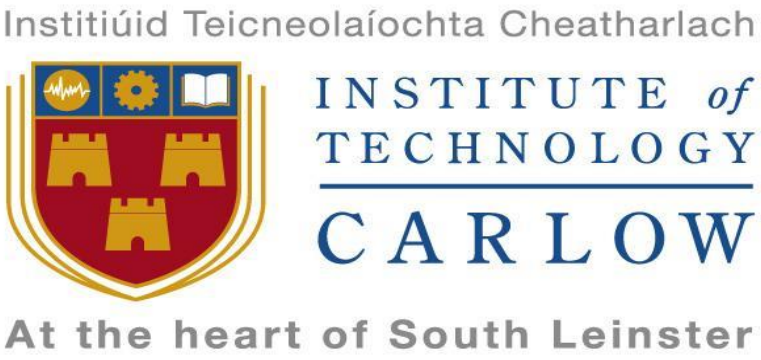
5th April 2017

Design document



Department of Computing and Networking

Software Development Degree

Project name: MaaP (Message as a Platform)

Student: Chihabeddine Ahmed

Student Number: C00210496

Supervisor: Joseph Kehoe

Table of Contents

1. Introduction

1.1 What is MaaP?

MaaP (Messaging as a platform) is a self-hosted messaging application built for android.
The front-end of the application build using Java and the backend is built purely on Node.js. The application will allow the user to authenticate, i.e. Login, Register, and Logout of the system. It will allow the user to send and receive messages to a particular user. And also it will allow a user to search for another user then send him an invitation and upon the invitation acceptance, they will be able to communicate with each other instantly. The goal of this project to build the application for many types of users such Collages, companies etc. where these users can host the application on their own servers and own the data. The functionality that separates this application from any other application is that the application will be developed in a micro-service architecture and it will open sources. The intention is to develop this application in an android platform for a good user experience and also allow me to explore android development from the front-side point view. This application will be developed across three iterations from October 2016 until April 2017 by a single developer.

1.2. Purpose of this document

The purpose of this document is to give an overview of the design of this application on a technical level, broken down on a by iteration basis. This document will cover System overview, Use Case overview, System sequence diagram, Class diagrams, Activity diagram, Database design and the User interface.

Some of the functionalities of this application such use case will be described in greater technical detail in the Functional Specification of this project, the results of all research done will be discussed in the Research Document and the overall results of this project will be discussed in detail in the Final Project Report.

.

2. Overview

2.1 System

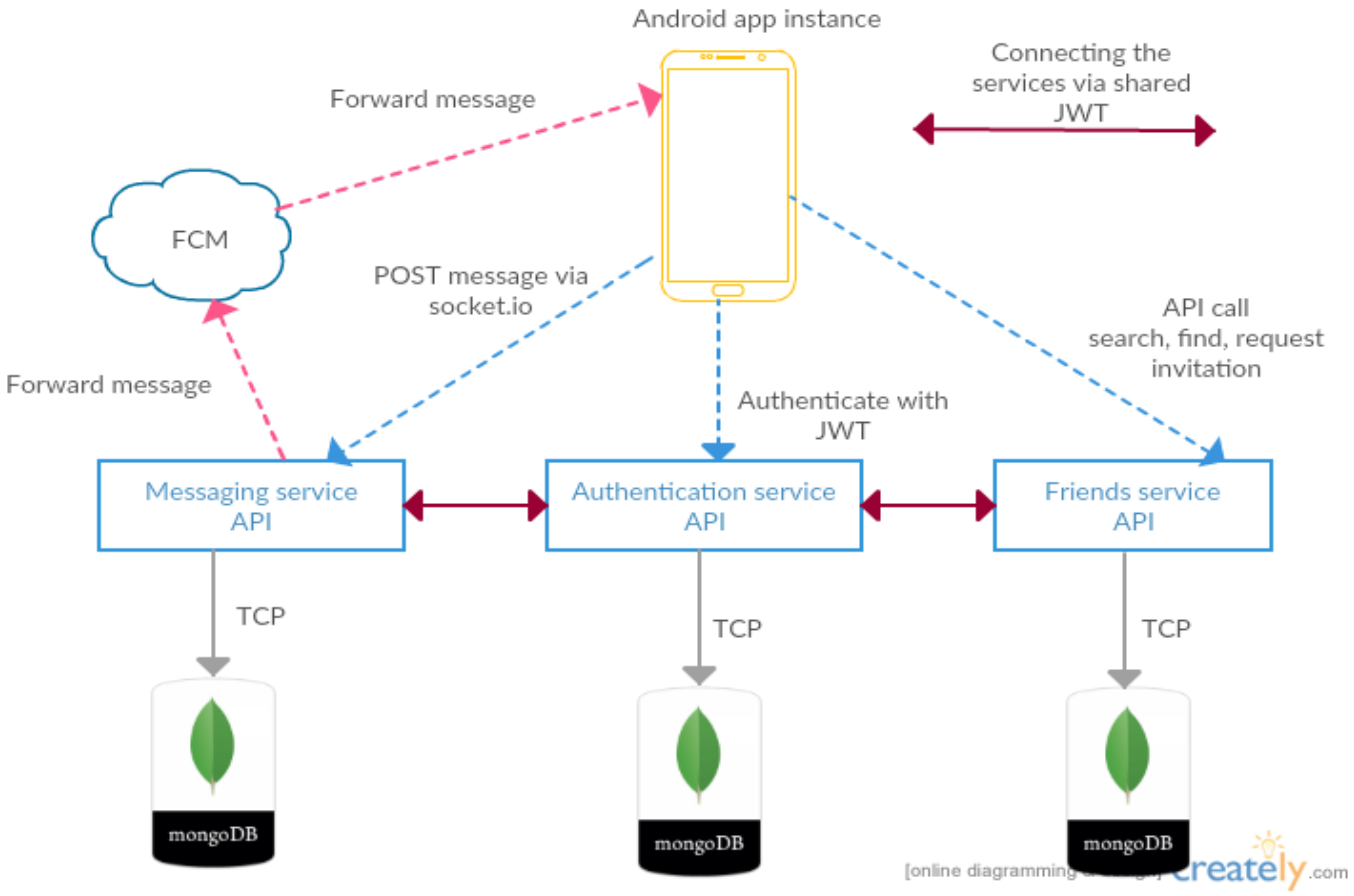


Figure: 1 (System overview)

2.2 Notifications Overview

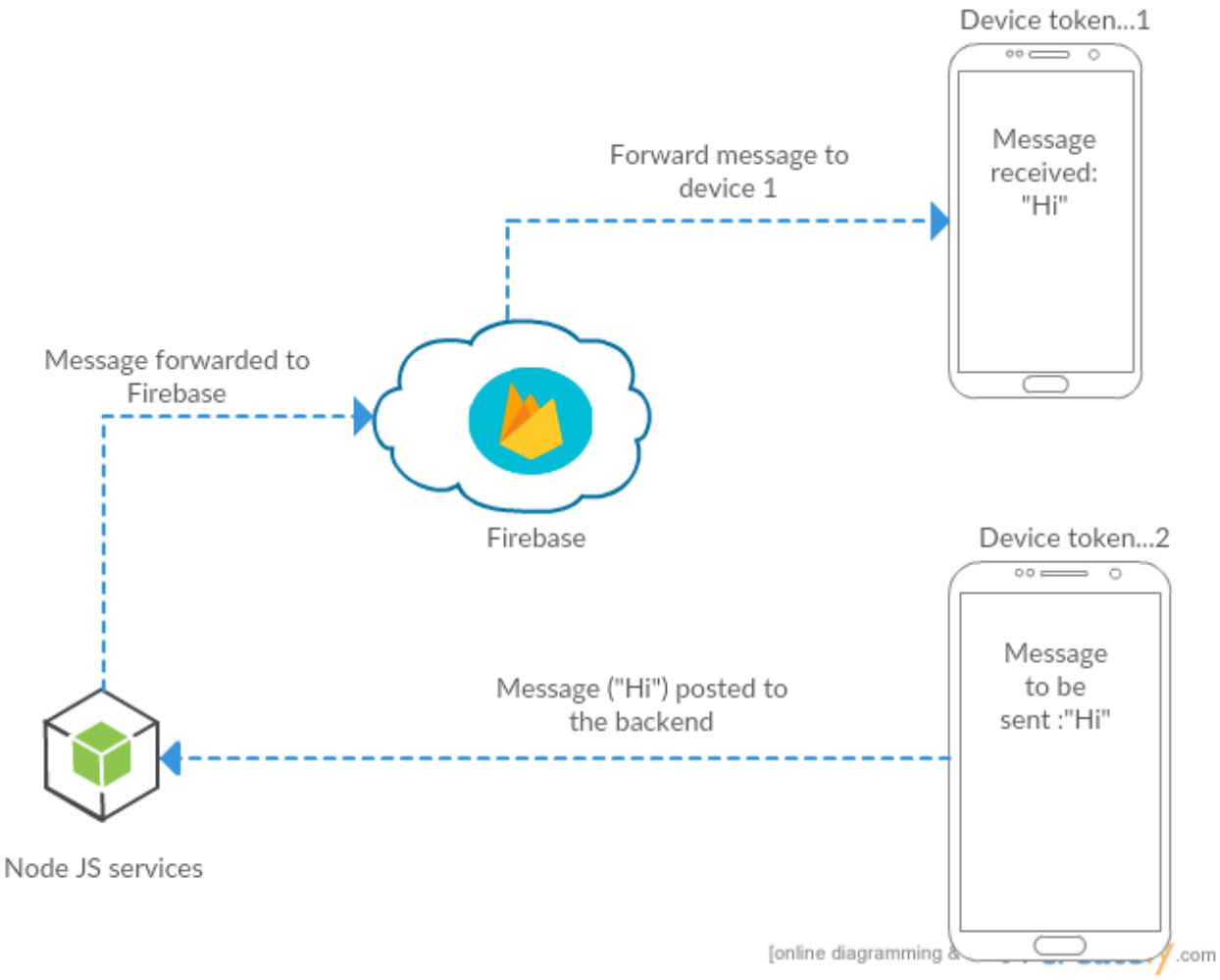


Figure 2: (Notifications overview)

3. Use Case Overview

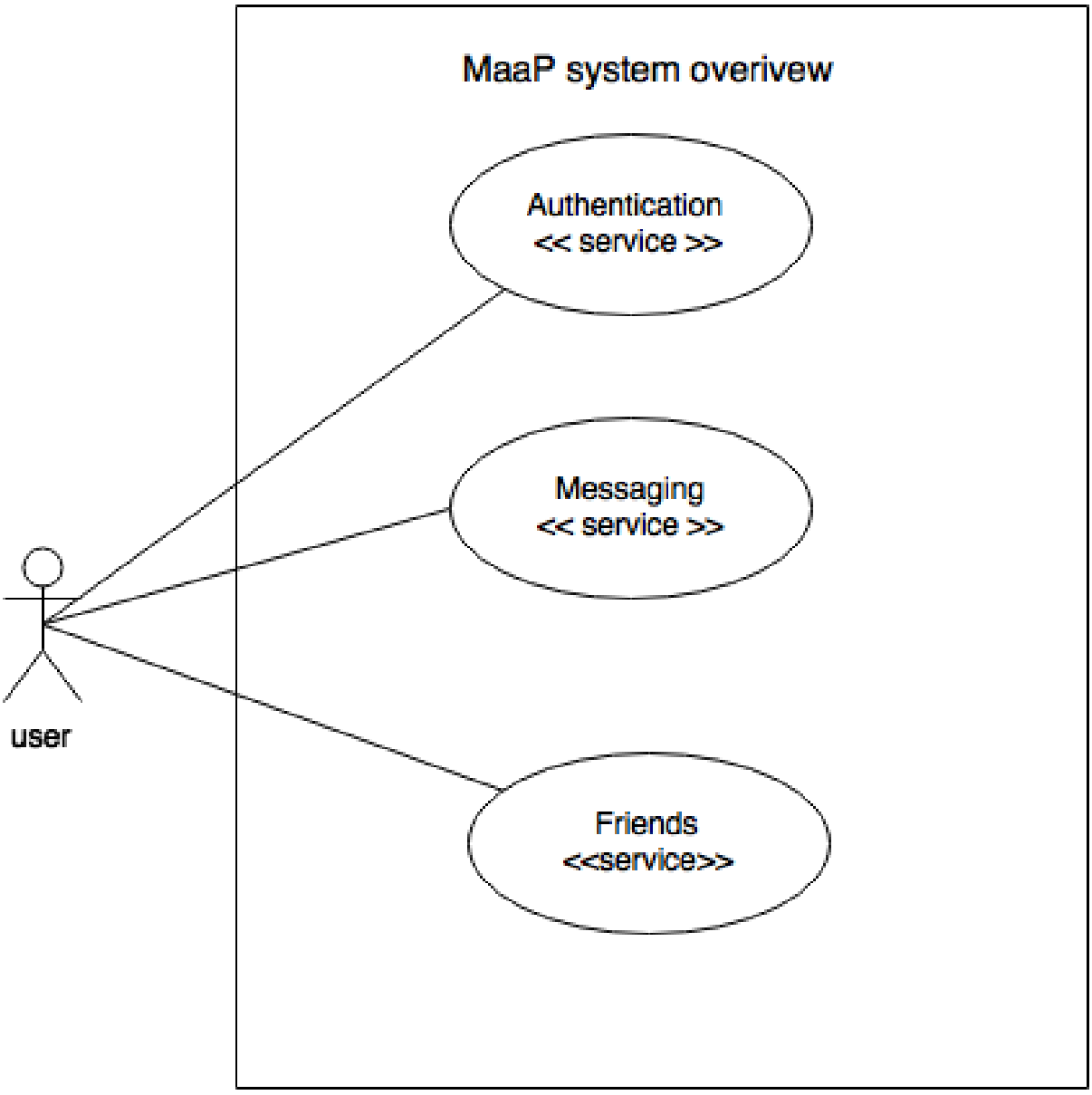


Figure: 3 (Use case overview)

4. System Sequence Diagrams

4.1 Authentication diagram detailed

4.1.1 Register

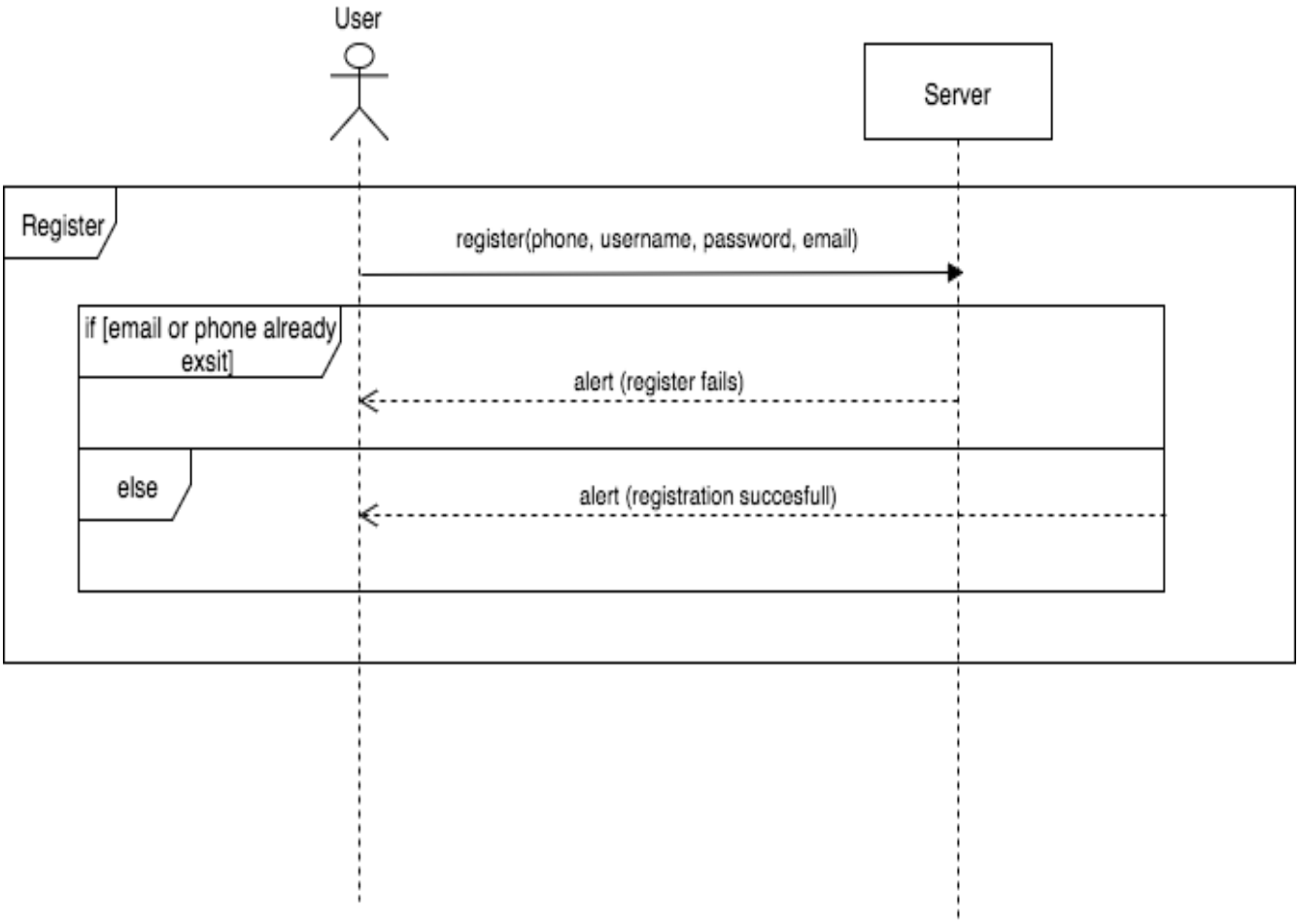


Figure: 4 (Register)

4.1.2 Login

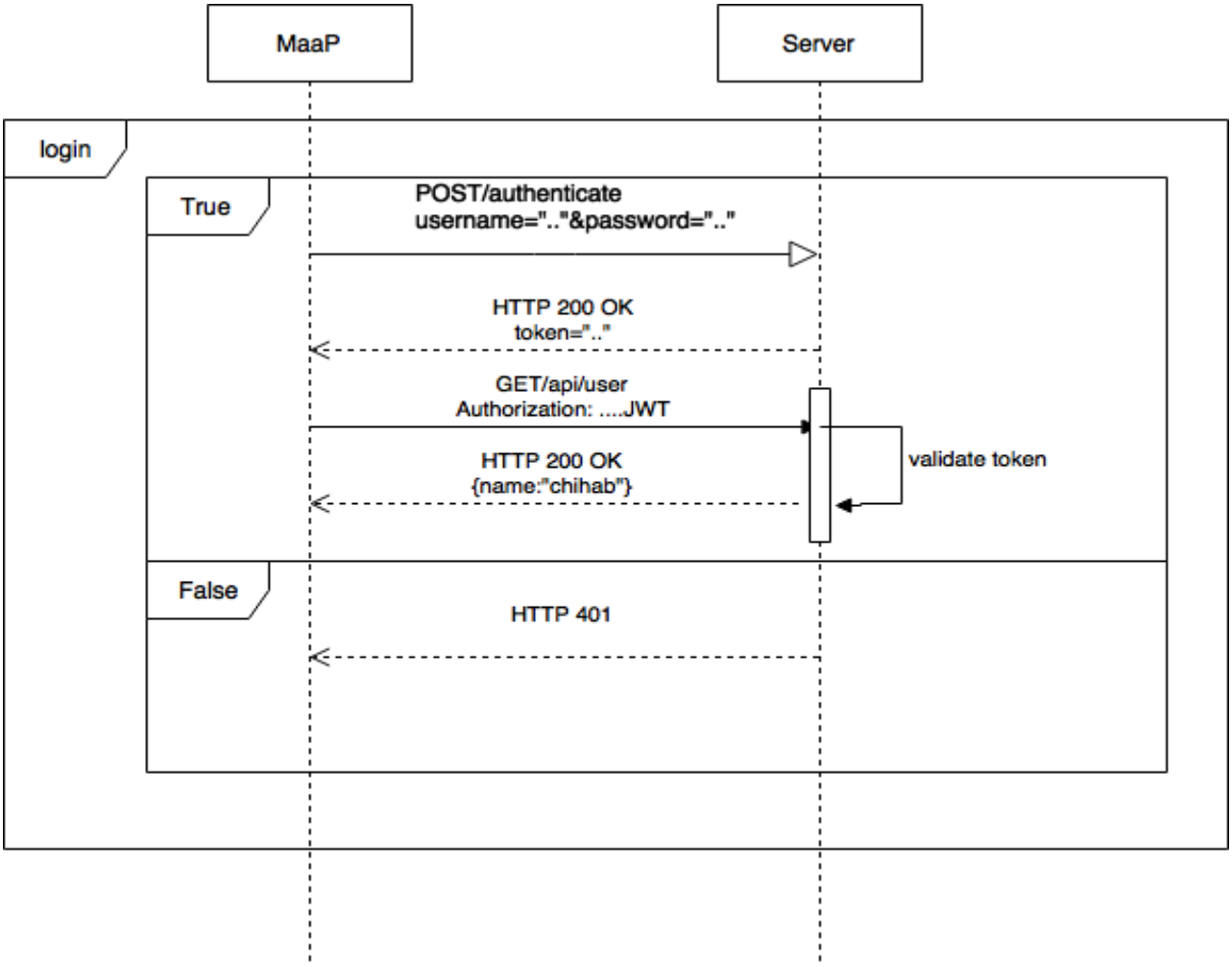


Figure: 5 (login)

4.1.3 Logout

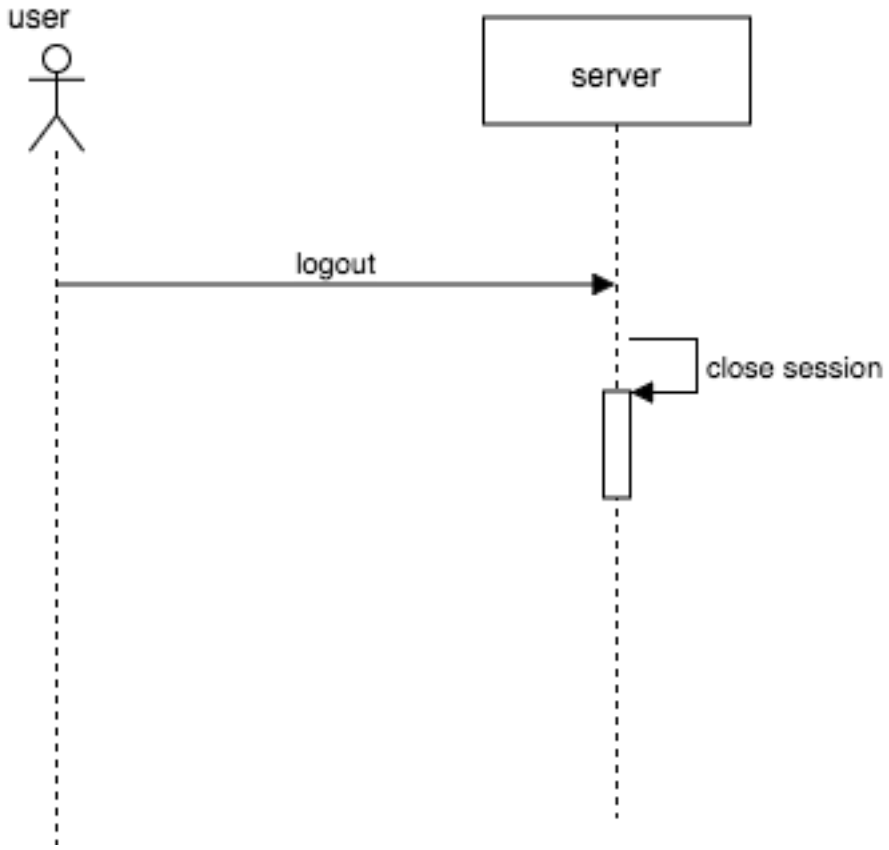


Figure: 6 (logout)

4.2 Messaging diagrams

4.2.1 Send / receive /display message

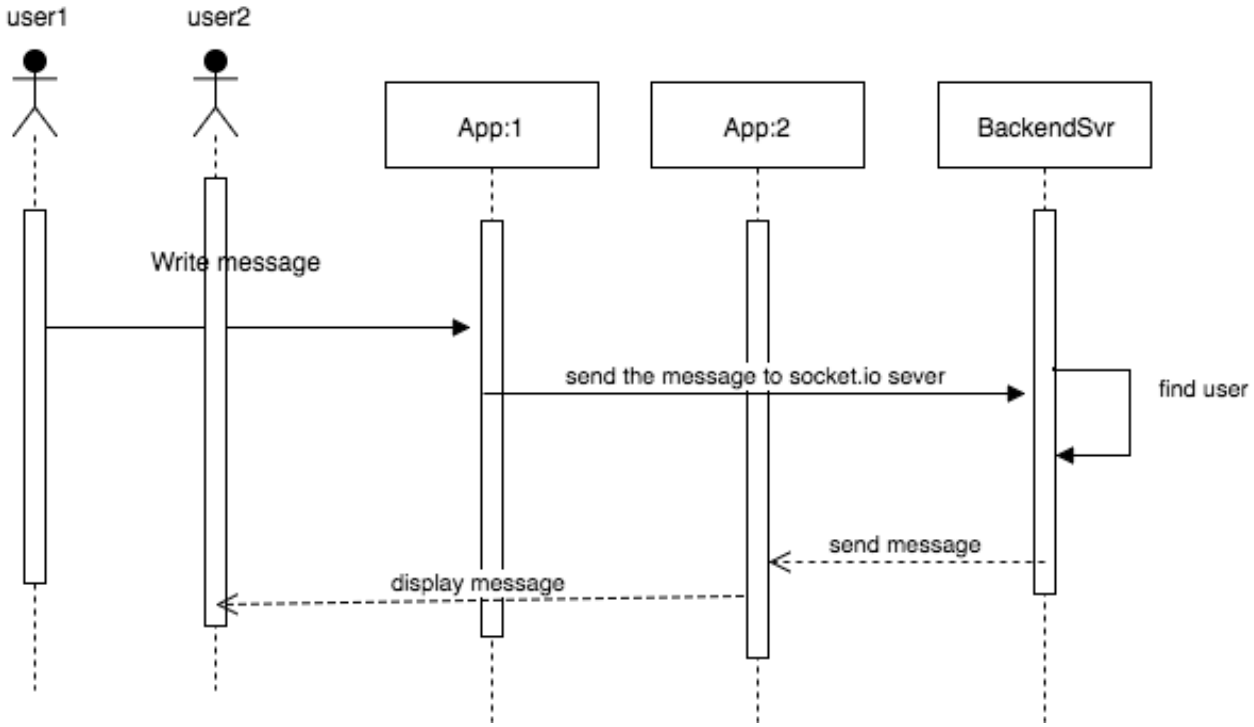


Figure: 6 (view, send)

4.3 Friending diagrams

4.3.1 Search friend

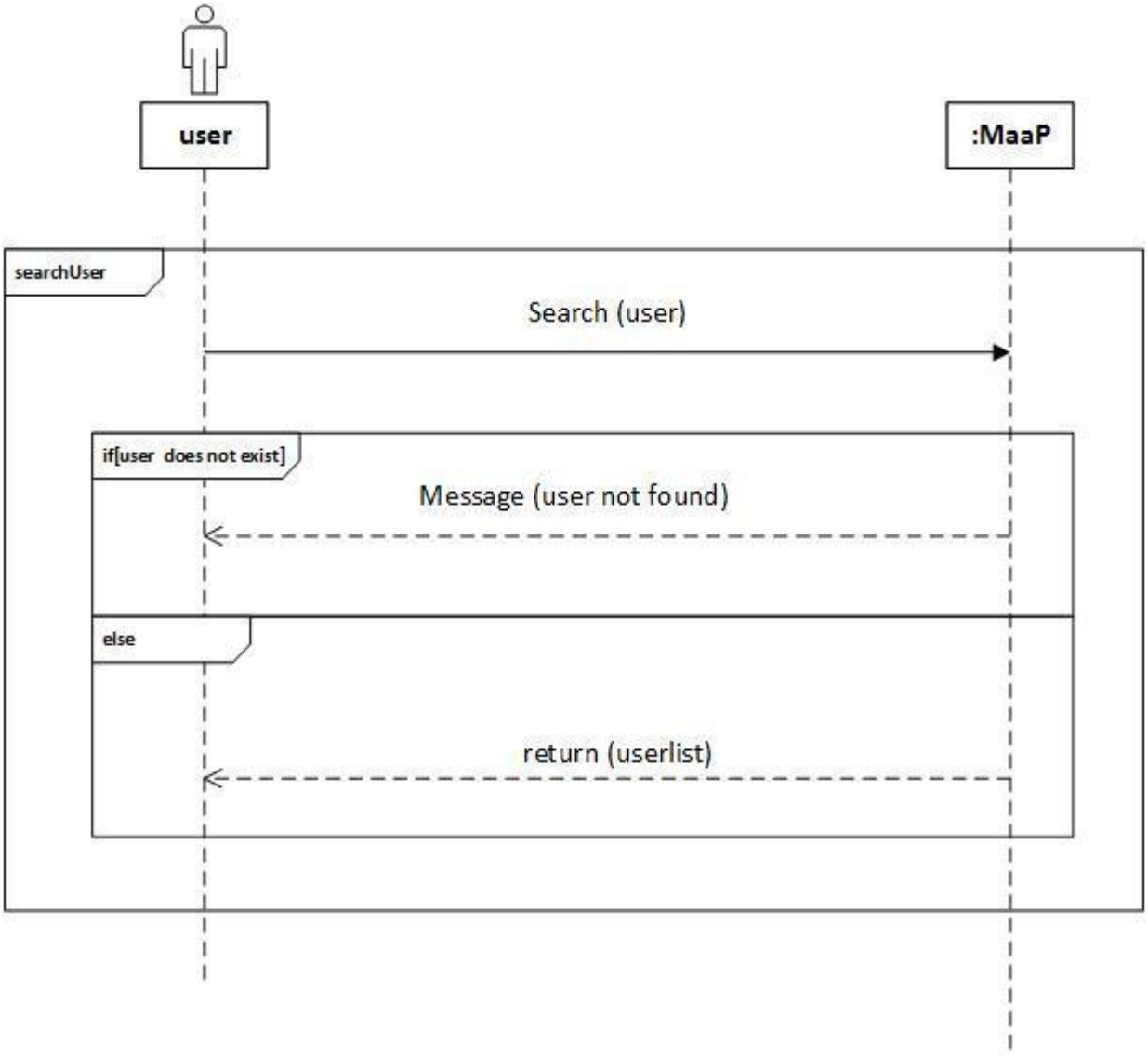


Figure: 7 (search user)

4.3.2 Add Friend and accepting request

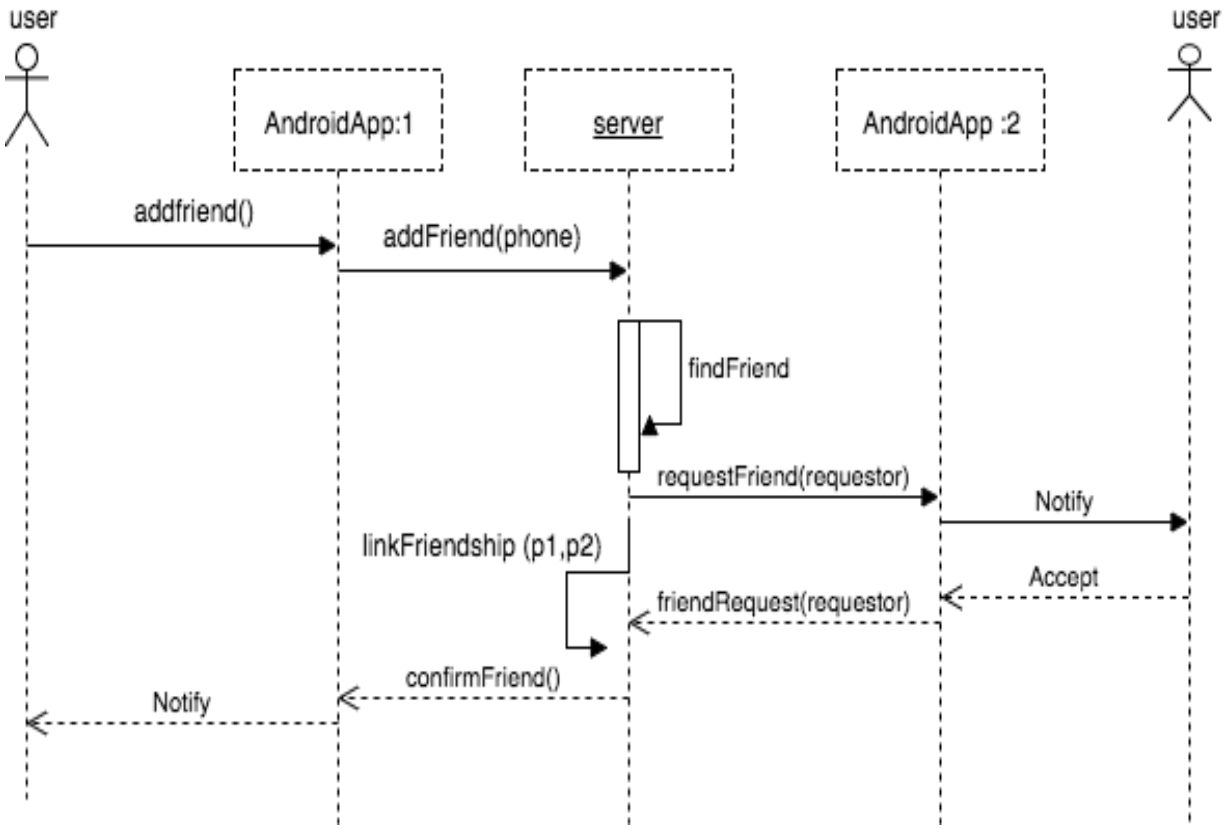


Figure: 8 (add, accepting friend)

5. Activities diagrams

5.1 Register Activity

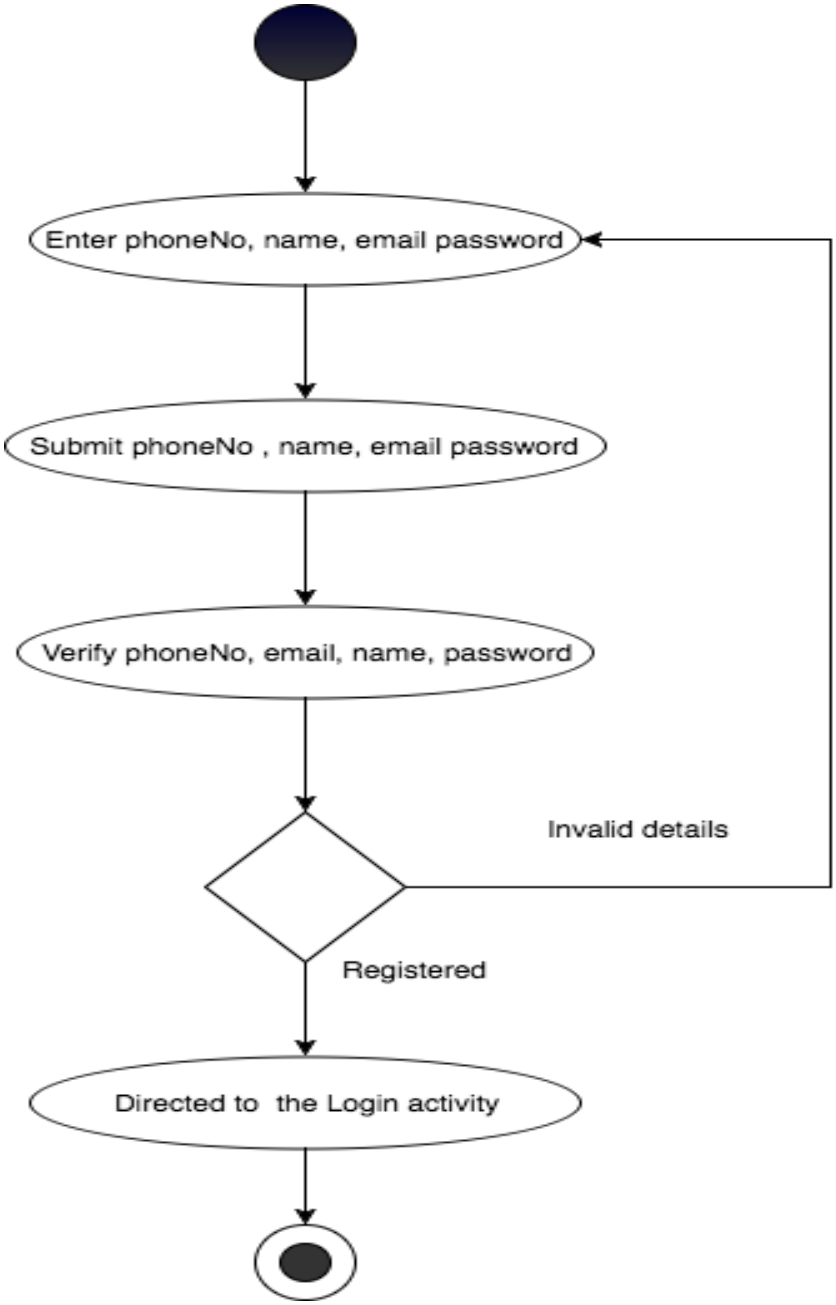


Figure: 9

5.2 Login Activity

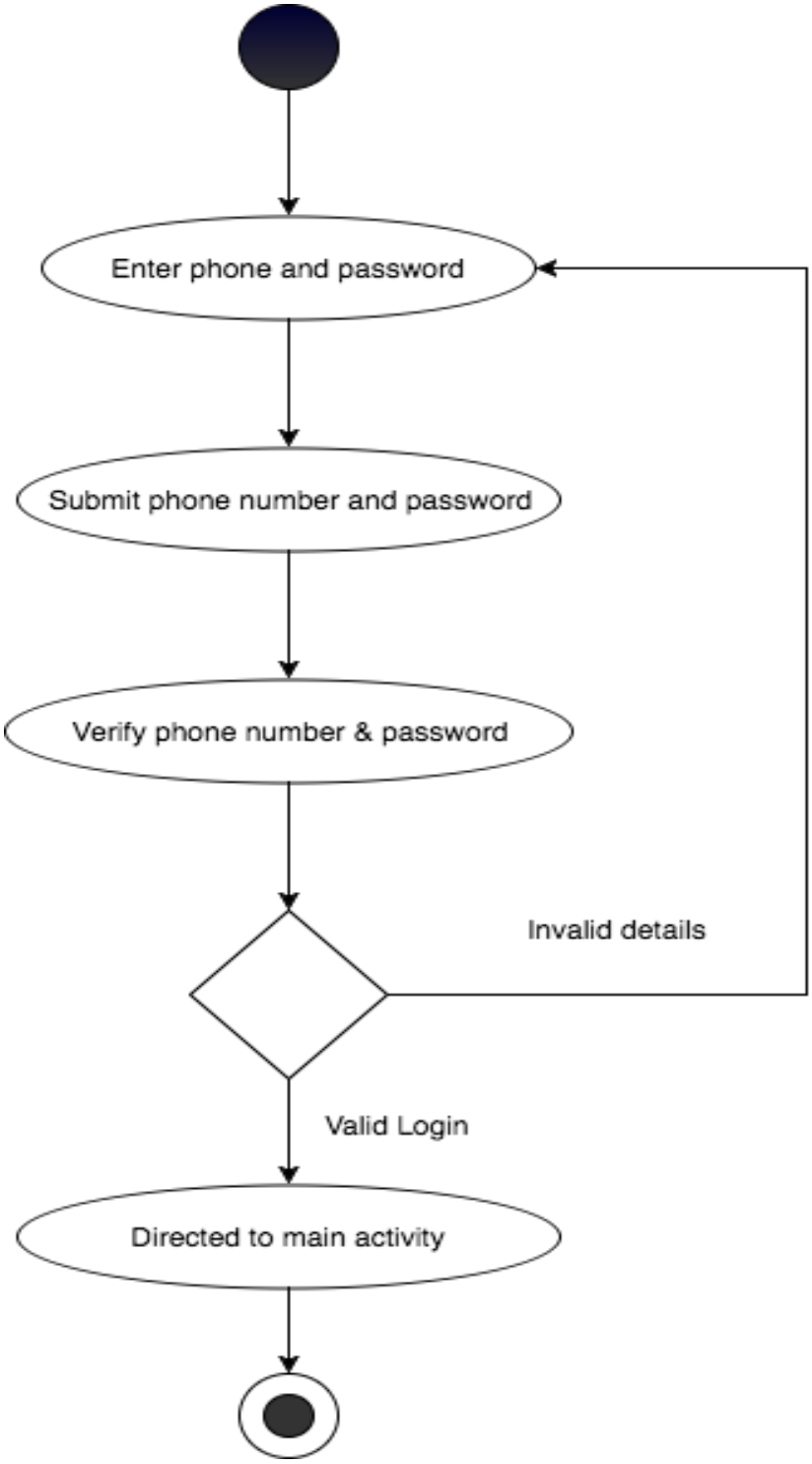


Figure: 10

5.3 Friending Activity

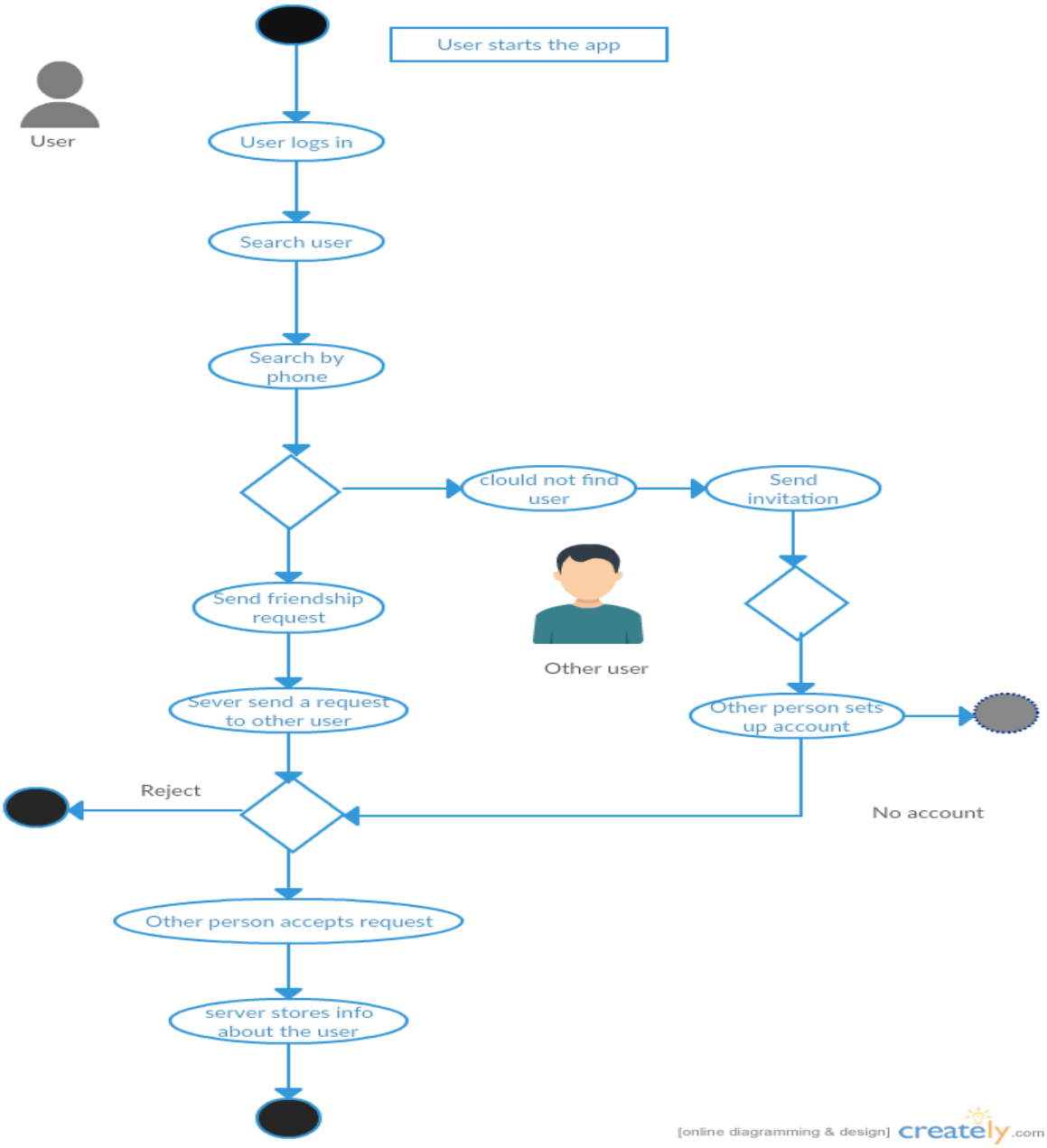


Figure: 11

5.3 Sending message Activity

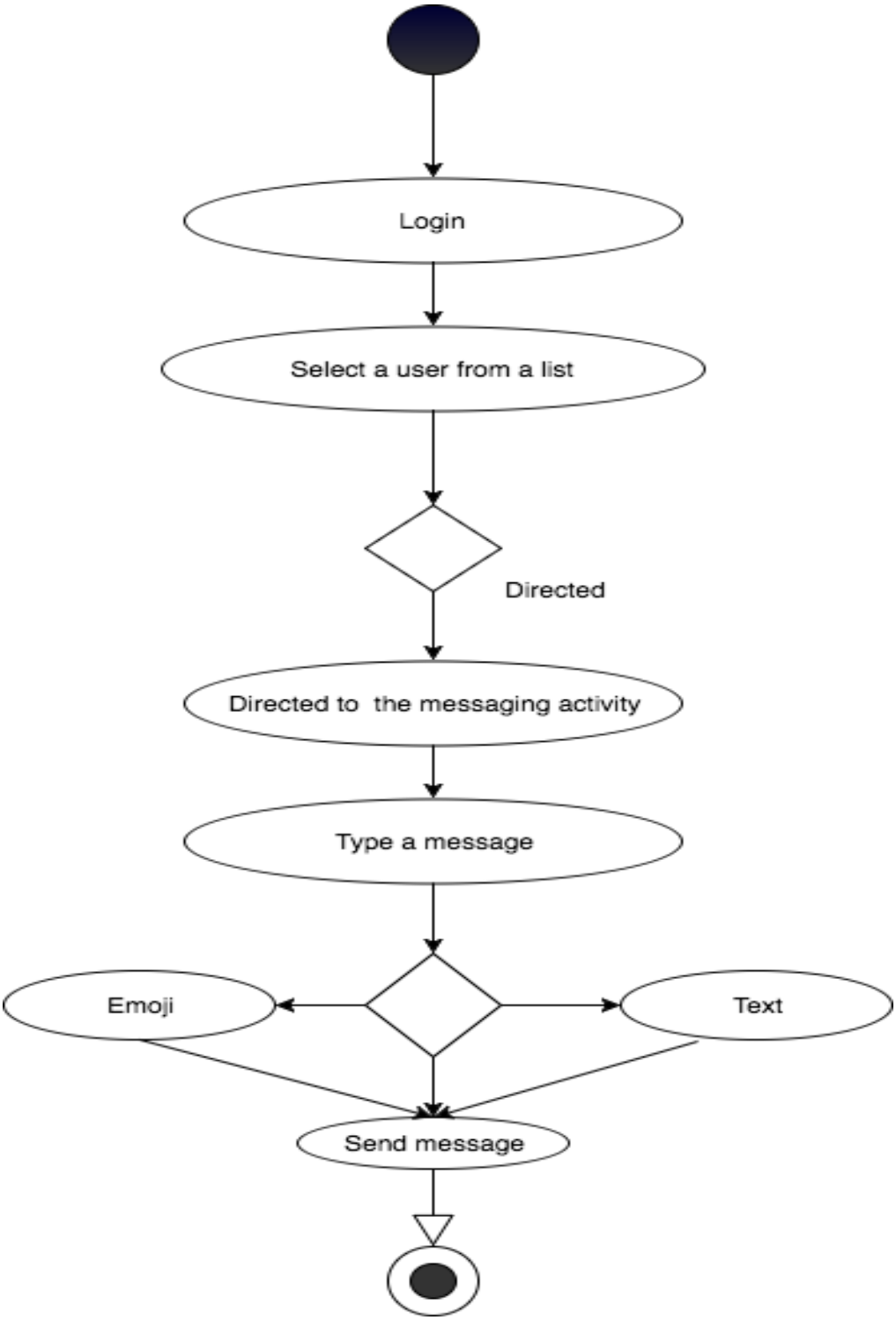


Figure: 12

6. Class diagram

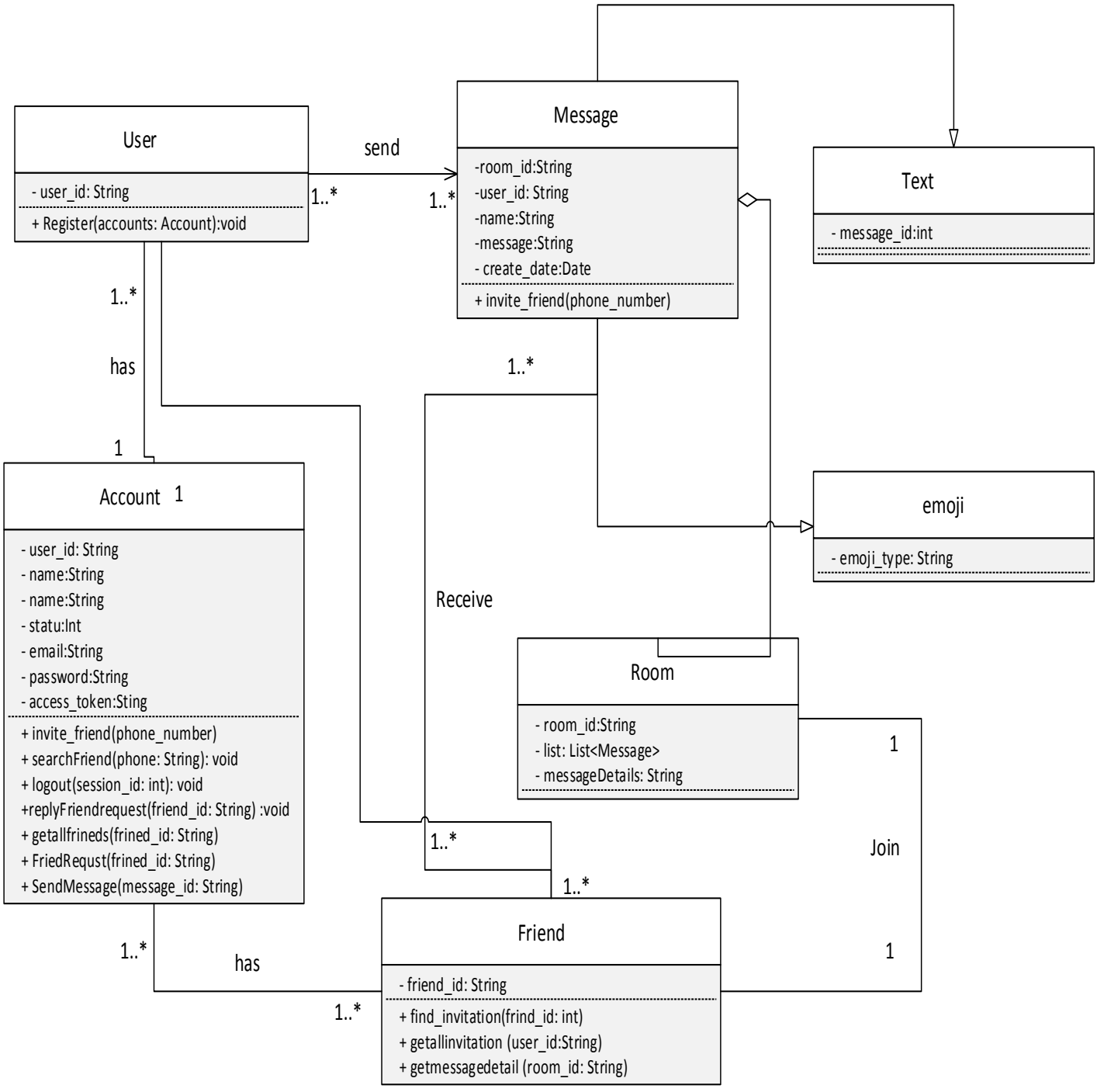


Figure: 13

7. Database Design

This system requires storage mechanism to store user data. Through the research, I have chosen Mongo database as a storage mechanism due to the given reasons. The date that will be stored is the Authentication, messaging and friend data. in this section, I will plan my database for our system and changes to the planned database will be updated at a later stage of the development process.

7.1 Authentication schema

Authentication Document.

```
{
        “_id”: ObjectId(“ 3354lk5334”),
        “name”: “chiahb”,
        “user_id”: “0883844888”,
        “password”:”12345”
        “email”: “adfa@gmail.com”,
        “createdAt”:”2017-01-1:43:03”,
        “access_token”: “P432OLD53llss”
        “notificatoin_list” : “ [element]”
}
```

7.2 Messaging schema

Messages Document

```
{
        “_id”: ObjectId(“ 3354lk5334”),
        “room_id”: “0883844888-12345678”,
        list [
                “user_id” : “0883844888”,
                “name” : “adam”,
                “message” : “How are you”,
                “data” : “ date created”,
                “ _id: “ Objectid(“34csdcse43tssf43ss”)
        ]
}
```

Recent messages document

```
{
        “_id” : Objectid(“dsfljasfle3ws”),
        “user_id : “1234567”,
        list [
                “room_id” : “0894015048-1234567”,
                “friend_name”: “chihab”,
                “sneder_id” :  “1234567”
                “sender_id” : “chihab”,
                “message” :  “How are you?” ,
                “date” : (message date that was sent),
                “_id” : Objectid(“dsfljasfle3ws”)
        ]

}
```

7.3 Friends schema

Friends Document

```
{
        “_id” : Objectid(“dsfljasfle3ws”),
        “user_id : “1234567”,
        list [
                “friend_id” : “0894015048”,
                “name”: “adam”,
                “status“: “1”
        ]

}
```

Invitations Document

```
{
        “_id” : Objectid(“dsfljasfle3wsfsdfdsfasf”),
        “user_id : “1234567”,
        list [
                “friend_id” : “0894015048”,
                “name”: “adam”,
                “message” : “you have got invitation request”
                “_id” :  Objectid (“dsfdasfewwessss9433”),
                “status“: “1”
        ]

}
```

8. API Design.

8.1 Authentication API

1. Login:

Description:
Identify and check the user details before allowing him/her to use the app.

Resource: /api/login
Method: GET
Host: Localhost: 4000
Content-Type: Application/json

Example:

```
{
        “phone”:”12345678“
        “password”:”1234“
}
```

Return:

- **200 +** return details about that user and direct him/her to main activity.
- **405 +** error message (use not found) if user_id (phone) does not exist or **401** (Unauthorized)

2. Register:

Create a new user by providing his/her name, phone number, email, and password. You will receive an object that holds the name, phone number, email, and password.
Resource: /api/register
Method: POST
Example:

```
{
        “_id”: ObjectID (“35645656kdbs”),
        “name”: “chihab”
        “user_id”: “0883844888”
        “password”: “12345”,
        “email”: “adf”gmail.com”,
        “createAt”: “2017-01–1:43:04”,
        “access_token”: “fdsaf454554fasdfa”,
        “notification_list”: “[elment]”
}
```

Return: **200** + Post the entered data to the database

3. Logout

Description:
Check if session is created then destroy that session

Host: localhost: 4000
Resource: /api/logout
Method: DELETE

Example:

```
{
        “sessionname”: “12345657”
}
```

Return: Session is destroyed.

8.2 Messaging API

1. Recent messages:

Description
Retrieve all of the recent messages upon the user authentication. all of the recent messages will be retrieved from the authenticated user.

Host: localhost: 4000
Resource: /api/get-merecent
Method: GET
Response:

- **400** + “system error”
- 2**00** + “success”

Result: Recent messages will be returned

1. Reply to a user

Description:
Reply to a user
Host: localhost: 4000
Resource: /api/friend-reply
Method: POST
Return: Post message to a user

Reply to a user's

2. Get message detials

Description
Get message details

Host: localhost: 4000
Resource: /api/get-mes-detail/messages
Method: GET

```
GET /api/get-mes-detail/room_id/body/
```

Response:

- 400 + error
- 200 + success

Return: Retrieves a message details.

8.3 Friends API

Description
Search for a friend, send him/her a friendship request, they can accept that request or reject a request. and also get all invitation sent by different users.

2. Request Friend

Resource: /api/request/Friend/
Method: GET
Response:

- **200** + "send invitation was successful"
- **400** + "invitation request was not sent"

2. Add Friend

Resource: /api/add-friend
Method: POST

```
location: /add-friend/635
```

Response:

- `200 + " added"`
- All friends that were added.

Resource: /api/search-friend/
Method: GET

```
GET /api/search-friend/phonenumber=1234
Response:
        200 + "user found"
        401 + "no user found"
Result:
        id: 43,
         name: "john",
        friends: [3]
```

Response:

- **400** + "User not found"
- **200** + "list of users associated with the search"

Result: a user name associated with the searched phone number.

Resource: /api/getall-invitations
Method: GET
Example
`GET /api/getall-invitation/name=chihab……..etc`
Response:

- 200 + "susses"
- 400 + " false"

Return: List of all invitations