

DI3

FYP Final Report

Geo-Social Network for HKUST

by

Hou Wai Man, Lo Wai Tong and Tsui Kam Ming

DI3

Advised by

Prof. Dimitris PAPADIAS

Submitted in partial fulfillment

of the requirements for COMP 4982

in the

Department of Computer Science

The Hong Kong University of Science and Technology

2012-2013

Date of submission: April 23, 2012

Table of Contents

Abstract.....	4
1. Introduction	5
1.1 Overview	5
1.2 Objectives.....	5
1.3 Literature Survey.....	6
2. Design.....	7
2.1 System Architecture	7
2.2 Database Design (Conceptual)	7
2.3 Database Design (MongoDB)	8
2.4 API Design.....	13
2.5 Use Case	18
2.5.1 Use Case Diagram	18
2.5.2 Actor Description	19
2.5.3 Use Case Description	19
2.6 User Interface Design	32
3. Implementation	45
3.1 Implementation of DBConnection and CollectionBase Class	45
3.2 Implementation of Web Service	46
3.3 Implementation of Front-end	47
3.4 Using Haversine Formula to find nearby friends	49
4. Testing.....	49
4.1 Testing Methodology	49
4.2 Test Cases.....	49
5. Evaluation	55
6. Discussion.....	55
7. Conclusion.....	56
8. References	57
9. Appendix A: Minutes.....	58
9.1 Minutes of the 1st Project Meeting	58
9.2 Minutes of the 2nd Project Meeting.....	59
9.3 Minutes of the 3rd Project Meeting	60
9.4 Minutes of the 4th Project Meeting	61
9.5 Minutes of the 5th Project Meeting	62
9.6 Minutes of the 6th Project Meeting	63
9.7 Minutes of the 7th Project Meeting	64
9.8 Minutes of the 8th Project Meeting	65
10. Appendix B: Project Planning.....	66
10.1 Division of Work	66
10.2 GANTT Chart	67
11 Appendix C: Required Hardware and Software	68
11.1 Hardware	68
11.2 Software.....	68

Abstract

This paper describes a Geo-Social Network that allows users to interact with their peers. The system allows users to share their personal status, check-in and finding nearby friends in the campus.

The system will keep track of the users' check-ins and allows them to share their views through postings and replies. The users can make friends and inviting them to their events. To provide user with smooth and convenient browsing experience, the system implement the functionalities with Ajax and easy-to-use interface. With the GUI, control logic and data being separated into different layers, the system can distribute and focus the workload on different servers, with improved security level. In addition, the overhead for data transfer could be reduced since part of the display logic is being done by client side.

The development was done using java, PHP and MongoDB. Apache tomcat server is used for providing web service through calling the system API of the Geo-Social Network, while apache is used for running PHP server pages. The web service, together with the system API, was implemented in java, the web service will response the request from PHP in terms of JSON. Upon receiving the data from web service, the PHP server pages will process the data and return it to the client for displaying the request result. The system was implemented with Object-Oriented and MVC approach for extensibility and flexibility.

1. Introduction

1.1 Overview

Geo-social Network focuses on providing location-based service to users, for example, check-in and finding nearby places for the current location. The project aims to develop a Geo-social Network as mobile and web application for HKUST students that would allow them to check-in and find their friends nearby, and to communicate on the personal status through postings. The web application acts as a thin client such that no additional software is needed to be installed for providing service and functionality.

In this project, a document-based NoSQL(Not only SQL) database – MongoDB[3] is used in constructing the system. MongoDB differs from typical relational database base that it organizes data in form of separated documents instead of rows in tables.

It also support a special type of index called Geospatial index that facilitate the searching of nearby location with the given coordinates (longitude and latitude). Besides the database structure, efficiency to retrieve data is also one major concern. Extra effort will be placed on designing the document structure to avoid joins to tradeoff efficiency with space by duplicating some of the values across different collections (tables).

To provide students with better location-based service experience, we use the HTML5 together with 3G/WiFi technology to acquire the location of the terminals (e.g. phones, laptops). Knowing the location of the users, we can provide more location specific service to our users.

Our goal is to create a Geo-Social Network for HKUST students to share their experience, as well as making new friends throughout their university life.

1.2 Objectives

Geo-social Network provides location-based service to users. Functions are provided based on the current user location. In order to facilitate user check-in, collection of location points in UST would have to be done in order to find out the venue that user is located.

The system consists of four major components:

- Database (MongoDB): a commercial document based database which is used in industry supporting Geospatial Indexing which can be used to find nearby location for the specified coordinates(latitude and longitude).
- System API with RESTful Web Service (Implemented in java)
- Web Server
- Graphical User Interface

The functions are grouped into different pages as the following:

1. Home Page
2. Wall Page
3. Profile Page
4. Friend Page

5. Notification Page
6. Event Page
7. Check-in Page

The system will have to record all the user check-ins and search for check-ins having shortest distance for the current user location and return the related users. Every users and events are provided with a “Wall” so that users and their friends can post comments and also replying the others. They may also create events for their friends to join.

1.3 Literature Survey

Foursquare

Foursquare[1] is a location-based application that allows users to check-in at different places. For every check-in, users can perform actions to interact with other users and to share their experience. It also rewards users with points and “badges” for every check-in. In return, users can get coupons or discounts as the number of days of check-ins is achieved.

Google+

Google+[2] is a Social Networking Services that help users to manage their friends in different social circles. Other functions of Google+ are quite similar to that of Facebook. But it does have a post recommendation that let users to explore some funny or special posts every day. Besides, Google+ can bridge with other service provided by Google such as Google Calendar, Play Store and Google Docs etc.

RESTful Web Services

A RESTful web service[5] is a web service implemented using HTTP and the principles of REST. It basically means that each unique URL is a representation of some object and emphasis on readability. Given that RESTful doesn't require message header like SOAP, it therefore saves bandwidth for data transfer. The main advantages of RESTful web services are: Lightweight, Human Readable Results and easy to setup [6].

Haversine Formula

Haversine Formula[7] is used for calculating the distance between two points on a sphere. With this formula, it makes us possible to get the distance between two users with their latitude and longitude location, the formula is given by:

$$\text{haversion}\left(\frac{d}{r}\right) = \text{haversion}(\phi_2 - \phi_1) + \cos(\phi_1) \cos(\phi_2) \text{haversion}(\lambda_2 - \lambda_1)$$

d is the distance between the two points

r is the radius of the sphere, i.e. radius of earth, in this project r is taken as 6367km

ϕ_1, ϕ_2 : latitude of point 1 and latitude of point 2

λ_1, λ_2 : longitude of point 1 and longitude of point 2

$$d = 2r \arcsin\left(\sqrt{\text{haversion}(\phi_2 - \phi_1) + \cos(\phi_1) \cos(\phi_2) \text{haversion}(\lambda_2 - \lambda_1)}\right)$$

2. Design

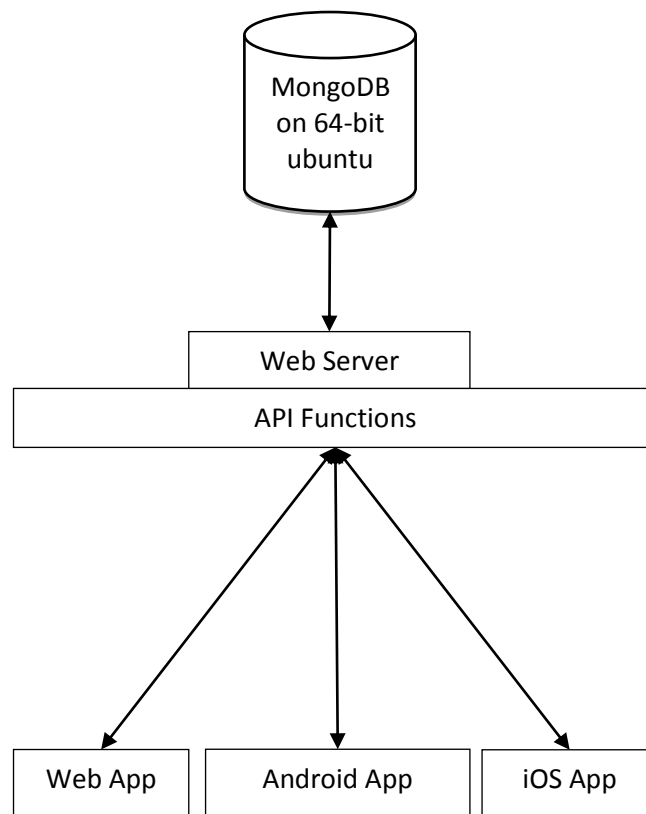
2.1 System Architecture

Back-end

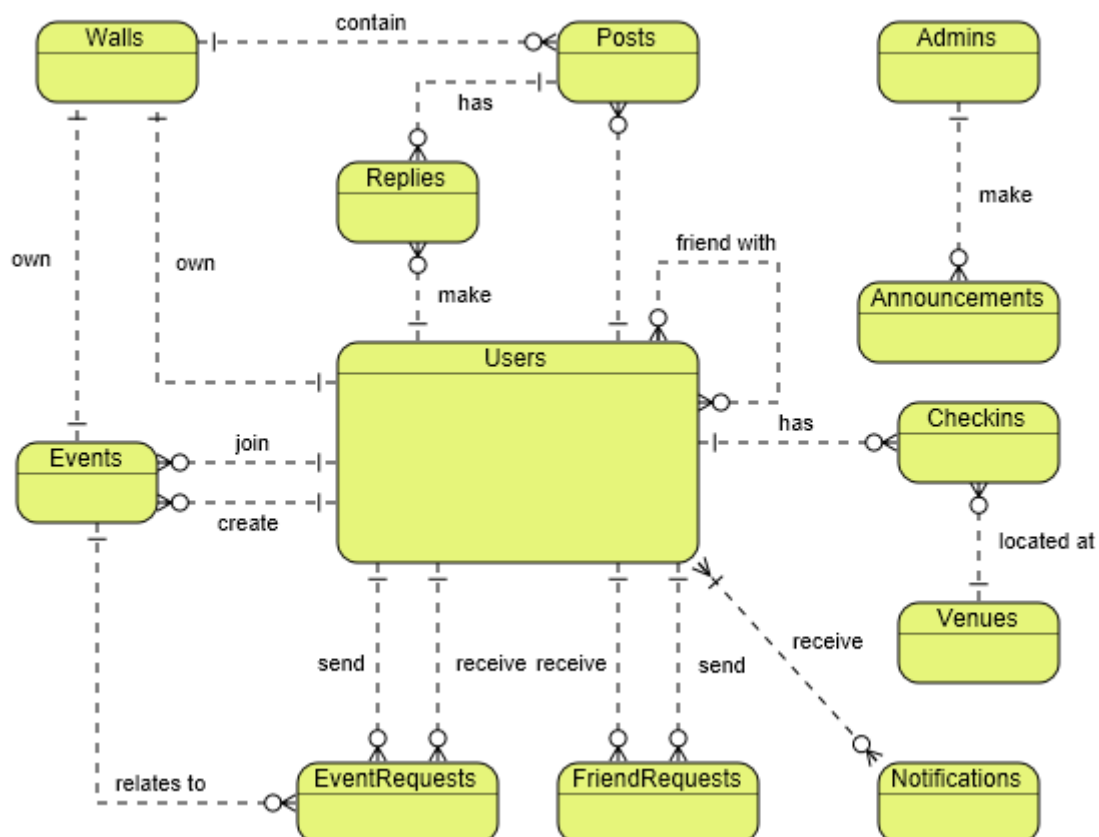
- MongoDB (database)
- System API (java)
- Server Setup (tomcat, apache)
- Web Service (RESTful)

Front-end

- Web Application (php)
- Web Pages (html, css, javascript)



2.2 Database Design (Conceptual)



2.3 Database Design (MongoDB)

Since MongoDB is a non-relational database, the following is the document structure stored in the “Collections” (similar to Tables in RDBMS) of the database. There are totally 11 collections in the database: Users, Admins, Announcements, Venues, Checkins, Posts, FriendRequests, Events, EventRequests, Notifications and IncrementKeys. In order to view the document structure clearly, optional fields are also shown below with “int” denoting numerical data, “str” denoting string/character data and “date” denoting datetime, “(opt)” represents that the field is optional. There are some data being duplicated in the database, e.g. userId, name and profilePic, this is to avoid expensive join operations and to increase the efficiency to retrieve data by the trade-off between space and efficiency.

```
// contact.emailAddress cannot duplicate, username must be unique
// Index on userId, wallId, contact.emailAddress
// 2d Index on lastCheckin.location
Users
{
  userId: int,
  wallId: int,
  username: str,
  pswd: str,
  firstName: str,
  middleName(opt): str,
  lastName: str,
  gender: "M"/"F",
  home_address(opt): [line1, line2, line3],
  contact:
  {
    emailAddress: str,
    emailBackup(opt): str,
    homePhone(opt): str,
    officePhone(opt): str,
    mobilePhone(opt): str,
  },
  profilePic: path,
  dateOfBirth: date,
  aboutMe(opt): str,
  friends:
  {
    count: int,
    list:
    [
      { userId: int, name: str, profilePic: str }, { userId: int, name: str, profilePic: str }.....
    ]
  },
  lastCheckin: {
    checkinId: int,
    venueId: int,
    venueName: str,
    location: [longitude, latitude],
    checkinDate: date
  },
}
```



```

lastLogin(opt):
{
  sessionId: str,
  loginDate: date
},
registerDate: date,
numOfRequest: { friendRequest: int, eventRequest: int, notification: int },
locked(opt): { lockDate: date, lockedBy: { adminId: int, email: str } },
}

```

// email cannot duplicate, username must be unique

// Index on username

Admins

```

{
  adminId: int,
  username: str,
  pswd: str,
  email: str,
  createDate: date
}

```

// Index on announceld

Announcements

```

{
  announceld: int,
  announceBy: { adminId: int, username: str },
  content: str,
  createDate: date,
  effectDate: date,
  expiryDate(opt): date,
  deleted(opt): { deleteDate: date, deletedBy: { adminId: int, email: str } }
}

```

// Index on venueld

// 2d Index on location

Venues

```

{
  venueld: int,
  venueName: str,
  location: [longitude, latitude],
  photoPath(opt): str,
  stats: { distinctCheckin: int, totalCheckin: int }
}

```

```
// Index on checkinId
// 2d Index on location
Checkins
{
  checkinId: int,
  user: { userId: int, name: str, profilePic: str },
  venueId: int,
  venueName: str,
  location: [longitude, latitude],<--checkin location, NOT Venue location
  checkinDate: date
}
```

```
// Index on postId, wallId
Posts
{
  postId: int,
  wallId: int,
  wallOwner: { userId: int, name: str, profilePic: path },
  wallOwner(for posts on event's wall): { eventId: int, title: str },
  author: { userId: int, name: str, profilePic: path },
  authorComment: str,
  photoPath(opt): str,
  numOfReply: int,
  postDate: date,
  lastReplyDate: date,
  replier: [
    { userId: int, name: str, profilePic: path },
    { userId: int, name: str, profilePic: path }.....
  ],
  reply:
  [
    {
      replyId: int,
      authorId: int,
      authorComment: str,
      replyDate: date,
      editDate(opt): date
    },
    {
      replyId: int,
      authorId: int,
      authorComment: str,
      replyDate: date
      editDate(opt): date
    }.....
  ]
}
```

```

// Index on requestId
FriendRequests
{
  requestId: int,
  from: { userId: int, name: str, profilePic: path },
  to: { userId: int, name: str, profilePic: path },
  requestMessage(opt): str,
  sendDate: date,
  replyDate(opt): date(empty if pending),
  status: "pending"/"accepted"/"rejected"
}

// Index on eventId, wallId
Events
{
  eventId: int,
  wallId: int,
  creator: { userId: int, name: str, profilePic: path },
  title: str,
  duration: { start: date, end: date },
  applyDate(opt): { start: date, end: date },
  venue: str,
  venue(for venue inside HKUST):
  {
    venueld: int,
    venueName: str,
    location: [longitude, latitude],
    photoPath(opt): str
  },
  eventContent: str,
  createDate: date,
  usersJoined:
  {
    count: int,
    list:
    [
      { userId: int, name: str, profilePic: path, dateAdded: date },
      { userId: int, name: str, profilePic: path, dateAdded: date }.....
    ]
  },
  usersInvited:
  {
    count: int,
    list:
    [
      { userId: int, name: str, profilePic: path, dateAdded: date },
      { userId: int, name: str, profilePic: path, dateAdded: date }.....
    ]
  }
}

```

```

// Index on evtrequestId
EventRequests
{
    evtrequestId: int,
    eventId: int,
    title: str,
    duration: [startDate, endDate],
    applyDate: [startDate, endDate],
    from: { userId: int, name: str, profilePic: path },
    to: { userId: int, name: str, profilePic: path },
    requestMessage(opt): str,
    sendDate: date,
    replyDate(opt): date(empty if pending),
    status: "pending"/"accepted"/"rejected"
}

// userIds is a list of users who should receive the notification
// 1) eventId attribute exist when a friend accept the event request
// 2) postId attribute exist when a friend reply a post or post on users wall
// 3) BOTH eventId and postId exist when a user posts on event's wall
Notifications
{
    userIds: [int, int, int...],
    friend: { userId: int, name: str, profilePic: path },
    eventId: int,
    postId: int,
    type: "event"/"friend"/"reply"/"post",
    dateAdded: date
}

// Storing the auto-increment Ids
IncrementKey
{
    idName: <id_name>,
    currentId: int
}

```

2.4 API Design

The following is a conceptual view of classes inside the system API.

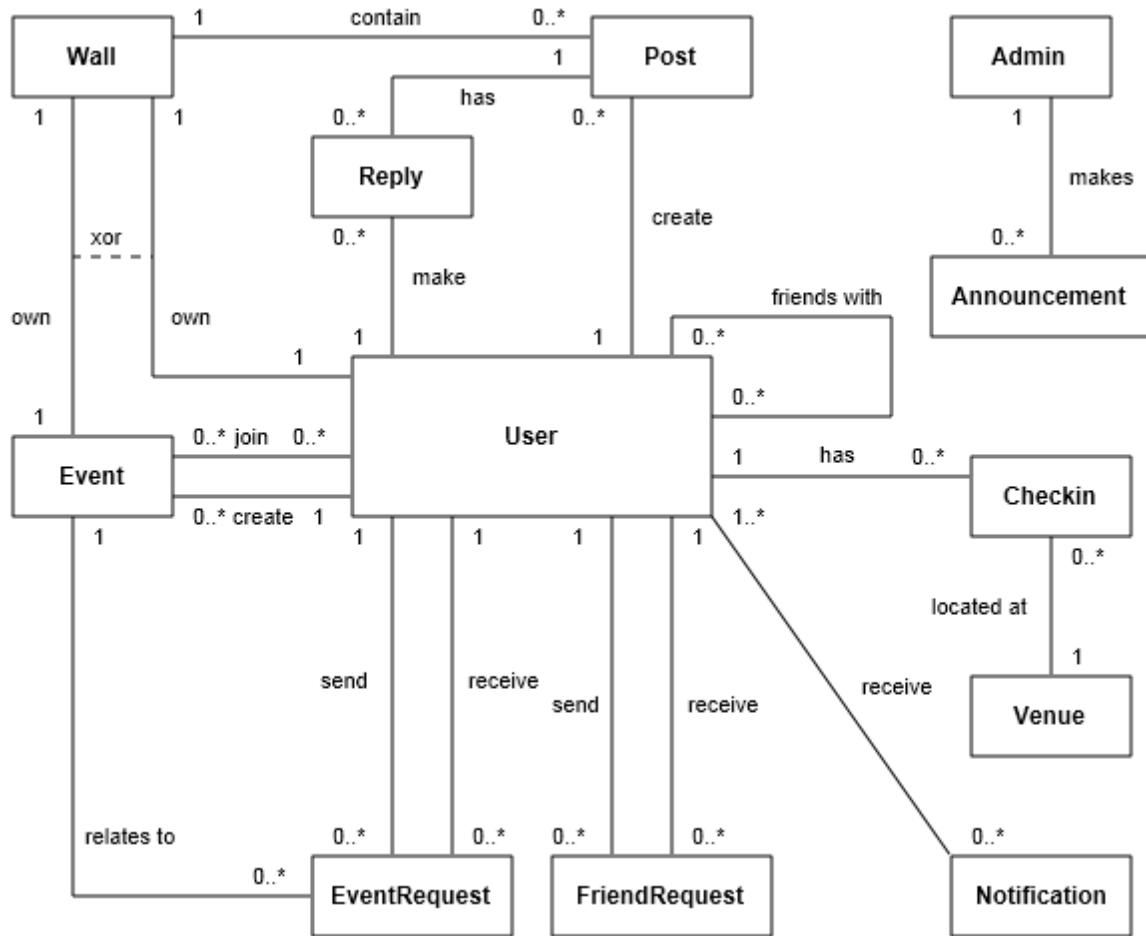


Diagram 1

The system API mainly comprise two parts, the database connection component and the collection class for providing the web service with functionality to access and manipulate the database. To provide interface for different database system in the future, the interface for collections are written using template parameters to suit the object type for different database drivers.

Diagram 2 is the class diagram for the database connection component, DBConnection class is responsible for connecting the database whereas GSNConstants class stores some constants in the system, such as the IP address and port number for the database (Mongo. With the database connection decoupled, the system can be extended and updated by overriding the methods in the DBConnection class in case the database system is switched. For simplicity, the attributes and operations of CollectionBase are not shown.

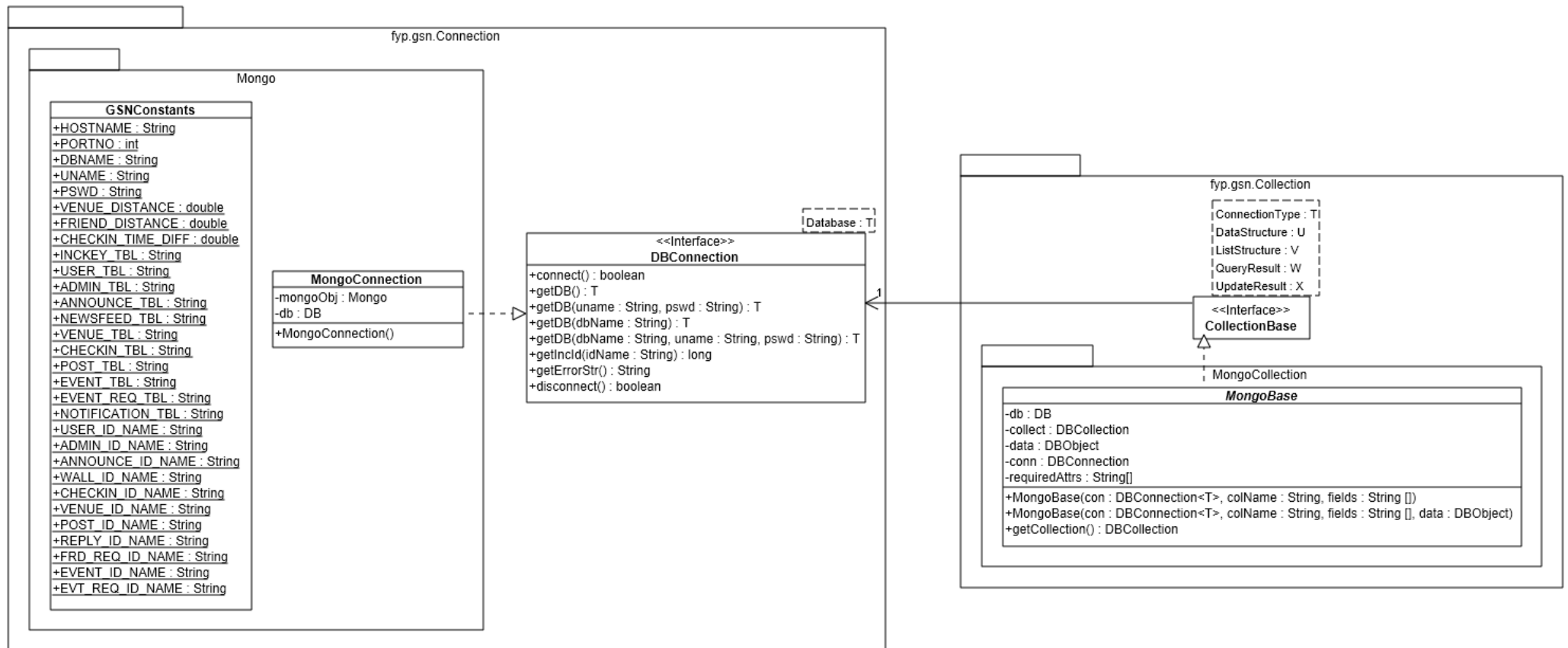


Diagram 2

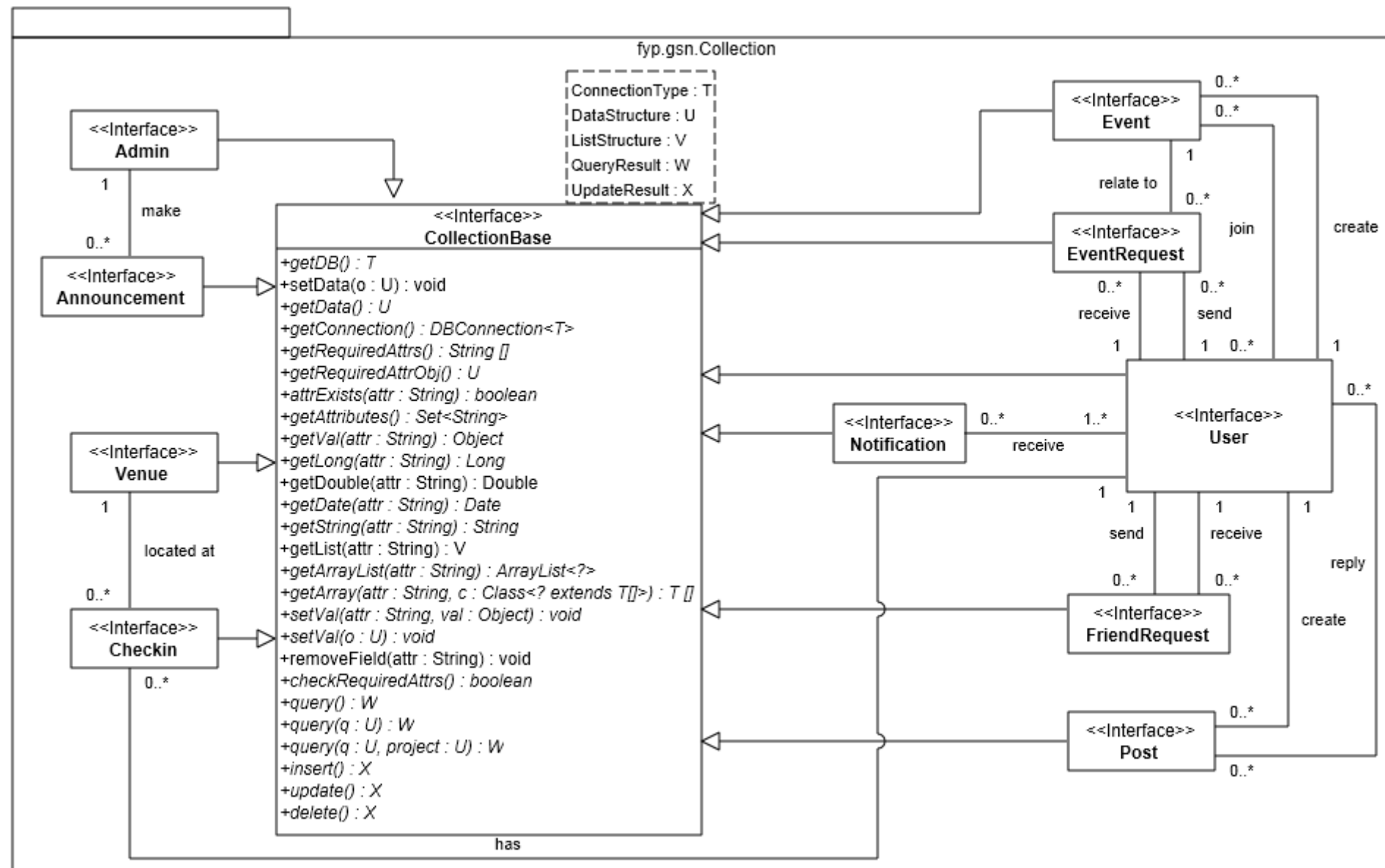


Diagram 3

The following diagram shows the methods for interface in diagram 3. For simplicity, all operations and attributes for CollectionBase are hidden.

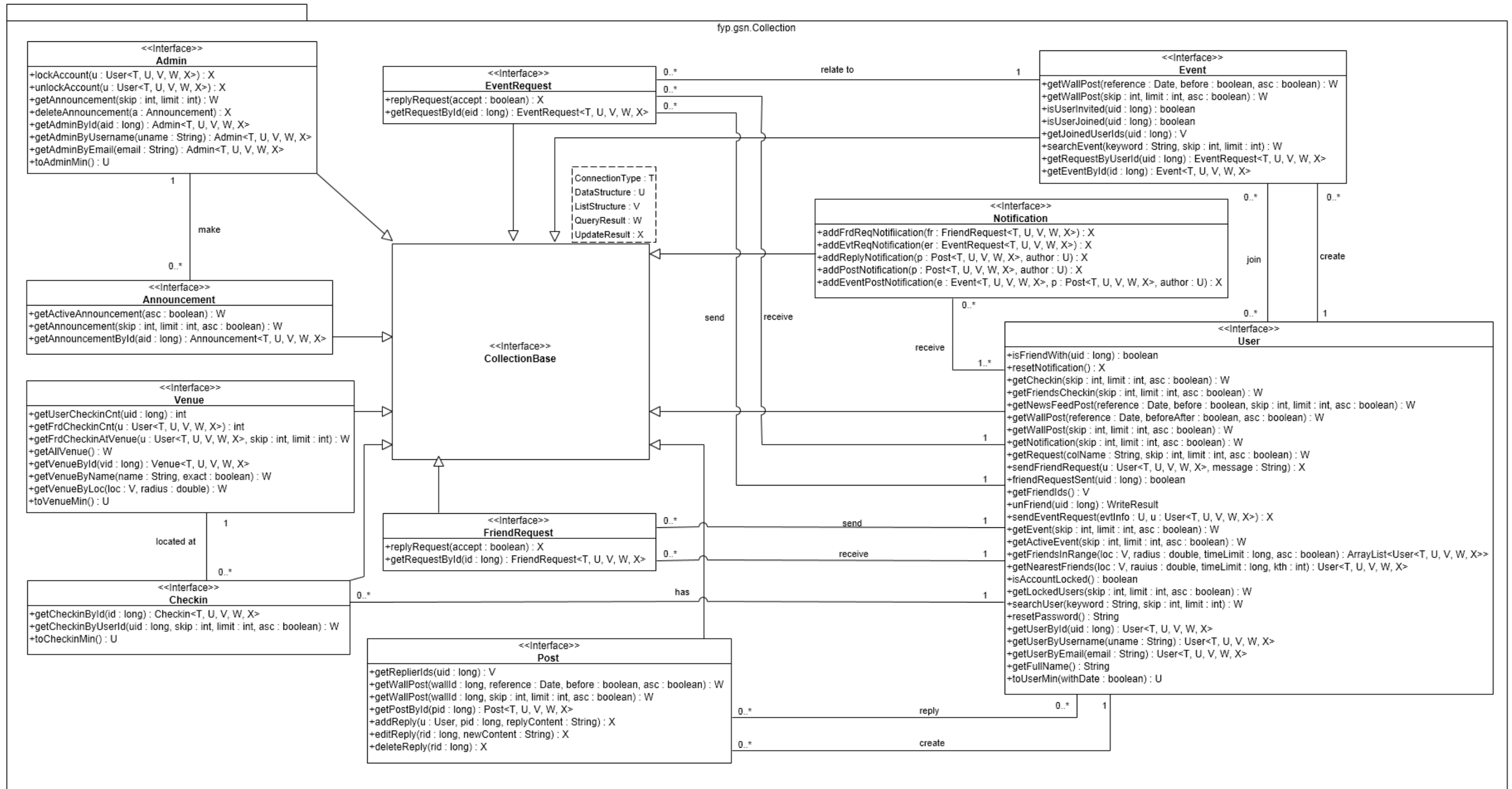


Diagram 4

All the classes in the following diagram implements the corresponding interface in diagram 4, for simplicity and readability, the realization relationship are not shown below. The MongoClient (fyp.gsn.Collection.MongoCollection) package is located inside the fyp.gsn.Collection package.

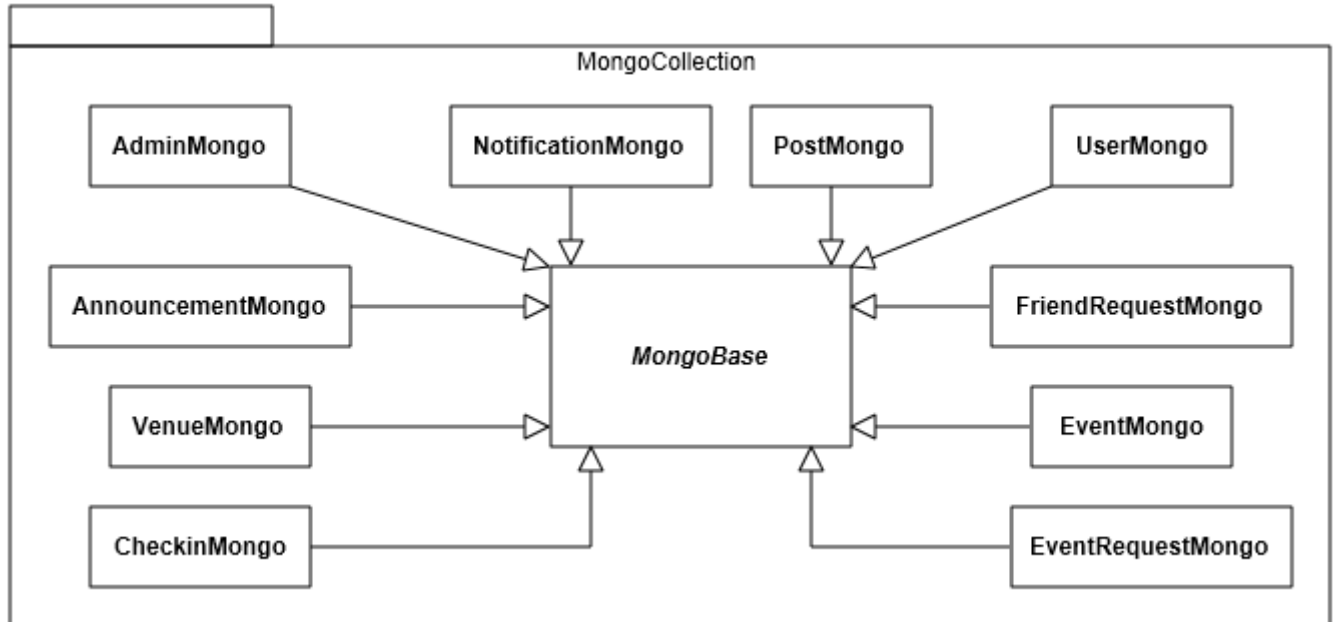
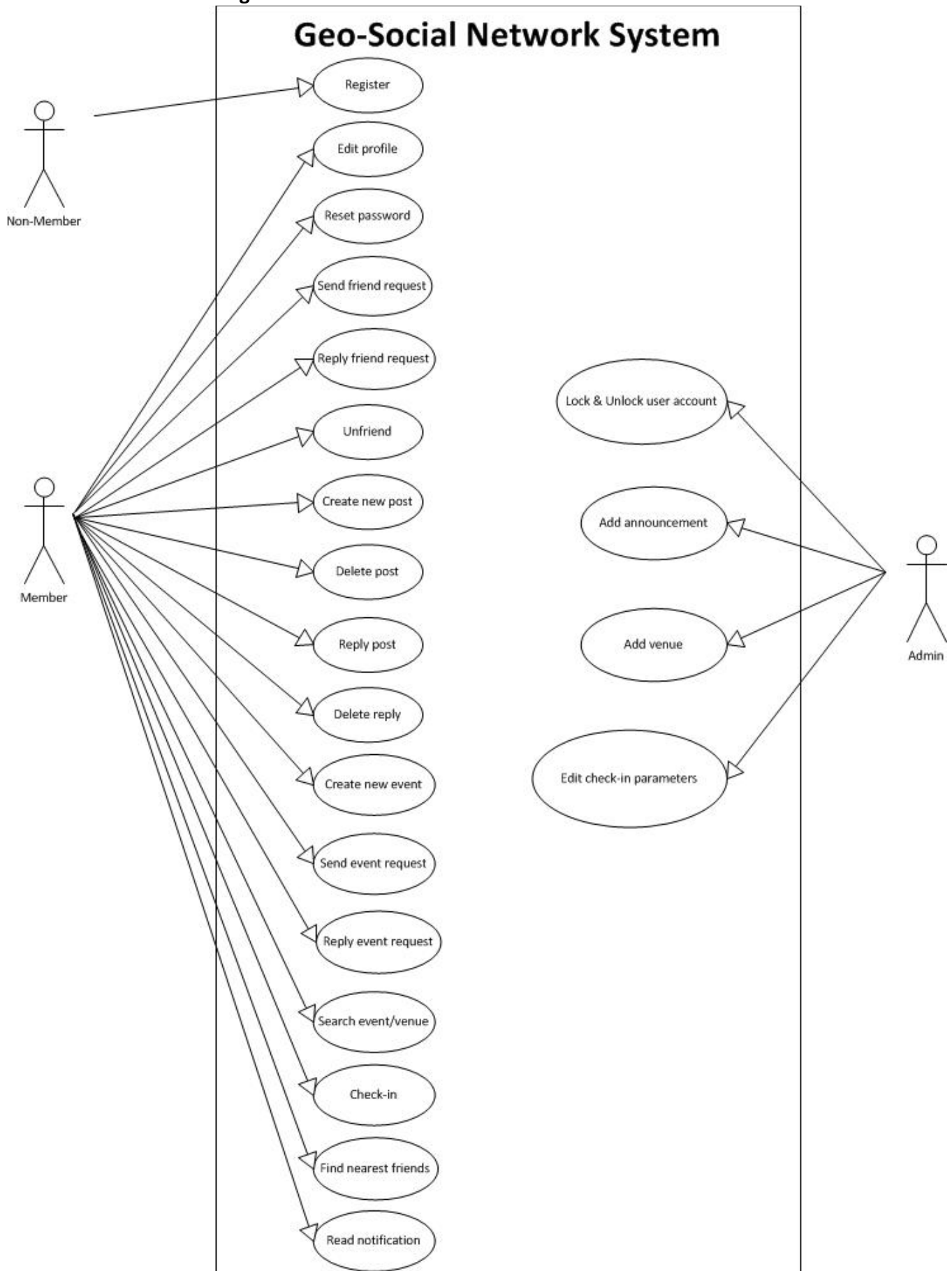


Diagram 5

2.5 Use Case

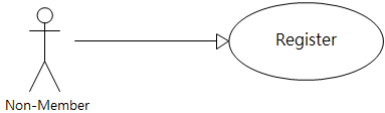
2.5.1 Use Case Diagram

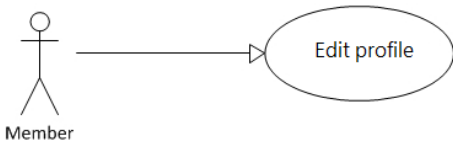


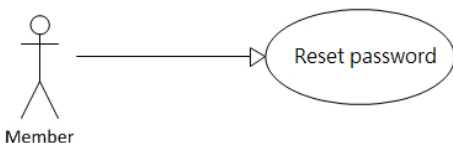
2.5.2 Actor Description

Actor	Description
Admin	This actor performs announcement management, user account management and venue management. She/he can configure the global parameters such as radius and time limit of check-in functions.
Non-Member	This actor can view the homepage of the system. She/he can register as a member to use our system.
Member	This actor is the main user of the system. She/he can perform a lot of social activities such as check-ins, make friends, create posts and create events. She/he will be interacted with other users most of the times.

2.5.3 Use Case Description

Use case 1: Register	
Description:	This use case describes how a user registers the system to become a member.
Use case diagram:	 <pre> graph LR Non-Member((Non-Member)) --- Register(Register) </pre>
Precondition:	The user has not yet registered.
Basic flows:	<ol style="list-style-type: none"> 1 The use case starts when the user clicks the register button. 2 Registration Page is shown. <p>{Enter information}:</p> <ol style="list-style-type: none"> 3 The user provides the following information in order to register: username, password, title, first name, last name, gender, email address, date of birth. Optional: profile picture, middle name, home address, backup email address, home/office/mobile phone number, and about me. <p>{Submit registration form}</p> <ol style="list-style-type: none"> 3.1 Submit button is clicked, user confirms the information is correct. <ol style="list-style-type: none"> 3.1.1 User selects yes, the new member is added into the database. 3.1.2 User selects no, return to the registration form. <ol style="list-style-type: none"> 4 The use case ends.
Alternative flows:	<p>A1: Missing or invalid information is entered</p> <p>At {Submit registration form} some information is invalid or missing.</p> <ol style="list-style-type: none"> 1. The system informs the user which part of the form is missed or invalid. 2. The flow back to {Enter information}. <p>A2: Username/Email address has already existed.</p> <p>At {Submit registration form} the system checks that the user username/email address already exists.</p> <ol style="list-style-type: none"> 1. The system informs the user that the username/email address has already existed. 2. The flow of events is resumed at {Enter information}.
Post-conditions:	User can login with the registered account.
Non-functional requirements:	<ol style="list-style-type: none"> 1. After registration, username and email address cannot be changed.

Use case 2: Edit profile	
Description:	This use case describes how a user edits his/her profile.
Use case diagram:	 <pre> graph LR Member[Member] --> EditProfile((Edit profile)) </pre>
Precondition:	User has logged in as Member.
Basic flows:	<ol style="list-style-type: none"> The use case starts when the user clicks his/her name to enter the profile page. Profile page is shown. User clicks the edit button. <p>{Enter information}:</p> <ol style="list-style-type: none"> The user provides the following information in order to edit profile: Optional: title, first name, last name, gender, date of birth, profile picture, middle name, home address, backup email address, home/office/mobile phone number, and about me. <p>{Submit form}</p> <ol style="list-style-type: none"> Submit button is clicked, user confirms the information is correct. <ol style="list-style-type: none"> User selects yes, the profile is edited. User selects no, return to the profile page. <ol style="list-style-type: none"> The use case ends.
Alternative flows:	<p>A1: Missing or invalid information is entered</p> <p>At {Submit form} some information is invalid or missing.</p> <ol style="list-style-type: none"> The system informs the user which part of the form is missed or invalid. The flow back to {Enter information}
Post-conditions:	User's profile is updated.
Non-functional requirements:	N/A

Use case 3: Reset password	
Description:	This use case describes how a user performs password recovery.
Use case diagram:	 <pre> graph LR Member[Member] --> ResetPassword((Reset password)) </pre>
Precondition:	User has registered an account.
Basic flows:	<ol style="list-style-type: none"> The use case starts when the user clicks forgot password button. Password recovery form is shown. <p>{Enter information}:</p> <ol style="list-style-type: none"> The user provides the following information in order to recovery the password: username, email address. <p>{Submit form}</p> <ol style="list-style-type: none"> Submit button is clicked, user confirms the information is correct. <ol style="list-style-type: none"> User selects yes, the system generates a new password and send it to the user's email. User selects no, return to the password recovery form. <ol style="list-style-type: none"> The use case ends.
Alternative flows:	<p>A1: Missing or invalid information is entered</p> <p>At {Submit form} some information is invalid or missing.</p> <ol style="list-style-type: none"> The system informs the user which part of the form is missed or invalid

	(username and email address do not match or not exist). 2. The flow back to {Enter information}
Post-conditions:	User can login using the new password.
Non-functional requirements:	N/A

Use case 4: Send friend request	
Description:	This use case describes how a user sends a friend request to others in order to make friends.
Use case diagram:	<pre> graph LR Member((Member)) --> SendFriendRequest([Send friend request]) </pre> <p>The diagram shows a stick figure actor labeled 'Member' connected by a solid line with an open arrowhead to an oval use case labeled 'Send friend request'.</p>
Precondition:	User has logged in as Member. Sender and receiver is not a friend of each other.
Basic flows:	<ol style="list-style-type: none"> The use case starts when the user searches for the member she/he wants to add. A list of member is shown. <p>{Enter messages}:</p> <ol style="list-style-type: none"> The user enters the message she/he want to send to receiver for making friend (Optional). <p>{Submit request}</p> <ol style="list-style-type: none"> 3.1 Submit button is clicked, user confirms the request is correct. <ol style="list-style-type: none"> 3.1.1 User selects yes, a friend request and notification is sent to the member selected. 3.1.2 User selects no, return to the list. <ol style="list-style-type: none"> The use case ends.
Alternative flows:	<p>A1: Friend request is sent before</p> <p>At {Submit request}</p> <ol style="list-style-type: none"> The system informs the user a friend request is sent before The flow back to {Enter message}.
Post-conditions:	N/A
Non-functional requirements:	N/A

Use case 5: Reply friend request	
Description:	This use case describes how a user replies a friend request. She/he can either accept or deny the friend request.
Use case diagram:	<pre> graph LR Member((Member)) --> ReplyFriendRequest([Reply friend request]) </pre> <p>The diagram shows a stick figure actor labeled 'Member' with an arrow pointing to an oval use case labeled 'Reply friend request'.</p>
Precondition:	User has logged in as Member. A friend request is received.
Basic flows:	<ol style="list-style-type: none"> 1 The use case starts when the user clicks on the notification of friend request. 2 A list of friend request is shown. <p>{Submit reply}</p> <ol style="list-style-type: none"> 2.1 Accept button is clicked, user and the request sender becomes a friend, and the sender will receive a notification. 2.2 Deny button is clicked, no notification will be sent to the sender. <ol style="list-style-type: none"> 3 The use case ends.
Alternative flows:	N/A
Post-conditions:	N/A
Non-functional requirements:	N/A

Use case 6: Unfriend	
Description:	This use case describes how a user unfriend a friend. The friend list will be updated.
Use case diagram:	<pre> graph LR Member((Member)) --> Unfriend([Unfriend]) </pre> <p>The diagram shows a stick figure actor labeled 'Member' with an arrow pointing to an oval use case labeled 'Unfriend'.</p>
Precondition:	User has logged in as Member.
Basic flows:	<ol style="list-style-type: none"> 1 The use case starts when the user clicks on Friends. 2 A list of friend is shown. 3 The user clicks on the unfriend button below the friend she/he wants to unfriend. <p>{Submit request}</p> <ol style="list-style-type: none"> 3.1 Unfriend button is clicked, user confirms the request is correct. <ol style="list-style-type: none"> 3.1.1 User selects yes, the friend selected will be unfriend. 3.1.2 User selects no, return to the friend list. <ol style="list-style-type: none"> 4 The use case ends.
Alternative flows:	N/A
Post-conditions:	N/A
Non-functional requirements:	N/A

Use case 7: Create new post	
Description:	This use case describes how a user creates a post on wall. Thus, other users can view and reply to the post.
Use case diagram:	<pre> graph LR Member((Member)) --> CreateNewPost(Create new post) </pre> <p>The diagram shows a stick figure actor labeled 'Member' connected by an arrow to an oval use case labeled 'Create new post'.</p>
Precondition:	User has logged in as Member.
Basic flows:	<ol style="list-style-type: none"> 1 The use case starts when the user wants to post something on its wall. 2 A text box for entering message is shown. <p>{Enter messages}:</p> <ol style="list-style-type: none"> 3 The user enters the message she/he wants to post. <p>{Submit post}</p> <ol style="list-style-type: none"> 3.1 Submit button is clicked, user's comment will be posted on the wall. 4 The use case ends.
Alternative flows:	<p>A1: Post on friend walls</p> <p>At the beginning of the use case, the user can go to friend's wall</p> <ol style="list-style-type: none"> 1 A text box for entering message is shown. 2 The flow back to {Enter message}. <p>After {Submit post}</p> <ol style="list-style-type: none"> 1 A post notification will be sent to the wall owner. 2 The use case ends. <p>A2: Post on event walls</p> <p>At the beginning of the use case, the user can go to event's wall</p> <ol style="list-style-type: none"> 1 A text box for entering message is shown. 2 The flow back to {Enter message}. <p>After {Submit post}</p> <ol style="list-style-type: none"> 1 A post notification will be sent to all members who have joined the event. 2 The use case ends.
Post-conditions:	The post can be viewed on the wall posted.
Non-functional requirements:	N/A

Use case 8: Delete post	
Description:	This use case describes how a user deletes a post.
Use case diagram:	<pre> graph LR Member((Member)) --> DeletePost([Delete post]) </pre> <p>The diagram shows a stick figure actor labeled 'Member' connected by a solid line with an open arrowhead to an oval use case labeled 'Delete post'.</p>
Precondition:	User has logged in as Member. There is at least a post on the wall that the user created.
Basic flows:	<ol style="list-style-type: none"> 1 The use case starts when the user wants to delete a post. 2 User goes to the wall that can see the post. 3 The user clicks the delete button below the post. {Submit request} <ol style="list-style-type: none"> 3.1 Delete button is clicked, user confirms the request is correct. <ol style="list-style-type: none"> 3.1.1 User selects yes, the post will be deleted. 3.1.2 User selects no, return to post. 4 The use case ends.
Alternative flows:	N/A
Post-conditions:	The post cannot be viewed.
Non-functional requirements:	N/A

Use case 9: Reply post	
Description:	This use case describes how a user replies to a post. Thus, other users can view and reply to your reply.
Use case diagram:	<pre> graph LR Member((Member)) --> ReplyPost([Reply post]) </pre> <p>The diagram shows a stick figure actor labeled 'Member' connected by a solid line with an open arrowhead to an oval use case labeled 'Reply post'.</p>
Precondition:	User has logged in as Member. There is at least a post on the wall the user wants to reply.
Basic flows:	<ol style="list-style-type: none"> 1 The use case starts when the user wants to reply a post. 2 User goes to the wall that can see the post. {Enter messages}: 3 The user enters the message she/he wants to reply under the post. {Submit reply} <ol style="list-style-type: none"> 3.1 Submit button is clicked, user's comment will be replied to the post. 3.2 Notification will be sent to all member who was replied the post and the member who created the post. 4 The use case ends.
Alternative flows:	N/A
Post-conditions:	The reply can be viewed on the post.
Non-functional requirements:	N/A

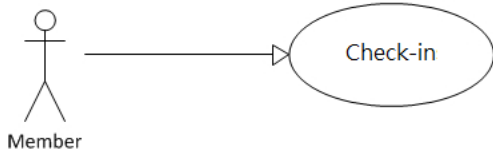
Use case 10: Delete reply	
Description:	This use case describes how a user deletes a reply. Thus, other users cannot view the reply.
Use case diagram:	<pre> graph LR Member((Member)) --> DeleteReply([Delete reply]) </pre> <p>The diagram shows a stick figure actor labeled 'Member' connected by a solid line with an open arrowhead to an oval use case labeled 'Delete reply'.</p>
Precondition:	User has logged in as Member. There is at least a reply that the user replied to a post.
Basic flows:	<ol style="list-style-type: none"> The use case starts when the user wants to delete a reply. User goes to the wall that can see the post and reply. The user clicks the delete button next to her/his reply. <p>{Submit request}</p> <ol style="list-style-type: none"> <ol style="list-style-type: none"> Delete button is clicked, user confirms the request is correct. <ol style="list-style-type: none"> User selects yes, the reply will be deleted. User selects no, return to post. The use case ends.
Alternative flows:	N/A
Post-conditions:	The reply cannot be viewed on the post.
Non-functional requirements:	N/A

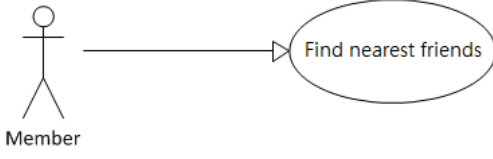
Use case 11: Create new event	
Description:	This use case describes how a user creates a new event for holding functions, gathering or parties. Thus, She/he can invite others to join the event.
Use case diagram:	<pre> graph LR Member((Member)) --> CreateNewEvent([Create new event]) </pre> <p>The diagram shows a stick figure actor labeled 'Member' connected by a solid line with an open arrowhead to an oval use case labeled 'Create new event'.</p>
Precondition:	User has logged in as Member.
Basic flows:	<ol style="list-style-type: none"> The use case starts when the user wants to create a new event for others to join. User goes to the event page and clicks create new events. Event creation form is shown. <p>{Enter details}:</p> <ol style="list-style-type: none"> The user enters the details of the event: title, start date, venue, and content. <p>{Submit event form}</p> <ol style="list-style-type: none"> <ol style="list-style-type: none"> Submit button is clicked, user confirms the information is correct. <ol style="list-style-type: none"> User selects yes, a new event to created and added in the database. User selects no, return to the creation form. The use case ends.
Alternative flows:	<p>A1: Missing or invalid information is entered</p> <p>At {Submit event form} some information is invalid or missing.</p> <ol style="list-style-type: none"> The system informs the user which part of the form is missed or invalid. The flow back to {Enter details}.
Post-conditions:	The event can be seen by the creator.
Non-functional requirements:	<ol style="list-style-type: none"> Start date should not be early than today.

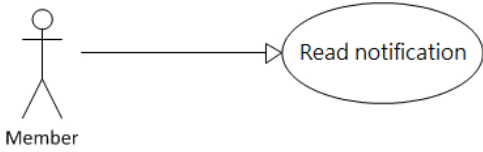
Use case 12: Send event request	
Description:	This use case describes how a user sends an event request to invite others to join the event. Thus, receiver can view the details of the event.
Use case diagram:	<pre> graph LR Member((Member)) --> UC([Send event request]) </pre> <p>The diagram shows a stick figure actor labeled 'Member' connected by a solid line with an open arrowhead to an oval use case labeled 'Send event request'.</p>
Precondition:	<p>User has logged in as Member.</p> <p>The user who is going to invite is not joined the event.</p>
Basic flows:	<ol style="list-style-type: none"> 1 The use case starts when the event creator wants to invite others to join the event. 2 The creator clicks send invitation and a list of friends is shown. <p>{Enter invitation messages}:</p> <ol style="list-style-type: none"> 3 The user enters the message she/he want to send (Optional). <p>{Submit request}</p> <ol style="list-style-type: none"> 3.1 Submit button is clicked, user confirms the request is correct. <ol style="list-style-type: none"> 3.1.1 User selects yes, an event request and notification is sent to the member selected. 3.1.2 User selects no, return to the list. <ol style="list-style-type: none"> 4 The use case ends.
Alternative flows:	<p>A1: Event request is sent before</p> <p>At {Submit request}</p> <ol style="list-style-type: none"> 1 The system informs the user an event request is sent before 2 The flow back to {Enter invitation message}. <p>A2: User joined the event already</p> <p>At {Submit request}</p> <ol style="list-style-type: none"> 1 The system informs the user that the receiver has joined the event already. 2 The flow back to {Enter invitation message}.
Post-conditions:	Receiver appears in the invitation list.
Non-functional requirements:	N/A

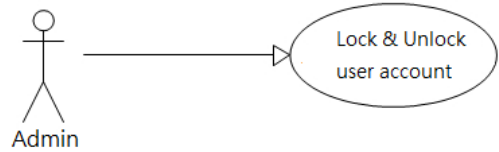
Use case 13: Reply event request	
Description:	This use case describes how a user replies an event request. She/he can either accept or deny the event request.
Use case diagram:	<pre> graph LR Member((Member)) --> UC13([Reply event request]) </pre> <p>The diagram shows a stick figure actor labeled 'Member' connected by a solid line with an open arrowhead to an oval use case labeled 'Reply event request'.</p>
Precondition:	User has logged in as Member. An event request is received.
Basic flows:	<ol style="list-style-type: none"> 1 The use case starts when the user clicks on the notification of event request. 2 A list of event request is shown. {Submit reply} <ol style="list-style-type: none"> 2.1 Accept button is clicked, user joined the event, and the sender will receive a notification. 2.2 Deny button is clicked, no notification will be sent to the sender. 3 The use case ends.
Alternative flows:	N/A
Post-conditions:	If user accepts the request, her/his name will appear in the joined list of the event
Non-functional requirements:	N/A

Use case 14: Search event/venue	
Description:	This use case describes how users search for events and venues. They can join the event or check-in after searching.
Use case diagram:	<pre> graph LR Member((Member)) --> UC14([Search event/venue]) </pre> <p>The diagram shows a stick figure actor labeled 'Member' connected by a solid line with an open arrowhead to an oval use case labeled 'Search event/venue'.</p>
Precondition:	User has logged in as Member.
Basic flows:	<ol style="list-style-type: none"> 1 The use case starts when the user wants to search events or venues using keyword. 2 User clicks on the textbox for searching. {Enter keyword}: 3 The user enters the keyword she/he wants to search. {Submit request} <ol style="list-style-type: none"> 3.1 Search button is clicked, the events and venues searched will be displayed separately. 4 The use case ends.
Alternative flows:	N/A
Post-conditions:	N/A
Non-functional requirements:	N/A

Use case 15: Check-in	
Description:	This use case describes how users check-in our system. Thus, others can see the check-in details of her/him. And others can find her/him by using check-in related functions.
Use case diagram:	 <pre> graph LR Member((Member)) --> CheckIn([Check-in]) </pre>
Precondition:	User has logged in as Member.
Basic flows:	<ol style="list-style-type: none"> 1 The use case starts when the user wants to check-in. 2 User turns on location service if the device is supported. 3 User clicks the check-in page. <p>{Get current location}:</p> <ol style="list-style-type: none"> 4 The user clicks get current location. 5 A list of venues near the user's position is shown. 6 User selects the venue she/he is located. <p>{Submit check-in}</p> <ol style="list-style-type: none"> 6.1 Check-in button is clicked, user confirms the information is correct. <ol style="list-style-type: none"> 6.1.1 User selects yes, a new check-in will be added to the database. 6.1.2 User selects no, return to the list of venues. 7 The use case ends.
Alternative flows:	N/A
Post-conditions:	A new check-in can be viewed by friends.
Non-functional requirements:	N/A

Use case 16: Find nearest friends	
Description:	This use case describes how users find the nearest friends, according to the last check-in of friends. Thus, she/he can know who is near her/him and the distance between them.
Use case diagram:	 <pre> graph LR Member((Member)) --> FindFriends([Find nearest friends]) </pre>
Precondition:	User has logged in as Member.
Basic flows:	<ol style="list-style-type: none"> 1 The use case starts when the user wants to find the friends which are near her/him. 2 User clicks find the nearest friends button. 3 A list of friend is shown 4 The details check-in information is displayed when user selects. 5 The use case ends.
Alternative flows:	N/A
Post-conditions:	N/A
Non-functional requirements:	N/A

Use case 17: Read notification	
Description:	This use case describes how users read notifications of friend requests, event requests, posts and replies. Thus. She/he can know the most updated information of activities.
Use case diagram:	 <pre> graph LR Member((Member)) --> ReadNotification([Read notification]) </pre>
Precondition:	User has logged in as Member. There is at least one notification of any kind of requests.
Basic flows:	<ol style="list-style-type: none"> 1 The use case starts when the user wants to read the notifications. 2 User clicks the notification tab, a list of notifications is shown. {Get details notifications}: <ol style="list-style-type: none"> 3 The user clicks on the notification she/he wants to see. 4 A detailed notification is shown 5 The use case ends.
Alternative flows:	N/A
Post-conditions:	The number of new notifications will be reset to 0
Non-functional requirements:	N/A

Use case 18: Lock & Unlock user account	
Description:	This use case describes how admin locks and unlocks a user's account when the users violate the rule of system. Thus, the user cannot login to the account.
Use case diagram:	 <pre> graph LR Admin((Admin)) --> LockUnlockAccount([Lock & Unlock user account]) </pre>
Precondition:	User has logged in as Admin. The user's account is not locked (Lock user account). The user's account is locked (Unlock user account).
Basic flows:	<ol style="list-style-type: none"> 1 The use case starts when the admin wants to lock a user's account. {Search the account}: <ol style="list-style-type: none"> 2 Admin searches for the user she/he wants to lock. {Lock the account}: <ol style="list-style-type: none"> 3 Lock/Unlock button is clicked, admin confirms the information is correct. <ol style="list-style-type: none"> 3.1.1 Admin selects yes, the user's account is locked. The user is not able to login. 3.1.2 Admin selects no, return to search the account. 4 The use case ends.
Alternative flows:	A1: Unlock a user account At {Search the account} <ol style="list-style-type: none"> 1 The system generates a list of locked accounts. At {Lock the account} <ol style="list-style-type: none"> 1 Unlock button is clicked, admin confirms the information is correct. <ol style="list-style-type: none"> 1.1.1 Admin selects yes, the user's account is unlocked. The user is able to login. 1.1.2 Admin selects no, return to search the account. 2 The use case ends.

Post-conditions:	The account which is locked cannot login. The account which is unlocked can login.
Non-functional requirements:	N/A

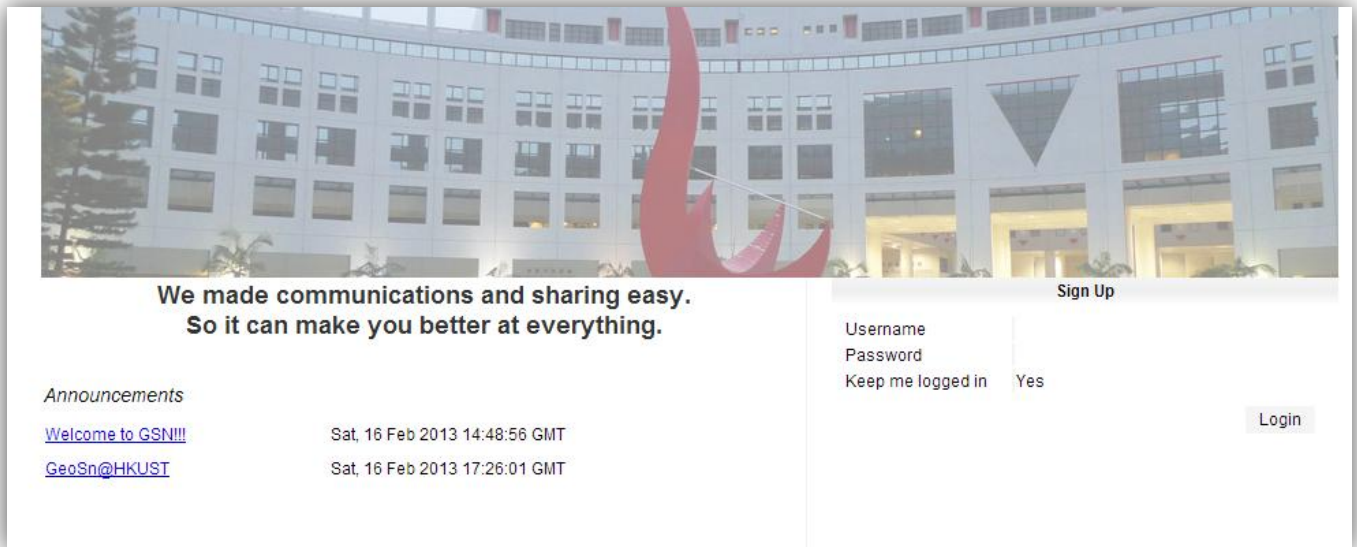
Use case 19: Add announcement	
Description:	This use case describes how admin adds new announcement which is displayed in the home page of our system. Thus, user can read the announcement when she/he goes to our website.
Use case diagram:	<pre> graph LR Admin((Admin)) --> AddAnnouncement((Add announcement)) </pre> <p>The diagram shows an actor labeled 'Admin' connected by a line to a use case labeled 'Add announcement' inside an oval.</p>
Precondition:	User has logged in as Admin.
Basic flows:	<ol style="list-style-type: none"> 1 The use case starts when the admin wants to add a new announcement to be displayed in the website. 2 Admin clicks add new announcements. <p>{Enter messages}:</p> <ol style="list-style-type: none"> 3 Admin enters the announcement content and effect date. Optional: expiry date. <p>{Submit the announcement}:</p> <ol style="list-style-type: none"> 4 Submit button is clicked, admin confirms the information is correct. <ol style="list-style-type: none"> 4.1.1 Admin selects yes, a new announcement is added in the database. 4.1.2 Admin selects no, return to enter messages. 5 The use case ends.
Alternative flows:	N/A
Post-conditions:	A new announcement is displayed.
Non-functional requirements:	<ol style="list-style-type: none"> 1 Effect date should be not early than today.

Use case 20: Add venue	
Description:	This use case describes how admin adds a new venue for user to check-in. Thus, the system has more precise location.
Use case diagram:	<pre> graph LR Admin[Admin] --> AddVenue((Add venue)) </pre> <p>The diagram shows an actor labeled 'Admin' connected by an arrow to a use case labeled 'Add venue'.</p>
Precondition:	User has logged in as Admin.
Basic flows:	<ol style="list-style-type: none"> 1 The use case starts when the admin wants to add a new venue for a new location point. 2 Admin clicks add new venue. <p>{Enter details}:</p> <ol style="list-style-type: none"> 3 Admin enters venue name, location point. Optional: photo. <p>{Submit the form}:</p> <ol style="list-style-type: none"> 4 Submit button is clicked, admin confirms the information is correct. <ol style="list-style-type: none"> 4.1.1 Admin selects yes, a new venue is added in the database. 4.1.2 Admin selects no, return to enter details. 5 The use case ends.
Alternative flows:	N/A
Post-conditions:	A new venue can be found by users.
Non-functional requirements:	N/A

Use case 21: Edit check-in parameters	
Description:	This use case describes how admin edits the check-in parameters (radius & time limit) which will be used for finding nearest friends.
Use case diagram:	<pre> graph LR Admin[Admin] --> EditCheckIn((Edit Check-in parameters)) </pre> <p>The diagram shows an actor labeled 'Admin' connected by an arrow to a use case labeled 'Edit Check-in parameters'.</p>
Precondition:	User has logged in as Admin.
Basic flows:	<ol style="list-style-type: none"> 1 The use case starts when the admin wants to edits the parameters for check-in related functions 2 Admin clicks edit parameters. <p>{Enter details}:</p> <ol style="list-style-type: none"> 3 Admin enters new radius and time limit. <p>{Submit the form}:</p> <ol style="list-style-type: none"> 4 Submit button is clicked, admin confirms the information is correct. <ol style="list-style-type: none"> 4.1.1 Admin selects yes, new radius and time limit are updated in the database. 4.1.2 Admin selects no, return to enter details. 5 The use case ends.
Alternative flows:	N/A
Post-conditions:	N/A
Non-functional requirements:	N/A

2.6 User Interface Design

Home Page



We made communications and sharing easy.
So it can make you better at everything.

Announcements

[Welcome to GSN!!!](#) Sat, 16 Feb 2013 14:48:56 GMT

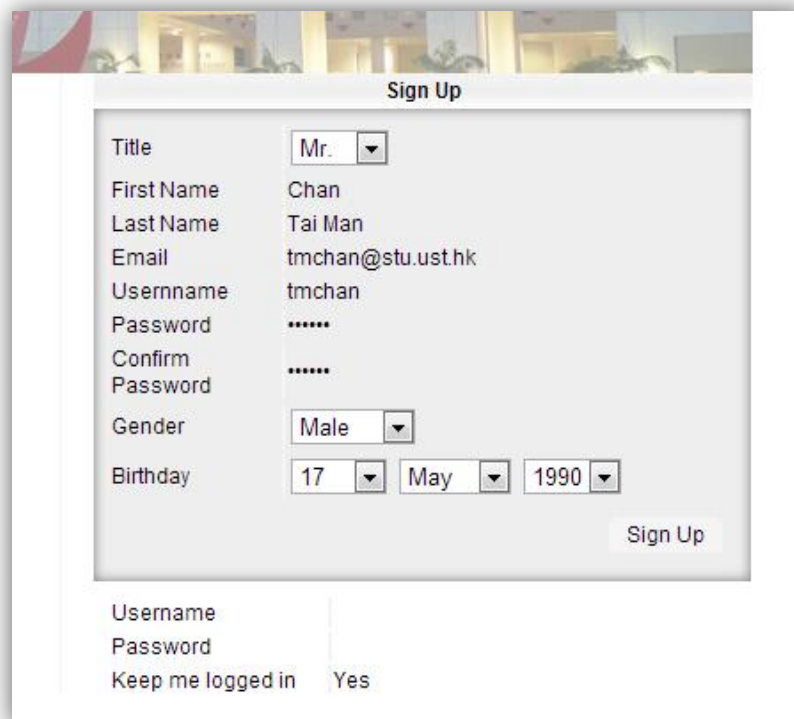
[GeoSn@HKUST](#) Sat, 16 Feb 2013 17:26:01 GMT

Sign Up

Username
Password
Keep me logged in Yes

Login

This is the home page of the system. The left side shows the active announcements, which are sorted by the posting time. The right side contains registration and login function. When the [Sign Up] button is clicked, a registration form will be expanded. User has to fill in all the information to register an account.



Sign Up

Title Mr. ▼

First Name Chan

Last Name Tai Man

Email tmchan@stu.ust.hk

Username tmchan

Password

Confirm Password

Gender Male ▼

Birthday 17 ▼ May ▼ 1990 ▼

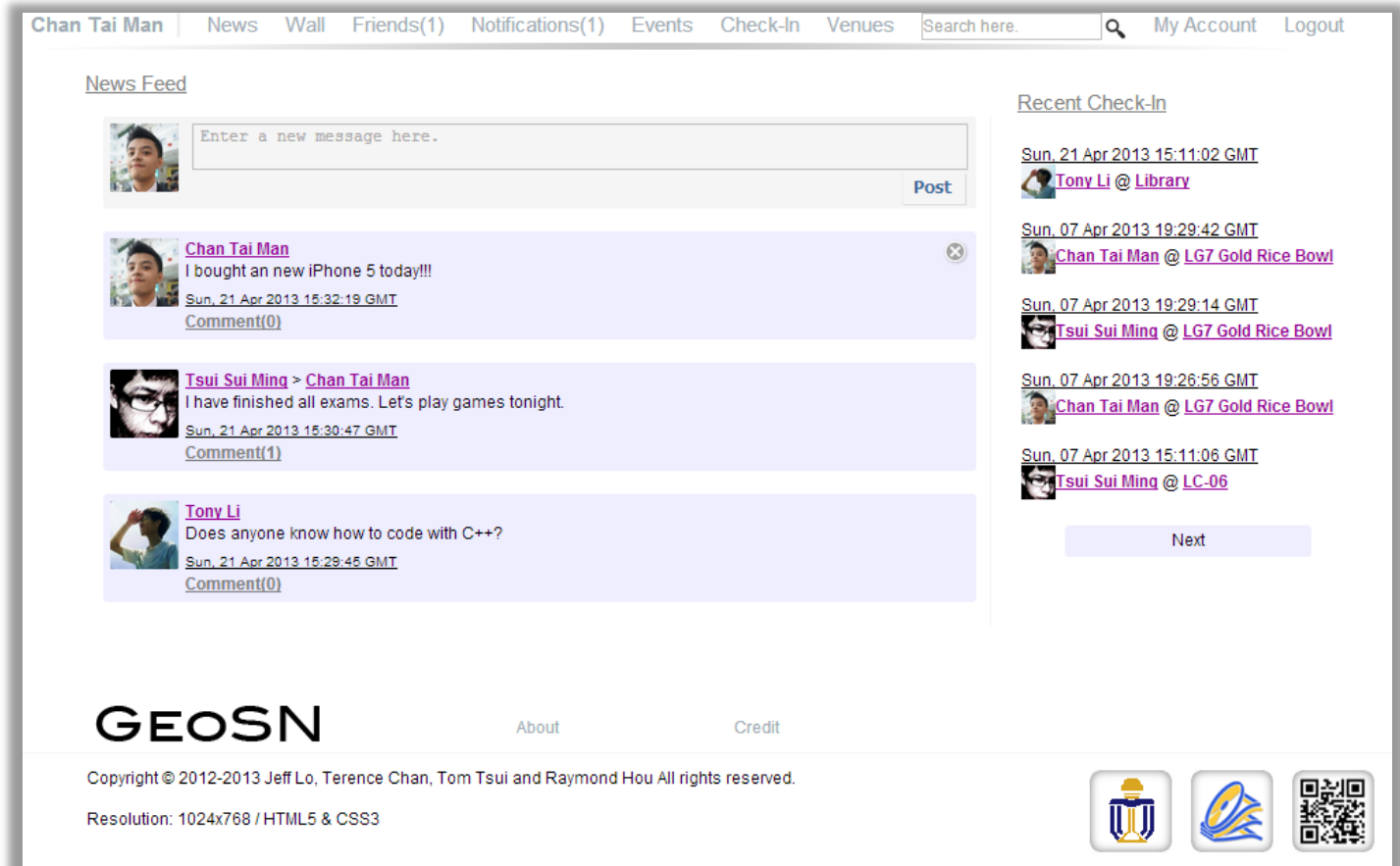
Sign Up

Username
Password
Keep me logged in Yes

After registration, the user can enter the username and password to login the system.

News Feed & Wall Page

After login the account, a news feed page will be shown below with a recent check-in histories next to it. Also, a menu will always appear at the top of page for accessing the pages conveniently, which consists of hyperlinks including profile, news feed, wall, friends, notifications, events, check-in, venues, account, logout and a search box at the middle.



The news feed page is a wall which can receive the news from user's friends such as user's post, their friends' post, replies of post, event news and check-in histories. User can post message on the own wall or their friend's wall. Also, the writer can remove the posted message or reply from the wall.

A list of check-in history about their friends is aligned to the right of news feed in which the time of check-in, the name of user and the name of venue are displayed. If the user clicks the name of venue, the location of the venue will be pointed out on the map.

Profile Page

Profile page displays the user's information, which can be read by other users. Also, if the user views his/her profile, the information can be modified.

The screenshot shows a web application interface for a user profile. At the top is a navigation bar with links: Chan Tai Man, News, Wall, Friends, Notifications(1), Events(1), Check-In, Venues, a search bar, My Account, and Logout. The main heading is "Chan Tai Man 's Profile". On the left is a profile picture of a young man with a "Choose File" button and "No file chosen" text, and an "Upload" button. To the right, there are three sections: "About" with the text "hello~I'm Chan Tai Man." and an "Edit" button; "Basic Info" with fields for Title (Mr.), Name (Chan Tai Man), Gender (Male), Birthday (28-10-2000), and Address (abc), each with an "Edit" button; and "Contact Info" with fields for Email (tmchan@stu.ust.hk), Email Backup (tmchan@gmail.com), Home Phone (23423423), Office Phone, and Mobile Phone (64534131), each with an "Edit" button. A green message "*Saved." is displayed at the bottom left.

Section	Field	Value	Action
About	About	hello~I'm Chan Tai Man.	Edit
Basic Info	Title	Mr.	Edit
	Name	Chan Tai Man	
	Gender	Male	
	Birthday	28-10-2000	
	Address	abc	
Contact Info	Email	tmchan@stu.ust.hk	Edit
	Email Backup	tmchan@gmail.com	
	Home Phone	23423423	
	Office Phone		
	Mobile Phone	64534131	

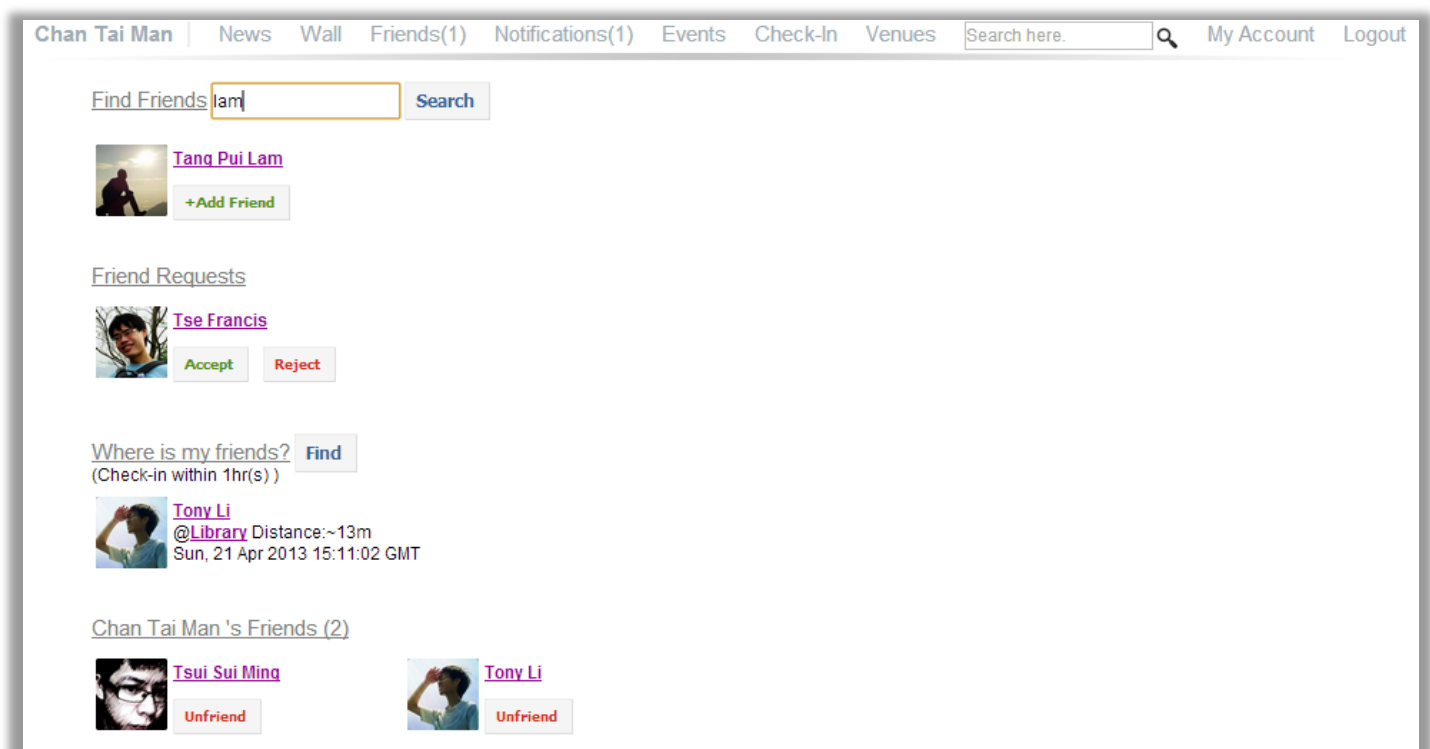
To change the information, click the [Edit] button. The related fields are changed to the input fields. After finished the modification, click [Save] button to keep the changes. A “*Saved” message will be shown at the bottom if the process is successful.

This screenshot shows the same profile page as the previous one, but with the information fields converted into input forms for editing. Each section now has a "Save" button. The "About" section has a text input field containing "hello~I'm Chan Tai Man." and a "Save" button. The "Basic Info" section has dropdown menus for Title (Mr.), Gender (Male), and Birthday (28 Oct 2000), and text input fields for Name (First Name: Chan, Last Name: Tai Man) and Address (abc), each with a "Save" button. The "Contact Info" section has text input fields for Email (tmchan@stu.ust.hk), Email Backup (tmchan@gmail.com), Home Phone (23423423), Office Phone, and Mobile Phone, each with a "Save" button. The "Choose File" button and "No file chosen" text are still present, along with the "Upload" button. The green message "*Saved." is no longer visible.

Section	Field	Value	Action
About	About	hello~I'm Chan Tai Man.	Save
Basic Info	Title	Mr.	Save
	Name	First Name: Chan Last Name: Tai Man	
	Gender	Male	
	Birthday	28 Oct 2000	
	Address	abc	
Contact Info	Email	tmchan@stu.ust.hk	Save
	Email Backup	tmchan@gmail.com	
	Home Phone	23423423	
	Office Phone		
	Mobile Phone		

Friend Page

In the friend page, user will be able to search other users, view friend requests, find the closest friends and check the friend list.



For searching users in the system, the user can enter a keyword to search. The function will return a list of users who have a matched keyword in their information including first name, last name and email address. If the friend is not in the user's friend list, the user can click [+Add Friend] button to send a friend request to ask for build a friendship.

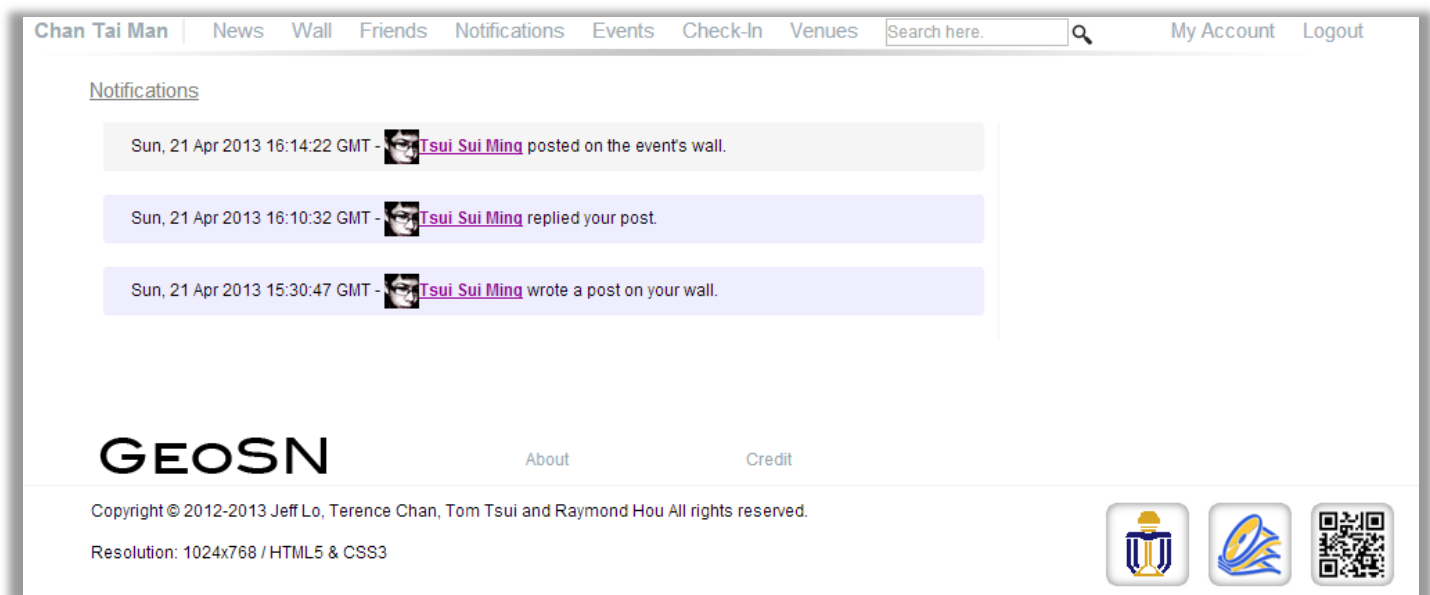
After the friend request is sent, the receiver can see it in his/her friend page and a notification for it is also created. The receiver can decide whether accept or reject the request.

The closest friend is the friend who is physically the closest to the user so that the user can meet their friend easily in the real world. The venue of friend is according to his/her check-in location within a certain period of time.

The last part shows a list of friends who accept the friend request and become a friend of the user. User can browser their page through the link.

Notifications Page

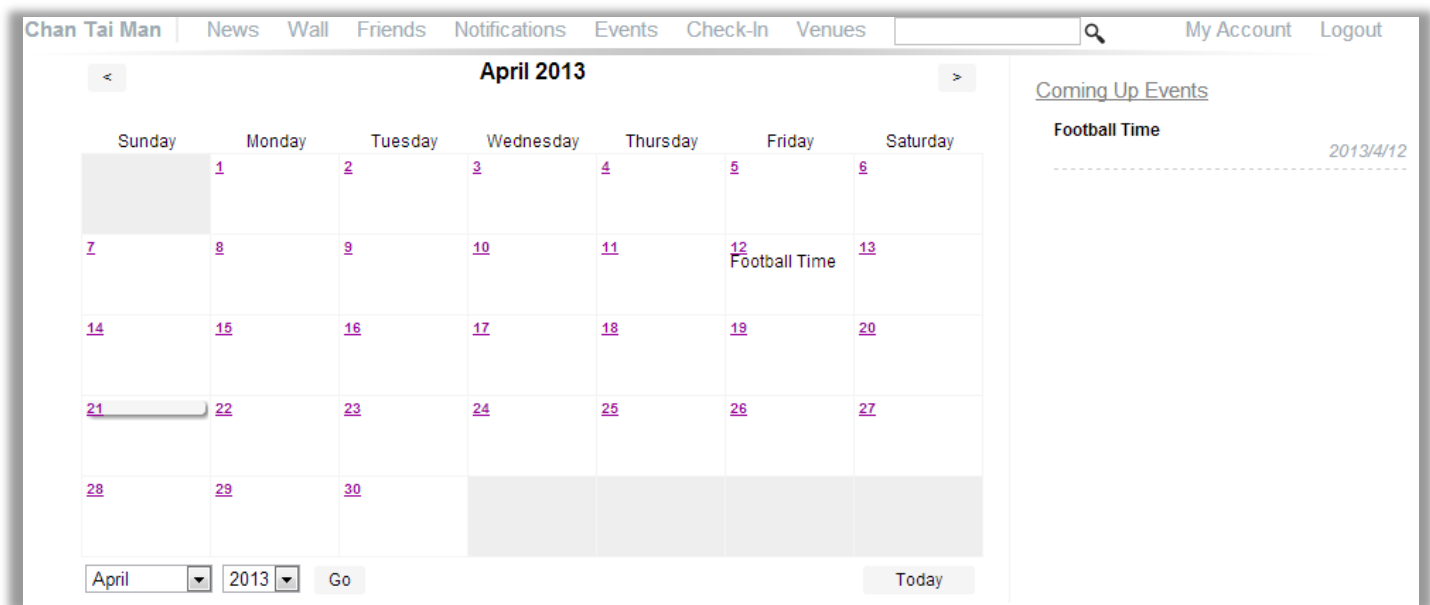
Any activities which related to the user are displayed in notifications page including a post or reply from a friend and a post from the event. For user-friendly, the number of unread messages is indicated in the menu.



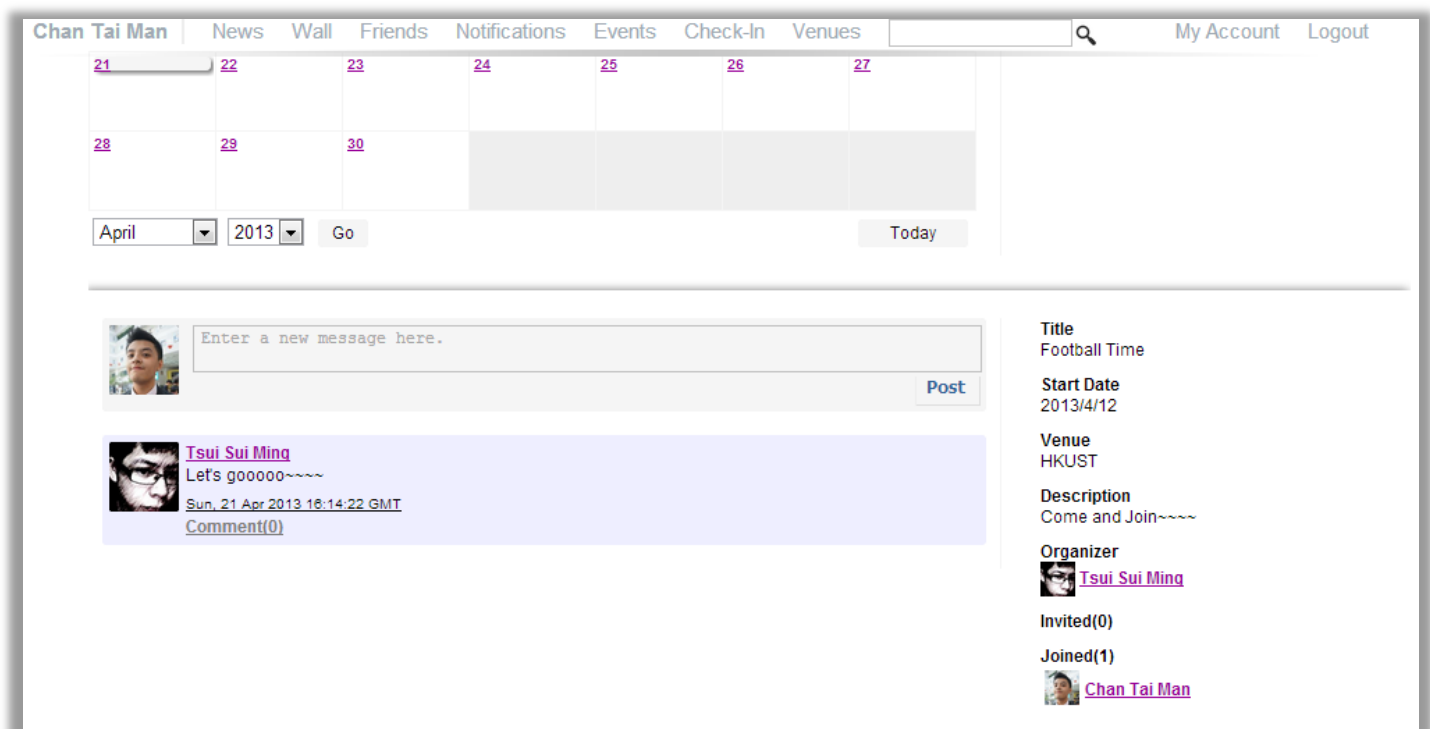
Each notification consists of posting time, related name of user and the type of notification. Using an underlying hyperlink in the notification, user can directly access the post by clicking the notification.

Event Page

The system can help user to invite their friend to join an event they held.



To create an event, the user can pick a date from the calendar. After fill in all required information, the event will be added to the list on the right. The event also contains a wall for posting messages. The information about the event is displayed and only the organizer can edit the details.



Check-In

User can check-in the venue in HKUST to meet their friends easily.

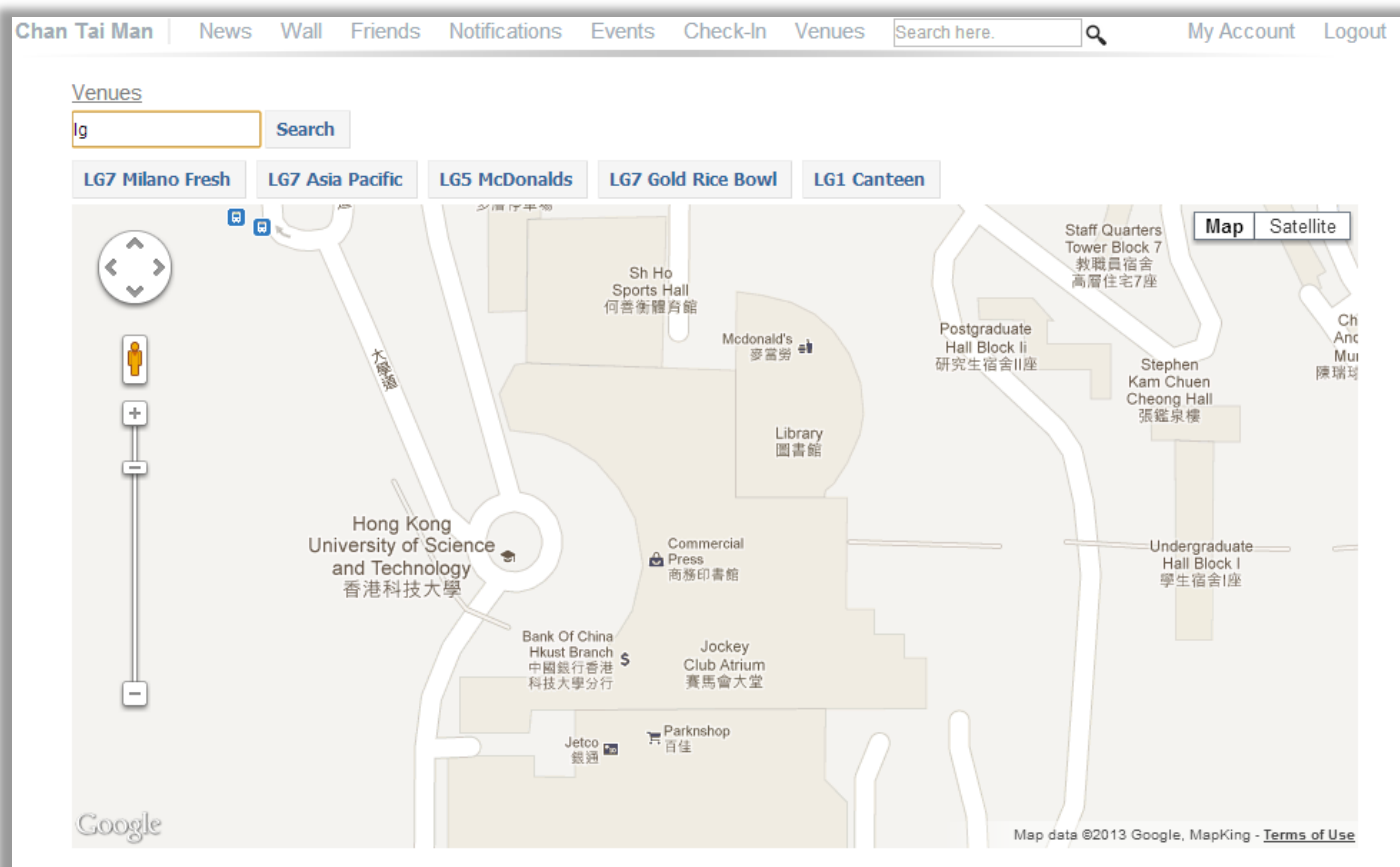
The screenshot displays the 'Check-In' section of a web application. At the top, a navigation bar includes links for 'Chan Tai Man', 'News', 'Wall', 'Friends', 'Notifications', 'Events', 'Check-In', 'Venues', a search bar, 'My Account', and 'Logout'. The 'Check-In' page features a 'Find Me' button, followed by the user's current coordinates: Longitude: 114.2158948 and Latitude: 22.3179785. Below this, a prompt asks the user to 'Select your closet venue to check-in'. A section titled 'Closest venues:' lists several options in buttons: 'LG5 McDonalds', 'LG7 Asia Pacific', 'LC-06', 'Chinese Restaurant', 'LG7 Gold Rice Bowl', 'OutvenueIde LT-A', 'LG1 Canteen', and 'LC-05'. There is also a search bar for 'Other Venues' with the word 'library' entered and a 'Search' button. A 'Library' button is located below the search bar. On the right side, a map shows the HKUST campus with a green pin indicating the user's location and several red pins for other venues. A callout box points to the 'LG7 Asia Pacific' venue. At the bottom left, a 'My Last 10 Check-In History' section lists previous check-ins with timestamps and venue names, such as '@ LG7 Gold Rice Bowl' and '@ LG5 McDonalds'. The map on the right is a Google Map with labels for various campus locations like 'Sh Ho Si + s Hall', 'McDonald's', 'Library', 'Commercial Press', 'Postgraduate Hall Block li', and 'Tower Block'.

To check in the venue, user needs to get his/her current position by clicking [Find Me] button. Using the Wi-Fi geo-location, the latitude and longitude can be got and be shown in the page. The current position is indicated on the map with a green pointer. Besides, the other places next to it are also pointed out on the map with a red pointer. User can click the pointer to view the name of venue.

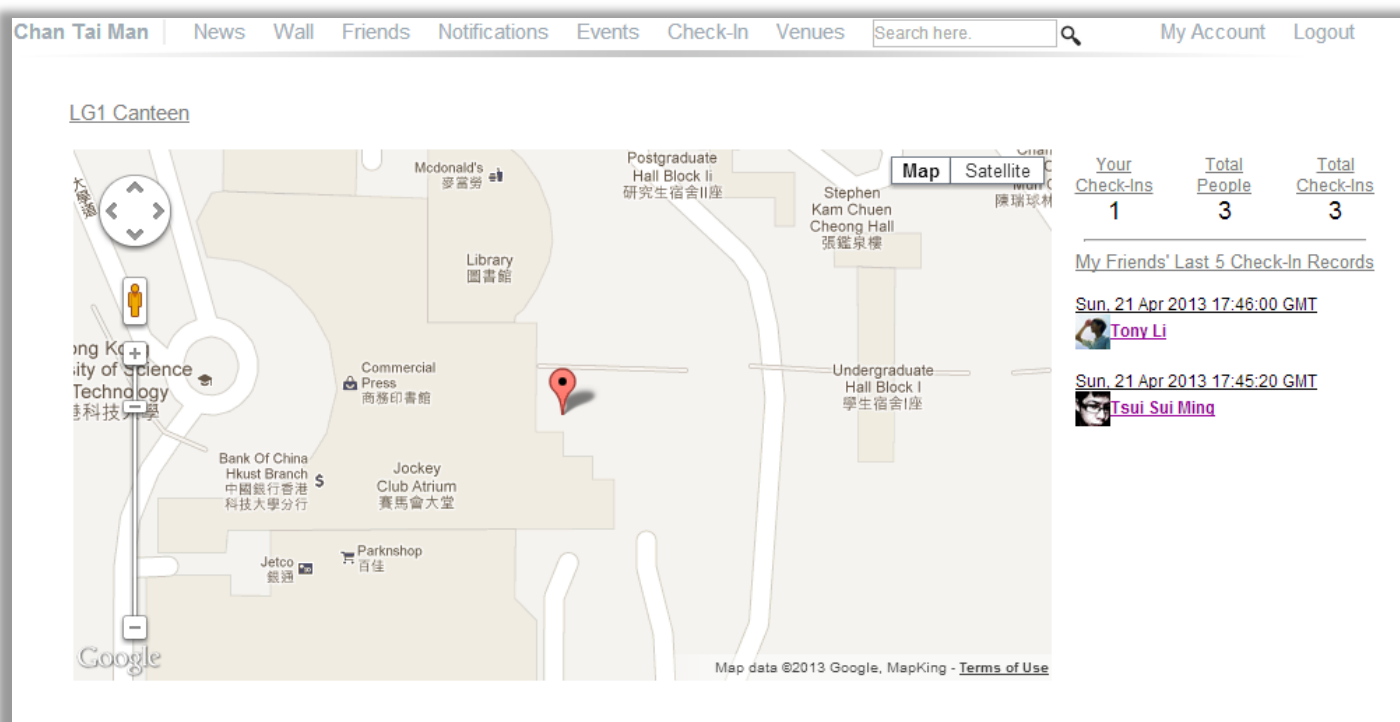
As location is not precise enough and is not able to acquire the altitude, the system will automatically generate a list of closest venues for user to choose. However, if the user cannot find the venue inside the list, they can also enter the name of venue to search manually, and then select the correct venue for check-in.

Venue Page

User can view the map of HKUST using Google Map and search the venue through entering the name in the textbox.



The name of venues which matched to the search keyword will be shown. If the user clicks the label, a page about the detailed information of the place will be loaded. The actual location of the venue is pointed out on the map. Moreover, the check-in statistics of it are aligned next to the map.



Search Page

User can search for users, events and venues through the textbox in the menu.

Tsui Sui Ming | News | Wall | Friends | Notifications(4) | Events | Check-In | Venues | 🔍 | My Account | Logout

Search for "man"

Friends

 [Chan Tai Man](#)

Events

 [Tsui Sui Ming](#) | Footballman Time | 2013/4/12

Venues

[Interdisciplinary Program Office, Office of the Dual Degree Program in Technology and Management \(Room 5569\)](#)

The corresponding results are classified in three different types which are friends, events and venues. Each record has a hyperlink to the related page for getting more information.

Account Page

User can change their password in this page.

Chan Tai Man | News | Wall | Friends | Notifications | Events | Check-In | Venues | 🔍 | My Account | Logout

[Edit Account](#)

Change Password

Current Password

New Password


Confirm Password

[Edit](#)

To change the password, user needs to enter the current password first for identification. Then, user enter the new password twice to ensure the correctness.


Browsing Friends

User can browse their friend's profile, wall and list of friend. When the user accesses their friend's page through the hyperlink, the menu will be changed. The left side is the friend's name, wall and friends while the right side is the short version of original menu of current user. If the user wants to quit the browse and back to his/her wall, just need to click his/her name again.

[Tsui Sui Ming](#) | [Wall](#) | [Friends](#) | 

[Chan Tai Man](#) | [Notifications](#) | [Logout](#)

[Tsui Sui Ming's Profile](#)



About
I'm Terence.

Basic Info

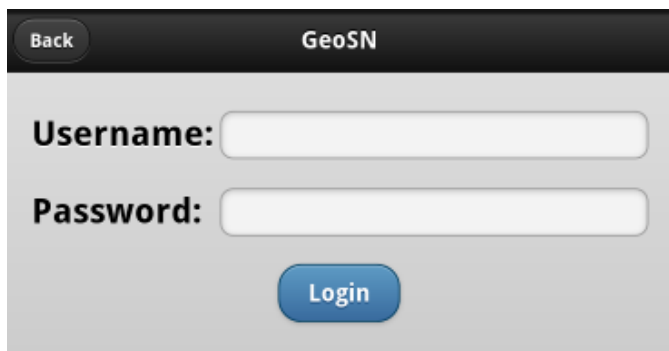
Title	Mr.
Name	Tsui Sui Ming
Gender	Male
Birthday	19-5-1989
Address	

Contact Info

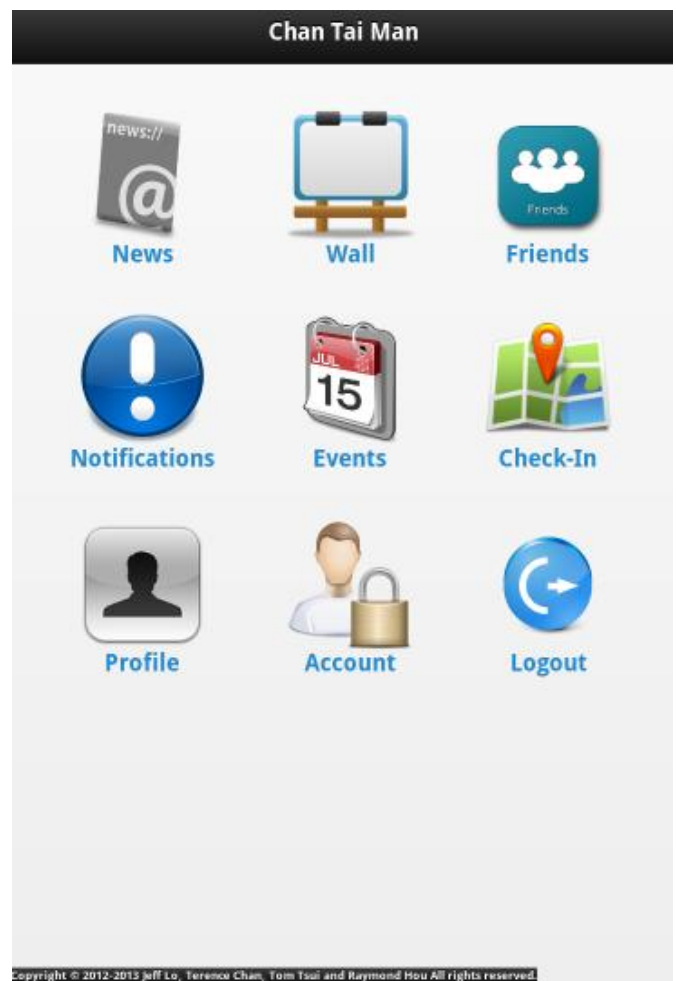
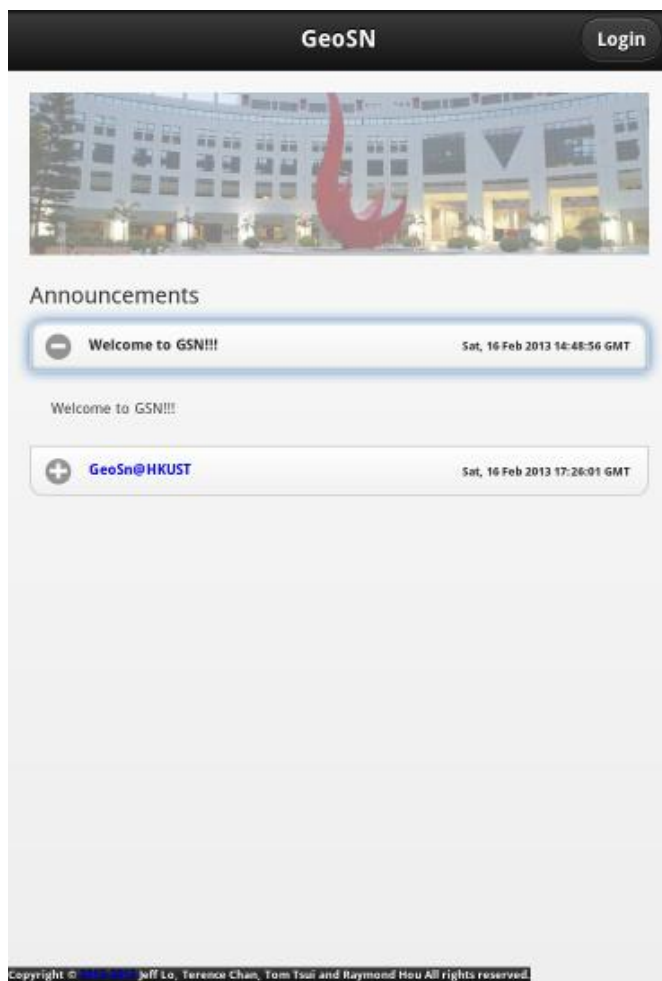
Email	smtsui@stu.ust.hk
Email Backup	smtsui@gmail.com
Home Phone	23456789
Office Phone	
Mobile Phone	

Mobile Pages

The system can be accessed through the mobile device. Different layout is displayed in the screen but can provide the same function as the desktop version.

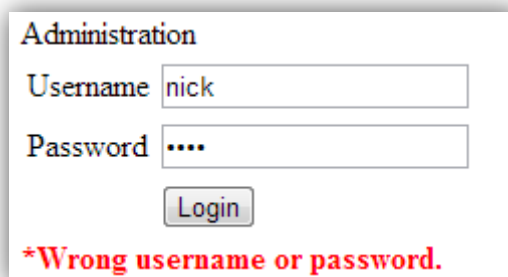


The image shows a mobile application interface for GeoSN. At the top, there is a dark header bar with a 'Back' button on the left and the text 'GeoSN' in the center. Below the header, the main area is light gray. It features a 'Username:' label followed by a text input field, and a 'Password:' label followed by another text input field. At the bottom of this section, there is a blue rounded rectangular button labeled 'Login'.



Admin Page

Administrator can login to the backend page to manage the system.



The screenshot shows a login form titled "Administration". It has two input fields: "Username" with the value "nick" and "Password" with masked characters "....". Below the fields is a "Login" button. At the bottom, a red error message reads: "*Wrong username or password."

Administrator need to enter the correct username and password to login the system. If the username or password is wrong, an error message will be shown below.

- [HOME](#)
- [Logout](#)
- [Announcements](#)
- [Venues](#)
- [Parameters](#)
- [User Admin](#)

After login the system, administrator is able to manage the system settings including the announcement, venue, parameter and users.

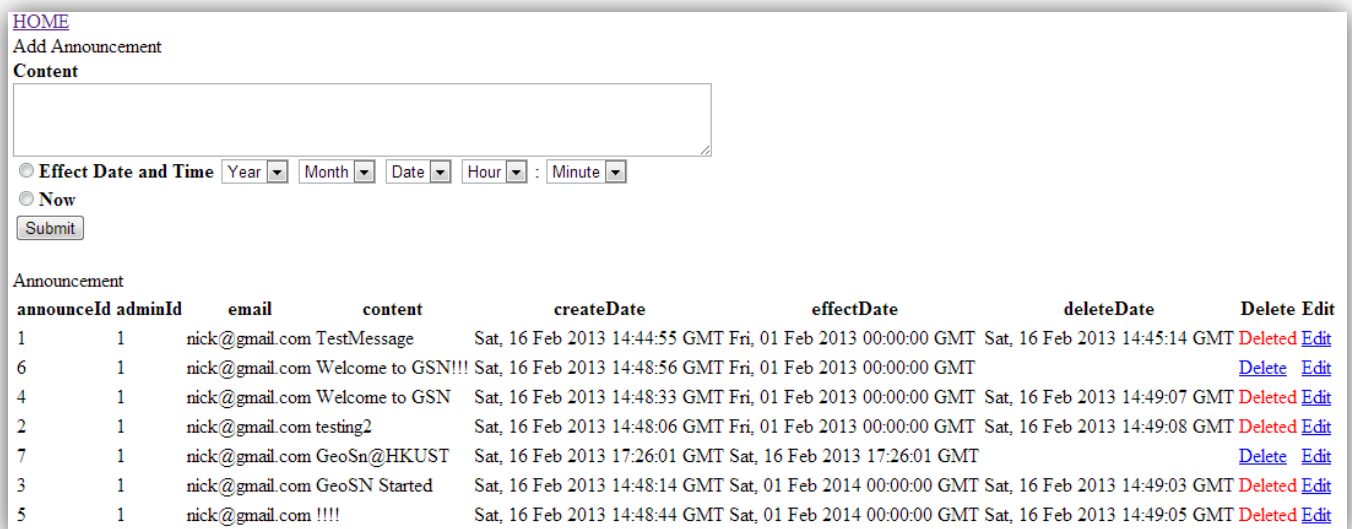
Announcements: Manage the messages shown in the index page.

Venues: Manage the venues for check-in.

Parameters: Manage the duration for finding the friend who check-in recently, and the radius parameter for searching the nearest friend.

User Admin: Manage the user accounts.

Announcements




The screenshot shows the "Announcements" management page. At the top, there's a "HOME" link and a "Add Announcement" section with a "Content" text area. Below the text area are radio buttons for "Effect Date and Time" (selected) and "Now". The "Effect Date and Time" section includes dropdowns for Year, Month, Date, Hour, and Minute, followed by a "Submit" button. Below this is a table of existing announcements.

announceId	adminId	email	content	createDate	effectDate	deleteDate	Delete	Edit
1	1	nick@gmail.com	TestMessage	Sat, 16 Feb 2013 14:44:55 GMT	Fri, 01 Feb 2013 00:00:00 GMT	Sat, 16 Feb 2013 14:45:14 GMT	Deleted	Edit
6	1	nick@gmail.com	Welcome to GSN!!!	Sat, 16 Feb 2013 14:48:56 GMT	Fri, 01 Feb 2013 00:00:00 GMT		Delete	Edit
4	1	nick@gmail.com	Welcome to GSN	Sat, 16 Feb 2013 14:48:33 GMT	Fri, 01 Feb 2013 00:00:00 GMT	Sat, 16 Feb 2013 14:49:07 GMT	Deleted	Edit
2	1	nick@gmail.com	testing2	Sat, 16 Feb 2013 14:48:06 GMT	Fri, 01 Feb 2013 00:00:00 GMT	Sat, 16 Feb 2013 14:49:08 GMT	Deleted	Edit
7	1	nick@gmail.com	GeoSn@HKUST	Sat, 16 Feb 2013 17:26:01 GMT	Sat, 16 Feb 2013 17:26:01 GMT		Delete	Edit
3	1	nick@gmail.com	GeoSN Started	Sat, 16 Feb 2013 14:48:14 GMT	Sat, 01 Feb 2014 00:00:00 GMT	Sat, 16 Feb 2013 14:49:03 GMT	Deleted	Edit
5	1	nick@gmail.com	!!!!	Sat, 16 Feb 2013 14:48:44 GMT	Sat, 01 Feb 2014 00:00:00 GMT	Sat, 16 Feb 2013 14:49:05 GMT	Deleted	Edit

To create an announcement, the administrator has to enter the content and the effect date when the announcement become active. The message will be displayed at the home page when it is active.

Venues

Insert New Venue
Venue Name:
Location:
Longitude:
Latitude:



Map data ©2013 Google, MapKing - Terms of Use

Venues

venueId	venueName	location (longitude)	location (latitude)	Edit Delete
75	Atrium	114.26371898879	22.337555519292	Edit Delete
73	Cafe	114.263439	22.336806	Edit Delete

To create a new venue, the administrator can click the “Get Current Position” button to locate the current position then fill in the venue name to create. Also, the records can be modified or deleted by administrator.

Parameters

Manage the duration for finding the friend who check-in recently, and the radius parameter for searching the nearest friend.

[HOME](#)

Check-In Time Limit For Finding Friends:
 Hour(s)


Check-In Radius For Finding Friends:
 Meter(m)

User Admin

Administrator can lock the user account to inhibit their login.

[HOME](#)

[Search](#)

Username	Name	Email	Lock
 tmchan	Chan Tai Man	tmchan@stu.ust.hk	<input type="checkbox"/>

3. Implementation

3.1 Implementation of DBConnection and CollectionBase Class

fyp.gsn.Connection.DBConnection.java

```
package fyp.gsn.Connection;

public interface DBConnection<T> {

    public boolean connect();

    public T getDB();

    public T getDB(String uname, String pswd);

    public T getDB(String dbName);

    public T getDB(String dbName, String uname, String pswd);

    // Get Auto-Increment Id for the given name
    public long getIncId(String idName);

    // Get error message
    public String getErrorStr();

    public boolean disconnect();

}
```

fyp.gsn.Collection.CollectionBase.java

```
package fyp.gsn.Collection;

import java.util.ArrayList;
import java.util.Date;
import java.util.Set;

import com.mongodb.DBObject;

import fyp.gsn.Connection.DBConnection;

/*
T: ConnectionType
U: DataStructure
V: ListStructure
W: QueryResult
X: UpdateResult
*/
public interface CollectionBase<T,U,V,W,X> {

    public T getDB();

    public void setData(U o);

    public U getData();

    public DBConnection<T> getConnection();

    public String[] getRequiredAttrs();

    public U getRequiredAttrObj();

    public boolean attrExists(String attr);

    public Set<String> getAttributes();

}
```

```

public Object getVal(String attr);

public Long getLong(String attr);

public Double getDouble(String attr);

public Date getDate(String attr);

public String getString(String attr);

public V getList(String attr);

public ArrayList<?> getArrayList(String attr);

public <Z> Z[] getArray(String attr, Class<? extends Z[]> c);

public void setVal(String attr, Object val);

public void setVal(DBObject o);

public void removeField(String attr);

public boolean checkRequiredAttrs();

public W query();

public W query(DBObject searchCriteria);

public W query(DBObject searchCriteria, DBObject fields);

public X insert();

public X update();

public X delete();

}

```

3.2 Implementation of Web Service

Web.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns="http://java.sun.com/xml/ns/javaee">
  <display-name>GSN_WS</display-name>
  <servlet>
    <servlet-name>Jersey REST Service</servlet-name>
    <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-
class>
    <init-param>
      <param-name>com.sun.jersey.config.property.packages</param-name>
      <param-value>fyp.gsn.ws</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Jersey REST Service</servlet-name>
    <url-pattern>*</url-pattern>
  </servlet-mapping>
</web-app>

```

The following functions are the administrative function which can be invoked by web application
Admin: /admin

```
//return: 0 = Fail, 1 = Succeed, 2 = No lock before  
unlockAccountWS: @Path("/unlockAccount/{aId}/{uId}")  
  
//return: 0 = Fail, 1 = Succeed, 2 = Already locked  
lockAccountWS: @Path("/lockAccount/{aId}/{uId}")  
  
//return: 0 = No record  
getAnnouncementWS: @Path("/getAnnouncement/{aId}"), Optional: skip: int, limit: int  
  
//return: 0 = Fail, 1 = Succeed  
deleteAnnouncementWS: @Path("/deleteAnnouncement/{aId}/{annId}")  
  
//return: 0 = Fail, 1 = Username exists, 2 = Email address exists, 3 = Succeed  
insertAdminWS: @Path("/insert/{username}/{pswd}/{email}")  
  
//return: 0 = Fail, 1 = Succeed, 2 = Email address exists  
updateAdminWS: @Path("/update/{aId}/{pswd}/{email}")  
  
//return: 0 = Fail, 1 = Succeed  
deleteAdminWS: @Path("/delete/{aId}")  
  
getAdminByIdWS: @Path("/getAdminById/{aId}")  
getAdminByUsernameWS: @Path("/getAdminByUsername/{uname}")  
getAdminByEmailWS: @Path("/getAdminByEmail/{email}")  
  
//return: 1 = Username does not exist, 2 = Incorrect password, 3 = Succeed  
checkLoginWS: @Path("/checkLogin/{uname|email}/{pswd}")
```

3.3 Implementation of Front-end

The front-end consists of web application and web pages which are implemented by MVC (Model-View-Controller) approach. In web page, HTML is the view part which display on the browser and the jQuery is the control layer to handle the input/output data between web page and web application. In web application, PHP is the model to send request and receive result from the web service through JSON (JavaScript Object Notation). Following this software architecture pattern, the functions can be easily separated according to related functionality, the code can be reused and easy to maintain. Also, using jQuery can improve the performance by only dealing with the necessary request rather than process the entire page once more which can save a lot of bandwidth and reduce the loads for the server.

The following sample codes are the administrative login function. The web page contains HTML code and javascript. When the user click the login button, it will trigger the btnLogin() function using jQuery to send the username, password and login request to the web application. When web application receives the request, it will use curlGet() function to ask web service to do the authenticate the username and password in the database. After the web service processed the request, it will return the result to the web application. If the password is correct, web application will allow the user to login the system. While the password is wrong, an error message will be displayed and the user is required to re-enter the correct password. In this way, we separated the Model, Control and View.

-----Web Page-----

```
<html>
<head>
<script type="text/javascript" src="../ext/js/plugin/jquery.js"></script>
<script type="text/javascript">
function btnLogin(){
$.post("admin.php?f=login",{ "username":$('#username').val(),"password":$('#password').val() },
    function(data){
        result=data;
        if(result=='fail')
            $('#errMsg').html('*Wrong username or password. ');
        else
            $('#main').html(result);
    },'html'
);
}
</script>
</head>
<body id="main">
Administration
<form action="admin.php?f=login" method="post">
<table>
    <tr><td>Username</td><td><input id="username" name="username" type="text"></td></tr>
    <tr><td>Password</td><td><input id="password" name="password" type="password"></td></tr>
    <tr><td></td><td><input type="button" value="Login" onclick="btnLogin();"></td></tr>
</table>
<span id="errMsg" style="font-weight:bold;color:#FF0000;"></span>
</form>
</body>
</html>
```

-----Web Application-----

```
if($_GET['f']=='login'){
    // Receive the data from web page
    $username=$_POST['username'];
    $password=$_POST['password'];

    // Sending request to the web service and store the result
    $result=curlGet('/admin/checkLogin/'.$username.'/'.$password);

    // Returning the result to the request sent from javascript
    if($result=='3'){ // login success
        $_SESSION['adminUsername']=$username;
        header( 'Location: home.php' );
    }else{ // login fail
        echo 'fail';
    }
}
```


3.4 Using Haversine Formula to find nearby friends

To find nearby friends according to the user's current location, we need to apply the Haversine formula[7] and convert the latitude and longitude into distance in order to search friends in defined range, e.g. 200m. The following is the pseudo code for applying the Haversine formula to find nearby friends:

Haversine formula: (Given coordinate of 2 users in (longitude, latitude), as (lon1, lat1), (lon2, lat2))

$d_{lon} = lon2 - lon1$

$d_{lat} = lat2 - lat1$

$a = \sin^2(d_{lat}/2) + \cos(lat1) * \cos(lat2) * \sin^2(d_{lon}/2)$

$c = 2 * \arcsin(\min(1, \sqrt{a}))$

$d = R * c$

where d is the distance between two users, and R stands for the radius of the Earth (in this project, we take R as 6367 km).

4. Testing

4.1 Testing Methodology

Upon finishing the implementation of the system, we will use different methods to ensure all functionality matches the requirement.

Black-box Testing: We test the functionality of the system such as whether the data is inserted correctly into database. A list of test cases will be designed for testing all functionality of the system. The output result will be compared to the expected result to find any errors.

White-box Testing: We test the integration between the API and the internal structures including the integration between web service and web server.

Usability Testing: We will invite our classmate to test the usability of the system which can justify the quality of the system. Their comments will be collected and analyzed to find the advantage and disadvantage of the system for any possible improvements.

4.2 Test Cases

User:

Test Data	Expected Result	Actual Result
Action: Register Username: tom Password: password First Name: Kam Ming Last Name: Tsui Gender: M Email: tom@gmail.com Date of Birth: 10-10-1991	A new user is added	A new user is added
Action: Register Username: raymond Password: password First Name: Wai Man Last Name: Hou Gender: M	A new user is added	A new user is added

Email: raymond@gmail.com Date of Birth: 5-8-1990		
Action: Register Username: jeff Password: password First Name: Wai Tong Last Name: Lo Gender: M Email: jeff@gmail.com Date of Birth: 7-15-1989	A new user is added	A new user is added
Action: Register Username: william Password: password First Name: Wai Ting Last Name: Chan Gender: M Email: tom@gmail.com Date of Birth: 8-5-1991	Duplicated email, an error message is shown	Duplicated email, an error message is shown
Action: Register Username: tom Password: password First Name: Wai Ting Last Name: Chan Gender: M Email: william@gmail.com Date of Birth: 8-5-1991	Duplicated username, an error message is shown	Duplicated username. , an error message is shown
Action: Register Username: william Password: password First Name: Wai Ting Last Name: Chan Gender: M Email: william@gmail.com Date of Birth: 8-5-1991	A new user is added	A new user is added
Action: Edit profile Username: tom Email backup: tomtsui@gmail.com Date of birth: 9-10-1991	Tom's profile is updated	Tom's profile is updated
Action: Edit profile Username: jeff First name: Wai Tong Remove the profile picture	Jeff's profile is updated and profile picture is removed	Jeff's profile is updated and profile picture is removed
Action: Reset password Username: minimum Email address: william@gmail.com	Username/Email address is not exist or do not match	Username/Email address is not exist or do not match

Admin:

Test Data	Expected Result	Actual Result
Action: Login Username: nicknick Password: password	Login failure	Login failure
Action: Login Username: nick Password: password123	Login failure	Login failure
Action: Login Username: nick Password: password	Login succeed	Login succeed
Action: Login Username: nick lock account: tom	Tom cannot login	Tom cannot login
Action: Login Username: nick unlock account: tom	Tom can login	Tom can login

Announcement:

Test Data	Expected Result	Actual Result
Action: Add announcement Admin Account: nick Content: Geo-Social Networking Service Started Effect Date: now	A new announcement is added	A new announcement is added
Action: Add announcement Admin Account: nick Content: If you have any question, please contact Nick at nick@gmail.com Effect Date: 1 day after	A new announcement is added (but not display yet)	A new announcement is added (but not display yet)
Action: Add announcement Admin Account: andy Content: Andy is now an new admin. Email address: andy@gmail.com Effect Date: now	A new announcement is added	A new announcement is added
Action: Add announcement Admin Account: andy Content: Andy's Test Announcement Effect Date: now	A new announcement is added	A new announcement is added
Action: Delete announcement Nick deleted Andy's Test Announcement (above)	The announcement will not be displayed	The announcement will not be displayed

FriendRequest:

Test Data	Expected Result	Actual Result
Action: Send friend request From: tom To: raymond	Raymond has a friend request	Raymond has a friend request
Action: Send friend request From: tom To: jeff	Jeff has a friend request	Jeff has a friend request
Action: Send friend request From: tom To: william	William has a friend request	William has a friend request
Action: Send friend request From: tom To: william	Friend request sent before	Ok, the button cannot click again
Action: Send friend request From: jeff To: raymond	Raymond has a friend request (total 2)	Raymond has a friend request (total 2)
Action: Send friend request From: jeff To: william	William has a friend request (total 2)	William has a friend request (total 2)
Action: Reply friend request User: Raymond Accept friend (both tom and jeff)	Added friends Tom and Jeff got notification	Added friends Tom and Jeff got notification
Action: Reply friend request User: Jeff accept friend (tom)	Added friend Tom got notification (2 in total)	Added friend Tom got notification (2 in total)
Action: Reply friend request User: William Accept friend (tom)	Added friend Tom got notification (3 in total)	Added friend Tom got notification (3 in total)
Action: Reply friend request User: William Reject friend (jeff)	The request is replied	The request is replied
Action: Send friend request From: tom To: william	Users are Friends already, cannot send request	Ok, add friend button disappeared
Action: Unfriend User: tom Unfriend: william	william is removed from tom's friend list	william is removed from tom's friend list

Checkin:

Test Data	Expected Result	Actual Result
Action: Check-in User: tom Venue: [anyone]	A new check in is added	A new check in is added
Action: Check-in User: raymond Venue: same as tom	A new check in is added	A new check in is added
Action: Check-in User: jeff Venue: venue next to tom	A new check in is added	A new check in is added

Action: Get friends@HKUST User: tom	Raymond & Jeff are found William is not found	Raymond & Jeff are found William is not found
Action: Find nearest friends User: tom	Raymond is found	Raymond is found

Event:

Test Data	Expected Result	Actual Result
Action: Create new event User Account: jeff Title: LG3 Gathering Duration: today Venue: LG3 Open Area Event Content: It's time to gathering. Let's fun together	A new event in is added	A new event in is added
Action: Create new event User Account: tom Title: FYP Group Conference Duration: tomorrow Venue: Group Study Rooms (LC) – LC-05 Event Content: Discussion on FYP Layout	A new event in is added	A new event in is added
Action: Search event/venue Keyword: FYP	Event: FYP Group Conference is found	Event: FYP Group Conference is found

EventRequest:

Test Data	Expected Result	Actual Result
Action: Send event request Event: FYP Group Conference From: tom To: Raymond	Raymond received an event request Raymond's name added to the Invited list	Raymond received an event request Raymond's name added to the Invited list
Action: Send event request Event: FYP Group Conference From: tom To: jeff	Jeff received an event request	Jeff received an event request
Action: Send event request Event: FYP Group Conference From: tom To: jeff	Jeff is invited before	Jeff is invited before
Action: Reply event request User: jeff Accept event request	Jeff joined the event (tom got notification) Jeff's name removed from the Invited list Jeff's name added to the Joined list	Jeff joined the event (tom got notification) Jeff's name removed from the Invited list Jeff's name added to the Joined list
Action: Reply event request User: raymond Reject event request	Raymond's name removed from the Invited list	Raymond's name removed from the Invited list
Action: Send event request Event: FYP Group Conference From: tom	Raymond received an event request Raymond's name added to the	Raymond received an event request Raymond's name added to the

To: Raymond	Invited list	Invited list
Action: Send event request Event: FYP Group Conference From: tom To: jeff	Jeff joined the event already, cannot send the event request	Jeff joined the event already, cannot select Jeff from the user list
Action: Reply event request User: raymond Accept event request	Raymond joined the event (tom got notification) Raymond's name removed from the Invited list Raymond's name added to the Joined list	Raymond joined the event (tom got notification) Raymond's name removed from the Invited list Raymond's name added to the Joined list

Post:

Test Data	Expected Result	Actual Result
Action: Create new post Author: tom Wall: tom's wall Comment: 1st Post on my wall	A new post is added	A new post is added
Action: Create new post Author: Raymond Wall: tom's wall Comment: I am the first one to say Hello on your wall	A new post is added (tom got notification)	A new post is added (tom got notification)
Action: Create new post Author: tom Wall: event "FYP Group Conference" wall Comment: Please come on time	A new post is added (jeff and raymond got notification)	A new post is added (jeff and raymond got notification)
Action: Reply post Author: jeff post: tom's post on event "FYP Group Conference" wall Reply: No problem	A new reply is added (tom got notification)	A new reply is added (tom got notification)
Action: Reply post Author: raymond post: tom's post on event "FYP Group Conference" wall Reply: Can I come late?	A new reply is added (tom and jeff got notification)	A new reply is added (tom and jeff got notification)
Action: Reply post Author: tom post: tom's post on event "FYP Group Conference" wall Reply: Sure NO	A new reply is added (raymond and jeff got notification)	A new reply is added (raymond and jeff got notification)
Action: Delete post Wall: tom's wall Delete post: 1st Post on my wall	The post is deleted	The post is deleted
Action: Delete reply Post: tom's post on event "FYP Group Conference" wall Delete comment: Sure NO	The reply is deleted	The reply is deleted

5. Evaluation

In the evaluation phase, we will analysis the system and justify our work. The evaluation is based on the following five aspects.

- **Reliability:** Correctness of functionality, exception handling, smoothness of system, etc.
- **Accuracy:** The accuracy of check-in location.
- **User Interface:** Whether the User Interface design is user-friendly.
- **Capacity:** The maximum number of users the system can handle simultaneously.
- **Reusability:** Modularity of the API. Application of the open-close principle.
- **Readability:** Whether the code is easy to understand and appropriate comments are added.

6. Discussion

The objective of our project is to implement a Geo-social network, and to facilitate communication between users. The system allows users to make friends, share their personal status, finding friends checked in nearby and inviting friends to their events.

Upon conducting system testing according to the use case as well as test cases in section 4.2, most of the functionalities are delivered as designed at the planning phase, while little modifications were made in some of the process flow without affecting the actual functionality. The user interface is also easily understandable for users with layouts for both PC and mobile devices.

In addition, there is proper integration between the web application, web services and system API so that the model, view and control are separated. With the web service, we avoided the web application to directly connecting to the database server, so as to reduce the visibility of the database. However, due to technical limitations, we are not able to test for the security of the servers' accessibility towards network attacks.

Throughout the project, there are some limitations as listed below:

WiFi Positioning: Due to the inevitable error and the high density of rooms in UST, the coordinates tend to be inaccurate. In some venues/locations, the WiFi signal is too weak to maintain a stable connection.

User Interface Consistency: Considering the compatibility and maintenance issue, a web application instead of a native application is being developed. Although, it would be able to support any devices that are able to access the webpages, but it is not possible to test the consistency on layout over all the browsers and all the versions. Therefore, only certain popular browsers such as Mozilla Firefox and Google Chrome are being used for testing.

Limitations in MongoDB: Though MongoDB is easily extensible and efficient, but it is not able to perform sort operations in the sub-documents such that it is being done externally and programmatically in the system API. Also, MongoDB doesn't support join operation implicitly, duplicate data has to be store over different collections, which increased the overhead for update operations to ensure data consistency.

7. Conclusion

In this project, we implemented a Geo-Social Network for students to make friends and to interact with each other. Throughout the development process, we have encountered various obstacles with the technologies being used. For example, all of the team members didn't have prior experience in implementing web service, the DBMS we use wasn't a relational DBMS having a schema-less nature. The complexity of the system architecture also required us to think more thoroughly for the integration and interface before implementing the system. The project gave us an opportunity to be in touch with different types of technology and enhanced our decision-making and analytical skills.

Upon finishing the project, we do think there are still much room for improvement and expansion to enrich the system's functionality and attractiveness:

Specific Activities at Venue: When a user check-in at particular venues, the system may trigger some events such as games, seasonal activities (Christmas, Easter etc.) or cooperate with some shops/restaurant to provide discount when they check-in, say, certain consecutive days.

User Interface on Mobile Layout: Since the speed and processor on mobile devices are less powerful than PCs, the amount of information displayed are more limited, and the data to be transmitted should be minimized to maximize the speed and efficiency to optimize response time for portable devices. So a separated data interface for mobile could possibly be developed to improve the performance.

Security on Client Side: Currently, most security and data validation are being done in the server side. It may consider adding more checking in client side using javascript to provide more protection and reduce the workload of the server. As a result, more errors can be avoided by double checking the data in both client side and server side.

Posts Visibility: Sometimes, the user may want to add information on the post for a particular group of users. If the system can provide the mechanism to hidden some of the contents in the post to specified users, it will be more convenient for users to control some sensitive information being viewed by others.

8. References

- [1] Foursquare. (2011). About foursquare. [Online]. Available: <https://foursquare.com/about/>
- [2] Google. (2012). Google+. [Online]. Available: <https://plus.google.com/>
- [3] MongoDB. (2012). Geospatial. [Online]. Available: <http://www.mongodb.org/display/DOCS/Geospatial+Indexing>
- [4] Wikipedia. (2012). Geo-social networking. [Online]. Available: http://en.wikipedia.org/wiki/Geosocial_networking
- [5] Wikipedia. (2013). Representational State Transfer (REST). [Online]. Available: http://en.wikipedia.org/wiki/Representational_state_transfer#RESTful_web_services
- [6] PETE FREITAG. (2013). REST vs SOAP Web Services. [Online]. Available: <http://www.petefreitag.com/item/431.cfm>
- [7] Wikipedia. (2013). Haversine Formula. [Online]. Available: http://en.wikipedia.org/wiki/Haversine_formula
- [8] Movable Type Ltd (2013). Great circle distance between 2 points. [Online]. Available: <http://www.movable-type.co.uk/scripts/gis-faq-5.1.html>

9. Appendix A: Minutes

9.1 Minutes of the 1st Project Meeting

Date: September 13, 2012

Time: 1:30 PM

Place: Room 3555

Present: Prof. Dimitris PAPADIAS, Nikos ARMENATZOGLOU(Nick), Tsui Kam Ming(Tom), Lo Wai Tong (Jeff), Chan Pak Lai(Terence), Hou Wai Man(Raymond)

Absent: None

Recorder: Chan Pak Lai

1. Approval of minutes

This was the first formal group meeting, so there were no minutes to approve.

2. Report on progress

- 2.1. All team members have read the project description.
- 2.2. All team members have done a research on related services using Google.
- 2.3. Terence has read some books for programming in JAVA.

3. Discussion items

- 3.1. Tom asked if there are any limitations.
 - Basically there is no limitation.
- 3.2. Nick explained the structure and required components to us.
 - There should be 2 servers: Database server and Web server.
 - Our service should support multi-platform (Web, Android, and iOS).

4. Goals for the coming week

- 4.1. All group members will think about the structure and have clear idea on how it works.
- 4.2. All group members will try to understand what web service is.
- 4.3. All group members will try to understand what MongoDB is.

5. Meeting adjournment and next meeting

The meeting was adjourned at 2:30 PM.

The date and time of the next meeting will be set later by e-mail.

9.2 Minutes of the 2nd Project Meeting

Date: September 17, 2012

Time: 2:30 PM

Place: Group Study Room (LC) – LC-04

Present: Nikos ARMENATZOGLOU(Nick), Tsui Kam Ming(Tom), Lo Wai Tong (Jeff), Chan Pak Lai(Terence),
Hou Wai Man(Raymond)

Absent: None

Recorder: Chan Pak Lai

1. Approval of minutes

The minute of the last meeting was approved without amendment.

2. Report on progress

- 2.1. All team members have a clear idea about the structure of the project.
- 2.2. All team members have an idea of what web service is.
- 2.3. All team members have tried the online shell of MongoDB.

3. Discussion items

- 3.1. We discussed the detail structure of the project.
 - The flow diagram.
- 3.2. We divided our project into 2 parts: Front-end and Back-end.
 - We further separated our team into 2 sub teams.
 - Raymond and Terence will focus on the front-end part.
 - Tom and Jeff will focus on the back-end part.

4. Goals for the coming week

- 4.1. All group members should clarify the application flow.
- 4.2. Tom and Jeff will finish installing MongoDB on their notebook.
- 4.3. Raymond and Terence will think about the User Interface design.

5. Meeting adjournment and next meeting

The meeting was adjourned at 4:00 PM.

The date and time of the next meeting will be set later by e-mail.

9.3 Minutes of the 3rd Project Meeting

Date: September 20, 2012

Time: 2:30 PM

Place: Room 3555

Present: Nikos ARMENATZOGLOU(Nick), Tsui Kam Ming(Tom), Lo Wai Tong (Jeff), Chan Pak Lai(Terence),
Hou Wai Man(Raymond)

Absent: None

Recorder: Chan Pak Lai

1. Approval of minutes

The minute of the last meeting was approved without amendment.

2. Report on progress

- 2.1. All team members have clarified the application flow.
- 2.2. Tom and Jeff installed MongoDB on their notebook.
- 2.3. Raymond and Terence have drafted the index page.

3. Discussion items

- 3.1. We discussed the set-up of MongoDB.
 - We discussed the problem when starting the MongoDB service.
 - We tried to use MongoDB to create some data.
 - Nick suggested using JMongoBrowser (GUI) to manipulate the DB.
- 3.2. We discussed the design of our web page.
 - We discussed what element should be placed on the index.
 - We discussed how to place the elements.

4. Goals for the coming week

- 4.1. Tom and Jeff should be familiar with Ubuntu.
- 4.2. Tom and Jeff should be familiar with MongoDB.
- 4.3. Tom and Jeff should be familiar with JMongoBrowser.
- 4.4. Tom and Jeff should check-out Foursquare.
- 4.5. Tom and Jeff will design the collections and documents.
- 4.6. Tom and Jeff will create a test dataset.
- 4.7. Tom and Jeff will collect the coordinates of venues.
- 4.8. Raymond and Terence should be familiar with PHP, HTML and Photoshop
- 4.9. Raymond and Terence design the functionality of the system.

5. Meeting adjournment and next meeting

The meeting was adjourned at 4:00 PM.

The date and time of the next meeting will be set later by e-mail.

9.4 Minutes of the 4th Project Meeting

Date: October 3, 2012

Time: 4:00 PM

Place: Room 3555

Present: Nikos ARMENATZOGLOU(Nick), Tsui Kam Ming(Tom), Lo Wai Tong (Jeff), Chan Pak Lai(Terence),
Hou Wai Man(Raymond)

Absent: None

Recorder: Chan Pak Lai

1. Approval of minutes

The minute of the last meeting was approved without amendment.

2. Report on progress

- 2.1. Tom and Jeff have created the collections and documents.
- 2.2. Tom and Jeff have created a test dataset.
- 2.3. We have collected the coordinates of about 80 venues.
- 2.4. Raymond and Terence designed the functionality of the system.

3. Discussion items

- 3.1. We discussed the functionality of our project.
 - We revised the functions we support.

4. Goals for the coming week

- 4.1. Functionality will be defined with more details.
- 4.2. Raymond and Terence will sketch the web page.
- 4.3. Tom and Jeff will design data representation and API.

5. Meeting adjournment and next meeting

The meeting was adjourned at 5:30 PM.

The date and time of the next meeting will be set later by e-mail.

9.5 Minutes of the 5th Project Meeting

Date: January 8, 2013

Time: 2:00 PM

Place: LG3

Present: Tsui Kam Ming(Tom), Lo Wai Tong (Jeff), Hou Wai Man(Raymond)

Absent: None

Recorder: Lo Wai Tong

1. Approval of minutes

The minute of the last meeting was approved without amendment.

2. Report on progress

- 2.1. We reallocated the work distribution due to change in group size (4 members to 3 members).
- 2.2. We have adjusted the Gantt chart.
- 2.3. We have refined the database structure and design.

3. Discussion items

- 3.1. We discussed some known issues and limitations of MongoDB, and discussed the solution on the problems.
- 3.2. We discussed and compared the web service between RESTful and SOAP(Axis2).
 - We will use RESTful web service instead of SOAP.
- 3.3. We discussed the user interface of the web application and the process flow of the functions.

4. Goals for the coming week

- 4.1. Tom will implement the API of the system.
- 4.2. Jeff will implement the web service using RESTFUL API.
- 4.3. Raymond will implement the web application.

5. Meeting adjournment and next meeting

The meeting was adjourned at 4:00 PM.

The date and time of the next meeting will be set later by e-mail.

9.6 Minutes of the 6th Project Meeting

Date: February 15, 2013

Time: 1:00 PM

Place: LG3

Present: Nikos ARMENATZOGLOU(Nick), Tsui Kam Ming(Tom), Lo Wai Tong (Jeff), Hou Wai Man(Raymond)

Absent: None

Recorder: Lo Wai Tong

1. Approval of minutes

The minute of the last meeting was approved without amendment.

2. Report on progress

- 2.1. We have refined the Database structure & design.
- 2.2. Tom has finished the majority part of API.
- 2.3. Jeff has finished the majority part of web service.
- 2.4. Raymond has finished some parts of web application.
- 2.5. Raymond successfully connected to web service and post/get data from DB.

3. Discussion items

- 3.1. We reported the FYP progress.
- 3.2. We did a demonstration on the finished functions.
- 3.3. We discussed the searching and query on the user checkin.
- 3.4. We further discussed on the FYP webpage layout.

4. Goals for the coming week

- 4.1. Tom will refine the API of DB.
- 4.2. Jeff will refine the web service.
- 4.3. Raymond will implement the rest of web application.

5. Meeting adjournment and next meeting

The meeting was adjourned at 3:00 PM.

The date and time of the next meeting will be set later by e-mail.

9.7 Minutes of the 7th Project Meeting

Date: March 13, 2013

Time: 1:00 PM

Place: LG3 Open Area

Present: Tsui Kam Ming(Tom), Lo Wai Tong (Jeff), Hou Wai Man(Raymond)

Absent: None

Recorder: Lo Wai Tong

1. Approval of minutes

The minute of the last meeting was approved without amendment.

2. Report on progress

2.1. Tom has refined the API of DB.

2.2. Jeff has refined the web service.

2.3. Raymond has finished the majority part of web application.

3. Discussion items

3.1. We tried to integrate our system.

3.2. We did a testing of the functions completed.

3.3. We further discussed on the FYP webpage layout.

4. Goals for the coming week

4.1. Tom will work on the final report.

4.2. Jeff will work on the final report.

4.3. Raymond will implement the mobile version of web application.

5. Meeting adjournment and next meeting

The meeting was adjourned at 3:00 PM.

The date and time of the next meeting will be set later by e-mail.

9.8 Minutes of the 8th Project Meeting

Date: April 3, 2013

Time: 4:00 PM

Place: LC-05

Present: Nikos ARMENATZOGLOU(Nick), Tsui Kam Ming(Tom), Lo Wai Tong (Jeff), Hou Wai Man(Raymond)

Absent: None

Recorder: Lo Wai Tong

1. Approval of minutes

The minute of the last meeting was approved without amendment.

2. Report on progress

2.1. Tom has finished the majority part of final report.

2.2. Jeff has finished part of final report.

2.3. Raymond has finished the web application and some parts of mobile version of web application.

3. Discussion items

3.1. We reported the FYP progress.

3.2. We did a demonstration on the whole system.

3.3. We discussed the new functions and pages need to be added.

3.4. We further discussed on the FYP webpage layout.

4. Goals for the coming week

4.1. Tom will add new functions and refine the API of DB.

4.2. Jeff will add new functions and refine the web service.

4.3. Raymond will refine the web application and implement the rest of mobile version of web application.

5. Meeting adjournment and next meeting

The meeting was adjourned at 7:00 PM.

The date and time of the next meeting will be set later by e-mail.

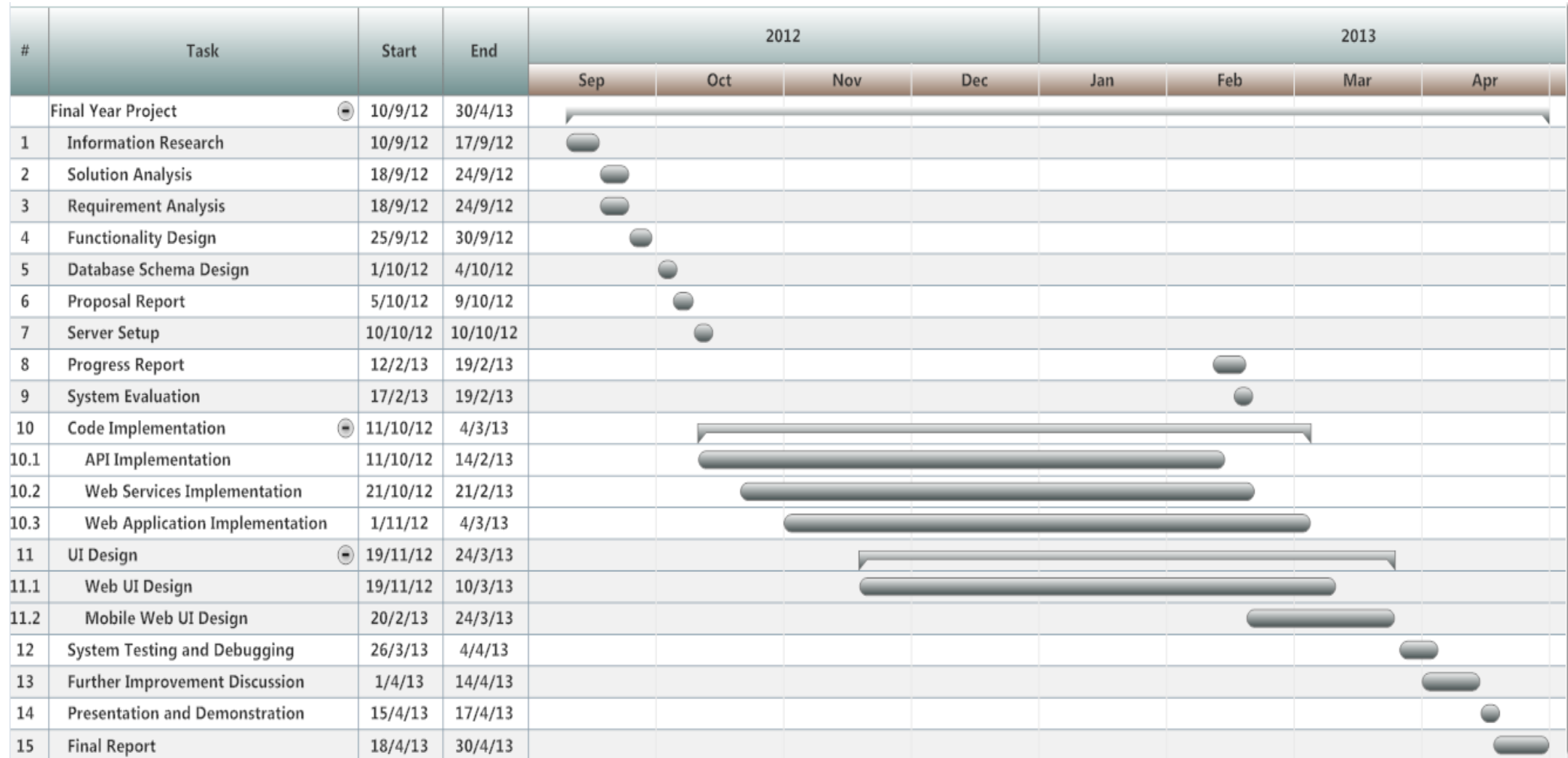
10. Appendix B: Project Planning

10.1 Division of Work

Task	Lo Wai Tong	Tsui Kam Ming	Hou Wai Man
Information Research	L	A	A
Requirement Analysis	L	L	L
Solution Analysis	A	L	A
Functionality Design	A	A	L
Proposal Report	A	A	L
Database Schema Design	A	L	
Server Setup	L	L	A
Web Services API Implementation	L	A	
Web UI Design	A		L
Web Application Implementation		A	L
Mobile Web UI Design	A		L
System Testing and Debugging	A	L	A
Progress Report	A	L	A
System Evaluation	L	L	A
Further Improvement Discussion	L	L	L
Final Report	A	L	L
Presentation and Demonstration	A	L	A

L: Leader for the task, A: Assistant for the task

10.2 GANTT Chart



11 Appendix C: Required Hardware and Software

11.1 Hardware

Computers (all computers should be able to connect to the internet):

- One computer with Ubuntu 12.04 64-bit (operating system), Java, eclipse IDE, MongoDB (with Java Driver) and JMongoBrowser installed to provide web service and database server
- One computer with Adobe Dreamweaver, Adobe Photoshop, Apache and PHP installed to act as the web server and for web application development
- One computer with HTML5 compatible web browsers as the client for testing

Mobile Devices (all mobile device should be able to connect to the internet):

Any smartphones/tablets with HTML5 compatible web browser as the client (for Android smartphone/tablet, recommended version is 2.3.3 or above)

11.2 Software

Java Platform, Standard Edition (Java SE) 1.7.0_05, with RESTful web service (JAX-RS) support

Eclipse IDE 4.2 Juno, Integrated Development Environment for java, with Apache Tomcat 7.0

MongoDB 2.2.0 (64-bit) and MongoDB Java Driver Version 2.9.1, database for the system

JMongoBrowser, GUI tool for manipulating the database

Apache 2.2 with PHP 5.3 for the web server

Notepad++ 5.8.2 for coding the webpage

Adobe Photoshop CS 5.5 for creating banners and images webpage

Mozilla Firefox 15.0+, Google Chrome, IE 9+, Safari 4.0, Browsers supporting HTML5 and javascript