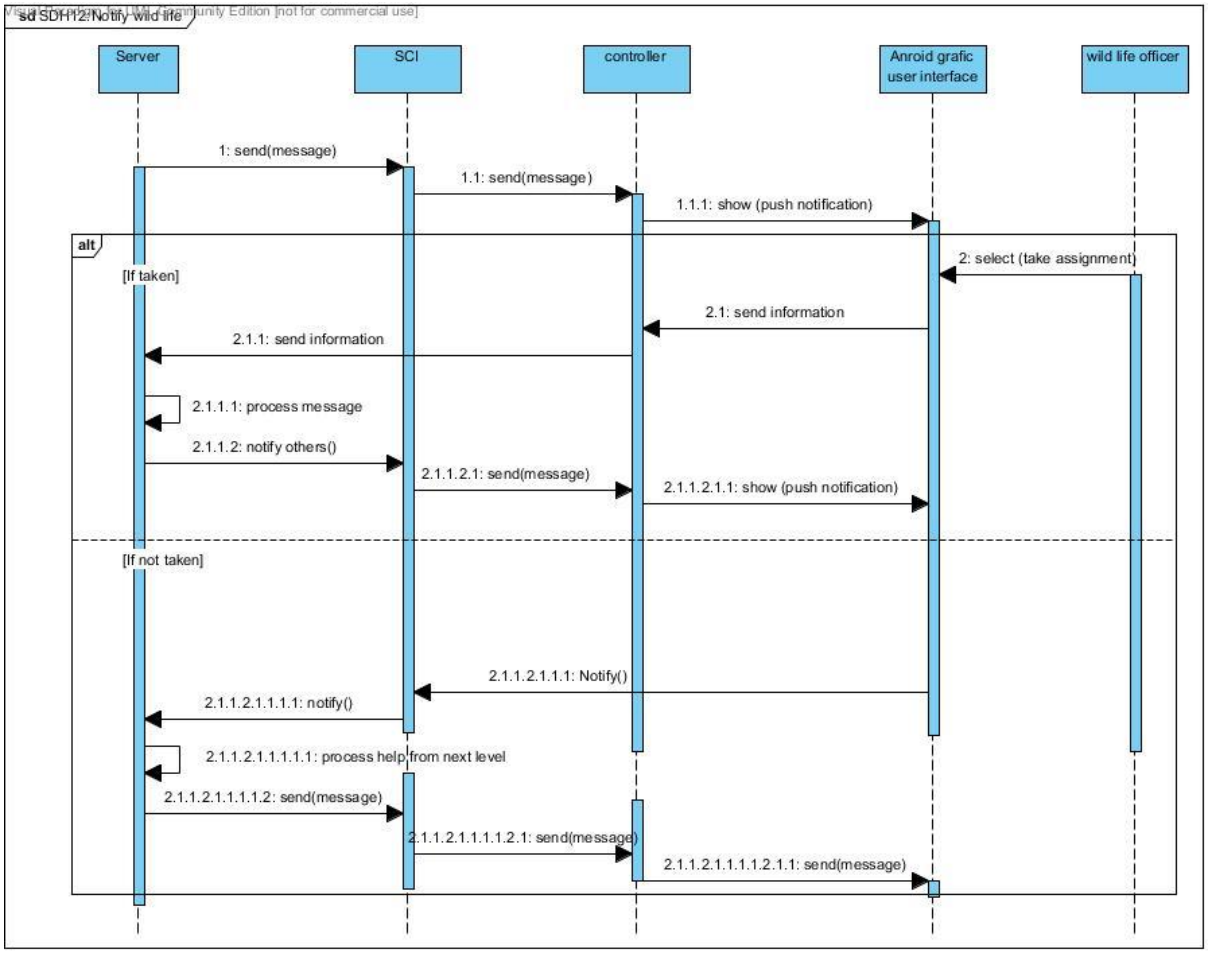
10. Start alarm



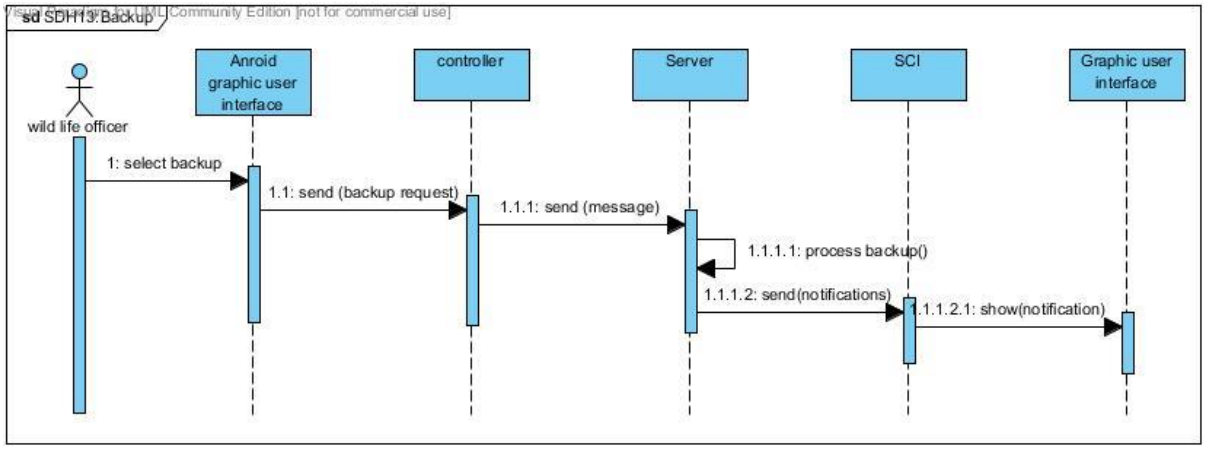
11. Stop alarm



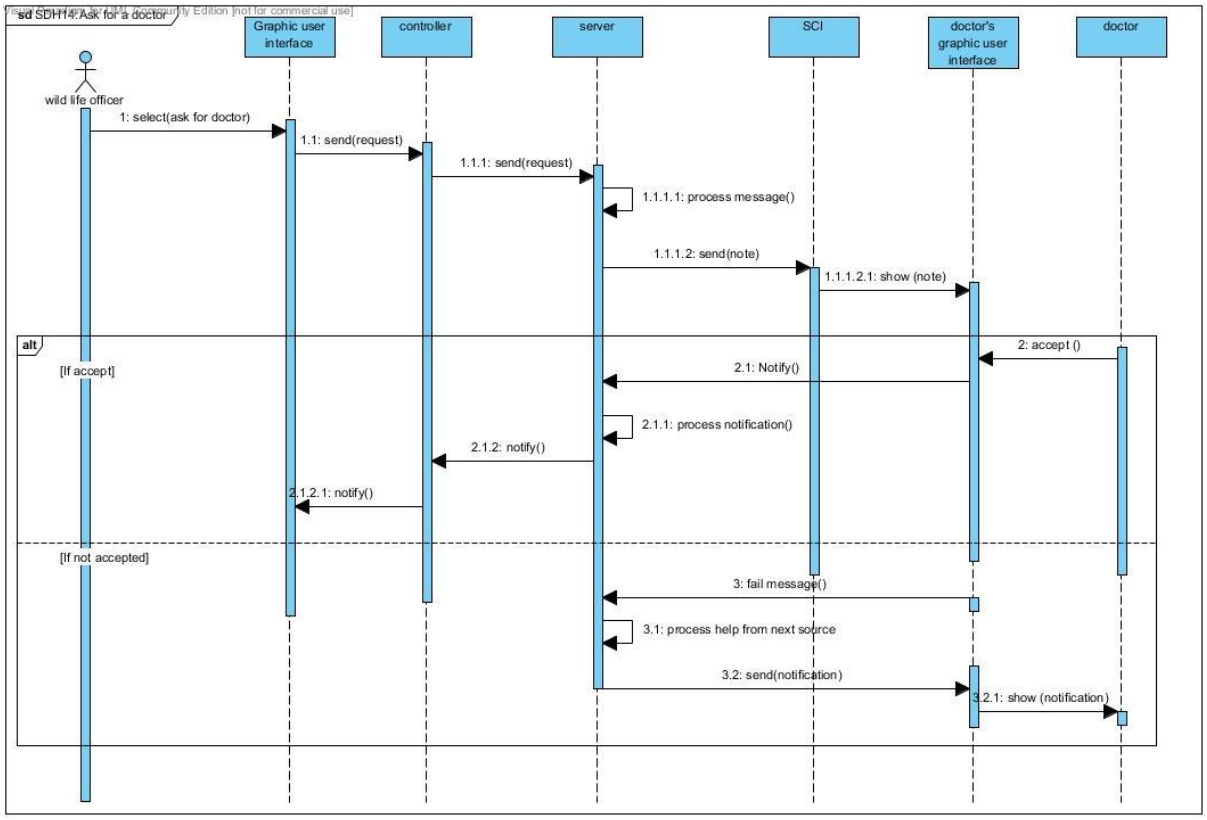
12. Notify Wild Life



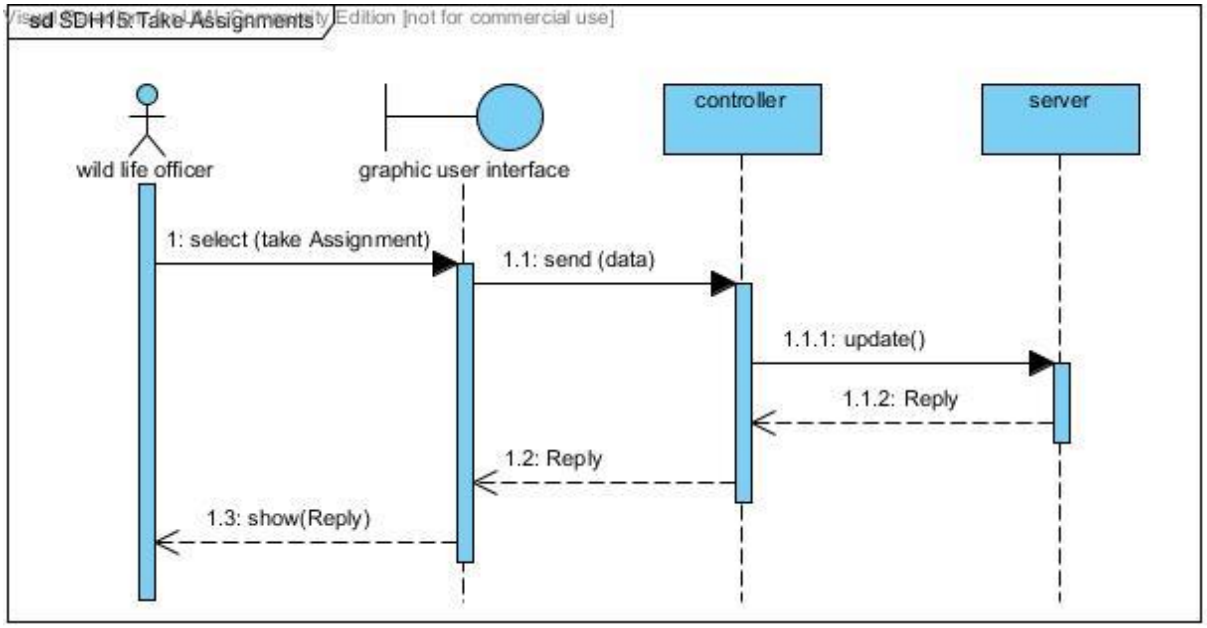
13. Ask backup



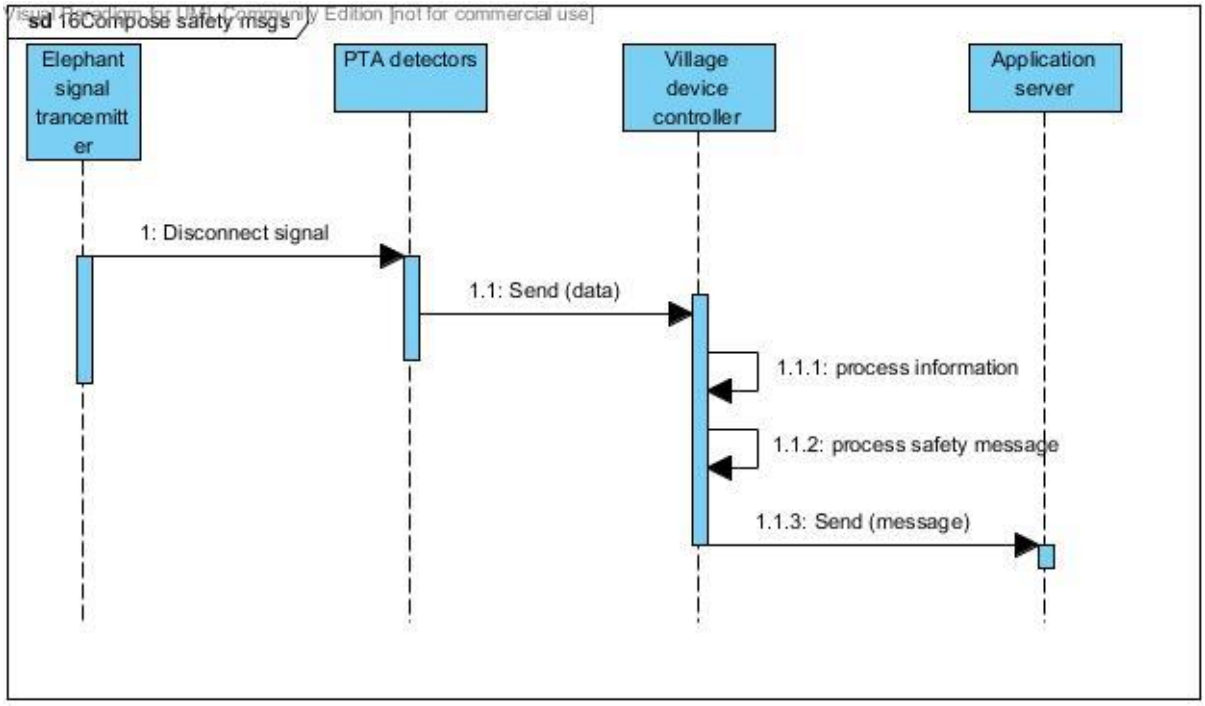
14. Ask for a doctor



15. Take Assignment

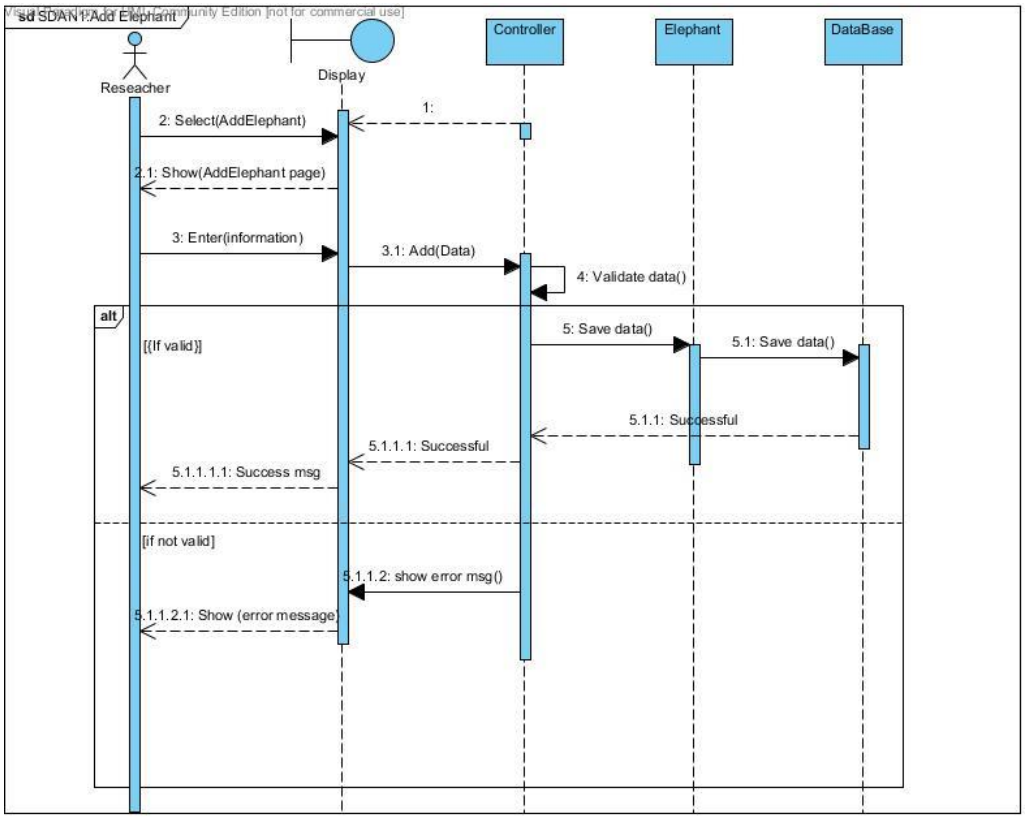


16. Compose Safety msgs

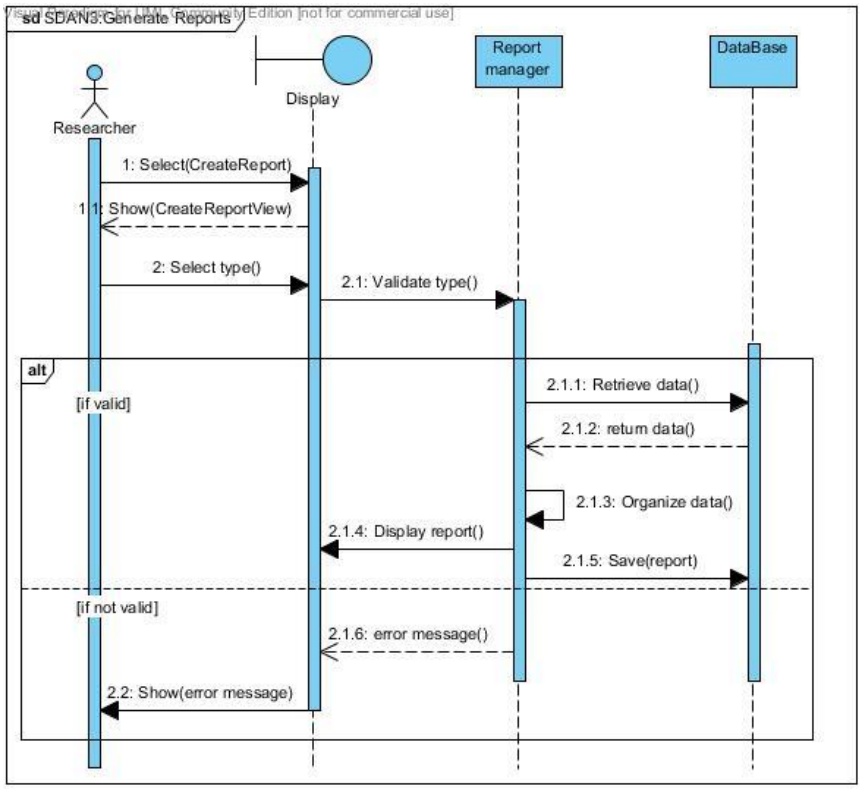


3.3.1. Analyser Sequence Diagrams

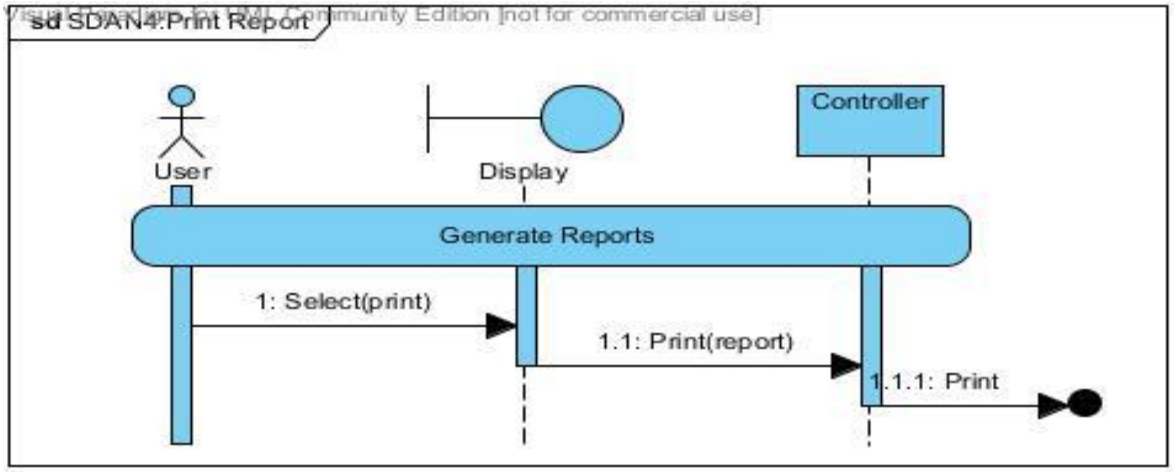
1.Add Elephant



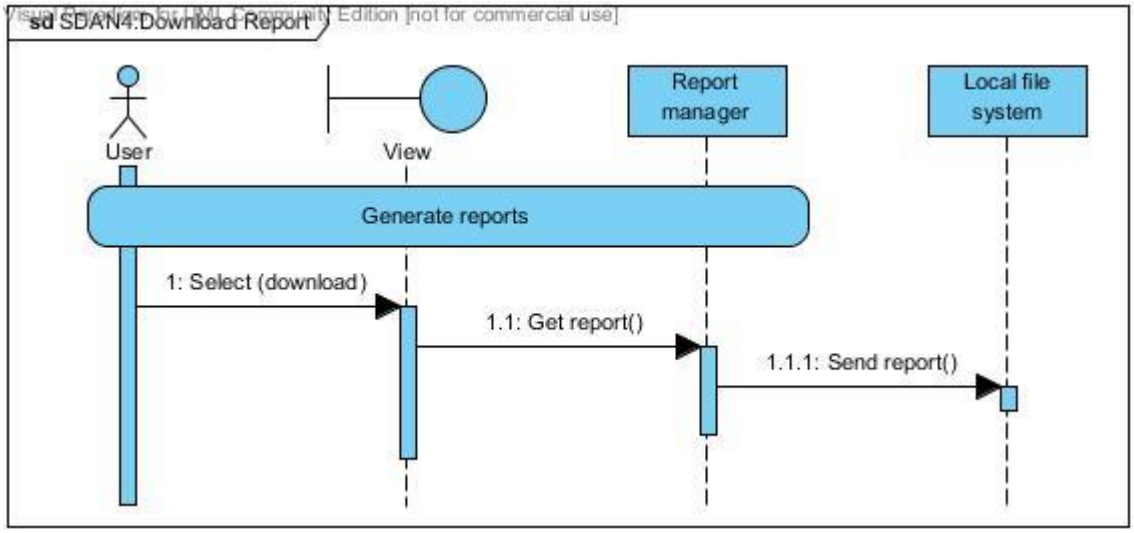
2. Generate Reports



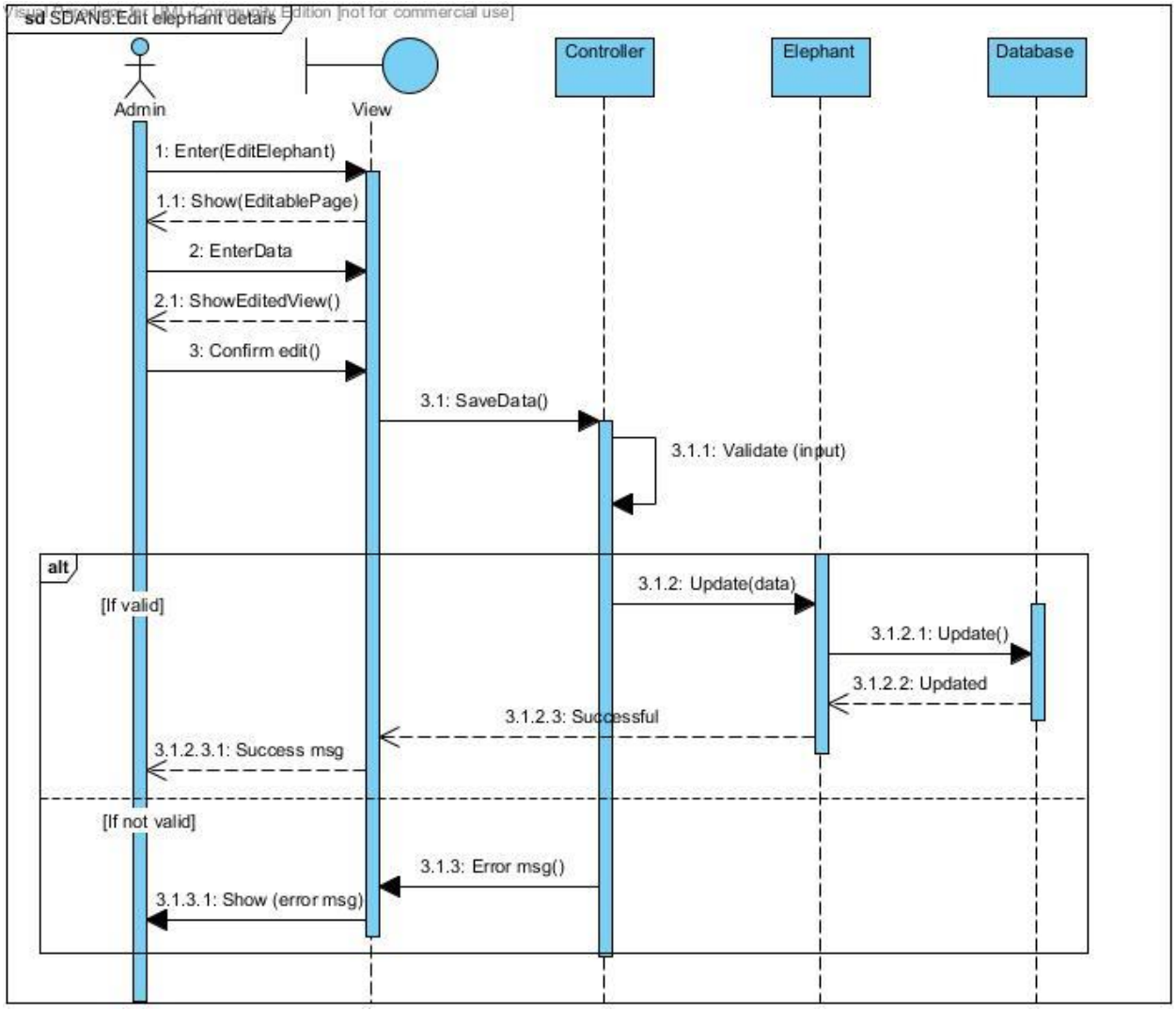
3. Print Reports



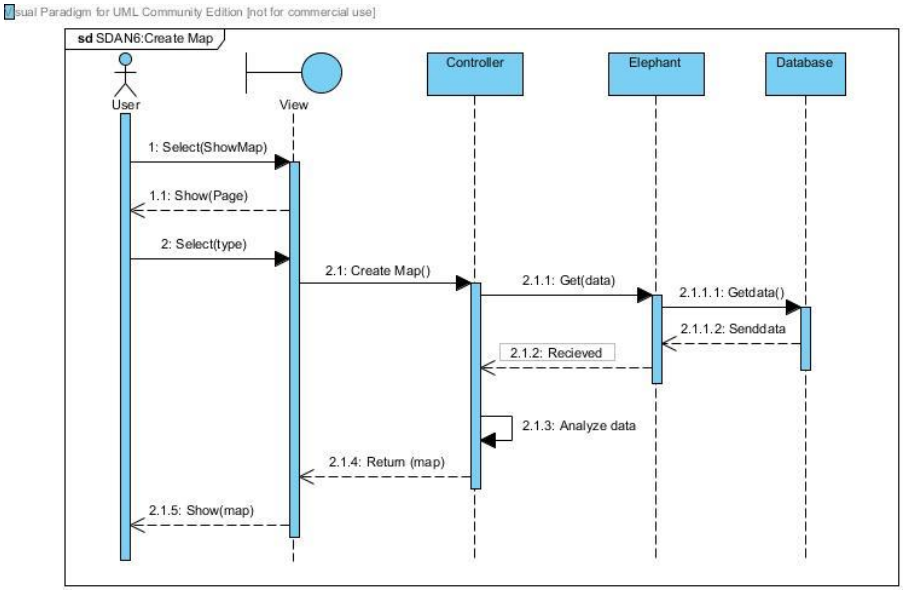
4. DownLoad Reports



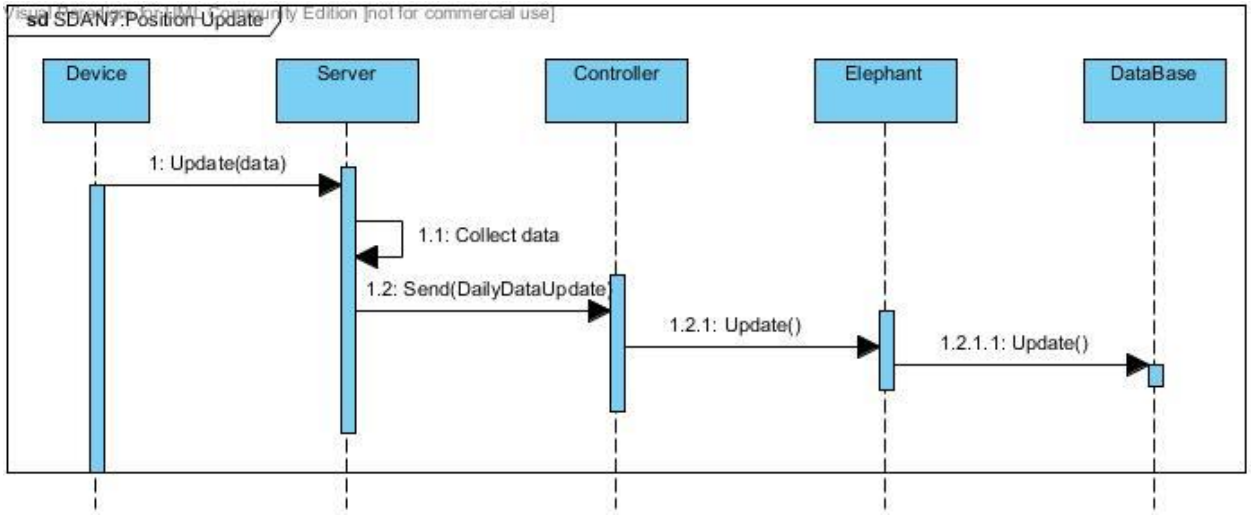
5. Edit elephant Details



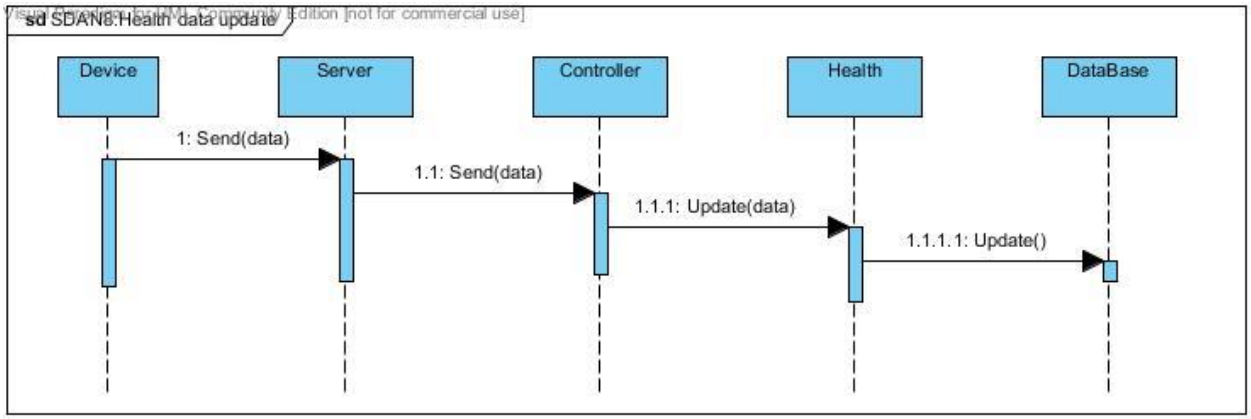
6. Create map



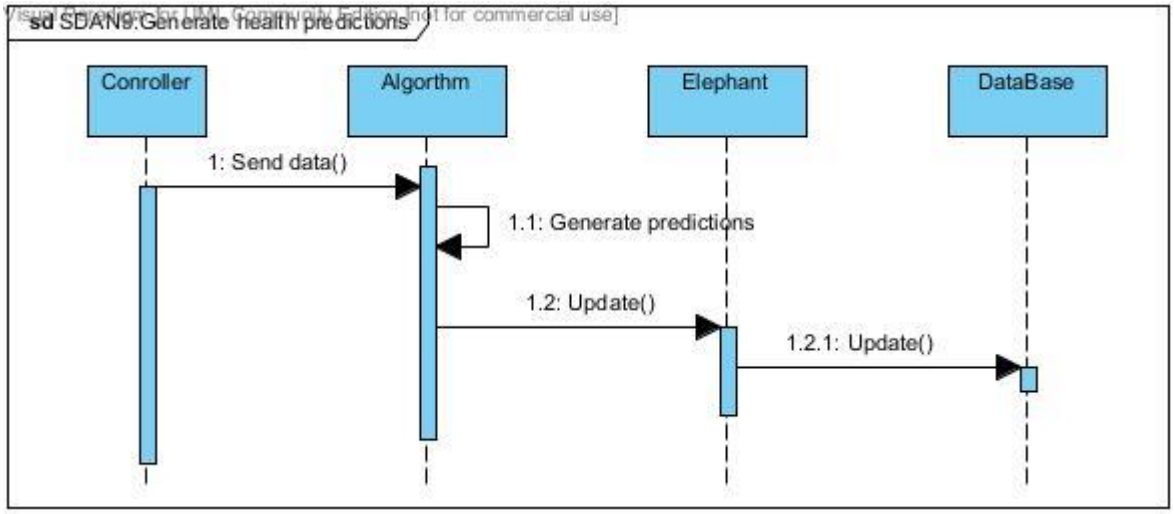
7. Position Update



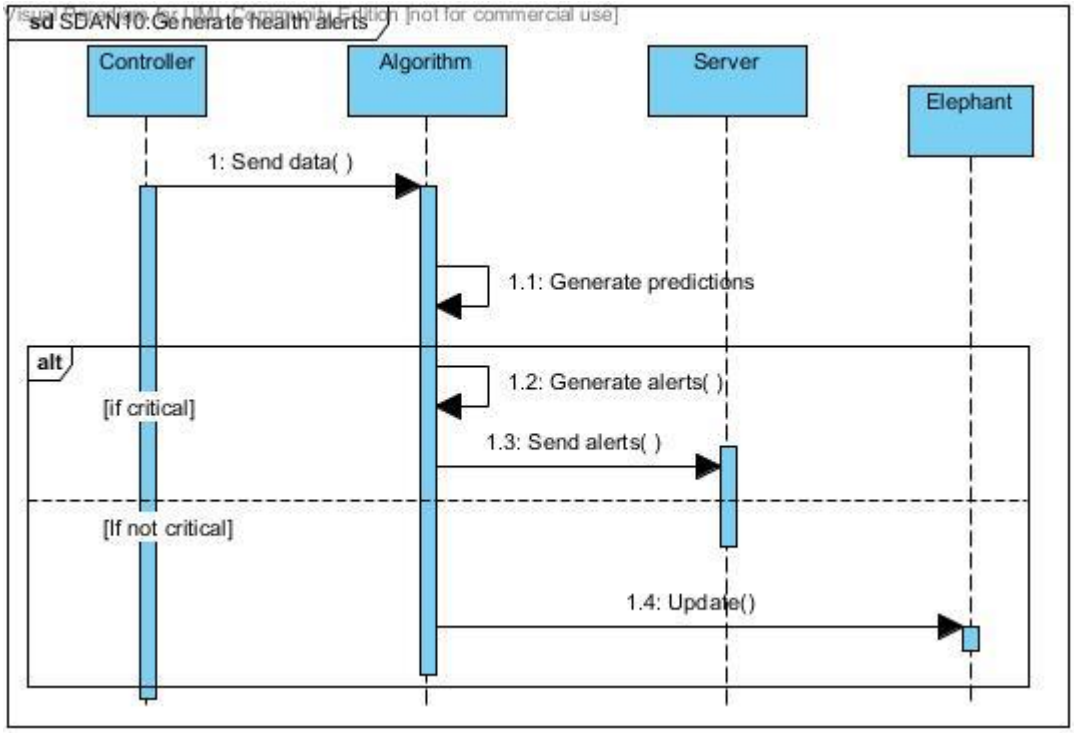
8. Health Data Update



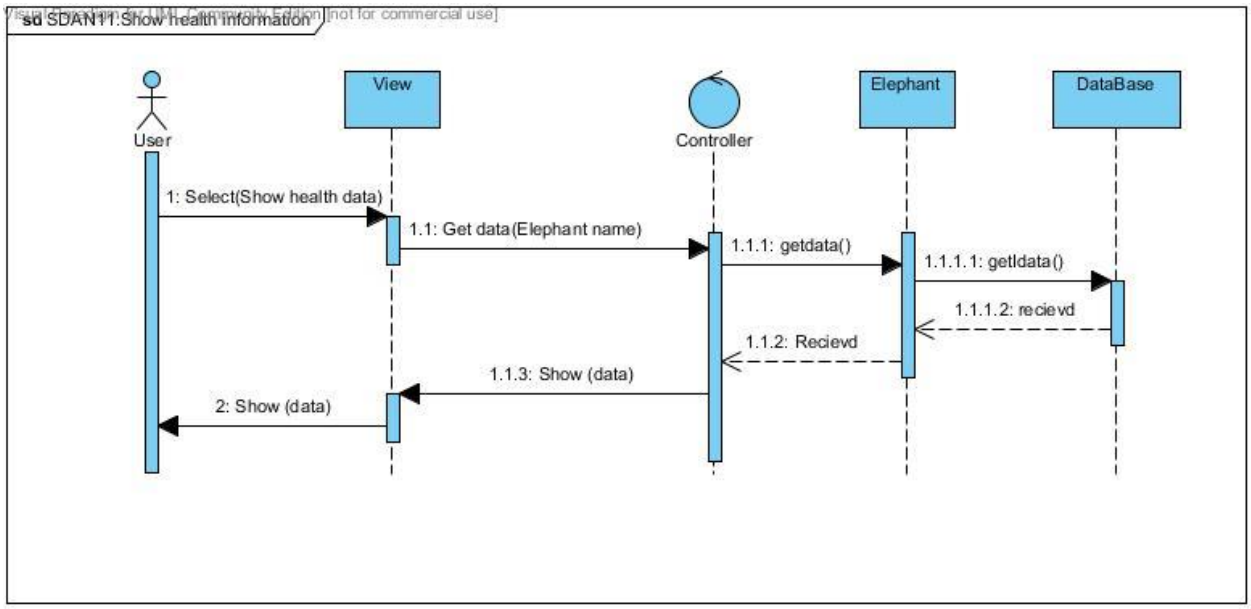
9. Generate health predictions



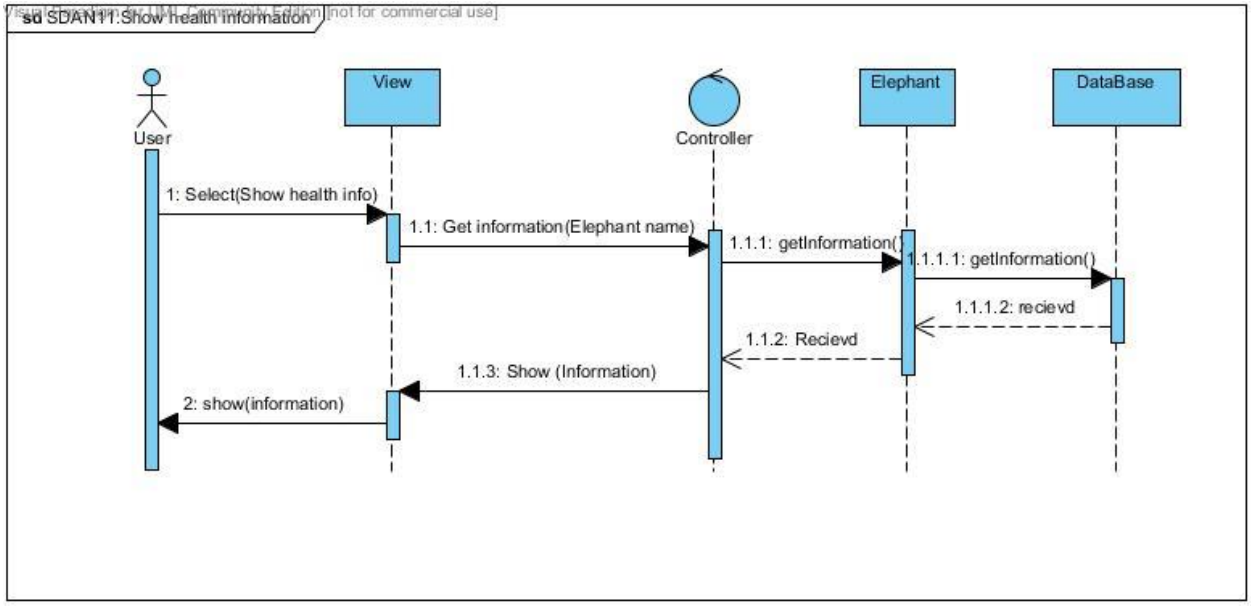
10. Generate health and safety alerts



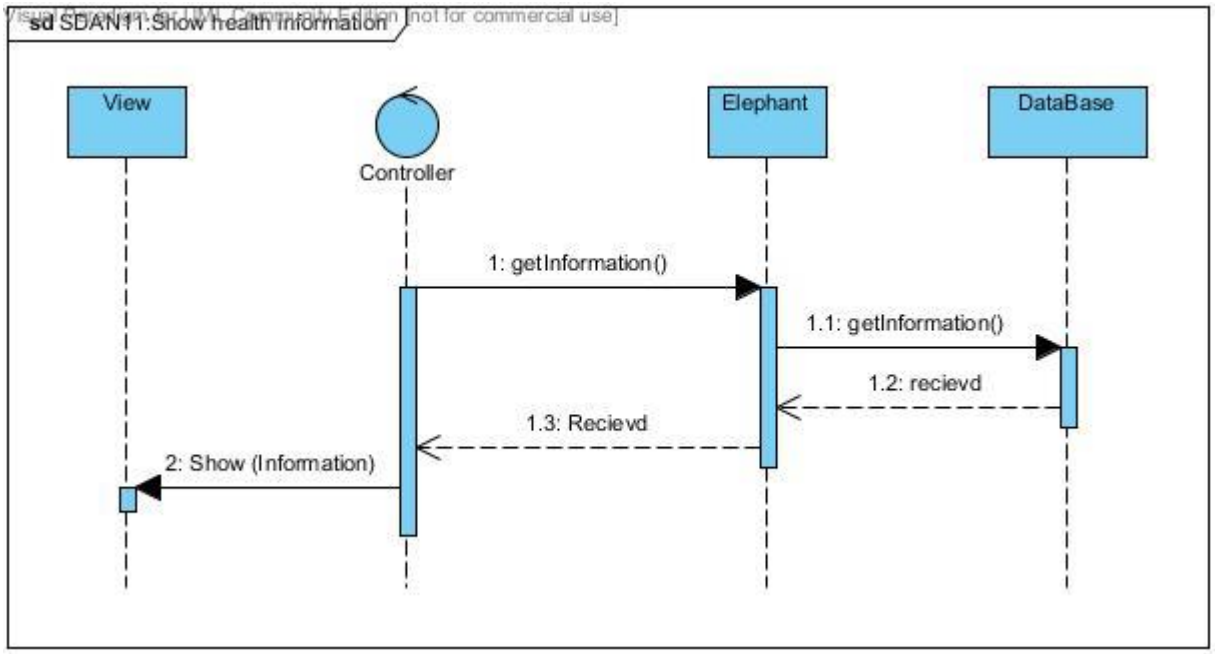
11. Show health data



12. Show health information (Predictions)

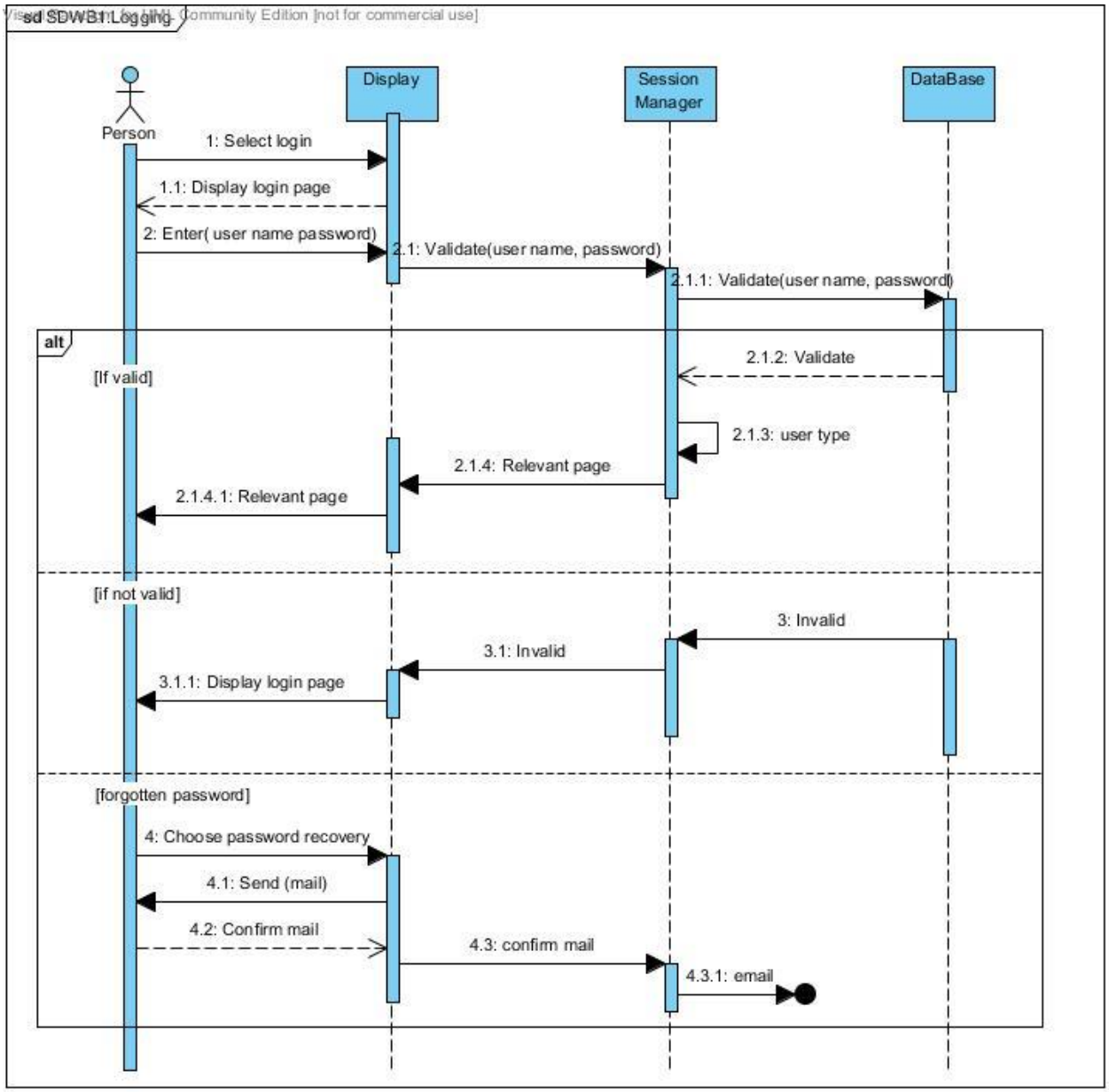


13. Auto show information on screen

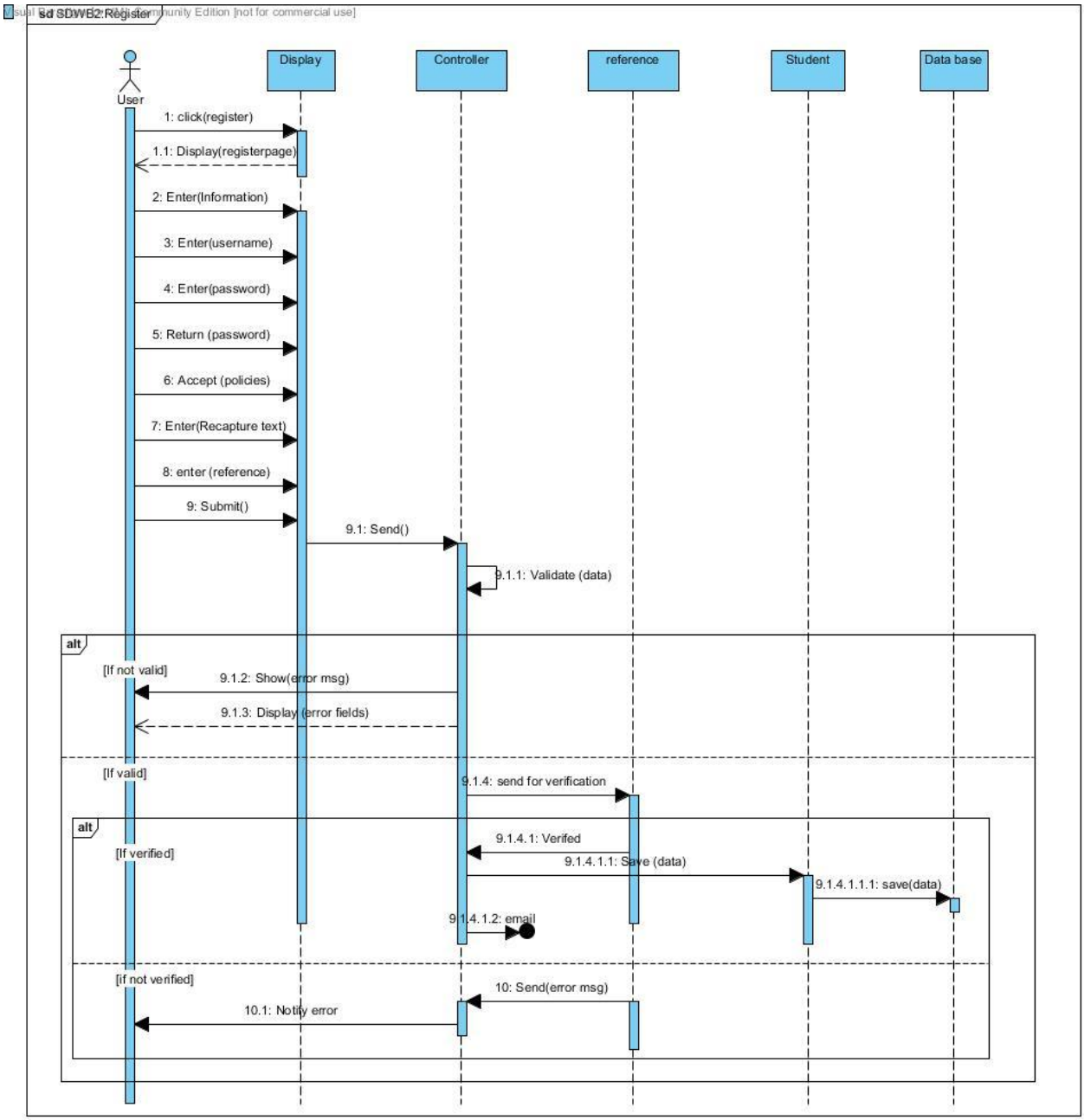


3.3.1. WEB Sequence Diagrams

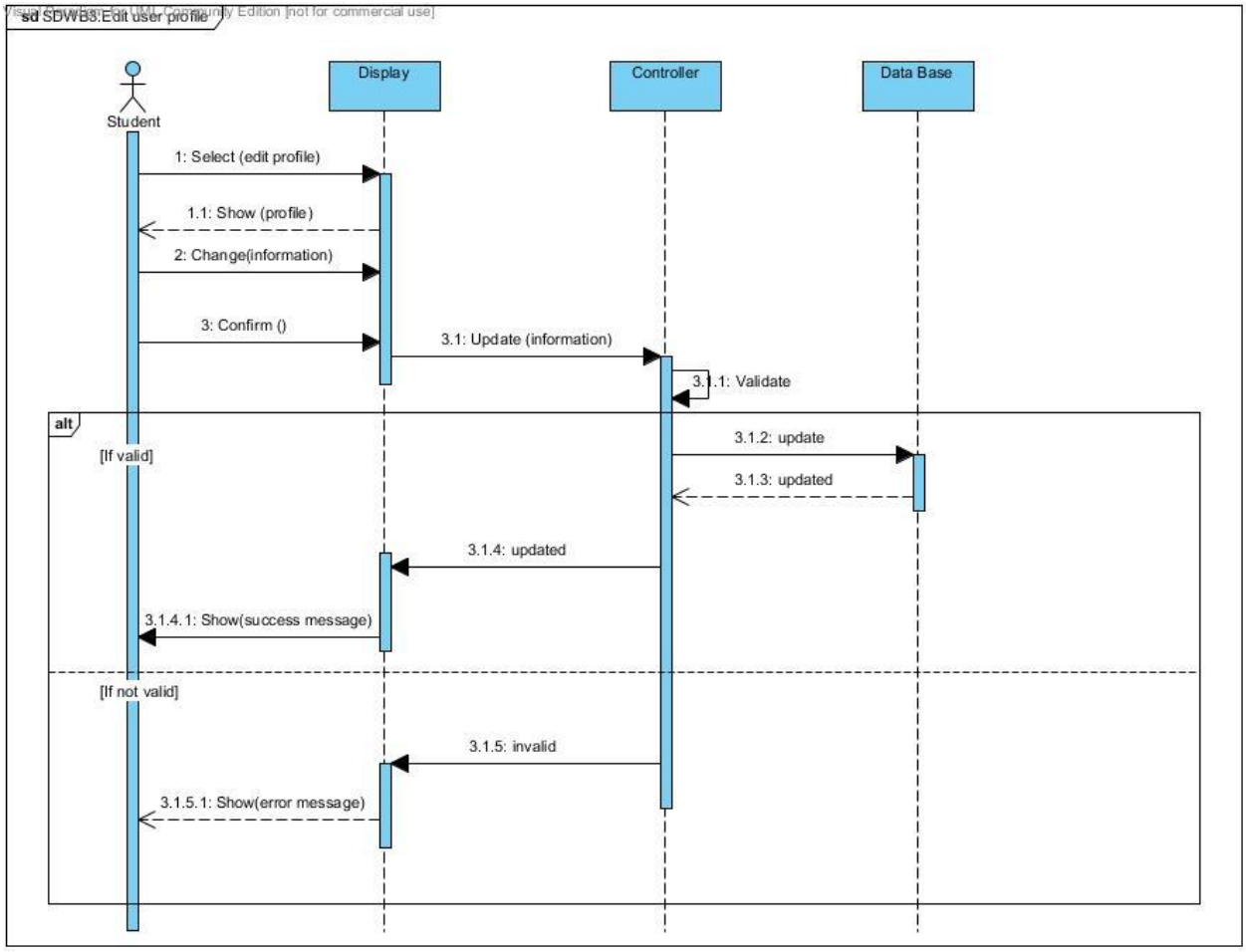
1. Login



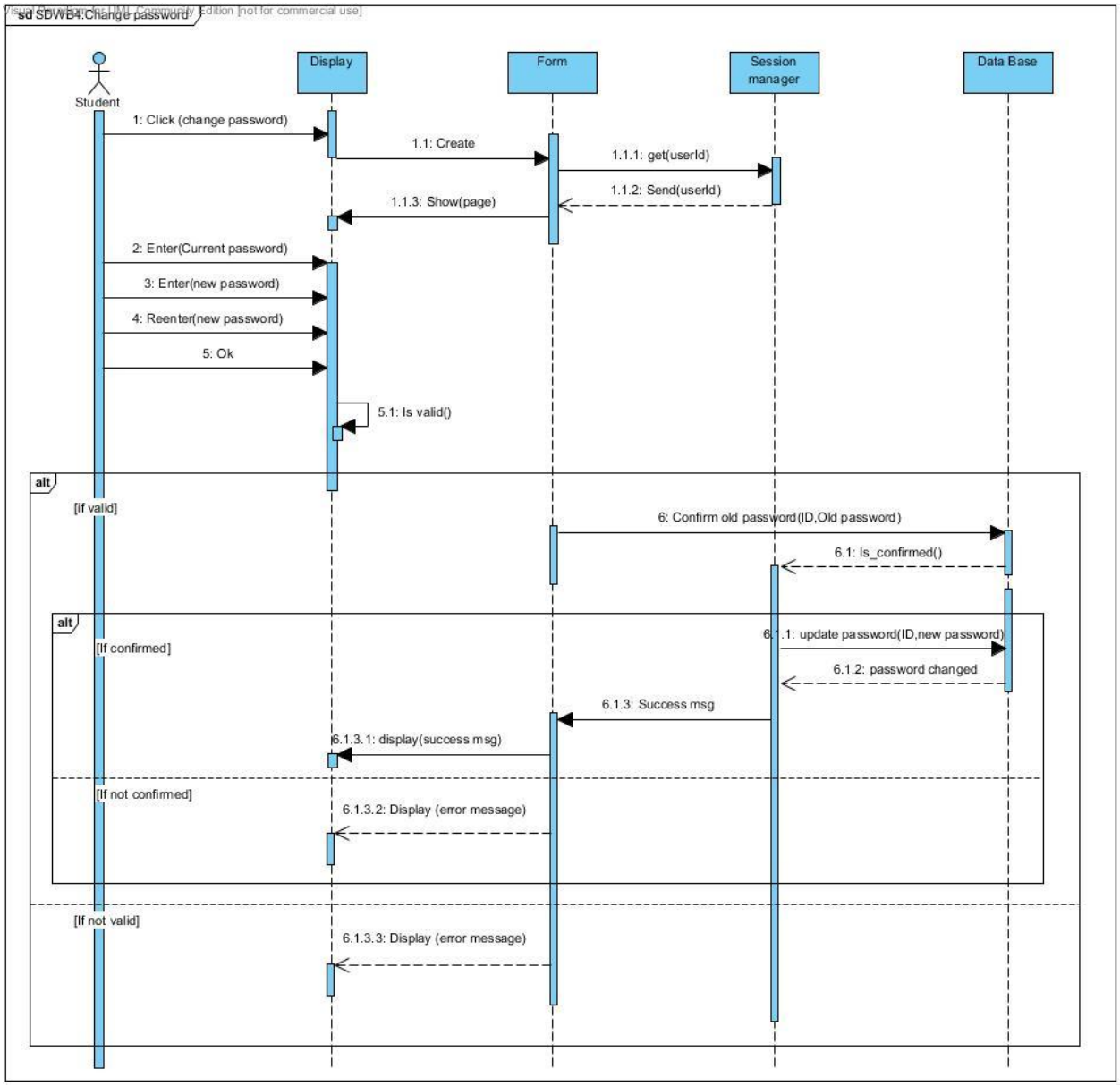
2. Register


sd SDWB2: Register
User
Display
Controller
reference
Student
Data base
1: click(register)
1.1: Display(registerpage)
2: Enter(Information)
3: Enter(username)
4: Enter(password)
5: Return (password)
6: Accept (policies)
7: Enter(Recapture text)
8: enter (reference)
9: Submit()
9.1: Send()
9.1.1: Validate (data)
alt
[If not valid]
9.1.2: Show(error msg)
9.1.3: Display (error fields)
[If valid]
9.1.4: send for verification
alt
[If verified]
9.1.4.1: Verifed
9.1.4.1.1: Save (data)
9.1.4.1.1.1: save(data)
9.1.4.1.2: email
[if not verified]
10: Send(error msg)
10.1: Notify error

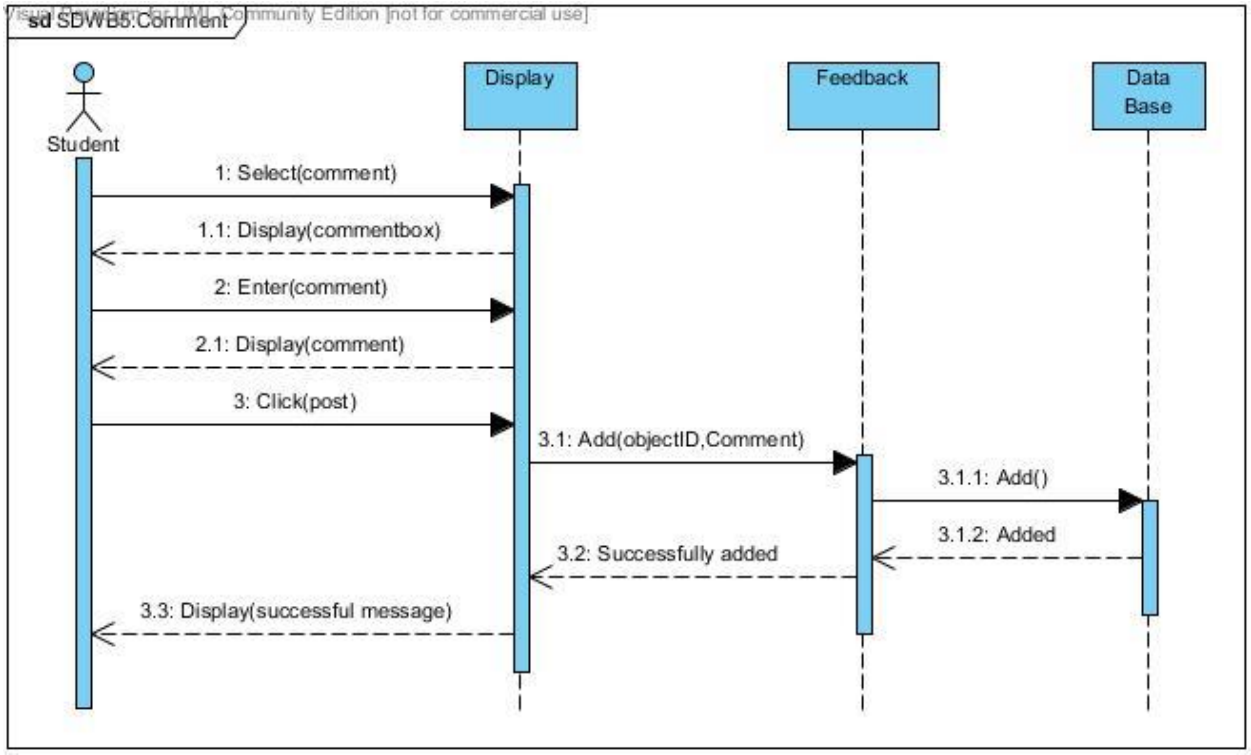
3. Edit user Profile



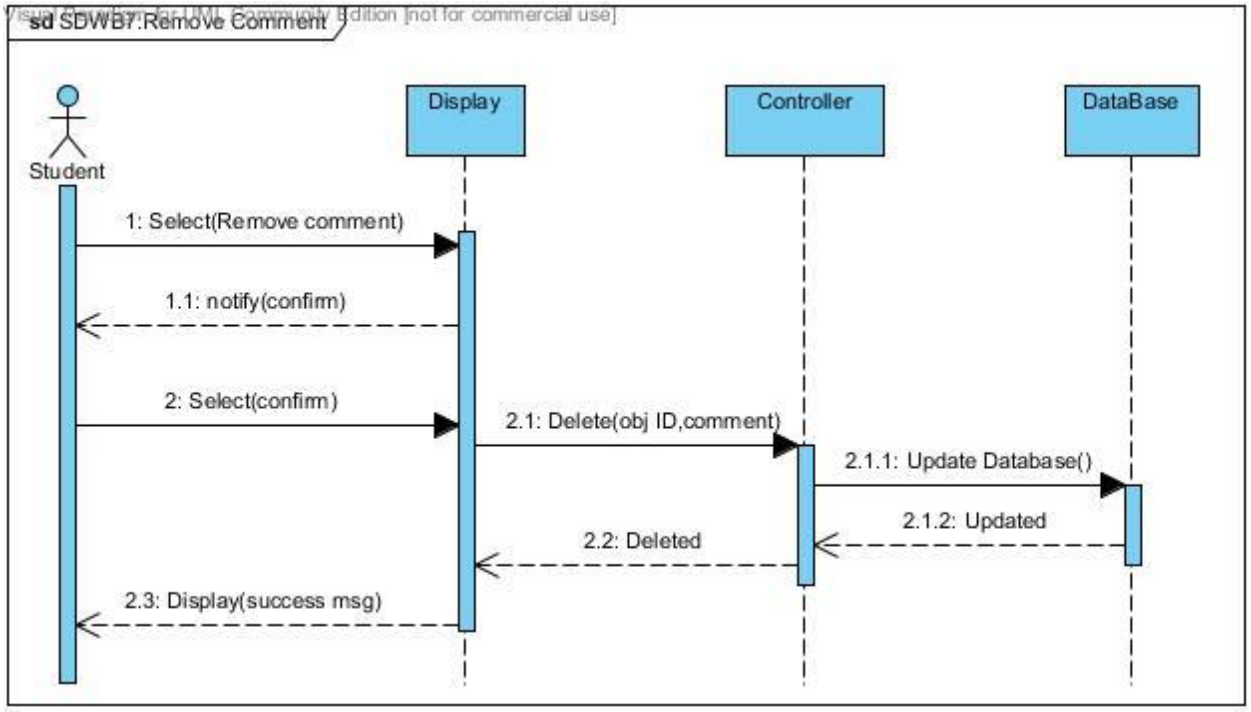
4. Change password



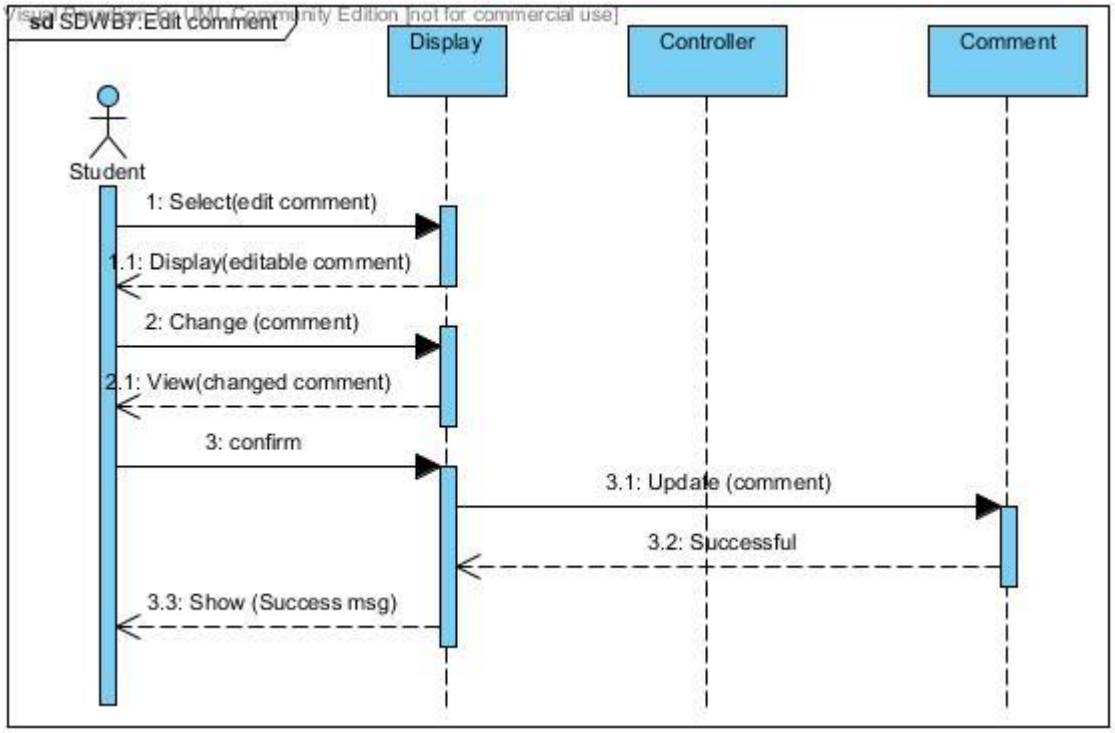
5. Comment



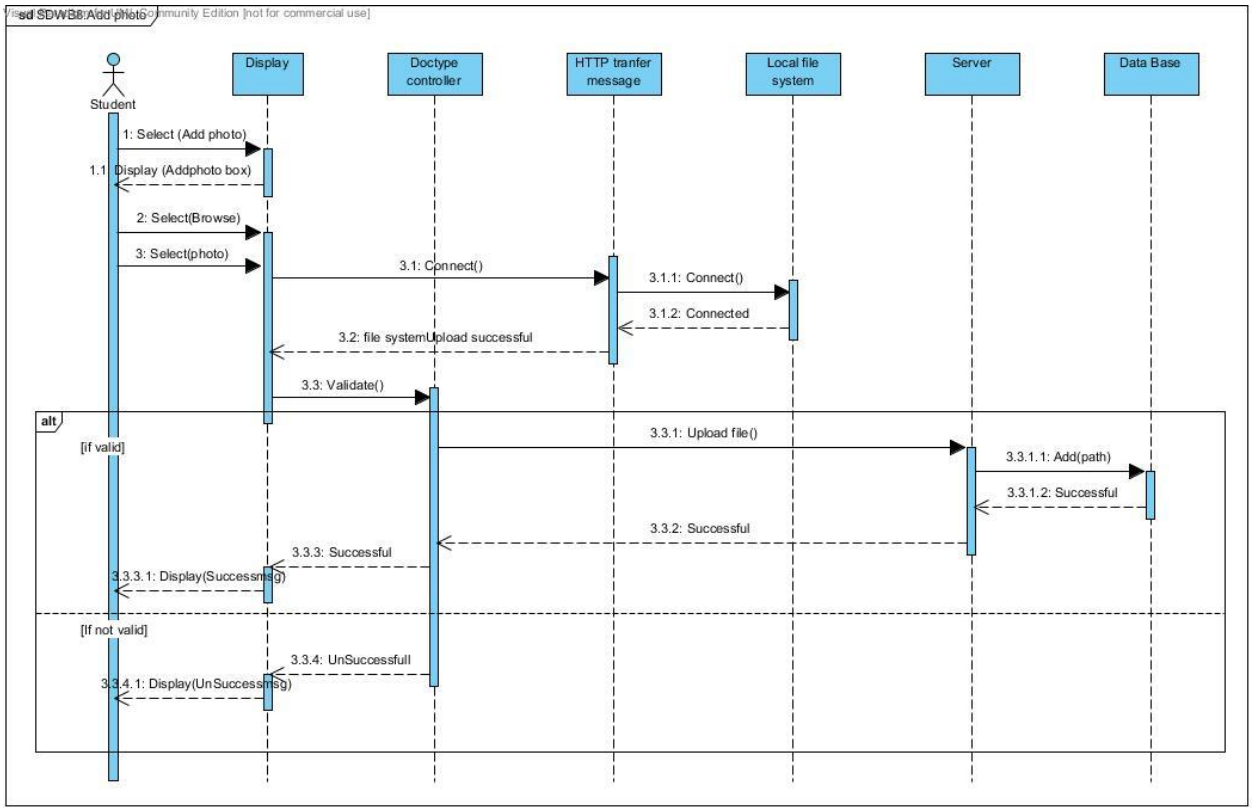
6. Remove comment



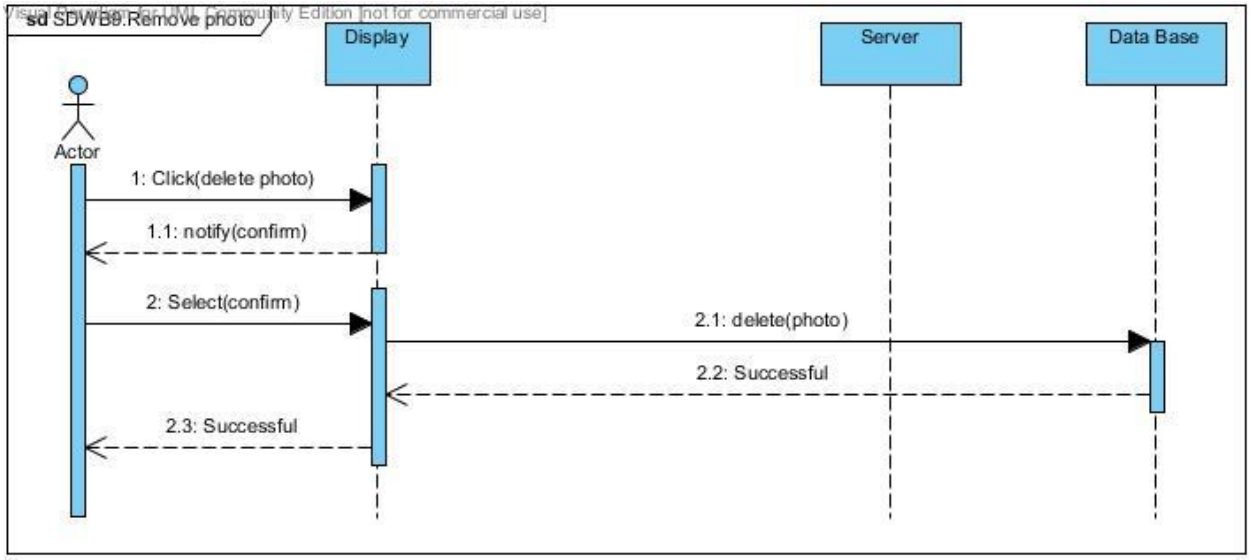
7. Edit comment



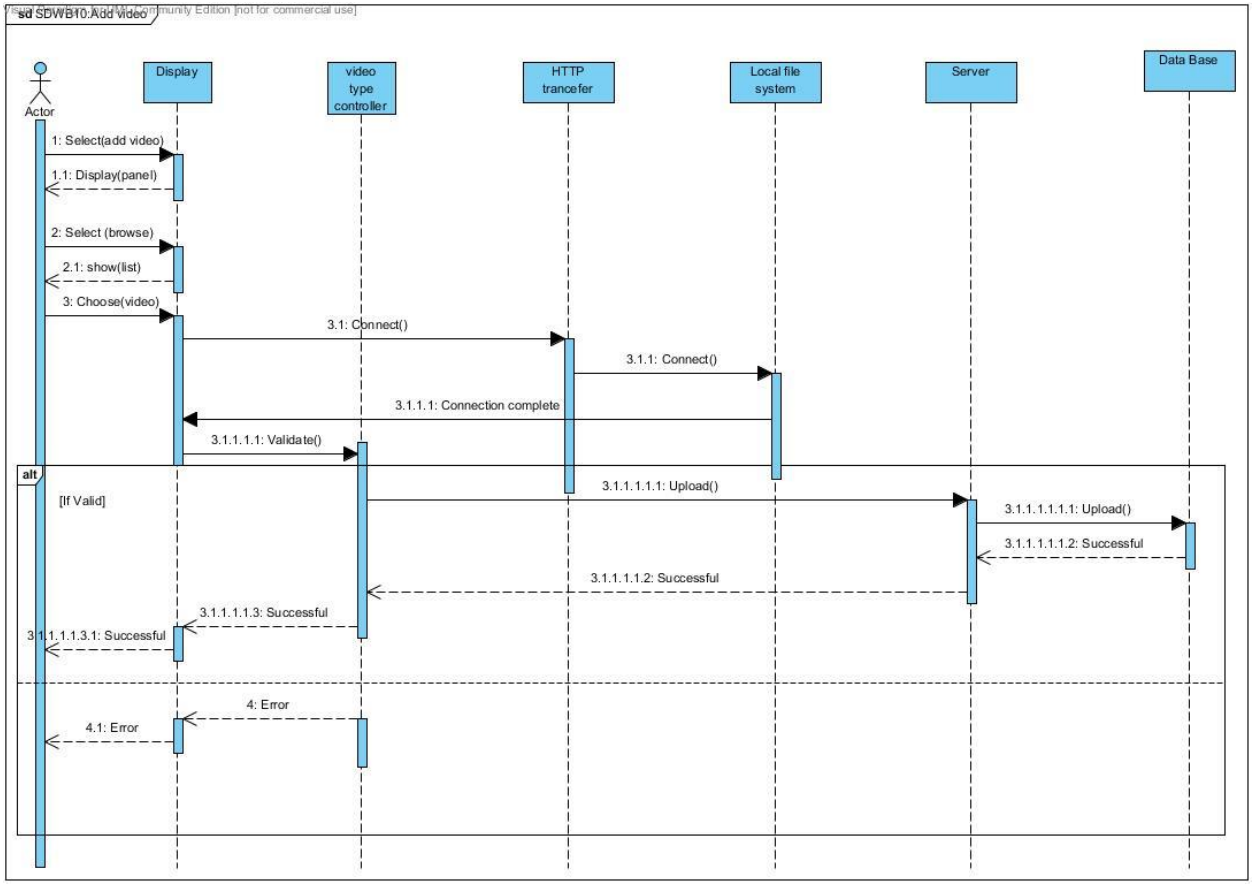
8. Add photo



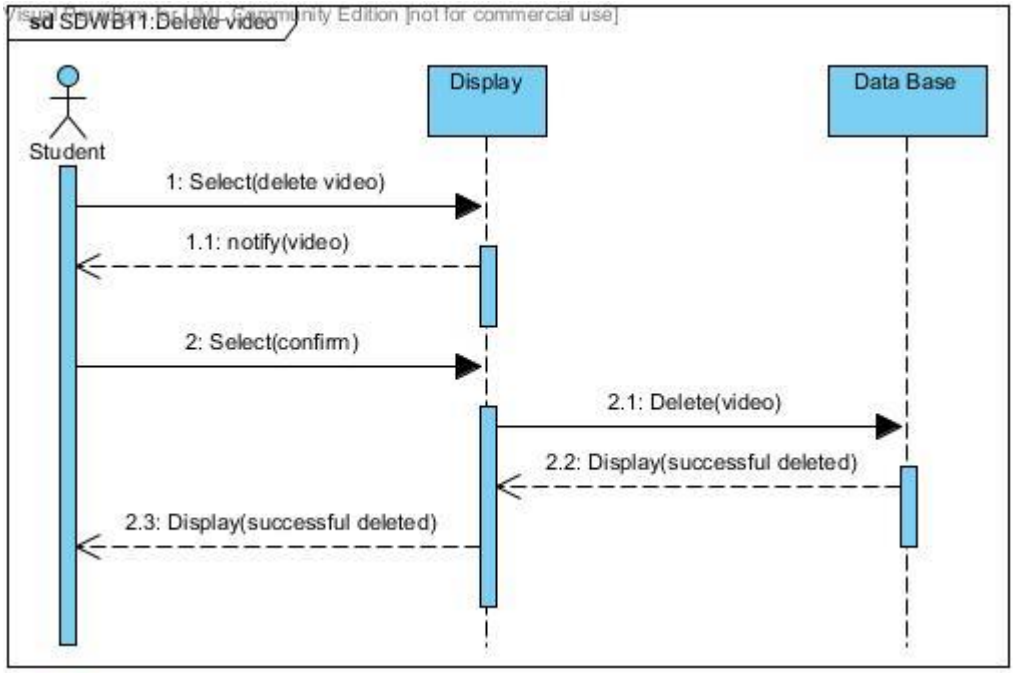
9. Remove photo



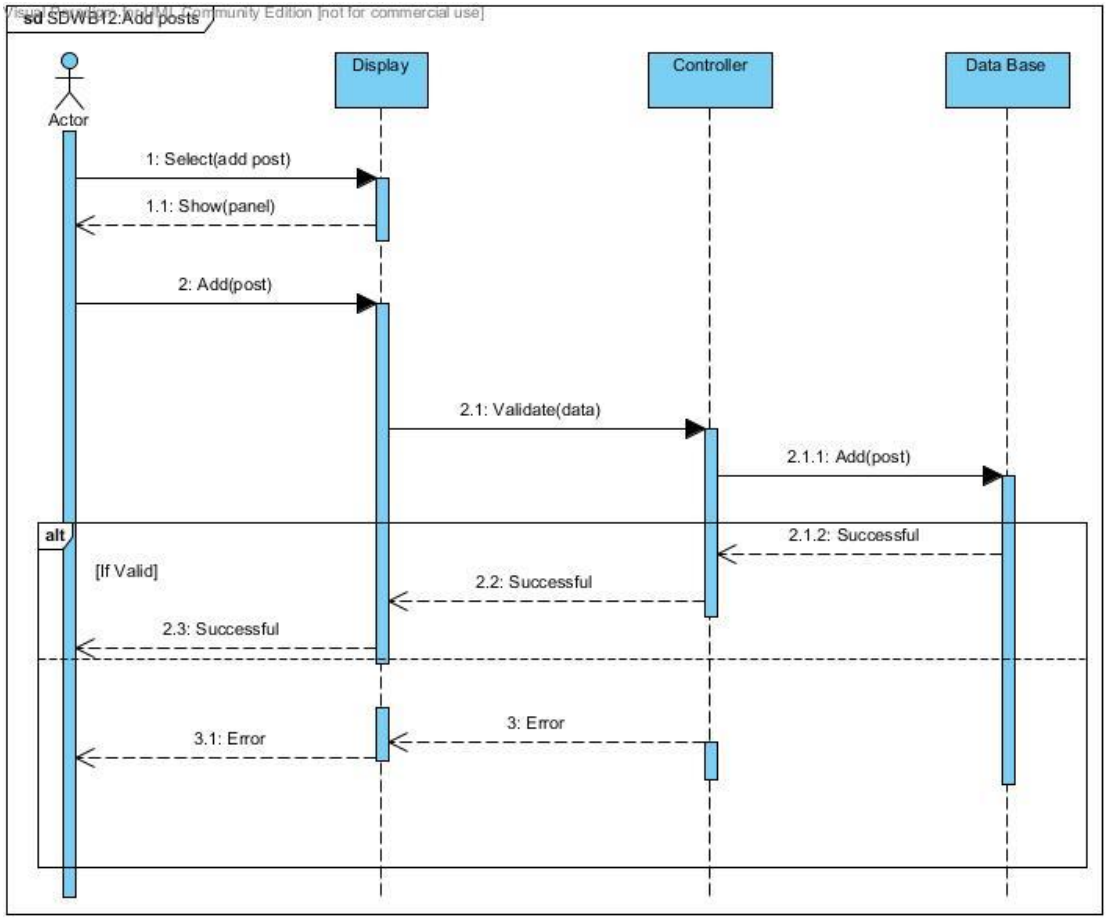
10. Add video



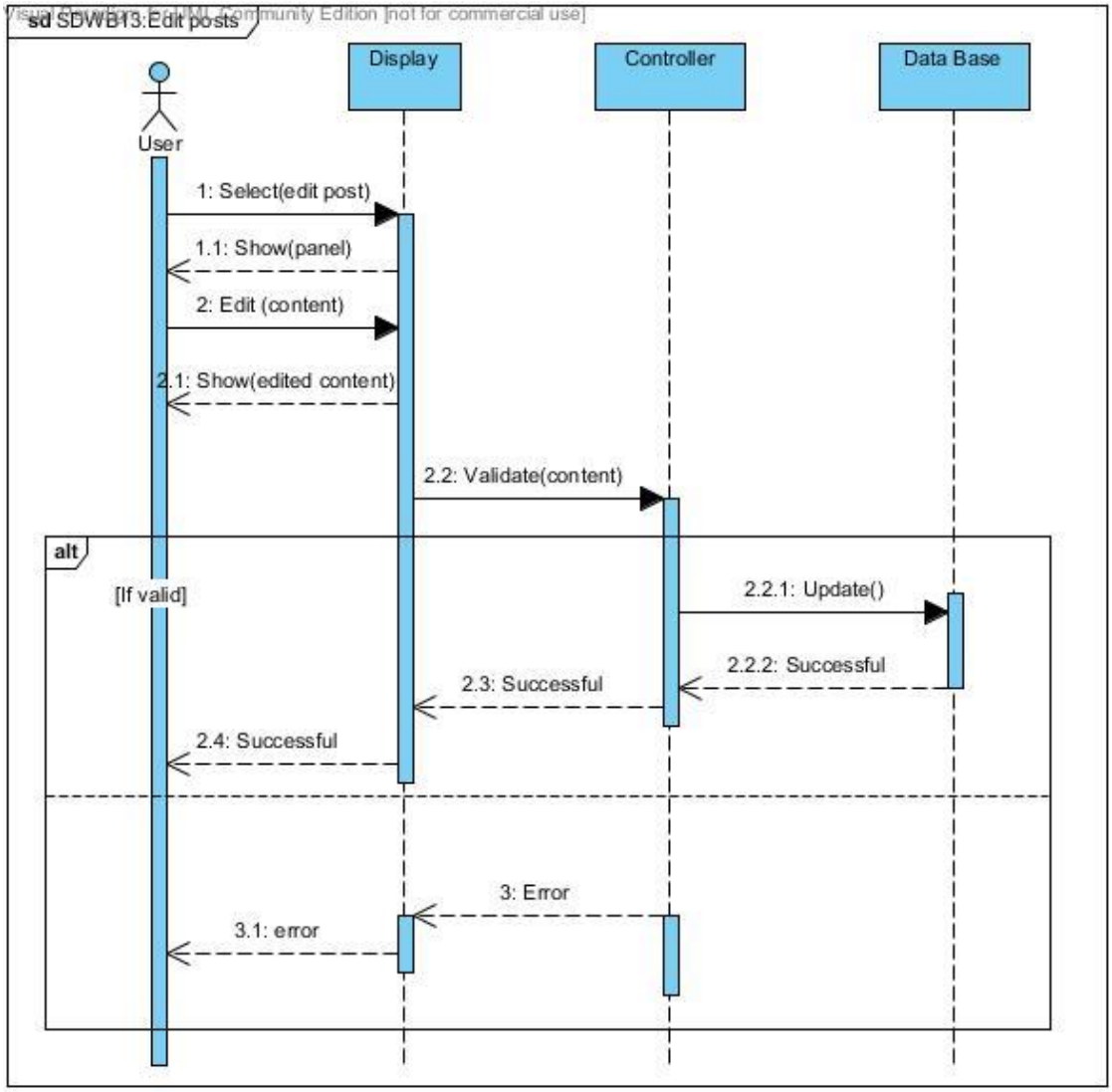
11. Delete video



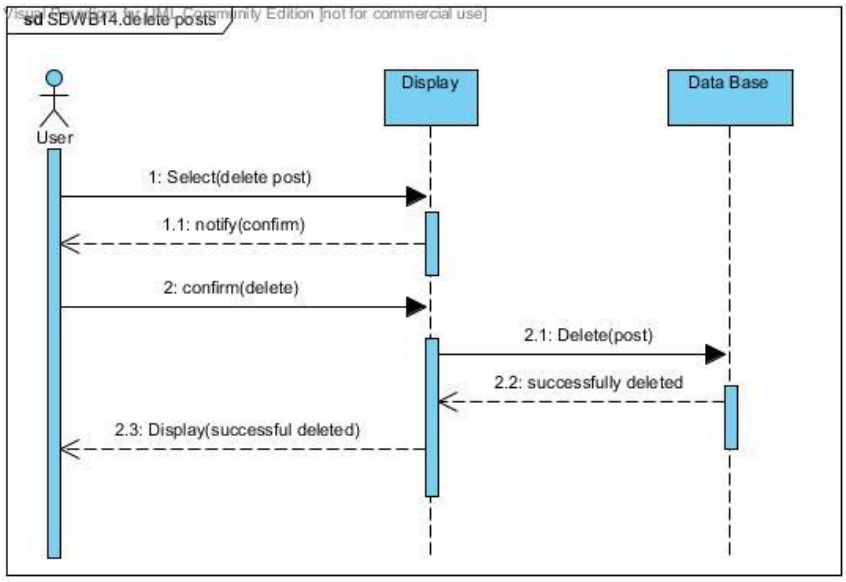
12. Add posts



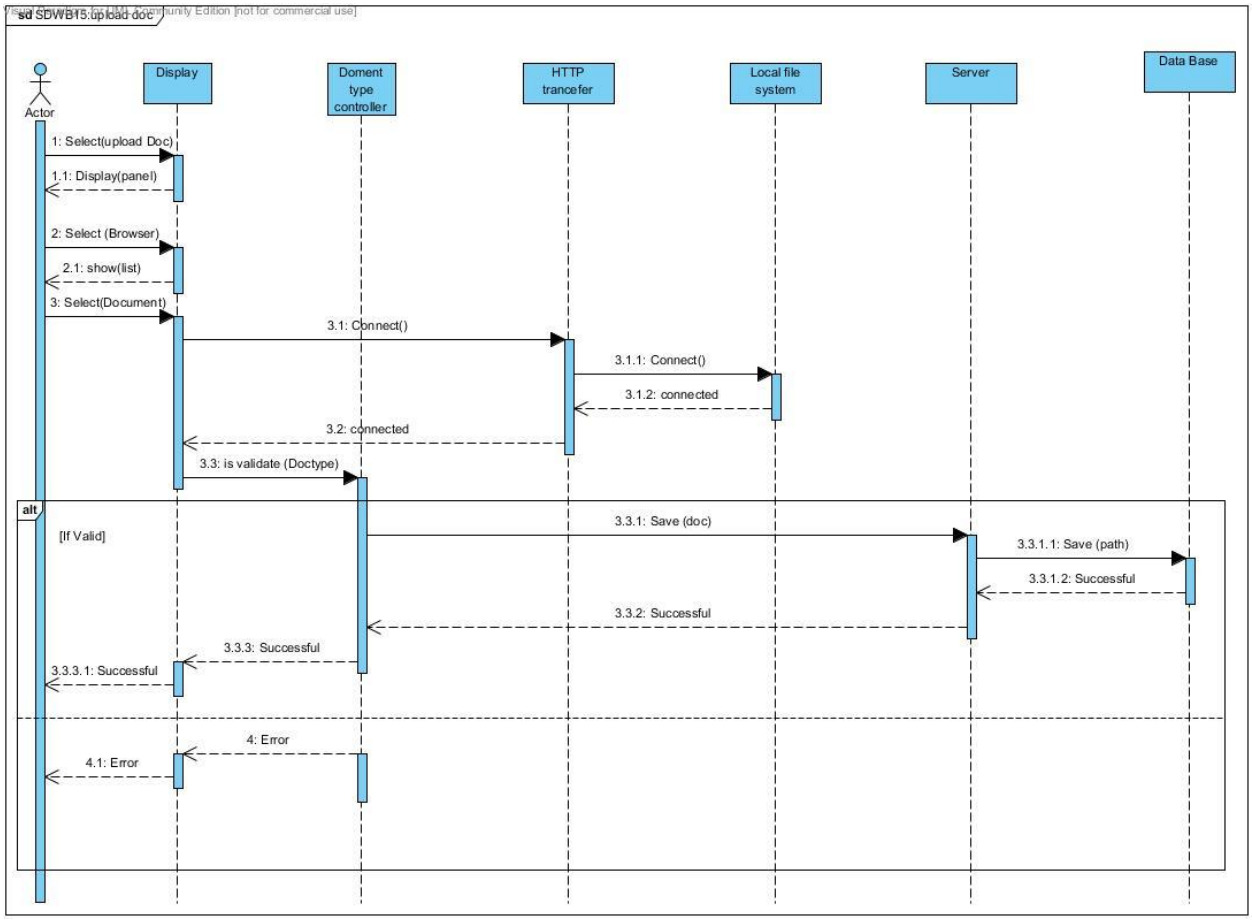
13. Edit posts



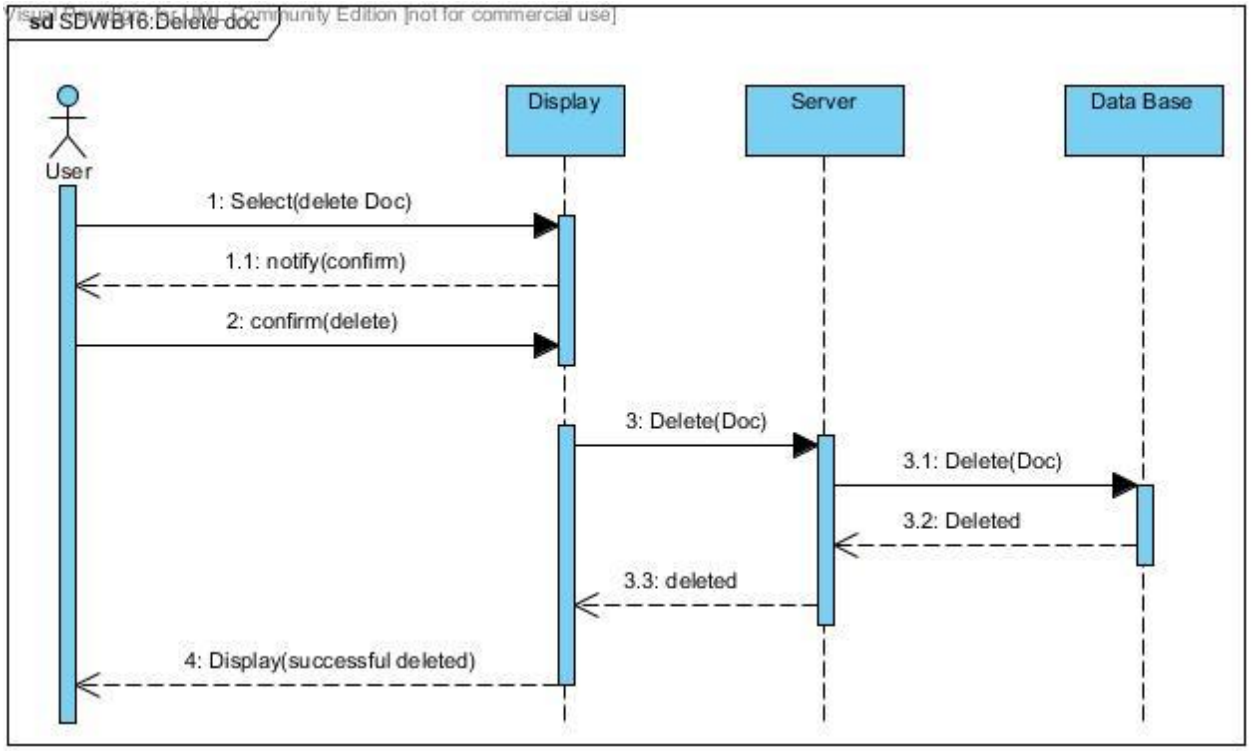
14. Delete posts



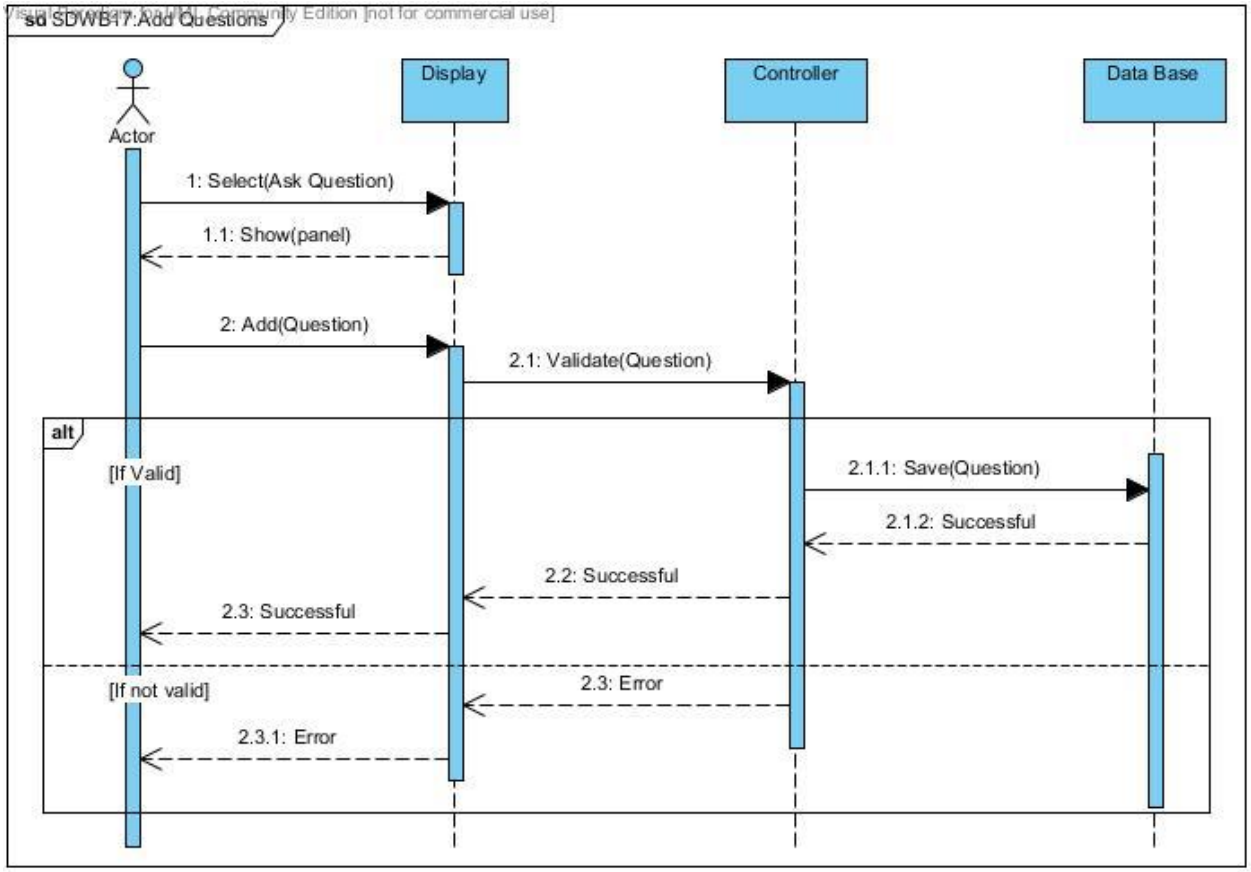
15. Upload docs



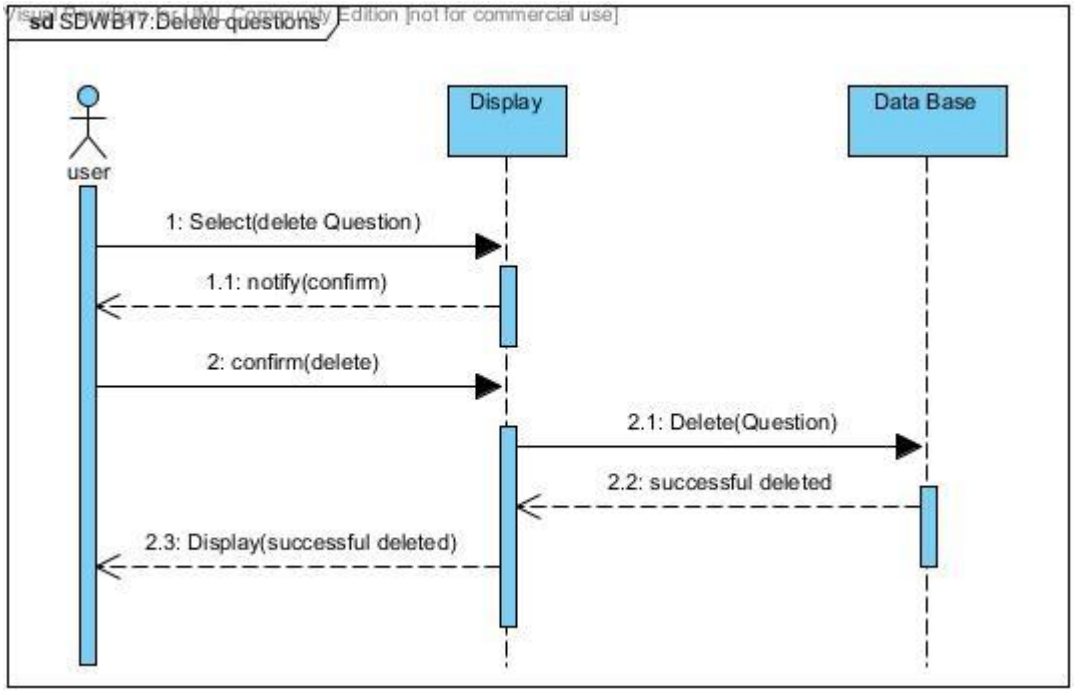
16. Delete docs



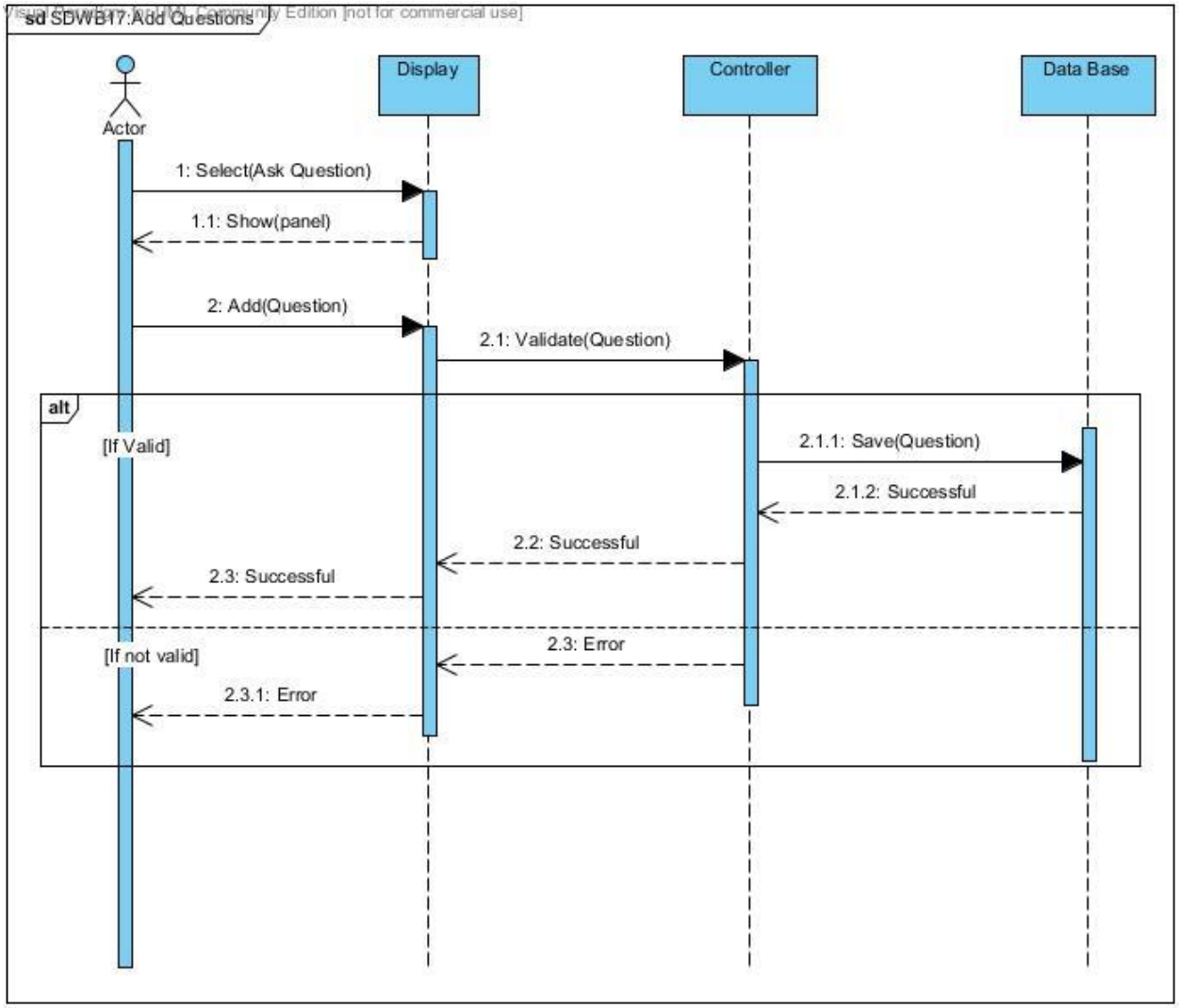
17. Ask Questions



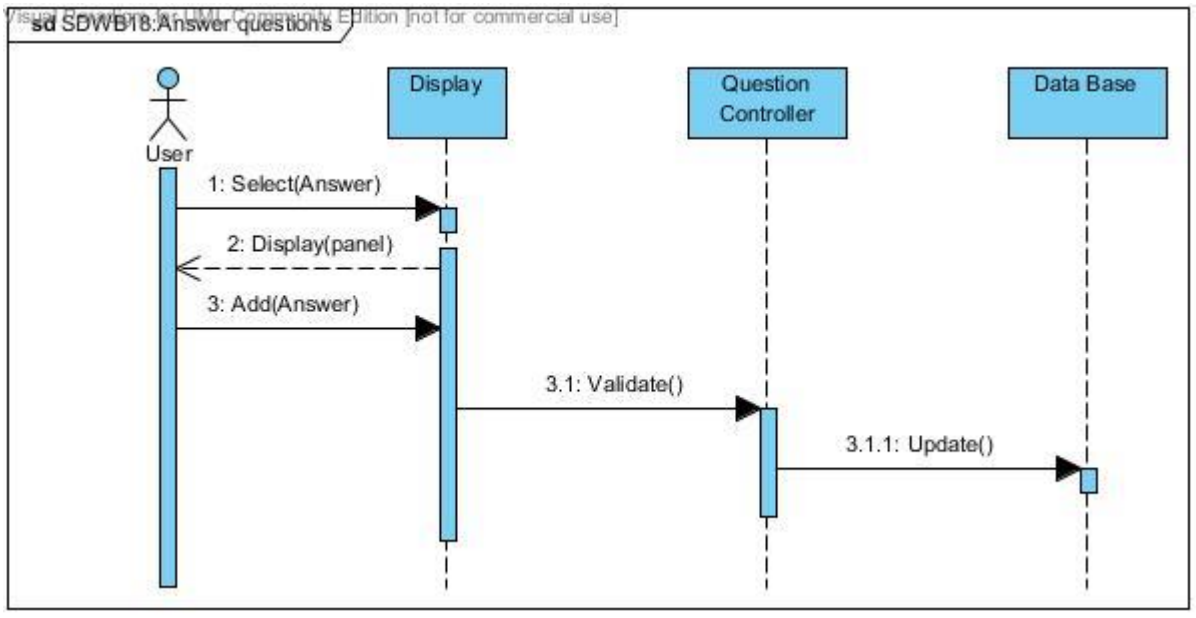
18. Delete Questions



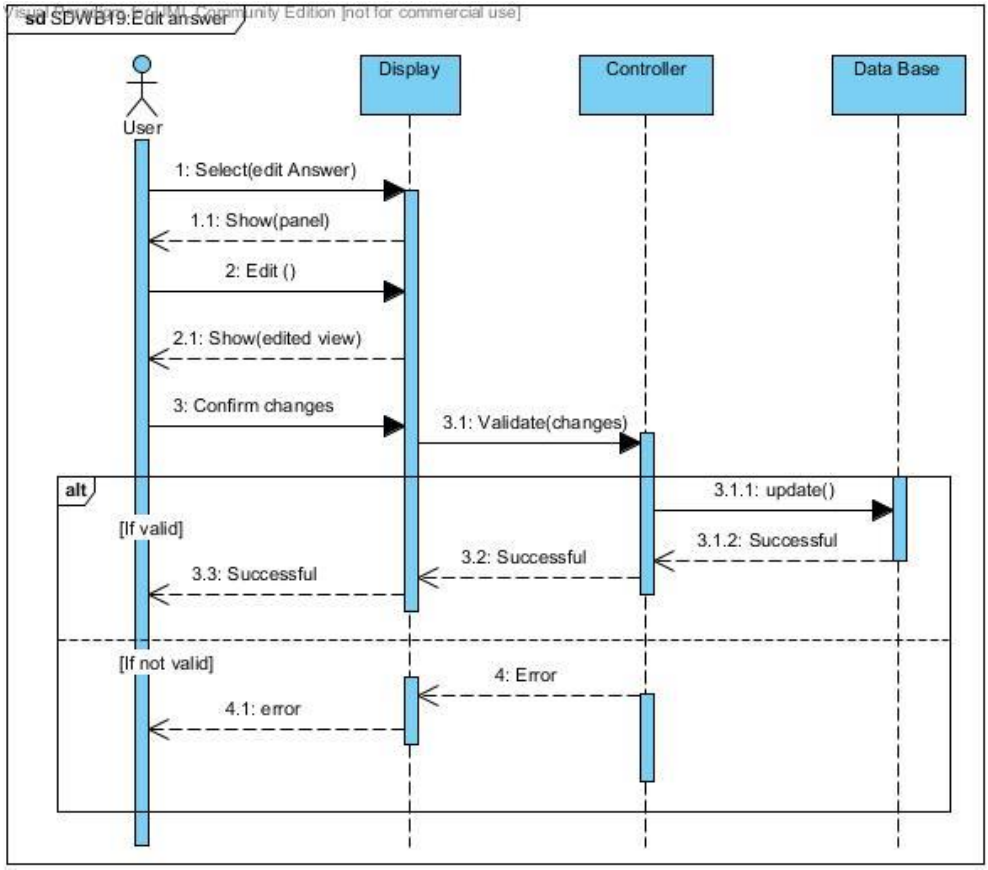
19. Edit Questions



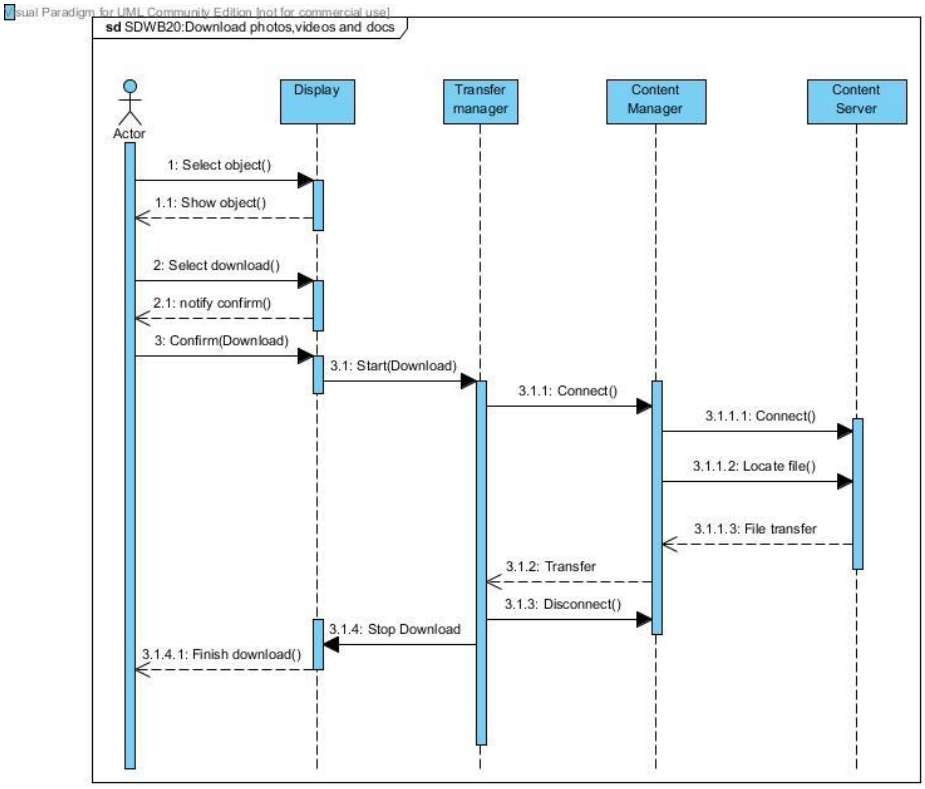
20. Answer Questions



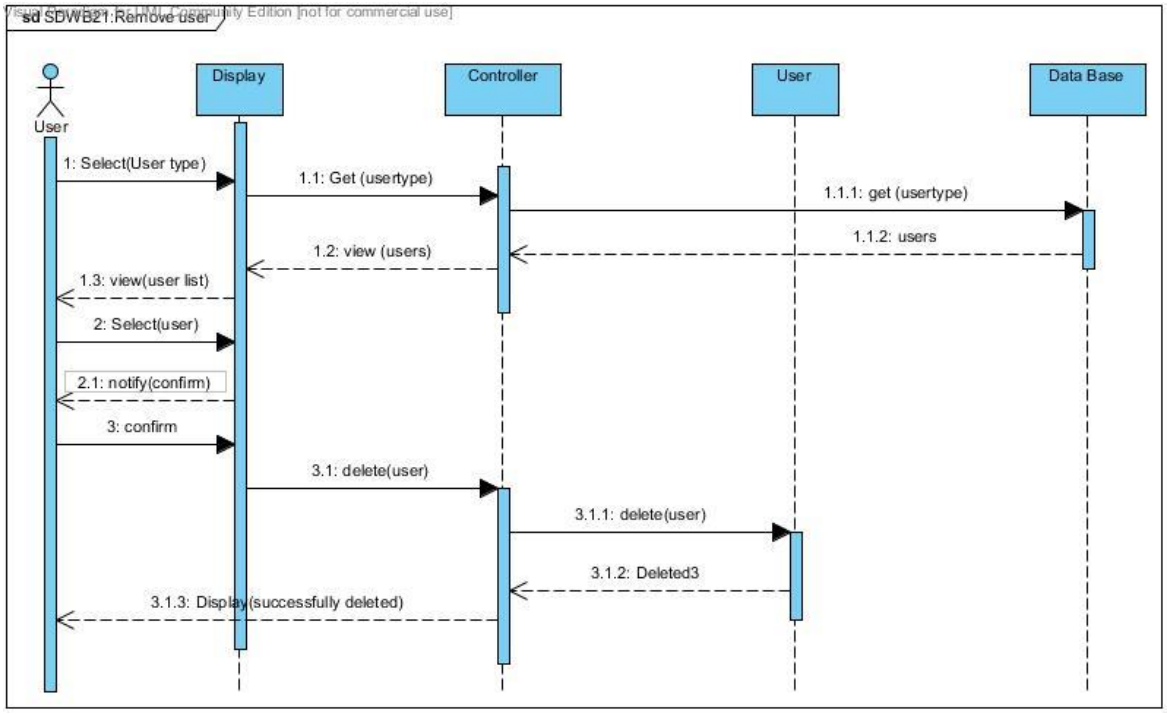
21. Edit Answer



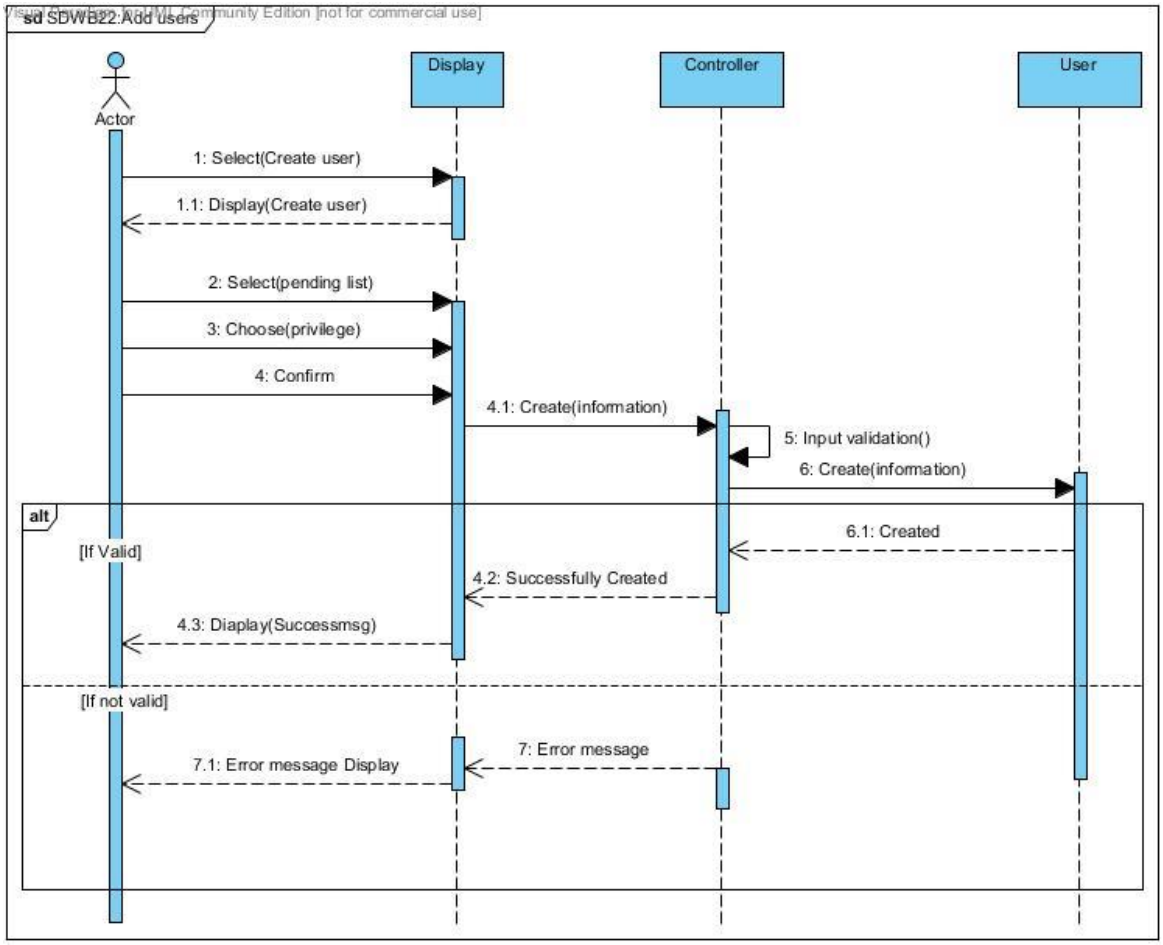
22. Download



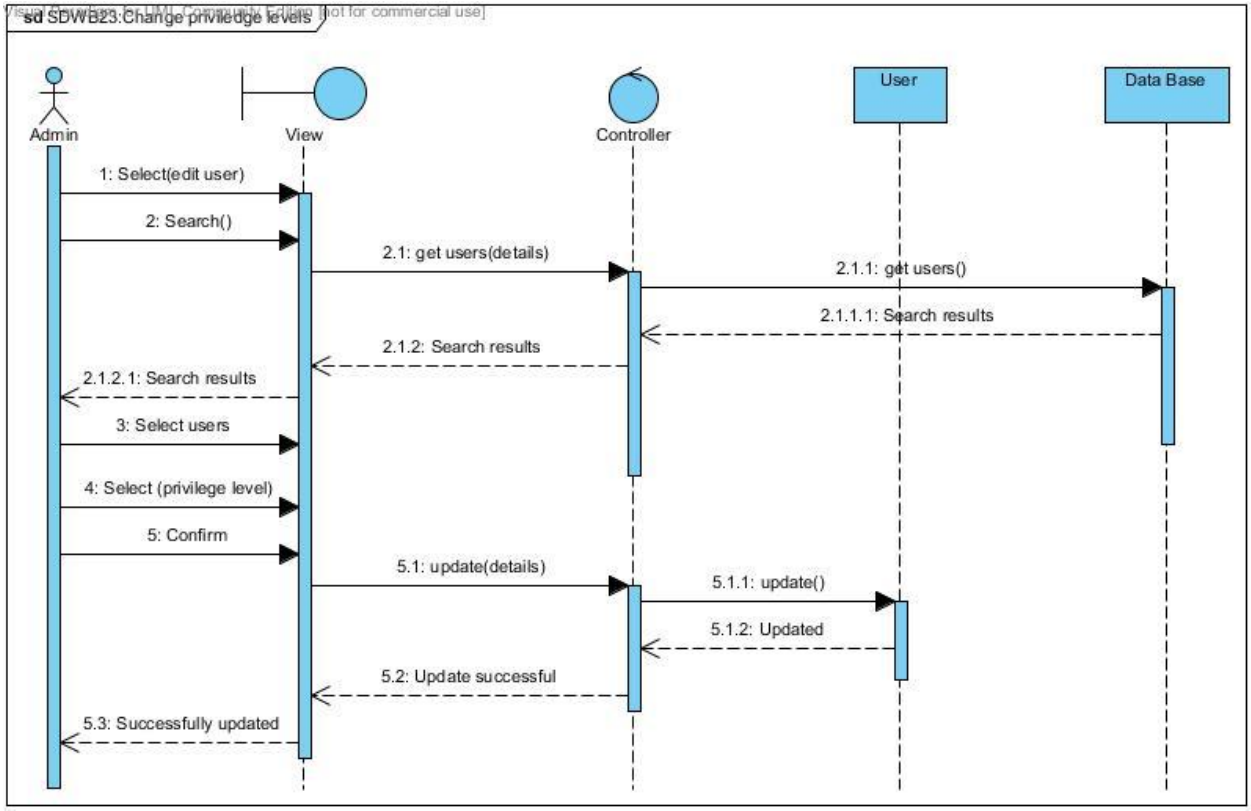
23. Remove user



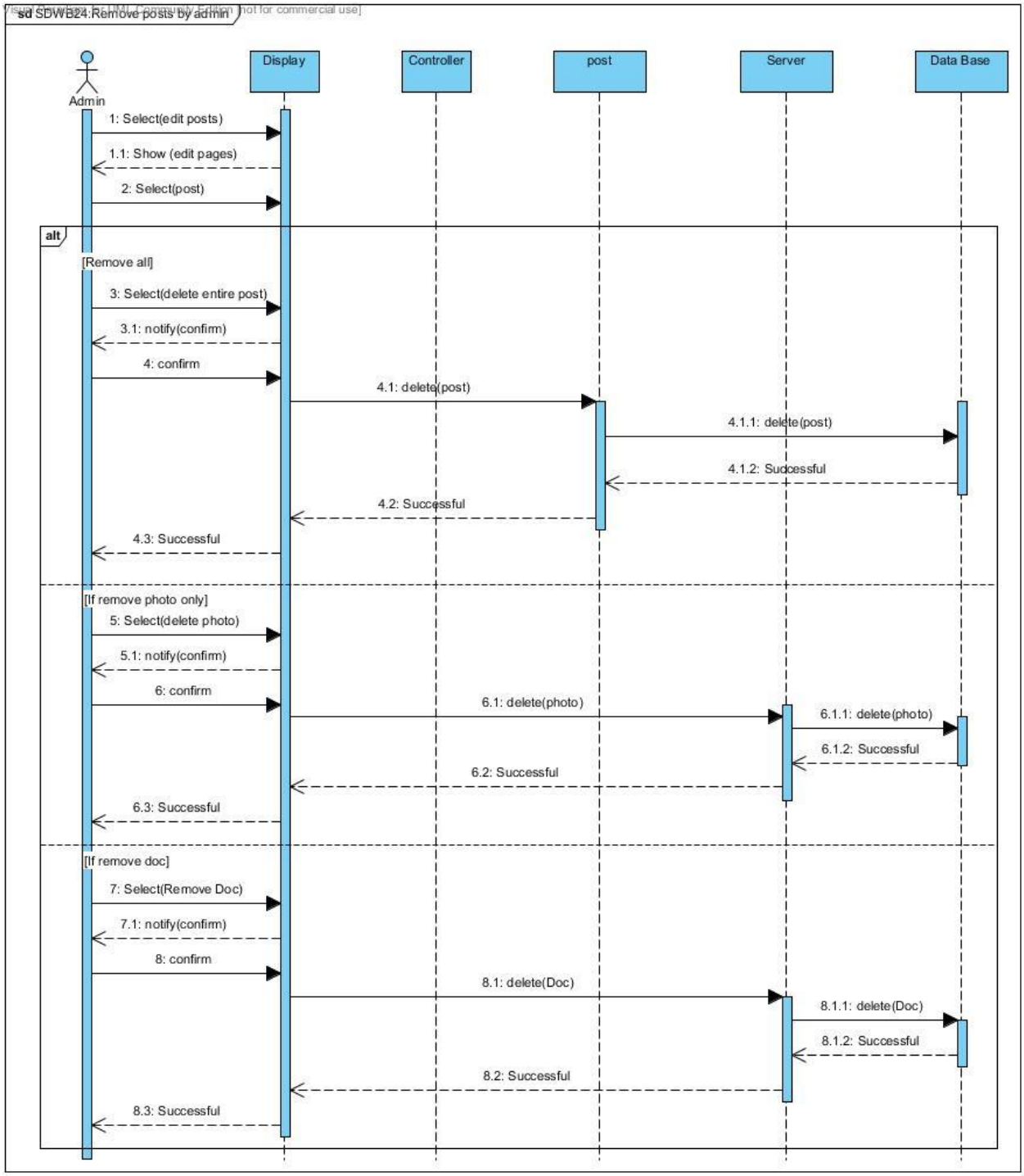
24. Add user



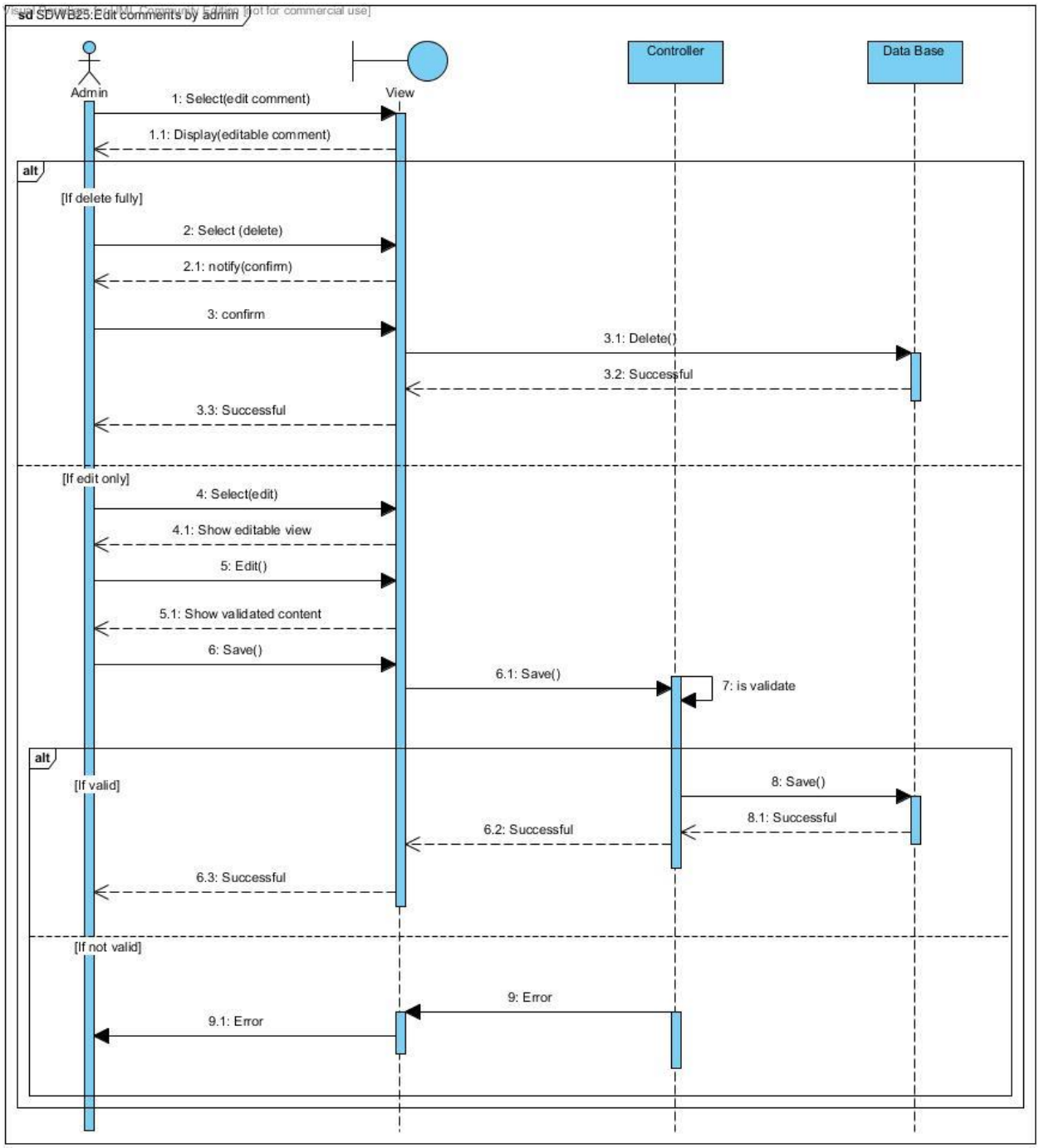
25. Change privileged levels by admin



26. Remove posts by admin



27. Edit comments by admin



3.3 ALGORITHM DESIGN

3.3.1. Filter algorithm for Map Component

1. Load Filter View

2. Select Elephant name

3. Select map type

4. Select ‘Show map’

Pseudo code

Map_period array = calculate date range from map type

Do while map_period = NULL

If map period matches elephant

    Return true

Else

    Return false

End Do

3.3.2. HealthData Filter Algorithm

1. Load Filter View

2. Select elephant name

3. Select data type

4. select freaquency

Pseudo code

data array = calculate from data type and frequency

Do while data= NULL

If data matches elephant

    Return true

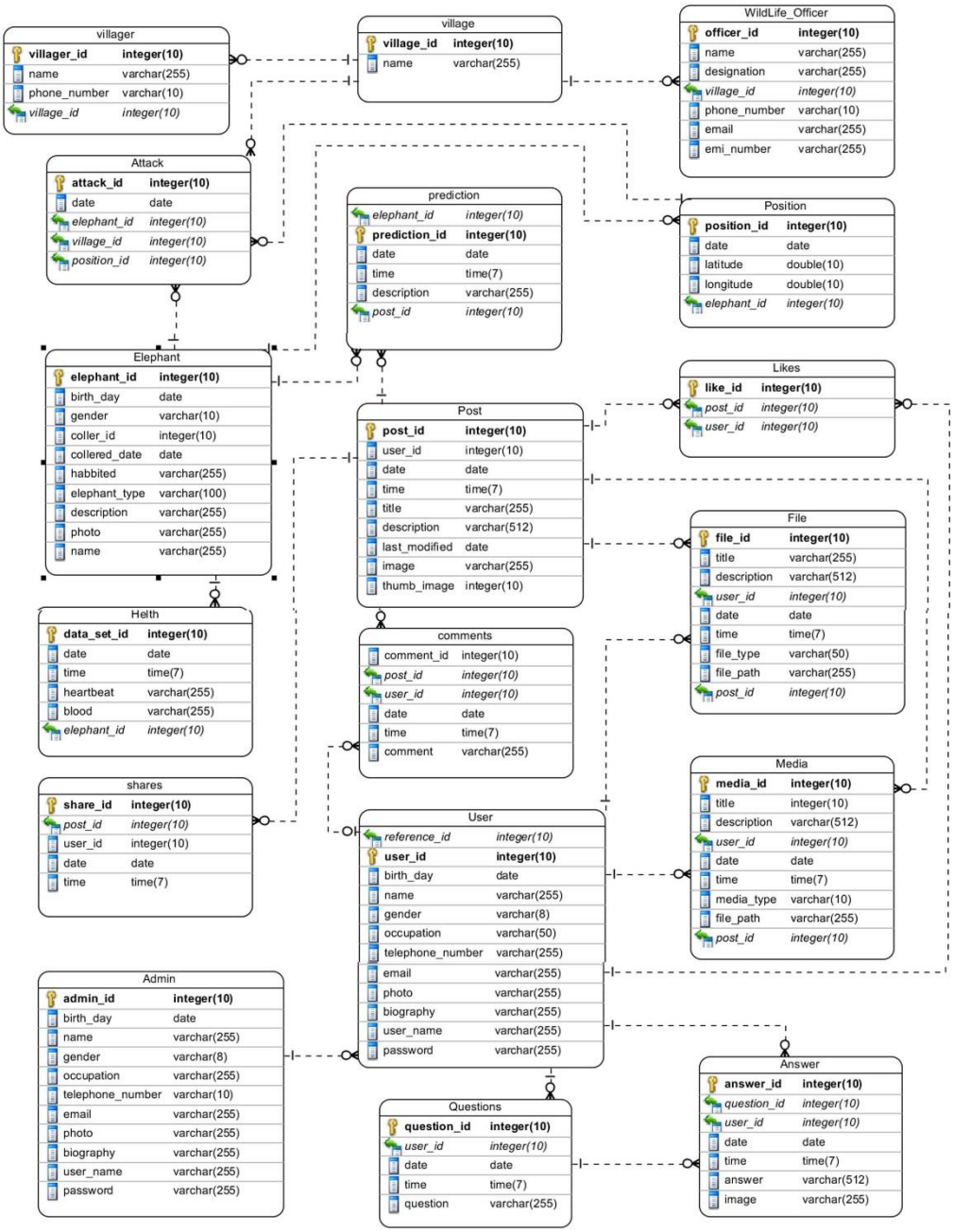
Else

    Return false

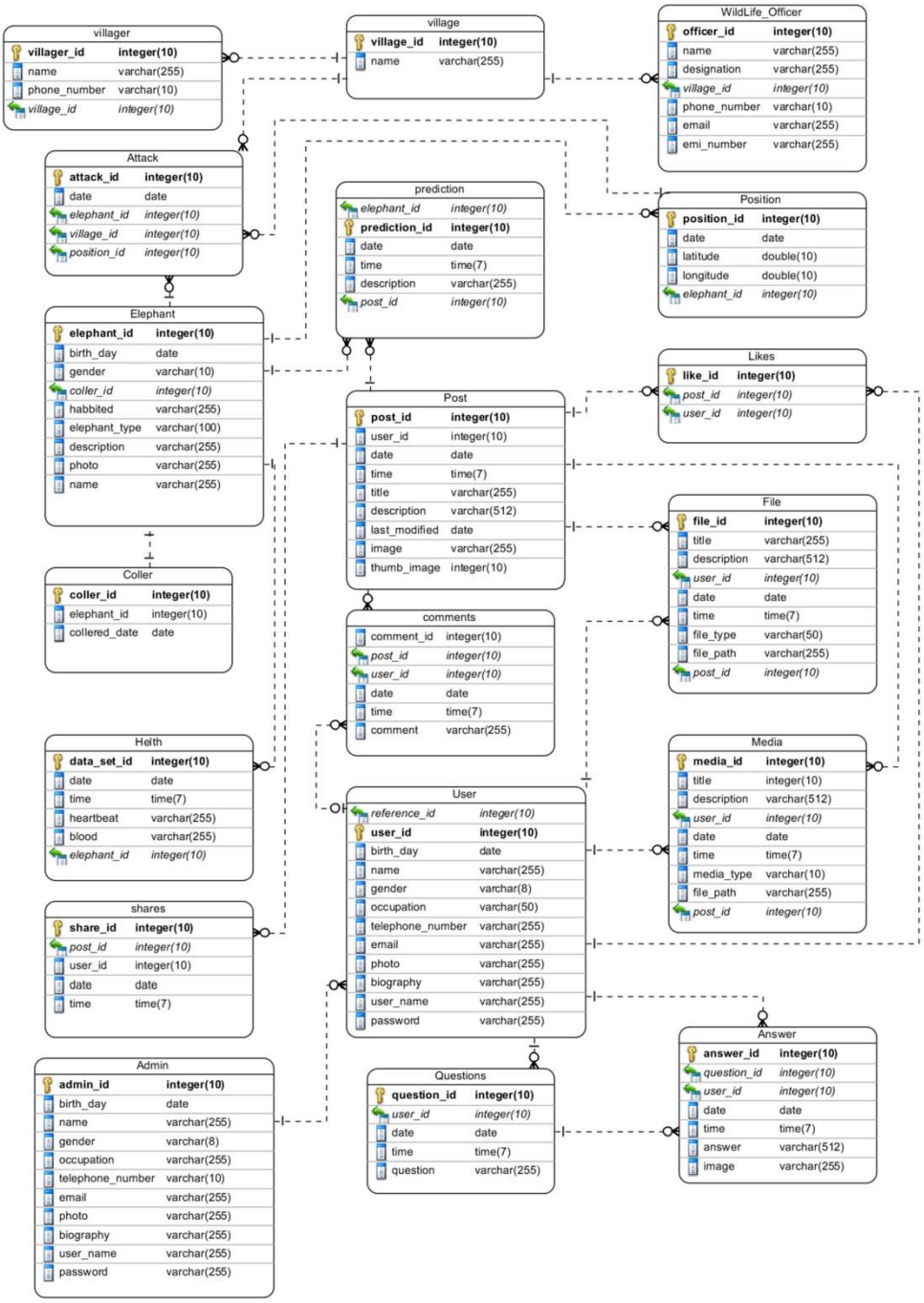
End Do

3.4 DATABASE DESIGN

3.4.1 Relational Model



3.4.2 Normalization/ Denormalization



3.4.3 Data Dictionary

Villager			
Field	**Data type**	**Null**	**Description**
Villager_id	**Integer(10)**	No	
name	Varchar(255)	No	
Phone_number	Varchar(10)	No	
Village_id	Integer(10)	No	

Attack			
Field	**Data type**	**Null**	**Description**
Attack_id	**Integer(10)**	No	
date	Date	No	
elephant_id	Integer(10)	No	
village_id	Integer(10)	No	
Position_id	Integer(10)	No	

Elephant			
Field	**Data type**	**Null**	**Description**
elephant_id	**Integer(10)**	No	
birth_day	Date	Yes	
gender	Varchar(10)	Yes	
coller_id	Integer(10)	Yes	
collered_date	Date	Yes	
habited	Varchar(255)	Yes	
elephant_type	Varchar(100)	No	
description	Varchar(255)	Yes	
photo	Varchar(255)	Yes	

Health			
Field	**Data type**	**Null**	**Description**
data_set_id	**Integer(10)**	No	
date	Date	No	
time	Time(7)	No	
heartbeat	Varchar(255)	Yes	
blood	Varchar(255)	Yes	
elephant_id	Integer(10)	No	

Admin			
Field	**Data type**	**Null**	**Description**
admin_id	**Integer(10)**	No	
birthday	Date	Yes	

name	Varchar(255)	No	
gender	Varchar(8)	No	
occupation	Varchar(255)	Yes	
telephone_number	Integer(10)	Yes	
email	Varchar(255)	No	
photo	Varchar(255)	Yes	
biography	Varchar(255)	Yes	
user_name	Varchar(255)	No	
password	Varchar(255)	No	

Village			
Field	**Data type**	**Null**	**Description**
village_id	**Integer(10)**	No	
name	Varchar(255)	No	

Prediction			
Field	**Data type**	**Null**	**Description**
elephant_id	Integer(10)	No	
prediction_id	**Integer(10)**	No	
date	Date	No	
time	Time(7)	No	
description	Varchar(255)	Yes	
post_id	Integer(10)	Yes	

Post			
Field	**Data type**	**Null**	**Description**
post_id	**Integer(10)**	No	
user_id	Integer(10)	No	
date	Date	No	
time	Time(7)	No	
title	Varchar(255)	Yes	
description	Varchar(512)	Yes	
last_modified	Date	Yes	
image	Varchar(255)	Yes	
thumb_image	Integer(10)	Yes	

Comments			
Field	**Data type**	**Null**	**Description**
comment_id	Integer(10)	No	
post_id	Integer(10)	No	
user_id	Integer(10)	No	
date	Date	No	
time	Time(7)	NO	
comment	Varchar(255)	No	

User			
Field	**Data type**	**Null**	**Description**
reference_id	Integer(10)	No	
user_id	**Integer(10)**	No	
birth_day	Date	No	
name	Varchar(255)	No	
gender	Varchar(8)	No	
occupation	Varchar(50)	Yes	
telephone_number	Varchar(255)	Yes	
email	Varchar(255)	NO	
photo	Varchar(255)	Yes	
biography	Varchar(255)	Yes	
user_name	Varchar(255)	No	
password	Varchar(255)	No	

Questions			
Field	**Data type**	**Null**	**Description**
question_id	**Integer(10)**	NO	
user_id	Integer(10)	No	
date	Date	No	
time	Time(7)	No	
question	Varchar(255)	NO	

Wildlife officer			
Field	**Data type**	**Null**	**Description**
Officer_id	**Integer(10)**	No	
name	Varchar(255)	No	
designation	Varchar(255)	No	
village_id	Integer(10)	NO	
phone_number	Varchar(10)	Yes	
email	Varchar(255)	No	
emi_number	Varchar(255)	No	

Position			
Field	**Data type**	**Null**	**Description**
position_id	**Integer(10)**	No	
date	Date	No	
latitude	Double(10)	No	
longitude	Double(10)	No	
elephant_id	Integer(10)	No	

Likes			
Field	**Data type**	**Null**	**Description**
like_id	**Integer(10)**	No	
post_id	Integer(10)	No	
user_id	Integer(10)	No	

File			
Field	**Data type**	**Null**	**Description**
file_id	**Integer(10)**	No	
title	Varchar(255)	Yes	
description	Varchar(512)	Yes	
user_id	Integer(10)	No	
date	Date	No	
time	Time(7)	No	
file_type	Varchar(50)	No	
file_path	Varchar(255)	No	
post_id	Integer(10)	No	

Media			
Field	**Data type**	**Null**	**Description**
media_id	**Integer(10)**	No	
title	Integer(10)	Yes	
description	Varchar(512)	Yes	
user_id	Integer(10)	No	
date	Date	No	
time	Time(7)	No	
media_type	Varchar(10)	No	
file_path	Varchar(255)	No	
post_id	Integer(10)	No	

Answer			
Field	**Data type**	**Null**	**Description**
answer_id	**Integer(10)**	No	
question_id	Integer(10)	No	
user_id	Integer(10)	No	
date	Date	No	
time	Time(7)	No	
answer	Varchar(512)	No	
image	Varchar(255)	Yes	

3.4.4. Indexes

Villager

Keyname	Type	Unique	Column
Primary	BTREE	Yes	Villager_ID
NAME	BTREE	No	name

Village

Keyname	Type	Unique	Column
Primary	BTREE	Yes	
NAME	BTREE	No	

Attack			
KeyName	Type	Unique	Column
PRIMARY	BTREE	yes	attack_id
E_ID	BTREE	yes	elephant_id

Prediction			
KeyName	Type	Unique	Column
PRIMARY	BTREE	yes	prediction_id
E_ID	BTREE	yes	elephant_id

WildLife			
KeyName	Type	Unique	Column
PRIMARY	BTREE	yes	officer_id
NAME	BTREE	No	name
T_PHONE	BTREE	Yes	phone_number
EMAIL	BTREE	Yes	email
EMI_NUMBER	BTREE	Yes	emi_number

Post			
KeyName	Type	Unique	Column
PRIMARY	BTREE	yes	post_id
AUTHER	BTREE	yes	user_id

Like			
KeyName	Type	Unique	Column
PRIMARY	BTREE	yes	like_id
AUTHER	BTREE	yes	user_id

Comment			
KeyName	Type	Unique	Column
PRIMARY	BTREE	yes	comment_id
AUTHER	BTREE	yes	user_id

File			
KeyName	Type	Unique	Column
PRIMARY	BTREE	yes	file_id
AUTHER	BTREE	yes	user_id

Share			
KeyName	Type	Unique	Column
PRIMARY	BTREE	yes	share_id
AUTHER	BTREE	yes	user_id

User			
KeyName	Type	Unique	Column
PRIMARY	BTREE	yes	user_id
NAME	BTREE	No	name
T_NUMBER	BTREE	Yes	phone_number
EMAIL	BTREE	Yes	email
USER_NAME	BTREE	Yes	user_name

Admin			
KeyName	Type	Unique	Column
PRIMARY	BTREE	yes	admin_id
NAME	BTREE	No	name
T_NUMBER	BTREE	Yes	phone_number
EMAIL	BTREE	Yes	email
USER_NAME	BTREE	Yes	user_name

Question			
KeyName	Type	Unique	Column
PRIMARY	BTREE	yes	question_id
AUTHER	BTREE	yes	user_id

Answer			
KeyName	Type	Unique	Column
PRIMARY	BTREE	yes	answer_id
Q_ID	BTREE	yes	question_id
AUTHER	BTREE	yes	user_id

Media			
KeyName	Type	Unique	Column
PRIMARY	BTREE	yes	media_id
AUTHER	BTREE	yes	user_id

Position			
KeyName	Type	Unique	Column
PRIMARY	BTREE	yes	position_id
E_ID	BTREE	yes	user_id
C_ID	BTREE	yes	coller_id
NAME	BTREE	No	name

3.5. USER INTERFACES

3.6. RULES AND GUIDELINES FOR INTERFACE DESIGN

3.6.1 User interface design framework

This Project we use Bootstrap for front view to get responsive (http://getbootstrap.com/) for Web interfaces.

Reason: - This framework is very much helpful for a clean and professional design. Since this is a very popular framework and developed by a reputed company, framework documentation is well formed and there are many places that we can get help.

3.6.2 User Input validation methods

Basically this system will do all the form and data validation in the Model level. Given below is an illustration for the basic form validation process via Model.

Reason: - The Spring framework runs on top of an Active Record design pattern to connect with the database. So that feature enables us to have this kind of validation architecture which is more secure and readable.

3.6.3 Alert messages decomposition

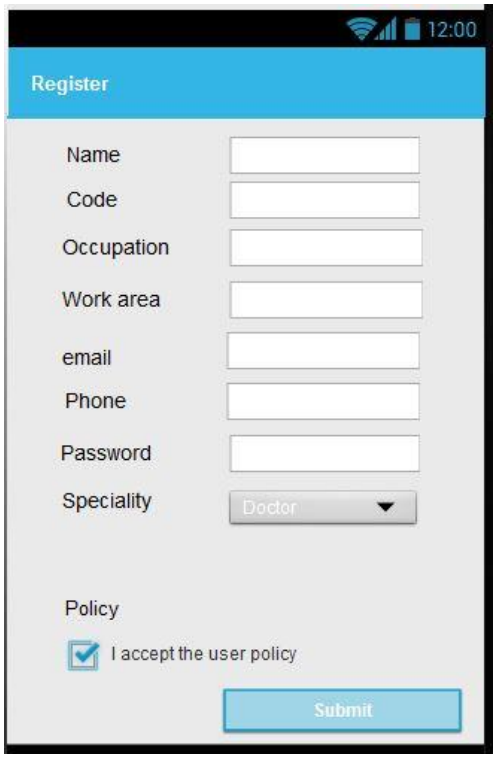
We'll be using following type of alert scheme for all over the system. So that people who are using this system will identify the type of the message just using the color of it.

3.7. USER INTERFACES DESIGN

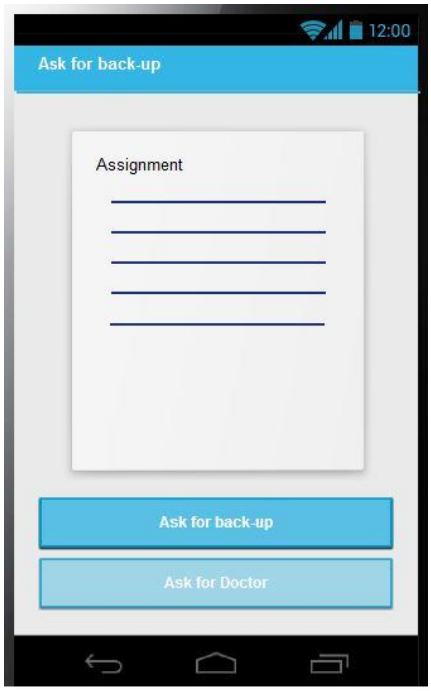
3.7.1. HEC part

1. Subscribe

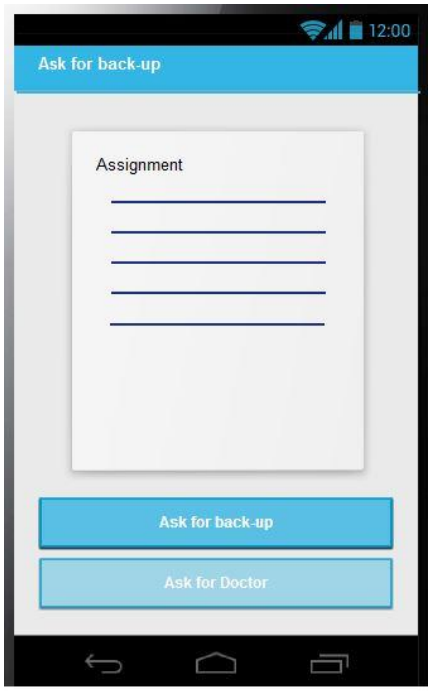
2. Register



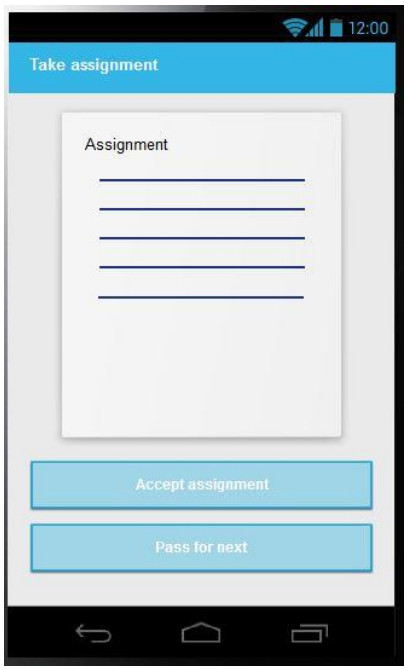
13. Ask for Backup



14. Ask for a Doctor

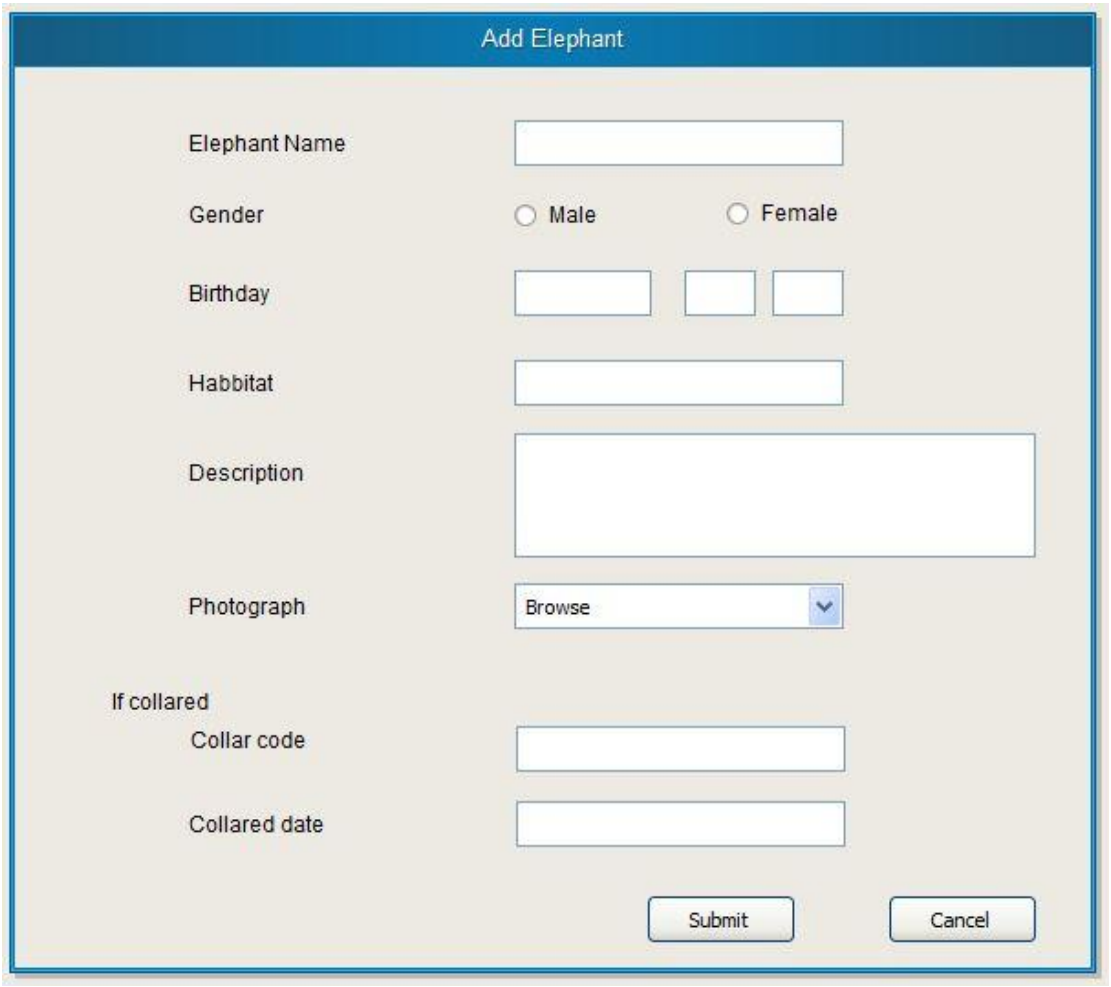


15. Take assignment

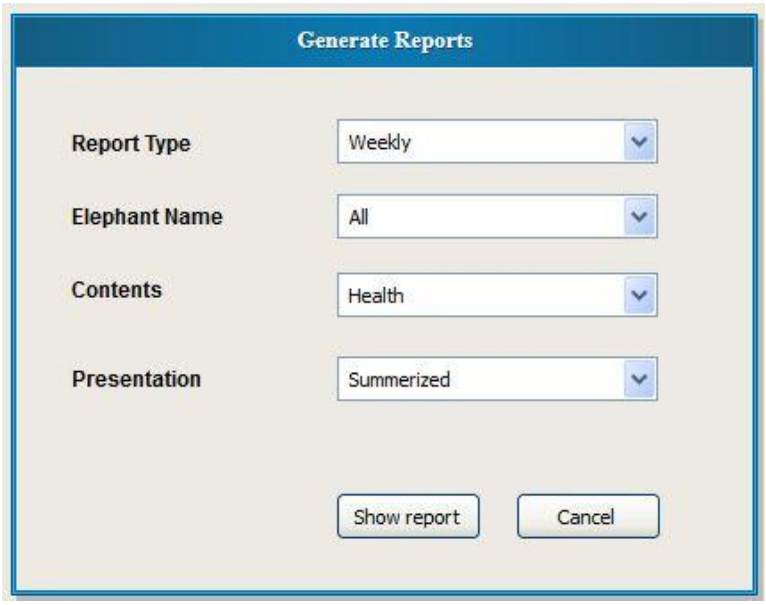


3.7.2. Analyzer

1. Add Elephant

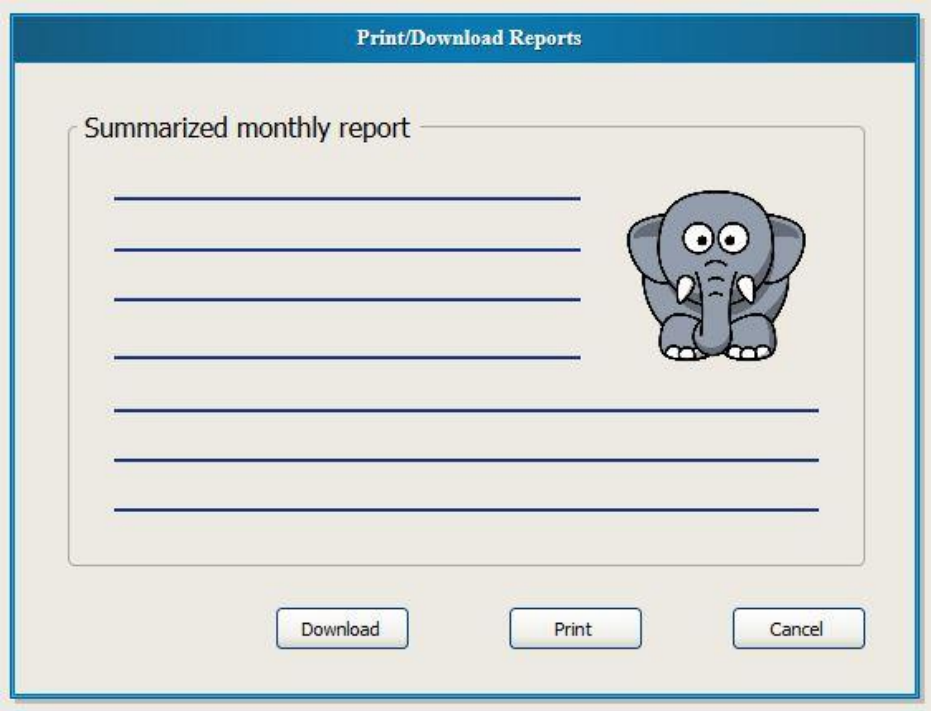


2. Generate reports

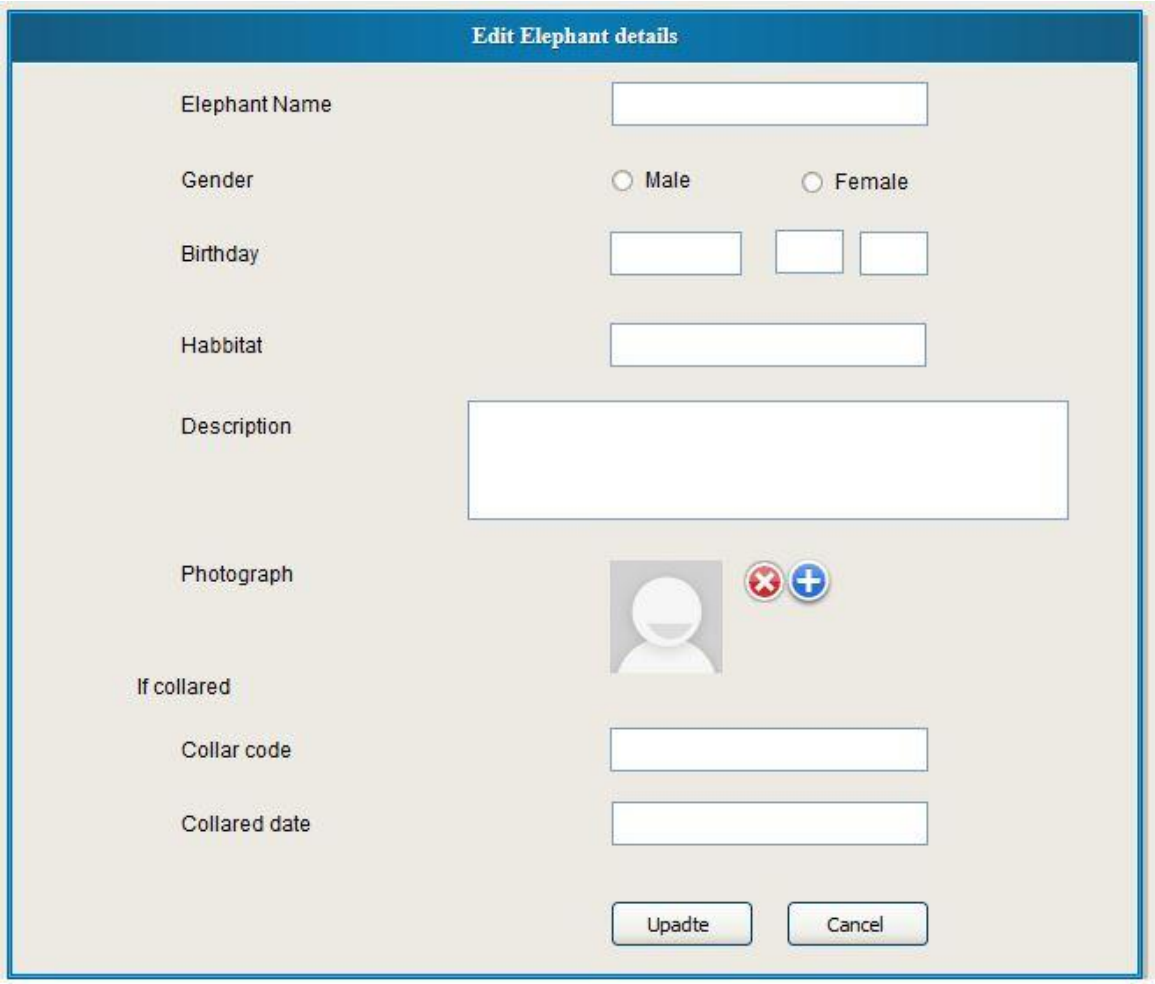


3. Print reports

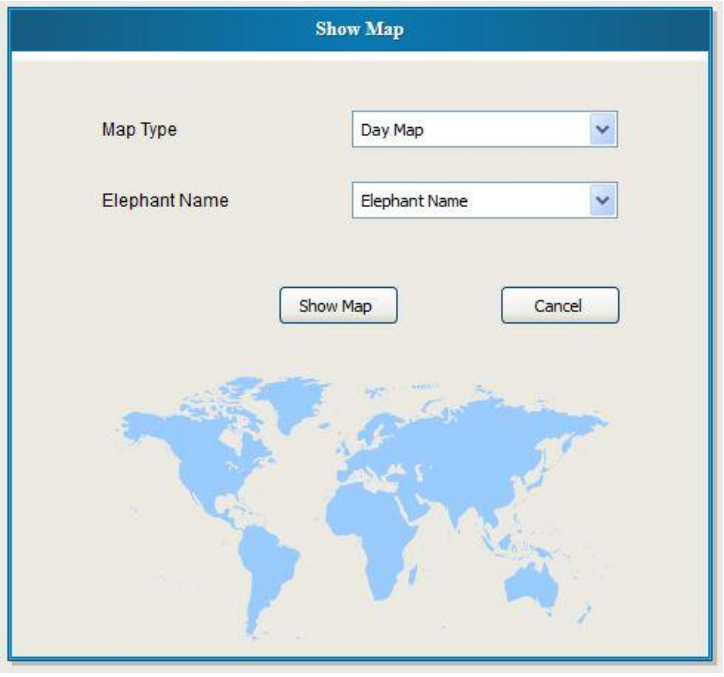
4. Download reports



5. Edit elephant Details



6. Show map



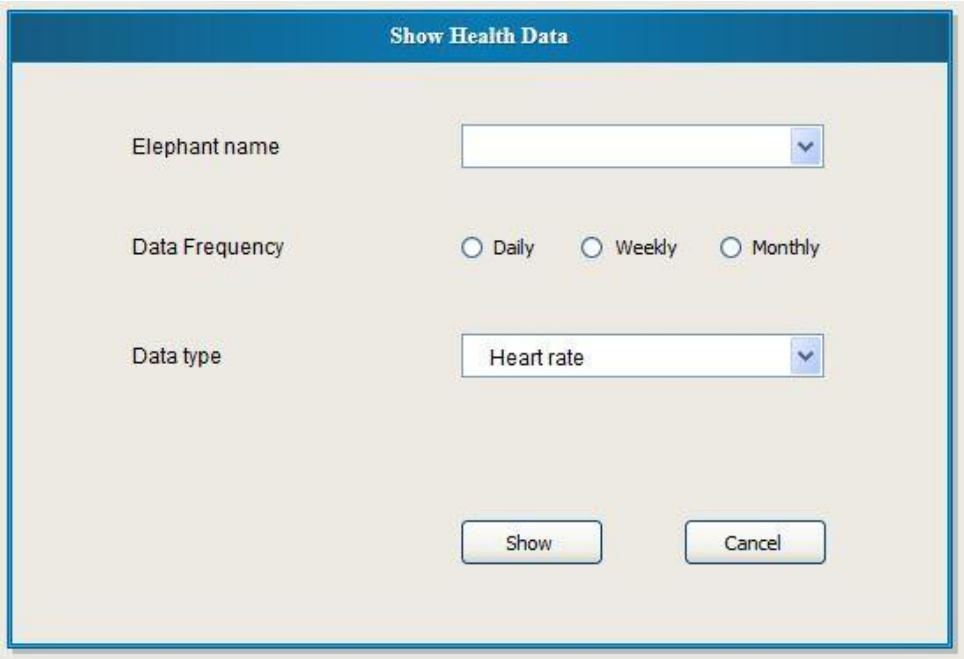
7. Position Update

8. Health data update

9. Generate health predictions

10. Generate health and safety alerts

11. Show health data

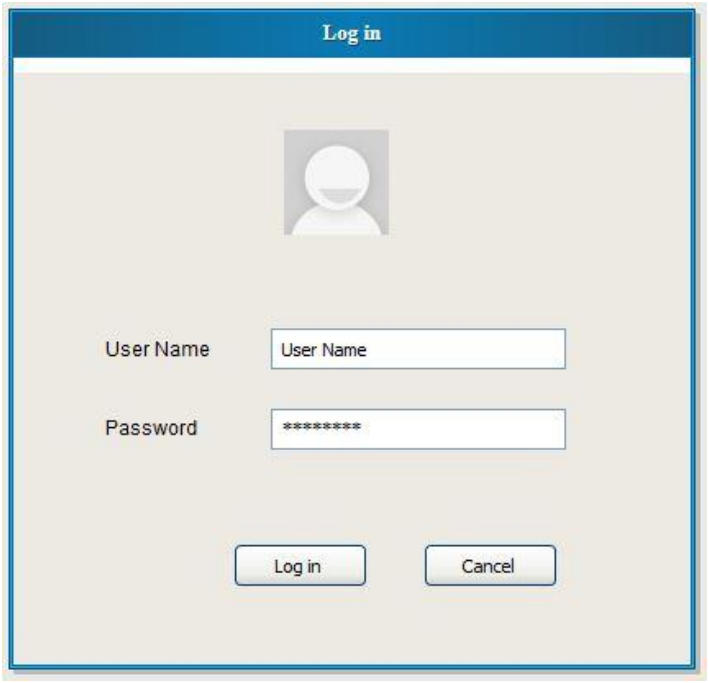


12. Show health information (Predictions)

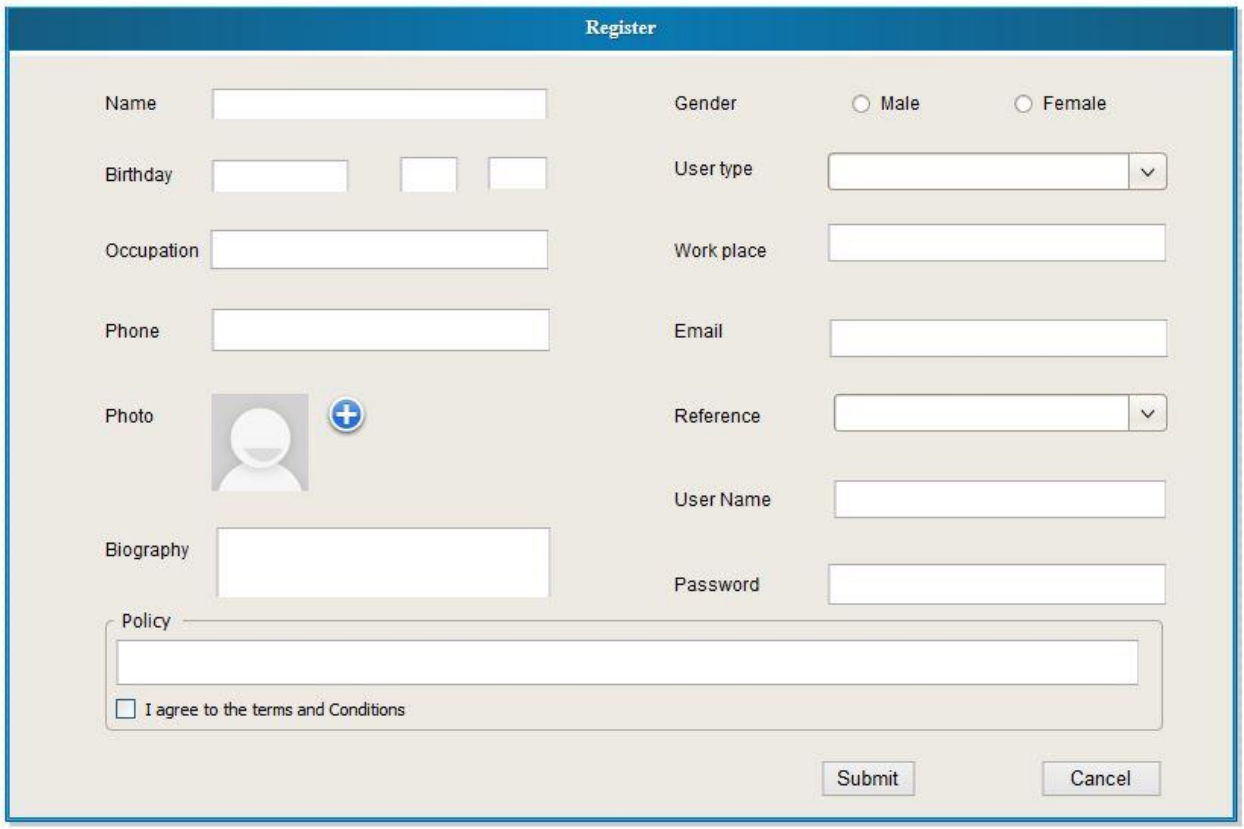
13. Auto show information on screen

3.7.3. Web part

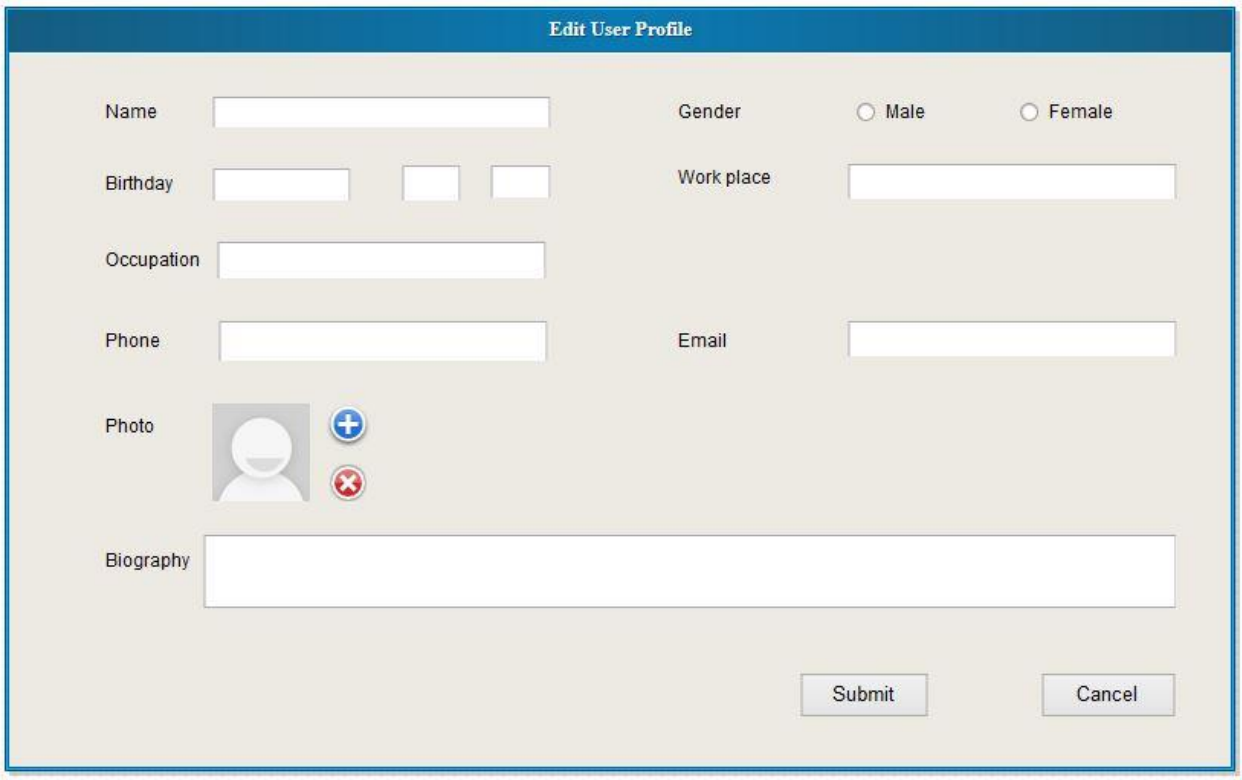
1. Login



2. Register

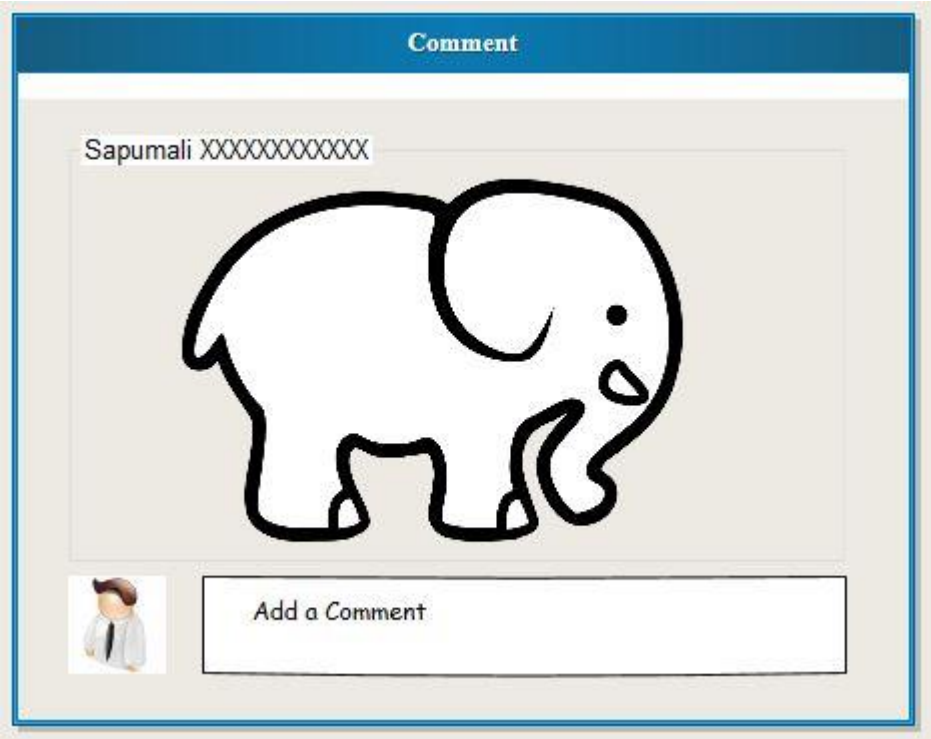


3. Edit User Profile

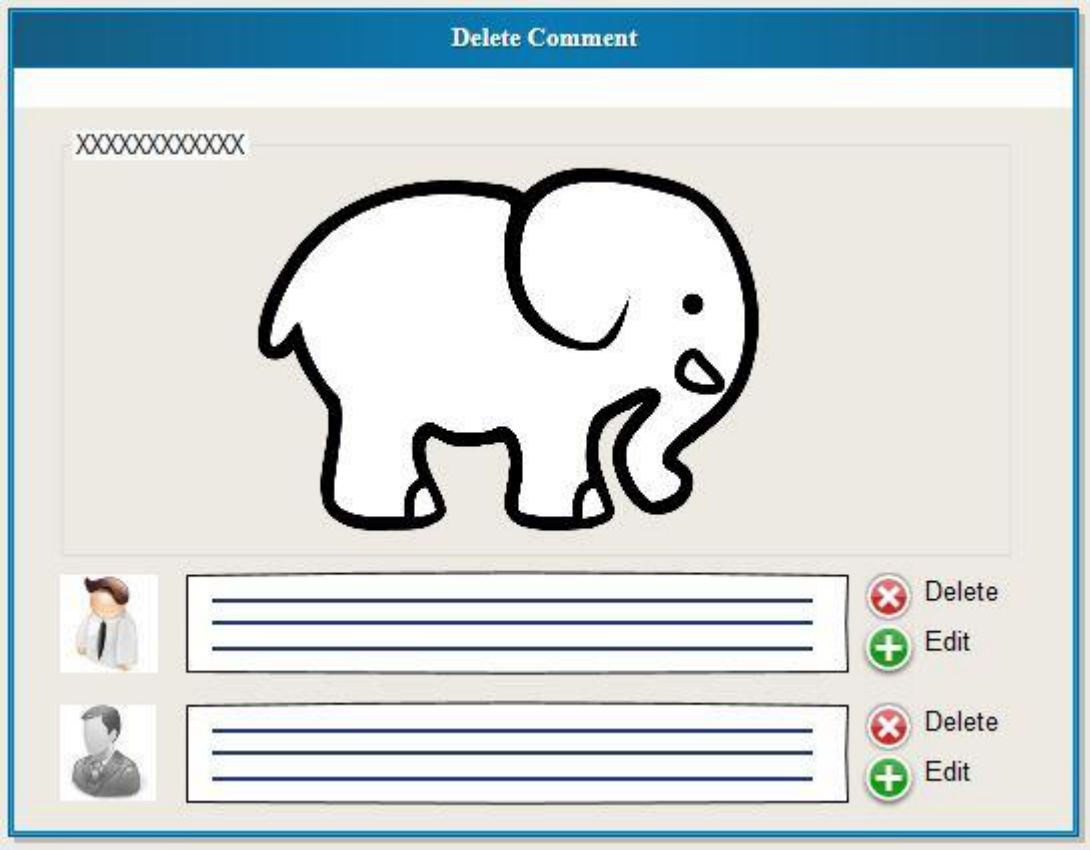


4. Change password

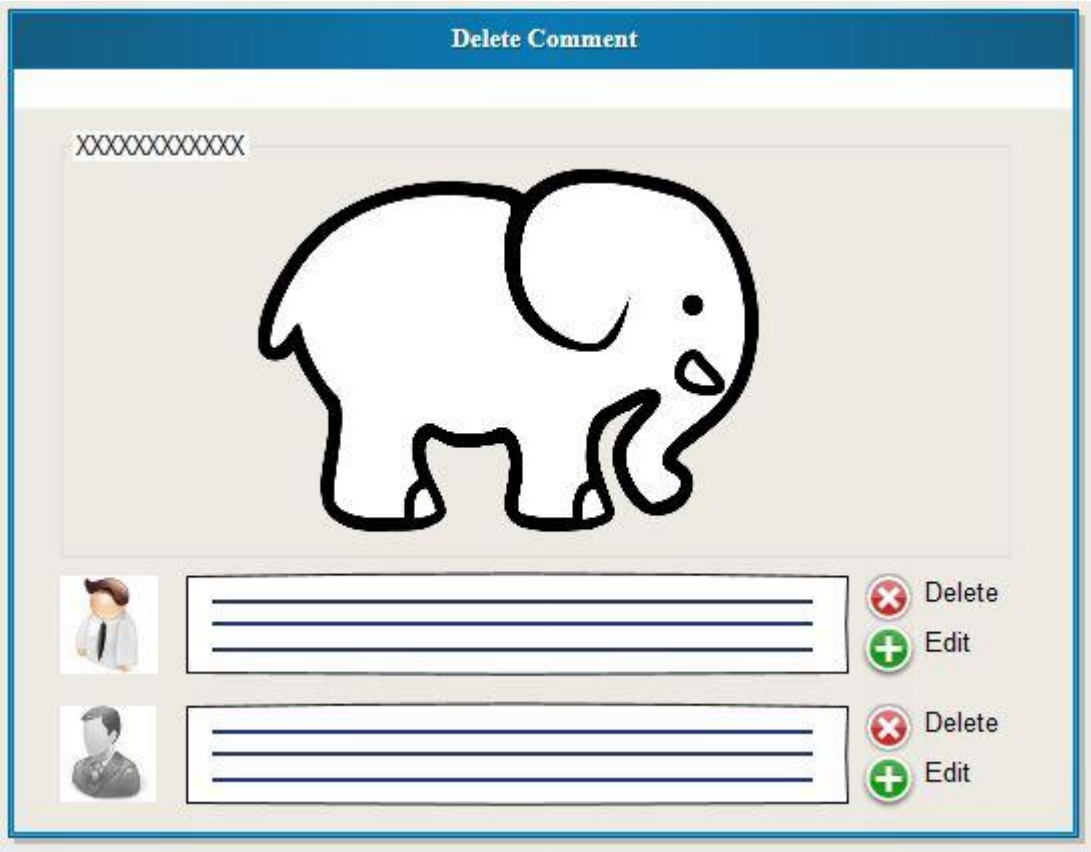
5. Comment



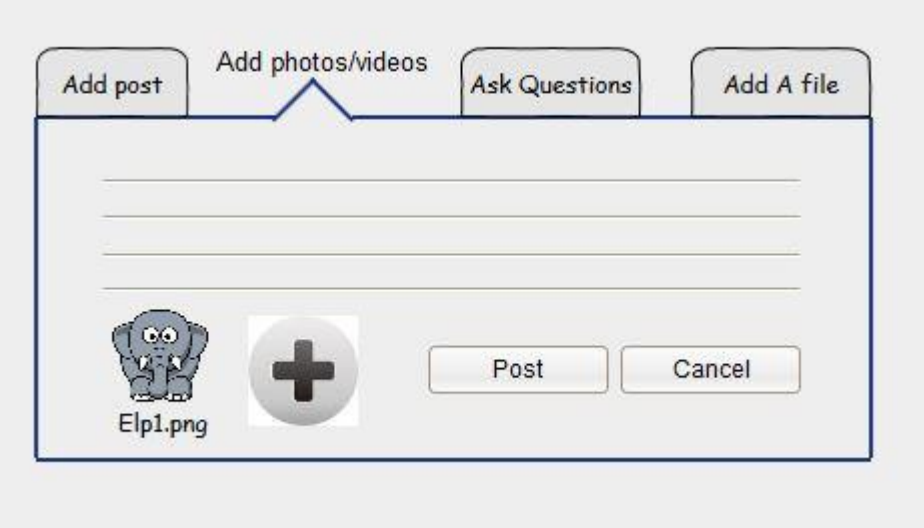
6. Remove Comment



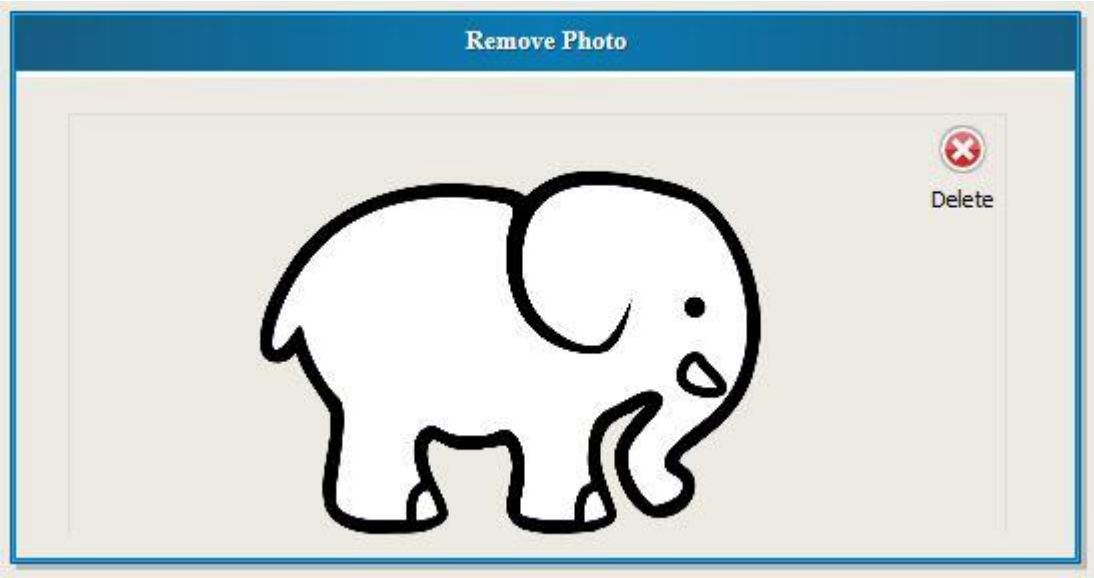
7. Edit Comment



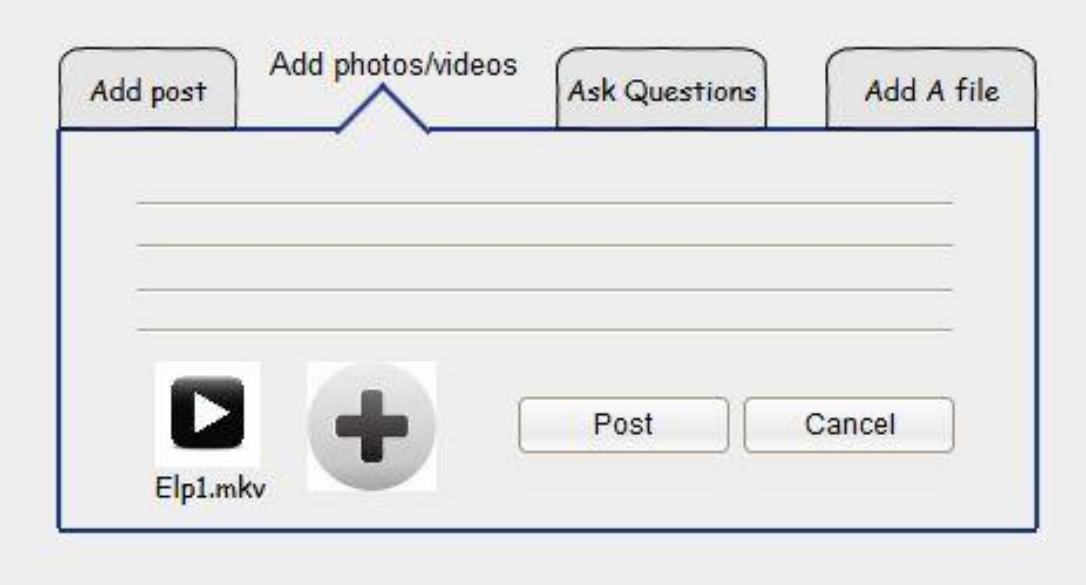
8. Add photo



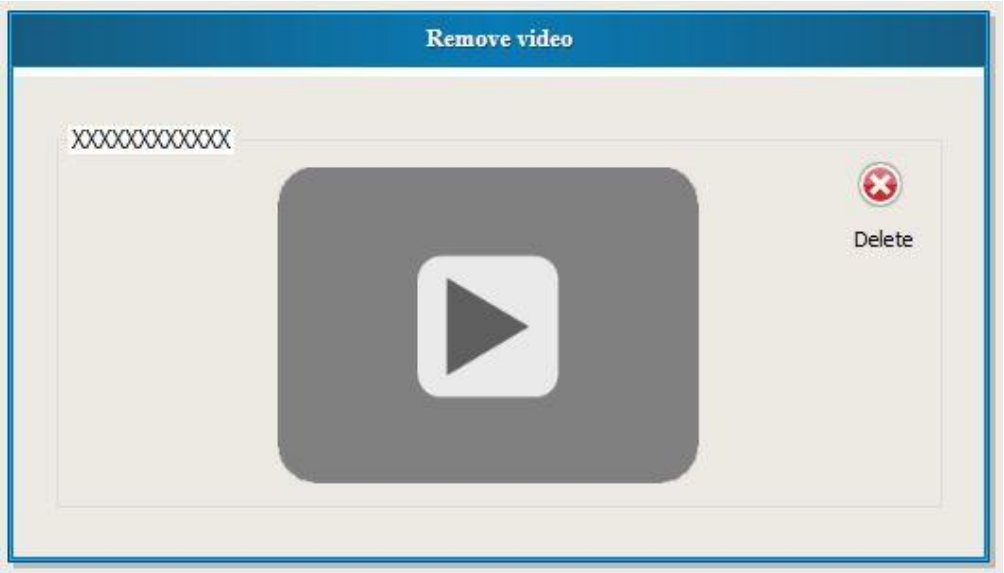
9. Remove photo



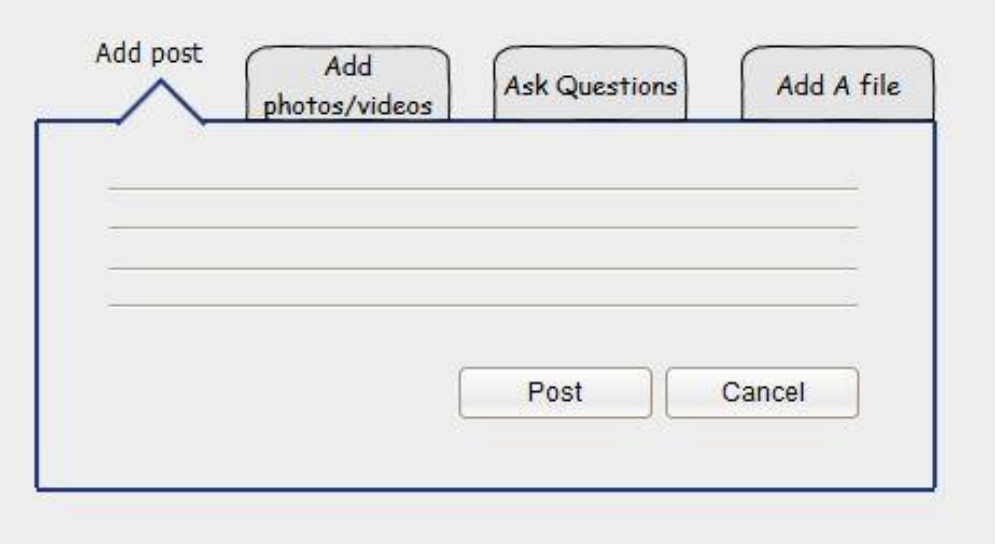
10. Add Video



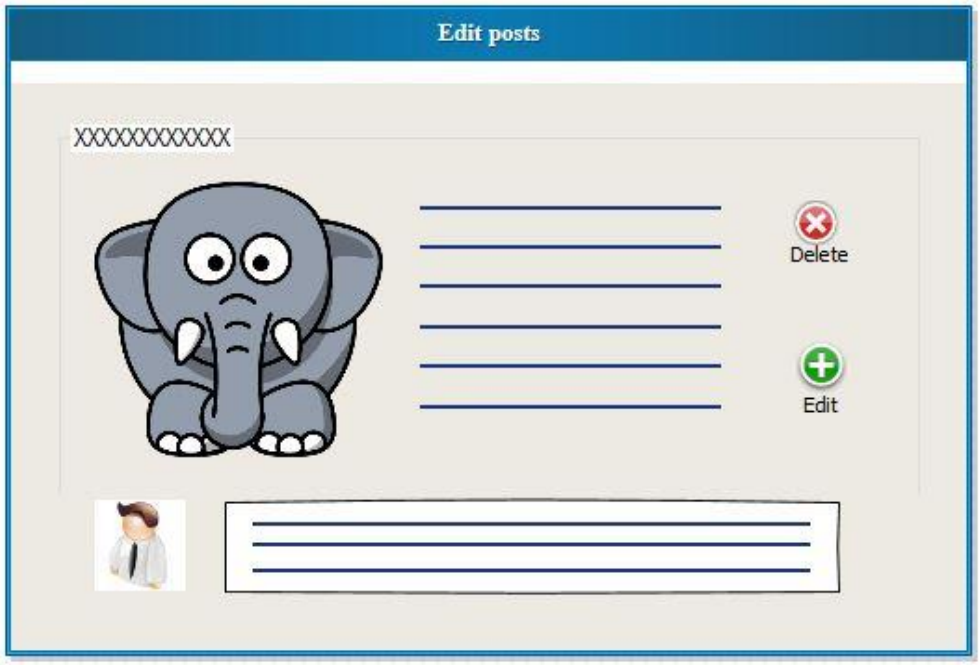
11. Remove video



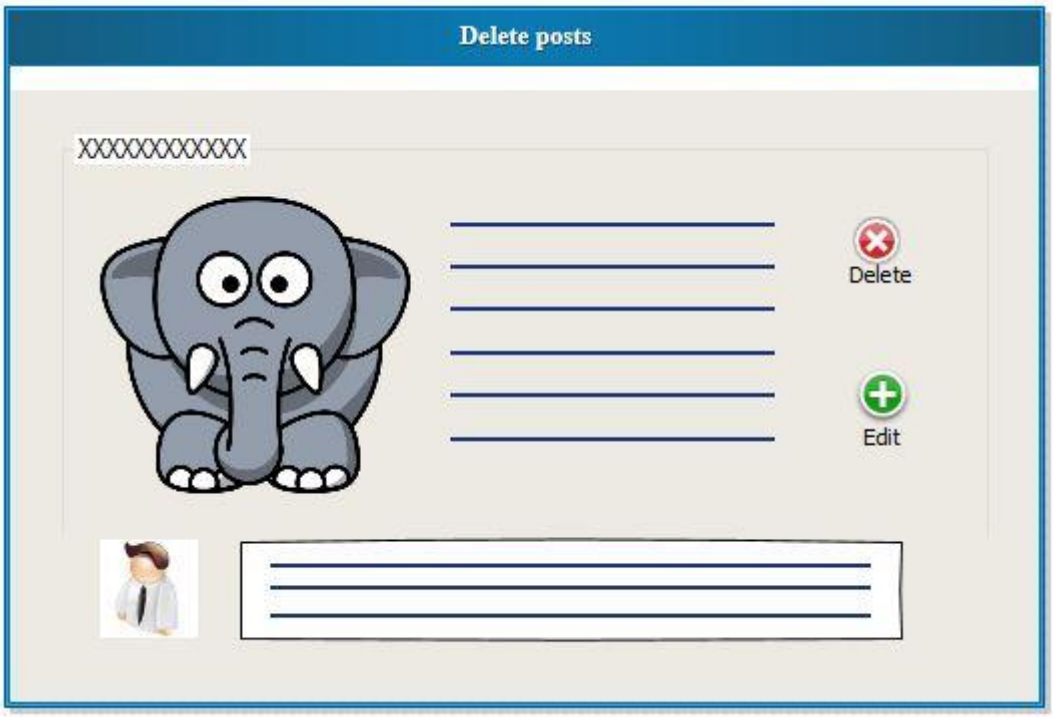
12. Add posts



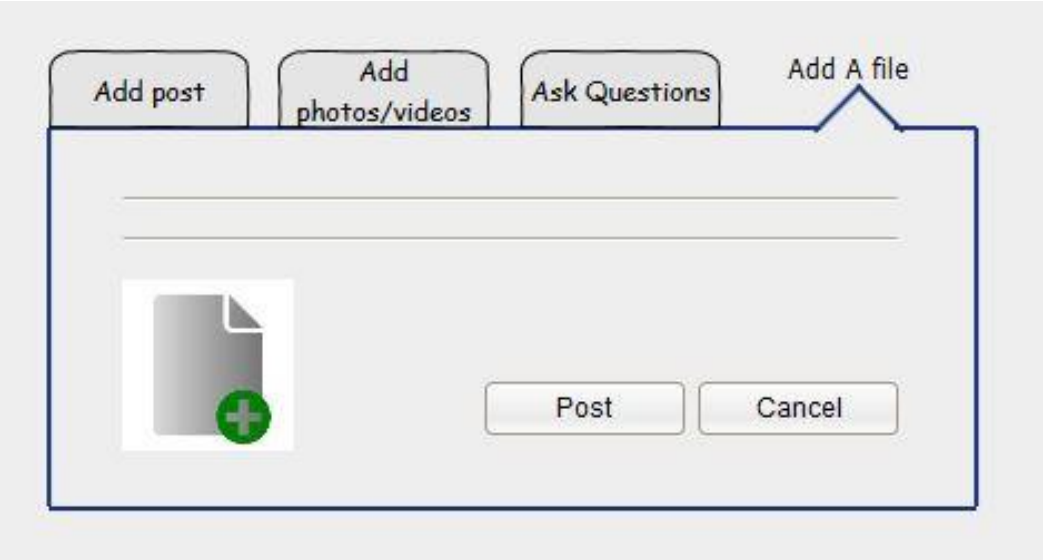
13. Edit posts



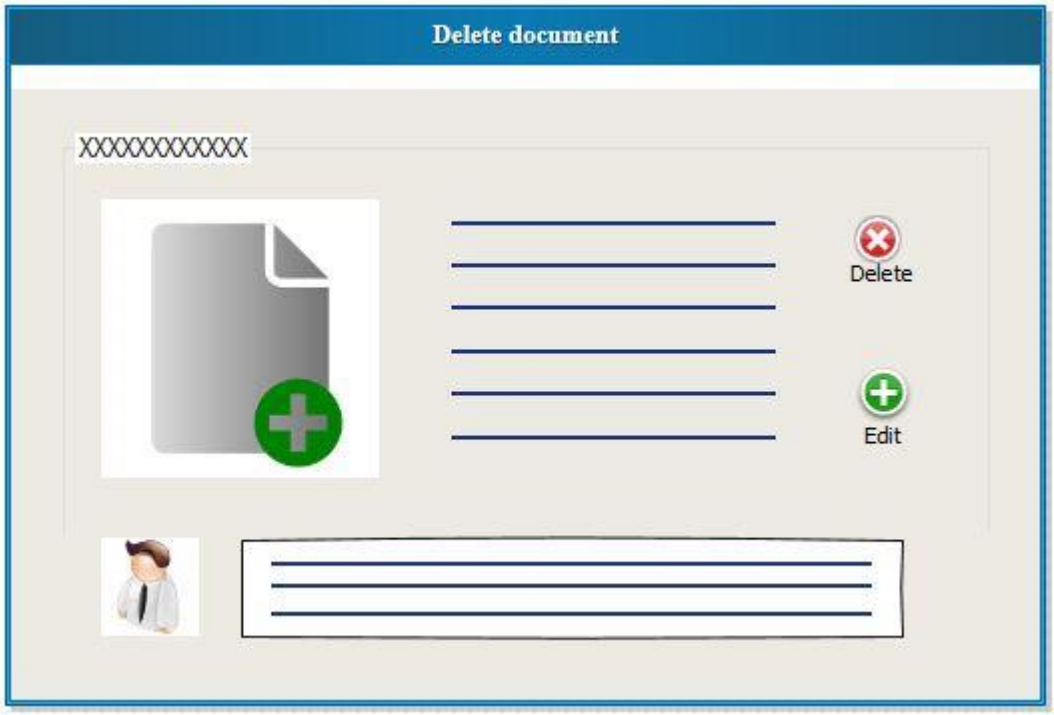
14. Delete Posts



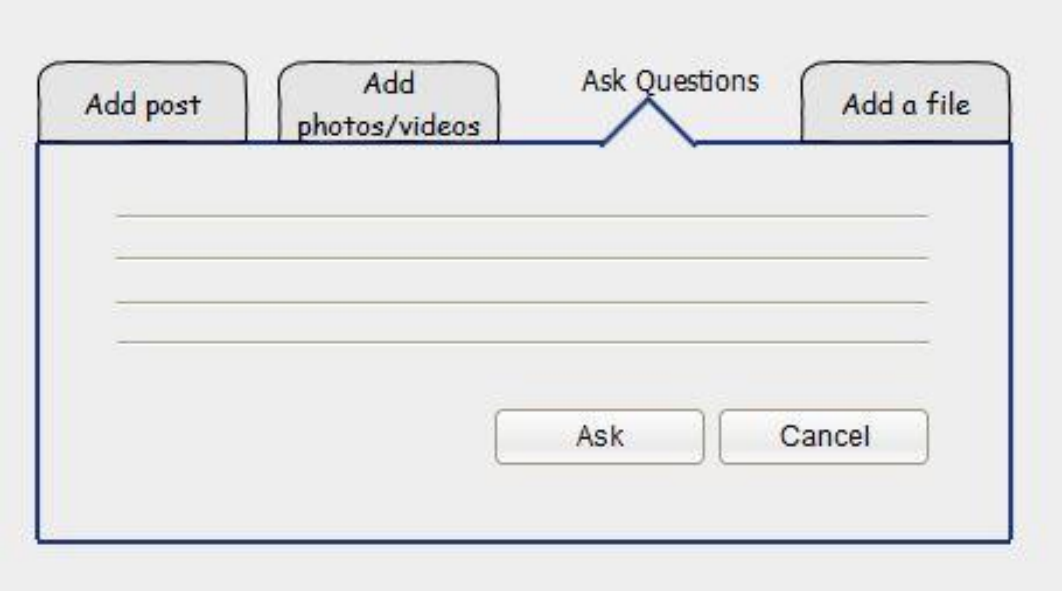
15. Upload Document



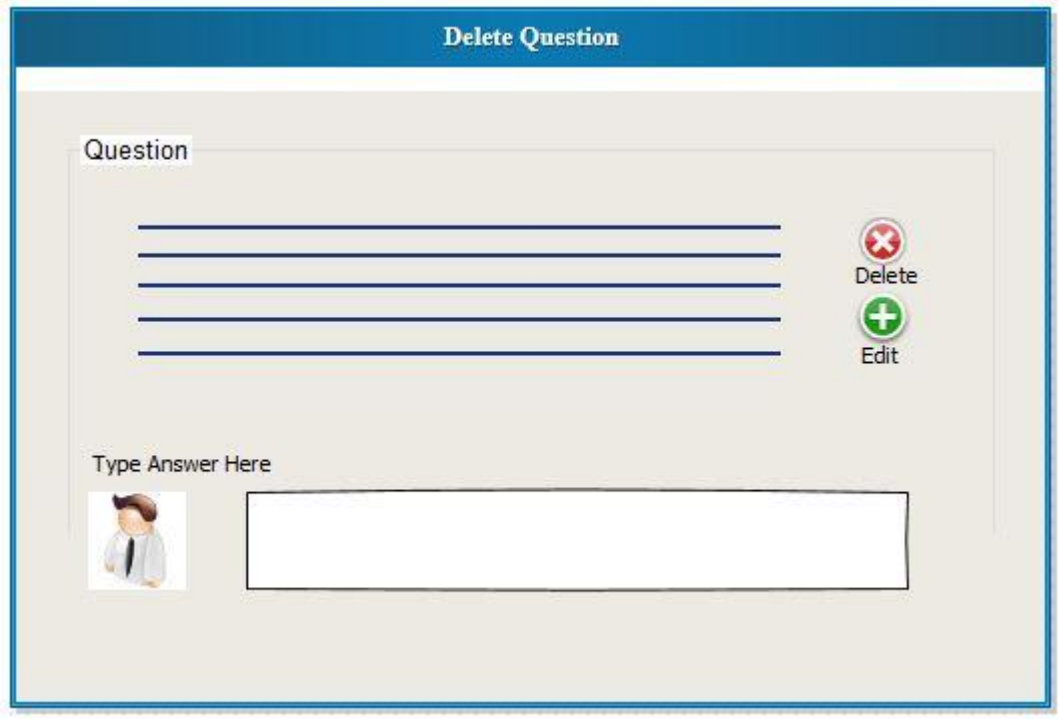
16. Delete Document



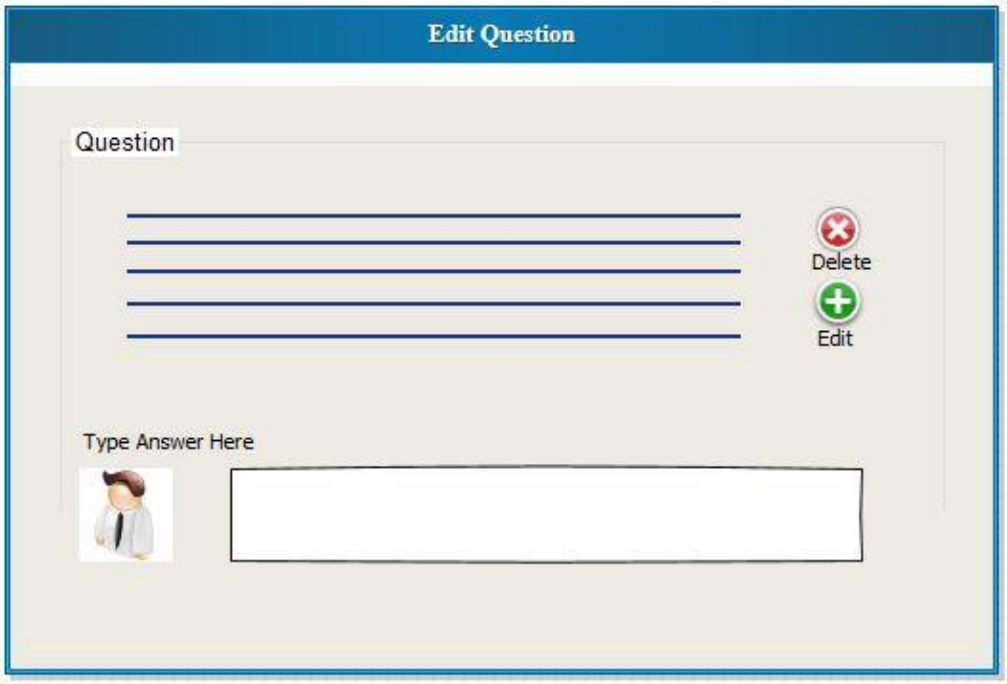
17. Ask Questions



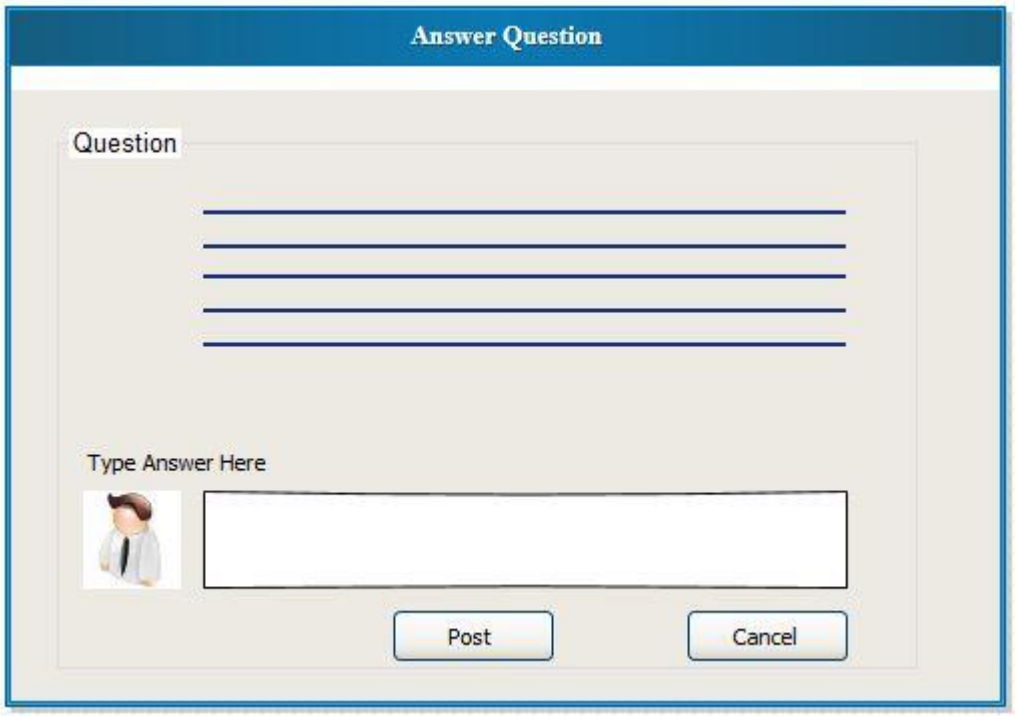
18. Delete Questions



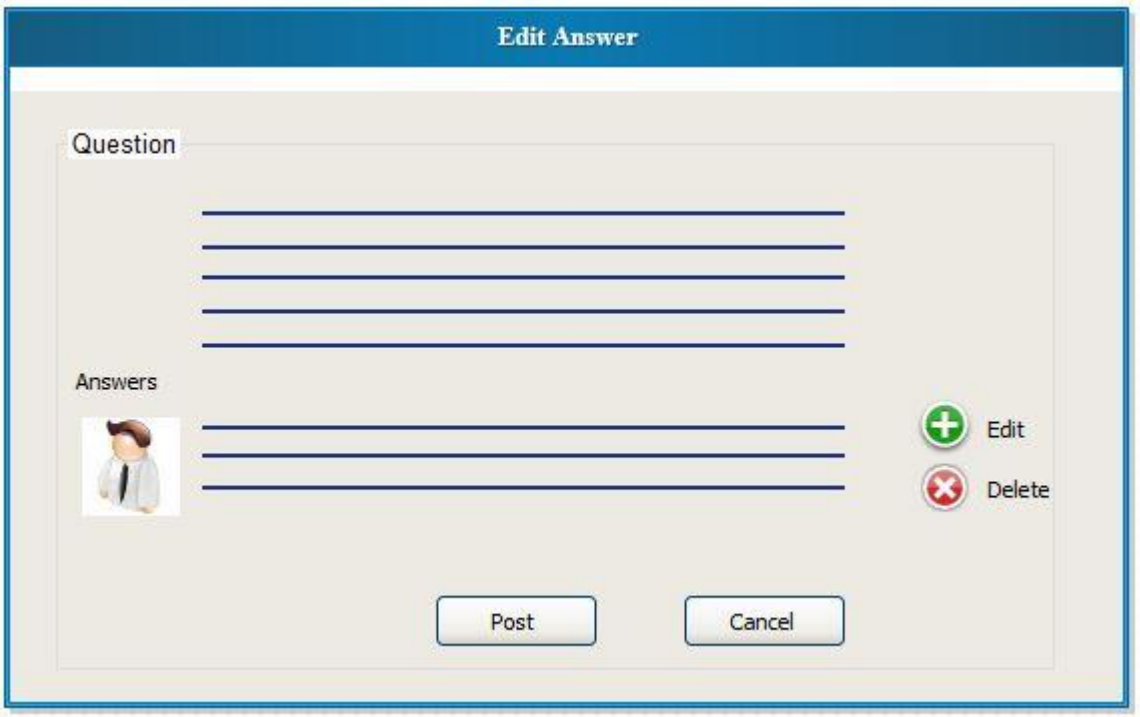
19. Edit Questions



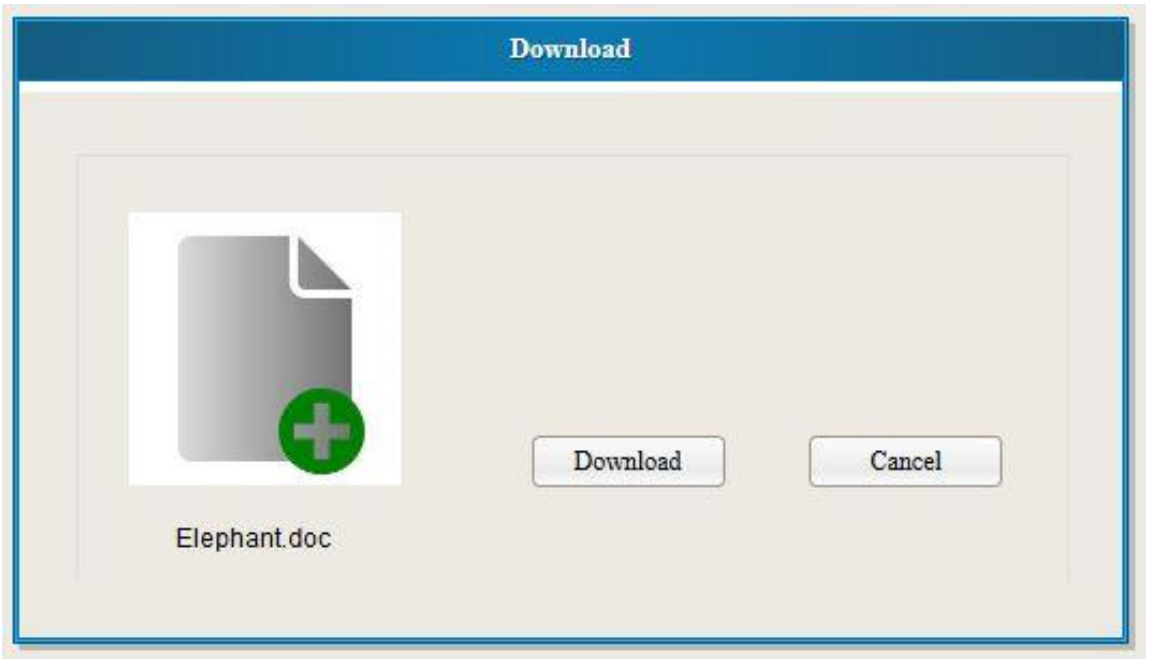
20. Answer Questions



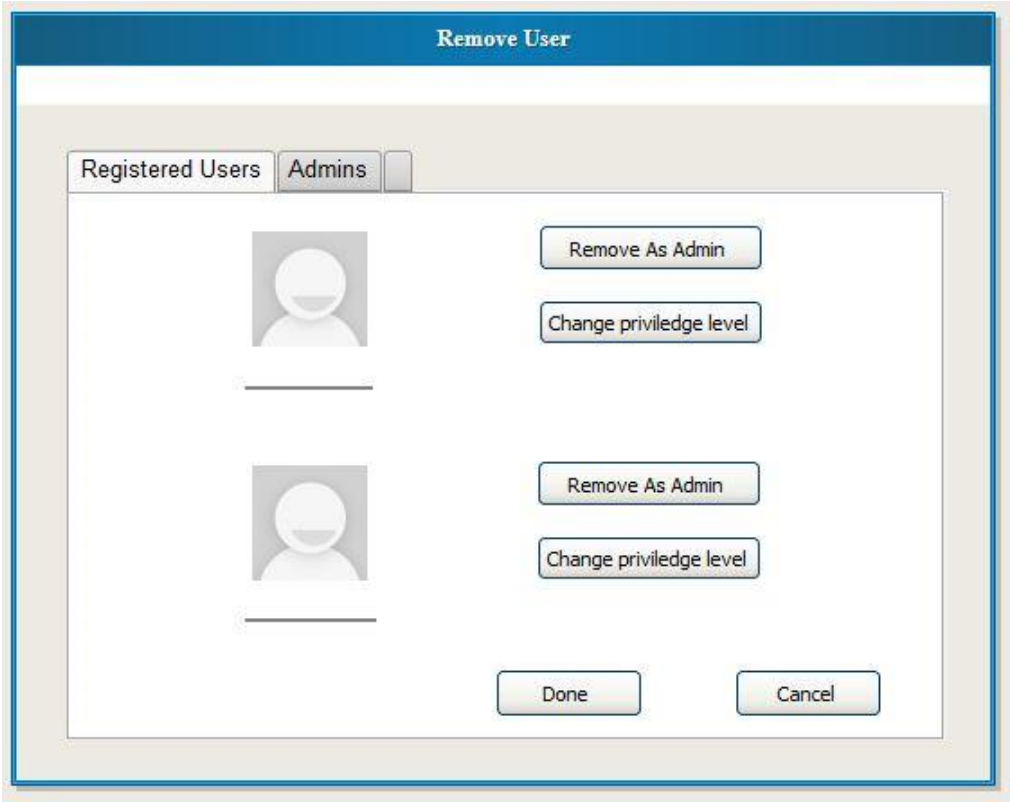
21. Edit Answer



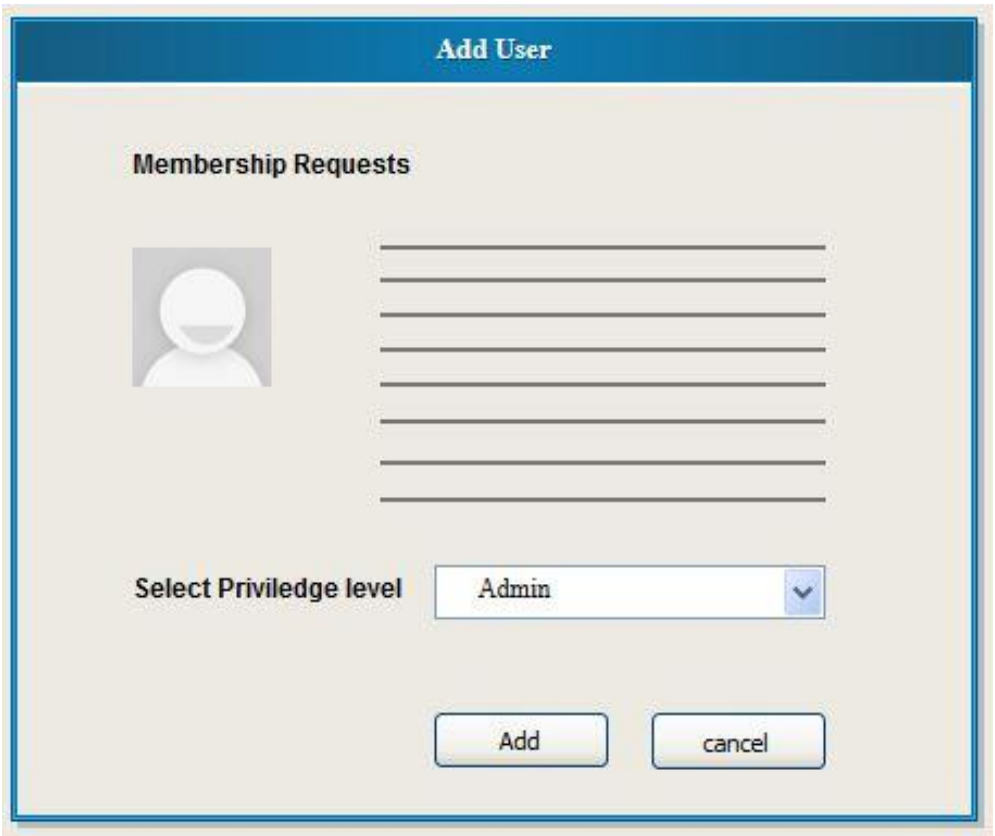
22. Download



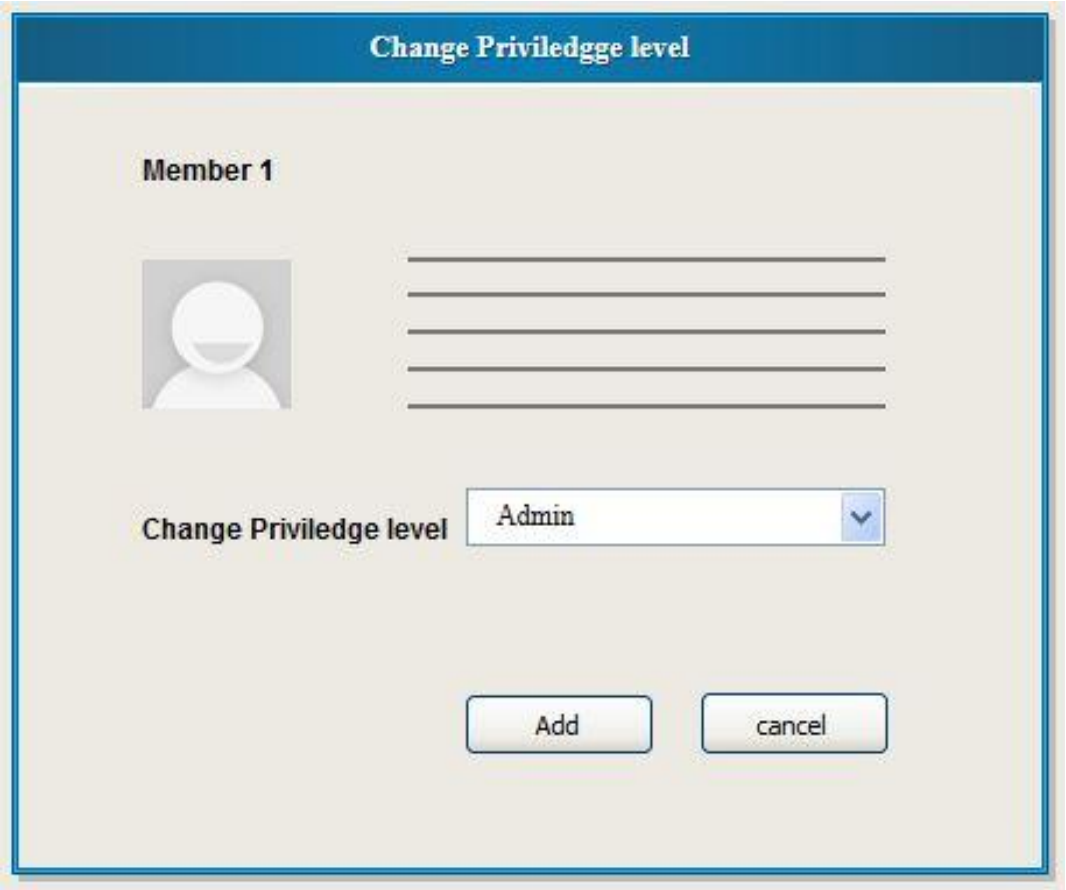
23. Remove User



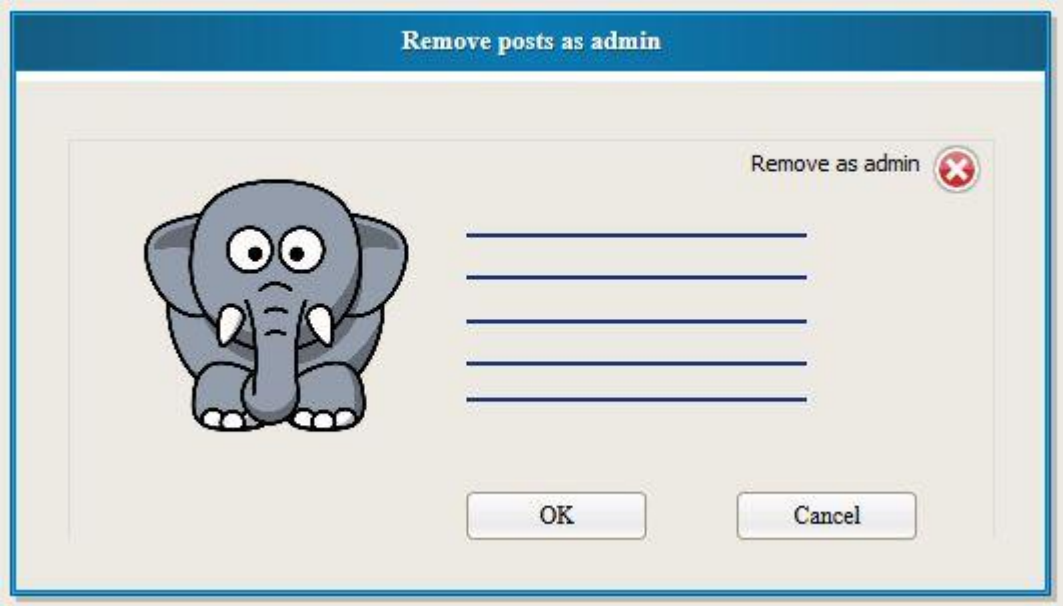
24.Add User



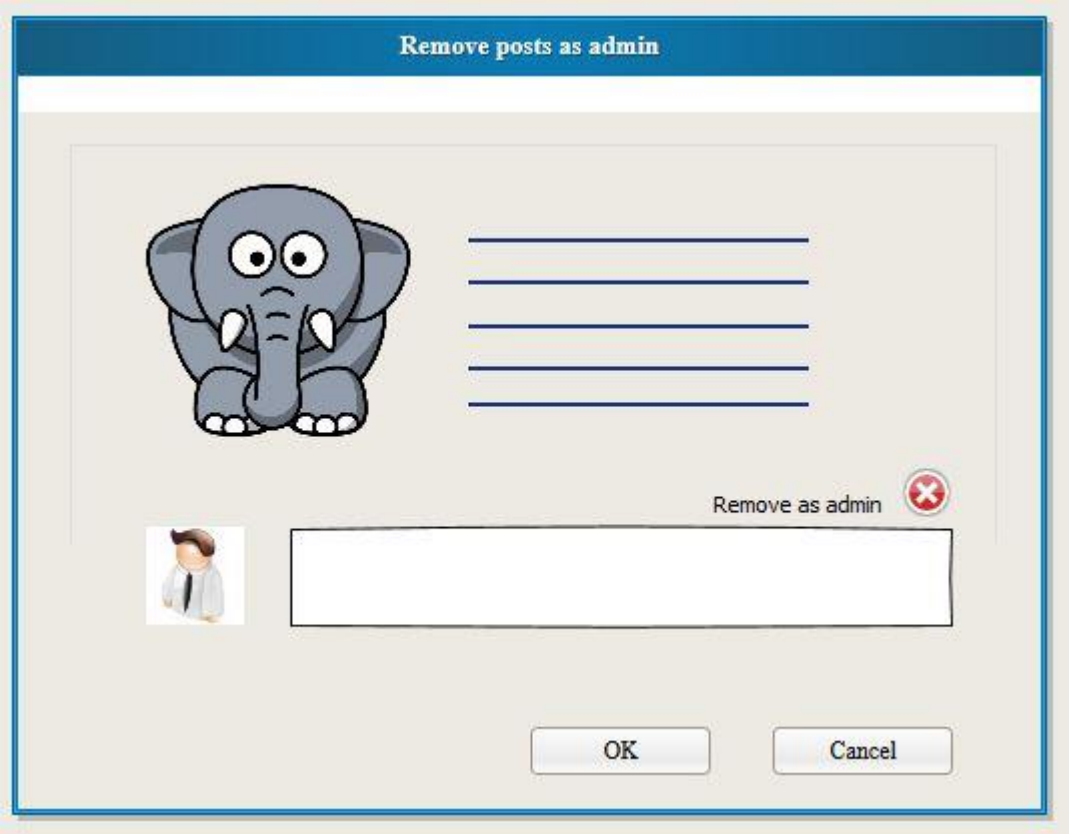
25. Change Priviledge levels by admin



26. Remove posts by admin



27. Edit Comments By admin



Group Members

ICT 09/10/051	D.R.T.Thilanka	2493
ICT 09/10/041	M.R.R.Y.W.Ranathunga	2482
ICT 09/10/049	D.M.M.A.D.Sumanarathna	2491
ICT 09/10/020	M.G.L.L.Jayasekara	2461
ICT 09/10/058	D.M.A.Kotikawatta	2468
ICT 09/10/011	M.G.S.S.Gunarathna	2454