

# **SOCIAL NETWORKING WEBSITE (Friendlink)**



**Developed By:**

**Deeba Mehboob**

Reg. #2010-FGI-606

Roll# 000030

**Javeria Akbar**

Reg. #2008-FGI-591

Roll# 000020

**Supervised by**

**Madam Sumera Qurat-ul-Ain**

Department of Computer Science  
University of the Punjab  
(2015)

## Final Approval

**Dated:** \_\_\_\_\_

It is certified that we have read the project report submitted by **Deeba Mehboob** (Reg.# **2010-FGI-606** , Roll# **000030**) and **Javeria Akbar** (Reg.# **2010-FGI-591** Roll# **000020**) and it is our judgment that this project is of sufficient standard to warrant its acceptance by the IMCG (POST GRADUATE) G-10/4 ISLAMABAD, Islamabad for the **BS Degree in Computer Science**

\_\_\_\_\_  
Project Examiner  
Name: Mrs Rozina Faheem  
Designation: Associate Professor  
Department of computer science,  
IMCG( PG) G-10/4 IBD

\_\_\_\_\_  
Project Supervisor  
Name: Mam.Qurat-ul-ain  
Designation: Lecturer  
Department of computer science,  
IMCG( PG) G-10/4 IBD

*A dissertation submitted to the  
Department of Computer Science,  
University of the Punjab  
as a partial fulfillment of the requirements  
for the award of the degree of  
Bachelors in Computer Science*

## **Dedication**

We dedicate this project to our beloved parents, respected teachers and all those who prayed for our success.

## **Declaration**

We hereby declare that this Software, neither as a whole nor as a part thereof has been copied out from any source. It is further declared that we have developed this Software entirely on the basis of our personal efforts made under the sincere guidance of our teachers and supervisor.

No portion of the work presented in this report has been submitted in support of any application for any other degree or qualification of this or any other university or institute of learning.

**Deeba Mehboob**

Reg. #2010-FGI-606

Roll# 000030

**Javeria Akbar**

Reg. #2010-FGI-591

Roll# 000020

## **Acknowledgement**

All praise to Almighty Allah, who gave us the understanding, courage and patience to complete this project.

Thanks to our parents and all well-wishers, who helped us in our most crucial times and it is due to their untiring efforts that we are at this position today.

We express our gratitude to our kind Teacher Miss Qurut-ul-Ain for providing us opportunity to learn and enhance our knowledge. She had been ready to help and guide us throughout the project in any way possible.

**Deeba Mehboob**

Reg. # 2010-FGI-606

Roll# 000030

**Javeria Akbar**

Reg. # 2010-FGI-591

Roll# 000020

## Project in Brief

<b>Project Title:</b>	SOCIAL NETWORKING WEBSITE (Freindlink)
<b>Objective:</b>	<ul style="list-style-type: none"><li>• User can check the visits on his/ her timeline</li><li>• Profile picture can be set different for friends and anonymous people</li><li>• Timeline can be sorted by name as well as by time</li></ul>
<b>Undertaken By:</b>	Deeba Mehboob Javeria Akbar
<b>Supervised By:</b>	Miss Qurut-ul-Ain
<b>Date Started:</b>	1 <sup>st</sup> December , 2014
<b>Date Completed:</b>	5 <sup>th</sup> March, 2015
<b>Tools Used:</b>	Net Beans IDE 8.0.2 MS Access 2013 MS Word 2013 MS Visio 2013  Language: HTML5, CSS3, JavaScript, Java
<b>System Used:</b>	Core i3, 2 GB RAM

## **Abstract**

People have used the idea of “SOCIAL NETWORKING SITE” loosely for over a century to connect to millions of complex relation between members of social systems at all scales, from interpersonal to international.

Social networking sites are not only for you to communicate or interact with other people globally but, this is also one effective way for business promotion. A lot of business minded people these days are now doing business online and use these social networking sites to respond to customer queries. It isn't just a social media site used to socialize with your friends but also, represents a huge pool of information from day to day living.

A social networking service is an online service, platform, or site that focuses on facilitating the building of social networks or social relations among people who, for example, share interests, activities, backgrounds, or real-life connections. A social network service consists of a representation of each user (often a profile), his/her social links, and a variety of additional services. Most social network services are web-based and provide means for users to interact over the Internet, such as e-mail and instant messaging. Online community services are sometimes considered as a social network service, though in a broader sense, social network service usually means an individual-centered service whereas online community services are group-centered. Social networking sites allow users to share ideas, activities, events, and interests within their individual networks.

Our project aims at using Java and JSP using Net Beans to make our social networking website. We use HTML to design the interface of the project (Presentation layer), Java and JSP to create the business logic and use Microsoft Access 2010 for creating the database design of our project at the back end.



## TABLE OF CONTENTS

<b>CHAPTER 1:INTRODUCTION.....</b>	<b>1</b>
1.1. Introduction .....	2
1.2. Literature review of Social Networking site: .....	3
1.3. Purpose of the SNS: .....	4
1.4. Need of SNS:.....	4
1.5. Drawbacks of the Existing System: .....	4
1.6. Benefits of the proposed system: .....	4
1.6.1. Maintaining different cover photos and profile pictures at a time:.....	4
1.6.2. Email sent in case of forgot password: .....	5
1.6.3. Profile visits: .....	5
1.6.4. By Name sorting: .....	5
1.7. Scope: .....	5
1.8. Definitions, Acronyms and Abbreviations:.....	5
1.9. Environment: .....	6
1.9.1. MS Access 2013: .....	6
1.9.2. Microsoft Word 2013:.....	6
1.9.3. Microsoft Visio 2013: .....	6
1.9.4. Net Beans 8.0.2: .....	6
1.9.5. Internet and Web Browser: .....	6
<b>CHAPTER 2:REQUIREMENT ANALYSIS.....</b>	<b>7</b>
Project Charter .....	8
2.1. Analysis: .....	9
2.2. Problem Analysis: .....	9
2.3. Object Oriented Analysis: .....	9
2.4. Unified Modeling Language: .....	9
Diagrams Overview: .....	10
2.5. Software Process: .....	10
2.5.1. Inception Phase: .....	10
2.5.2. Elaboration Phase: .....	10
2.5.3. Construction Phase: .....	11
2.5.4. Transition Phase:.....	11
2.6. Project idea:.....	11
2.7. Project Goal:.....	11
2.8. Objectives:.....	11
2.8.1. Maintaining different cover photos and profile pictures at a time:.....	11
2.8.2. Email sent in case of forgot password: .....	11
2.8.3. Profile visits: .....	12
2.8.4. By Name sorting: .....	12
2.9. Overall Description: .....	12
2.9.1. Product Perspective:.....	12
2.9.2. Product Functions: .....	12
2.9.3. User characteristics: .....	12
2.9.4. Constraints: .....	12
2.9.5. Assumptions and dependencies: .....	12
2.10. Roles and Responsibilities:.....	13
<b>CHAPTER 3:SYSTEM ANALYSIS .....</b>	<b>14</b>
3.1. System Analysis: .....	15
3.2. Specific Functional Requirements: .....	15

3.2.1.	Account Management Module:.....	15
3.2.2.	Profile Visits Module:.....	15
3.2.3.	Email send Module: .....	15
3.2.4.	Timeline sorting Module: .....	15
3.2.5.	Two different PP'S and CP'S uploading Module:.....	15
3.2.6.	Manage posts Module: .....	15
3.3.	Non Functional Requirements: .....	16
3.3.1.	Performance Requirements .....	16
3.3.2.	Graphical User Interface Requirements .....	16
3.3.3.	Portability.....	16
3.4.	User Characteristics.....	16
<b>CHAPTER 4:SYSTEM DESIGN .....</b>		<b>17</b>
4.1.	System Design Defining: .....	18
4.2.	Database Tables: .....	18
4.2.1.	Table Layout_person .....	19
4.2.2.	Table Layout_Posts.....	19
4.2.3.	Table Layout_like .....	19
4.2.4.	Table Layout_Comments .....	20
4.2.5.	Table Layout_friendRequests .....	20
4.2.6.	Table Layout_profileVisits .....	20
4.2.7.	Table Layout_messages .....	21
4.2.8.	Table Layout_friends.....	21
4.3.	Entity Relationship Diagram: .....	22
<b>CHAPTER 5:OBJECT-ORIENTED ANALYSIS &amp; DESIGN .....</b>		<b>23</b>
5.1	Domain Model: .....	24
5.2.	Use case: .....	25
5.3.	Fully Dressed Use Cases:.....	29
5.3.1.	Sign Up .....	29
5.3.2.	Sign In.....	30
5.3.3.	Sign Out .....	31
5.3.4.	Add Cover and Profile Photo.....	32
5.3.5.	Request Password .....	33
5.3.6.	Delete Account.....	34
5.3.7.	Edit Name .....	35
5.3.8.	Edit Password.....	36
5.3.9.	Edit Cover and DP .....	37
5.3.10.	Edit public cover and DP .....	38
5.3.11.	Add Post .....	39
5.3.12.	Edit Post .....	40
5.3.13.	Delete Post.....	41
5.3.14.	Like Post.....	42
5.3.15.	Comment on Post .....	43
5.3.16.	View Home.....	44
5.3.17.	View My Likes .....	45
5.3.18.	View My Posts .....	46
5.3.19.	View Friend Requests .....	47
5.3.20.	Accept Friend Request .....	48
5.3.21.	Cancel Friend Request.....	49
5.3.22.	Add Friend.....	50
5.3.23.	Delete Friend .....	51

5.3.24.	View Friends.....	52
5.3.25.	View Profile Visits .....	53
5.3.26.	Reset Profile Visits .....	54
5.3.27.	Send Message .....	55
5.3.28.	View Messages .....	56
5.3.29.	Reply to Message.....	57
5.3.30.	Delete Messages .....	58
5.3.31.	Search .....	59
5.3.32.	View Profile .....	60
5.4.	System Sequence Diagrams: .....	61
5.4.1.	Sign up .....	61
5.4.2.	Sign out .....	61
5.4.3.	Sign in .....	62
5.4.4.	Add cover and profile photo .....	62
5.4.5.	Request Password .....	63
5.4.6.	Delete Account.....	63
5.4.7.	Edit Name .....	64
5.4.8.	Edit Password.....	64
5.4.9.	Edit cover and DP .....	64
5.4.10.	Edit public cover and DP .....	65
5.4.11.	Add post.....	65
5.4.12.	Delete post .....	65
5.4.13.	like post.....	66
5.4.14.	Comment on post .....	66
5.4.15.	View my Likes .....	66
5.4.17.	View My Posts.....	67
5.4.18.	View Friend Requests .....	67
5.4.19.	Accept Friend Request.....	67
5.4.20.	Cancel Friend Request .....	68
5.4.21.	Add Friend .....	68
5.4.22.	Delete Friend.....	68
5.4.23.	View profile visits.....	69
5.4.24.	Reset profile visits.....	69
5.4.25.	Send Message.....	69
5.4.26.	Reply to Message.....	70
5.4.27.	Delete Message .....	70
5.4.28.	Search.....	70
5.5.	Class Diagram: .....	71
<b>CHAPTER 6:IMPLEMENTATION .....</b>		<b>72</b>
	DAO Class .....	73
	Add Cover and DP (Private and Public) .....	97
	Uploading Pictures (Cover, DP, Post) .....	98
	Sorting Timelines.....	102
	Profile Visit.....	103
	Sending Email/Requesting Password .....	104
<b>CHAPTER 7:TESTING .....</b>		<b>107</b>
7.1.	Testing:.....	108
7.2.	Testing Objectives:.....	108
7.3.	Types of Testing:.....	108
7.3.1.	Blackbox Testing .....	108

7.3.2. Whitebox Testing.....	108
7.4. Test Cases: .....	109
7.4.1. Sign Up .....	109
7.4.2. Sign In.....	110
7.4.3. Sign Out .....	111
7.4.4. Request Password .....	112
7.4.5. Delete Account.....	113
7.4.7. Edit Password.....	114
7.4.7. Add Post.....	115
7.4.8. Edit Post .....	116
7.4.9. Delete Post .....	117
7.4.10. Comment on a post .....	118
7.4.11. View Friend Requests .....	119
7.4.12. Accept Friend Request .....	120
7.4.13. Accept Friend Request .....	121
7.4.14. Add Friend .....	122
7.4.15. Delete Friend.....	123
7.4.16. Reset Profile Visits .....	124
7.4.17. Send Message.....	125
7.4.18. View Messages .....	126
7.4.19. Reply to messages.....	127
7.4.20. Delete Messages.....	128
7.4.21. Delete Message .....	129
7.4.22. Search.....	130
7.4.23. View Profile .....	131
<b>CHAPTER 8:SCREEN SHOTS .....</b>	<b>132</b>
8.1. Welcome Page.....	133
8.2. Sign In Page .....	133
8.3. Forgot Your Password?.....	133
8.4. Add Cover and profile photo.....	134
8.5. My Posts Page: .....	134
8.6. Search.....	135
8.7. View Profile .....	135
8.8. Messages .....	136
8.9. Replying to messages:.....	136
8.10. Friend Requests: .....	137
8.11. Profile Visits:.....	137
8.12. Friends: .....	138
8.13. Home .....	138
8.14. Edit Profile: .....	139
8.15. Commenting on a post:.....	139
<b>CHAPTER 9:CONCLUSION.....</b>	<b>140</b>
9. Conclusion: .....	141
9.1. Benefits.....	141
9.2. Features.....	141
<b>REFERENCES.....</b>	<b>142</b>

# **CHAPTER 1**

## **INTRODUCTION**

## 1.1. Introduction

Social Networking - It's the way the 21st century communicates now. Social networking is the grouping of individuals into specific groups, like small rural communities or a neighborhood subdivision. Although social networking is possible in person, especially in the workplace, universities, and high schools, it is most popular online. This is because unlike most high schools, colleges, or workplaces, the internet is filled with millions of individuals who are looking to meet other people.

Social network is the mapping and measuring of relationships and flows between people, groups, organizations, computers, URLs, and other connected information/knowledge entities. The nodes in the network are the people and groups while the links show relationships or flows between the nodes. Social network provides both a visual and a mathematical analysis of human relationships.

Social Networking Website project itself is a huge project comprising various features like profile updating, friend's list organization and various other applications to enhance the overall look and feel of the website. However, in this project we are basically working on four essential features or modules (Public and Private DP and Cover Photo, Password Sending to Mail, Timeline Visits & Sorting of Timeline)

- **Public and Private DP & Cover Photo:**

This module helps the user's to set different profile photo and cover photo for their friend's in the friend list & at the same time they can select and show different profile picture and cover photo to other persons.

- **Password Sending To Mail:**

This module helps the user to retrieve his password through mail in case of forgot password.

- **Sorting Of Timeline:**

This module helps the user to sort his/her timeline by time as well as by name.

- **Profile Visits:**

This module helps the user to view who visits his timeline in a day along with name and time and date.

Profiles and Friends lists are two key features on social network sites. The third is a public commenting feature ('Testimonials', 'Comments', 'The Wall'). This feature allows individuals to comment on their Friends' profiles. These comments are displayed prominently and visible for anyone who has access to that profile Social networking sites are not only for you to communicate or interact with other people globally but, this is also one effective way for business promotion. A lot of business minded people these days are now doing business online and use these social networking sites to respond to customer queries. It isn't just a social media site used to socialize with your friends but also, represents a huge pool of information from day to day living.

A social networking service is an online service, platform, or site that focuses on facilitating the building of social networks or social relations among people who, for example, share interests, activities, backgrounds, or real-life connections. A social network service consists of a representation of each user (often a profile), his/her social links, and a variety of additional services. Most social network services are web-based and provide means for users to interact over the Internet, such as e-mail and instant messaging. Online community services are sometimes considered as a social network service, though in a broader sense, social network service usually means an individual-centered service whereas online community services are group-centered. Social networking sites allow users to share ideas, activities, events, and interests within their individual networks.

## **1.2. Literature review of Social Networking site:**

As of May 2013, almost three quarters (72%) of online U.S. adults use social networking sites, up from 67% in late 2012. When we first started asking about social networking sites in February 2005, just 8% of online adults said they used social networking sites.

Today, social networking site use is a major activity for internet users from a wide range of demographic groups. Younger adults are especially avid adopters, but social networking continues to grow in popularity for older adults as well. Six out of ten internet users ages 50-64 are social networking site users, as are 43% of those ages 65 and older. Although online seniors are less likely than other age groups to use social networking sites, adoption rates for those 65 and older have tripled in the last four years (from 13% in the spring of 2009 to 43% now).

The main types of social networking services are those that contain category places (such as former school year or classmates), means to connect with friends (usually with self-description pages), and a recommendation system linked to trust. Popular methods now combine many of Facebook, Google+, YouTube, LinkedIn, Instagram, Pinterest, Tumblr and Twitter widely used worldwide; Nexopia in Canada; Badoo, Bebo, VKontakte (Russia), Delphi (also called Delphi Forums), Draugiem.lv (mostly in Latvia), Hi5 (Europe), Hyves (mostly in The Netherlands), iWiW (mostly in Hungary), Nasza-Klasa, Soup (mostly in Poland), Glocals in Switzerland, Skyrock, The Sphere, StudiVZ (mostly in Germany), Tagged, Tuenti (mostly in Spain), and XING in parts of Europe; Hi5 and Orkut in South America and Central America; Mxit in Africa; and Cyworld, Mixi, Orkut, renren, weibo and Wretch in Asia and the Pacific Islands.

Many of these early communities focused on bringing people together to interact with each other through chat rooms, and encouraged users to share personal information and ideas via personal web pages by providing easy-to-use publishing tools and free or inexpensive web space. Some communities - such as Classmates.com - took a different approach by simply having people link to each other via email addresses. In the late 1990s, user profiles became a central feature of social networking sites, allowing users to compile lists of "friends" and search for other users with similar interests. New social networking methods were developed by the end of the 1990s, and many sites began to develop more advanced features for users to find and manage friends. This newer generation of social networking sites began to flourish with the emergence of SixDegrees.com in 1997 followed by Make out club in 2000, Hub Culture and Friendster in 2002 and soon became part of the Internet

mainstream. Friendster was followed by MySpace and LinkedIn a year later, and eventually Bebo. Friendster became very popular in the Pacific Island. Orkut became the first social networking in Brazil and then also grow fast in India (Madhavan, 2007). Attesting to the rapid increase in social networking sites' popularity, by 2005, it was reported that MySpace was getting more page views than Google. Facebook, launched in 2004, became the largest social networking site in the world in early 2009. Facebook was first introduced (in 2004) as a Harvard social networking (Cassidy, 2006).

### **1.3. Purpose of the SNS:**

The main purpose of the social networking site is that it facilitates the users in a number of ways. It provides the extra functionalities of profile visits, sorting of timeline, at a time different Public and private cover photos and the retrieval of password through the mail which are not provided by most of the social networking sites like Twitter, Facebook, LinkedIn etc.

### **1.4. Need of SNS:**

As the technology goes vaster day by day the risk of data loss and mismanagement is also growing vigorously. Hacking is becoming common and most of the people don't feel safe to share their valuable information like pictures, videos etc. also some new features should be introduced for the easiness of the users like profile visits so one can check who visits his/ her timeline in a day. However there are a number of facts that arises the need of the SNS so that one application should be made that must be updated according to the needs of the users and provide them with extra features.

### **1.5. Drawbacks of the Existing System:**

- User cannot upload different profile pictures and cover photos at a time
- User cannot view which person has visited his/her timeline.
- Only by time sorting is applicable.
- Password is not sent to user in case of forgetting password.
- Difficult to use interface.

### **1.6. Benefits of the proposed system:**

#### **1.6.1. Maintaining different cover photos and profile pictures at a time:**

SNS facilitates the user's that they can manage two different cover photos and different profile pictures at a same time, One as public and the other as private. User can show different profile picture and cover photo to his friends and different one to the anonymous



### **1.6.2. Email sent in case of forgot password:**

If user forgot his password then he can request for it and a mail is sent to user's id along with his password.

### **1.6.3. Profile visits:**

User can view how many people visits his timeline with profile visits module. User can view the person name along with time and date.

### **1.6.4. By Name sorting:**

By name sorting is also implemented along with the by time sorting. User can view posts in an alphabetical order with by name sorting and according to time in by time sorting. By time sorting is the default one.

## **1.7. Scope:**

SNS is a web application which is easy to run. It has a distinctive and user friendly GUI (graphical user interface).

- Access all the time.
- Provides login facility to the users.
- Provides the user with the option to check profile visits. User can also manage different profile pictures and cover photos at a time.
- Allows the user to access the system 24 hours a day and throughout the year.
- Allows the User to add the profile details, add/delete posts, likes and maintain them.
- Allows new users to avail account to become a user of SNS.

The features that are described above can be implemented. The features fulfill the requirements of all the users. The success criterion for the system is based in the level up to which the features described in this document are implemented in the system.

## **1.8. Definitions, Acronyms and Abbreviations:**

SNS	Social Networking Website
MAD	Microsoft Access Database
JSP	Java Server Pages
UML	Unified Modeling Language
PP	Profile Picture
CP	Cover Photo
IMCG	Islamabad Model College For Girls

## **1.9. Environment:**

### **1.9.1. MS Access 2013:**

MS Access is used to create the database. A table is a collection of related data entries.

### **1.9.2. Microsoft Word 2013:**

For documenting the project, we used the MS-Word 2013.

### **1.9.3. Microsoft Visio 2013:**

Microsoft Visio is a 2D-object drawing application and is part of the Microsoft Office suite. We used this for drawing Use-cases, Domain Model, and Sequence Diagram etc.

### **1.9.4. Net Beans 8.0.2:**

Net Beans is an integrated development environment (IDE) for developing primarily with Java, but also with other languages, in particular PHP, C/C++, and HTML5. It is also an application platform framework for Java desktop applications and others.

The Net Beans IDE is written in Java and can run on Windows, OS X, Linux, Solaris and other platforms supporting a compatible JVM .The Net Beans Platform allows applications to be developed from a set of modular software components called modules. Applications based on the Net Beans Platform (including the Net Beans IDE itself) can be extended by third party developers. The Net Beans Team actively supports the product and seek feature suggestions from the wider community. Every release is preceded by a time for Community testing and feedback.

### **1.9.5. Internet and Web Browser:**

Friendlink can be accessed through a web browser (Google Chrome, Internet Explorer etc.)

# **CHAPTER 2**

## **REQUIREMENT ANALYSIS**

## **Project Charter**

### **Project Title:**

Social Networking Site (friendlink)

### **Project Start Date:**

1<sup>st</sup> December, 2014

### **Project Finish Date:**

5<sup>th</sup> March 2015

### **Project Manager:**

1. Deeba Mehboob

([sohni2414@yahoo.com](mailto:sohni2414@yahoo.com))

2. Javeria Akbar

( [jakbar9220@yahoo.com](mailto:jakbar9220@yahoo.com) )

## 2.1. Analysis:

The word is from the ancient Greek ἀνάλυσις (analysis, "a breaking up", from ana- "up, throughout" and lysis "a loosening") Analysis is the process of breaking a complex topic or substance into smaller parts to gain a better understanding of it.

Analysis emphasizes an investigation of the problem and requirements, rather than a solution. Analysis is a broad term, best qualified, as in requirements analysis (an investigation of the requirements) or object oriented analysis (an investigation of the domain objects) Analysis can be summarized in one phrase “do the right things”

## 2.2. Problem Analysis:

Problem analysis is the process of understanding real world problems, user’s needs and proposing solutions to meet those needs. The goals of problem analysis are to gain better understanding of a problem being solved, before development begins.

## 2.3. Object Oriented Analysis:

During Object Oriented Analysis there is an emphasis on finding and describing the objects — or concepts — in the problem domain.

## 2.4. Unified Modeling Language:

“The Unified Modeling Language is a visual language for specifying, visualizing, modifying, constructing and documenting the artifacts of an object-oriented software-intensive system under development.”

The word visual in the definition is a key point – the UML is the de-facto standard diagramming notation for drawing or presenting pictures (with some text) related to software—primarily object oriented software.

Unified Modeling Language (UML) is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created, by the Object Management Group. It was first added to the list of OMG adopted technologies in 1997, and has since become the industry standard for modeling software-intensive systems

UML includes a set of graphic notation techniques to create visual models of object-oriented software-intensive systems.

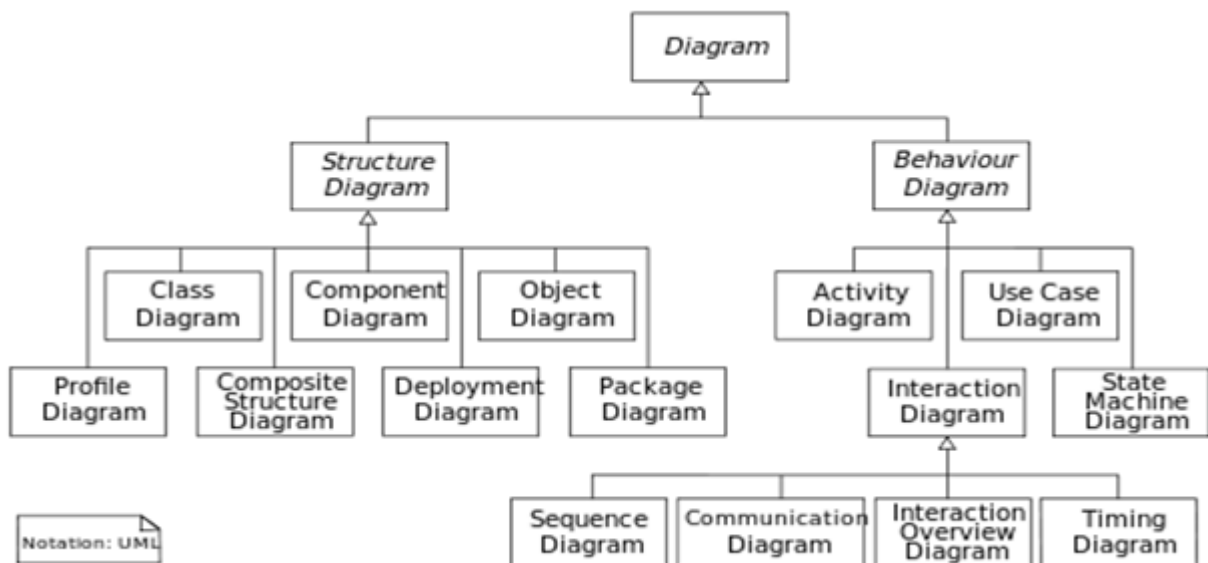
UML offers a standard way to visualize a system's architectural blueprints, including elements such as: - Activities, Actors, Business Processes, Database schemas, (logical) Components, programming language statements, Reusable software components

UML combines techniques from data modeling (entity relationship diagrams), business modeling (work flows), object modeling, and component modeling. It can be used with all processes, throughout the software, and across different implementation technologies. UML has synthesized the notations of the Booch method, the Object-modeling technique (OMT) and Object-oriented software engineering (OOSE) by fusing them into a single, common and widely usable modeling language. UML aims to be a standard modeling language which can

model concurrent and distributed systems. UML is a de facto industry standard, and is evolving under the control of the Object Management Group (OMG).

UML models may be automatically transformed to other representations (e.g. Java) by means of QVT-like transformation languages. UML is extensible, with two mechanisms for customization: profiles and stereotypes.

## Diagrams Overview:



## 2.5. Software Process:

The software process chosen is RUP (Rational Unified Process). The RUP is a process framework from IBM Rational. Its goal is to ensure the production of high quality software that meets the needs of its end-users, within a predictable schedule and budget. The RUP lifecycle organizes the tasks into phases and iterations. According to RUP lifecycle project has four phases:

### 2.5.1. Inception Phase:

During inception phase problem is identified and analyzed and a solution is proposed for solving the problem in the form of a proposed system. In Inception phase, we have identified the problem and proposed a solution for it. Vision document and software requirement specification were made at the end of this phase.

### 2.5.2. Elaboration Phase:

The elaboration phase is where the project starts to take shape. In this phase the problem domain analysis is done and the architecture of the project gets its basic form. We modeled the actors and their goals identified in inception phase in a use-case model to identify the use

cases of SNS. Use case descriptions are also developed during this phase. A description of the software architecture was finalized. A development plan was made for the overall project. System design and database design has been made in this phase. For system design UML diagram, system sequence diagrams, interaction diagrams were made. And for database design database schema was made and entity relationship diagram was made.

### **2.5.3. Construction Phase:**

System has been implemented in this phase. Testing is also done in this phase. For testing purpose we developed test cases.

### **2.5.4. Transition Phase:**

Transition phase involves the deployment and product services of the system developed in construction phase

## **2.6. Project idea:**

SNS is a web application which is easy to run. It has a distinctive and user friendly GUI (graphical user interface). SNS is accessible all the time. Provides login facility to the users. Provides the user with the option to check profile visits. User can also manage different profile pictures and cover photos at a time .Allows the user to access the system 24 hours a day and throughout the year.

Allows new users to avail account to become a user of SNS. User provides the detail to system and system perform the functionalities and generate result. User simply sign in and can be add, view, search and update the posts, comments pictures.

## **2.7. Project Goal:**

The main goal is to provide a system that performs several functions that existing systems don't. It performs profile visits, by time sorting, email sending, different PP and CP.

## **2.8. Objectives:**

### **2.8.1. Maintaining different cover photos and profile pictures at a time:**

SNS facilitates the user's that they can manage two different cover photos and different profile pictures at a same time, One as public and the other as private. User can show different profile picture and cover photo to his friends and different one to the anonymous.

### **2.8.2. Email sent in case of forgot password:**

If user forgot his password then he can request for it and a mail is sent to user's id along with his password.

### **2.8.3. Profile visits:**

User can view how many people's visits his time line with profile visits module. User can view the person name along with time and date.

### **2.8.4. By Name sorting:**

By name sorting is also implemented along with the by time sorting. User can view posts in an alphabetical order with by name sorting and according to time in by time sorting. By time sorting is the default one.

## **2.9. Overall Description:**

### **2.9.1. Product Perspective:**

Social networking site is designed for many users to share posts, comments and much more. User first signs in and then it can be add, view, update or delete the record at every time. The complete overview of the system is as shown in the overview diagram above:

- The product to be developed has interactions with the users, Administrator and others.

### **2.9.2. Product Functions:**

SNS performs the following functions. The SNS provides different type of services based on the type of users

- The user should be provided with the updated information about the user and friends.
- Provisions for the users to add, update, view and search their own information.
- The user is provided with interfaces to add/delete the record and information.

### **2.9.3. User characteristics:**

The users of the SNS are professionals, students and many other. The users are assumed to have basic knowledge of the computers and social networking site. The administrators of the system must have more knowledge of the internals of the system and is able to rectify the small problems that may arise due to disk crashes, power failures and other catastrophes to maintain the system. The proper user interface, user's manual, help and the guide to use and maintain the system must be sufficient to educate the users on how to use the system without any problems.

### **2.9.4. Constraints:**

- The information of all the users must be stored in a database that is accessible by the SNS.
- The user information system must be compatible with the applications system.
- The SNS is runnable in any computer and is running all 24 hours day.
- The users must have their correct usernames and passwords to access SNS facilities.

### **2.9.5. Assumptions and dependencies:**

- The users have sufficient knowledge of computers and Social networking site.
- The users know the English language, as the user interface will be provided in English



- The product must be installed in system.

## 2.10. Roles and Responsibilities:

The following table will show the roles and responsibilities of the team members in order to complete the project on specified time.

Role	Responsibility
<b>Project Manager</b> Deeba Mehboob Javeria Akbar	Project Manager is the leader of the team. She will define all the roles and responsibilities of the team members. She assigns the jobs to the team members and make sure that all of them are doing their work in a proper way.
<b>Requirement Specialist</b> Deeba Mehboob Javeria Akbar	Requirement Specialist defines all the requirements of all the clients. She interacts with the clients and understands what he exactly wants to make. She gather the entire functional requirement, through any mean of requirement gathering technique or through prototype.
<b>Designer</b> Deeba Mehboob Javeria Akbar	Designer specifies the requirement described by the Requirement Specialist in a diagrammatical shape. So that all the artifacts and documents can easily be managed that can be needed in elaboration phase.
<b>Database Designer</b> Deeba Mehboob Javeria Akbar	Database Designer is responsible to manage all the data in the database. She manages the data in the form of tables inside the Database.
<b>Architect</b> Deeba Mehboob Javeria Akbar	Architect is responsible for managing all types of artifacts related to the software design. She defines all the classes and interfaces that would be required to implement the business logic, their relation and their responsibilities in the project.
<b>Programmer</b> Deeba Mehboob Javeria Akbar	Programmer is responsible for translating all the design into executable format.
<b>Tester</b> Deeba Mehboob Javeria Akbar	Tester, tests the implemented modules by the programmer. She makes sure that the programmer has developed the correct software. She establishes the test standards to which the system should confirm for acceptance.

# **CHAPTER 3**

## **SYSTEM ANALYSIS**

### **3.1. System Analysis:**

System Analysis is a software engineering task that bridges the gap between system level requirements engineering and software design.

### **3.2. Specific Functional Requirements:**

Functional requirements of a system define the internal workings of the software. The functional requirements of SNS are as follows:

#### **3.2.1. Account Management Module:**

##### **3.2.1.1. Sign Up**

- This module asks the new users to sign up first to use the SNS

##### **3.2.1.2. Login**

- This module provides the facility of login to the system.
- User provides the ID and password.
- System verify the ID and Password, if ID and Password is valid then user login successfully otherwise shows the error message.

##### **3.2.1.3. Sign out**

- This module sign out the user account.
- Current application closed.

#### **3.2.2. Profile Visits Module:**

**User Interface**– This module helps the users to see his/ her time line visit in a day along with the user name date and time.

#### **3.2.3. Email send Module:**

System will send an email to user's email id along with his password selected from the database in case of forgot password

#### **3.2.4. Timeline sorting Module:**

After successfully insertion of the required data and uploading of pictures, comments, likes user can sort his / her timeline By Name of the user's .By name sorting displays all the posts according to alphabetical order.

#### **3.2.5. Two different PP'S and CP'S uploading Module:**

The main task of this particular component is to provide the facility to user that he/she can select two different PP'S and CP'S at a same time. One PP and CP is shown to friends and the other one is shown

#### **3.2.6. Manage posts Module:**

- **Add posts:** this will help the user to add new posts
- **View posts:** this helps the user to view posts
- **Edit posts:** this helps the user to edit posts
- **Delete posts:** this helps the user to delete the posts

### **3.3. Non Functional Requirements:**

#### **3.3.1. Performance Requirements**

##### **3.3.1.1. Response Time**

- Less than 10 seconds for all non-database related functions.
- Database dependent for all database related functions.

##### **3.3.1.2. Startup Time**

- Startup time will be less than 25 seconds

##### **3.3.1.3. Shut down Time**

Shut down time will be less than 25 seconds.

#### **3.3.2. Graphical User Interface Requirements**

User interface will be consistent.

#### **3.3.3. Portability**

##### **3.3.3.1. Machine**

The minimum recommended computer hardware for the system should be:

- Pentium IV
- 512 MB RAM
- Standard I/O device
- Microsoft Windows XP

##### **3.3.3.2. Language**

The software will support English language only.

##### **3.3.3.3. Usability**

The software shall provide user friendly interfaces so that the user is able to use the system. He/she should just have basic knowledge of computer for using the system.

### **3.4. User Characteristics**

User of the software is the any person around the world including students, doctors, engineers, and every person related to every field

# **CHAPTER 4**

## **SYSTEM DESIGN**

## 4.1. System Design Defining:

Systems design is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirement. Systems design could see it as the application of systems theory to product development. It implies a systematic and rigorous approach to design—an approach demanded by the scale and complexity of many systems problems. There is some overlap with the disciplines of systems analysis, systems architecture and systems engineering. In database design, design of the database is made. For that table layout is shown and Entity Relationship diagram is made. In table layout table name, its primary key and purpose is given along with table definition and table field's description.

We will describe following in this chapter.

- Database Tables
- Entity relationship Diagram

## 4.2. Database Tables:

Database tables are used to organize and group our data by common characteristics or principles. Our database can contain as many tables as we need to organize our data (limited only by the amount of storage space on our hard disk).

No.	Tables
1	person
2	post
3	like
4	comment
5	friendRequests
6	profileVisits
7	messages
8	friend

#### 4.2.1. Table Layout\_person

<b>Table name</b>	person
<b>Primary key</b>	EmailAddress
<b>Purpose</b>	It stores the user record.

Field name	Data type	Length	Allow nulls
EmailAddress	Short Text	50	No
Password	Short Text	50	No
FirstName	Short Text	50	No
LastName	Short Text	50	No
ProfilePhotoPath	Short Text	50	Yes
CoverPhotoPath	Short Text	50	Yes
PublicProfilePath	Short Text	50	Yes
PublicCoverPath	Short Text	50	Yes

#### 4.2.2. Table Layout\_Posts

<b>Table name</b>	Posts
<b>Primary key</b>	PostID
<b>Purpose</b>	It stores the user posts

Field name	Data type	Length	Allow nulls
Post	Long Text	Maximum	No
EmailAddress	Short Text	50	No
time	Date/Time	General Date	No
PostID	AutoNumber	Long Integer	No
PicPath	Short Text	50	Yes

#### 4.2.3. Table Layout\_like

<b>Table name</b>	like
<b>Primary key</b>	postID, likedby
<b>Purpose</b>	It stores the details of user likes.

Field name	Data type	Length	Allow nulls
postID	Number	Long Integer	No
likedby	Short Text	50	No

#### 4.2.4. Table Layout\_Comments

<b>Table name</b>	Comments
<b>Primary key</b>	PostID, commentId
<b>Purpose</b>	It stores the user's comments

Field name	Data type	Length	Allow nulls
commentId	AutoNumber	Long Integer	No
PostID	Number	Long Integer	No
Comment	Long Text	Maximum	No
Commentby	Short Text	50	No
Time	Date/Time	General Date	No

#### 4.2.5. Table Layout\_friendRequests

<b>Table name</b>	friendRequests
<b>Primary key</b>	requestID, EmailAddress
<b>Purpose</b>	It stores the user's friends requests

Field name	Data type	Length	Allow nulls
EmailAddress	Short Text	50	No
RequestID	Short Text	50	No

#### 4.2.6. Table Layout\_profileVisits

<b>Table name</b>	profileVisits
<b>Primary key</b>	visitID
<b>Purpose</b>	It stores the user's profile visits

Field name	Data type	Length	Allow nulls
visitID	Number	Long Integer	No
EmailAddress	Short Text	Maximum	No
visitedBy	Short Text	50	No
Time	Date/Time	General Date	No



#### 4.2.7. Table Layout\_messages

<b>Table name</b>	messages
<b>Primary key</b>	msgID
<b>Purpose</b>	It stores the user's messages

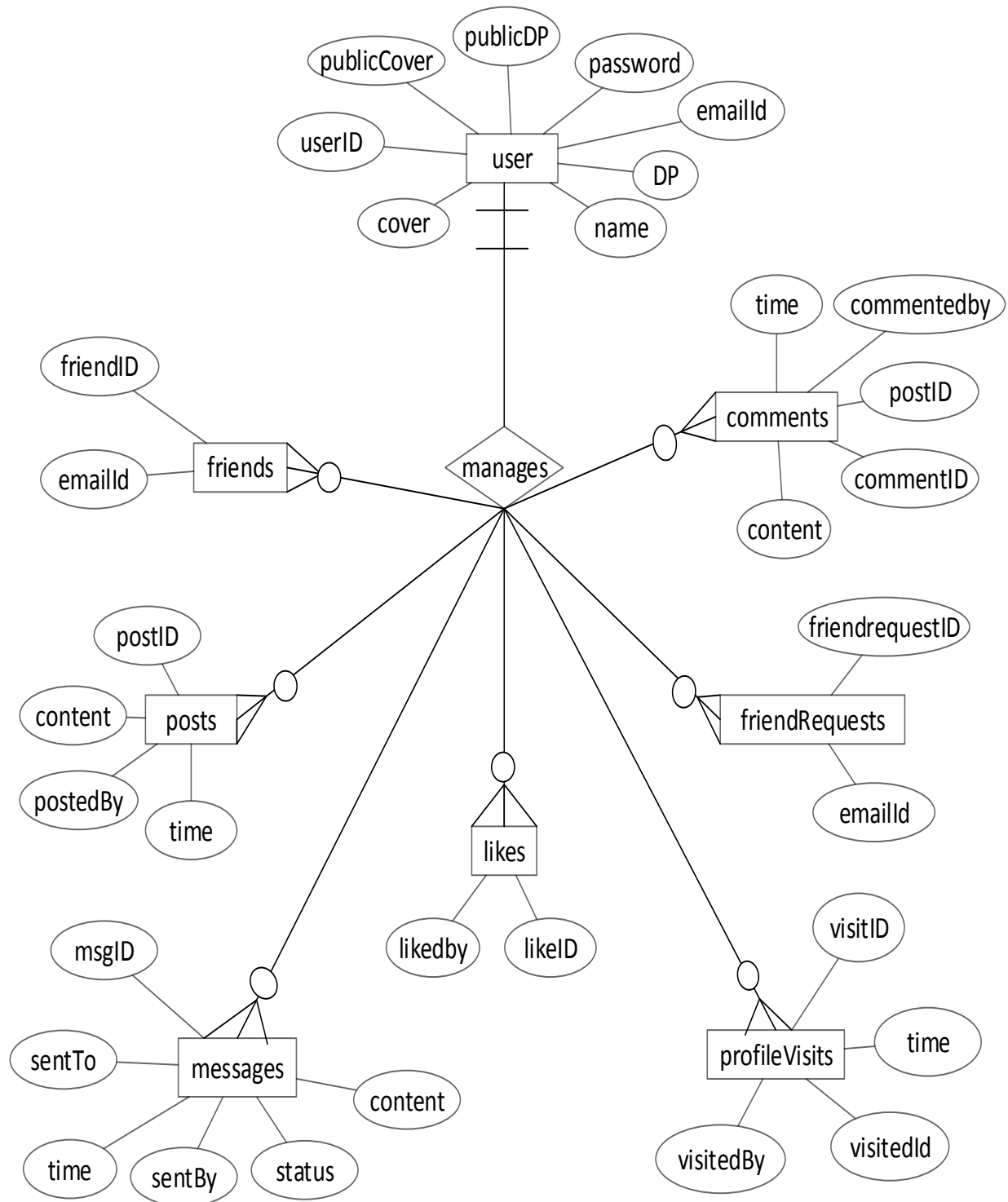
Field name	Data type	Length	Allow nulls
msgID	AutoNumber	Long Integer	No
message	Long Text	Maximum	No
sentTo	Short Text	50	No
sentBy	Short Text	50	No
Time	Date/Time	General Date	No
Status	Yes/No		No

#### 4.2.8. Table Layout\_friends

<b>Table name</b>	friends
<b>Primary key</b>	friendID, friendId
<b>Purpose</b>	It stores the user's friends

Field name	Data type	Length	Allow nulls
EmailAddress	Short Text	50	No
friendID	Short Text	50	No

### 4.3. Entity Relationship Diagram:



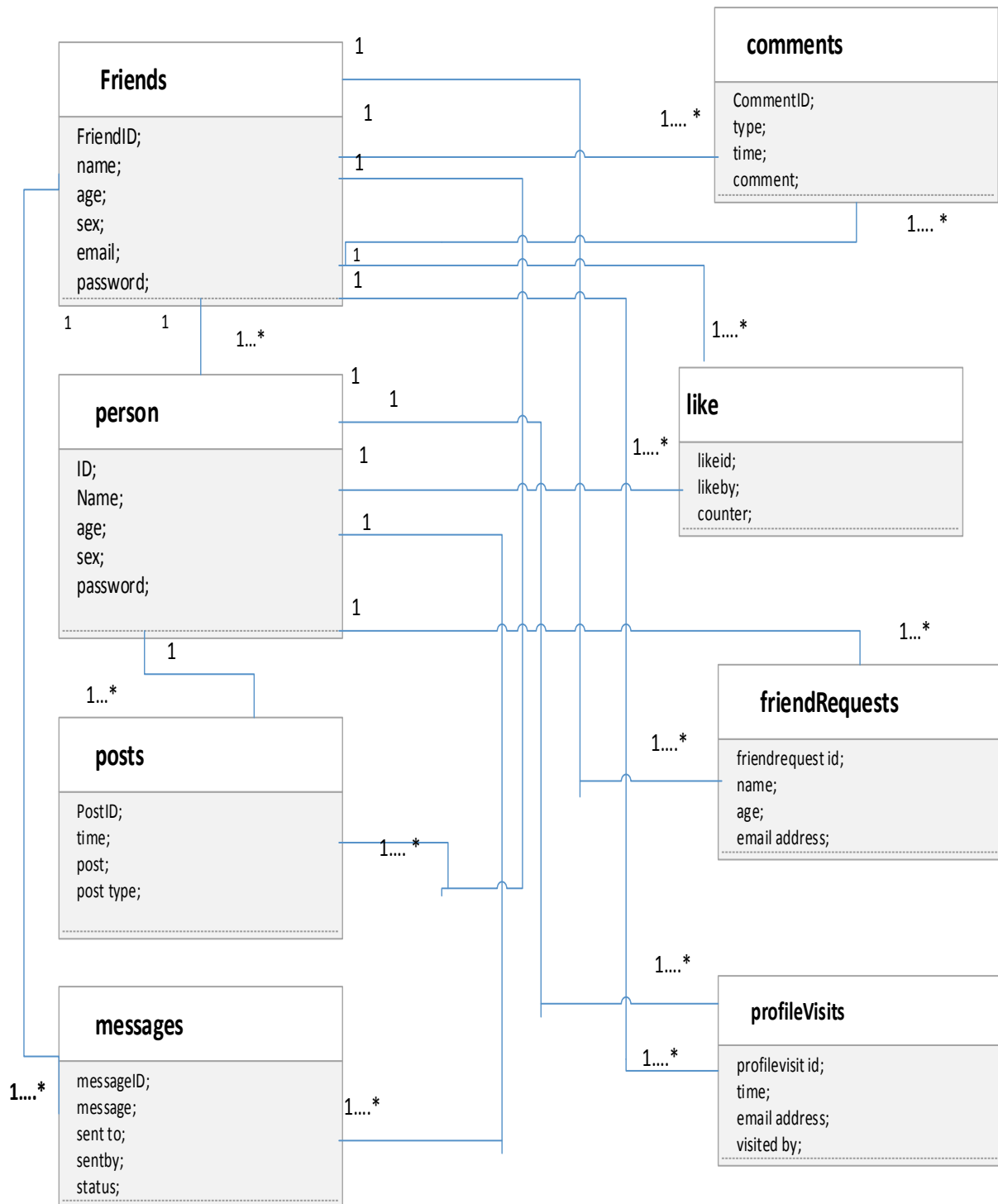
# **CHAPTER 5**

## **OBJECT-ORIENTED**

## **ANALYSIS & DESIGN**

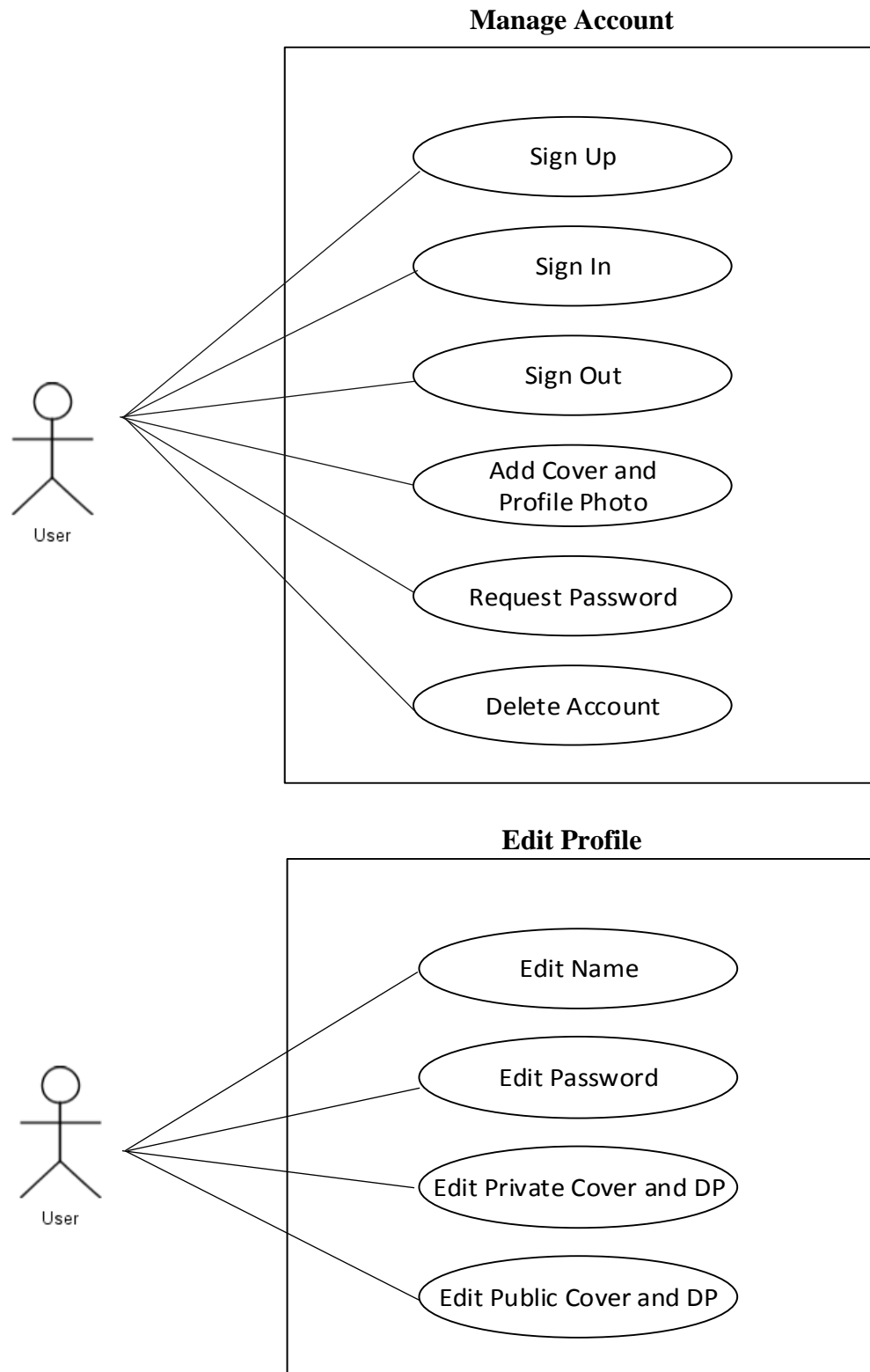
## 5.1 Domain Model:

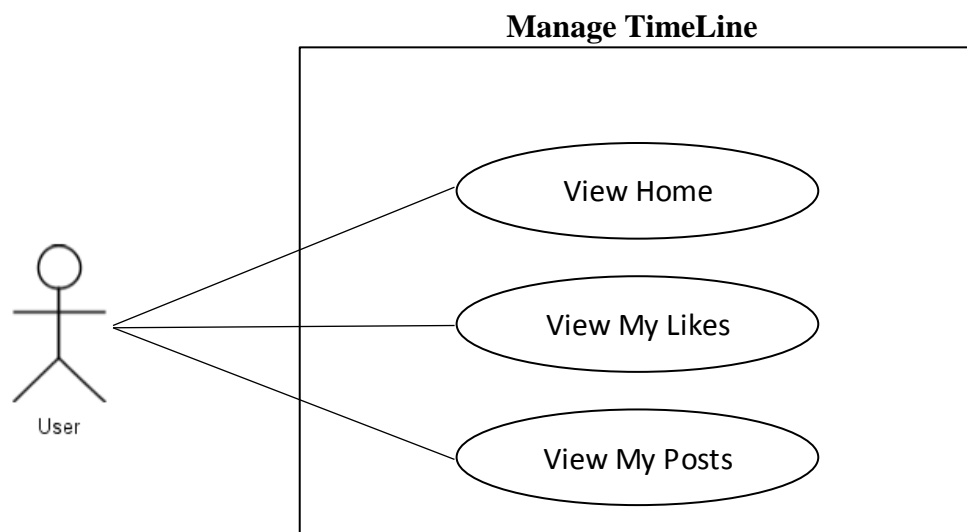
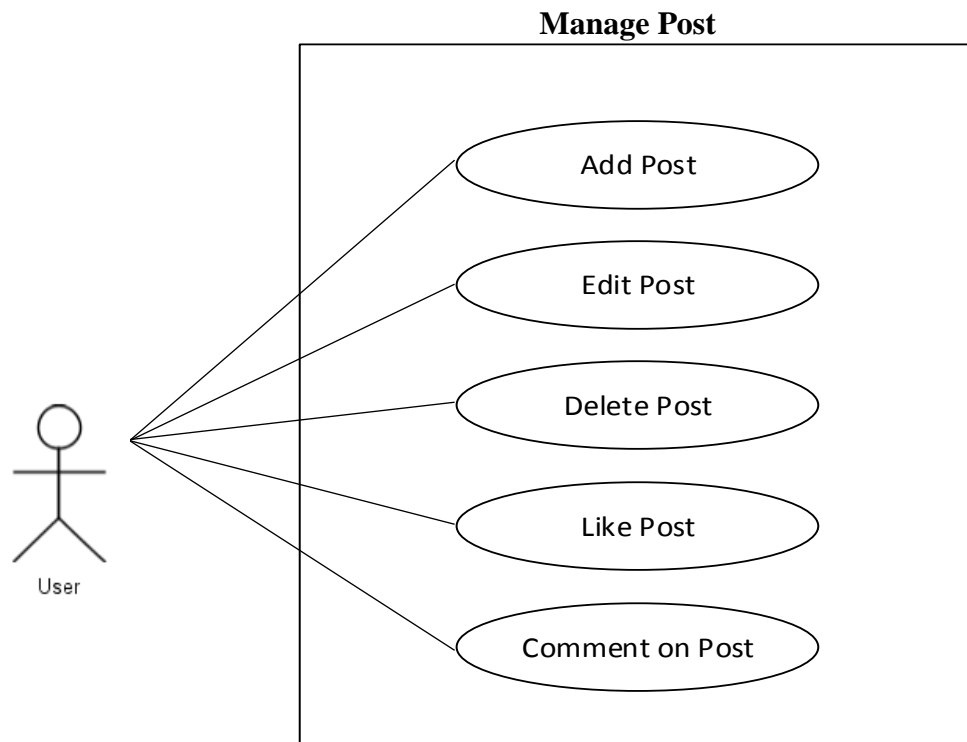
A Domain Model is an object model of a problem domain. Elements of a domain model are Domain Object classes, and the relationships between them.

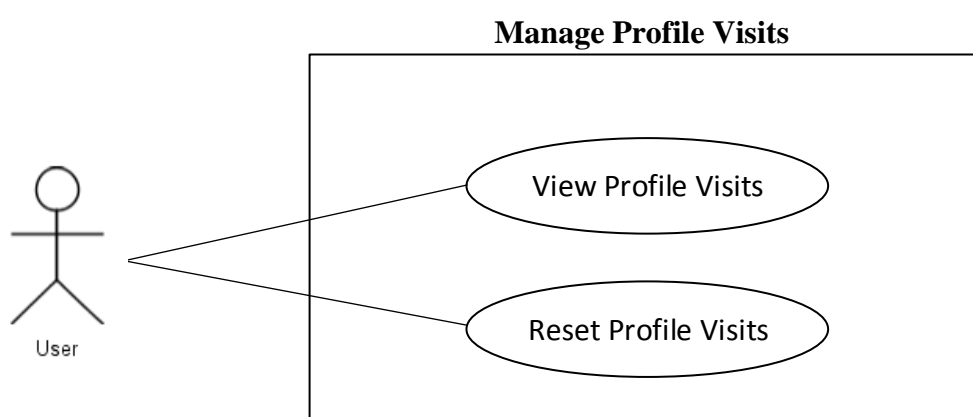
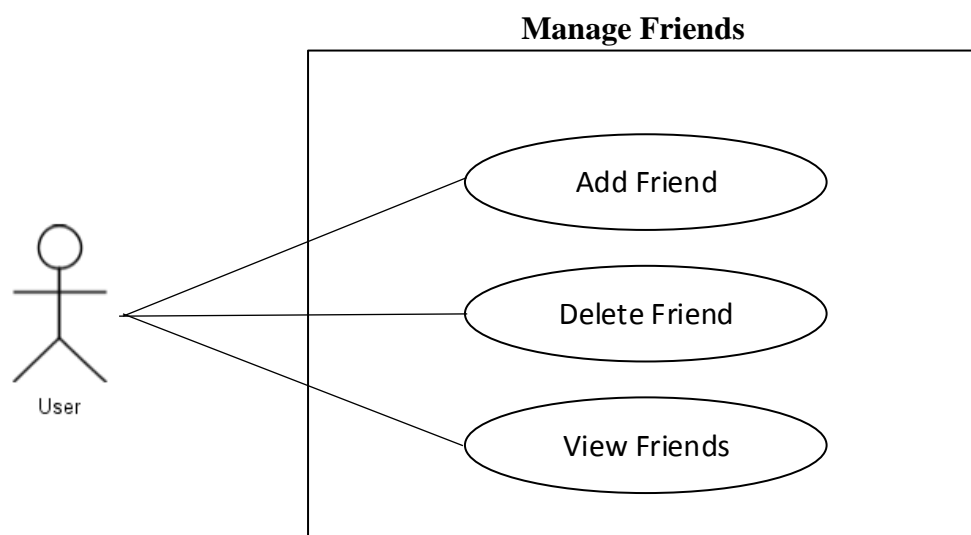
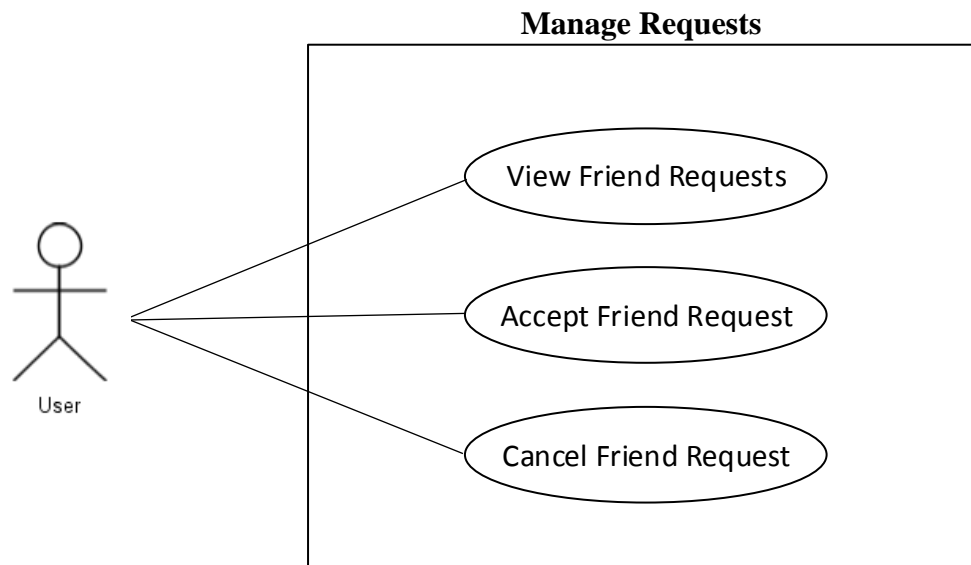


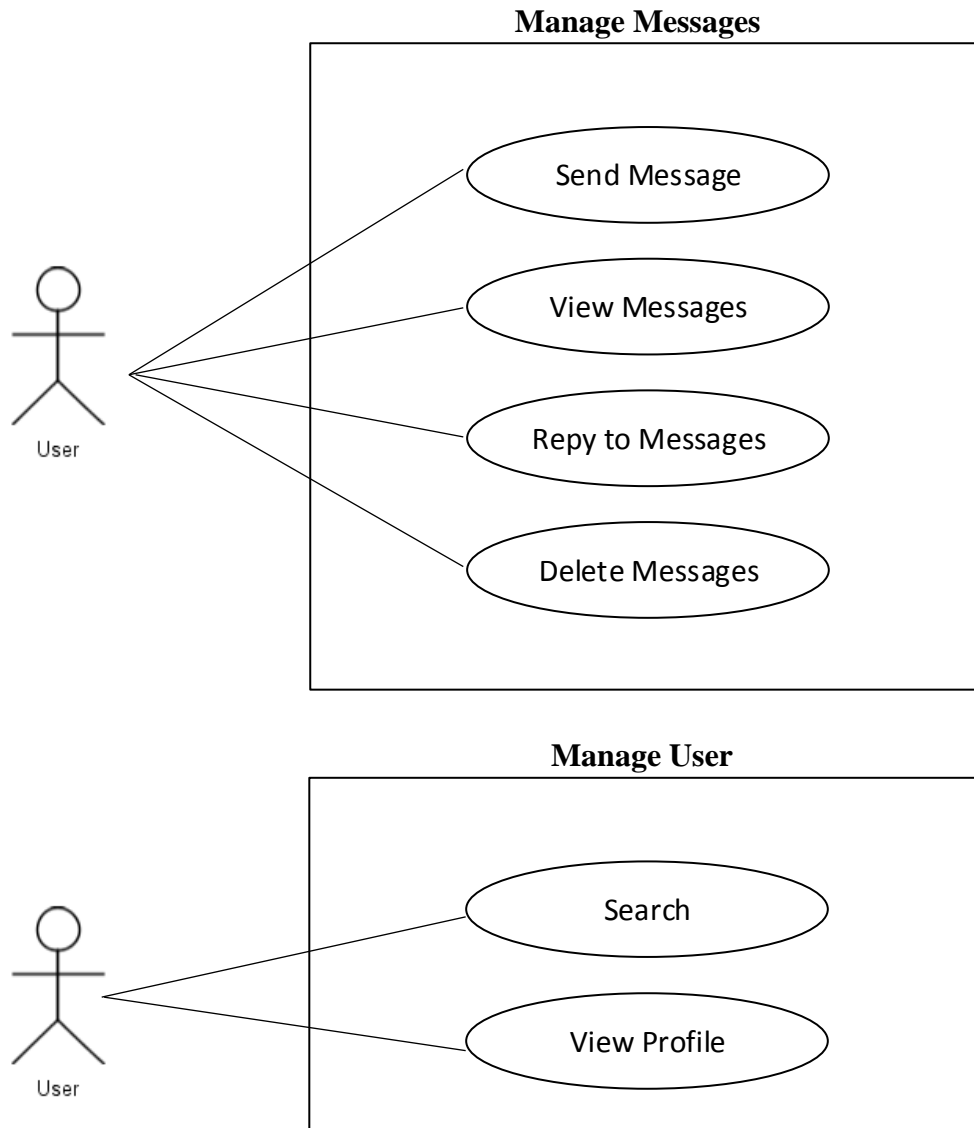
## 5.2. Use case:

A use case is a collection of related success and failure scenarios that describes an actor using a system to support a goal.











## 5.3. Fully Dressed Use Cases:

### 5.3.1. Sign\_Up

- **Use Case ID:** UC-001
- **Use Case Name:** Sign Up
- **Use Case Description:** This use case deals with the user creating an account.
- **Primary Actor:** User
- **Stake holders and interests:**
  - User: to create an account.
- **Pre-Condition:** User should have opened the site.
- **Main Success Scenario:**

User Action	System Response
1. User opens site.	
	2. System shows Sign up and Sign In form to user.
3. User fills in Sign up form.	
4. User clicks on Join now button.	
	5. System verifies email and password constraints and redirects user to the Add cover and Profile Photo page.

- **Post-Condition:** User's account created successfully.
- **Alternative Flows:**
  - 5(a):** If user has entered invalid Email Id.  
System displays error message of invalid email id.
  - 5(b):** If user has entered short password.  
System displays error message of short password.
  - 5(c):** If user has entered email id which is associated with another account.  
System displays error message and shows Sign in form.
- **Frequency:** High
- **Priority:** High
- **Cross-Reference:** SRS-1.1

### 5.3.2. Sign In

- **Use Case ID:** UC-002
- **Use Case Name:** Sign In
- **Use Case Description:** This use case deals with the user logging into the site.
- **Primary Actor:** User
- **Stake holders and interests:**
  - User: to log into the site.
- **Pre-Condition:** User must have an account, valid username and password.
- **Main Success Scenario:**

User Action	System Response
1. User opens site.	
	2. System shows Sign up and Sign in form to the user.
3. User fills in Sign in form.	
4. User clicks on Sign in button.	
	5. System verifies email address and password and redirects user to Home.

- **Post-Condition:** User successfully logged into the site and is on home page.
- **Alternative Flows:**
  - 5(a): If user has entered wrong email address and/or password.  
System displays error message and sign in form.
- **Frequency:** High
- **Priority:** High
- **Cross-Reference:** SRS-1.2

### 5.3.3. Sign Out

- **Use Case ID:** UC-003
- **Use Case Name:** Sign Out
- **Use Case Description:** This use case deals with the user signing out the site.
- **Primary Actor:** User
- **Stake holders and interests:**
  - User: to logout account.
- **Pre-Condition:** User should be logged in.
- **Main Success Scenario:**

User Action	System Response
1. User clicks on Sign out button.	
	2. System logs out user and shows sign in and sign up form.

- **Post-Condition:** User successfully logged out from the site and is on welcome page.
- **Alternative Flows:**
- **Frequency:** High
- **Priority:** High
- **Cross-Reference:** SRS-1.3

### 5.3.4. Add Cover and Profile Photo

- **Use Case ID:** UC-004
- **Use Case Name:** Add Cover and Profile Photo
- **Use Case Description:** This use case deals with adding cover and profile photos.
- **Primary Actor:** User
- **Stake holders and interests:**
  - User: to add cover and profile photo.
- **Pre-Condition:** User should have signed up and is on add cover and profile photo page.
- **Main Success Scenario:**

User Action	System Response
	1. System shows files uploading form to user.
2. User chooses photos and clicks on upload button one by one.	
	3. System upload images and shows them to user.
4. User clicks on Next button.	
	5. System redirects user to his timeline.

- **Post-Condition:** User successfully logged into the site and is on his timeline page.
- **Alternative Flows:**
  - 2(a): If user has clicked on skip button.  
System redirects user to his timeline.
  - 2(b): If user chooses cover not satisfying constraints.  
System informs user to choose cover with following constraints.
- **Frequency:** High
- **Priority:** High
- **Cross-Reference:** SRS-1.4

### 5.3.5. Request Password

- **Use Case ID:** UC-005
- **Use Case Name:** Request Password
- **Use Case Description:** This use case deals with reminding forgotten password.
- **Primary Actor:** User
- **Stake holders and interests:**
  - User: to request forgotten password.
- **Pre-Condition:** User should have an account.
- **Main Success Scenario:**

User Action	System Response
1. User clicks on forgot password link.	
	2. System displays forgot password form to user.
3. User enters his email address and clicks on request button.	
	4. System sends an email containing password to entered email address and informs user.

- **Post-Condition:** Password reminded successfully.
- **Alternative Flows:**
  - 3(a): If user has clicked on cancel button.  
System redirects user to welcome page.
  - 3(b): If user enters unregistered email id.  
System displays error message with forgot password form.
- **Frequency:** High
- **Priority:** High
- **Cross-Reference:** SRS-1.5

### 5.3.6. Delete Account

- **Use Case ID:** UC-006
- **Use Case Name:** Delete Account
- **Use Case Description:** This use case deals with deleting account.
- **Primary Actor:** User
- **Stake holders and interests:**
  - User: to delete his account
- **Pre-Condition:** User should be logged in and is on edit profile page.
- **Main Success Scenario:**

User Action	System Response
1. User clicks on delete account link.	
	2. System asks user for confirmation.
3. User confirms deletion of account.	
	4. System deletes user account, informs user and redirects user to welcome page.

- **Post-Condition:** User account deleted successfully.
- **Alternative Flows:**
  - 3(a): If user has clicked on cancel button.  
System doesn't delete account
- **Frequency:** High
- **Priority:** High
- **Cross-Reference:** SRS-1.6

### 5.3.7. Edit Name

- **Use Case ID:** UC-007
- **Use Case Name:** Edit Name
- **Use Case Description:** This use case deals with changing name.
- **Primary Actor:** User
- **Stake holders and interests:**
  - User: to change his name.
- **Pre-Condition:** User should be logged in and is on edit profile page.
- **Main Success Scenario:**

User Action	System Response
	1. System displays edit profile form to user.
2. User changes name in change name form and clicks on save changes button.	
	3. System save changes and shows confirmation message.

- **Post-Condition:** Name successfully changed.
- **Alternative Flows:**
  - 2(a): If user has clicked on cancel button.  
System redirects user to his timeline.
- **Frequency:** High
- **Priority:** High
- **Cross-Reference:** SRS-1.7

### 5.3.8. Edit Password

- **Use Case ID:** UC-008
- **Use Case Name:** Edit Password
- **Use Case Description:** This use case deals with changing password.
- **Primary Actor:** User
- **Stake holders and interests:**
  - User: to change his password.
- **Pre-Condition:** User should be logged in and is on edit profile page
- **Main Success Scenario:**

User Action	System Response
	1. System displays edit profile forms to user.
2. User fills in change password form and clicks on save password button.	
	3. System saves new password and shows confirmation message.

- **Post-Condition:** Password changed successfully.
- **Alternative Flows:**
  - 2(a): If user has clicked on cancel button.  
System redirects user to his timeline.
  - 2(b): If user enters wrong old password.  
System displays error message.
  - 2(c): If user enters short new password.  
System displays error message.
  - 2(d): If user enters unmatched new passwords.  
System displays error message.
- **Frequency:** High
- **Priority:** High
- **Cross-Reference:** SRS-1.8



### 5.3.9. Edit Cover and DP

- **Use Case ID:** UC-009
- **Use Case Name:** Edit Cover and DP
- **Use Case Description:** This use case deals with changing cover and profile photo.
- **Primary Actor:** User
- **Stake holders and interests:**
  - User: to change his cover and profile photo
- **Pre-Condition:** User should be logged in and is on edit profile page
- **Main Success Scenario:**

User Action	System Response
	1. System displays change button on both cover and profile photo.
2. User chooses new photo and clicks on change button.	
	3. System saves new image and shows it to user.

- **Post-Condition:** Cover and/or profile photo changed successfully.
- **Alternative Flows:**
  - 2(a): If user has clicked on cancel button.  
System redirects user to his timeline.
  - 2(b): If user chooses cover not satisfying constraints.  
System informs user to choose cover with following constraints.
- **Frequency:** High
- **Priority:** High
- **Cross-Reference:** SRS-1.9

### 5.3.10. Edit public cover and DP

- **Use Case ID:** UC-010
- **Use Case Name:** Edit public cover and DP
- **Use Case Description:** This use case deals with changing public cover and profile photo.
- **Primary Actor:** User
- **Stake holders and interests:**
  - User: to change his public cover and profile photo
- **Pre-Condition:** User should be logged in and is on edit profile page
- **Main Success Scenario:**

User Action	System Response
	1. System displays edit profile forms to user.
2. User chooses new photo in change public cover and/or change public DP form and clicks on edit button.	
	3. System saves new image and shows it to user.

- **Post-Condition:** Public cover and/or profile photo changed successfully.
- **Alternative Flows:**
  - 2(a):** If user has clicked on cancel button.  
System redirects user to his timeline.
  - 2(b):** If user chooses cover not satisfying constraints.  
System informs user to choose cover with following constraints.
- **Frequency:** High
- **Priority:** High
- **Cross-Reference:** SRS-1.10

### 5.3.11. Add Post

- **Use Case ID:** UC-011
- **Use Case Name:** Add Post
- **Use Case Description:** This use case deals with adding new post.
- **Primary Actor:** User
- **Stake holders and interests:**
  - User: to add new post.
- **Pre-Condition:** User should be logged in and is on home/my-posts page.
- **Main Success Scenario:**

User Action	System Response
	1. System displays post form to user.
2. User fills in form and clicks on post button.	
	3. System saves new post.

- **Post-Condition:** New post added successfully.
- **Alternative Flows:**
  - 2(a): If user uploads image.  
System displays uploaded image.
  - 2(a)(i): If user clicks on close button.  
System removes image.
- **Frequency:** High
- **Priority:** High
- **Cross-Reference:** SRS-1.11

### 5.3.12. Edit Post

- **Use Case ID:** UC-012
- **Use Case Name:** Edit Post
- **Use Case Description:** This use case deals with editing a post.
- **Primary Actor:** User
- **Stake holders and interests:**
  - User: to edit a post.
- **Pre-Condition:** User should be logged in and is on his posts page.
- **Main Success Scenario:**

User Action	System Response
1. User clicks on edit button of a post which he wants to edit.	
	2. System shows edit post form to user.
3. User fills in form and clicks on edit post button.	
	4. System save changes and redirects user to his posts.

- **Post-Condition:** Post edited successfully.
- **Alternative Flows:**
  - 3(a): If user goes back.  
System don't save changes.
- **Frequency:** High
- **Priority:** High
- **Cross-Reference:** SRS-1.12

### 5.3.13. Delete Post

- **Use Case ID:** UC-013
- **Use Case Name:** Delete Post
- **Use Case Description:** This use case deals with deleting a post.
- **Primary Actor:** User
- **Stake holders and interests:**
  - User: to delete a post.
- **Pre-Condition:** User should be logged in and is on his posts page.
- **Main Success Scenario:**

User Action	System Response
1. User clicks on delete button of a post which he wants to delete.	
	2. System deletes post

- **Post-Condition:** Post deleted successfully.
- **Alternative Flows:**
- **Frequency:** High
- **Priority:** High
- **Cross-Reference:** SRS-1.13

### 5.3.14. Like Post

- **Use Case ID:** UC-014
- **Use Case Name:** Like Post
- **Use Case Description:** This use case deals with liking a post.
- **Primary Actor:** User
- **Stake holders and interests:**
  - User: to like a post.
- **Pre-Condition:** User should be logged in and is on my-posts/my-likes/home/view-profile page.
- **Main Success Scenario:**

User Action	System Response
1. User clicks on like button of a post which he wants to like.	
	2. System saves liked post in user's likes.

- **Post-Condition:** Post liked successfully.
- **Alternative Flows:**
  - 2(a): If user clicks on unlike button.  
System removes post from user's likes.
- **Frequency:** High
- **Priority:** High
- **Cross-Reference:** SRS-1.14

### 5.3.15. Comment on Post

- **Use Case ID:** UC-015
- **Use Case Name:** Comment on Post
- **Use Case Description:** This use case deals with commenting on a post.
- **Primary Actor:** User
- **Stake holders and interests:**
  - User: to comment on a post.
- **Pre-Condition:** User should be logged in and is on my-posts/my-likes/home/view-profile page.
- **Main Success Scenario:**

User Action	System Response
1. User clicks on comment button of a post on which he wants to comment.	
	2. System shows comment form and comments on that post to user.
3. User fills in comment form and clicks on comment button.	
	4. System saves comment.

- **Post-Condition:** Commented on post successfully.
- **Alternative Flows:**
  - 3(a): If user goes back.  
System doesn't save comment.
- **Frequency:** High
- **Priority:** High
- **Cross-Reference:** SRS-1.15

### 5.3.16. View Home

- **Use Case ID:** UC-016
- **Use Case Name:** View Home
- **Use Case Description:** This use case deals with viewing home.
- **Primary Actor:** User
- **Stake holders and interests:**
  - User: to view timeline.
- **Pre-Condition:** User should be logged in.
- **Main Success Scenario:**

User Action	System Response
1. User clicks on home link.	
	2. System shows home to user with posts of friends sorted by time.

- **Post-Condition:** Home viewed successfully.
- **Alternative Flows:**
  - 2(a):** If user clicks on sort by name link.  
System shows timeline to user with posts sorted by time.
- **Frequency:** High
- **Priority:** High
- **Cross-Reference:** SRS-1.16



### 5.3.17. View My Likes

- **Use Case ID:** UC-017
- **Use Case Name:** View My Likes
- **Use Case Description:** This use case deals with viewing user's likes.
- **Primary Actor:** User
- **Stake holders and interests:**
  - User: to view his likes.
- **Pre-Condition:** User should be logged in.
- **Main Success Scenario:**

User Action	System Response
1. User clicks on my likes link.	
	2. System shows likes to user with posts sorted by time.

- **Post-Condition:** User's likes viewed successfully.
- **Alternative Flows:**
  - 2(a): If user clicks on sort by name link.  
System shows likes to user with posts sorted by time.
- **Frequency:** High
- **Priority:** High
- **Cross-Reference:** SRS-1.17

### 5.3.18. View My Posts

- **Use Case ID:** UC-018
- **Use Case Name:** View My Posts
- **Use Case Description:** This use case deals with viewing user's posts.
- **Primary Actor:** User
- **Stake holders and interests:**
  - User: to view his posts.
- **Pre-Condition:** User should be logged in.
- **Main Success Scenario:**

User Action	System Response
1. User clicks on my posts link.	
	2. System shows user's posts to him with posts sorted by time.

- **Post-Condition:** Timeline viewed successfully.
- **Alternative Flows:**
- **Frequency:** High
- **Priority:** High
- **Cross-Reference:** SRS-1.18

### 5.3.19. View Friend Requests

- **Use Case ID:** UC-019
- **Use Case Name:** View Friend Requests.
- **Use Case Description:** This use case deals with viewing friend requests.
- **Primary Actor:** User
- **Stake holders and interests:**
  - User: to view friend requests.
- **Pre-Condition:** User should be logged in.
- **Main Success Scenario:**

User Action	System Response
	1. System shows count of un-responded friend requests to user on top.
2. User clicks on friend requests link.	
	3. System displays list of friend requests to user.

- **Post-Condition:** Friend requests viewed successfully.
- **Alternative Flows:**
- **Frequency:** High
- **Priority:** High
- **Cross-Reference:** SRS-1.9

### 5.3.20. Accept Friend Request

- **Use Case ID:** UC-020
- **Use Case Name:** Accept Friend Request.
- **Use Case Description:** This use case deals with accepting a friend request.
- **Primary Actor:** User
- **Stake holders and interests:**
  - User: to accept a friend request.
- **Pre-Condition:** User should be logged in and is on friend requests or view profile page.
- **Main Success Scenario:**

User Action	System Response
	1. System shows list of friend requests to user.
2. User clicks on accept button of friend request he wants to accept.	
	3. System removes that request and add that user in friends.

- **Post-Condition:** Friend request accepted successfully.
- **Alternative Flows:**
  - 2(a): If user clicks on name of user that requested.  
System shows profile of that user.
  - 2(a)(i): If user clicks on accept request button.  
System removes that request and add that user in friends.
- **Frequency:** High
- **Priority:** High
- **Cross-Reference:** SRS-1.20

### 5.3.21. Cancel Friend Request

- **Use Case ID:** UC-021
- **Use Case Name:** Cancel Friend Request.
- **Use Case Description:** This use case deals with canceling a friend request.
- **Primary Actor:** User
- **Stake holders and interests:**
  - User: to cancel a friend request.
- **Pre-Condition:** User should be logged in and is on friend requests or view profile page.
- **Main Success Scenario:**

User Action	System Response
	1. System shows list of friend requests to user.
2. User clicks on cancel button of friend request he wants to cancel.	
	3. System removes that request.

- **Post-Condition:** Friend Request cancelled successfully.
- **Alternative Flows:**
- **Frequency:** High
- **Priority:** High
- **Cross-Reference:** SRS-1.21

### 5.3.22. Add Friend

- **Use Case ID:** UC-022
- **Use Case Name:** Add Friend
- **Use Case Description:** This use case deals with adding a friend.
- **Primary Actor:** User
- **Stake holders and interests:**
  - User: to add a friend.
- **Pre-Condition:** User should be logged in and is on user's profile whom he wants to add.
- **Main Success Scenario:**

User Action	System Response
1. User clicks on add friend button	
	2. System saves request and shows requested button to user.

- **Post-Condition:** Friend request sent successfully.
- **Alternative Flows:**
  - 2(a): If user clicks on requested button.  
System removes request and shows add friend button to user.
- **Frequency:** High
- **Priority:** High
- **Cross-Reference:** SRS-1.22

### 5.3.23. Delete Friend

- **Use Case ID:** UC-023
- **Use Case Name:** Delete Friend
- **Use Case Description:** This use case deals with unfriending a friend.
- **Primary Actor:** User
- **Stake holders and interests:**
  - User: to unfriend a friend.
- **Pre-Condition:** User should be logged in and is on friend's profile whom he wants to unfriend.
- **Main Success Scenario:**

User Action	System Response
1. User clicks on friend button.	
	2. System asks user for confirmation.
3. User confirms unfriending.	
	4. System deletes friend from user's friends and shows add friend button to user.

- **Post-Condition:** Friend unfriended successfully.
- **Alternative Flows:**
- **2(a):** If user clicks on cancel button.  
System doesn't delete friend.
- **Frequency:** High
- **Priority:** High
- **Cross-Reference:** SRS-1.23

### 5.3.24. View Friends

- **Use Case ID:** UC-024
- **Use Case Name:** View Friends
- **Use Case Description:** This use case deals with viewing friends.
- **Primary Actor:** User
- **Stake holders and interests:**
  - User: to view friends.
- **Pre-Condition:** User should be logged in.
- **Main Success Scenario:**

User Action	System Response
1. User clicks on friends link.	
	2. System displays list of user's friends to user.

- **Post-Condition:** Friends viewed successfully.
- **Alternative Flows:**
- **Frequency:** High
- **Priority:** High
- **Cross-Reference:** SRS-1.24



### 5.3.25. View Profile Visits

- **Use Case ID:** UC-025
- **Use Case Name:** View Profile Visits.
- **Use Case Description:** This use case deals with viewing profile visits.
- **Primary Actor:** User
- **Stake holders and interests:**
  - User: to view profile visits.
- **Pre-Condition:** User should be logged in.
- **Main Success Scenario:**

User Action	System Response
	1. System shows count of profile visits to user on top.
2. User clicks on profile visits link.	
	3. System displays list of profile visits to user.

- **Post-Condition:** Profile visits viewed successfully.
- **Alternative Flows:**
- **Frequency:** High
- **Priority:** High
- **Cross-Reference:** SRS-1.25

### 5.3.26. Reset Profile Visits

- **Use Case ID:** UC-026
- **Use Case Name:** Reset Profile Visits.
- **Use Case Description:** This use case deals with resetting profile visits.
- **Primary Actor:** User
- **Stake holders and interests:**
  - User: to reset profile visits.
- **Pre-Condition:** User should be logged in and is on profile visits page.
- **Main Success Scenario:**

User Action	System Response
	1. System shows list of profile visits to user with reset button on top of list.
2. User clicks on reset button.	
	3. System removes all profile visits of user.

- **Post-Condition:** Profile visits cleared successfully.
- **Alternative Flows:**
- **Frequency:** High
- **Priority:** High
- **Cross-Reference:** SRS-1.27

### 5.3.27. Send Message

- **Use Case ID:** UC-027
- **Use Case Name:** Send Message
- **Use Case Description:** This use case deals with sending a message.
- **Primary Actor:** User
- **Stake holders and interests:**
  - User: to send a message.
- **Pre-Condition:** User should be logged in and is on user's profile whom he wants to send message.
- **Main Success Scenario:**

User Action	System Response
1. User clicks on message button	
	2. System displays message form to user.
3. User fills in form and clicks on send button.	
	4. System sends message and shows confirmation message to user.

- **Post-Condition:** Message sent successfully.
- **Alternative Flows:**
  - 3(a): If user clicks on cancel button.  
System hides message form and doesn't send message.
- **Frequency:** High
- **Priority:** High
- **Cross-Reference:** SRS-1.27

### 5.3.28. View Messages

- **Use Case ID:** UC-028
- **Use Case Name:** View Messages
- **Use Case Description:** This use case deals with viewing messages.
- **Primary Actor:** User
- **Stake holders and interests:**
  - User: to view messages.
- **Pre-Condition:** User should be logged in.
- **Main Success Scenario:**

User Action	System Response
	1. System shows count of message senders to user with messages link.
2. User clicks on messages link.	
	3. System shows list of message senders to user.
4. User clicks on view all button to view messages from particular sender.	
	5. System shows messages of that particular sender.

- **Post-Condition:** Messages viewed successfully.
- **Alternative Flows:**
- **Frequency:** High
- **Priority:** High
- **Cross-Reference:** SRS-1.28

### 5.3.29. Reply to Message

- **Use Case ID:** UC-029
- **Use Case Name:** Reply to Message
- **Use Case Description:** This use case deals with replying to message.
- **Primary Actor:** User
- **Stake holders and interests:**
  - User: to reply to message.
- **Pre-Condition:** User should be logged in and has opened messages of that user whom he wants to reply.
- **Main Success Scenario:**

User Action	System Response
	1. System displays reply form.
2. User fills in form and clicks on reply button.	
	3. System sends message.

- **Post-Condition:** Message replied successfully.
- **Alternative Flows:**
- **Frequency:** High
- **Priority:** High
- **Cross-Reference:** SRS-1.29

### 5.3.30. Delete Messages

- **Use Case ID:** UC-030
- **Use Case Name:** Delete Messages
- **Use Case Description:** This use case deals with deleting messages.
- **Primary Actor:** User
- **Stake holders and interests:**
  - User: to delete messages.
- **Pre-Condition:** User should be logged in and has opened messages.
- **Main Success Scenario:**

User Action	System Response
	1. System displays list of message senders to user.
2. User clicks on view all button of that sender whose messages he wants to delete.	
	3. System shows messages of that sender.
4. User clicks delete button on that message which he wants to delete.	
	5. System asks for confirmation.
6. User confirms deletion.	
	7. System deletes that message.

- **Post-Condition:** Messages deleted successfully.
- **Alternative Flows:**
  - 2(a):** If user clicks on delete all button.  
System asks for confirmation.
  - 2(a)(i):** If user confirms deletion  
System deletes all messages.
  - 2(a)(ii):** If user clicks cancel.  
System don't delete messages.
  - 6(a):** If user clicks on cancel button.  
System doesn't delete message.
- **Frequency:** High
- **Priority:** High
- **Cross-Reference:** SRS-1.30

### 5.3.31. Search

- **Use Case ID:** UC-031
- **Use Case Name:** Search
- **Use Case Description:** This use case deals with searching people.
- **Primary Actor:** User
- **Stake holders and interests:**
  - User: to search people.
- **Pre-Condition:** User has opened site.
- **Main Success Scenario:**

User Action	System Response
	1. System displays search form to user.
2. User enters first and last name and clicks on search button.	
	3. System displays all people with entered first name and last name.

- **Post-Condition:** Searched successfully.
- **Alternative Flows:**
  - 2(a): If user enters only first name.  
System displays all people with entered first name.
  - 2(b): If user enters only last name.  
System displays all people with entered last name.
- **Frequency:** High
- **Priority:** High
- **Cross-Reference:** SRS-1.31

### 5.3.32. View Profile

- **Use Case ID:** UC-032
- **Use Case Name:** View Profile
- **Use Case Description:** This use case deals with viewing profile of people.
- **Stake holders and interests:**
  - User: to view profile of people.
- **Pre-Condition:** User has logged in and is on timeline/my-likes/friends/friend-requests/profile-visits/messages/search-results page.
- **Main Success Scenario:**

User Action	System Response
1. User clicks on name of person.	
	2. System displays profile of that person whose name is clicked.

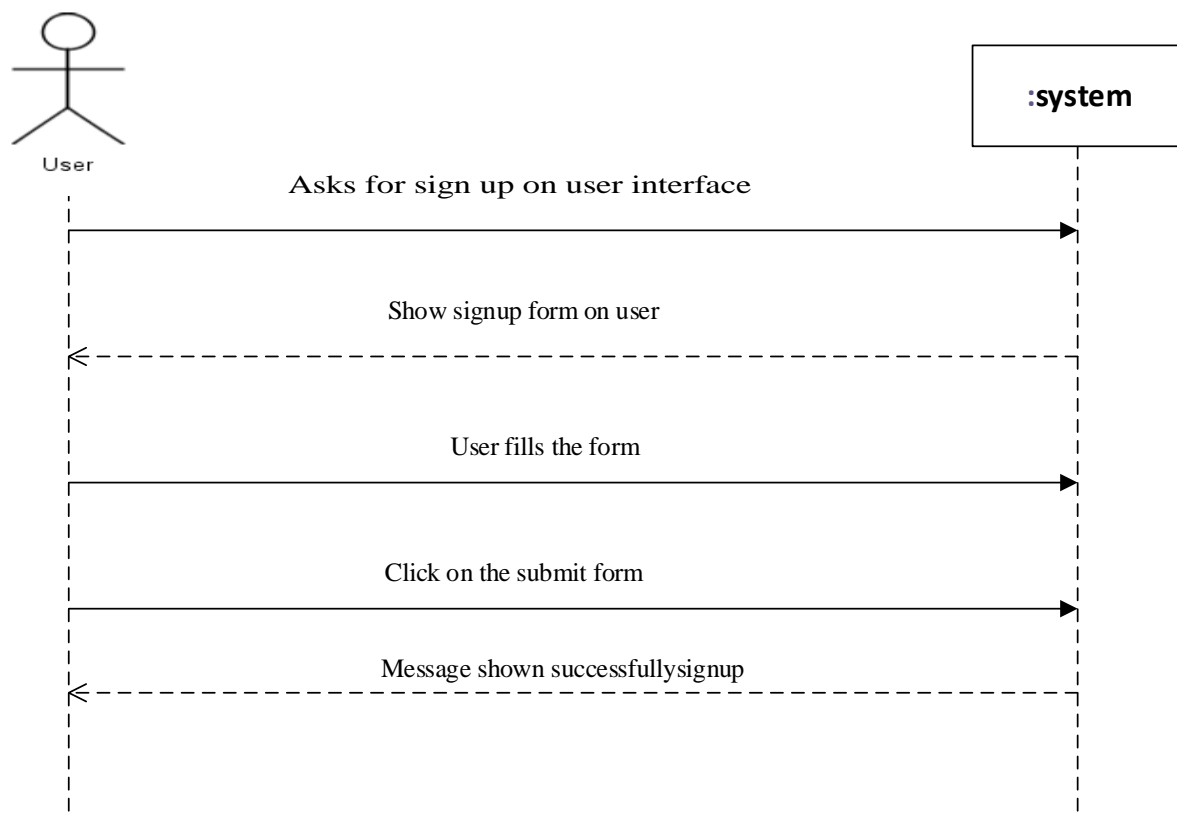
- **Post-Condition:** Profile viewed successfully.
- **Alternative Flows:**
- **Frequency:** High
- **Priority:** High
- **Cross-Reference:** SRS-1.32



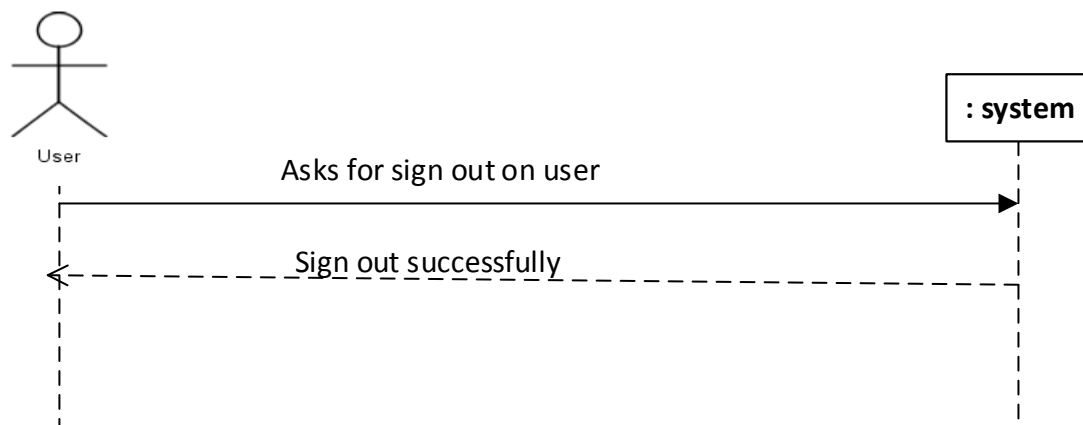
## 5.4. System Sequence Diagrams:

A sequence diagram, in the context of UML, represents object collaboration and is used to define event sequences between objects for a certain outcome. A sequence diagram is an essential component used in processes related to analysis, design and documentation. Sequence Diagrams for the whole System are given below:

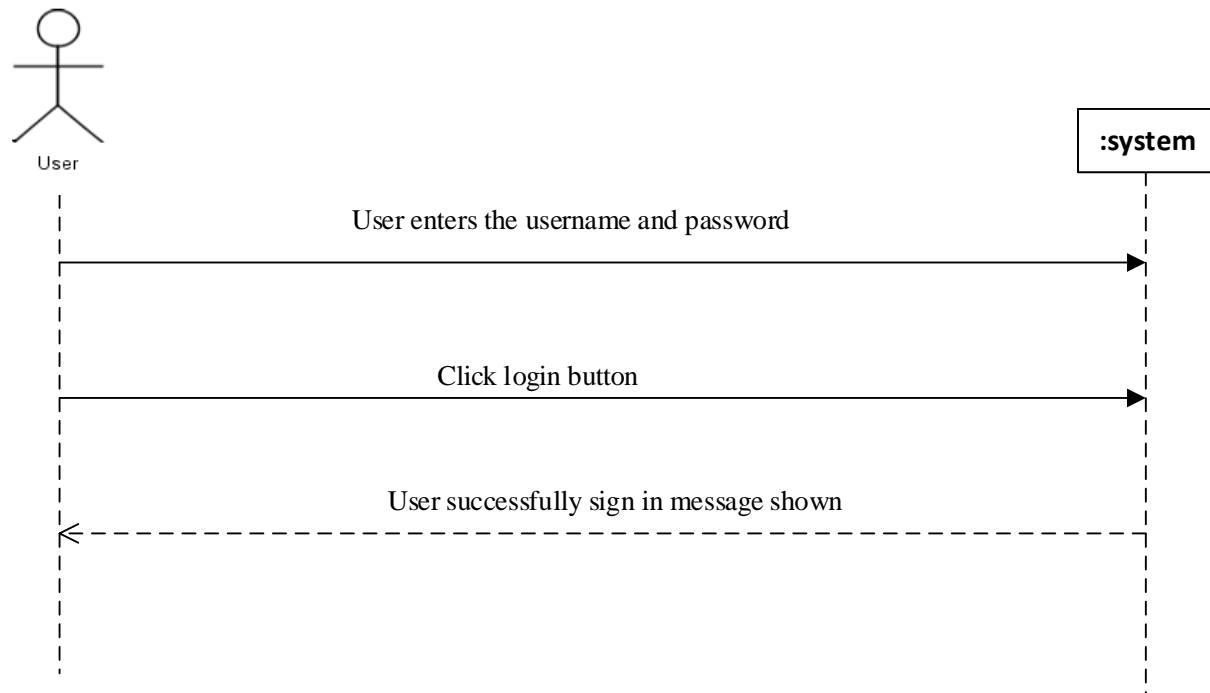
### 5.4.1. Sign up



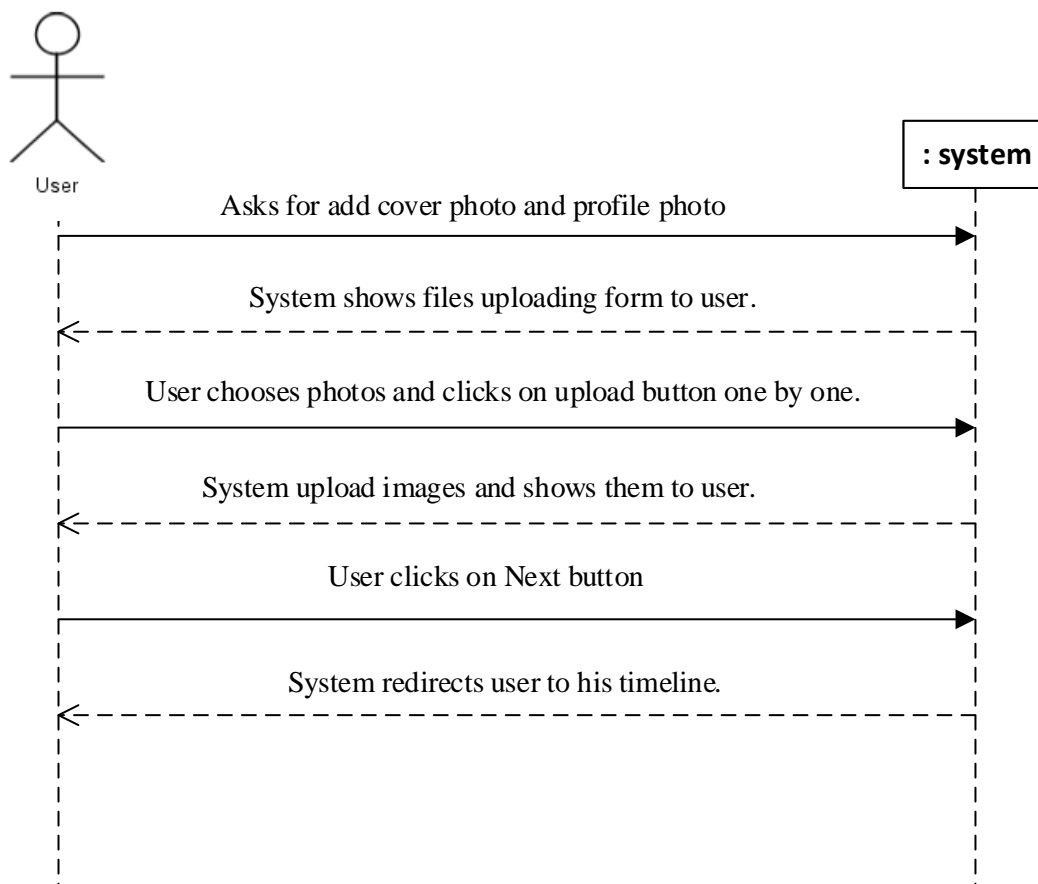
### 5.4.2. Sign out



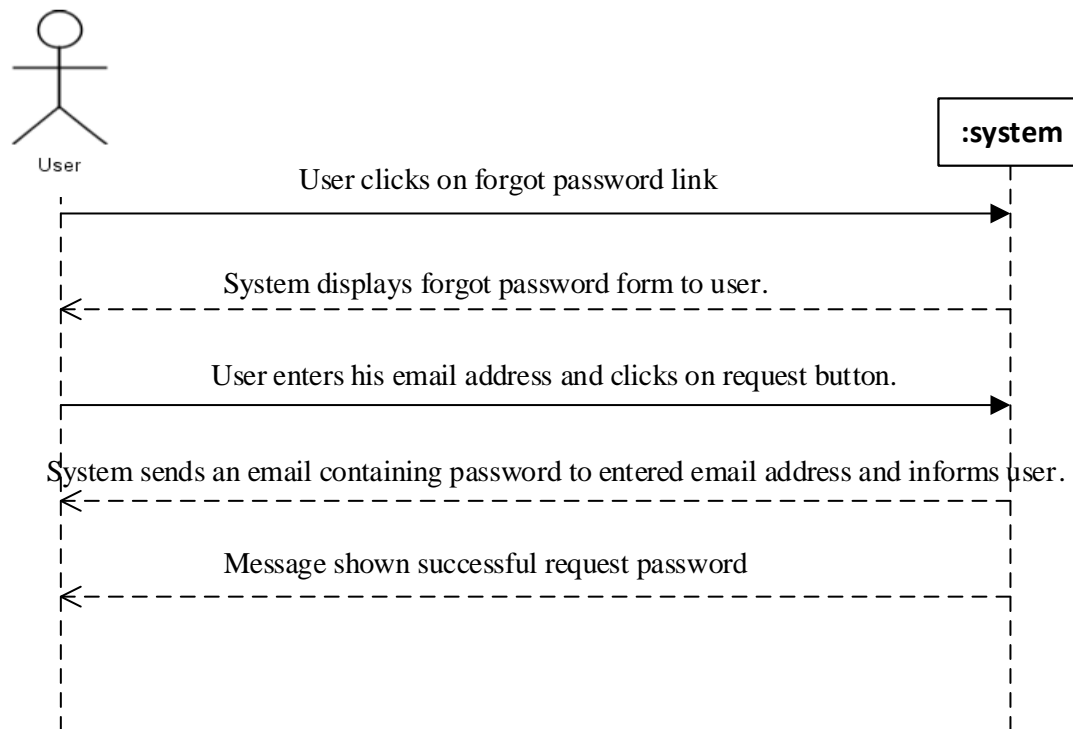
### 5.4.3. Sign in



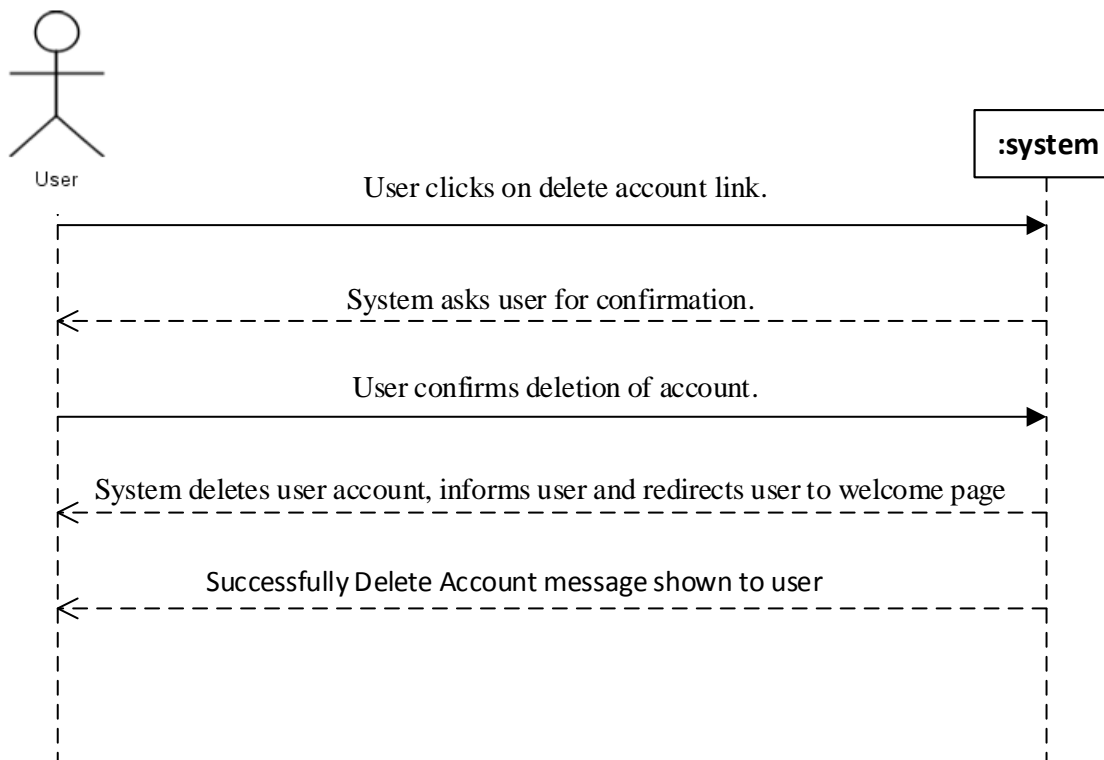
### 5.4.4. Add cover and profile photo



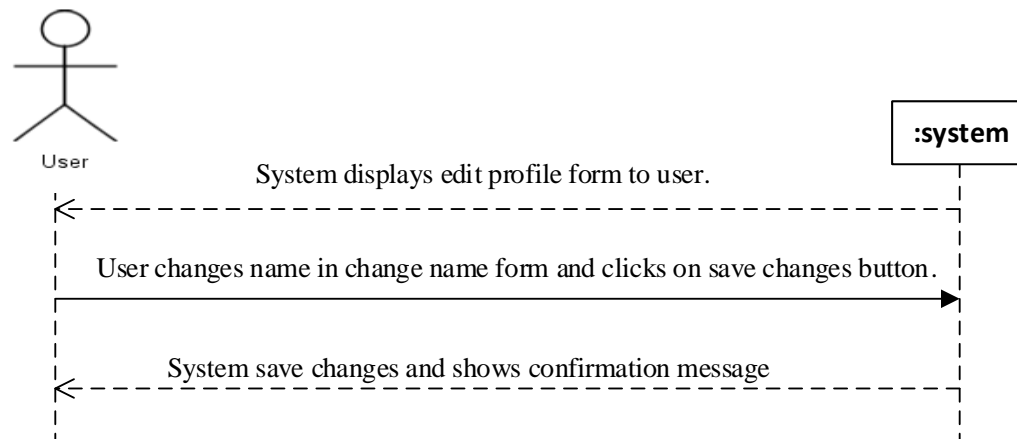
### 5.4.5. Request Password



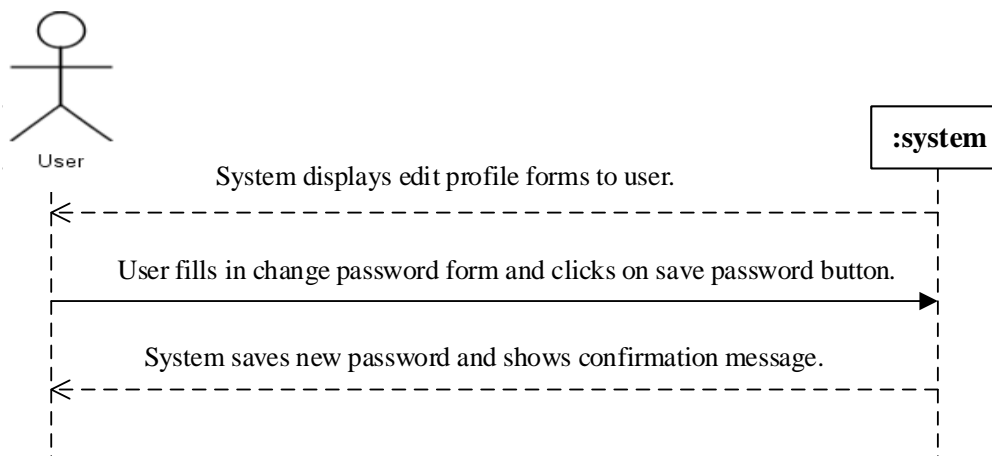
### 5.4.6. Delete Account



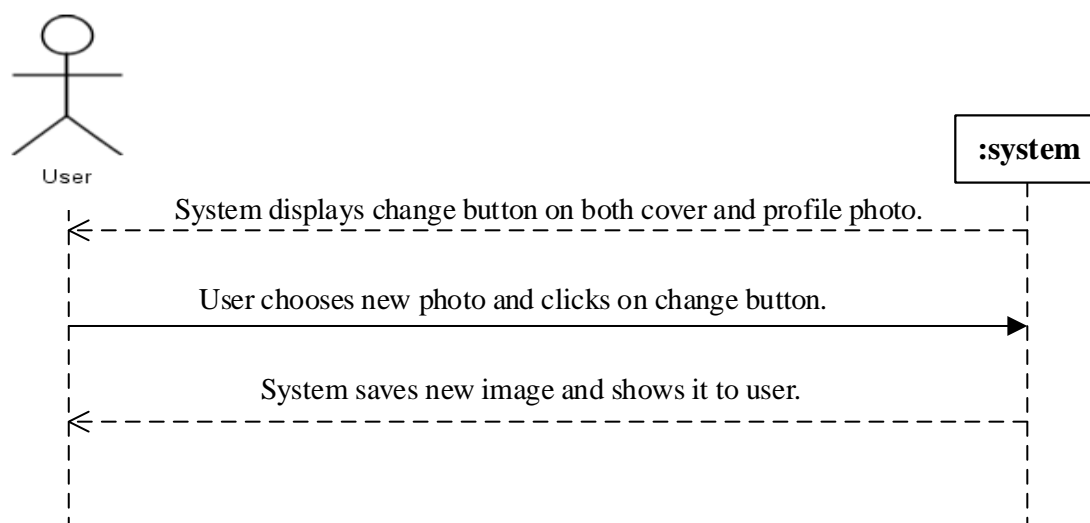
### 5.4.7. Edit Name



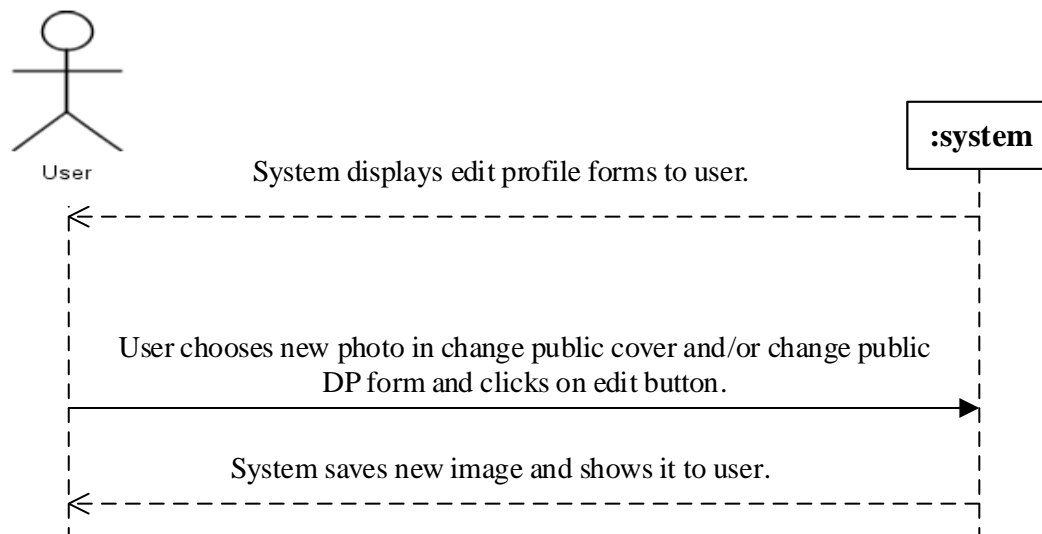
### 5.4.8. Edit Password



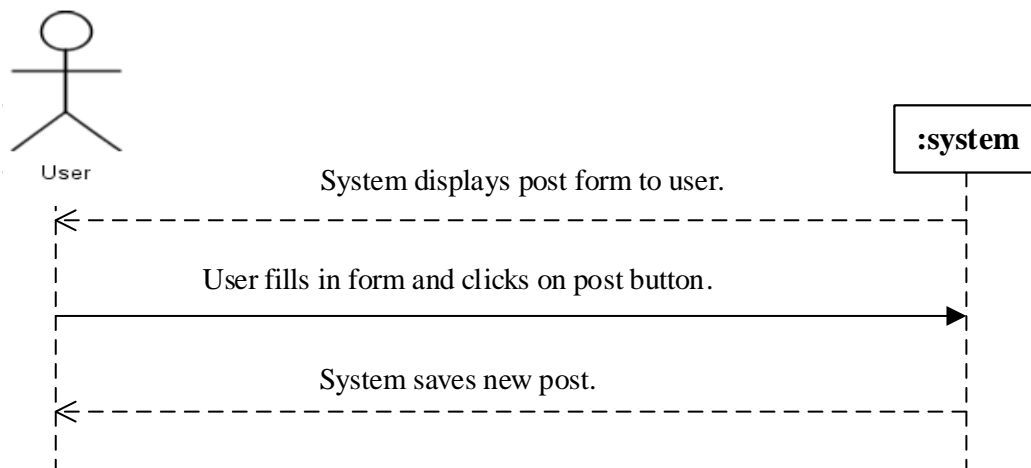
### 5.4.9. Edit cover and DP



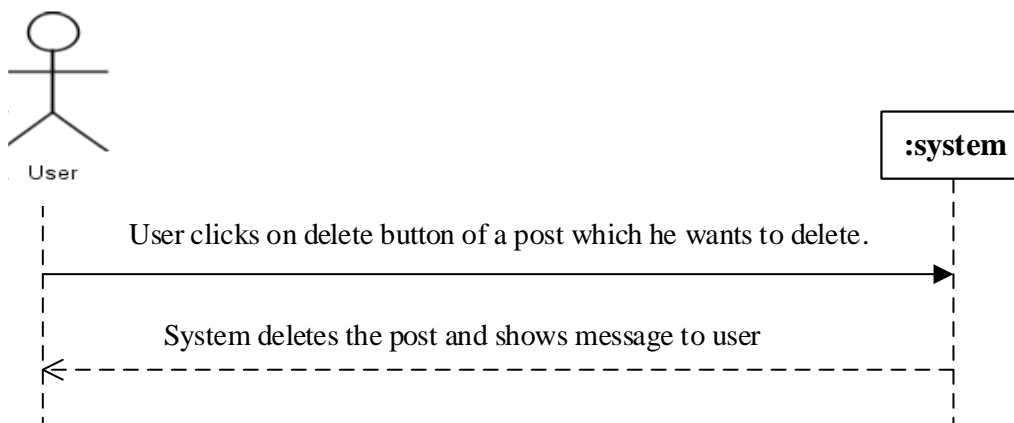
#### 5.4.10. Edit public cover and DP



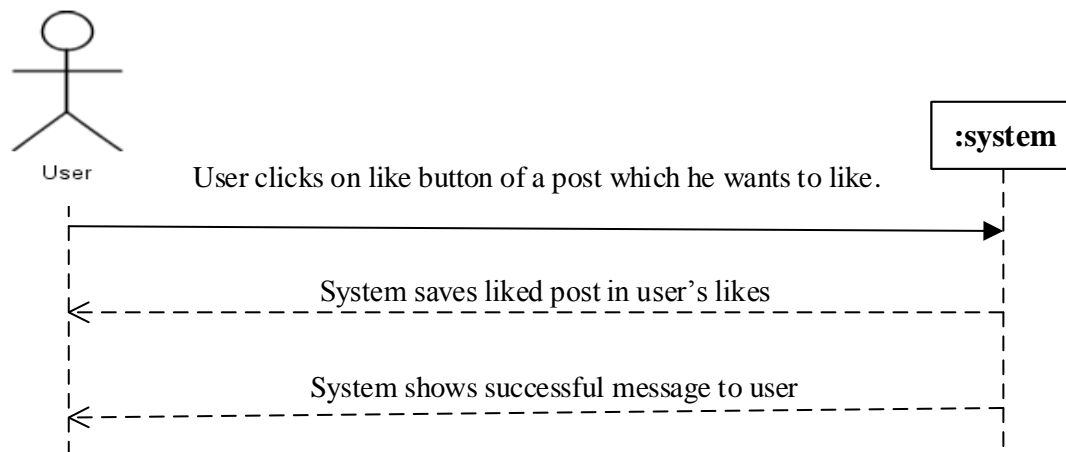
#### 5.4.11. Add post



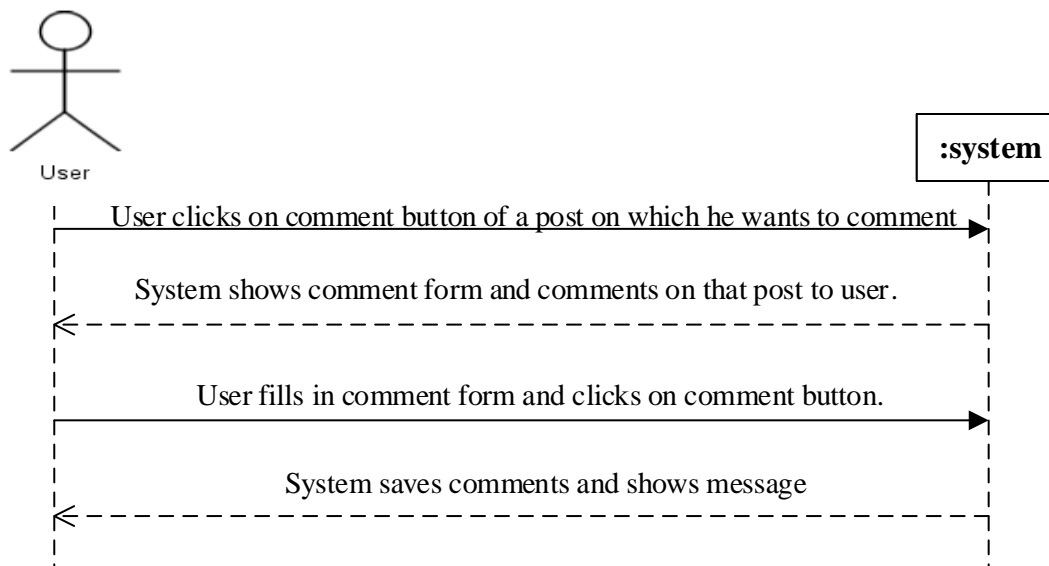
#### 5.4.12. Delete post



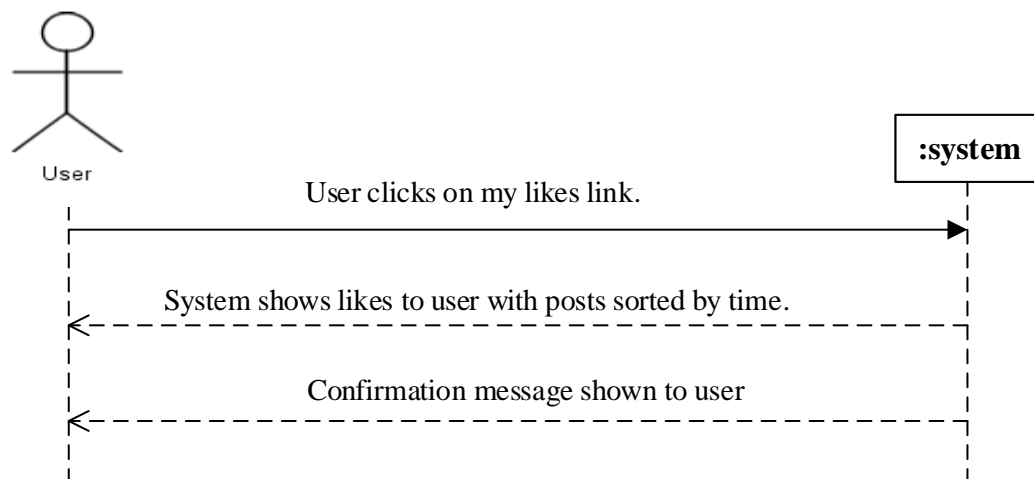
### 5.4.13. like post



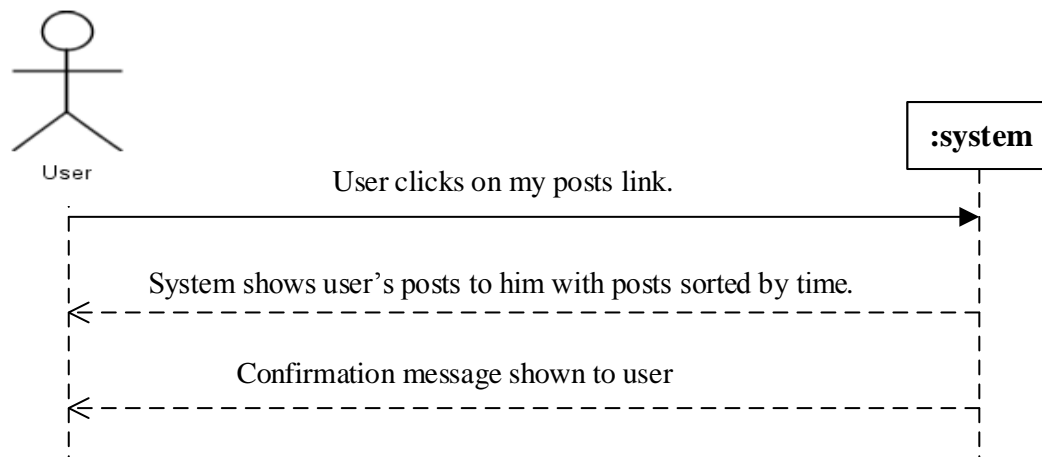
### 5.4.14. Comment on post



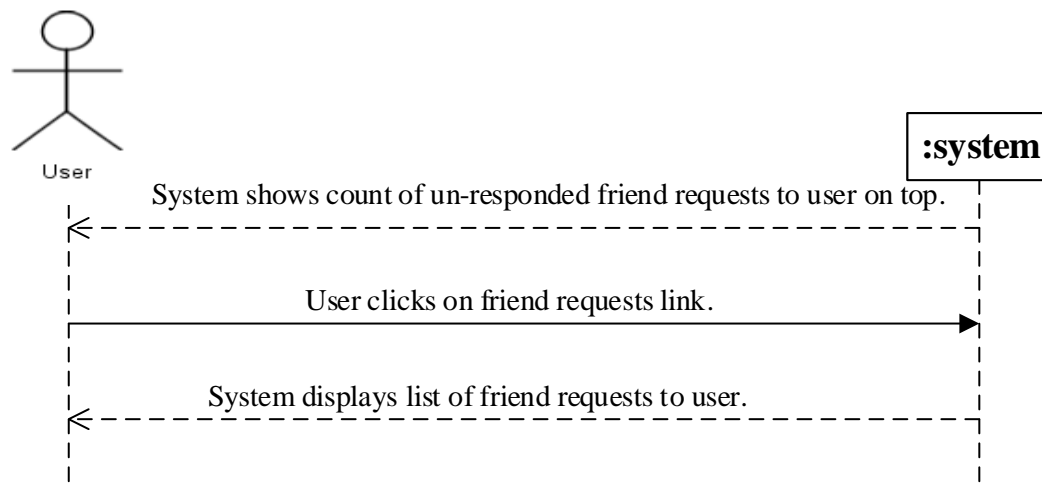
### 5.4.15. View my Likes



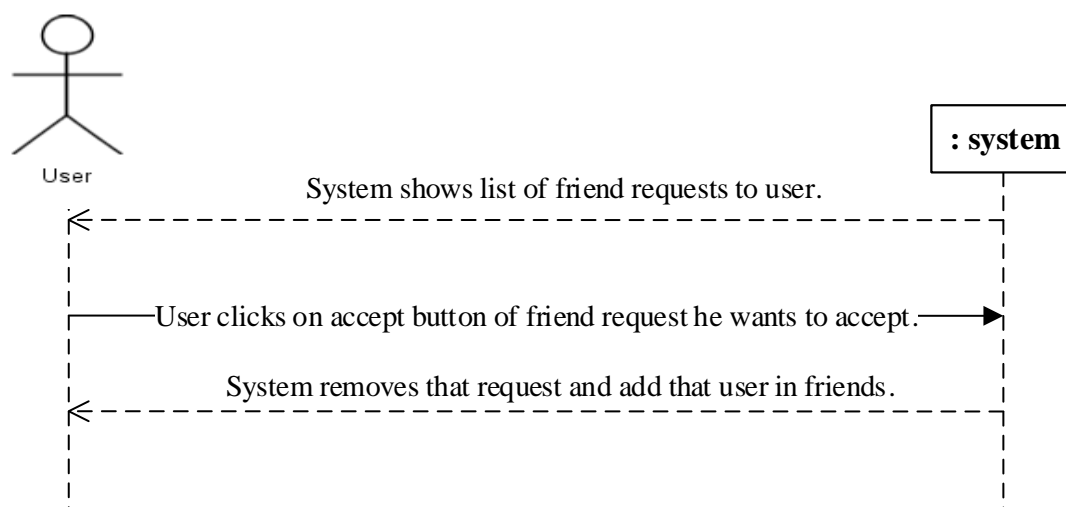
### 5.4.17. View My Posts



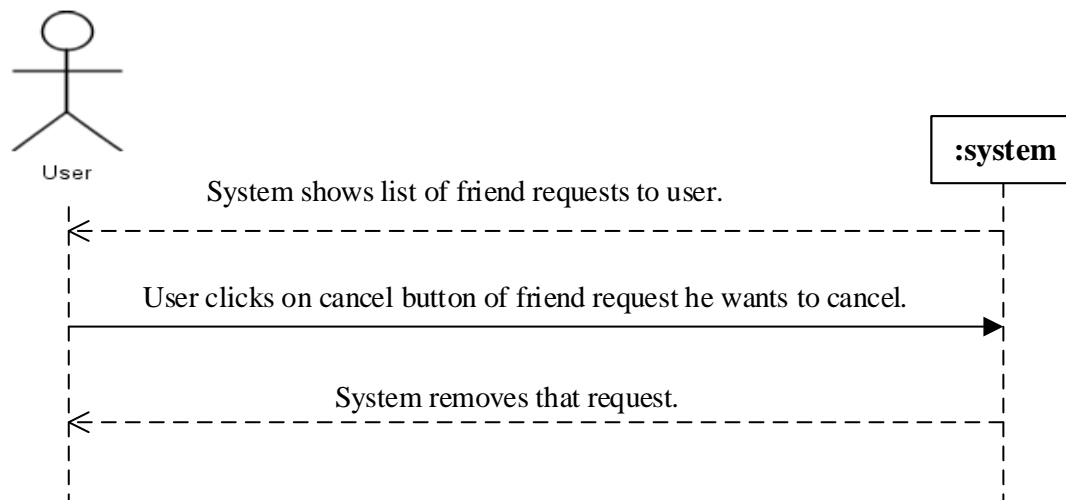
### 5.4.18. View Friend Requests



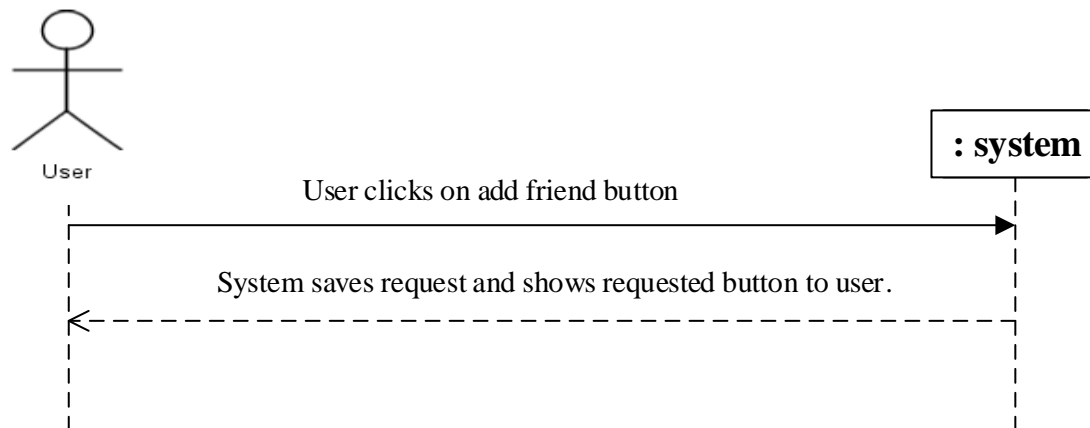
### 5.4.19. Accept Friend Request



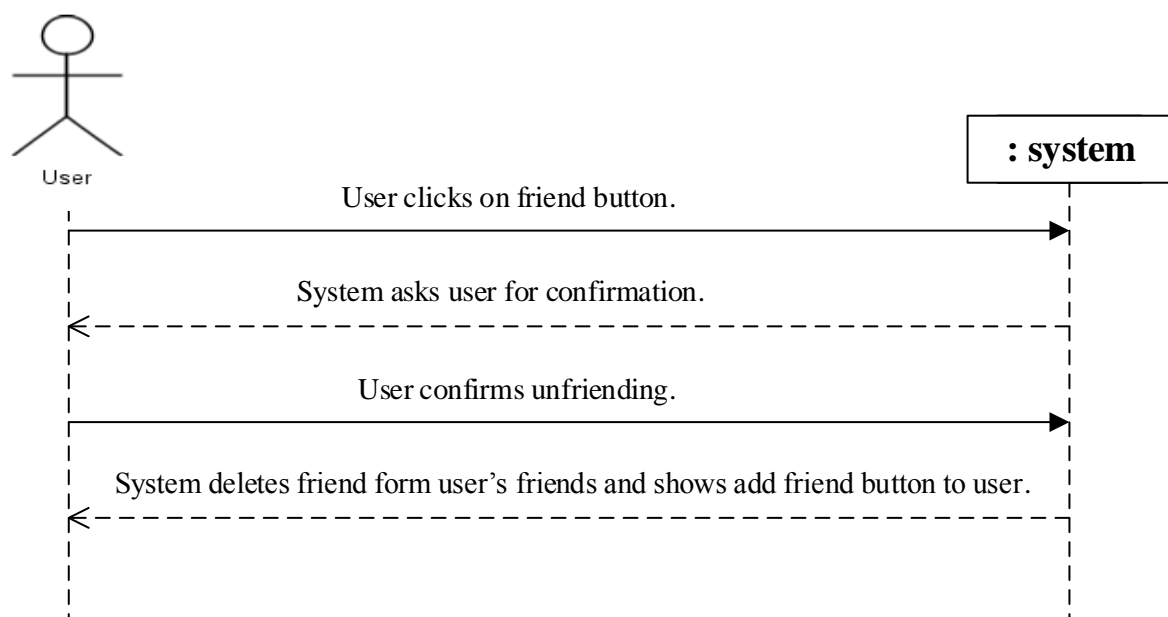
### 5.4.20. Cancel Friend Request



### 5.4.21. Add Friend

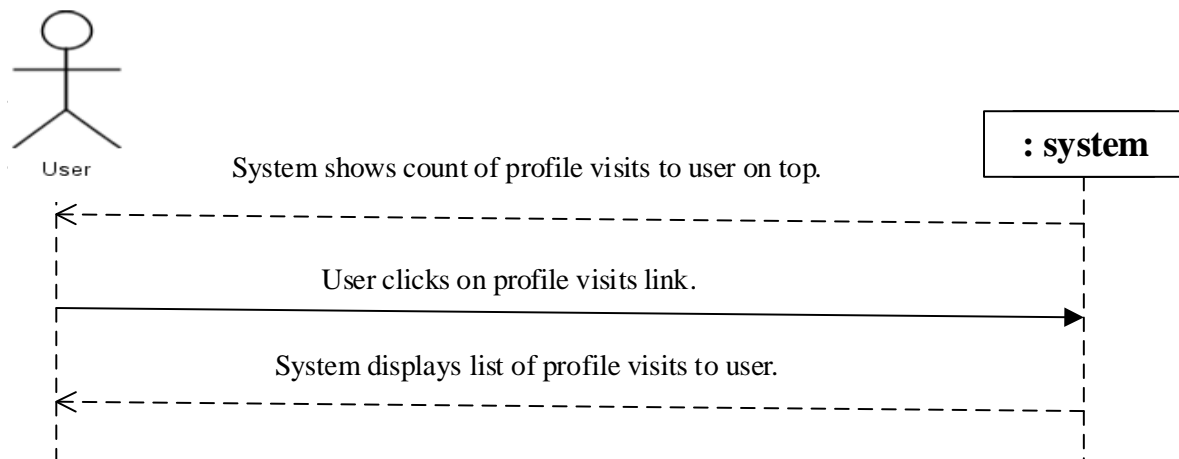


### 5.4.22. Delete Friend

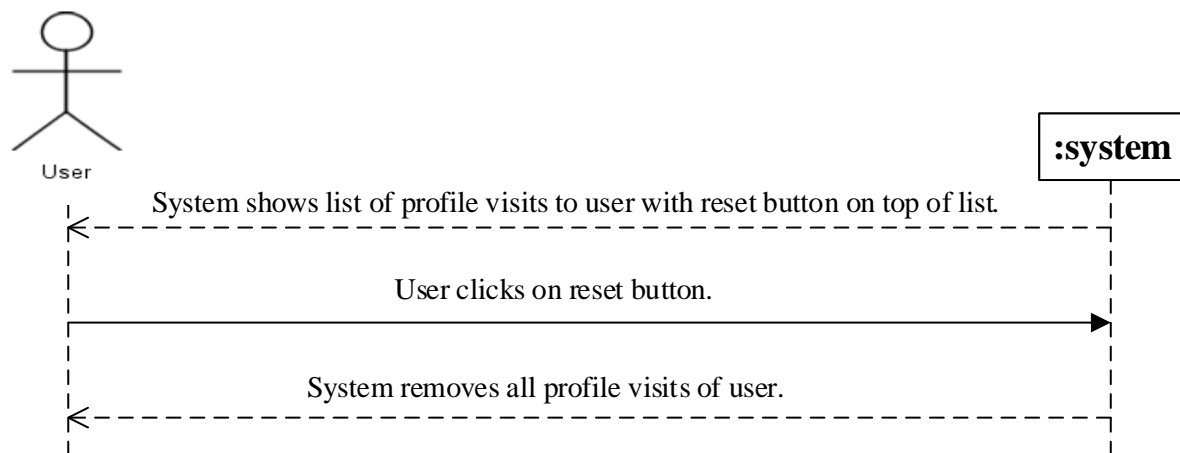




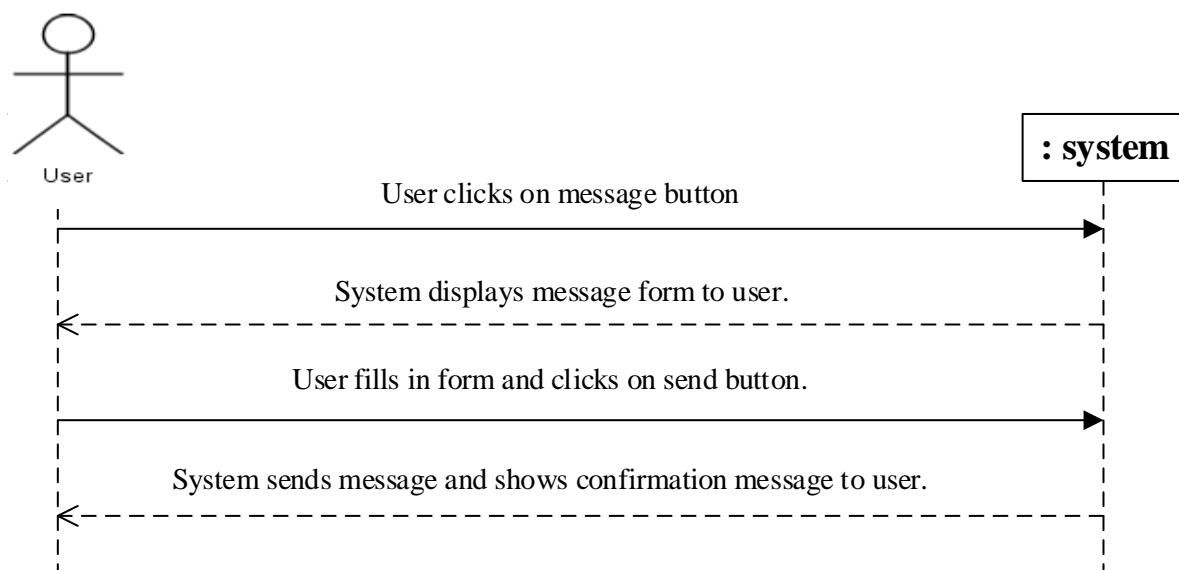
### 5.4.23. View profile visits



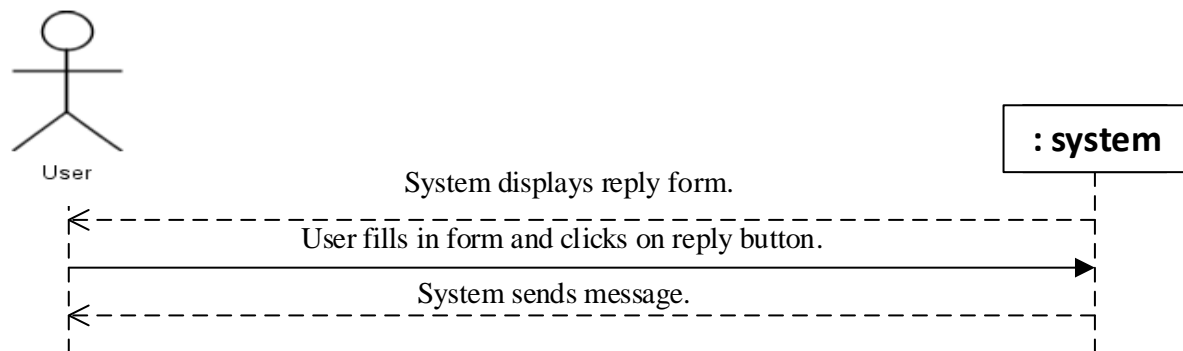
### 5.4.24. Reset profile visits



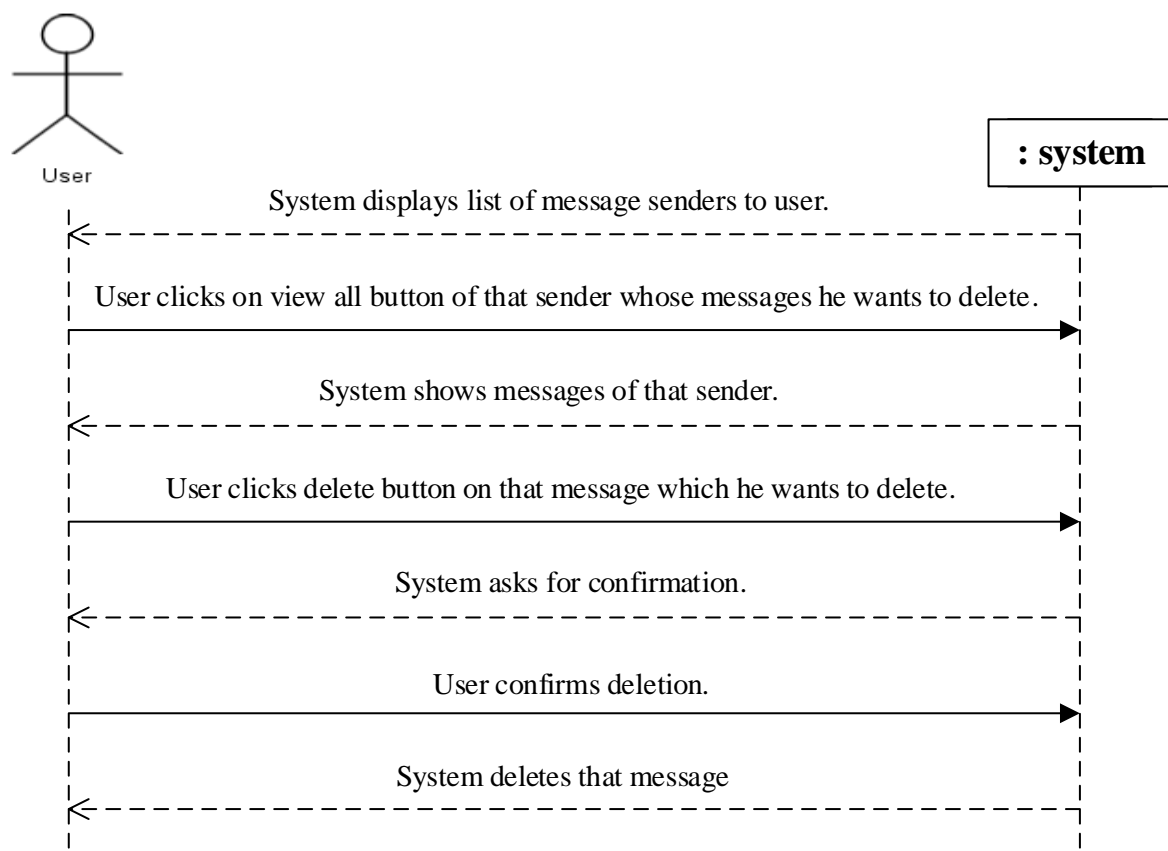
### 5.4.25. Send Message



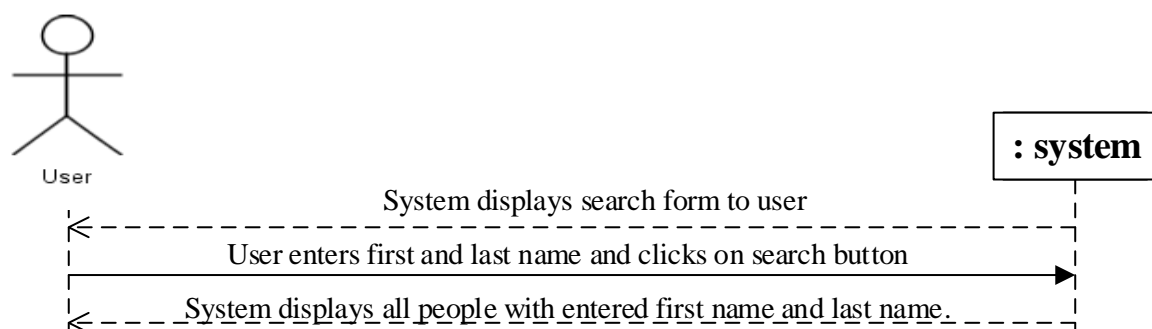
### 5.4.26. Reply to Message



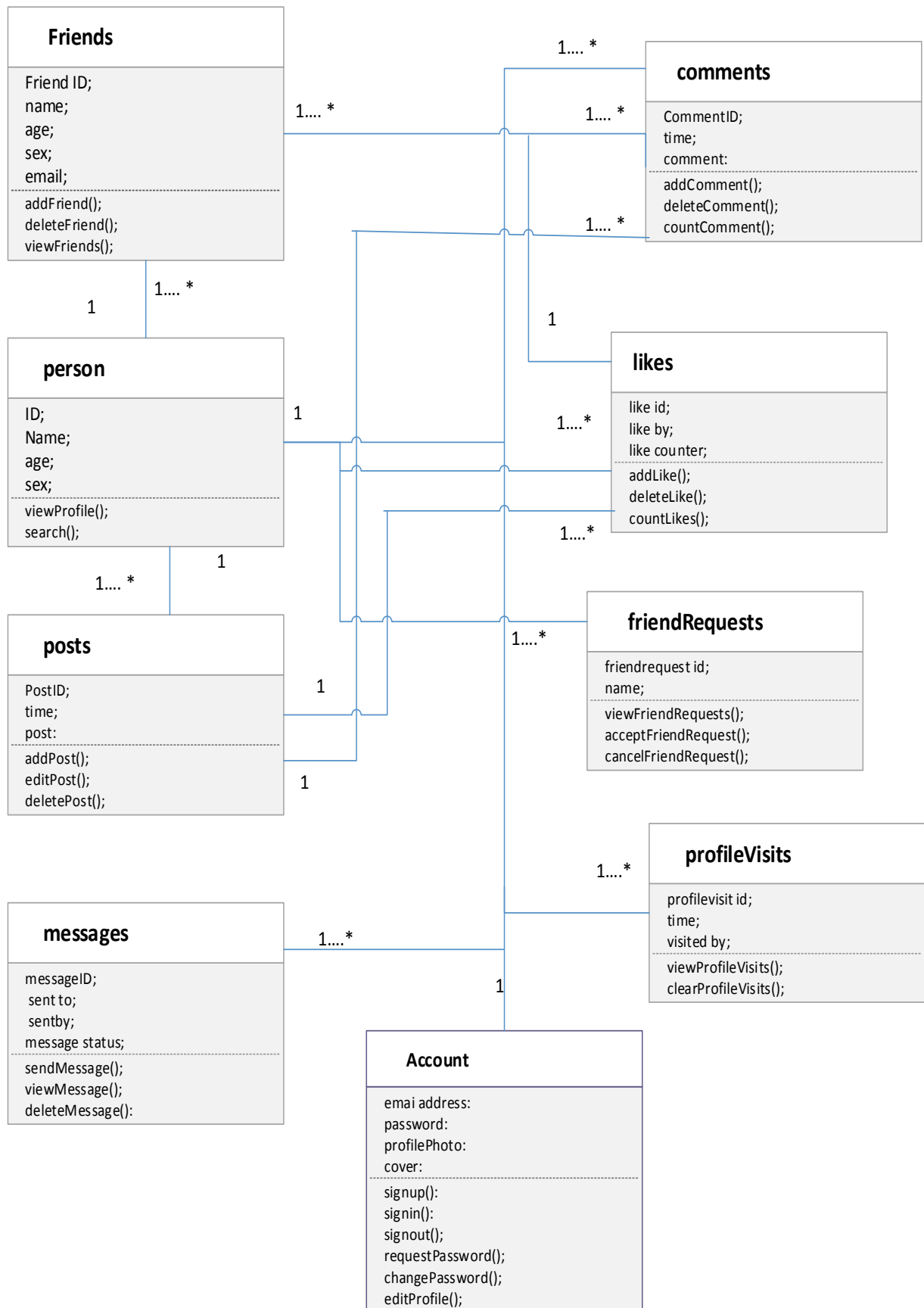
### 5.4.27. Delete Message



### 5.4.28. Search



## 5.5. Class Diagram:



# **CHAPTER 6**

## **IMPLEMENTATION**

## DAO Class

```
package daos;
import beans.person;
import java.sql.*;
import java.util.ArrayList;
public class personDAO {
    private Connection con;
// default constructor
    public personDAO() throws
        ClassNotFoundException, SQLException {
        establishConnection();
    }
// method used to establish connection with db
    private void establishConnection()
        throws ClassNotFoundException, SQLException {
// establishing connection
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        String conUrl = "jdbc:odbc:Site";
        con = DriverManager.getConnection(conUrl);
    }
    private void closeCon() {
        try {
            if (this.con != null) {
                con.close();
            }
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
    public boolean userExists(String e, String p) throws SQLException {
        boolean check = false;
        try {
            establishConnection();
            String sql = "SELECT * FROM person WHERE EmailAddress=\'" + e + "\' AND
Password=\'" + p + "\'";
            PreparedStatement pst = con.prepareStatement(sql);
            ResultSet rs = pst.executeQuery();
            if (rs.next()) {
                check = true;
            }
        } catch (ClassNotFoundException | SQLException ex) {
            System.out.println(ex);
        } finally {
            closeCon();
        }
        return check;
    }
    public String sendPassword(String e) throws SQLException {
        String pass = null;
        try {
```

```

        establishConnection();
        String sql = "SELECT * FROM person WHERE EmailAddress = ?";
        PreparedStatement pst = con.prepareStatement(sql);
        pst.setString(1, e);
        ResultSet rs = pst.executeQuery();
        if (rs.next()) {
            pass = rs.getString("Password");
        }
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
    return pass;
}

public boolean verifyEmail(String e) throws SQLException {
    boolean email = false;
    try {
        establishConnection();
        String sql = "SELECT * FROM person WHERE EmailAddress = ?";
        PreparedStatement pst = con.prepareStatement(sql);
        pst.setString(1, e);
        ResultSet rs = pst.executeQuery();
        if (rs.next()) {
            email = true;
        }
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
    return email;
}

public void verifySignup(String f, String l, String e, String p) throws SQLException {
    try {
        establishConnection();
        String sql1 = "SELECT * FROM person";
        PreparedStatement pStmt = con.prepareStatement(sql1,
        ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
        ResultSet rs1 = pStmt.executeQuery();
        rs1.moveToInsertRow();
        rs1.updateString("FirstName", f);
        rs1.updateString("LastName", l);
        rs1.updateString("EmailAddress", e);
        rs1.updateString("Password", p);
        rs1.insertRow();
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
}

```

```

    }
}
public void addCrDp(String p, String c, String pp, String pc, String e) {
    try {
        establishConnection();
        String sql = "SELECT * FROM person WHERE EmailAddress = ?";
        PreparedStatement pStmt = con.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
        pStmt.setString(1, e);
        ResultSet rs = pStmt.executeQuery();
        if (rs.next()) {
            if (p != null) {
                rs.updateString("ProfilePhoto", p);
            }
            if (c != null) {
                rs.updateString("CoverPhoto", c);
            }
            if (pp != null) {
                rs.updateString("PublicProfile", pp);
            }
            if (pc != null) {
                rs.updateString("PublicCover", pc);
            }
            rs.updateRow();
        }
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
}

public person getPersonP(String e) {
    person p = null;
    try {
        establishConnection();
        String sql = "SELECT * FROM person WHERE EmailAddress = ?";
        PreparedStatement pStmt = con.prepareStatement(sql);
        pStmt.setString(1, e);
        ResultSet rs = pStmt.executeQuery();
        if (rs.next()) {
            p = new person();
            p.setEmailId(e);
            p.setProPhoto(rs.getString("ProfilePhoto"));
            p.setPublicProPhoto(rs.getString("PublicProfile"));
            p.setCovPhoto(rs.getString("CoverPhoto"));
            p.setPublicCovPhoto(rs.getString("PublicCover"));
            p.setFirstName(rs.getString("FirstName"));
            p.setLastName(rs.getString("LastName"));
        }
    } catch (ClassNotFoundException | SQLException ex) {

```

```

        System.out.println(ex);
    } finally {
        closeCon();
    }
    return p;
}

public person header(String e) {
    person p = null;
    try {
        establishConnection();
        String sql = "SELECT * FROM person WHERE EmailAddress = ?";
        PreparedStatement pStmt = con.prepareStatement(sql);
        pStmt.setString(1, e);
        ResultSet rs = pStmt.executeQuery();
        if (rs.next()) {
            p = new person();
            p.setProPhoto(rs.getString("ProfilePhoto"));
            p.setFirstName(rs.getString("FirstName"));
            p.setLastName(rs.getString("LastName"));
        }
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
    return p;
}

public void editProfile(String fn, String ln, String e) {
    try {
        establishConnection();
        String sql = "SELECT * FROM person WHERE EmailAddress = ?";
        PreparedStatement pStmt = con.prepareStatement(sql,
ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
        pStmt.setString(1, e);
        ResultSet rs = pStmt.executeQuery();
        if (rs.next()) {
            rs.updateString("FirstName", fn);
            rs.updateString("LastName", ln);
            //rs.updateString("EmailAddress", ne);
            rs.updateRow();
        }
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
}

public void changePass(String p, String e) {
    try {
        establishConnection();

```



```

        String sql = "SELECT * FROM person WHERE EmailAddress = ?";
        PreparedStatement pStmt = con.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
        pStmt.setString(1, e);
        ResultSet rs = pStmt.executeQuery();
        if (rs.next()) {
            rs.updateString("Password", p);
            rs.updateRow();
        }
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
}

public ArrayList getPersons(String f, String l) {
    ArrayList list = new ArrayList();
    try {
        establishConnection();
        String sql = "SELECT * FROM person WHERE FirstName = '\" + f + '\" AND
        LastName = '\" + l + '\"";
        PreparedStatement pStmt = con.prepareStatement(sql);
        //pStmt.setString(1, f);
        //pStmt.setString(1, l);
        ResultSet rs = pStmt.executeQuery();
        while (rs.next()) {
            person p = new person();
            p.setEmailId(rs.getString("EmailAddress"));
            p.setCovPhoto(rs.getString("CoverPhoto"));
            p.setProPhoto(rs.getString("ProfilePhoto"));
            p.setPublicCovPhoto(rs.getString("PublicCover"));
            p.setPublicProPhoto(rs.getString("PublicProfile"));
            p.setFirstName(rs.getString("FirstName"));
            p.setLastName(rs.getString("LastName"));
            list.add(p);
        }
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
    return list;
}

public ArrayList getPersonsFN(String f) {
    ArrayList list = new ArrayList();
    try {
        establishConnection();
        String sql = "SELECT * FROM person WHERE FirstName = ? ";
        PreparedStatement pStmt = con.prepareStatement(sql);
        pStmt.setString(1, f);

```

```

        ResultSet rs = pStmt.executeQuery();
        while (rs.next()) {
            person p = new person();
            p.setEmailId(rs.getString("EmailAddress"));
            p.setCovPhoto(rs.getString("CoverPhoto"));
            p.setProPhoto(rs.getString("ProfilePhoto"));
            p.setPublicCovPhoto(rs.getString("PublicCover"));
            p.setPublicProPhoto(rs.getString("PublicProfile"));
            p.setFirstName(rs.getString("FirstName"));
            p.setLastName(rs.getString("LastName"));
            list.add(p);
        }
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
    return list;
}

public ArrayList getPersonsLN(String l) {
    ArrayList list = new ArrayList();
    try {
        establishConnection();
        String sql = "SELECT * FROM person WHERE LastName = ? ";
        PreparedStatement pStmt = con.prepareStatement(sql);

        pStmt.setString(1, l);
        ResultSet rs = pStmt.executeQuery();
        while (rs.next()) {
            person p = new person();
            p.setEmailId(rs.getString("EmailAddress"));
            p.setCovPhoto(rs.getString("CoverPhoto"));
            p.setProPhoto(rs.getString("ProfilePhoto"));
            p.setPublicCovPhoto(rs.getString("PublicCover"));
            p.setPublicProPhoto(rs.getString("PublicProfile"));
            p.setFirstName(rs.getString("FirstName"));
            p.setLastName(rs.getString("LastName"));
            list.add(p);
        }
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
    return list;
}

public ArrayList getPosts(String e) {
    ArrayList list = new ArrayList();
    try {

```

```

        establishConnection();
        String sql = "SELECT * FROM posts WHERE EmailAddress = ? ORDER BY time
DESC";
        PreparedStatement pStmt = con.prepareStatement(sql);
        pStmt.setString(1, e);
        ResultSet rs = pStmt.executeQuery();
        while (rs.next()) {
            person p = new person();
            p.setPostId(Double.parseDouble(rs.getString("postID")));
            p.setPost(rs.getString("Post"));
            p.setPic(rs.getString("pic"));
            p.setTime(rs.getString("time"));
            list.add(p);
        }
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
}
return list;
}

public void addPosts(String e, String p, Timestamp t, String pic) {
    try {
        establishConnection();
        String sql = "SELECT * FROM posts";
        PreparedStatement pStmt = con.prepareStatement(sql,
ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
        ResultSet rs = pStmt.executeQuery();
        rs.moveToInsertRow();
        rs.updateString("EmailAddress", e);
        rs.updateString("Post", p);
        rs.updateTimestamp("time", t);
        rs.updateString("pic", pic);
        rs.insertRow();
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
}

public void editPost(String p, double pid) {
    try {
        establishConnection();
        String sql = "SELECT * FROM posts WHERE postID = ?";
        PreparedStatement pStmt = con.prepareStatement(sql,
ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
        pStmt.setDouble(1, pid);
        ResultSet rs = pStmt.executeQuery();
        if (rs.next()) {
            rs.updateString("Post", p);

```

```

        rs.updateRow();
    }
} catch (ClassNotFoundException | SQLException ex) {
    System.out.println(ex);
} finally {
    closeCon();
}
}
}
public void delPost(double pid) {
    try {
        establishConnection();
        String sql = "DELETE FROM posts WHERE postID = ?";
        PreparedStatement pStmt = con.prepareStatement(sql,
ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
        pStmt.setDouble(1, pid);
        ResultSet rs = pStmt.executeQuery();
        if (rs.next()) {
            rs.updateRow();
        }
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
}
}
public void addFriend(String e, String r) throws SQLException {
    try {
        establishConnection();
        String sql = "SELECT * FROM friendRequest";
        PreparedStatement pStmt = con.prepareStatement(sql,
ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
        ResultSet rs = pStmt.executeQuery();
        rs.moveToInsertRow();
        rs.updateString("EmailAddress", r);
        rs.updateString("requestId", e);
        rs.insertRow();
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
}
}
public boolean checkFriend(String e, String fe) throws SQLException {
    boolean check = false;
    try {
        establishConnection();
        String sql = "SELECT * FROM friends WHERE EmailAddress=\"\" + e + \"' AND
friend_id=\"\" + fe + \"'";
        PreparedStatement pst = con.prepareStatement(sql);

```

```

        ResultSet rs = pst.executeQuery();

        if (rs.next()) {
            check = true;
        }
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
    return check;
}

public void unfriend(String e, String ue) {
    try {
        establishConnection();
        String sql = "DELETE FROM friends WHERE EmailAddress = ? AND friend_id = ?";
        PreparedStatement pStmt = con.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
        pStmt.setString(1, e);
        pStmt.setString(2, ue);
        ResultSet rs = pStmt.executeQuery();
        if (rs.next()) {
            rs.updateRow();
        }
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
}

public ArrayList friendList(String e) {
    ArrayList list = new ArrayList();
    try {
        establishConnection();
        String sql = "SELECT * FROM person, friends WHERE
person.EmailAddress=friends.friend_id AND friends.EmailAddress = ?";
        PreparedStatement pStmt = con.prepareStatement(sql);
        pStmt.setString(1, e);
        ResultSet rs = pStmt.executeQuery();
        while (rs.next()) {
            person p = new person();
            p.setEmailId(rs.getString("EmailAddress"));
            p.setCovPhoto(rs.getString("CoverPhoto"));
            p.setProPhoto(rs.getString("ProfilePhoto"));
            p.setFirstName(rs.getString("FirstName"));
            p.setLastName(rs.getString("LastName"));
            list.add(p);
        }
    } catch (ClassNotFoundException | SQLException ex) {

```

```

        System.out.println(ex);
    } finally {
        closeCon();
    }
    return list;
}

public ArrayList getPosts_TL(String e) {
    ArrayList list = new ArrayList();
    try {
        establishConnection();
        String sql = "SELECT distinct posts.postID, posts.Post, posts.EmailAddress,
posts.time, posts.pic \n"
            + "FROM posts, friends \n"
            + "WHERE (posts.EmailAddress = friends.friend_id \n"
            + "AND friends.EmailAddress = ? ) OR posts.EmailAddress = ?\n"
            + "ORDER BY time DESC";
        PreparedStatement pstmt = con.prepareStatement(sql);
        pstmt.setString(1, e);
        pstmt.setString(2, e);
        ResultSet rs = pstmt.executeQuery();
        while (rs.next()) {
            person p = new person();
            p.setPostId(Double.parseDouble(rs.getString("postID")));
            p.setPost(rs.getString("Post"));
            p.setPic(rs.getString("pic"));
            p.setTime(rs.getString("time"));
            p.setEmailId(rs.getString("EmailAddress"));
            list.add(p);
        }
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
    return list;
}

public void likePost(double pid, String u) {
    try {
        establishConnection();
        String sql = "SELECT * FROM like";
        PreparedStatement pstmt = con.prepareStatement(sql,
ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
        ResultSet rs = pstmt.executeQuery();
        rs.moveToInsertRow();
        rs.updateDouble("postID", pid);
        rs.updateString("likedby", u);
        rs.insertRow();
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {

```

```

        closeCon();
    }
}

public boolean checkUliked(double pid, String ue) {

    boolean check = false;
    try {
        establishConnection();
        String sql = "SELECT * FROM like WHERE postID = ? AND likedby = ?";
        PreparedStatement pst = con.prepareStatement(sql);
        pst.setDouble(1, pid);
        pst.setString(2, ue);
        ResultSet rs = pst.executeQuery();
        if (rs.next()) {
            check = true;
        }
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
    return check;
}

public ArrayList getLikes(String e) {
    ArrayList list = new ArrayList();
    try {
        establishConnection();
        String sql = "SELECT * \n"
            + "FROM posts \n"
            + "WHERE postID IN \n"
            + "                (SELECT postID\n"
            + "                FROM like \n"
            + "                WHERE likedby = ?)\n"
            + "ORDER BY time DESC";
        PreparedStatement pStmt = con.prepareStatement(sql);
        pStmt.setString(1, e);
        ResultSet rs = pStmt.executeQuery();
        while (rs.next()) {
            person p = new person();
            p.setPostId(Double.parseDouble(rs.getString("postID")));
            p.setPost(rs.getString("Post"));
            p.setPic(rs.getString("pic"));
            p.setTime(rs.getString("time"));
            p.setEmailId(rs.getString("EmailAddress"));
            list.add(p);
        }
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
}

```

```

    }
    return list;
}
public void unlikePost(double pid, String u) {
    try {
        establishConnection();
        String sql = "DELETE FROM like WHERE postID = ? AND likedby = ?";
        PreparedStatement pStmt = con.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
        pStmt.setDouble(1, pid);
        pStmt.setString(2, u);
        ResultSet rs = pStmt.executeQuery();
        if (rs.next()) {
            rs.updateRow();
        }
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
}

public ArrayList getPosts_TL1(String e) {
    ArrayList list = new ArrayList();
    try {
        establishConnection();
        String sql = "SELECT DISTINCT posts.postID, posts.Post, posts.EmailAddress,
posts.time, posts.pic, FirstName\n"
        + "FROM posts, friends, person\n"
        + "WHERE (posts.EmailAddress=friends.friend_id And
posts.EmailAddress=person.EmailAddress And friends.EmailAddress=[?]) \n"
        + "Or (posts.EmailAddress=[?] And
posts.EmailAddress=person.EmailAddress)\n"
        + "ORDER BY FirstName;";
        PreparedStatement pStmt = con.prepareStatement(sql);
        pStmt.setString(1, e);
        pStmt.setString(2, e);
        ResultSet rs = pStmt.executeQuery();
        while (rs.next()) {
            person p = new person();
            p.setPostId(Double.parseDouble(rs.getString("postID")));
            p.setPost(rs.getString("Post"));
            p.setPic(rs.getString("pic"));
            p.setTime(rs.getString("time"));
            p.setEmailId(rs.getString("EmailAddress"));
            list.add(p);
        }
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
}

```



```

    }
    return list;
}
public ArrayList getLikes1(String e) {
    ArrayList list = new ArrayList();
    try {
        establishConnection();
        String sql = "SELECT postID, post, time, FirstName, posts.EmailAddress,
person.EmailAddress, posts.pic\n"
            + "FROM posts, person\n"
            + "WHERE posts.EmailAddress=person.EmailAddress AND postID IN
(SELECT postID FROM like \n"
            + "                WHERE likedby = ?)\n"
            + "ORDER BY FirstName;";
        PreparedStatement pStmt = con.prepareStatement(sql);
        pStmt.setString(1, e);
        ResultSet rs = pStmt.executeQuery();
        while (rs.next()) {
            person p = new person();
            p.setPostId(Double.parseDouble(rs.getString("postID")));
            p.setPost(rs.getString("Post"));
            p.setPic(rs.getString("pic"));
            p.setTime(rs.getString("time"));
            p.setEmailId(rs.getString("EmailAddress"));
            list.add(p);
        }
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
    return list;
}
public person getPost(double pi) throws SQLException {
    person p = null;
    try {
        establishConnection();
        String sql = "SELECT * FROM posts WHERE postID = ?";
        PreparedStatement pst = con.prepareStatement(sql);
        pst.setDouble(1, pi);
        ResultSet rs = pst.executeQuery();
        if (rs.next()) {
            p = new person();
            p.setPostId(pi);
            p.setEmailId(rs.getString("EmailAddress"));
            p.setPost(rs.getString("Post"));
            p.setPic(rs.getString("pic"));
            p.setTime(rs.getString("time"));
        }
    } catch (ClassNotFoundException | SQLException ex) {

```

```

        System.out.println(ex);
    } finally {
        closeCon();
    }
    return p;
}

public boolean checkDuplicateComment(double p, String e, String c) {
    boolean cdc = false;
    try {
        establishConnection();
        String sql = "SELECT * FROM comment WHERE (postID = ? AND commentby = ?
AND comment = ?)";
        PreparedStatement pStmt = con.prepareStatement(sql);
        pStmt.setDouble(1, p);
        pStmt.setString(2, e);
        pStmt.setString(3, c);
        ResultSet rs = pStmt.executeQuery();
        if (rs.next()) {
            cdc = true;
        }
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
    return cdc;
}

public void addComments(double p, String e, String c, Timestamp t) {
    try {
        establishConnection();
        String sql = "SELECT * FROM comment";
        PreparedStatement pStmt = con.prepareStatement(sql,
ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
        ResultSet rs = pStmt.executeQuery();
        rs.moveToInsertRow();
        rs.updateDouble("postId", p);
        rs.updateString("commentby", e);
        rs.updateString("comment", c);
        rs.updateTimestamp("time", t);
        rs.insertRow();
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
}

public ArrayList getComments(double pi) {
    ArrayList list = new ArrayList();
    try {
        establishConnection();

```

```

        String sql = "SELECT * FROM comment WHERE postID = ?";
        PreparedStatement pStmt = con.prepareStatement(sql);
        pStmt.setDouble(1, pi);
        ResultSet rs = pStmt.executeQuery();
        while (rs.next()) {
            person p = new person();
            p.setPostId(pi);
            p.setEmailId(rs.getString("commentby"));
            p.setPost(rs.getString("comment"));
            p.setTime(rs.getString("time"));
            list.add(p);
        }
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
    return list;
}

public int countComments(double pi) {
    int c = 0;
    try {
        establishConnection();
        String sql = "SELECT * FROM comment WHERE postID = ?";
        PreparedStatement pStmt = con.prepareStatement(sql);
        pStmt.setDouble(1, pi);
        ResultSet rs = pStmt.executeQuery();
        while (rs.next()) {
            c++;
        }
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
    return c;
}

public int countLikes(double pi) {
    int c = 0;
    try {
        establishConnection();
        String sql = "SELECT * FROM like WHERE postID = ?";
        PreparedStatement pStmt = con.prepareStatement(sql);
        pStmt.setDouble(1, pi);
        ResultSet rs = pStmt.executeQuery();
        while (rs.next()) {
            c++;
        }
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    }
}

```

```

    } finally {
        closeCon();
    }
    return c;
}

public void acceptRequest(String e, String fe) throws SQLException {
    try {
        establishConnection();
        String sql = "SELECT * FROM friends";
        PreparedStatement pstmt = con.prepareStatement(sql,
ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
        ResultSet rs = pstmt.executeQuery();
        rs.moveToInsertRow();
        rs.updateString("EmailAddress", e);
        rs.updateString("friend_id", fe);
        rs.insertRow();
        String sql1 = "SELECT * FROM friends";
        PreparedStatement pstmt1 = con.prepareStatement(sql1,
ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
        ResultSet rs1 = pstmt1.executeQuery();
        rs1.moveToInsertRow();
        rs1.updateString("EmailAddress", fe);
        rs1.updateString("friend_id", e);
        rs1.insertRow();
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
}

public boolean checkRequest(String e, String fe) throws SQLException {
    boolean check = false;
    try {
        establishConnection();
        String sql = "SELECT * FROM friendRequest WHERE EmailAddress=\'" + fe + "\'
AND requestId=\'" + e + "\'";
        PreparedStatement pst = con.prepareStatement(sql);
        ResultSet rs = pst.executeQuery();
        if (rs.next()) {
            check = true;
        }
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
    return check;
}

```

```

public boolean checkSentRequest(String e, String fe) throws SQLException {

```

```
        boolean check = false;
        try {
            establishConnection();
            String sql = "SELECT * FROM friendRequest WHERE EmailAddress=\'" + e + "\'
AND requestId=\'" + fe + "\'";
            PreparedStatement pst = con.prepareStatement(sql);
            ResultSet rs = pst.executeQuery();
            if (rs.next()) {
                check = true;
            }
        } catch (ClassNotFoundException | SQLException ex) {
            System.out.println(ex);
        } finally {
            closeCon();
        }
        return check;
    }

    public void cancelRequest(String e, String ue) {
        try {
            establishConnection();
            String sql = "DELETE FROM friendRequest WHERE EmailAddress = ? AND
requestId = ?";
            PreparedStatement pStmt = con.prepareStatement(sql,
ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
            pStmt.setString(1, e);
            pStmt.setString(2, ue);
            ResultSet rs = pStmt.executeQuery();
            if (rs.next()) {
                rs.updateRow();
            }
        } catch (ClassNotFoundException | SQLException ex) {
            System.out.println(ex);
        } finally {
            closeCon();
        }
    }

    public int getFriendRequests(String e) {
        int c = 0;
        try {
            establishConnection();
            String sql = "SELECT * FROM friendRequest WHERE EmailAddress = ?";
            PreparedStatement pStmt = con.prepareStatement(sql);
            pStmt.setString(1, e);
            ResultSet rs = pStmt.executeQuery();
            while (rs.next()) {
                c++;
            }
        } catch (ClassNotFoundException | SQLException ex) {
            System.out.println(ex);
        } finally {
            closeCon();
        }
    }
}
```

```

        closeCon();
    }
    return c;
}
public ArrayList friendRequests(String e) {
    ArrayList list = new ArrayList();
    try {
        establishConnection();
        String sql = "SELECT * FROM person, friendRequest WHERE
person.EmailAddress=friendRequest.requestId AND friendRequest.EmailAddress = ?";
        PreparedStatement pStmt = con.prepareStatement(sql);
        pStmt.setString(1, e);
        ResultSet rs = pStmt.executeQuery();
        while (rs.next()) {
            person p = new person();
            p.setEmailId(rs.getString("EmailAddress"));
            p.setCovPhoto(rs.getString("CoverPhoto"));
            p.setProPhoto(rs.getString("ProfilePhoto"));
            p.setPublicCovPhoto(rs.getString("PublicCover"));
            p.setPublicProPhoto(rs.getString("PublicProfile"));
            p.setFirstName(rs.getString("FirstName"));
            p.setLastName(rs.getString("LastName"));
            list.add(p);
        }
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
    return list;
}
public void addVisits(String e, String r, Timestamp t) throws SQLException {
    try {
        establishConnection();
        String sql1 = "SELECT * FROM profileVisits";
        PreparedStatement pStmt1 = con.prepareStatement(sql1,
ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
        ResultSet rs1 = pStmt1.executeQuery();
        rs1.moveToInsertRow();
        rs1.updateString("EmailAddress", r);
        rs1.updateString("visitedBy", e);
        rs1.updateTimestamp("Time", t);
        rs1.insertRow();
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
}
}

```

```

public int getProfileVisits(String e) {
    int c = 0;
    try {
        establishConnection();
        String sql = "SELECT * FROM profileVisits WHERE EmailAddress = ?";
        PreparedStatement pStmt = con.prepareStatement(sql);
        pStmt.setString(1, e);
        ResultSet rs = pStmt.executeQuery();
        while (rs.next()) {
            c++;
        }
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
    return c;
}

public ArrayList profileVisits(String e) {
    ArrayList list = new ArrayList();
    try {
        establishConnection();
        String sql = "SELECT * FROM person, profileVisits WHERE
person.EmailAddress=profileVisits.visitedBy AND profileVisits.EmailAddress = ? ORDER
BY Time desc";
        PreparedStatement pStmt = con.prepareStatement(sql);
        pStmt.setString(1, e);
        ResultSet rs = pStmt.executeQuery();
        while (rs.next()) {
            person p = new person();
            p.setEmailId(rs.getString(1));
            p.setCovPhoto(rs.getString("CoverPhoto"));
            p.setProPhoto(rs.getString("ProfilePhoto"));
            p.setPublicCovPhoto(rs.getString("PublicCover"));
            p.setPublicProPhoto(rs.getString("PublicProfile"));
            p.setFirstName(rs.getString("FirstName"));
            p.setLastName(rs.getString("LastName"));
            p.setTime(rs.getString("Time"));
            list.add(p);
        }
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
    return list;
}

public void resetVisits(String e) {
    try {
        establishConnection();

```

```

        String sql = "DELETE FROM profileVisits WHERE EmailAddress = ?";
        PreparedStatement pStmt = con.prepareStatement(sql,
ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
        pStmt.setString(1, e);
        ResultSet rs = pStmt.executeQuery();
        while (rs.next()) {
            rs.updateRow();
        }
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
}

public void sendMessage(String m, String st, String sb, Timestamp t) throws
SQLException {
    try {
        establishConnection();
        String sql = "SELECT * FROM messages";
        PreparedStatement pStmt = con.prepareStatement(sql,
ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
        ResultSet rs = pStmt.executeQuery();
        rs.moveToInsertRow();
        rs.updateString("message", m);
        rs.updateString("sentTo", st);
        rs.updateString("sentBy", sb);
        rs.updateTimestamp("time", t);
        rs.insertRow();
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
}

public ArrayList getAllMessagesP(String e) {
    ArrayList list = new ArrayList();
    try {
        establishConnection();
        String sql = "select sentBy, max(time) as time1\n"
            + "from messages where sentTo = ?\n"
            + "group by sentBy order by max(time) desc;";
        PreparedStatement pStmt = con.prepareStatement(sql);
        pStmt.setString(1, e);
        ResultSet rs = pStmt.executeQuery();
        while (rs.next()) {
            String u = rs.getString("sentBy");
            list.add(u);
        }
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    }
}

```



```

    } finally {
        closeCon();
    }
    return list;
}

public double getLatestMsg(String st, String sb) {
    double mi = 0;
    try {
        establishConnection();
        String sql = "SELECT max(msgId) AS msgId1, max(time) as time1\n"
            + "FROM messages\n"
            + "WHERE (sentTo = ? AND sentBy = ?) OR (sentTo = ? AND sentBy = ?)";
        PreparedStatement pStmt = con.prepareStatement(sql);
        pStmt.setString(1, st);
        pStmt.setString(2, sb);
        pStmt.setString(3, sb);
        pStmt.setString(4, st);
        ResultSet rs = pStmt.executeQuery();
        if (rs.next()) {
            mi = rs.getDouble("msgId1");
        }
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
    return mi;
}

public person getMsgP(double mi) throws SQLException {
    person p = null;
    try {
        establishConnection();
        String sql = "SELECT * FROM messages WHERE msgId = ?";
        PreparedStatement pst = con.prepareStatement(sql);
        pst.setDouble(1, mi);
        ResultSet rs = pst.executeQuery();
        if (rs.next()) {
            p = new person();
            p.setPostId(mi);
            p.setPost(rs.getString("message"));
            p.setTime(rs.getString("time"));
            p.setEmailId(rs.getString("sentBy"));
        }
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
    return p;
}

```

```

public ArrayList getMessages(String u, String uv) {
    ArrayList list = new ArrayList();
    try {
        establishConnection();
        String sql = "SELECT * \n"
            + "FROM messages \n"
            + "WHERE (sentTo = ? AND sentBy = ?) \n"
            + "OR (sentTo = ? AND sentBy = ?)\n"
            + "ORDER BY time DESC";
        PreparedStatement pstmt = con.prepareStatement(sql);
        pstmt.setString(1, u);
        pstmt.setString(2, uv);
        pstmt.setString(3, uv);
        pstmt.setString(4, u);
        ResultSet rs = pstmt.executeQuery();
        while (rs.next()) {
            person p = new person();
            p.setEmailId(rs.getString("sentBy"));
            p.setPostId(Double.parseDouble(rs.getString("msgId")));
            p.setPost(rs.getString("message"));
            p.setTime(rs.getString("time"));
            list.add(p);
        }
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
    return list;
}

public void delAllMsgs(String st, String sb) {
    try {
        establishConnection();
        String sql = "DELETE \n"
            + "FROM messages \n"
            + "WHERE (sentTo = ? AND sentBy = ?) \n"
            + "OR (sentTo = ? AND sentBy = ?)";
        PreparedStatement pstmt = con.prepareStatement(sql,
            ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
        pstmt.setString(1, st);
        pstmt.setString(2, sb);
        pstmt.setString(3, sb);
        pstmt.setString(4, st);
        ResultSet rs = pstmt.executeQuery();
        while (rs.next()) {
            rs.updateRow();
        }
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
    }
}

```

```

        closeCon();
    }
}
public void delMsg(double mid) {
    try {
        establishConnection();
        String sql = "DELETE FROM messages WHERE msgId = ?";
        PreparedStatement pStmt = con.prepareStatement(sql,
ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
        pStmt.setDouble(1, mid);
        ResultSet rs = pStmt.executeQuery();
        if (rs.next()) {
            rs.updateRow();
        }
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
}
public int getUnreadMessages(String e) {
    int c = 0;
    try {
        establishConnection();
        String sql = "SELECT DISTINCT sentBy\n"
            + " FROM messages\n"
            + "WHERE sentTo = ?\n"
            + " AND status=0;";
        PreparedStatement pStmt = con.prepareStatement(sql);
        pStmt.setString(1, e);
        ResultSet rs = pStmt.executeQuery();
        while (rs.next()) {
            c++;
        }
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
    return c;
}
public int getUnreadMessagesP(String st, String sb) {
    int c = 0;
    try {
        establishConnection();
        String sql = "SELECT *\n"
            + " FROM messages\n"
            + "WHERE (sentTo = ? AND sentBy = ?)\n"
            + " AND status = ? ";
        PreparedStatement pStmt = con.prepareStatement(sql);

```

```

        pstmt.setString(1, st);
        pstmt.setString(2, sb);
        pstmt.setBoolean(3, false);
        ResultSet rs = pstmt.executeQuery();
        while (rs.next()) {
            c++;
        }
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
    return c;
}

public void markMsgsAsRead(String u, String uv) {
    try {
        establishConnection();
        String sql = "UPDATE messages\n"
            + "SET status = true\n"
            + "WHERE (sentTo = ? AND sentBy = ?) AND status = false;";
        PreparedStatement pstmt = con.prepareStatement(sql);
        pstmt.setString(1, u);
        pstmt.setString(2, uv);
        int r = pstmt.executeUpdate();
        System.out.println(r + " rows updated!");
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
}

public void deleteAccount(String e) {
    try {
        establishConnection();
        String sql = "DELETE FROM person WHERE EmailAddress =?";
        PreparedStatement pstmt = con.prepareStatement(sql,
            ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
        pstmt.setString(1, e);
        pstmt.executeUpdate();
        String sql1 = "DELETE FROM profileVisits WHERE EmailAddress = ? OR
visitedBy = ?";
        PreparedStatement pstmt1 = con.prepareStatement(sql1);
        pstmt1.setString(1, e);
        pstmt1.setString(2, e);
        int r = pstmt1.executeUpdate();
        System.out.println(r + " rows updated of 'profileVisits'!");
        String sql2 = "DELETE FROM posts WHERE EmailAddress = ?";
        PreparedStatement pstmt2 = con.prepareStatement(sql2);
        pstmt2.setString(1, e);
        r = pstmt2.executeUpdate();
    }
}

```

```

        System.out.println(r + " rows updated of 'posts!'");
        String sql3 = "DELETE FROM like WHERE likedby = ?";
        PreparedStatement pStmt3 = con.prepareStatement(sql3);
        pStmt3.setString(1, e);
        r = pStmt3.executeUpdate();
        System.out.println(r + " rows updated of 'like!'");
        String sql4 = "DELETE FROM friends WHERE EmailAddress = ? OR friend_id =
?";
        PreparedStatement pStmt4 = con.prepareStatement(sql4);
        pStmt4.setString(1, e);
        pStmt4.setString(2, e);
        r = pStmt4.executeUpdate();
        System.out.println(r + " rows updated of 'friends!'");
        String sql5 = "DELETE FROM comment WHERE commentby = ?";
        PreparedStatement pStmt5 = con.prepareStatement(sql5);
        pStmt5.setString(1, e);
        r = pStmt5.executeUpdate();
        System.out.println(r + " rows updated of 'comment!'");
        String sql6 = "DELETE FROM messages WHERE sentTo = ? OR sentBy= ?";
        PreparedStatement pStmt6 = con.prepareStatement(sql6);
        pStmt6.setString(1, e);
        pStmt6.setString(2, e);
        r = pStmt6.executeUpdate();
        System.out.println(r + " rows updated of 'messages!'");
        String sql7 = "DELETE FROM friendRequest WHERE EmailAddress = ? OR
requestId = ?";
        PreparedStatement pStmt7 = con.prepareStatement(sql7);
        pStmt7.setString(1, e);
        pStmt7.setString(2, e);
        r = pStmt7.executeUpdate();
        System.out.println(r + " rows updated of 'friendRequest!'");
    } catch (ClassNotFoundException | SQLException ex) {
        System.out.println(ex);
    } finally {
        closeCon();
    }
}
}
}

```

## Add Cover and DP (Private and Public)

```

import daos.personDAO;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.SQLException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

```

```
import javax.servlet.http.HttpSession;
public class addCrDp extends HttpServlet {
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        HttpSession session = request.getSession(true);
        String pro = (String) session.getAttribute("UploadMsg");
        String ppro = (String) session.getAttribute("UploadMsgp");
        String cov = (String) session.getAttribute("UploadMsg1");
        String pcov = (String) session.getAttribute("UploadMsgp1");
        String eml = (String) session.getAttribute("UserRequest");
        try {
            personDAO pdao = new personDAO();
            pdao.addCrDp(pro, cov, ppro, pcov, eml);
            session.setAttribute("UserLoggedIn", eml);
            response.sendRedirect("Home.jsp");

        } catch (ClassNotFoundException | SQLException ex) {
            Logger.getLogger(joinnow.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        processRequest(request, response);
    }
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        processRequest(request, response);
    }
}
```

## Uploading Pictures (Cover, DP, Post)

```
import daos.personDAO;
import java.awt.image.BufferedImage;
import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.util.*;
import javax.imageio.ImageIO;
import javax.servlet.http.HttpSession;
import org.apache.commons.fileupload.FileItem;
```

```
import org.apache.commons.fileupload.disk.DiskFileItemFactory;
import org.apache.commons.fileupload.servlet.ServletFileUpload;
public class UploadDP extends HttpServlet {
    private boolean isMultipart;
    private String filePath;
    private final int maxFileSize = 5000 * 1024;
    private final int maxMemSize = 5000 * 1024;
    private File file;
    @Override
    public void init() {
        // Get the file location where it would be stored.
        filePath = getServletContext().getInitParameter("file-upload");
    }
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        HttpSession session = request.getSession(true);
        String user = (String) session.getAttribute("UserLoggedIn");
        PrintWriter out = response.getWriter();
        String msg;
        // Check that we have a file upload request
        isMultipart = ServletFileUpload.isMultipartContent(request);
        if (!isMultipart) {
            msg = "No file uploaded.";
        }
        DiskFileItemFactory factory = new DiskFileItemFactory();
        // maximum size that will be stored in memory
        factory.setSizeThreshold(maxMemSize);
        // Location to save data that is larger than maxMemSize.
        factory.setRepository(new File("d:\\temp"));
        // Create a new file upload handler
        ServletFileUpload upload = new ServletFileUpload(factory);
        // maximum file size to be uploaded.
        upload.setSizeMax(maxFileSize);
        try {
            // Parse the request to get file items.
            List fileItems = upload.parseRequest(request);

            // Process the uploaded file items
            Iterator i = fileItems.iterator();
            while (i.hasNext()) {
                FileItem fi = (FileItem) i.next();
                if (!fi.isFormField()) {
                    // Get the uploaded file parameters
                    String fieldName = fi.getFieldName();
                    String fileName = fi.getName();
                    //String contentType = fi.getContentType();
                    //boolean isInMemory = fi.isInMemory();
                    long sizeInBytes = fi.getSize();
                    BufferedInputStream is = new BufferedInputStream(fi.getInputStream());
```

```

BufferedImage bimg = ImageIO.read(is);
int w = bimg.getWidth();
int h = bimg.getHeight();
// Write the file
if (fileName.lastIndexOf("\\") >= 0) {
    file = new File(filePath
        + fileName.substring(fileName.lastIndexOf("\\")));
} else {
    file = new File(filePath
        + fileName.substring(fileName.lastIndexOf("\\") + 1));
}
//fi.write(file);
if (null != fieldName) {
    switch (fieldName) {
        case "editPP":
            fi.write(file);
            personDAO pdao = new personDAO();
            pdao.addCrDp(null, null, fileName, null, user);
            session.setAttribute("EditPP", "Public Profile Photo Successfully
changed!");
            response.sendRedirect("editProfile.jsp");
            break;
        case "editPC":
            if (w >=800 && h > 150) {
                fi.write(file);
                personDAO pdao1 = new personDAO();
                pdao1.addCrDp(null, null, null, fileName, user);
                session.setAttribute("EditPC", "Public Cover Photo Successfully
changed!");
            } else {
                session.setAttribute("pubCovErr", "Your cover photo should be at least
800 wide and it's height should be between 150 and 800!");
            }
            response.sendRedirect("editProfile.jsp");
            break;

        case "file1":
            fi.write(file);
            session.setAttribute("UploadMsg", fileName);
            response.sendRedirect("profile.jsp");
            break;
        case "filep1":
            fi.write(file);
            session.setAttribute("UploadMsgp", fileName);
            response.sendRedirect("profile.jsp");
            break;
        case "file2":
            if (w >=800 && h > 150) {
                fi.write(file);
                session.setAttribute("UploadMsg1", fileName);
            }
    }
}

```



```

        } else {
            session.setAttribute("priCovErr", "Your cover photo should be at least
800 wide and it's height should be between 150 and 800!");
        }
        response.sendRedirect("profile.jsp");
        break;
    case "filep2":
        if (w >=800 && h > 150 ) {
            fi.write(file);
            session.setAttribute("UploadMsgp1", fileName);
        } else {
            session.setAttribute("pubCovErr", "Your cover photo should be at least
800 wide and it's height should be between 150 and 800!");
        }
        response.sendRedirect("profile.jsp");
        break;
    case "file4":
        fi.write(file);
        session.setAttribute("UploadMsg", fileName);
        response.sendRedirect("editProfile.jsp");
        break;
    case "file3":
        if (w >=800 && h > 150) {
            fi.write(file);
            session.setAttribute("UploadMsg1", fileName);
        } else {
            session.setAttribute("pubCovErr", "Your cover photo should be at least
800 wide and it's height should be between 150 and 800!");
        }
        response.sendRedirect("editProfile.jsp");
        break;
    case "postPic1":
        fi.write(file);
        session.setAttribute("PostPic", fileName);
        response.sendRedirect("timeline.jsp");
        break;
    case "postPic2":
        fi.write(file);
        session.setAttribute("PostPic", fileName);
        response.sendRedirect("Home.jsp");
        break;
    }
}
}
}
} catch (Exception ex) {
    System.out.println(ex);
}
}
@Override

```

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    processRequest(request, response);
}
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    processRequest(request, response);
}
```

## Sorting Timelines

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
public class sort extends HttpServlet {
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        HttpSession session = request.getSession(true);
        String time = request.getParameter("byTime");
        String Ltime = request.getParameter("LbyTime");
        String name = request.getParameter("byName");
        if (time != null) {
            session.setAttribute("Sort", "time");
            response.sendRedirect("timeline.jsp");
        } else if (name != null) {
            session.setAttribute("Sort", "name");
            response.sendRedirect("timeline.jsp");
        } else if (Ltime != null) {
            session.setAttribute("Sort", "time");
            response.sendRedirect("mylikes.jsp");
        } else {
            session.setAttribute("Sort", "name");
            response.sendRedirect("mylikes.jsp");
        }
    }
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        processRequest(request, response);
    }
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
```

```
    processRequest(request, response);  
}
```

## Profile Visit

```
import beans.person;  
import daos.personDAO;  
import java.io.IOException;  
import java.io.PrintWriter;  
import java.sql.SQLException;  
import java.util.logging.Level;  
import java.util.logging.Logger;  
import javax.servlet.ServletException;  
import javax.servlet.http.HttpServlet;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import javax.servlet.http.HttpSession;  
public class viewProfile extends HttpServlet {  
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        response.setContentType("text/html;charset=UTF-8");  
        PrintWriter out = response.getWriter();  
        HttpSession session = request.getSession(true);  
        String getUser = request.getParameter("getEmail");  
        String user = (String) session.getAttribute("UserLoggedIn");  
        //String view = (String) session.getAttribute("viewPro");  
        String view = request.getParameter("viewPro");  
        personDAO pdao;  
        try {  
            pdao = new personDAO();  
            person p;  
            if (user == null) {  
                session.setAttribute("Check", "Sign in to continue..");  
                response.sendRedirect("SignIn.jsp");  
            } else {  
                if (view != null) {  
                    session.setAttribute("viewPosts", view);  
                    p = pdao.getPersonP(view);  
                    session.setAttribute("viewUser", p);  
                    if (user.equals(view)) {  
                        response.sendRedirect("Home.jsp");  
                    } else {  
                        long time = System.currentTimeMillis();  
                        java.sql.Timestamp date = new java.sql.Timestamp(time);  
                        pdao.addVisits(user, view, date);  
                        response.sendRedirect("viewprofile.jsp");  
                    }  
                } else {  
                    session.setAttribute("viewPosts", getUser);  
                    p = pdao.getPersonP(getUser);  
                }  
            }  
        } catch (SQLException ex) {  
            Logger.getLogger(viewProfile.class.getName()).log(Level.SEVERE, null, ex);  
        }  
    }  
}
```

```

        session.setAttribute("viewUser", p);
        if (user.equals(getUser)) {
            response.sendRedirect("Home.jsp");
        } else {
            long time = System.currentTimeMillis();
            java.sql.Timestamp date = new java.sql.Timestamp(time);
            pdao.addVisits(user, getUser, date);
            response.sendRedirect("viewprofile.jsp");
        }
    }
}
} catch (ClassNotFoundException | SQLException ex) {
    Logger.getLogger(viewProfile.class.getName()).log(Level.SEVERE, null, ex);
}
}
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    processRequest(request, response);
}
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    processRequest(request, response);
}
}

```

## Sending Email/Requesting Password

```

import daos.personDAO;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.SQLException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.util.*;
import javax.mail.*;
import javax.mail.internet.*;
import javax.servlet.http.HttpSession;
public class forgotPassword extends HttpServlet {
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        HttpSession sess = request.getSession(true);
        String password = null;
        String email = null;
    }
}

```

```

boolean emailExists;
PrintWriter out = response.getWriter();
try {
    email = request.getParameter("forPass");
    //out.println(email);
    personDAO pdao = new personDAO();
    emailExists = pdao.verifyEmail(email);
    if (emailExists == false) {
        sess.setAttribute("ConfirmMsg1", "Email doesn't exist!");
        response.sendRedirect("forgotPassword.jsp");
    }
    password = pdao.sendPassword(email);
    //out.println(password);
} catch (ClassNotFoundException | SQLException ex) {
    Logger.getLogger(forgotPassword.class.getName()).log(Level.SEVERE, null, ex);
}
//Sender's email ID needs to be mentioned
String from = "*****";
String pass = "*****";
//Recipient's email ID needs to be mentioned.
String to = email;
String host = "smtp.mail.yahoo.com";
// Get system properties
Properties properties = System.getProperties();
// Setup mail server
properties.put("mail.smtp.starttls.enable", "true");
properties.put("mail.smtp.host", host);
properties.put("mail.smtp.user", from);
properties.put("mail.smtp.password", pass);
properties.put("mail.smtp.port", "587");
properties.put("mail.smtp.auth", "true");
// Get the default Session object.
Session session = Session.getDefaultInstance(properties);
try {
    // Create a default MimeMessage object.
    MimeMessage message = new MimeMessage(session);

    // Set From: header field of the header.
    message.setFrom(new InternetAddress(from));
    // Set To: header field of the header.
    message.addRecipient(Message.RecipientType.TO,
        new InternetAddress(to));
    // Set Subject: header field
    message.setSubject("Request for Password!");
    // Now set the actual message
    message.setText("Your password is: " + password);
    // Send message
    Transport transport = session.getTransport("smtp");
    transport.connect(host, from, pass);
    transport.sendMessage(message, message.getAllRecipients());
}

```

```
        transport.close();
        String msg = "Password sent. Check your mail!";
        sess.setAttribute("ConfirmMsg", msg);
        response.sendRedirect("forgotPassword.jsp");
    } catch (MessagingException mex) {
        mex.printStackTrace();
    }
}

@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    processRequest(request, response);
}

@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    processRequest(request, response);
}
```

# **CHAPTER 7**

## **TESTING**

## **7.1. Testing:**

Software testing is an analysis conducted to provide information about quality of product with respect to the context in which it is intended to operate. Testing is the process of executing program with the intent of finding an error.

## **7.2. Testing Objectives:**

Testing should systematically uncover different classes of errors in a minimum amount of time and with a minimum amount of effort. A secondary benefit of testing is that it demonstrates that the software appears to be working as stated in the specifications. The data collected through testing can also provide an indication of the software's reliability and quality. But, testing cannot show the absence of defect -- it can only show that software defects are present.

## **7.3. Types of Testing:**

There are two basics of software testing: blackbox testing and whitebox testing.

### **7.3.1. Blackbox Testing**

Black box testing is a testing technique that ignores the internal mechanism of the system and focuses on the output generated against any input and execution of the system. It is also called functional testing.

### **7.3.2. Whitebox Testing**

White box testing is a testing technique that takes into account the internal mechanism of a system. It is also called structural testing and glass box testing.

Black box testing is often used for validation and white box testing is often used for verification.



## 7.4. Test Cases:

### 7.4.1. Sign Up

Test Case ID:	TC-001	
Associated Use Case :	UC-001:Sign Up	
Functionality to be Tested:	Create account for user.	
Actor:	User	
Pre-conditions :	None	
Input Data:	First Name Last Name Email Address Password	
Normal Flow	Expected Results	Actual Results
1. User enters all required information and clicks on sign up button.	System saves all information and redirects user to next form.	Pass
Alternate Flow	Expected Results	Actual Results
1. User enters invalid email id.	System displays an error message	
2. User enters short password.	System displays an error message	
Date:		
Tester Name:	Deeba Mehboob, Javeria Akbar	

### 7.4.2. Sign In

<b>Test Case ID:</b>	TC-002	
<b>Associated Use Case :</b>	UC-002:Sign In	
<b>Functionality to be Tested:</b>	Sign in.	
<b>Actor:</b>	User	
<b>Pre-conditions :</b>	Sign up	
<b>Input Data:</b>	Email Address Password	
<b>Normal Flow</b>	<b>Expected Results</b>	<b>Actual Results</b>
1. User enters email address and password and clicks on sign in button.	System verifies email id and password and redirects user to timeline.	Pass
<b>Alternate Flow</b>	<b>Expected Results</b>	<b>Actual Results</b>
1. User enters wrong email address and/or password.	System displays an error message	
<b>Date:</b>		
<b>Tester Name:</b>	Deeba Mehboob, Javeria Akbar	

### 7.4.3. Sign Out

<b>Test Case ID:</b>	TC-003	
<b>Associated Use Case :</b>	UC-003:Sign Out	
<b>Functionality to be Tested:</b>	User signs out.	
<b>Actor:</b>	User	
<b>Pre-conditions :</b>	User should have logged in.	
<b>Input Data:</b>	None	
<b>Normal Flow</b>	<b>Expected Results</b>	<b>Actual Results</b>
1. User clicks on sign out button.	System signs out user.	Pass
<b>Alternate Flow</b>	<b>Expected Results</b>	<b>Actual Results</b>
1. User does not click on sign out button.	System doesn't sign out user.	
<b>Date:</b>		
<b>Tester Name:</b>	Deeba Mehboob, Javeria Akbar	

#### 7.4.4. Request Password

<b>Test Case ID:</b>	TC-004	
<b>Associated Use Case :</b>	UC-005:Request Password	
<b>Functionality to be Tested:</b>	Retrieving password.	
<b>Actor:</b>	User	
<b>Pre-conditions :</b>	User should have an account.	
<b>Input Data:</b>	Email Address	
<b>Normal Flow</b>	<b>Expected Results</b>	<b>Actual Results</b>
1. User clicks on forgot password link.	System shows forgot password form.	Pass
2. User enters email address and clicks on request button.	System sends password on entered email address and shows confirmation message.	Pass
<b>Alternate Flow</b>	<b>Expected Results</b>	<b>Actual Results</b>
1. User does not clicks on required link.	System does not show forgot password form.	
2. User enters email id that has no account associated with it	System displays an error message	
<b>Date:</b>		
<b>Tester Name:</b>	Deeba Mehboob, Javeria Akbar	

### 7.4.5. Delete Account

Test Case ID:	TC-005	
Associated Use Case :	UC-006:Delete Account	
Functionality to be Tested:	Deleting user account.	
Actor:	User	
Pre-conditions :	User should be logged in.	
Input Data:	None	
Normal Flow	Expected Results	Actual Results
1. User clicks on delete your account button.	System asks for confirmation.	Pass
2. User confirms deletion.	System deletes account, shows confirmation message and redirects user to welcome page.	Pass
Alternate Flow	Expected Results	Actual Results
1. User does not click on required button.	System does not asks for confirmation.	
2. User cancels confirmation.	System does not delete account.	
Date:		
Tester Name:	Deeba Mehboob, Javeria Akbar	

### 7.4.7. Edit Password

<b>Test Case ID:</b>	TC-007	
<b>Associated Use Case :</b>	UC-008:Edit Password	
<b>Functionality to be Tested:</b>	Changing password.	
<b>Actor:</b>	User	
<b>Pre-conditions :</b>	User should be logged in and is on edit profile page.	
<b>Input Data:</b>	Old password New password	
<b>Normal Flow</b>	<b>Expected Results</b>	<b>Actual Results</b>
1. User enters old and new passwords and clicks on save password button.	System save changes and shows confirmation message.	Pass
<b>Alternate Flow</b>	<b>Expected Results</b>	<b>Actual Results</b>
1. User enters wrong old or new password.	System displays error message.	
2. User doesn't click on save button.	System doesn't save new password.	
<b>Date:</b>		
<b>Tester Name:</b>	Deeba Mehboob, Javeria Akbar	

### 7.4.7. Add Post

Test Case ID:	TC-007	
Associated Use Case :	UC-011:Add Post	
Functionality to be Tested:	Adding new post.	
Actor:	User	
Pre-conditions :	User should be logged in and is on home/my-posts page.	
Input Data:	Post Image	
Normal Flow	Expected Results	Actual Results
1. User fills in post form and/or uploads image.	System shows uploaded image.	Pass
2. User clicks on post button.	System saves and shows new post.	Pass
Alternate Flow	Expected Results	Actual Results
1. User doesn't upload image.	System doesn't show image.	
2. User clicks on close button.	System removes image.	
3. User doesn't click on post button.	System doesn't save post.	
Date:		
Tester Name:	Deeba Mehboob, Javeria Akbar	

**7.4.8. Edit Post**

Test Case ID:	TC-008	
Associated Use Case :	UC-012: Edit Post	
Functionality to be Tested:	Editing a post.	
Actor:	User	
Pre-conditions :	User should be logged in and is on my-posts page.	
Input Data:	Post	
Normal Flow	Expected Results	Actual Results
1. User clicks on edit button of post.	System shows edit post form to user.	Pass
2. User changes text and clicks on edit post button.	System saves changed text.	Pass
Alternate Flow	Expected Results	Actual Results
1. User doesn't click on edit button.	System doesn't show edit post form.	
2. User doesn't clicks on edit post button.	System doesn't save changed text.	
Date:		
Tester Name:	Deeba Mehboob, Javeria Akbar	



### 7.4.9. Delete Post

<b>Test Case ID:</b>	TC-009	
<b>Associated Use Case :</b>	UC-013: Delete Post	
<b>Functionality to be Tested:</b>	Deleting a post.	
<b>Actor:</b>	User	
<b>Pre-conditions :</b>	User should be logged in and is on my-posts page.	
<b>Input Data:</b>	None	
<b>Normal Flow</b>	<b>Expected Results</b>	<b>Actual Results</b>
1. User clicks on delete button of post.	System shows message to confirm deletion.	Pass
2. User confirms deletion.	System deletes post.	Pass
<b>Alternate Flow</b>	<b>Expected Results</b>	<b>Actual Results</b>
1. User doesn't click on delete button.	System doesn't show message to confirm.	
2. User clicks on cancel button.	System doesn't delete post.	
<b>Date:</b>		
<b>Tester Name:</b>	Deeba Mehboob, Javeria Akbar	

#### 7.4.10. Comment on a post

<b>Test Case ID:</b>	TC-010	
<b>Associated Use Case :</b>	UC-015: Comment on Post	
<b>Functionality to be Tested:</b>	Commenting on a post.	
<b>Actor:</b>	User	
<b>Pre-conditions :</b>	User should be logged in and is on my-posts/my-likes/home/view-profile page.	
<b>Input Data:</b>	Comment	
<b>Normal Flow</b>	<b>Expected Results</b>	<b>Actual Results</b>
1. User clicks on comment button of post.	System shows comment form to user.	Pass
2. User fills in form and clicks on comment button.	System saves comment.	Pass
<b>Alternate Flow</b>	<b>Expected Results</b>	<b>Actual Results</b>
1. User doesn't click on comment button of post.	System doesn't show comment form.	
2. User doesn't clicks on comment button.	System doesn't save comment.	
<b>Date:</b>		
<b>Tester Name:</b>	Deeba Mehboob, Javeria Akbar	

#### 7.4.11. View Friend Requests

<b>Test Case ID:</b>	TC-011	
<b>Associated Use Case :</b>	UC-019: View Friend Requests	
<b>Functionality to be Tested:</b>	Viewing friend requests.	
<b>Actor:</b>	User	
<b>Pre-conditions :</b>	User should be logged in.	
<b>Input Data:</b>	None	
<b>Normal Flow</b>	<b>Expected Results</b>	<b>Actual Results</b>
1. User clicks on friends request link.	System displays list of friend requests.	Pass
<b>Alternate Flow</b>	<b>Expected Results</b>	<b>Actual Results</b>
1. User doesn't click on desired link.	System doesn't show friend requests.	
<b>Date:</b>		
<b>Tester Name:</b>	Deeba Mehboob, Javeria Akbar	

#### 7.4.12. Accept Friend Request

<b>Test Case ID:</b>	TC-012	
<b>Associated Use Case :</b>	UC-020: Accept Friend Request	
<b>Functionality to be Tested:</b>	Accepting a friend request.	
<b>Actor:</b>	User	
<b>Pre-conditions :</b>	User should be logged in and is on friends request page.	
<b>Input Data:</b>	None	
<b>Normal Flow</b>	<b>Expected Results</b>	<b>Actual Results</b>
1. User clicks accept button of a friend request.	System removes that request and add that user as a friend.	Pass
<b>Alternate Flow</b>	<b>Expected Results</b>	<b>Actual Results</b>
1. User doesn't click on accept button.	System doesn't remove request and doesn't add that user as friend.	
<b>Date:</b>		
<b>Tester Name:</b>	Deeba Mehboob, Javeria Akbar	

### 7.4.13. Accept Friend Request

<b>Test Case ID:</b>	TC-013	
<b>Associated Use Case :</b>	UC-021: Cancel Friend Request	
<b>Functionality to be Tested:</b>	Cancelling a friend request.	
<b>Actor:</b>	User	
<b>Pre-conditions :</b>	User should be logged in and is on friends request page.	
<b>Input Data:</b>	None	
<b>Normal Flow</b>	<b>Expected Results</b>	<b>Actual Results</b>
1. User clicks cancel button of a friend request.	System removes that request.	Pass
<b>Alternate Flow</b>	<b>Expected Results</b>	<b>Actual Results</b>
1. User doesn't click on accept button.	System doesn't remove request.	
<b>Date:</b>		
<b>Tester Name:</b>	Deeba Mehboob, Javeria Akbar	

#### 7.4.14. Add Friend

<b>Test Case ID:</b>	TC-014	
<b>Associated Use Case :</b>	UC-022: Add Friend	
<b>Functionality to be Tested:</b>	Adding a friend (Sending a friend request).	
<b>Actor:</b>	User	
<b>Pre-conditions :</b>	User should be logged in and is on profile of user to be sent request.	
<b>Input Data:</b>	None	
<b>Normal Flow</b>	<b>Expected Results</b>	<b>Actual Results</b>
1. User clicks on add friend button.	System sends request and shows requested button instead.	Pass
<b>Alternate Flow</b>	<b>Expected Results</b>	<b>Actual Results</b>
1. User doesn't click on add friend button.	System doesn't send request.	
<b>Date:</b>		
<b>Tester Name:</b>	Deeba Mehboob, Javeria Akbar	

**7.4.15. Delete Friend**

Test Case ID:	TC-015	
Associated Use Case :	UC-023: Delete Friend	
Functionality to be Tested:	Deleting a friend (Unfriending a friend).	
Actor:	User	
Pre-conditions :	User should be logged in and is on profile of user to be unfriended.	
Input Data:	None	
Normal Flow	Expected Results	Actual Results
1. User clicks on friend button.	System shows message to confirm.	Pass
2. User confirms unfriending.	System removes that user as friend and shows add friend button instead.	Pass
Alternate Flow	Expected Results	Actual Results
1. User doesn't click on friend button.	System doesn't remove user as friend.	
2. User clicks cancel button.	System doesn't remove user as friend.	
Date:		
Tester Name:	Deeba Mehboob, Javeria Akbar	

#### 7.4.16. Reset Profile Visits

<b>Test Case ID:</b>	TC-016	
<b>Associated Use Case :</b>	UC-026: Reset Profile Visits	
<b>Functionality to be Tested:</b>	Clearing profile visits.	
<b>Actor:</b>	User	
<b>Pre-conditions :</b>	User should be logged in and has opened profile visits link.	
<b>Input Data:</b>	None	
<b>Normal Flow</b>	<b>Expected Results</b>	<b>Actual Results</b>
1. User clicks on reset button.	System removes all profile visits and shows message, 'No profile visits'	Pass
<b>Alternate Flow</b>	<b>Expected Results</b>	<b>Actual Results</b>
1. User doesn't click on reset button.	System doesn't remove profile visits.	
<b>Date:</b>		
<b>Tester Name:</b>	Deeba Mehboob, Javeria Akbar	



**7.4.17. Send Message**

Test Case ID:	TC-017	
Associated Use Case :	UC-027: Send Message	
Functionality to be Tested:	Sending a message.	
Actor:	User	
Pre-conditions :	User should be logged in and has opened profile of that user whom he wants to send message.	
Input Data:	Message	
Normal Flow	Expected Results	Actual Results
1. User clicks on message button.	System displays message form.	Pass
2. User enters message and clicks on send button.	System sends message and shows confirmation message.	Pass
Alternate Flow	Expected Results	Actual Results
1. User doesn't click on message button.	System doesn't display message form.	
2. User clicks on cancel button.	System doesn't send message and hides message form.	
Date:		
Tester Name:	Deeba Mehboob, Javeria Akbar	

#### 7.4.18. View Messages

<b>Test Case ID:</b>	TC-018	
<b>Associated Use Case :</b>	UC-028: View Messages	
<b>Functionality to be Tested:</b>	Viewing messages.	
<b>Actor:</b>	User	
<b>Pre-conditions :</b>	User should be logged in.	
<b>Input Data:</b>	None	
<b>Normal Flow</b>	<b>Expected Results</b>	<b>Actual Results</b>
1. User clicks on messages link.	System displays list of message senders.	Pass
2. User clicks on view all button of a particular sender.	System displays messages of that user.	Pass
<b>Alternate Flow</b>	<b>Expected Results</b>	<b>Actual Results</b>
1. User doesn't click on messages link.	System doesn't display message senders.	
2. User doesn't click on view all button.	System doesn't display messages of particular user.	
<b>Date:</b>		
<b>Tester Name:</b>	Deeba Mehboob, Javeria Akbar	

#### 7.4.19. Reply to messages

<b>Test Case ID:</b>	TC-019	
<b>Associated Use Case :</b>	UC-029: Reply to messages	
<b>Functionality to be Tested:</b>	Replying to messages.	
<b>Actor:</b>	User	
<b>Pre-conditions :</b>	User should be logged in and has opened messages of that user whom he wants to reply.	
<b>Input Data:</b>	Reply	
<b>Normal Flow</b>	<b>Expected Results</b>	<b>Actual Results</b>
1. User enters reply and click on reply button.	System sends reply and shows it to user.	Pass
<b>Alternate Flow</b>	<b>Expected Results</b>	<b>Actual Results</b>
1. User doesn't click on reply button.	System doesn't send reply.	
<b>Date:</b>		
<b>Tester Name:</b>	Deeba Mehboob, Javeria Akbar	

### 7.4.20. Delete Messages

Test Case ID:	TC-020	
Associated Use Case :	UC-030: Delete Messages	
Functionality to be Tested:	Deleting messages.	
Actor:	User	
Pre-conditions :	User should be logged in and has opened messages.	
Input Data:	None	
Normal Flow	Expected Results	Actual Results
1. User clicks on delete all button of that user whose messages he wants to delete.	System asks user to confirm.	Pass
2. User clicks on ok button.	System deletes all messages with that user.	Pass
Alternate Flow	Expected Results	Actual Results
1. User doesn't click on delete all button.	System doesn't ask for confirmation.	
2. User clicks on cancel button.	System don't delete messages.	
Date:		
Tester Name:	Deeba Mehboob, Javeria Akbar	

**7.4.21. Delete Message**

<b>Test Case ID:</b>	TC-021	
<b>Associated Use Case :</b>	UC-030: Delete Messages	
<b>Functionality to be Tested:</b>	Deleting a message.	
<b>Actor:</b>	User	
<b>Pre-conditions :</b>	User should be logged in and has opened messages of that user whose messages he wants to delete.	
<b>Input Data:</b>	None	
<b>Normal Flow</b>	<b>Expected Results</b>	<b>Actual Results</b>
1. User clicks on delete button of that message which he wants to delete.	System asks user to confirm.	Pass
2. User clicks on ok button.	System deletes that message.	Pass
<b>Alternate Flow</b>	<b>Expected Results</b>	<b>Actual Results</b>
1. User doesn't click on delete button.	System doesn't ask for confirmation.	
2. User clicks on cancel button.	System doesn't delete that message.	
<b>Date:</b>		
<b>Tester Name:</b>	Deeba Mehboob, Javeria Akbar	

**7.4.22. Search**

<b>Test Case ID:</b>	TC-022	
<b>Associated Use Case :</b>	UC-031: Search	
<b>Functionality to be Tested:</b>	Searching for users.	
<b>Actor:</b>	User	
<b>Pre-conditions :</b>	User should be logged in.	
<b>Input Data:</b>	First Name Last Name	
<b>Normal Flow</b>	<b>Expected Results</b>	<b>Actual Results</b>
1. User enters first name and last name and clicks on search button.	System displays all users with entered first name and last name.	Pass
2. User enters only first name and clicks on search button.	System displays all users with entered first name.	Pass
3, User enters only last name and clicks on search button.	System displays all users with last name.	Pass
<b>Alternate Flow</b>	<b>Expected Results</b>	<b>Actual Results</b>
1. User doesn't click on search button.	System don't display results.	
2. If no users are found.	System display message. 'No users found'	
<b>Date:</b>		
<b>Tester Name:</b>	Deeba Mehboob, Javeria Akbar	

### 7.4.23. View Profile

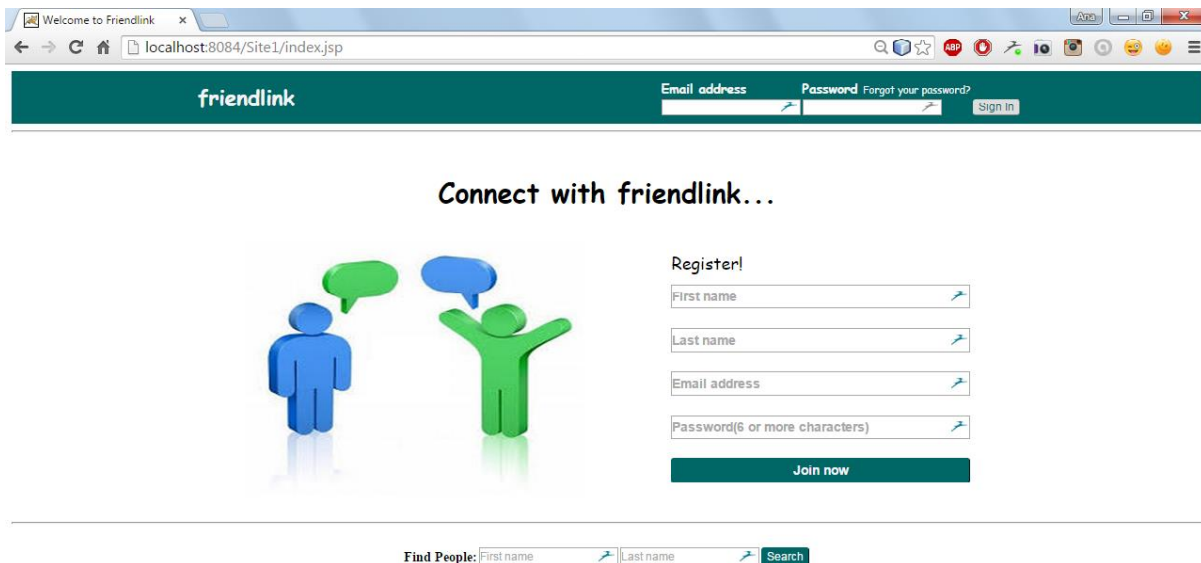
<b>Test Case ID:</b>	TC-023	
<b>Associated Use Case :</b>	UC-032: View Profile	
<b>Functionality to be Tested:</b>	Viewing user profile.	
<b>Actor:</b>	User	
<b>Pre-conditions :</b>	User has logged in and is on home/my-likes/friends/friend-requests/profile-visits/messages/search-results page.	
<b>Input Data:</b>	None	
<b>Normal Flow</b>	<b>Expected Results</b>	<b>Actual Results</b>
1. User clicks on name of user.	System displays profile of that user.	Pass
<b>Alternate Flow</b>	<b>Expected Results</b>	<b>Actual Results</b>
1. User doesn't click on name of user,	System doesn't display profile of that user.	
<b>Date:</b>		
<b>Tester Name:</b>	Deeba Mehboob, Javeria Akbar	

## **CHAPTER 8**

### **SCREEN SHOTS**

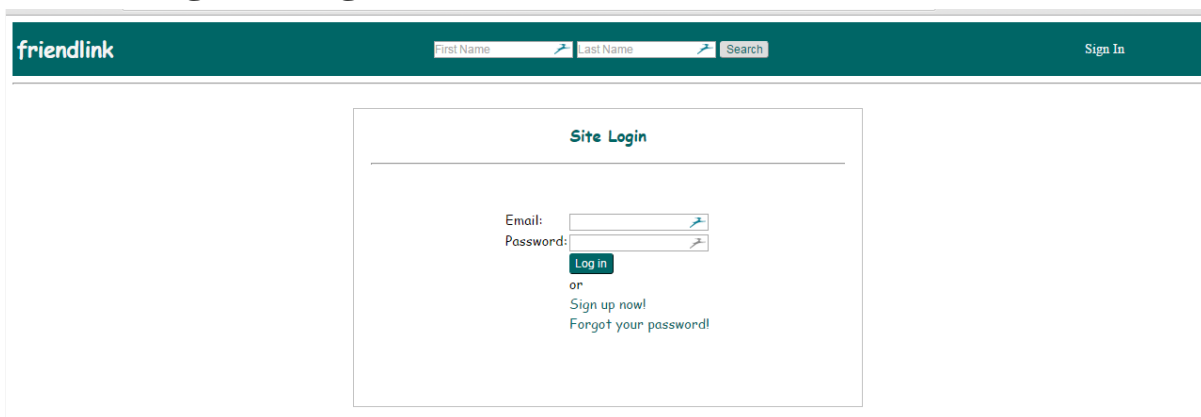


## 8.1. Welcome Page



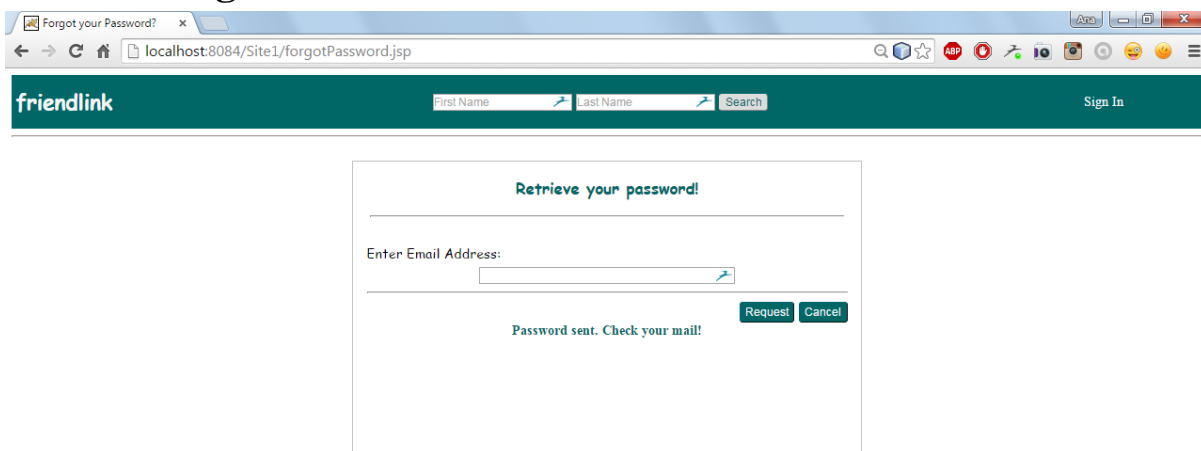
The screenshot shows a web browser window with the address bar displaying 'localhost:8084/Site1/index.jsp'. The page has a dark green header with the 'friendlink' logo on the left and a login section on the right. The login section includes fields for 'Email address' and 'Password', a link for 'Forgot your password?', and a 'Sign In' button. Below the header, the main content area is titled 'Connect with friendlink...'. On the left, there is an illustration of two stylized human figures, one blue and one green, with speech bubbles. On the right, there is a 'Register!' section with input fields for 'First name', 'Last name', 'Email address', and 'Password(6 or more characters)', followed by a 'Join now' button. At the bottom, there is a 'Find People:' section with input fields for 'First name' and 'Last name', and a 'Search' button.

## 8.2. Sign In Page



The screenshot shows a web browser window with the address bar displaying 'localhost:8084/Site1/index.jsp'. The page has a dark green header with the 'friendlink' logo on the left, a search bar with 'First Name' and 'Last Name' fields and a 'Search' button in the center, and a 'Sign In' button on the right. The main content area is titled 'Site Login'. It contains a form with 'Email:' and 'Password:' labels, each followed by an input field. Below the password field is a 'Log in' button. Underneath the 'Log in' button, there is a link that says 'or Sign up now!'. At the bottom of the form, there is a link that says 'Forgot your password!'.

## 8.3. Forgot Your Password?



The screenshot shows a web browser window with the address bar displaying 'localhost:8084/Site1/forgotPassword.jsp'. The page has a dark green header with the 'friendlink' logo on the left, a search bar with 'First Name' and 'Last Name' fields and a 'Search' button in the center, and a 'Sign In' button on the right. The main content area is titled 'Retrieve your password!'. It contains a form with the label 'Enter Email Address:' followed by an input field. Below the input field, there is a 'Request' button and a 'Cancel' button. At the bottom of the form, there is a message that says 'Password sent. Check your mail!'.

## 8.4. Add Cover and profile photo

The screenshot shows a web form titled "Add Profile and Cover Photo" within the "friendlink" application. The form is divided into four sections for uploading different types of photos:

- Upload Private Profile photo:** Includes a "Choose File" button, a "No file chosen" status, and an "Upload" button. A small preview image of a person's face is shown.
- Upload Private Cover photo:** Includes a "Choose File" button, a "No file chosen" status, and an "Upload" button. A small preview image of a colorful abstract design is shown.
- Upload Public Profile photo:** Includes a "Choose File" button, a "No file chosen" status, and an "Upload" button. A small preview image of a person's face with text overlay is shown.
- Upload Public Cover photo:** Includes a "Choose File" button, a "No file chosen" status, and an "Upload" button. A small preview image of a sunset scene is shown.

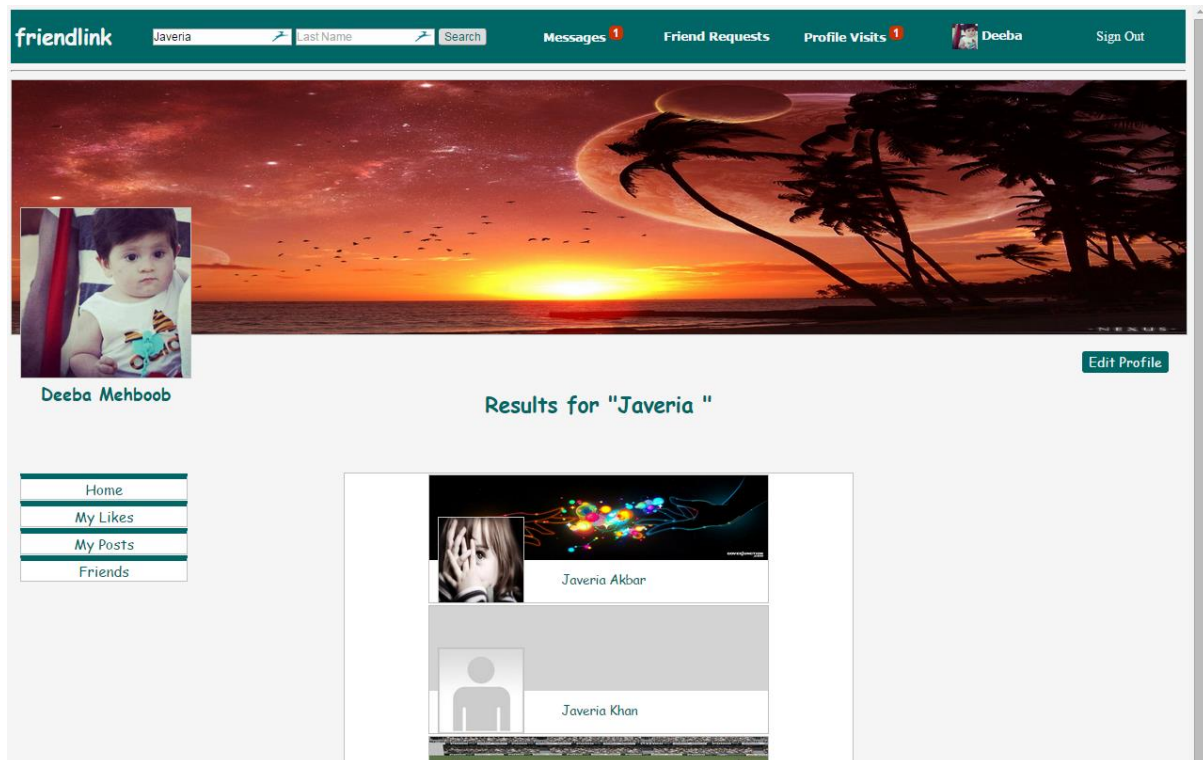
At the bottom of the form is a "Next" button.

## 8.5. My Posts Page:

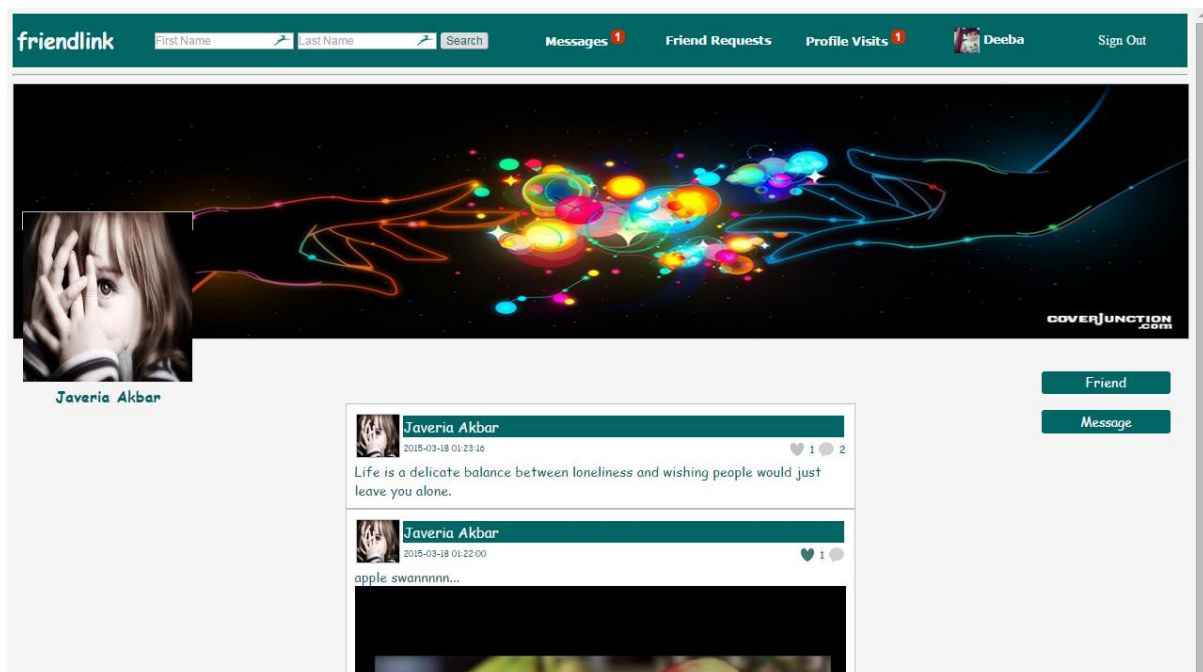
The screenshot shows the "My Posts Page" of the "friendlink" application. The page layout includes:

- Header:** The "friendlink" logo, search bars for "First Name" and "Last Name", and navigation links for "Messages", "Friend Requests", "Profile Visits", "Javeria" (user profile), and "Sign Out".
- Profile Section:** A large cover photo (a colorful abstract design) and a profile picture (a person's face). Below the profile picture is the name "Javeria Akbar" and an "Edit Profile" button.
- Post Creation:** A section with an "Upload Picture" button, a "Choose File" button, a "No file chosen" status, and a text input field labeled "Write...". Below this is a "Post" button.
- Post List:** A list of posts by "Javeria Akbar". The first post, dated "2015-03-18 01:23:16", has the text "Life is a delicate balance between loneliness and wishing people would just leave you alone." and shows 1 like and 1 comment. The second post, dated "2015-03-18 01:22:00", has the text "apple swannnnn..." and shows a video player.
- Left Sidebar:** A menu with links for "Home", "My Likes", "My Posts", and "Friends".

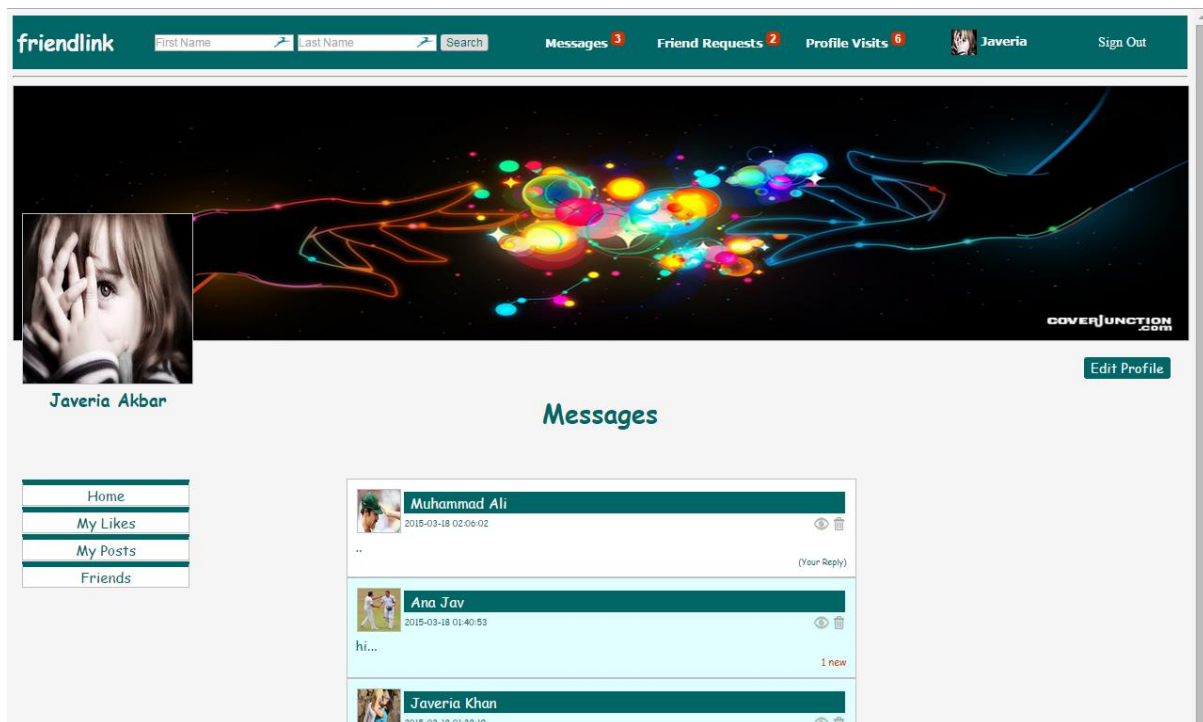
## 8.6. Search



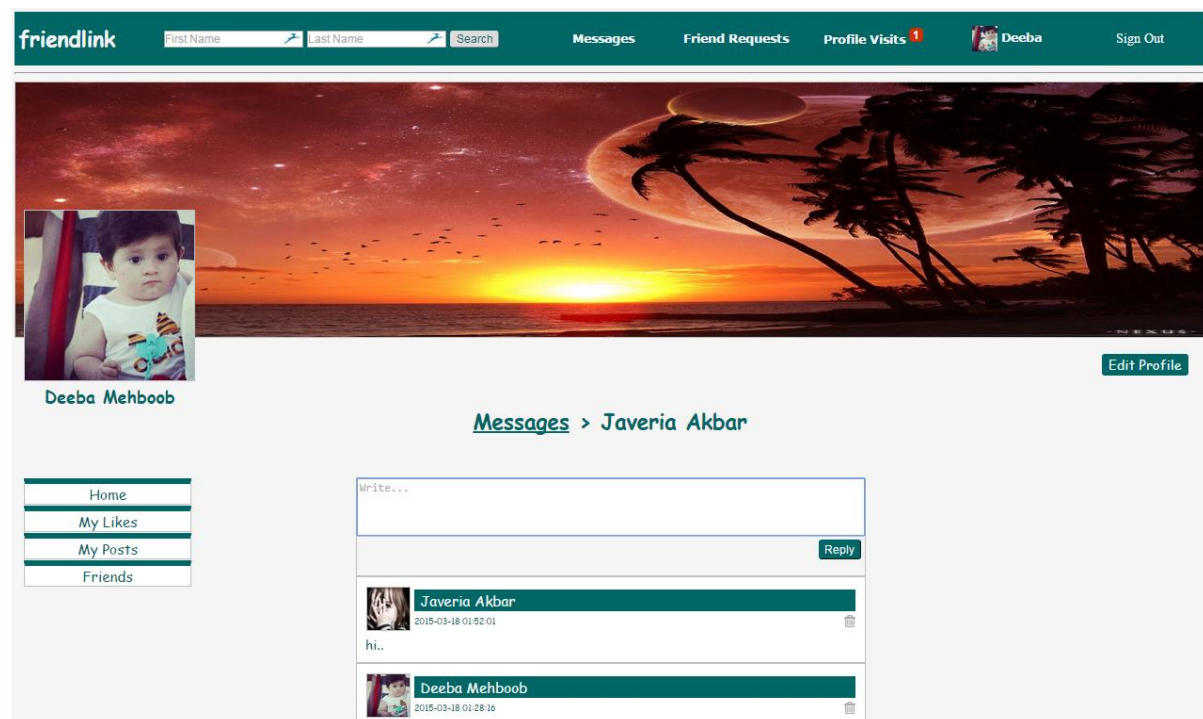
## 8.7. View Profile



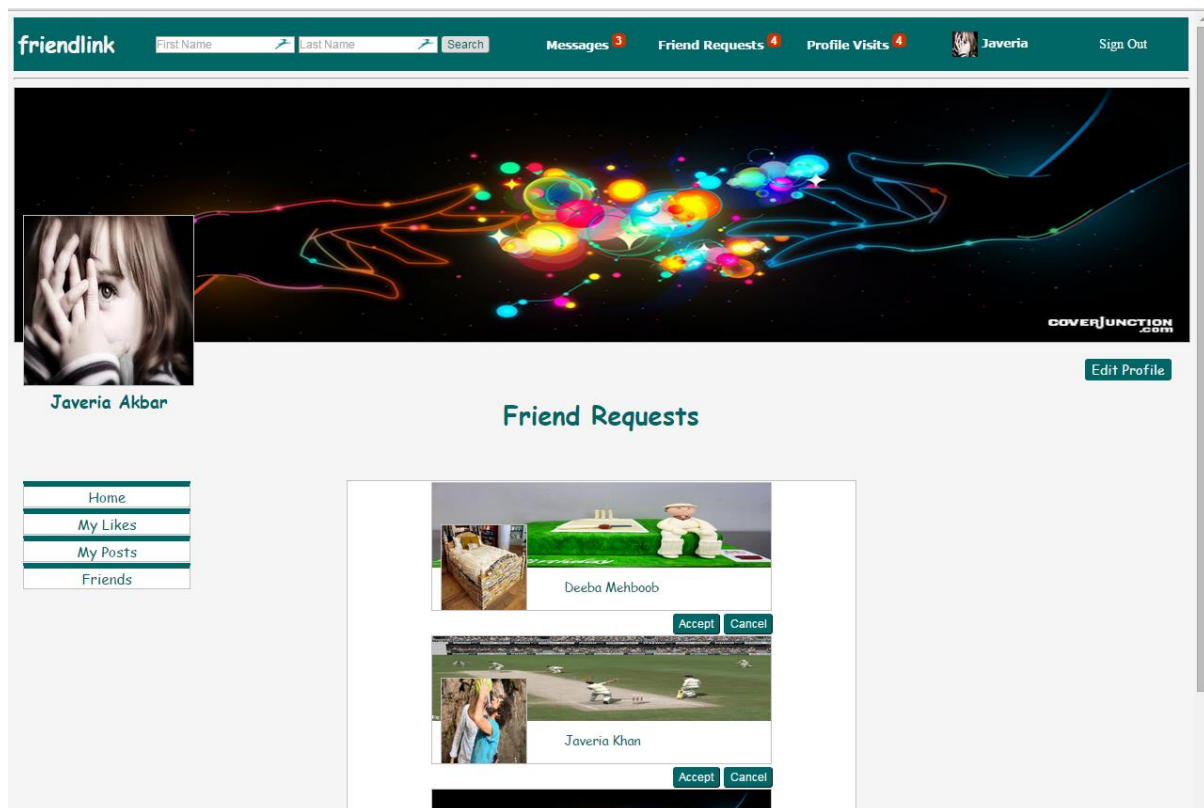
## 8.8. Messages



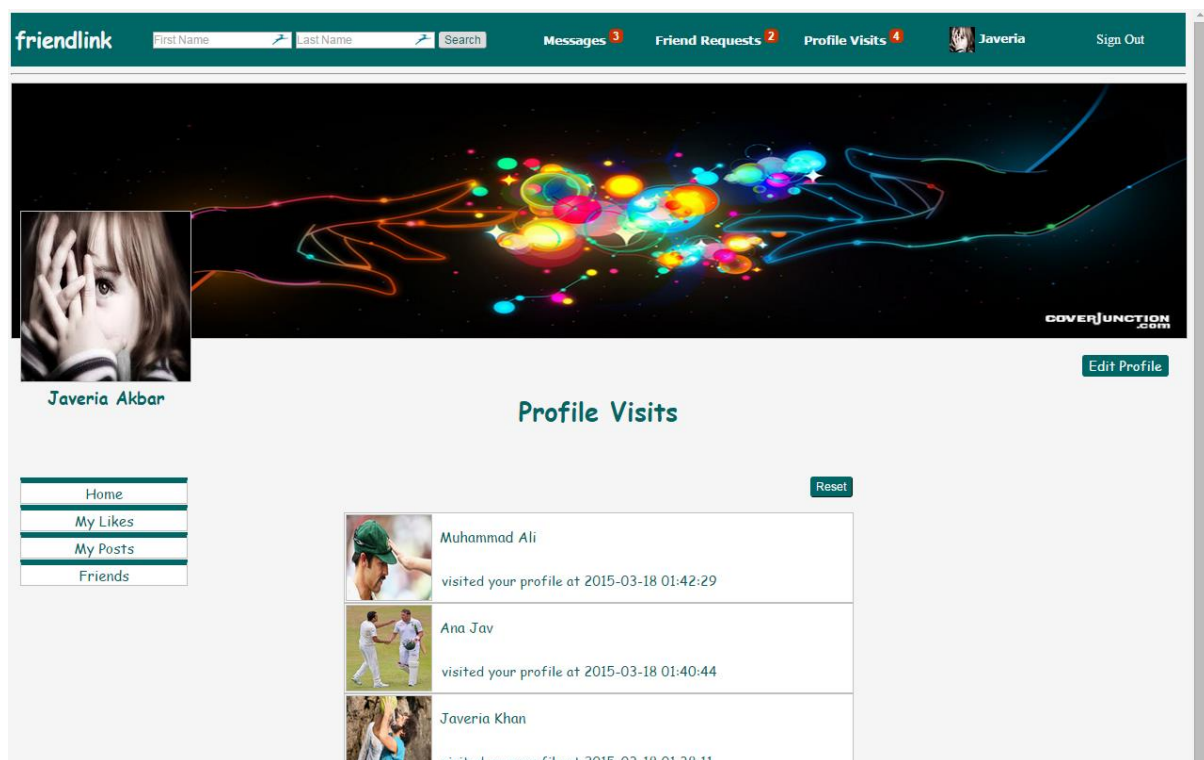
## 8.9. Replying to messages:



## 8.10. Friend Requests:

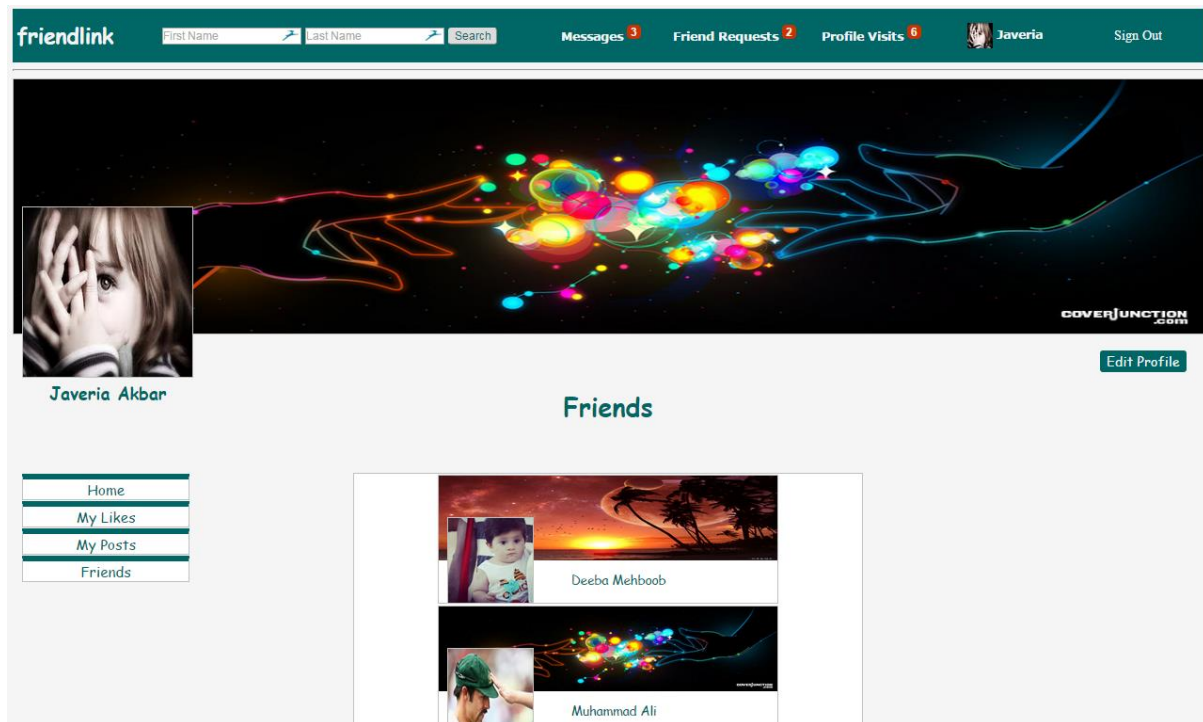


## 8.11. Profile Visits:

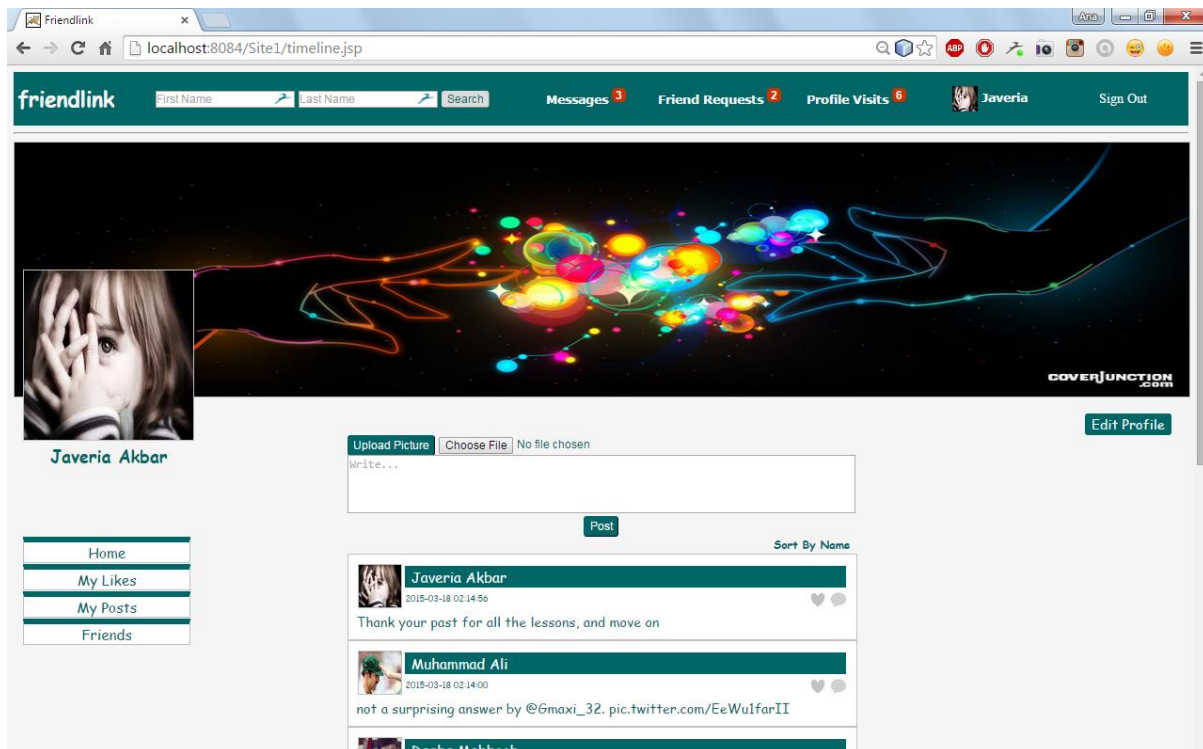




## 8.12. Friends:



## 8.13. Home



## 8.14. Edit Profile:

friendlink First Name Last Name Search Messages 3 Friend Requests 2 Profile Visits 6 Javeria Sign Out

Choose File No file chosen  
Change

Choose File No file chosen  
Change

**Change your Name**

First Name: Javeria  
Last Name: Akbar  
Save Changes

**Change your Password**

Enter current Password:  
Enter new Password:  
Enter new Password Again:  
Save Password

**Edit public profile photo:**  
Edit Choose File No file chosen

**Edit public cover photo:**  
Edit Choose File No file chosen

COVERJUNCTION.COM

## 8.15. Commenting on a post:

Javeria Akbar localhost:8084/Site1/viewPost.jsp friendlink First Name Last Name Search Messages Friend Requests Profile Visits 1 Deeba Sign Out

**Deeba Mehboob** Edit Profile

Home  
My Likes  
My Posts  
Friends

**Javeria Akbar**  
2015-03-18 02:14:50  
Thank your past for all the lessons, and move on  
Write...  
Comment

# **CHAPTER 9**

# **CONCLUSION**



## **9. Conclusion:**

### **9.1. Benefits**

The advantages of using a social networking site are as follows

- User can select two different DPs and CP's at the same time.one for the friends and the other for the other users simultaneously
- Person van view his timeline visits in a day
- Timeline can be sorted by name also
- Email is sent to user in case of forgetting password
- A large database of knowledge can be added to and kept up-to-date. It can store more knowledge than a person.
- The system cannot 'forget' or get facts wrong.
- It survives forever. There is no loss of knowledge

### **9.2. Features**

Features of SNS include:

- Timeline (User Interface )
- Account Management
- Data Entry (Enter required information)
- Profile Visits
- Public cover photo and Private cover photo
- View Messages, friend requests, delete messages, like posts, delete posts, edit posts, delete friend, upload pictures, upload messages, unlike posts
- By name sorting
- Email sending

## REFERENCES

- <http://www.google.com>
- <http://www.vutube.edu.pk>
- <http://www.w3schools.com>
- <http://www.theserverside.com>
- <http://www.sw-engineering-candies.com>
- <http://stackoverflow.com>
- <http://www.thesitewizard.com>
- <http://www.javatpoint.com>
- <http://www.codeproject.com>