WEB BASED APPLICATION FOR LOCATION SPECIFICATION BASED ON GOOGLE MAPS

PREM SICHARNUGRIST

SITTIDEJ THONGKUM

A SPECIAL PROJECT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR

THE DEGREE OF BECHELOR OF SCIENCE

INTERNATIONAL PROGRAMS, FACULTY OF SCIENCE

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

Special Project Title	Web Based Application for Location Specification Based On Google Maps
Student Names	Mr. Prem Sichanugrist Mr. Sittidej Thongkum
Faculty	Science
Program	Computer Science (International Program)
Special Project Advisor	Asst. Prof. Dr. Sarun Intakosum

ABSTRACT

The popularity of mapping application is raising incredible high. However, the existing applications are still not convenient and completed as they should. For instance, they are lacking of Thailand local information, the providers do not allow users to make changes to their system, inconvenient search engine, etc. The purpose of this special problem is to create a rich feature web-based mapping application based on Google Maps. The system was developed by using Ruby on Rails, JavaScript, and MySQL. In this Web Application, users are allow to publish new location with additional information into system, search for location, browse by category or tag, add incoming event on a location and more.

Acknowledgement

We would like to express our gratitude to all those who support us to successfully finish this project. I want to thank Asst. Prof. Dr. Sarun Intakosum, our project advisor, who provided useful advices and comments all the time of research and writing of this thesis.

We are deeply indebted to Asst. Prof. Dr. Veera Boonjing and Mr. Santana Oudomying for their good cares and helpful suggestions.We also want to thank every staff in the international office for information related to senior project.

My friends from Department of Computer Science International Program who supported us in our research work. I want to thank them for all help, goodwill, and friendship.

We are extremely grateful to say thank you at this opportunity.

Prem Sicharnugrist

Sittidej Thongkham

Table of Contents

List of Tables

List of Figures

Chapter 1
Introduction

1.1 Statements of Problem

A web-based map service is a service that provides geographical information via web browser by displaying maps or satellite images based on geographical location. Started at 2005, Google rolled out a web-based map service named Google Local, which later renamed to Google Maps. This service was specifically aimed to provide geographical map for some big cities, such as New York or Los Angeles, before later spread out to support more cities and countries around the world.

So finally in 2007, Google Maps starts to provide a satellite images for Thailand, which later followed by maps. By using Google Maps API, new web applications in Thailand are created to provide information based on geological location.

However, there isn't any web application that provides good geographical information of specific location such as search for location, since Google Maps doesn't contains enough information about locations in Thailand. There also isn't any web application that would provide useful information about specific location, such as local map, directories, feedback, or events that is going to be there. This problem makes a web-based map platform not so useful as it should be. Lastly, this information might not be periodically updated, so the information on the application's database might get obsolete in one day.

1.2 Purpose

From those stated problems, we are gosing to build a web application that stores and provides a geographical location and location's information via web interface. This web application should allow people to mark a location in the map to minimize complexity of finding geographical location's information. Also, we would allow user to add more useful information about that location by themselves, under control of community moderators, to keep the freshness of the information on the website.

1.3 Scope

The web application will be created based on Ruby on Rails framework, written in Ruby, for a fast and agile development process. It will be deploy on a web server running Apache web server with MySQL database

storage engine, and should be accessible over the Internet.

Even the above information specifies a scope of our application; there are some limitations that we should mention that it beyond our scope:

1. We will not be able to guarantee that all of the information on the application's database is 100% accurate, since it's a user generated content. However, a good moderation system should allow us to archive at least 90% of accurate data.
2. We will not be able to guarantee that the information on the map is accurate, since we're going to use geographical information based on Google Maps.
3. We will not be able to guaranteed user's experience, which contains uncontrollable factors such as connection speed, web browsers, and user's operating system.
4. We are going to make it compatible to major web browsers only, such as Microsoft Internet Explorer 6/7, Mozilla Firefox 2.0/3.0, Safari 2.0/3.0 and Opera 9. We're going to exclude a mobile platform for this instance, as some of the mobile phone still lacking some required features.

1.4 Expected Results

We're going to have a web application that provides a geographical location for Internet users in Thailand, along with useful information about that specific location. It also provides a map interface for any third-party to use as a reference. Also, it opens a new way of communication that people can generally give their feedback or response to a location, which might help to promote a particular location to be more popular.

Chapter 2

Related Literatures

2.1 Survey of Current Web-based Mapping Application

In early days, mapping application was not as popular as today since it was so inconvenient. People used world map to seek for countries. Travelers used road map to find their way to the destination. We used to draw a map to represent our house for visitors. There were many different categories of maps focus on one certain purpose.

Since Google released Google Maps in 2006, the best ever mapping application has been introduced to people. Google Maps can display map, satellite images and hybrid (combination map and satellite images). Google Maps is a zoom able mapping application. From the beginning, users may start from a world map view and then zoom to a specific district in one country.

Google provides Google Maps API for users to embed the application into website. Currently, there are several websites that use Google Maps API to develop their own applications which come with new features or apply with other applications. The following websites provide web-based mapping application based on Google Maps API.

2.1.1 Wikimapia

Wikimapia is a website that offers you an online mapping service. It combines Google Maps with a wiki system. It allows users to add information in any location on earth.

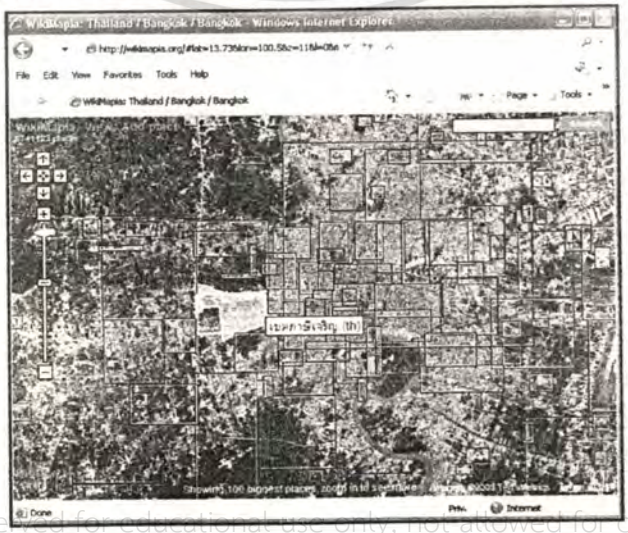

Figure 2.1: Wikimapia website

Features

1. Tag a location - Wikimapia allows user to mark a location (tag) by placing a polygon frame. Users can add information such as title, description, category, embedded videos and images.

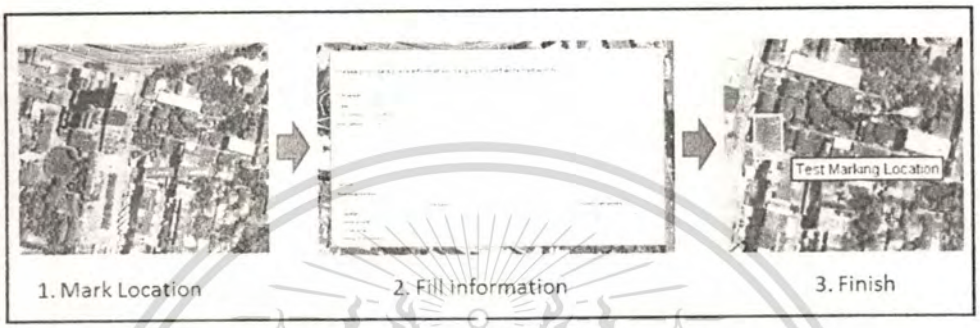

Figure 2.2: Wikimapia's process of adding new location

2. Geo tools - tools for measuring distance and area.

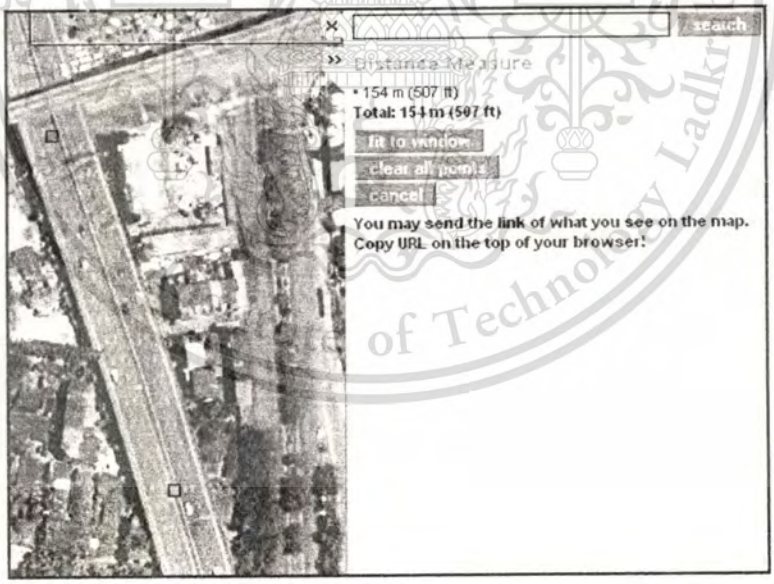

Figure 2.3: Wikimapia's measure distance feature

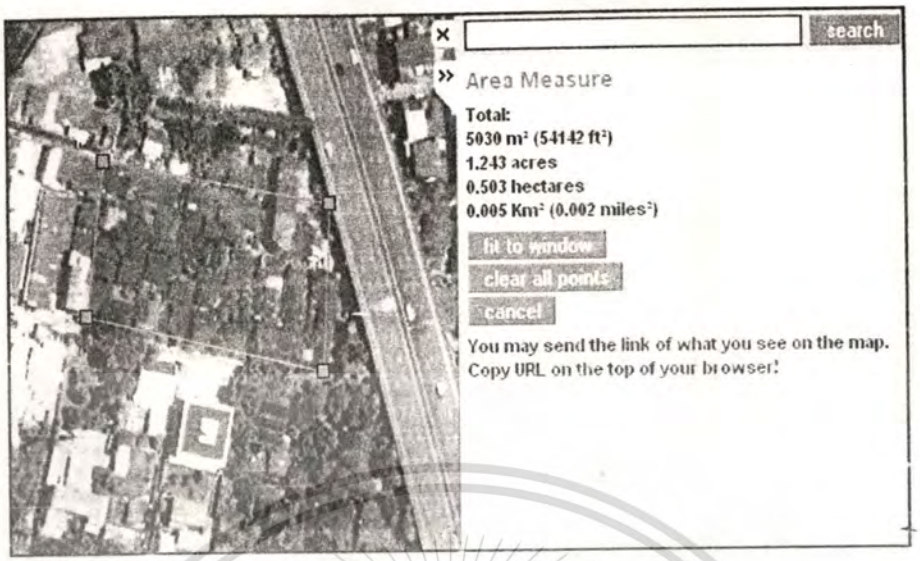

Figure 2.4: Wikimapia's measure area feature

3. Category Filter – choose to display locations that belong to the selected category

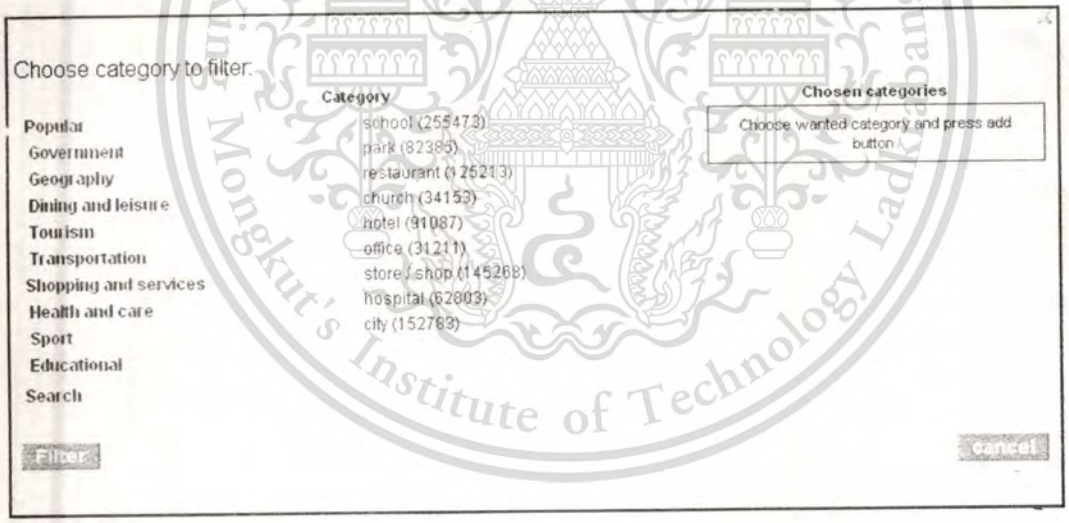

Figure 2.5: Wikimapia's category filters

Criticizes

1. No drawing tools for marking location. Wikimapia provides marking frame for adding location. The shape of frame can be editable by adding nodes or move position. However, if wikimapia provides additional drawing tools such as triangle, rectangle, circle and more, it would be more convinient for users to mark location more beautifully and more correctly.

2.1.2 Iam.in.th

Website iam.in.th provides information about events in Thailand. This web site aims to help people organize their time to event and also help agencies promote their event.

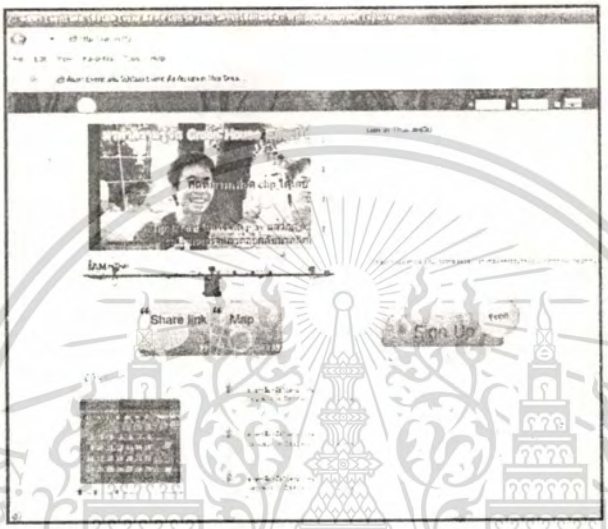

Figure 2.6: Iam.in.th website

Features

1. Social Bookmark - A feature that can store, classify, share and search event. Social bookmark stores information at the web site instead of storing into web browser. By doing this, users can join and get involve together, which follows the idea of web 2.0.

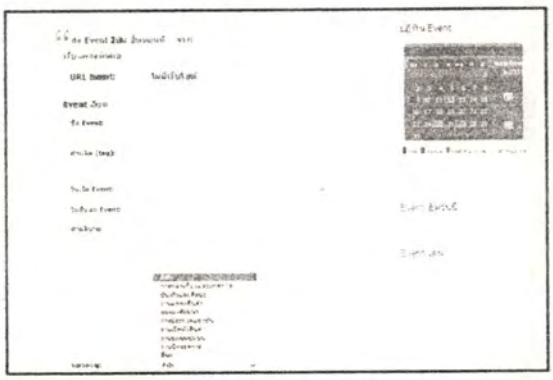

Figure 2.7: Iam.in.th interface of add new event

2. Internet TV – Currently, there are 4 channels available in Internet TV of I am in Thailand. Channel 1 promotes the incoming events, it on air twice a month. Channel 2 - 4 show extra content about the event such as interview of the organizer, behind the scene and snap shots of the event. Unfortunately, these three channels have no fixed on-air time due to the issue of opportunity to participate in events.
3. Podcast – The service that provides updated information about the event via mail client or RSS reader.
4. Google Maps mash-up – I am in Thailand uses Google Maps to display the location of the event.

Figure 2.8: Locate an event on Iam.in.th using Google Maps

Criticizes

1. No additional information about the location. It should be more convenient for users to be able to see information about location that is holding the event. This information may help users to make their decision.
2. No method to handle multiple events at the same location. Seeing two marks of two different events which is holding at the same place is not a good sight for users. If multiple events can be displayed by one marker, the screen is organized.

2.1.3 Buffet-Thailand

Website Buffet-Thailand gives information about hotel or buffet restaurant in Thailand. Users can find information of the buffet by cuisine, price, and hotel name. Buffet-Thailand allows users to share any comment about the restaurant, pictures, and ratings.

Figure 2.9: Buffet-Thailand website

Features

1. Buffet restaurant information – Restaurant name, host, cuisine and prices

Figure 2.10: Buffet resturant's information on Buffet-Thailand

2. **Buffet Map - Locating the buffet restaurant by using Google Maps**

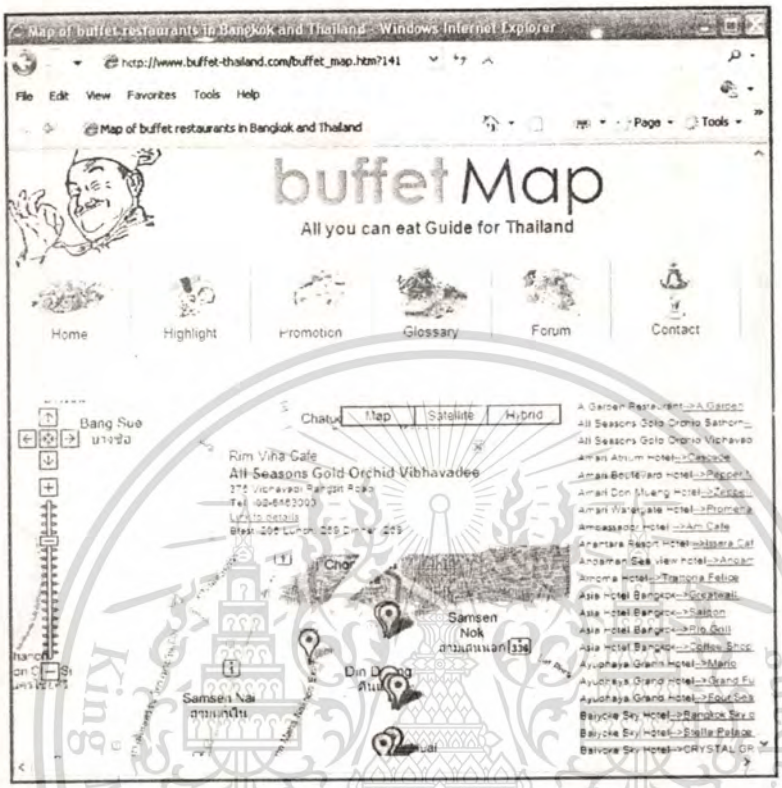

Figure 2.11: BuffetMap interface on Buffet-Thailand

Criticizes

1. Lack of search engine. Buffet-Thailand shows the list of buffet restaurant which can be sorted by restaurant name, host, cuisine, and prices. However, Buffet-Thailand doesn't provide a searching service for users. So, it might be inconvenient when users want to specify many criteria for searching such as find the buffet restaurant that serves Japanese food in Bangkok which fits to budget of 1,000 Baths.
2. No accumulative rating. Since Buffet-Thailand lets their users to give rating of the buffet restaurant, but they doesn't provides the overall rating of the certain restaurant. It is difficult for users to count the rating by themselves and it is inconvenient after all.
3. Users can't add new buffet restaurant. Without users help, this could slows down the growth of website.

4. Buffet Restaurant information can't be editable.

2.1.4 Longdo Map

Longdo Map, a web-based map application, focuses on Thailand Map Only. The map engine of this website is created by their developers. However, Longdo Map also provides Google Maps engine for their users as the alternative choice.

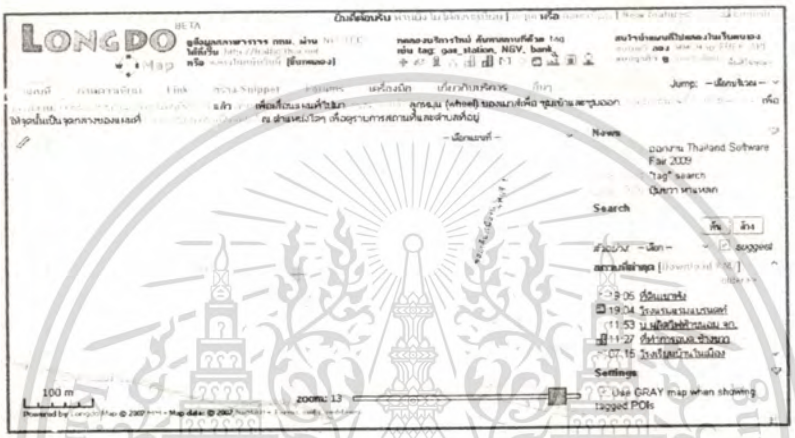

Figure 2.12: Longdo Map Interface

Features

1. Add new Location along with the location information.
2. Updated content announcement.
3. Various type of map. For example, road map, traffic map, train map and more.

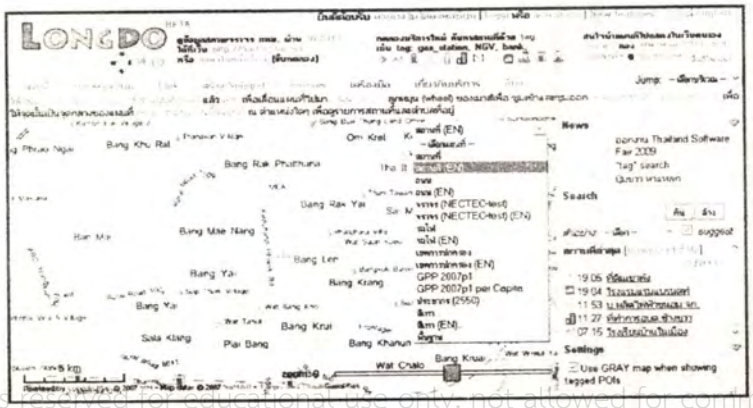

Figure 2.13: Categories in Longdo Map

4. **Google Maps engine that connect to their own database.**

Criticizes

1. Location information is not editable.
2. Can't embedded map

Chapter 3

Web Based Application for Location Specification Based on Google Maps

3.1 System Overview

This system is a web based application for location specification based on Google map that allow users to add a location and its information into system, such as address, description and incoming events. Location's information is organized by category and tag that's easy to browse. The system also provides searching function that should be able to search through categories, tags, and location's information. We also want this system to be a social networking application as well. Therefore, functions that related to social networking concept are included into the system; there are comments on location, shared photos and participation with friends. The system is under care of moderator who responsible for feedback, correctness of information and user manner.

3.2 System Requirements

According to the survey from Chapter 2 and system overview in Chapter 3, if we're going to build a web-based map application that provide event information, these requirements are necessary:

1. Allow user to publish a new location to database based on latitude and longitude of that location.
2. Allow user to search for a location by using keyword, or browse through locations by categories.
3. Allow user to view location's information, along with its geological location.
4. Allow user to pass that location's or event's information to their friends.
5. Allow user to add a useful feedback, or tag to the location or event.
6. Allow user to add a new event that will be in that particular location.
7. Allow user to add photos that had been taken in that location.
8. Allow user to send report of incorrect location, abusive feedback, or wrong information.

We also have called our application as LocalMapia, as the word derived from Local and Map. We

will use this name to refer to our application.

3.3 System Design

3.3.1 System Architecture

Figure 3.1 illustrates the structure of the system or site map. From index page, user can browses through the following contents:

1. Search – search engine for finding location in website.
2. Browse – redirect user to category and tag page, both of them will contain link to location's page according to content. From location page, user can view location's detail, map, and incoming events. They also can add incoming event from this page as well.
3. Share – page for adding location and information.

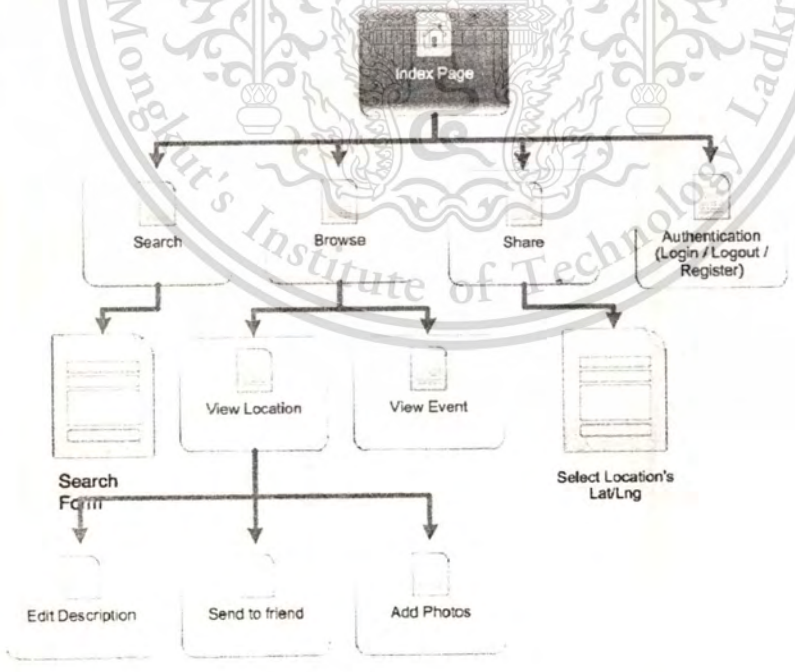

Figure 3.1: LocalMapia's Sitemap

3.3.2 Class Diagram

Since we're creating the application using MVC, we firstly design our application as a group of models that interacts with each other. In Figure 3.2, you will see how each model connects.

You'll notice that User model associates with many other models, since our application tends to be user oriented (He or she will creates the location, event by themselves,) and that Location and Event models have an association to its _version table, which means that every changes made to the model, we should backup the previous version just in case something went wrong.

3.3.3 Database

As we collect information from user, we need a database to store the information. We'll need to create database tables to map with our models we talked about, as shown in table 3.1. There're 18 tables in the database, and each database will be mapped as an object using ActiveRord. In this section, we're going to show you only main tables in the database. For the -

No.	Table Name	Description
1	categories	table that stores categories
2	category_fields	table that stores
3	event_categories	table that stores categories which related to event
4	event_logs	table that stores user action
5	event_photos	table that stores data about which photo belong to one event
6	event_versions	table that stores previous version of event information
7	events	table that stores events
8	events_tags	table that stores tags which related to event
9	events_users	table that stores user id whose going to participate the event
10	feedbacks	table that stores comment from users about the event
11	locations	table that stores location information
12	locations_categories	table that stores which location belong to which category
13	locations_region	table that stores latitude and lontitude of locations
14	locations_tag	table that stores tags which related to location
15	locations_versions	table that stores previous version of location information

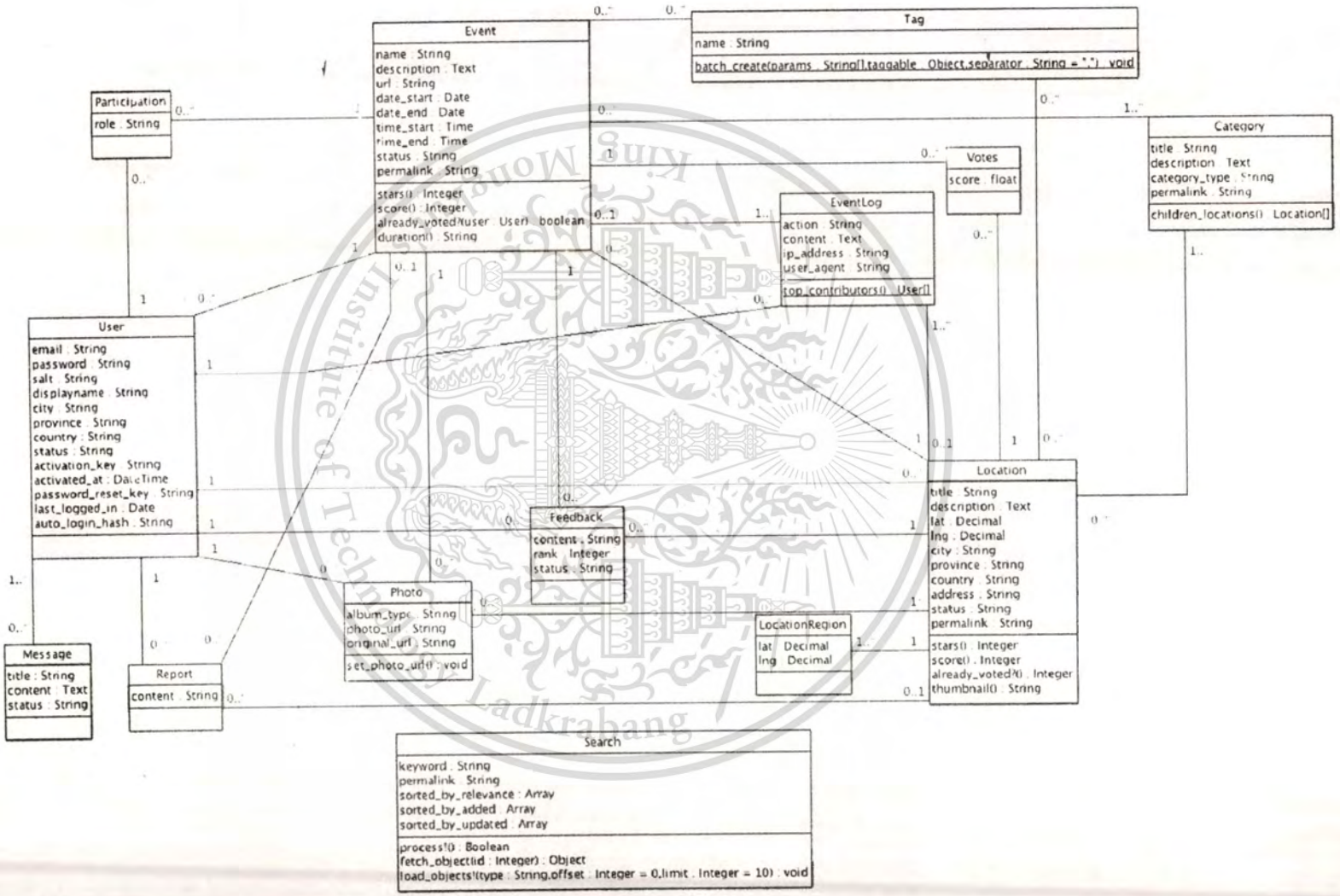

Figure 3.2: LocalMapia's class diagram

Table 3.1 (cont)

No.	Table Name	Description
16	messages	table that stores private messages among users
17	participations	table that stores user's will to participate in the event
18	photos	table that stores photos
19	reports	table that stores user abusement report
20	search	stores the result set of search
21	sessions	table that stores user session
22	tags	table that stores tag
23	users	table that stores user information
24	users_friends	table that sotres friends of user

Table 3.1 – Locations Table

No.	field name	data type	description
1	user_id	integer	user id
2	parent_id	integer	category id
3	title	string	location title
4	description	text	location's description
5	lat	float	latitude point
6	lng	float	longtitude point
7	city	string	city
8	province	string	province
9	country	string	country
10	status	string	record status
11	score	integer	rating
12	vote_count	integer	recommendation vote
13	additional_info	text	additional information
14	created_at	datetime	created date and time
15	updated_at	datetime	updated date and time
16	permalink	string	permalink
17	address	string	location address
18	version	integer	version number

Table 3.2 – Categories Table

No.	field name	data type	description
1	parent_id	integer	parent id
2	title	string	category title
3	description	text	description
4	navigation	string	stores parent category
5	photo	string	photo
6	category_type	string	category type
7	category_type	string	category type

Table 3.3 – Feedbacks Table

No.	field name	data type	description
1	location_id	integer	location id that the feedback
2	event_id	integer	event id where the feed back
3	user_id	integer	poster id
4	content	text	feedback
5	rank	string	ranking
6	status	string	feedback status
7	created_at	datetime	created date and time
8	updated_at	datetime	updated date and time

Table 3.4 – Users Table

No.	field name	data type	description
1	email	string	user's e-mail
2	encrypted password	string	user's password
3	salt	string	stores hash code for user password encryption
4	displayname	string	user's display name
5	city	string	city where user belongs
6	province	string	province where user belongs
7	country	string	country where user belongs
8	status	string	record status
9	activation key	string	activation key
10	activated_at	datetime	account activated date and time
11	password_reset_key	string	password reset key
12	last_logged_in	date	last logged in
13	created_at	datetime	created date and time
14	updated_at	datetime	updated date and time

3.3.4 Sequence Diagrams

In this section, we'll show sequence diagrams that illustrate how the system works in background. However, there are many sequence diagrams that describe our system, so we'll show you a few class diagrams in this section.

3.3.4.1 Add new location

For adding a new location, first user will navigate to /locations/new/step1, which will call LocationsController in the step1 action. That LocationsController will create a new instance of Location model, and returns a map for user to select region. User will

then submit the region to /locations, which will call create action in LocationsController to store those data temporary in the session and redirects user back to /locations/new/step2. User will need to enter location's information on that form and submit it too /locations. An create action will be caled again, but now it will store the location in the database and redirects user back to the newly created location's page.

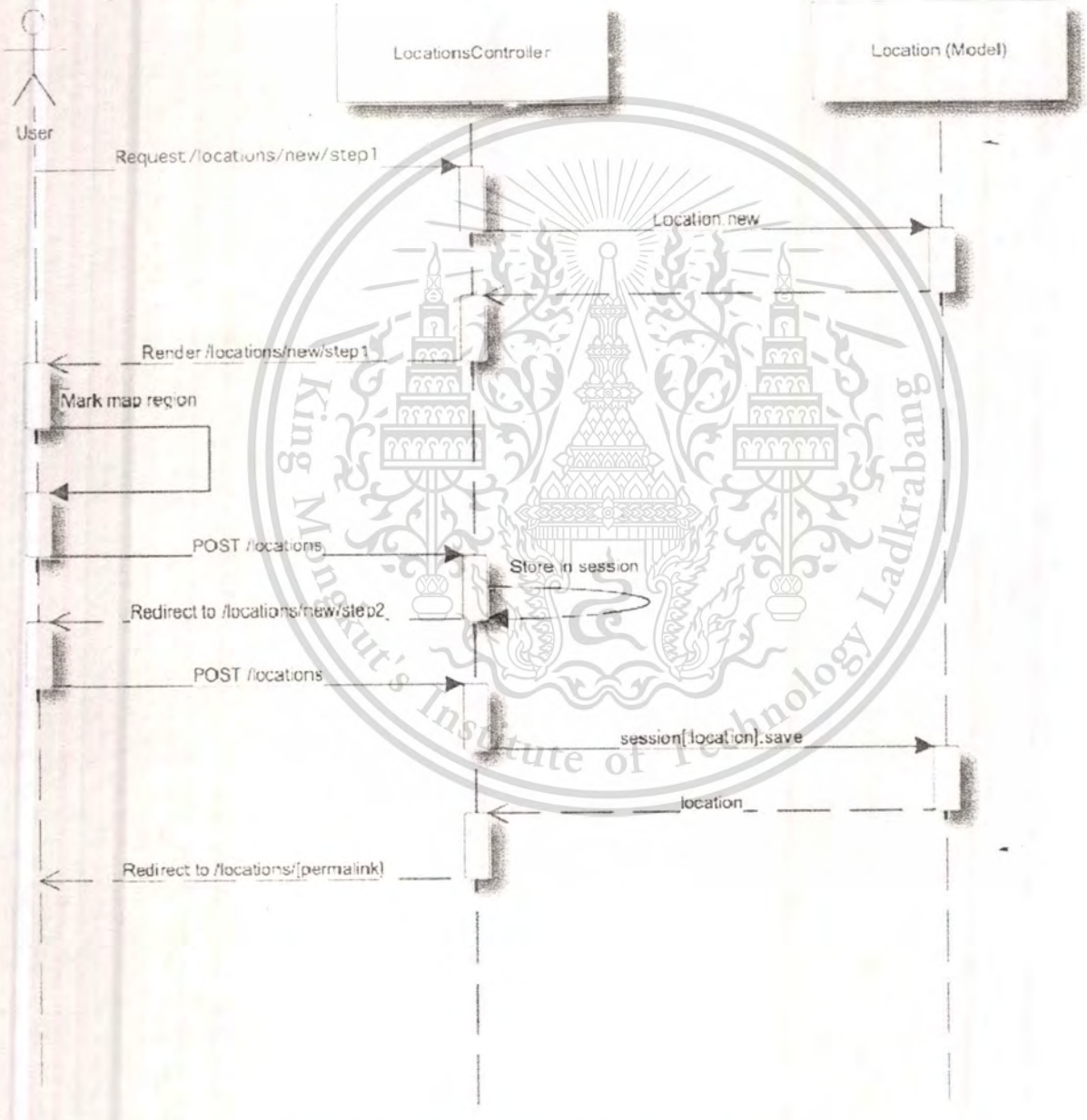

Figure 3.3: Sequence diagram of adding new location

3.3.4.2 Display new location

To display a location, user will first make a request to location's permalink, such as /locations/[id]-[location_name]. It will calls LocationController's show action, which will load a corresponding location from Location model. After that, LocationsController will render that show action with an API call to Google Maps API. Then the browser will issue the call to API and renders the map with its corresponding region on the page.

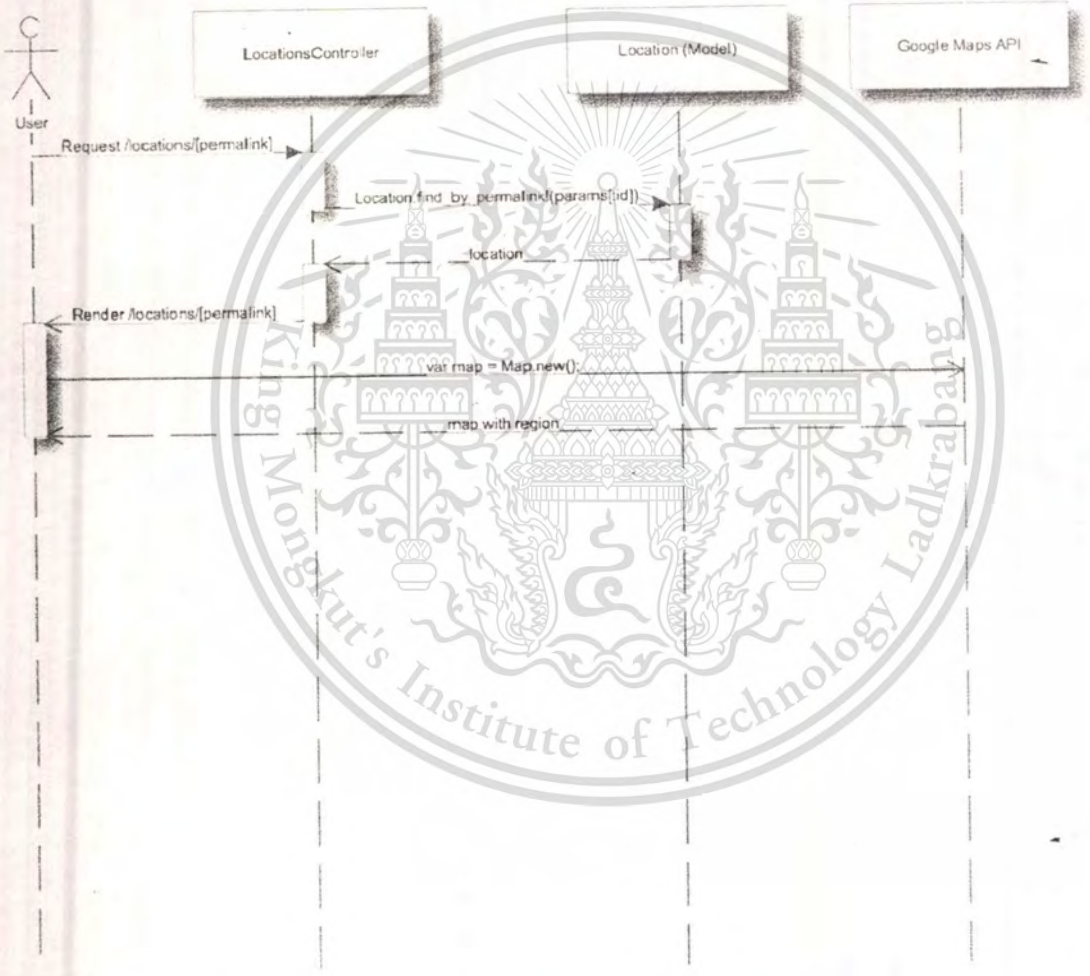

Figure 3.4: Sequence diagram of displaying location process

3.3.4.3 Search for locations

To search for a location, user first make a GET request to /searches, which will calls index action in SearchesController. That index action then will create a new search model, and renders a form back to user.

After that, user will enter the search term and submit a POST request to /searches. This will make a call to SearchesController's create action, and it will search for data by following this procedure:

- Search for locations and events based on the keyword, and store the id of those objects along with number of occurance of the keyword in an array, which wili be used for sorting the result based on occurance.
- Lookup on Tag for a tag that has the same name as the keyword, store that tag for later use.
- If a tag is found, then load locations and events that associates with that tag, along with number of occurance of that keyword in those objects. Then store those data into the array.
- Union those arrays created from step 1 and step 3 to remove the duplicate data from the array.
- Sort those data based on the number of occurance, created date, and last updated date.
- Store those arrays into the database

After the search process is done, those result will be stored in the database as an array. Then SearchesController will redirects that user to the search result. The whole process is shown in Figure 3.5

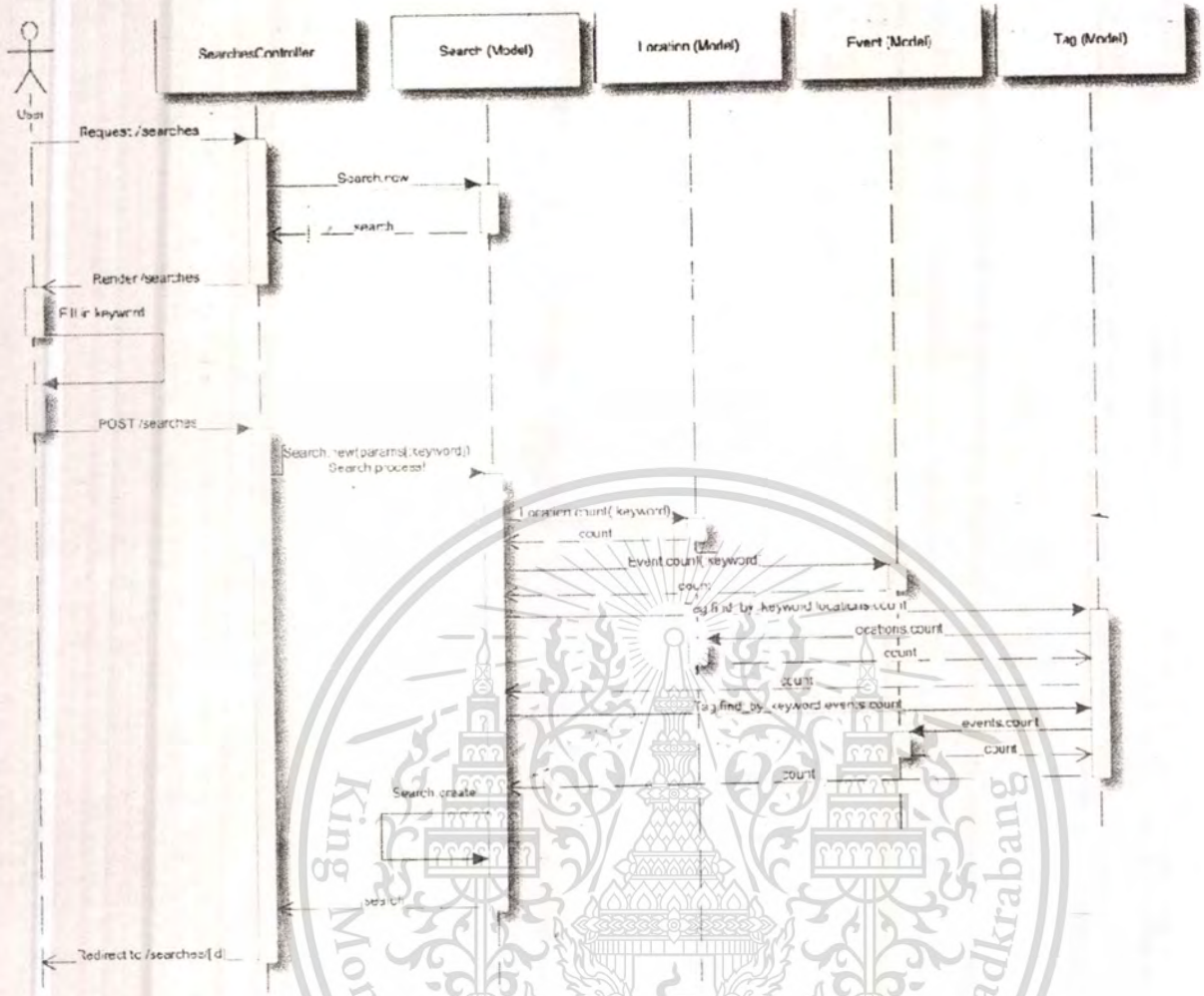

Figure 3.5: Sequence diagram of search process

3.3.5 Interaction with Google Map

Google allows developers to embed the Google maps via Google Maps API. JavaScript takes roll as the script language that used to interact with the Google Maps API. In this section, a few sample of JavaScirpt codes will be shown here as example of how do we interact with Google Maps engine.

3.3.5.1 Publish new location

In this process, we'll ask user to mark a location region by mark nodes on the map and javascript call Google Maps API to generate the grid from these node. Figure 3.6 illustrates the javascript that control this process.

```
function initialize_map(lat, lng, zoom){
 map = new GMap2(document.getElementById("add-location-map"));
 map.setCenter(new GLatLng(lat, lng), zoom);
 map.setUIToDefault();

 // ...

 $('#mark_region_button').click(function(){
  region = new GPolygon([], "#f33f00", 5, 1, "ff0000", 0.2);
  map.addOverlay(region);
  region.enableDrawing();
  map.disableDragging();
  map.disableDoubleClickZoom();
  map.disableScrollWheelZoom();

  GEvent.addListener(region, "endline", function(){
   //
   // ... Trigger next button to activate
   //
  });
 });

 // ...
}
```

Figure 3.6: Javascript code used for mark region

From Figure 3.6, first we need to create object of GPolygon, this object will be used to draw a grip that crops over the region. And then we add this object to Google Maps using method addOverlay(). After that, method enableDrawing() will make this GPolygon object to drawable.

3.3.5.2 View Map

In view map process, we also use javascript to display the map. From figure 3.6, we first create the map object, GMap2. Second step, we create a variable called latlng, it is used to store value of coordinate latitude and longitude. This step can be done by using method GLanLng().

```
function initialize_map(lat, lng, regions){
 map = new GMap2(document.getElementById("map-display"));
 var latlng = new GLatLng(lat, lng);
 map.setCenter(latlng, 17);
 map.disableDragging();
 map.disableDoubleClickZoom();
 map.disableScrollWheelZoom();
 map.addOverlay(new GMarker(latlng));
 map.addOverlay(new GPolygon(regions, "#f33f00", 2, 1, "#ff0000", 0.2));
}
```

Figure 3.7: Javascript code used for display map

Chapter 4

Result and Evaluation

4.1 System Operation

4.1.1 Index page or Home page

When users access to Localmapia website, they'll be redirected to the index page. This page gives user the overview of the site, and also some information that users might be interest. In the index page, users will see as in Figure 4.1, and it contains the following:

A. Log-in field or user's account information in case that they've logged in.

B. Link to search function

C. Link to browse location page

D. Link to publish new location page

E. Top featured places

F. Top recommended places

G. Recently added locations

H. Updated locations

I. Top contributors

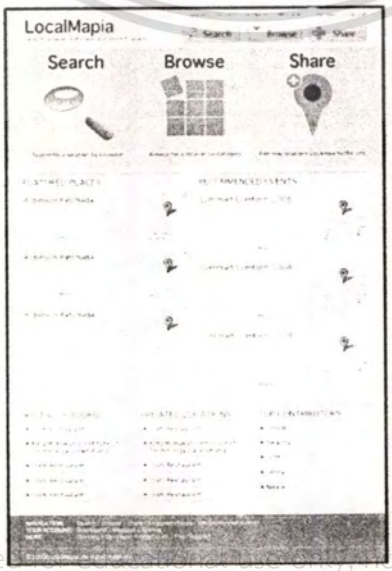

Figure 4.1: LocalMapia's homepage

4.1.2 Publish a new location

There will be 2 steps in publishing a new location, they are locate, and add information. Figure 4.2 illustrates the first step, locate. In locate process, user should navigate to a specific location he wants to add, and then mark the location by dragging a gird to cover area, and then click next.

Figure 4.2: Add new location – first step

After done with step 1, user will be redirected to step 2, as shown in figure 4.3. In this step, user will add some information such as name, description, tags, and category. After finished this step, users are done, see figure 4.4.

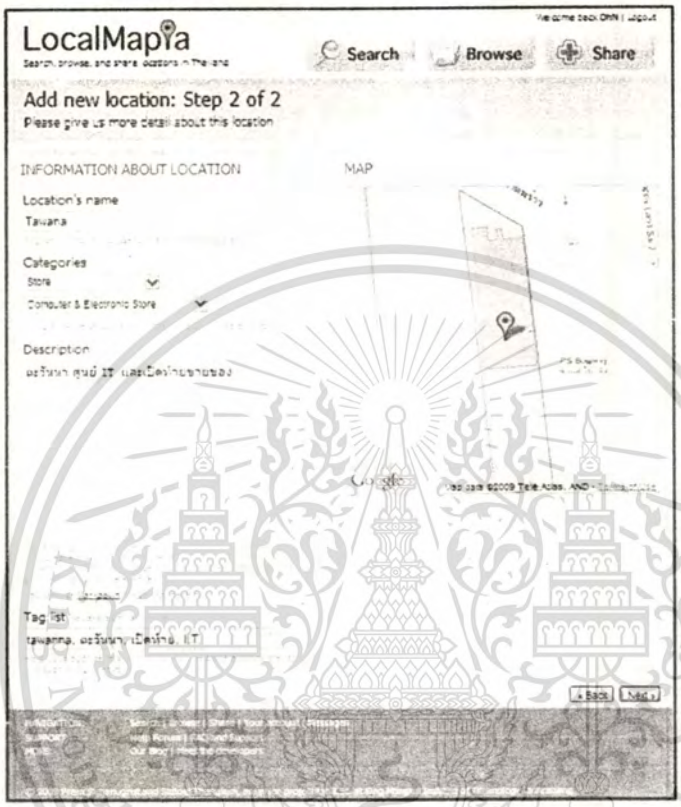

Figure 4.3: Add new location – second step

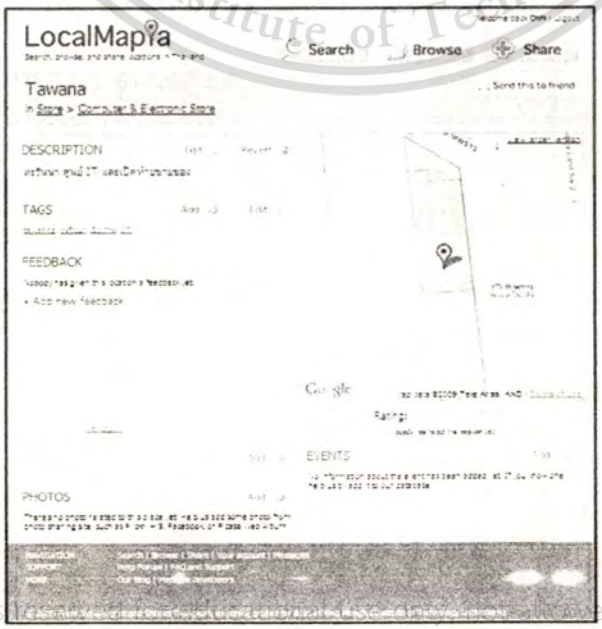

Figure 4.4: Add new location – result page

4.1.3 Search for location

User can navigate to the search page, and then specify a keyword in the given box. The result will be displayed on the result page, ordered by user-selected criteria, as shown in Figure 4.5.

Figure 4.5: Search result page

4.1.4 Browse for location

Users can browse for a location based on category or tag. There will be category listing and tag listing as you can see in Figure 4.6. Note that the tag list will display only 30 popular tags in the system. The bigger tag size means the larger amount of item which is using that tag.

Figure 4.6: Browse page containing categories and tags listing

After user has chosen a category or tag, user will be sent to listing page. The listing page will look almost identical to search page, apart from search box. This page is shown in Figure 4.7.

Figure 4.7: Category / tag listing page

4.1.5 View location's information

In this page, shown in Figure 4.8, user will be able to view location's information that stored in server, such as description and tags. User can also modify these contents, but some of them might have to be approved by moderator first. And more, user can add feedback and rating by a form provided in this page.

Figure 4.8: Location's detail page

4.1.6 Add new events

User will be able to add information about the event that will be held in this particular location. This information will be made public and listed in location's information page. This is shown in Figure 4.9.

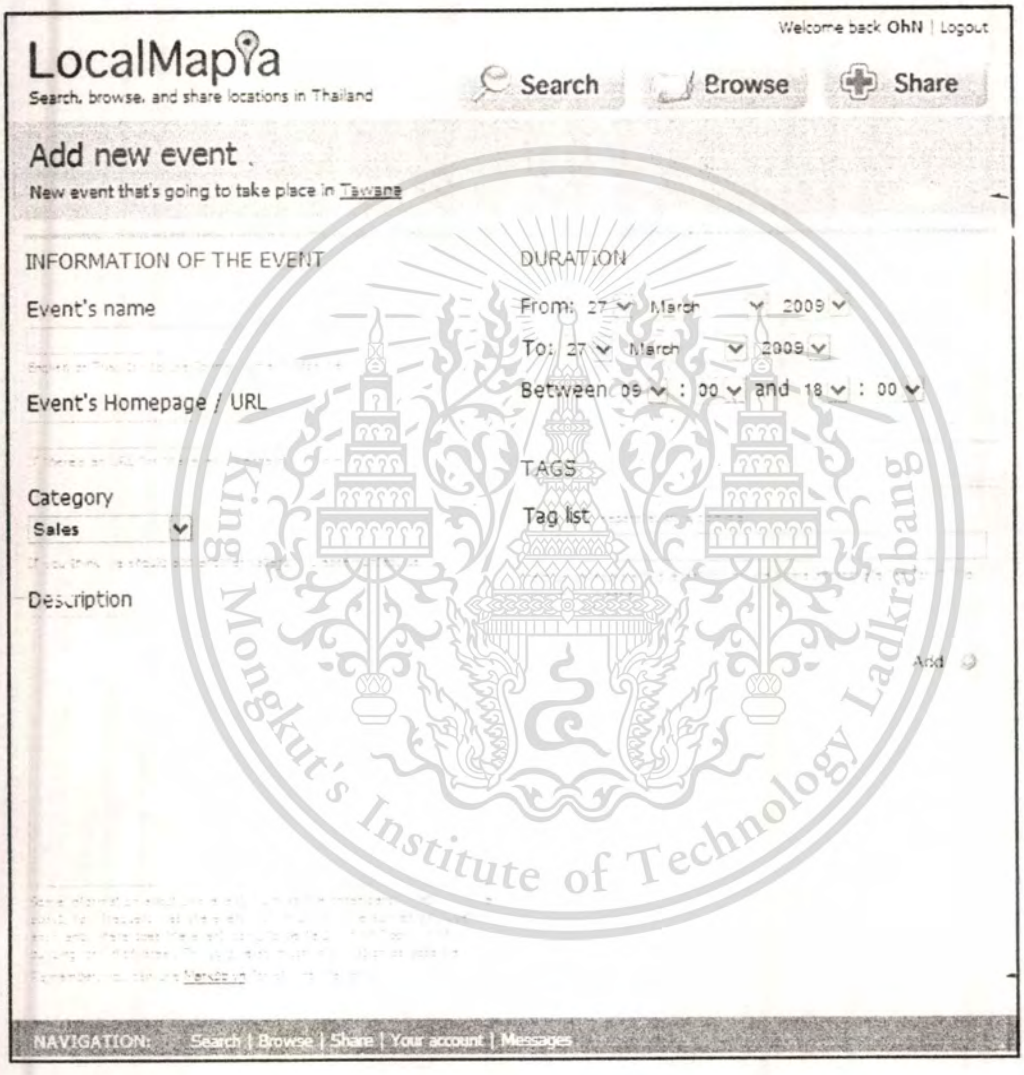

Figure 4.9: Add new event page

4.1.7 View event's information

User can access to this page from browse page or location's information. View event's information page will contain information about the event such as date and time, description and tags. This page also allows user to add feedback from attending the event. It is shown in Figure 4.10.

Figure 4.10: Event's information page

4.1.8 Report inappropriate content

When users find out that there is an inappropriate content on our website, they can report the issue to system moderator via report inappropriate content link. The link is located at the

right-bottom of location and event page. The report form is shown in Figure 4.11.

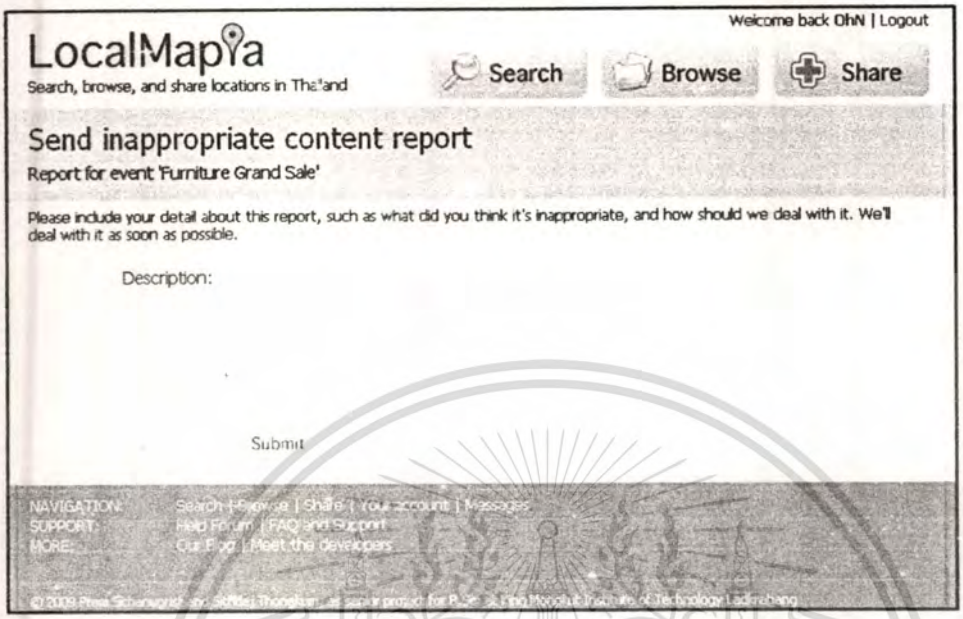

Figure 4.11: Report inappropriate content form

4.2 Evaluation

Our system is a rich feature mapping application. Therefore, we set an evaluation technique by comparing features with current major web mapping application.

The candidates that going to be our comparator are Google Map, Wikimapia, Map Longdo, and Iam.in.th. Table 4.1 shows the comparison result.

Table 4.1 – Feature comparison table

No.	Feature	Google Map	Wikimapia	Map Longdo	Iam.in.th	**Localmapia**
1	add location	x	√	√	x	√
2	edit location's information	x	√	x	x	√
3	add incoming event	x	x	√	√	√
4	give comment	x	√	x	√	√
5	search		√	√	√	√
6	category filter	x	√	√	√	√
7	tag	x	x	x	√	√
8	measurement tools		√	x	x	x
9	display updated information	x	x	√	√	√
10	upload photos	x	√	√	x	√

From Table 4.1, Google Map is lacking of ability to add or edit content in their system since

Google provides all of information in this system. Wikimapia is far better than that Google Map, their map is editable, it also provide filters and users can be able to give comment. But Wikimapia do not have feature of social networking and updated content announcement. Map Longdo is a mapping application that focuses on Thailand only. In Longdo Map, users can add location with some information but this information is not editable once it is added. The updated content, new location will be announced in their webpage. However, the user cannot give any comment to a specific location. Therefore, they also can't participate with each other as well. Iam.in.th is doing well in many areas. Although their map can't add any location but it shows where the event is located. Our system, Localmapia is the richest feature mapping application among others. In our system, user can add location and they also could be able to edit any content about location. Although, we do not provide measurement tools for users but if they encourage us to have this feature for them, we'll do it as they wish.

Even though our system is the richest feature one, but users decide the success of this application. The best one may not be the succeeded one. Also, it will take some time until our application is wildly spread to the users.

Chapter 5

Conclusion and Recommendation

5.1 Conclusion

Currently, the mapping applications are becoming a well-used application in our life. Users may use it for general purpose such as finding direction, embedded map, or browse and view. Unfortunately, the existing applications are lacking of local information and their features are limited. For example, Google Map's users can't add any addition information into the system while Wikimapia allows their users to do so. However, Wikimapia system doesn't have a photo hosting function.

We've developed the web application that can store and provide a geographical location and location's information via web interface. Users can add a location into the map and additional information about location such as address, events, and photos. In this system, user's actions are under control of community moderators, to keep freshness of the information on website.

We've almost successfully satisfy with our requirements in the first place, except for some parts that couldn't be done in time, or not well-documented. Even we're done with this project for the university, however, we'll try to introduce new features to fulfill those requirements and some more additions that should be added to make this application better.

5.2 Problems and Solutions

Problem: We were unable to run our application using preferred application stack, consist of Apache with mod_rails, and MySQL

Solution: Use an alternative stack, consist of Nginx running as reverse proxy load-balancers to clusters of application server using Thin. This reduces memory usage by almost 50%.

Problem: We were trying to find the way to allow people to customize their posing, but we think we shouldn't allow them to use HTML because it open more risks to the attacker

Solution: We've decided to use Markdown as our syntax formatting. Markdown provides lots of flexibility over styling the text. Also, there're lots of plug-ins that we can

include into our project immediately

5.3 Recommendation

There are many areas where we can improve our system. For example:

1. District filtering. The system should be able to display the locations in the specified district.
2. Event recommendation based on user's interests. The system asks users about things that he likes or dislike and then generate a list of events that they'd loved.
3. Calendar synchronization. When users found the interesting event on website, they should be able to mark the date that event will be held or the date that they'll go there. For user convenient, the system should be able to sync these dates with calendar from application such as Microsoft Outlook, Mozilla Thunderbird, Mobile Me, or Google Calendars.
4. Share location information from Google Maps. The system should be able to retrieve location information from Google Maps database and display in system itself. This way may help us increase the numbers of information in the system.
5. Implements some kind of search algorithm, such as full text search, to rank the search result more related to user's keyword.
6. Implements custom fields for each type of location. The model and structures had already been designed, but the system hasn't been implemented yet.

References

http://code.google.com/apis/maps/ – Google Maps API documentation

http://maps.google.com – Google Maps

http://git-scm.com – Git's official website

http://jquery.com – jQuery, the lightweight Javascript framework

http://daringfireball.net/projects/markdown/syntax – Markdown syntax

http://www.mysql.com – MySQL Website

http://github.com – Online Git repository and community

http://rubyonrails.org – Ruby on Rails official website

Appendix A

User Manual

A.1 Registeration

I. Access to Localmapia website via this URL, localmapia.com. From home page, click register link at the top left of home page.

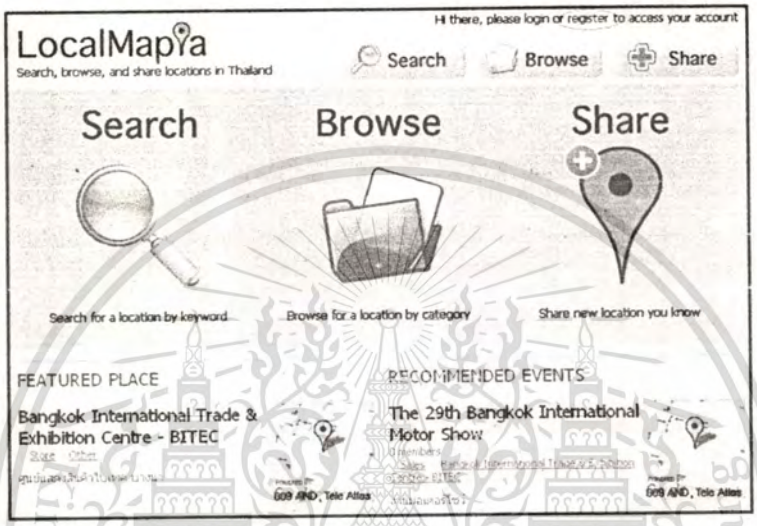

Figure A.1: Register link

II. The site will display a register form. User needs to enter 4 information, E-mail, password, password confirmation, and display name, and then press Register Button.

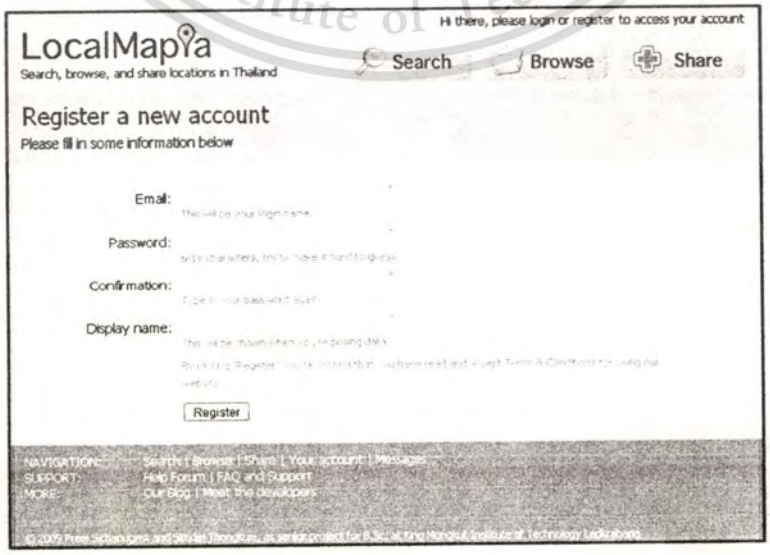

Figure A.2: Registration form

III. **The site will display the register result and it has sent you an activation e-mail.**

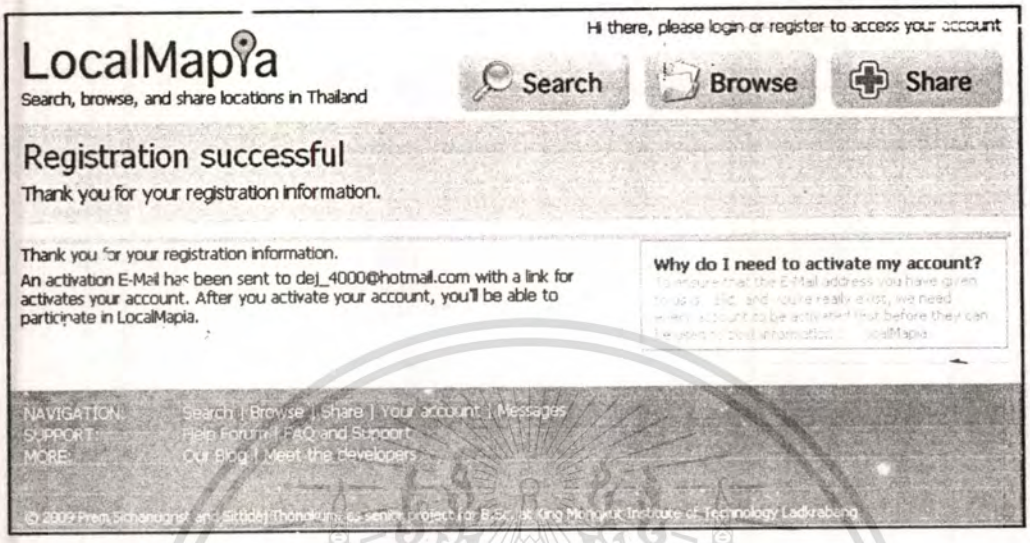

Figure A.3: Registeration result

IV. Go to your mail box and open the activation e-mail. Subject of the e-mail is written as "[Localmapia] Your activation instructions". This e-mail contains the link to activate your locamapia account. Click on the link, you'll be redirected to localmapia website and the activation is completed.

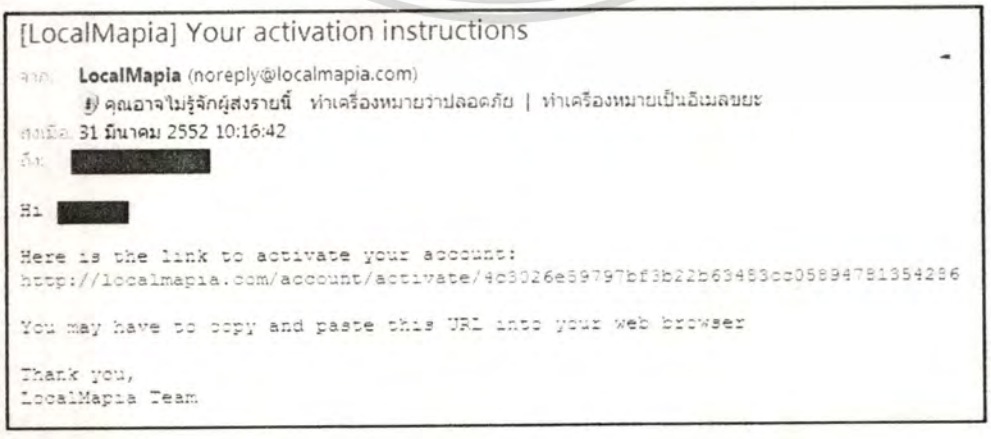

Figure A.4: Example of activation e-mail

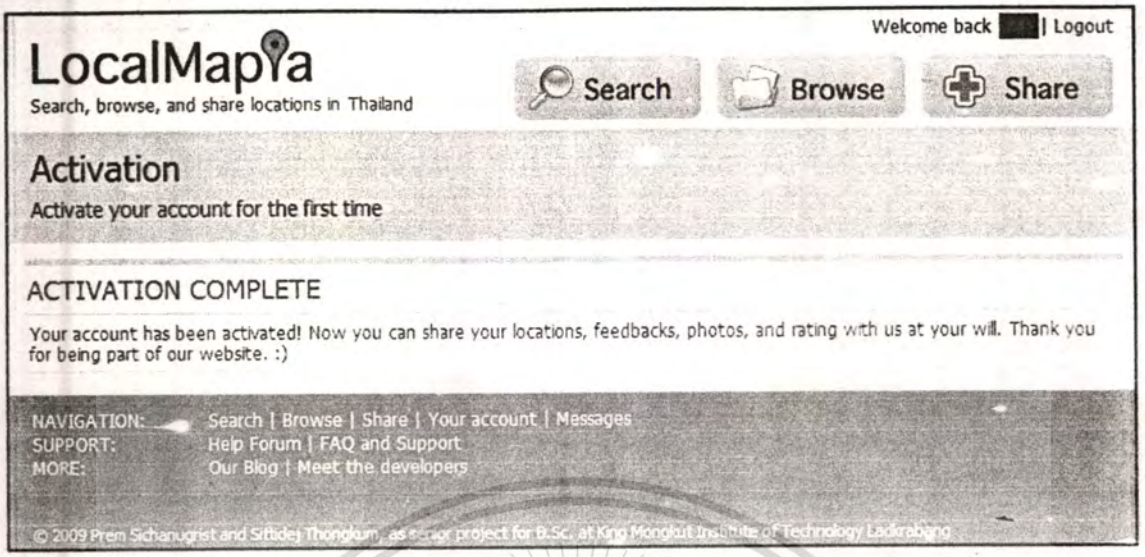

Figure A.5: Activation result

A.2 Log-in

I. From home page, click login link at top left corner.

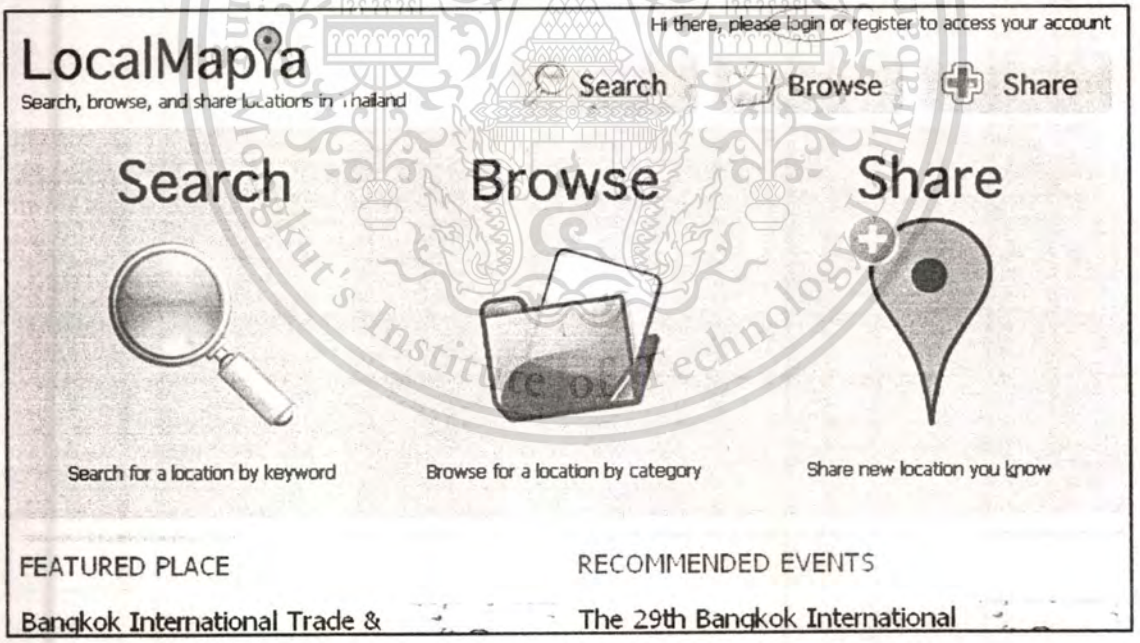

Figure A.6: Login link

II. The link will redirect you to a log-in form. After you've entered username and password, click submit button.

Figure A.7: Login form

III. You'll be redirected to home page. You can see you log-in status from text bar at top left of home page. If you are logged in, it should display a word "Welcome back" and follows by your user name.

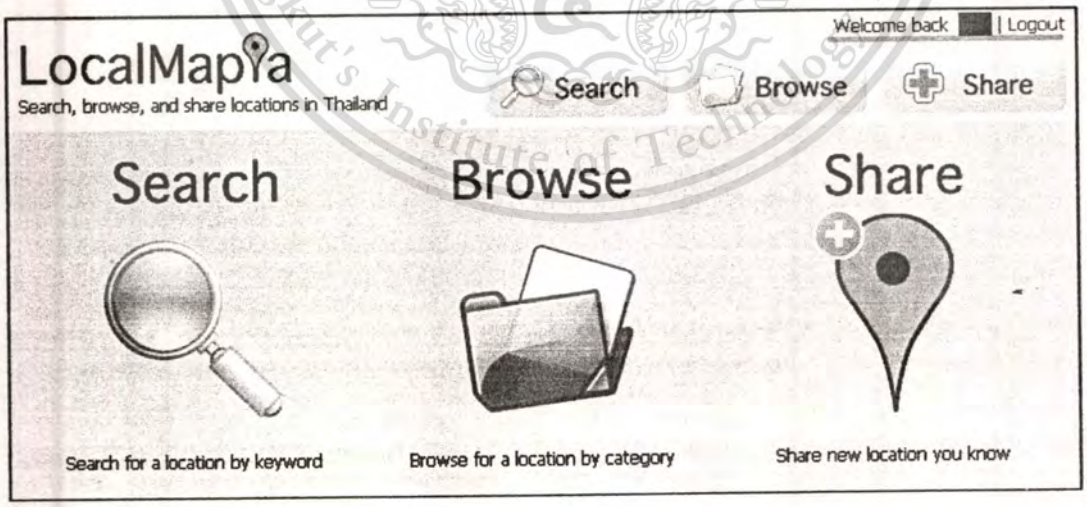

Figure A.8: Login status

A.3 Browse

I. From Home page, click browse.

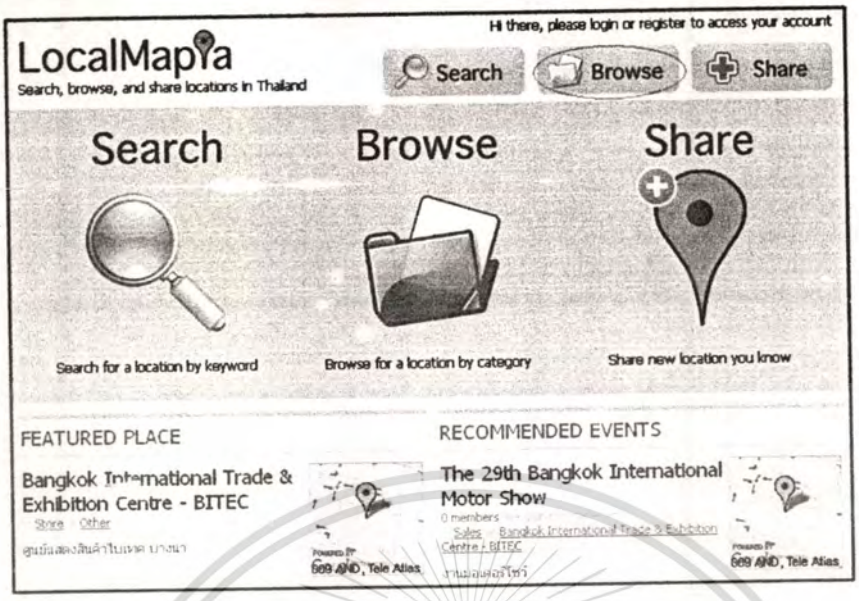

Figure A.9: Link to browse page

II. You'll be redirected to browse page. There are 3 sections in this page, browse for location, browse for event and browse by tag. In more details, Browse for location contains location category and sub category, Browse for event contains event category, and Browse by tag contains tag header.

III. Click on any link, the site will show you locations related to the content.

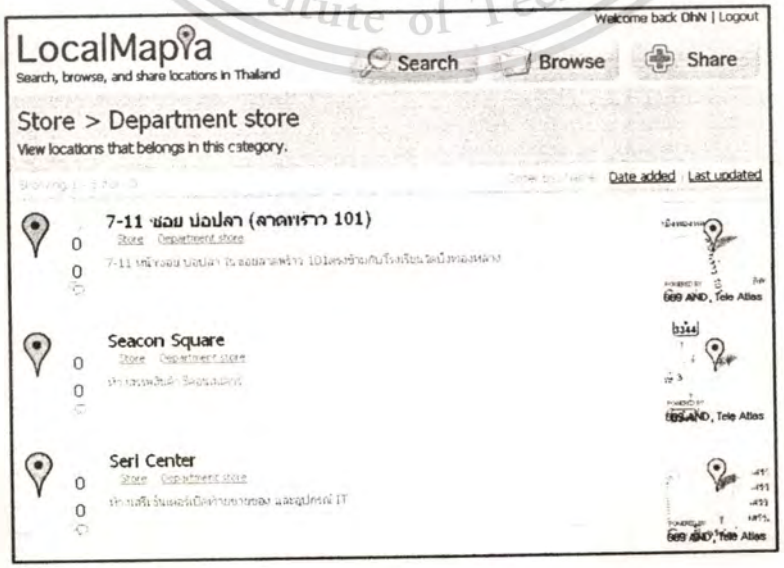

Figure A.10: Browse listing

A.4 Add new Location

I. Click Share

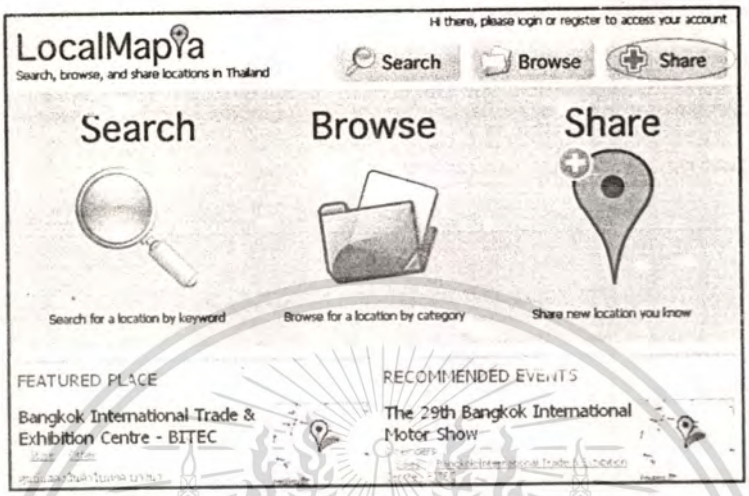

Figure A.11: Share button

Browse through map. Click mark region button to start mark a location. Now draw a grid to cover the area. If you grid doesn't look good, you may click reset button to erase the grid, and then click mark region button again to start over. After you're done with mark region click next.

Figure A.12: Add new location – step 1

II. Fill in location information, then click next.

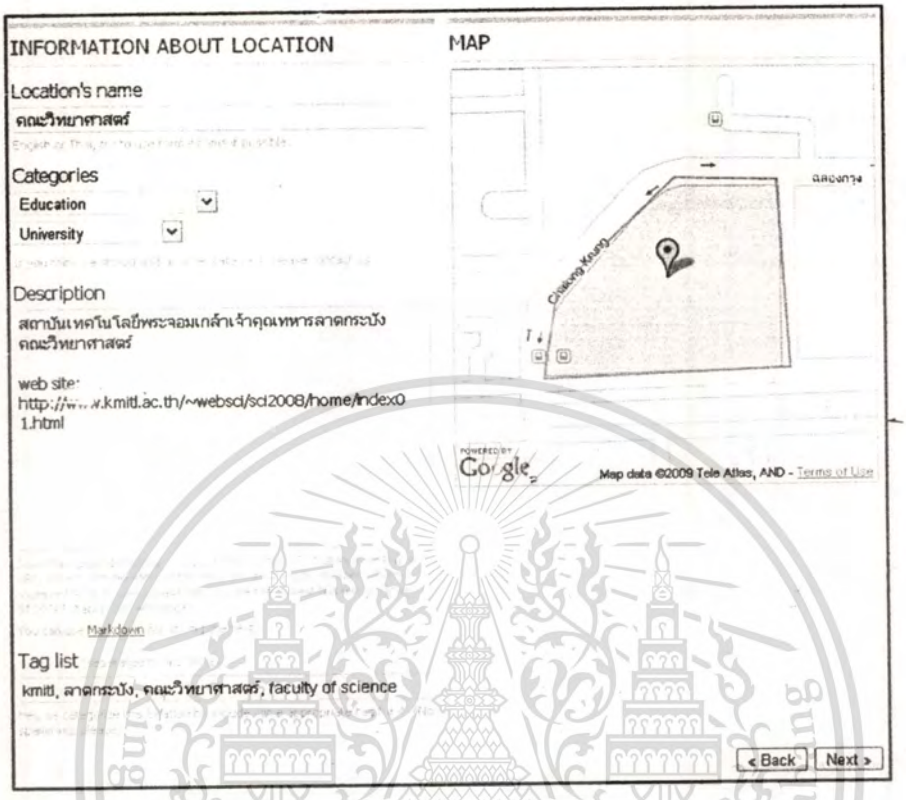

Figure A.13: Add new location – step 2

III. You're successfully added location into web site and will be redirected to the location page.

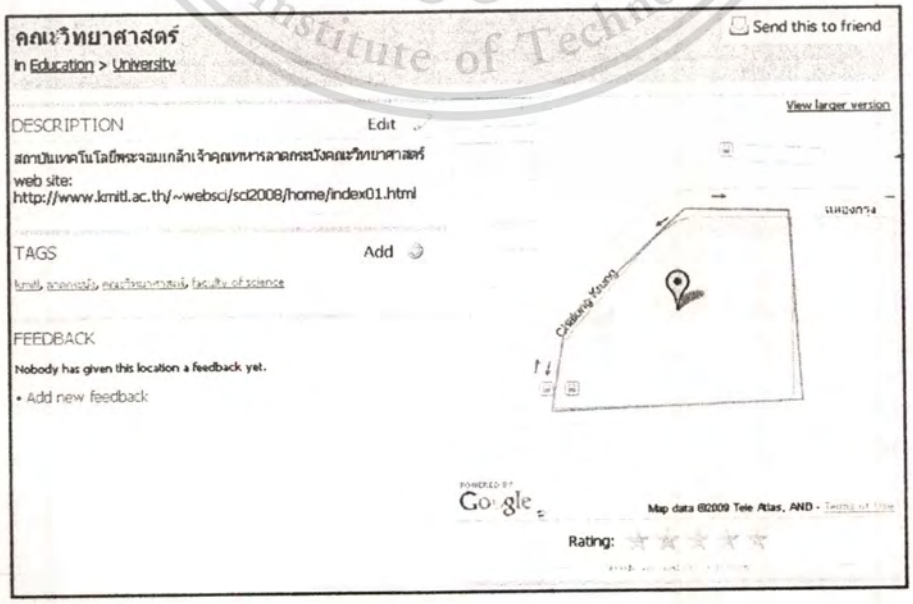

Figure A.14: Location page

A.5 Add Photos

I. From location page, in photo section, clicks Add. The field for enter photo URL will appear.Copy your photo url from Flickr into this field and then press Add button.

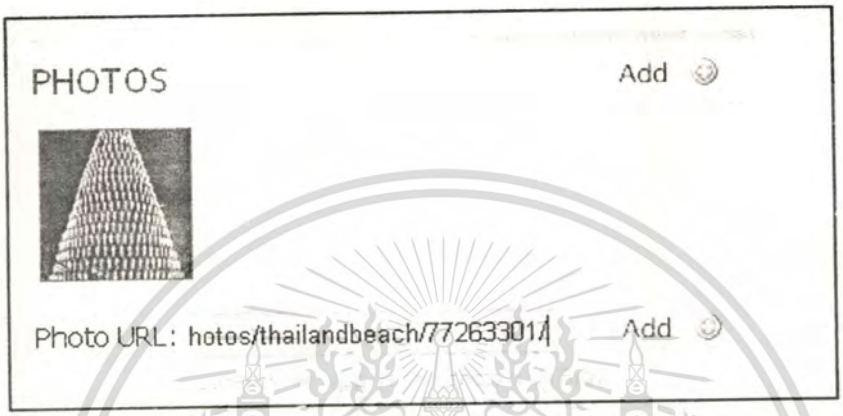

Figure A.15: Add photo link

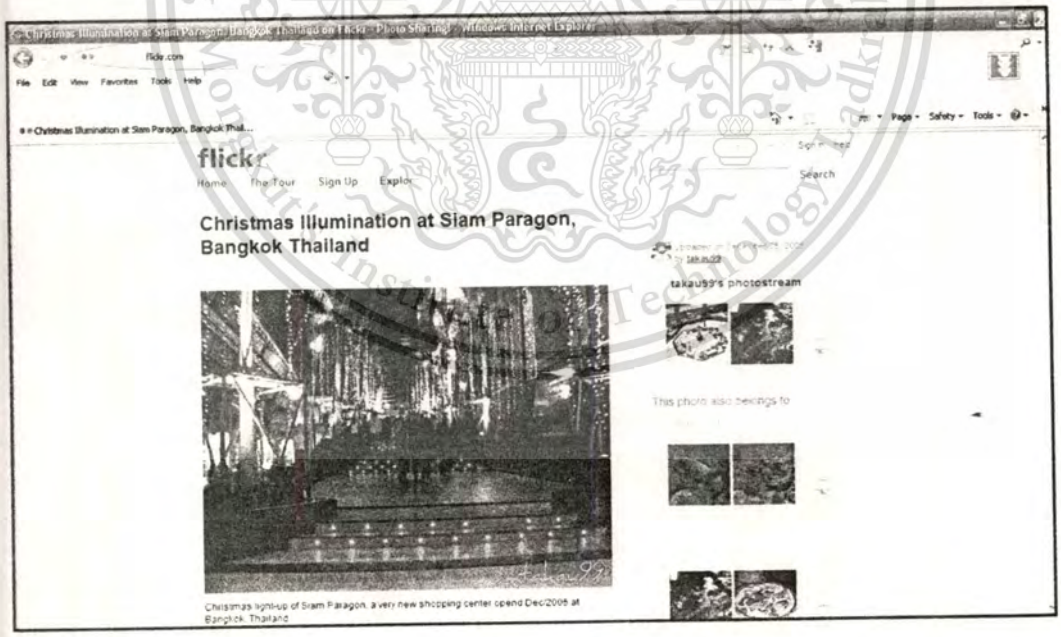

Figure A.16: Photo from Flickr

II. Done, The photo will be displayed in Photo section

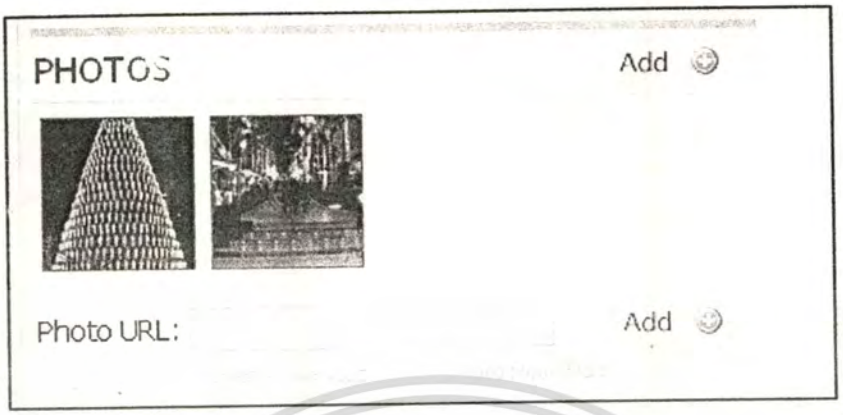

Figure A.17: List of photos including new photo

A.6 Add new Event

I. From location page, click add button in events section

II. Web site will display the adding new event form. Fill in event information, and then press add button.

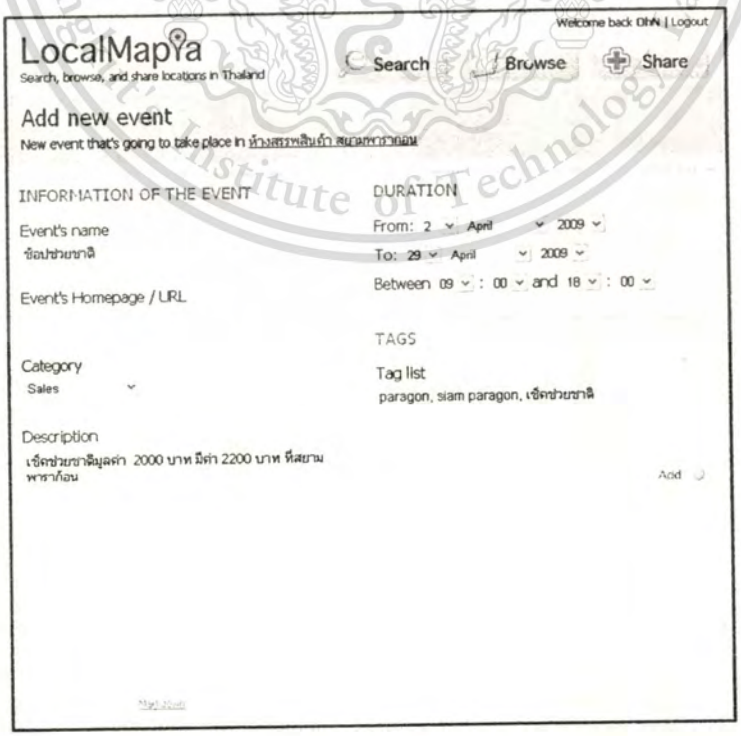

Figure A.18: Add new event form

III. **You'll be redirected to the event page. The new event will be showed in events section of location page.**

A.7 Search

I. From home page, click search

II. Enter key word into search field, and then press search.

III. The result contains locations and events that related to key word.

Figure A.19: Search result

A.8 Sending an event to friend

I. From event page, and then click send this to friend button.

II. Enter friend e-mail, and then click send.

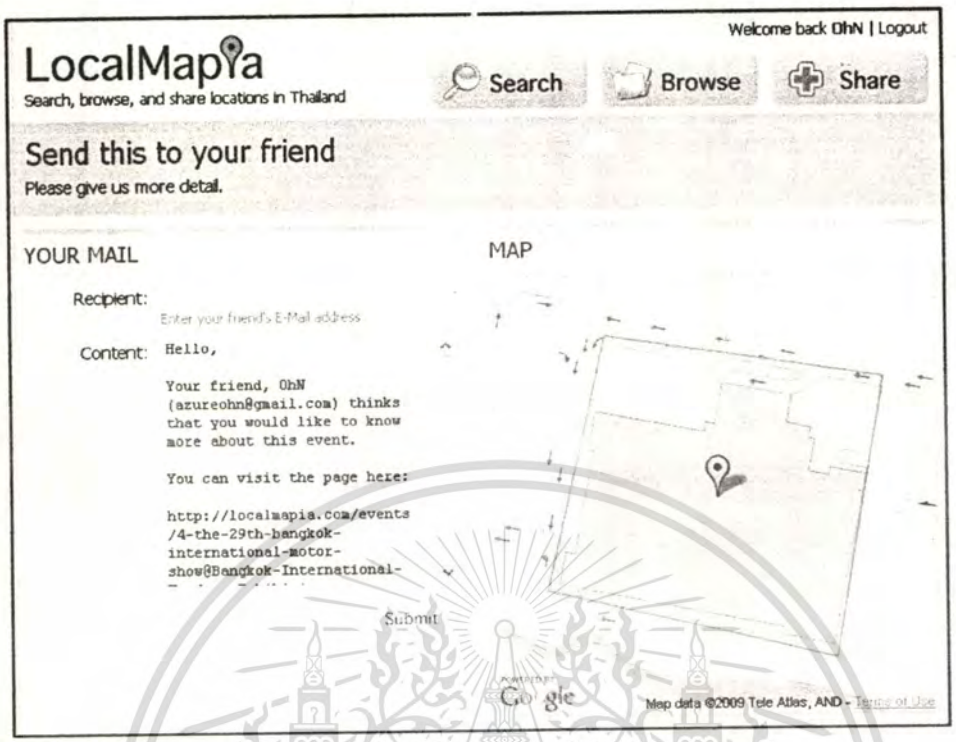

Figure A.20: Mail to friend

A.9 Sending an inappropriate report

I. Click at report inappropriate content link

II. The system will show you a report form. Fill in the issue, then click submit.

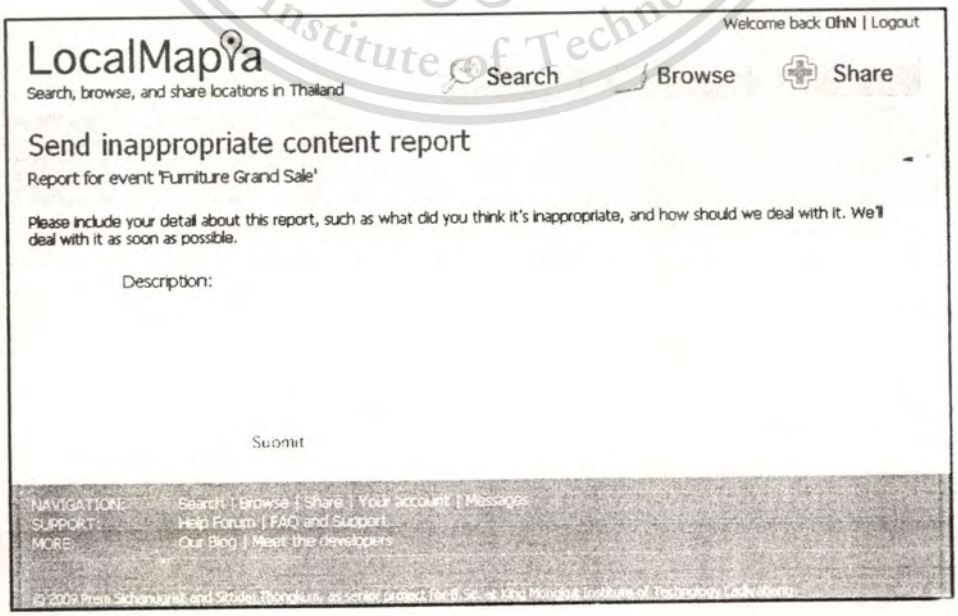

Figure A.21: Inappropriate content report form

III. The system will display the sending result.

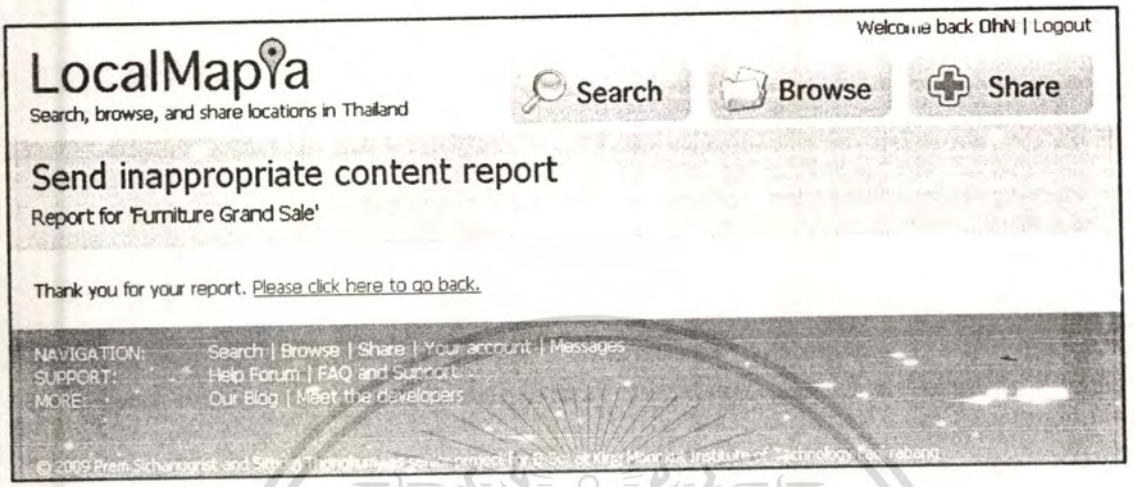

Figure A.22: Result of report inappropriate content

Appendix B

Database System

In Chapter 3, we've mentioned about database in this system and we explained some of tables in detailed already. In this section, we'll continue describe detail of the rest table in our database. However, please note that, the table that has been explained already will not be included in this section.

Table B.1– Categories Table

No.	field name	data type	description
1	parent_id	integer	parent id
2	title	string	category title
3	description	text	description
4	navigation	string	stores parent category
5	photo	string	photo
6	category_type	string	category type
7	category_type	string	category type

Table B.2 – Event_categories Table

No.	field name	data type	description
1	event_id	integer	event id
2	category	integer	category id

Table B.3 – Event_logs table

No.	field name	data type	description
1	id	integer	log id
2	user_id	integer	user id of the log
3	action	string	user action
4	content	text	content
5	ip_address	string	user ip address
6	user_agent	string	Store user's user agent data, such as operating system and browser version
7	created_at	datetime	created date and time
8	updated_at	datetime	updated date and time

Table B.4 – Events_photos table

No.	field name	data type	description
1	event_id	integer	event id
2	title	string	title
3	photo	string	photo url
4	status	string	status
5	created_at	datetime	created date and time
6	updated_at	datetime	updated date and time
7	created_at	datetime	created date and time
8	updated_at	datetime	updated date and time

Table B.5 – Events table

No.	field name	data type	description
1	id	integer	event id
2	location_id	integer	location id
3	name	string	event name
4	description	text	event description
5	url	string	event page url
6	date_start	date	start date
7	date_end	date	end date
8	time_start	time	start time
9	time_end	time	end time
10	status	string	status

Table B.6 – Events_tags table

No.	field name	data type	description
1	event_id	integer	event id
2	tag_id	integer	tag id

Table B.7 – Events_users table

No.	field name	data type	description
1	event_id	integer	event id
2	user_id	integer	user id whose going to participate

Table B.8 – Feedbacks Table

No.	field name	data type	description
1	location_id	integer	location id that the feedback
2	event_id	integer	event id where the feed back
3	user_id	integer	poster id
4	content	text	feedback
5	rank	string	ranking
6	status	string	feedback status
7	created_at	datetime	created date and time
8	updated_at	datetime	updated date and time

Table B.9 – Locations Table

No.	field name	data type	description
1	user_id	integer	user id
2	parent_id	integer	category id
3	title	string	location title
4	description	text	location's description
5	lat	float	latitude point
6	lng	float	longtitude point
7	city	string	city
8	province	string	province
9	country	string	country
10	status	string	record status
11	score	integer	rating
12	vote_count	integer	recommendation vote
13	additional_info	text	additional information
14	created_at	datetime	created date and time
15	updated_at	datetime	updated date and time
16	permalink	string	permalink
17	address	string	location address
18	version	integer	version number

Table B.10 – Locations_regions table

No.	field name	data type	description
1	location_id	integer	location id
2	lat	decimal	latitude point
3	lng	decimal	longtitude point
4	created_at	datetime	created date and time
5	updated_at	datetime	updated date and time

Table B.11 – Locations_tags table

No.	field name	data type	description
1	location_id	integer	location id
2	tag_id	integer	tag id

Table B.12 – Messages table

No.	field name	data type	description
1	from_id	integer	sender id
2	to_id	integer	recipient id
3	title	string	message title
4	content	text	message content
5	status	string	message status
6	created_at	datetime	created date and time
7	updated_at	datetime	updated date and time

Table B.13 – Participations table

No.	field name	data type	description
1	user_id	integer	user id
2	event_id	integer	event id
3	role	string	role of this user in this particular event, will be used in the future to give more permission to event organizers
4	created_at	datetime	created date and time
5	updated_at	datetime	updated date and time

Table B.14 – Photos table

No.	field name	data type	description
1	user_id	integer	owner id
2	location_id	integer	location id where the photo belongs to
3	event_id	integer	event id where the photo belongs to

Table B.11 (cont)

No.	field name	data type	description
6	original_url	string	Store URL to the original photo
7	created_at	datetime	created date and time
8	updated_at	datetime	updated date and time

Table B.15 – Reports table

No.	field name	data type	description
1	user_id	integer	user id
2	object_type	string	Store the type of object that has been reported, such as Location or Event
3	object_id	string	Store the id of object that has been reported
4	content	text	report content
5	created_at	datetime	created date and time
6	updated_at	datetime	updated date and time

Table B.16 – Searches table

No.	field name	data type	description
1	keyword	string	keyword that user input
2	sorted_by_relevence	text	Result set as an array, based on relevence (number of occurance)
3	sorted_by_added	text	Result set as an array, newer comes first
4	sorted_by_updated	text	Result set as an arrry, last updated item comes first
5	created_at	datetime	created date and time
6	updated_at	datetime	updated date and time

Table B.17 – Tags table

No.	field name	data type	description
1	name	string	tag name

Table B.18 – Users_friends table

No.	field name	data type	description
1	user_id	integer	user id
2	friend_id	integer	user's friend id

Table B.19 – Votes table

No.	field name	data type	description
1	user_id	integer	user id
2	votable_id	integer	vote id
3	votable_type	string	vote type
4	created_at	datetime	created date and time
5	updated_at	datetime	updated date and time
6	score	integer	total score

Table B.20 – Users Table

No.	field name	data type	description
1	email	string	user's e-mail
2	encrypted password	string	user's password
3	salt	string	stores hash code for user password encryption
4	displayname	string	user's display name
5	city	string	city where user belongs
6	province	string	province where user belongs
7	country	string	country where user belongs
8	status	string	record status
9	activation key	string	activation key
10	activated_at	datetime	account activated date and time
11	password_reset_key	string	password reset key
12	last_logged_in	date	last logged in
13	created_at	datetime	created date and time
14	updated_at	datetime	updated date and time

Appendix C
Source code

From Chapter 3, we've illustrated the UML diagram already. In this section, we'll show you a source code of this system. However, there's some limitation: please use our source code for educational purpose only.

C.1 Application Controllers

C.1.1 accounts_controller

```
class AccountsController < ApplicationController
  before_filter :check_login, :except => [:new, :create, :status,
:activate]

  def show
  end

  def edit
  end

  def update
  end

  def new
    @u = User.new
  end

  def create
    @u = User.new(params[:user])
    if @u.save
      Mailer.deliver_activation_email(@u)
    else
      @u.password = @u.password_confirmation = ""
      render :new
    end
  end

  def status
    render :status, :layout => false
  end

  def activate
    if @u = User.find_by_activation_key(params[:id])
      @u.activate!
      @session.login! :email => @u.email, :password => @u.password
    end
  end

end
```

Figure C.1: AccountsController class

C.1.2 applications_controller

```
# Filters added to this controller apply to all controllers in the
application.
# Likewise, all the methods added will be available for all
controllers.

class ApplicationController < ActionController::Base
  helper :all # include all helpers, all the time

  include Authentication::Helpers

  # See ActionController::RequestForgeryProtection for details
  # Uncomment the :secret if you're not using the cookie session
store
  protect_from_forgery

  # See ActionController::Base for details
  # Uncomment this to filter the contents of submitted sensitive data
parameters
  # from your application log (in this case, all fields with names
like "password").
  filter_parameter_logging :password, :password_confirmation

  protected

  helper_method :location_permalink, :event_permalink,
:category_permalink, :object_permalink
  def object_permalink(object)
    object.class == Location ? location_permalink(object) :
event_permalink(object)
  end

  def location_permalink(location)
    URI.escape "/locations/#{location.permalink}"
  end

  def event_permalink(event)
    URI.escape "/events/#{event.permalink}"
  end

  def category_permalink(category)
    URI.escape
"/#{category.category_type}s/in/#{category.parent.permalink+"/" if
category.parent}#{category.permalink}"
  end

  helper_method :moderator?, :admin?
  def moderator?
    logged_in? and @user.moderator?
  end

  def admin?
    logged_in? and @user.admin?
  end
end
```

Figure C.2: ApplicationsController class

C.1.3 categories_controller

```
class CategoriesController < ApplicationController
  def index
    @location_categories = Category.first_level.for_location(:include
=> :children, :order => "title")
    @event_categories = Category.for_event(:order => "title")
    @tags = Tag.find_by_sql("SELECT tags.*, (COALESCE(c1.count_all,0)
+ COALESCE(c2.count_all,0)) AS sum_count_all FROM tags LEFT JOIN
(SELECT tag_id, COUNT(*) AS count_all FROM events_tags GROUP BY
tag_id) AS c1 ON c1.tag_id = tags.id LEFT JOIN (SELECT tag_id,
COUNT(*) AS count_all FROM locations_tags GROUP BY tag_id) as c2 ON
c2.tag_id = tags.id ORDER BY sum_count_all DESC LIMIT 30")
    @tag_min_count = @tags.last.sum_count_all.to_i
    @tags.sort!{ |x,y| x.name <=> y.name }
  end

  def show
    prepare_offset_vars
    @category = Category.find_by_permalink!(params[:parent])
    if params[:id].blank? and params[:object_type] == "locations"
      @collection =
@category.send("children_#{params[:object_type]}", {:order => order,
:offset => @offset, :limit => 10})
    elsif params[:id].blank?
      @collection = @category.send("#{params[:object_type]}", {:order
=> order, :offset => @offset, :limit => 10})
    else
      @category = @category.children.find_by_permalink!(params[:id],
:include => :parent)
      @collection = @category.send(params[:object_type]).all(:order
=> order, :offset => @offset, :limit => 10)
    end
  end

  protected

  def prepare_offset_vars
    params[:type] = "title" unless %w(title added
updated).include?(params[:type])
    params[:page] ||= 1

    @offset = (params[:page].to_i - 1) * 10 rescue 0
  end

  def order
    case params[:type]
    when "title"
      params[:object_type] == "events" ? "name" : "title"
    when "added"
      "id DESC"
    when "updated"
      "updated_at DESC"
    end
  end
end
```

Figure C.3: CategoriesController class

C.1.4 events_controller

```
class EventsController < ApplicationController
  before_filter :check_login, :except => [:index, :show]
  before_filter :load_event, :only => [:show, :edit, :update,
:destroy]
  before_filter :find_location

  def new
    @event = @location.events.build
    @categories = Category.first_level.for_event(:order => :title)
  end

  def create
    @event = @location.events.build(params[:event])
    if @event.save
      @user.event_logs.create(:action => "create_event", :content =>
@event.id, :user_agent => request.env["HTTP_USER_AGENT"], :ip_address
=> request.env["X-Real-IP"])
      redirect_to event_permalink(@event)
    else
      ActiveRecord::Base.logger.info @event.errors.inspect
      @categories = Category.first_level.for_event(:order => :title)
      render :new
    end
  end

  def show
  end

  def edit
  end

  def update
    if @event.update_attributes(params[:event])
      flash[:notice] = "Event has been updated successfully."
      @user.event_logs.create(:action => "update_event", :content =>
@event.id, :user_agent => request.env["HTTP_USER_AGENT"], :ip_address
=> request.env["X-Real-IP"])
      redirect_to event_permalink(@event)
    else
      render :edit
    end
  end

  private

  def find_location
    if params[:location_id]
      @location = Location.find(params[:location_id])
    else
      @location = @event.location
    end
  end
```

```
  def load_event
    @event = if params[:id][/^[0-9]+$/]
        Event.find(params[:id])
      else
        Event.find(params[:id].split(/-/,2)[0])
      end
    redirect_to event_permalink(@event), :status => 301 if
params[:id][/^[0-9]+$/] and params[:action] == "show"
  end
end
```

Figure C.4: EventsController class

C.1.5 feedbacks_controller

```
class FeedbacksController < ApplicationController
  before_filter :check_login, :load_commentable

  def create
    unless logged_in?
      flash[:error] = "You need to be logged in first before you can
post a feedback."
    else
      @feedback =
@commentable.feedbacks.build(params[:feedback].merge({:user_id =>
@user.id}))
      if @feedback.save
        flash[:notice] = "Your feedback has been added"
      end
    end
    redirect_to params[:event_id] ? event_permalink(@commentable) :
location_permalink(@commentable)
  end

  private

  def load_commentable
    @commentable = if params[:event_id]
        Event.find(params[:event_id])
      else
        Location.find(params[:location_id])
      end
  end
end
```

Figure C.5: FeedbacksController class

C.1.6 locations_controller

```
class LocationsController < ApplicationController
  before_filter :check_for_login, :only => [:step1, :step2, :create,
```

```
:edit, :update, :destroy]
  before_filter :load_location, :only => [:show, :edit, :update,
:destroy]

  def index
    redirect_to browse_path
  end

  def new
    redirect_to :step1_new_location
  end

  def step1
    @location = Location.new
  end

  def step2
    redirect_to :action => "step1" unless session[:location]
    @location = session[:location]
    @first_level_categories = Category.first_level.for_location
    @second_level_categories =
Category.second_level(@first_level_categories.first.id).for_location
rescue []
  end

  def create
    if params[:location][:latlng].present? # From step 1
      session[:location] =
Location.new(params[:location].merge(:user_id => @user.id))
      redirect_to :action => "step2"
    else
      session[:location].attributes = params[:location]
      if session[:location].save
        @location = session[:location]
        session[:location] = nil
        @user.event_logs.create(:action => "create_location",
:content => @location.id, :user_agent =>
request.env["HTTP_USER_AGENT"], :ip_address => request.env["X-Real-
IP"])
        redirect_to location_permalink(@location)
      else
        flash[:error] = "There's something wrong with the data you
entered. Please fix them and submit it again."
        redirect_to :action => "step2"
      end
    end
  end

  def show
  end

  def edit
  end

  def update
    if @location.update_attributes(params[:location])
      flash[:notice] = "Location has been updated successfully."
      @user.event_logs.create(:action => "update_location", :content
=> @location.id, :user_agent => request.env["HTTP_USER_AGENT"],
:ip_address => request.env["X-Real-IP"])
```

```
      redirect_to location_permalink(@location)
    else
      render :edit
    end
  end

  def destroy
  end

  def categories
    @categories = if params[:parent_id].blank?
        Category.for_location.first_level
      else
        Category.for_location.second_level(params[:parent_id])
      end
    render :layout => false
  end

  private

  def check_for_login
    unless logged_in?
      flash[:notice] = "You're just one step away from adding new
location. Please login or register below:"
      redirect_to login_path
    end
  end

  def load_location
    @location = if params[:id][/^[0-9]+$/]
        Location.find(params[:id])
      else
        Location.find_by_permalink!(params[:id])
      end
    redirect_to location_permalink(@location), :status => 301 if
params[:id][/^[0-9]+$/] and params[:action] == "show"
  end

end
```

Figure C.6: LocationsController class

C.1.7 mails_controller

```
class MailsController < ApplicationController
  before_filter :check_login, :load_mailable

  def new
    @mail = Mail.new
    @mail.content = "Hello,\n\nYour friend, #{@user.displayname}
(#{@user.email}) thinks that you would like to know more about this
#{@mailable.class.to_s.downcase}.\n\nYou can visit the page
here:\n\nhttp://localmapia.com#{object_permalink(@mailable)}\n\n\nTha
nk you,\n\nLocalMapia Team"
  end
```

```
  def create
    @mail = Mail.new
    @mail.content = params[:mail][:content]
    @mail.recipient = params[:mail][:recipient]
    Mailer.deliver_invite_email(@user, @mail)

    flash[:notice] = "Your emai has been sent!"
    redirect_to object_permalink(@mailable)
  end

  private

  def load_mailable
    @mailable = if params[:event_id]
        Event.find(params[:event_id])
      else
        Location.find(params[:location_id])
      end
    @location = @mailable.location rescue @mailable
  end
end
```

Figure C.7: MailsController class

C.1.8 participations_controller

```
class ParticipationsController < ApplicationController
  before_filter :find_event

  def create
    if logged_in?
      if @participation =
@event.participations.find_by_user_id(@user.id)
        @participation.destroy
      else
        @event.participations.create(:user_id => @user.id)
      end
      respond_to do |wants|
        wants.js
      end
    else
      render :text => "You must be logged in first!", :status => 500
unless logged_in?
    end
  end

  private

  def find_event
    @event = Event.find(params[:event_id])
  end
end
```

Figure C.8: ParticipationsContorller class

C.1.9 photos_controller

```
class PhotosController < ApplicationController
  before_filter :check_login, :find_attachable

  def create
    @photo =
@attachable.photos.build(params[:photo].merge(:album_type =>
"flickr"))
    @photo.user = @user

    respond_to do |wants|
      wants.js do
        unless @photo.save
          render :text => "alert('URL you entered
#{@photo.errors.on(:original_url)}');
$('#photo_original_url').val('');"
        end
      end
    end
  end

  private
  def find_attachable
    @attachable = if params[:event_id]
        Event.find(params[:event_id])
      else
        Location.find(params[:location_id])
      end
  end
end
```

Figure C.9: PhotosController class

C.1.10 public_controller

```
class PublicController < ApplicationController
  def index
    @featured_places = Location.featured
    @featured_events = Event.featured.active
    @recently_added = Location.recently_added
    @recently_updated = Location.recently_updated
    @top_contributers = EventLog.top_contributors
  end
end
```

Figure C.10: PublicController class

C.1.11 reports_controller


```
class ReportsController < ApplicationController
```

```
  def new
    if %w(event location).include? params[:type]
      @object = params[:type] == "event" ? Event.find(params[:id]) :
Location.find(params[:id])
      @report = Report.new(:object => @object)
    else
      raise ActiveRecord::RecordNotFound, "Unable to find an object
you want to report"
    end
  end

  def create
    @report = @user.reports.build(params[:report])
    unless @report.save
      render :new
    end
  end
end
```

Figure C.11: ReportsController class

C.1.12 searchs_controller

```
class SearchesController < ApplicationController
  before_filter :process_search_vars, :only => :show

  def index
    @search = Search.new
  end

  def create
    @search = Search.new(:keyword => params[:search][:keyword])
    params[:search][:keyword].strip!
    if params[:search][:keyword].present? and
params[:search][:keyword].mb_chars.size >= 2
      redirect_to search_path(@search.permalink) if @search.process!
    end
  end

  def show
    @search = Search.find_by_permalink(params[:id])
    @search.load_objects!("sorted_by_#{params[:type]}", @offset)
  end

  protected

  def process_search_vars
    params[:type] = "relevance" unless %w(relevance added
updated).include?(params[:type])
    params[:page] ||= 1

    @offset = (params[:page].to_i - 1) * 10 rescue 0
  end
end
```


Figure C.12: SearchesController class

C.1.13 sessions_controller

```
class SessionsController < ApplicationController
  include Authentication::Controller
  before_filter :load_new_user, :only => :new

  private

  def load_new_user
    @u = User.new
  end
end
```

Figure C.13: SessionsController class

C.1.14 tags_controller

```
class TagsController < ApplicationController
  before_filter :load_taggable, :only => :create
  before_filter :check_login, :only => :create

  def create
    Tag.batch_create(params[:tag], @taggable)
    flash[:notice] = "Your tags has been added"
    redirect_to params[:event_id] ? event_permalink(@taggable) :
location_permalink(@taggable)
  end

  def show
    prepare_offset_vars
    @tag = Tag.find_by_name!(params[:id])
    @collection_count = @tag.locations.size + @tag.events.size
    @collection = (@tag.locations + @tag.events).sort{ |x,y|
order(x,y) }[@offset, 10]
  end

  private

  def load_taggable
    @taggable = if params[:event_id]
        Event.find(params[:event_id])
      else
        Location.find(params[:location_id])
      end
  end

  def prepare_offset_vars
    params[:type] = "title" unless %w(title added
updated).include?(params[:type])
    params[:page] ||= 1
    @offset = (params[:page].to_i - 1) * 10 rescue 0
```

```
  end

  def order(x,y)
    case params[:type]
    when "title"
      (x.respond_to?(:title) ? x.title : x.name) <=>
(y.respond_to?(:title) ? y.title : y.name)
    when "added"
      (y.id <=> x.id)
    when "updated"
      (y.updated_at <=> x.updated_at)
    end
  end
end
```

Figure C.14: TagsController class

C.1.15 votes_controller

```
class VotesController < ApplicationController
  before_filter :load_votable
  def create
    if logged_in?
      @votable.votes.create(:user_id => @user.id, :score =>
params[:score].to_f / 10)
      respond_to do |wants|
        wants.js
      end
    else
      render :text => "You must be logged in first!", :status => 500
unless logged_in?
    end
  end
  private
  def load_votable
    @votable = if params[:event_id]
        Event.find(params[:event_id])
      else
        Location.find(params[:location_id])
      end
  end
end
```

Figure C.15: VotesController class

C.1.16 watches_controller

```
class WatchersController < ApplicationController
  before_filter :find_event

  def create
    if logged_in?
      if @event.watcher_ids.include? @user.id
        @event.watchers.delete(@user)
```

```
      else
        @event.watchers << @user
      end
      respond_to do |wants|
        wants.js
      end
    else
      render :text => "You must be logged in first!", :status => 500
unless logged_in?
    end
  end

  private

  def find_event
    @event = Event.find(params[:event_id])
  end
end
```

Figure C.16: WatchesController

C.2 Model

C.2.1 category

```
      class Category < ActiveRecord::Base
  has_many :category_fields
  belongs_to :parent, :class_name => "Category"
  has_many :children, :class_name => "Category", :foreign_key =>
'parent_id'
  has_and_belongs_to_many :locations, :join_table =>
"locations_categories"
  has_and_belongs_to_many :events, :join_table => "events_categories"

  CATEGORY_TYPE = %w(location event)
  serialize :navigation

  validates_presence_of :title
  validates_inclusion_of :category_type, :in => CATEGORY_TYPE
  before_save :calculate_navigation, :set_permalink

  named_scope :for_location, :conditions => { :category_type =>
"location" }
  named_scope :for_event, :conditions => { :category_type => "event"
}
  named_scope :first_level, :conditions => { :parent_id => nil }
  named_scope :second_level, lambda { |parent_id| { :conditions => {
:parent_id => parent_id }}}

  private

  def children_locations(opts={})
    Location.all({:select => "locations.*", :from =>
"locations_categories", :joins => "JOIN locations ON
locations_categories.location_id = locations.id", :conditions =>
```

```
"locations_categories.category_id IN (SELECT id FROM categories WHERE
parent_id = #{id})"}.merge(opts))
  end

  def validate
    if self.new_record? # Validate on create only
      self.errors.add(:title, "should be unique (#{{:title =>
self.title, :category_type => self.category_type, :parent_id =>
self.parent_id}.inspect})") if Category.count(:conditions => {:title
=> self.title, :category_type => self.category_type, :parent_id =>
self.parent_id}) > 0
    end
  end

  def calculate_navigation
    self.navigation = [self.title]
    object = self
    if object.parent
      self.navigation.unshift object.parent.title
    end
  end

  def set_permalink
    self.permalink = title.downcase.gsub(/(\/| )+/, '-')
  end
end
```

Figure C.17: Category class

C.2.2 category_field

```
class CategoryField < ActiveRecord::Base
  belongs_to :category

  FIELD_TYPE = %w(text textarea select checkbox date time datetime)
  serialize :collection

  validates_presence_of :name
  validates_inclusion_of :field_type, :in => FIELD_TYPE
end
```

Figure C.18: CategoryField class

C.2.3 event

```
class Event < ActiveRecord::Base
  has_many :feedbacks
  has_many :photos
  has_many :event_photos
  has_many :participations
  has_many :participants, :through => :participations, :source =>
:user
```

```
  has_many :votes, :as => :votable
  belongs_to :location
  has_and_belongs_to_many :tags
  has_and_belongs_to_many :categories, :join_table =>
:events_categories
  has_and_belongs_to_many :watchers, :class_name => "User"

  STATUS = %w(deleted hidden flagged normal featured)
  before_validation_on_create :set_default_status
  after_create :set_permalink
  acts_as_versioned

  validates_presence_of :name
  validates_inclusion_of :status, :in => STATUS
  validates_presence_of :date_start, :date_end
  validates_format_of :url, :with => /^http(s|):\/\//, :allow_blank
=> true
  validates_presence_of :location

  default_scope :conditions => {:status => STATUS[2..-1]}
  named_scope :active, :conditions => ["date_end > ?", Date.today],
:order => "date_start DESC"
  named_scope :featured, :conditions => {:status => "featured"},
:order => "created_at DESC", :limit => 3

  def tag_list=(str)
    self.tags.clear
    str.split(/,/).each do |tag_name|
      self.tags << Tag.find_or_create_by_name(tag_name.strip)
    end
  end

  def tag_list
    @tag_list ||= tags.collect{ |t| t.name }.join(", ")
  end

  # monkeypatch - make it acts like a location
  def title
    name
  end

  def time_start
    self[:time_start] ||= Time.parse("9:00")
  end

  def time_end
    self[:time_end] ||= Time.parse("18:00")
  end

  # Temporary - We'll use a single category for a while
  def category_id=(category_id)
    return if self.category_ids.include? category_id
    self.category_ids = [category_id]
  end

  def category_id
    self.category_ids.first
  end

  def category
```

```
    self.categories.first
  end

  def stars
    unless @stars
      stars = (score * 10 / votes.count)
      puts stars.to_s[-1]
      @stars = case stars.to_s[-1]
        when 49..53; stars.to_i / 10 * 10 + 5
        when 54..57; stars.to_i / 10 * 10 + 10
        when 48; stars.to_i
        else 0
        end
    end
    @stars
  end

  def score
    @score ||= votes.sum(:score)
  end

  def already_voted?(user)
    votes.exists?(:user_id => user.id)
  end

  def duration
    date = if date_start.year == date_end.year
        if date_start.month == date_end.month
          "#{date_start.strftime("%B %d")} - #{date_end.strftime("%d,
%Y")}."
        else
          "#{date_start.strftime("%B %d")} - #{date_end.strftime("%B
%d, %Y")}."
        end
      else
        "#{date_start.strftime("%B %d, %Y")} -
#{date_end.strftime("%B %d, %Y")}."
      end
    time = formatted_time(time_start) + " until " +
formatted_time(time_end)

    "#{date} #{time}"
  end

  def flagged?
    status == "flagged"
  end

  private

  def set_permalink
    self.permalink = "#{self.id}-#{self.name.downcase.gsub(/ /, "-
")}@#{location.title.gsub(/ /, "-")}"
    self.save
  end

  def formatted_time(time)
    if time.hour.zero? and time.minute.zero?
      "Midnight"
    elsif time.hour == "12" and time.minute.zero?
```

```
      "Noon"
    else
      time.strftime("%I:%M %p")
    end
  end

  def set_default_status
    self.status = "normal"
  end

  def validate_on_create
    errors.add(:category, "should not be blank") if
self.categories.size.zero?
    errors.add(:date_end, "should come after start date") if
self.date_start and self.date_end and self.date_start.to_date >
self.date_end.to_date
    errors.add(:time_end, "should come after start time") if
self.time_start and self.time_end and self.time_start > self.time_end
  end
end
```

Figure C.19: Event class

C.2.4 event_log

```
class EventLog < ActiveRecord::Base
  belongs_to :user

  validates_presence_of :user
  validates_presence_of :action

  def self.top_contributors
    all(:conditions => {:action => %w(create_location update_location
create_event update_event)}, :group => "user_id", :select =>
"COUNT(DISTINCT action, content) AS count_all, event_logs.*", :order
=> "count_all DESC", :limit => 5)
  end
end
```

Figure C.20: EventLog class

C.2.5 event_photo

```
class EventPhoto < ActiveRecord::Base
  belongs_to :event

  STATUS = %w(deleted hidden normal)
  before_validation :set_default_status

  validates_presence_of :title
  validates_presence_of :event
  validates_inclusion_of :status, :in => STATUS
```

```
  private

  def set_default_status
    self.status ||= "normal"
  end
end
```

Figure C.21: EventPhoto class

C.2.6 feedback

```
class Feedback < ActiveRecord::Base
  belongs_to :location
  belongs_to :event
  belongs_to :user

  STATUS = %w(deleted normal)
  before_create :set_default_rank
  before_validation :set_default_status

  validates_presence_of :user
  validates_presence_of :content
  validates_inclusion_of :status, :in => STATUS

  private

  def set_default_rank
    self.rank = 0
  end

  def set_default_status
    self.status ||= "normal"
  end

  def validate
    self.errors.add(:location, "or event should be selected") if
self.location.nil? and self.event.nil?
  end
end
```

Figure C.22: Feedback class

C.2.7 location

```
class Location < ActiveRecord::Base
  has_many :events, :dependent => :destroy
  has_many :feedbacks, :dependent => :destroy
  has_many :photos, :dependent => :destroy
  has_many :regions, :class_name => "LocationRegion", :dependent =>
:destroy
  has_many :children, :class_name => "Location", :foreign_key =>
```

```
:parent_id
  has_many :votes, :as => :votable
  belongs_to :user
  belongs_to :parent, :class_name => "Location"
  has_and_belongs_to_many :categories, :join_table =>
"locations_categories"
  has_and_belongs_to_many :tags
  accepts_nested_attributes_for :categories
  acts_as_versioned

  STATUS = %w(hidden flagged normal featured)
  before_validation_on_create :set_default_status
  before_validation_on_create :fetch_location_information
  after_create :set_permalink

  validates_presence_of :user
  validates_presence_of :title
  validates_presence_of :description
  validates_presence_of :lat, :lng, :country
  validates_inclusion_of :status, :in => STATUS

  default_scope :conditions => {:status => STATUS[1..-1]}
  named_scope :featured, :conditions => "status = 'featured'", :order
=> "created_at DESC", :limit => 3
  named_scope :recently_added, :limit => 5, :order => "created_at
DESC"
  named_scope :recently_updated, :limit => 5, :order => "updated_at
DESC"

  def tag_list=(str)
    self.tags.clear
    str.split(/,/).each do |tag_name|
      self.tags << Tag.find_or_create_by_name(tag_name.strip)
    end
  end

  def tag_list
    @tag_list ||= tags.collect{ |t| t.name }.join(", ")
  end

  def latlng=(latlngs)
    lat, lng = latlngs.first.split(/\|/)
    min_lat = max_lat = lat.to_f
    min_lng = max_lng = lng.to_f

    latlngs.each do |latlng|
      lat, lng = latlng.split(/\|/)
      self.regions.build(:lat => lat.to_f, :lng => lng.to_f)

      # cache it
      min_lat = lat.to_f if lat.to_f < min_lat
      min_lng = lng.to_f if lng.to_f < min_lng
      max_lat = lat.to_f if lat.to_f > max_lat
      max_lng = lng.to_f if lng.to_f > max_lng
    end
    self.lat = (min_lat + max_lat) / 2.0
    self.lng = (min_lng + max_lng) / 2.0
  end

  def regions_for_google
    @regions_for_google = self.regions.collect{ |r| "new
```

```
GLatLng(#{r.lat}, #{r.lng})" }.join(",") + ",new
GLatLng(#{self.regions.first.lat}, #{self.regions.first.lng})"
  end

  # Temporary - We'll use a single category for a while
  def category_id=(category_id)
    return if self.category_ids.include? category_id
    self.category_ids = [category_id]
  end

  def category_id
    self.category_ids.first
  end

  def category
    self.categories.first
  end

  def thumbnail
    @thumbnail ||=
"http://maps.google.com/staticmap?center=#{lat},#{lng}&zoom=15&size=1
00x100&markers=#{lat},#{lng}&key=#{GOOGLE_API_KEY}&sensor=false"
  end

  def flagged?
    status == "flagged"
  end

  def stars
    unless @stars
      stars = (score * 10 / votes.count)
      puts stars.to_s[-1]
      @stars = case stars.to_s[-1]
        when 49..53; stars.to_i / 10 * 10 + 5
        when 54..57; stars.to_i / 10 * 10 + 10
        when 48; stars.to_i
        else 0
        end
    end
    @stars
  end

  def score
    @score ||= votes.sum(:score)
  end

  def already_voted?(user)
    votes.exists?(:user_id => user.id)
  end

  private

  def fetch_location_information
    logger.info "=> Request reverse geocoding data from Google ..."
    response = `curl "http://maps.google.com/maps/geo?q=#{ self.lat
},#{ self.lng
}&output=csv&oe=utf8&sensor=false&key=#{GOOGLE_API_KEY}&hl=en"`
    if response.present?
      status, accuracy, address = response.split(',', 3)
      if status == "200"
```

```
        logger.info "** #{address}"
        self.country, self.province, self.city = address[1..-
2].split(/, /).reverse
        self.address = address
      end
    end
    self.country ||= "Thailand"
    self.province ||= "Bangkok"
  end

  def set_default_status
    self.status = "normal"
  end

  def validate
    errors.add(:category, "should not be blank") if
self.categories.size.zero?
  end

  def set_permalink
    self.permalink = "#{self.id}-#{self.title.downcase.gsub(/ /, "-
")}"
    self.save
  end
end
```

Figure C.23: Location class

C.2.8 location_region

```
class LocationRegion < ActiveRecord::Base
  belongs_to :location
end
```

Figure C.24: LocationRegion class

C.2.9 mail

```
class Mail
  attr_accessor :recipient, :content

  def valid?
    true
  end

  def errors
    []
  end
  def new_record?
    true
  end
end
```

Figure C.25: Mail class

C.2.10 mailer

```
class Mailer < ActionMailer::Base
  def activation_email(user)
    from "LocalMapia <noreply@localmapia.com>"
    recipients user.email
    subject "[LocalMapia] Your activation instructions"
    body :user => user
  end
  def invite_email(user, email)
    from "LocalMapia <noreply@localmapia.com>"
    recipients email.recipient
    subject "[LocalMapia] You've been invited!"
    body :content => email.content
  end
end
```

Figure C.26: Mailer class

C.2.11 message

```
class Message < ActiveRecord::Base
  belongs_to :from, :class_name => "User", :foreign_key => "from_id"
  belongs_to :to, :class_name => "User", :foreign_key => "to_id"

  STATUS = %w(deleted unread read)
  before_validation :set_default_status

  validates_presence_of :title
  validates_presence_of :content
  validates_presence_of :from, :to
  validates_inclusion_of :status, :in => STATUS

  private
  def set_default_status
    self.status ||= "unread"
  end
end
```

Figure C.27: Message class

C.2.12 participation

```
class Participation < ActiveRecord::Base
  belongs_to :user
  belongs_to :event
```

```
  ROLE = %w(normal organizer)
  before_validation :set_default_role

  validates_presence_of :user
  validates_presence_of :event
  validates_inclusion_of :role, :in => ROLE

  private
  def set_default_role
    self.role ||= "normal"
  end
end
```

Figure C.28: Participation class

C.2.13 photo

```
require 'open-uri'
require 'hpricot'

class Photo < ActiveRecord::Base
  belongs_to :user
  belongs_to :location
  belongs_to :event

  ALBUM_TYPE = %w(flickr myspace picasa hi5)

  validates_presence_of :user
  validates_presence_of :photo_url
  validates_format_of :original_url, :with => /http(s|):\/\//
  before_validation_on_create :set_photo_url

  private

  def validate
    self.errors.add(:location, "or event should be selected") if
self.location_id.nil? and self.event_id.nil?
  end

  def validate_on_create
    case album_type
    when "flickr"
      self.errors.add(:original_url, "is invalid") unless
self.original_url =~ /flickr.com\/photos\/(.+)\/(.+)\/$/
    else
      self.errors.add(:original_url, "is not supported") unless
self.errors.on(:original_url)
    end
  end

  def set_photo_url
    case album_type
    when "flickr"
      photo_id = original_url.match(/photos\/(.+)\/(.+)\/$/)[2]
      api_url =
```

```
"http://api.flickr.com/services/rest/?method=flickr.photos.getSizes&a
pi_key=351c014287d91f20d02c0e9635883f30&photo_id=#{photo_id}"
      logger.info "=> Requesting photo sizes from flickr"
      doc = Hpricot.parse(open(api_url).read)
      self.photo_url = (doc % "size:first")[:source]
    end
  rescue
    self.errors.add(:original_url, "is not supported") unless
self.errors.on(:original_url)
  end
end
```

Figure C.29: Photo class

C.2.14 report

```
class Report < ActiveRecord::Base
  belongs_to :user
  belongs_to :object, :polymorphic => true

  validates_presence_of :user
  validates_presence_of :object
end
```

Figure C.30: Report class

C.2.15 search

```
require 'digest'

class Search < ActiveRecord::Base
  serialize :sorted_by_relevance
  serialize :sorted_by_added
  serialize :sorted_by_updated

  def process!
    # Find each collection, then stores it to global array using
merge_result_data
    merge_result_data :location, Location.all(:conditions => ["title
LIKE ? OR description LIKE ?", "%#{keyword}%", "%#{keyword}%"])
    merge_result_data :event, Event.all(:conditions => ["name LIKE ?
OR description LIKE ?", "%#{keyword}%", "%#{keyword}%"])

    if tag = Tag.find_by_name(keyword, :include => [:locations,
:events])
      merge_result_data :location, tag.locations
      merge_result_data :event, tag.events
    end

    # Do not save data if we can't find any record
    return false if result_objects.empty?
```

```
    # Sort the object ids based on criteria
    self.sorted_by_relevance = result_object_ids.sort do |y,x| # Max
-> Min

      (count_occurance(keyword, fetch_object(x).title) +
count_occurance(keyword, fetch_object(x).description)) <=>
(count_occurance(keyword, fetch_object(y).title) +
count_occurance(keyword, fetch_object(y).description))
    end
    self.sorted_by_added = result_object_ids.sort do |y,x|
      fetch_object(x).created_at <=> fetch_object(y).created_at
    end
    self.sorted_by_updated = result_object_ids.sort do |y,x|
      fetch_object(x).updated_at <=> fetch_object(y).updated_at
    end

    # Done! Now save the data
    self.permalink =
Digest::SHA1.hexdigest("#{rand(2**100)}#{Time.now.to_s}") # Avoid
collision
    self.save!
  end

  def fetch_object(id)
    result_objects.assoc(id)[1] rescue nil
  end

  def load_objects!(type, offset=0, limit=10)
    # construct array for using with search
    location_ids, event_ids = [], []
    self.send(type).each do |object_type_and_id|
      object_type, object_id = object_type_and_id.split(/-/)
      if object_type == "location"
        location_ids << object_id
      else
        event_ids << object_id
      end
    end

    merge_result_data :location,
Location.find_all_by_id(location_ids)
    merge_result_data :event, Event.find_all_by_id(event_ids)
  end

  private

  def merge_result_data(type, objects)
    self.result_objects = result_objects | objects.collect{ |obj|
["#{type}-#{obj.id}", obj] }
  end

  def result_objects
    @result_objects ||= []
  end

  def result_objects=(args)
    @result_objects = args
  end

  def result_object_ids
```

```
    result_objects.collect{ |r| r.first }
  end

  def count_occurance(needle, haystack)
    count = 0
    haystack = haystack.clone
    while haystack[needle]
      haystack.sub!(needle, '')
      count += 1
    end
    count
  end
end
```

Figure C.31: Search class

C.2.16 tag

```
class Tag < ActiveRecord::Base
  has_and_belongs_to_many :locations
  has_and_belongs_to_many :events

  validates_presence_of :name

  def self.batch_create(params, taggable, separator=",")
    ActiveRecord::Base.transaction do
      taggable_tag_ids = taggable.tag_ids
      params[:name].split(separator).each do |tag|
        tag_object = Tag.find_or_create_by_name(:name => tag.strip)
        taggable.tags << tag_object unless taggable_tag_ids.include?
tag_object.id
      end
    end
  end
end
```

Figure C.32: Tag class

C.2.17 user

```
class User < ActiveRecord::Base
  include Authentication::Model

  has_many :feedbacks
  has_many :participations
  has_many :participate_events, :through => :participations, :source
=> :event
  has_many :messages, :foreign_key => :to_id
  has_many :sent_messages, :foreign_key => :from_id
  has_many :reports
  has_many :event_logs
  has_many :locations
```

```
  has_many :photos
  has_and_belongs_to_many :friends, :class_name => "User",
:association_foreign_key => :friend
  has_and_belongs_to_many :watched_events, :class_name => "Event"

  attr_encrypted :password, :key => :salt, :encode => true
  attr_accessor :password_confirmation
  STATUS = %w(banned monitored normal moderator admin)

  validates_uniqueness_of :email, :message => "has already been used"
  validates_format_of :email, :with => /(.+?)@(.+?)\.(.+)?/
  validates_uniqueness_of :displayname
  validates_presence_of :displayname
  validates_length_of :password, :in => 6..16

  before_create :set_default_status, :generate_activation_key!

  named_scope :online, :conditions => ["status NOT IN (?)",
[ not_activated", "banned"]]

  def salt
    self[:salt] ||=
Digest::SHA2.new(512).hexdigest(rand(2**1000).to_s)
  end

  def activate!
    self.activation_key = nil
    self.activated_at = Time.now
    self.status = "normal"
    self.save
  end

  def update_login!
    self.last_logged_in = Time.now
    self.save
  end

  def displayname=(displayname)
    self[:displayname] = displayname.strip
  end

  def reset_password!(reset_key = nil)
    if reset_key.nil?
      update_attributes(:password_reset_key =>
Digest::SHA1.hexdigest(rand(2**1000).to_s+salt))
      return password_reset_key
    elsif reset_key == password_reset_key
      self[:password_reset_key] = nil
      generate_random_password!
      return true
    else
      return false
    end
  end

  def admin?
    status == "admin"
  end

  def moderator?
```

```
    admin? || status == "moderator"
  end

  private

  def set_default_status
    self[:status] = "not_activated"
  end

  def generate_activation_key!
    self[:activation_key] =
Digest::SHA1.hexdigest(rand(2**1000).to_s+salt)
  end

  def generate_random_password!
    self.password = self.password_confirmation =
Digest::SHA1.hexdigest(rand(2**1000).to_s+salt)[0...8]
    save!
  end

  def validate
    self.errors.add(:password_confirmation, "is not match") if
encrypted_password_changed? and self.password !=
self.password_confirmation
  end
end
```

Figure C.33: User class

C.2.18 vote

```
class Vote < ActiveRecord::Base
  belongs_to :votable, :polymorphic => true
end
```

Figure C.34: Vote class

Appendix D

Google Maps API Tutorial

In this appendix, we're going to talk about Google Maps API. We used Google Maps API extensively during our development of LocalMapia, and would like to provide you a quick reference on embed a Google Maps on to your website. Please note that this is just a quick reference, you can find the full API documentation at:

http://code.google.com/apis/maps/documentation/reference.html

1. Register for an API key

In order to use Google Maps API, you need to sign up for a developer API key. Also, you need a Google Account, or your Gmail account, which you can register one for free. You'll need to go to following page to register for an API key:

http://code.google.com/apis/maps/signup.html

After sign up, note down you API key, as you'll need to use it later.

2. Your first Google Maps application

Google Maps API is written in JavaScript and supports by many modern browser. You'll need to embed those JavaScript code provided by Google by use this following code:

```
<script src="http://maps.google.com/maps?file=api
&amp;v=2&amp;key=[your API key]&sensor=true_or_false"
type="text/javascript"></script>
```

Note: You need to set the sensor value according to your usage of the API. Set this value to true if you're incorporate your application with some kind of GPS device. However, for this simple application, you can set it to false

After that, you'll need to insert a snippet of JavaScript to embed the map into your webpage. The code we provided below will replace an element with an id `map_canvas` with a map interface, and set the center of the map to latitude 37.4419 and longitude -122.1419.

```
    <script type="text/javascript">

    function initialize() {
      if (GBrowserIsCompatible()) {
        var map = new GMap2(document.getElementById("map_canvas"));
        map.setCenter(new GLatLng(37.4419, -122.1419), 13);
        map.setUIToDefault();
      }
    }

    </script>
```

Note: You should call map.setCenter(); immediately after create a new GMap2 object.

Then in <body> tag in your HTML document, add an event callback to it to trigger the code above after page loads, and another callback to clean up the memory usage after user close the page, which provides by Google.

```
<body onload="initialize()" onunload="GUnload()">
```

So, after you assemble those pieces together, your code should be look closely to this:

```
<!DOCTYPE html "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-
8"/>
    <title>Google Maps JavaScript API Example</title>
    <script
src="http://maps.google.com/maps?file=api&amp;v=2&amp;key=abcdefg&sen
sor=true_or_false"
            type="text/javascript"></script>
    <script type="text/javascript">

    function initialize() {
      if (GBrowserIsCompatible()) {
        var map = new GMap2(document.getElementById("map_canvas"));
        map.setCenter(new GLatLng(37.4419, -122.1419), 13);
        map.setUIToDefault();
      }
    }

    </script>
  </head>
  <body onload="initialize()" onunload="GUnload()">
    <div id="map_canvas" style="width: 500px; height: 300px"></div>
  </body>
</html>
```

After you open that file in your HTML browser, you should see the map on the screen as in Figure D.1. If not, make sure that you have an internet connection, since Google Maps API requires internet connection to run.

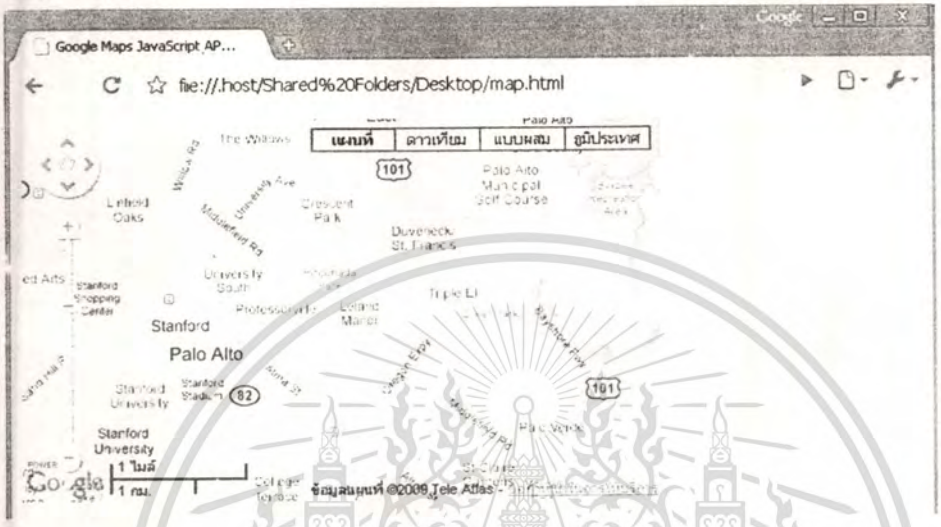

Figure D.1: Basic map page

3. Map control and map type

By using Google Maps API, you can actually customize the map the way you want. From the example in the last section, you could see that we called `map.setUIToDefault();` to set the look and feel as the default on. We can customize those by these functions.

Note: you'll need to remove the line map.setUIToDefault(); from example code, as it will reset everything back to default

3.1 Remove map type

To remove map type, use this function:

```
map.removeMapType(G_HYBRID_MAP);
```

This will remove the Hybrid map from map type panel. You can change the `G_HYBRID_MAP` to any map type specify in Google Maps API documentation.

3.2 Add map controls

There're various map controls that you can choose from. The default map control that comes with `setUITodefault()` is `GLargeMapControl3D()`. You can specify the type of control by using this method:

```
map.addControl(new GSmallMapControl());
```

Please refer to the full documentation for the list of supported map control types.

You'll also want a map type control, so user will be able to select the map type. You can add the control the same way as you add previous map control.

```
map.addControl(new GMapTypeControl());
```

The result of the above code will look like this:

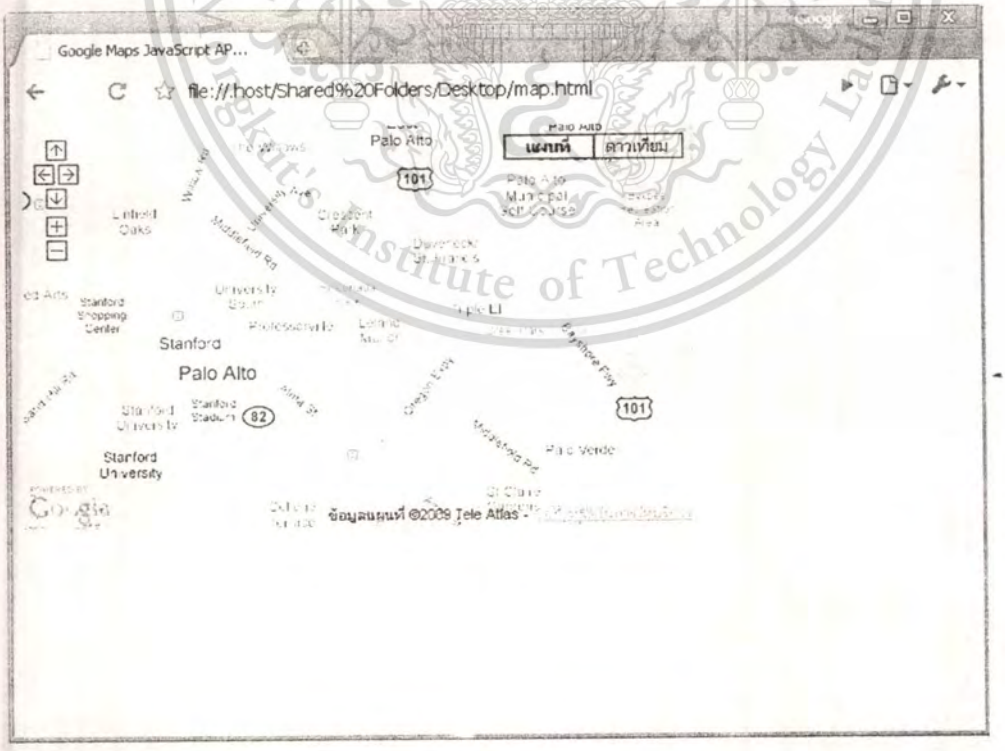

Figure D.2: Map with small map control and removed hybrid type

4. Marker

You can add a marker on any latitude and longitude you want by using the GMarker class. By defaults, the constructor takes an argument of `GLatLng` object, and will display a default red marker.

To add a new marker, simply use this code:

```
var marker = new GMarker(new GLatLng(37.4419, -122.1419));
map.addOverlay(marker);
```

This will add a new maker in the middle of the map from previous example. Also note that you'll need to call `addOverlay()` method to add the new marker to the map.

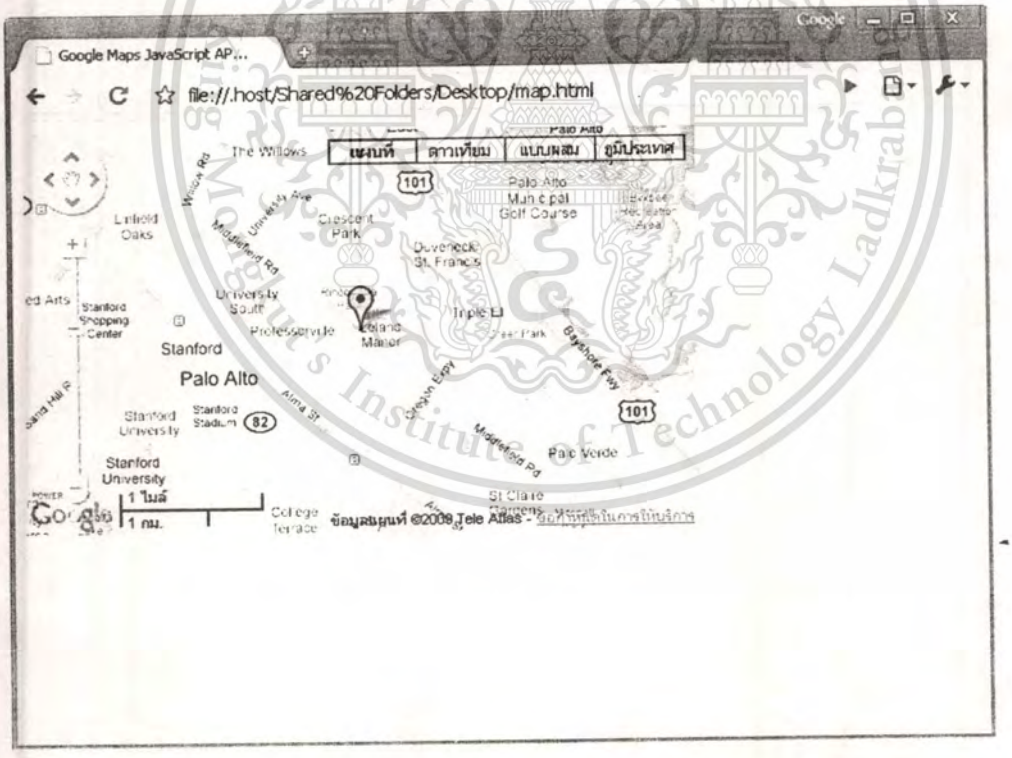

Figure D.3: Map with a marker

There're many ways that you can customize the marker, such as use you own photo as a marker instead. For more information about this, please look at the API documentation.

5. Line and polygon

You can also draw a line on the map using the API by using `GPolyline` class. The constructor of this class accepts an array of `GLatLng` as a parameter, and you can specify any latitude, longitude you want.

To add a new line, or polyline, use this code:

```
var polyline = new GPolyline([
new GLatLng(37.4419, -123.1419),
new GLatLng(37.8439, -120.1419)
], "#ff0000", 10);
map.addOverlay(polyline);
```

This code will add a new 10 pixels width red polyline on the map. You can customize the color and width of the line as you want.

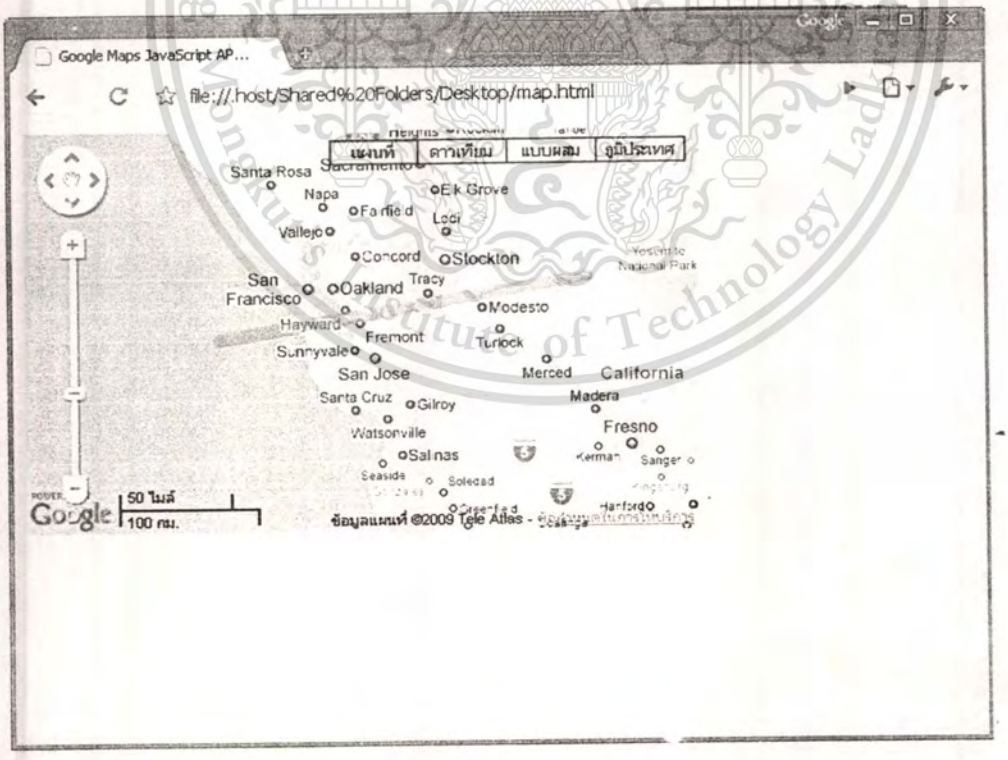

Figure D.4: Map with a GPolyline

To add a new polygon on the map, usually useful for mark a region, you can use

`GPolygon` class. This class's constructor takes almost the same argument as `GPolyline`.

```
var polygon = new GPolygon([
new GLatLng(37.4519, -122.1319),
new GLatLng(37.4319, -122.1319),
new GLatLng(37.4319, -122.1519),
new GLatLng(37.4519, -122.1519),
new GLatLng(37.4519, -122.1319)
], "#f33f00", 5, 1, "#ff0000", 0.2);
map.addOverlay(polygon);
```

The above code will add a new red polygon on the map. The forth parameter is the opacity of the border, which 1.0 means solid and 0 means transparent. Fifth parameter is the fill color, and sixth parameter is the opacity of the fill.

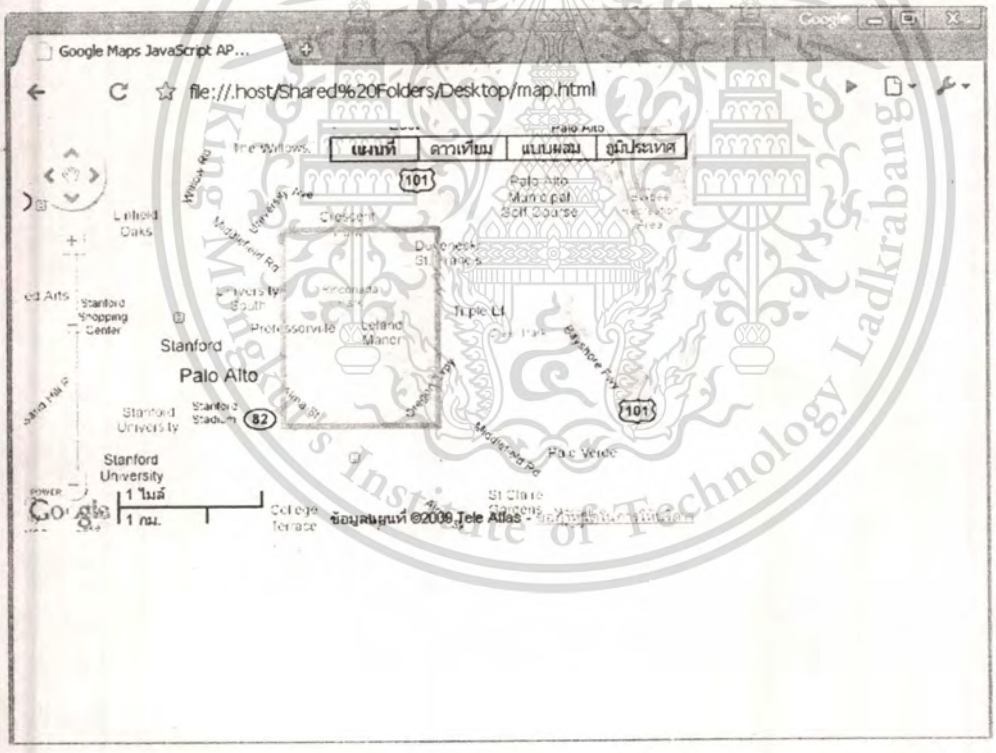

Figure D.5: Map with a GPolygon

6. Conclusion

In order to use the Google Maps API, we would also like you to consult the Google Maps API site at:

`http://code.google.com/apis/maps/`

On that page, you'll find lots of information regarding Google Maps API usage, also some example to get you starts in no time. There's a lot more features that we were unable to cover in this appendix, and we would like to encourage you to look on the site yourself.

Appendix E

Installation

In order to run our web application, there're some application stack that needs to be installed first. These stack includes a Ruby intepreter, MySQL database server, Ruby external libraries called Ruby gems, Ruby on Rails framework, and Mongrel web server. We'll show you how to install this application stack on a computer running on Ubuntu Linux. You'll find out that the command will be the same as in Debian-like Linux distributions.

E.1 Installing Ruby and MySQL server

To install Ruby and MySQL server, we'll use Debian's Advance Packaging Tool, or APT, to manage the installation for us. While you're logged in as root, issue this command:

```
root$ apt-get -y install build-essential libssl-dev libreadline5-dev
mysql-server openssl ruby ruby1.8-dev irb rdoc ri libopenssl-ruby
lbmysqlclient15-dev
```

During the process, you'll be prompted to enter your MySQL root password. You can set the password to anything you want, but please make it secure and note it down, as you'll need to use it again during the installation process.

E.2 Installing RubyGems package manager

You'll need to install RubyGems package manager to install external libraries, or gems, to your machine. You need to download the source code from RubyGems website and compile it ourselves, since RubyGems version in APT repository is not up-to-date as the date of this writing.

```
root$ wget http://rubyforge.org/frs/download.php/45905/rubygems-
1.3.1.tgz
root$ tar xvf  rubygems-1.3.1.tgz
root$ cd rubygems-1.3.1
root$ ruby setup.rb
```

E.3 Install Ruby on Rails framework and gems

You'll need to use gem command to install the framework into your machine. Run these commands as root

```
root$ gem install rails BlueCloth haml hpricot mislav-will_paginate
mongrel mysql shuber-attr_encrypted --source http://gems.github.com
```

E.4 Check out the code from repository, or copy the code from CD

If you have an internet connection and familiar with Git, then you can clone our source code from our repository on Github (http://github.com) at git://github.com/sikachu/localmapia.git. By defaults, the source code will be located inside a folder name localmapia. Otherwise, copy the source code from the CD to your desired folder.

E.5 Setup the database

First, you need to copy the file config/database-sample.yml and make a copy as config/database.yml.

```
$ cp config/database-sample.yml config/database.yml
```

Now open that file in the text editor. Edit the database setting for production environment (under production: block) to your setting:

```
production:
  adapter: mysql
  username: root
  password: [your password]
  encoding: utf8
  timeout: 5000
```

Save and close the file, then issue this command to create the database and tables:

```
$ rake db:create RAILS_ENV=production
$ rake db:schema:load RAILS_ENV=production
```

Please note that we set RAILS_ENV to be production, to specify that we wants to run our code in production environment.

After you done with those steps, you can start the server by using this command:

```
$ script/server -e production
```

After that, you can use your web browser to navigate to http://localhost:3000, and you'll see our website running on your computer.

Appendix F

Design Diagrams

During the development process, there're lots of diagram we created to illustrate the algorithm and design of the whole program. In this appendix, we're going to show you those designs.

F.1 Class diagram

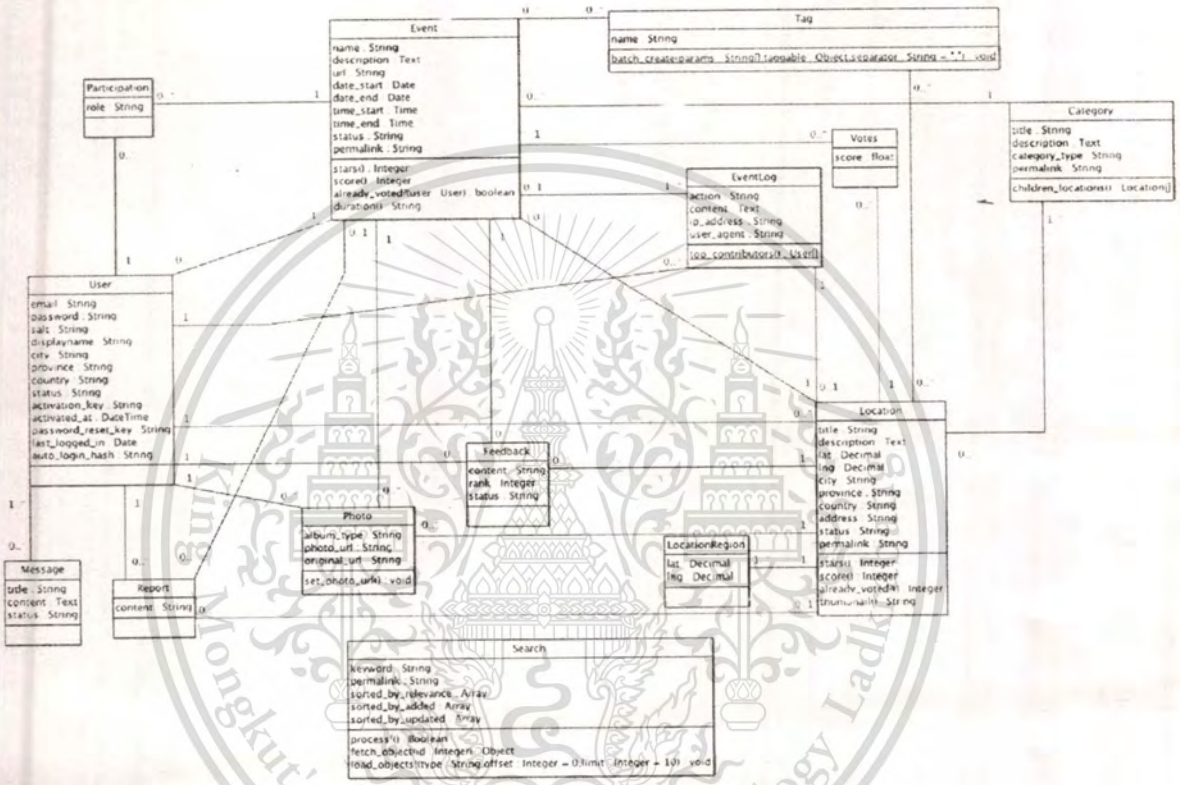

Figure F.1: LocalMapia's class diagram

As we already shown in Chapter 3, our design first started with some classes as models. We would design on how each model associates with each other, and then transfer those associations and design into the database tables.

For the result database tables, you can see Appendix B for more information.

F.2 Sequence Diagrams

During the development, we create a lot of sequence diagrams to confirm the interaction between each object and user. We've collected them and shown all below.

F.2.1 Add new location

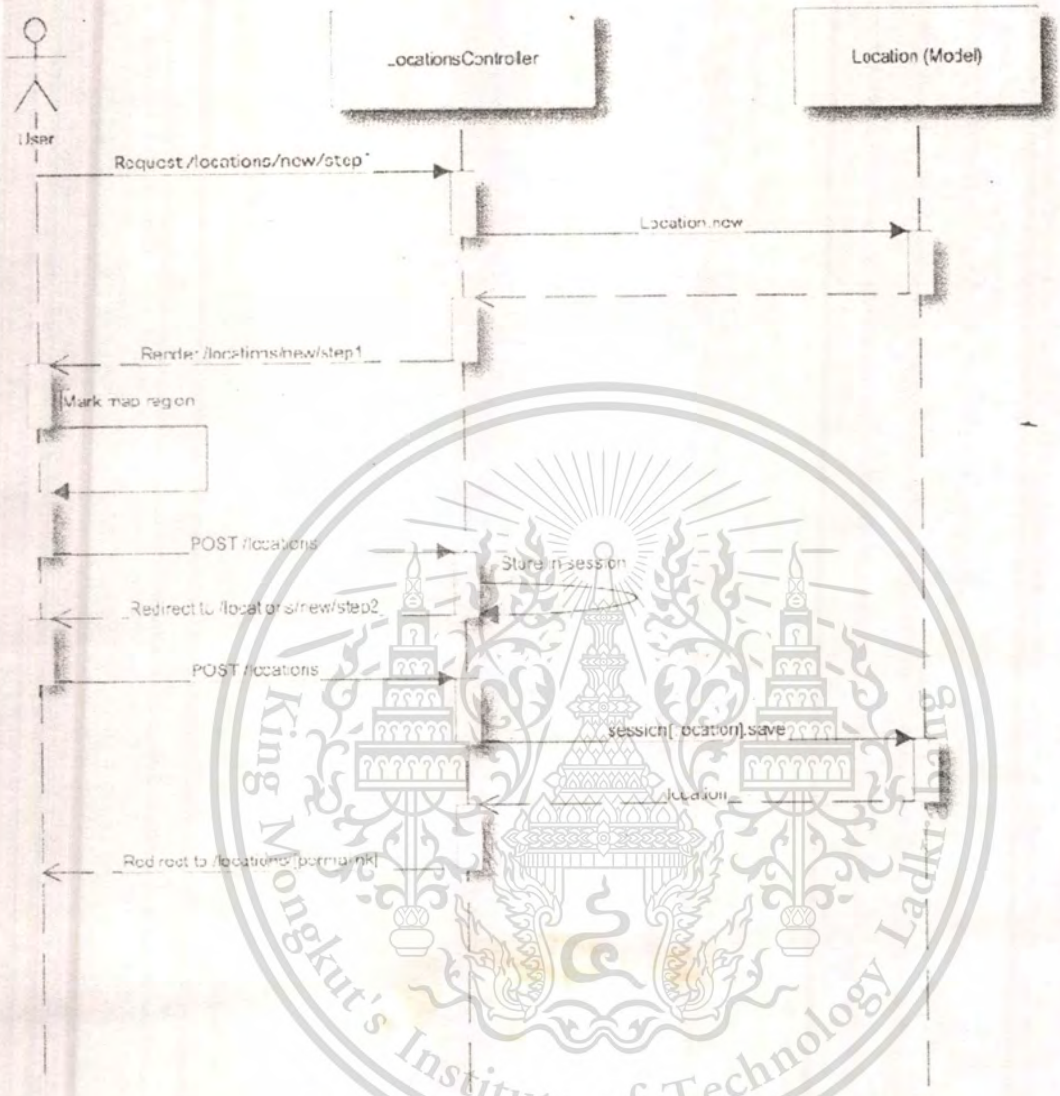

Figure F.2: Sequence diagram of adding new location

For adding a new location, first user will navigate to /locations/new/step1, which will call LocationsController in the step1 action. That LocationsController will create a new instance of Location model, and returns a map for user to select region. User will then submit the region to /locations, which will call create action in LocationsController to store those data temporary in the session and redirects user back to /locations/new/step2. User will need to enter location's information on that form and submit it too /locations. An create action will be caled again, but now it will store the location in the database and redirects user back to the newly created location's page.

F.2.2 Display new location

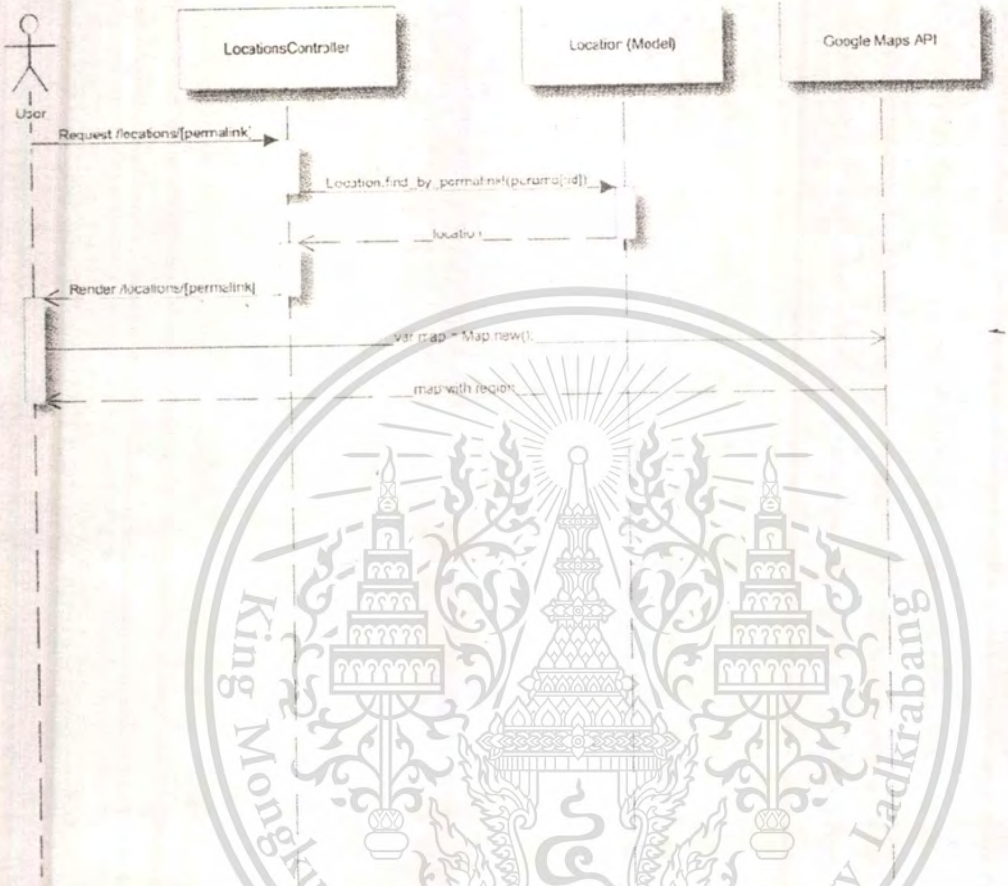

Figure F.3: Sequence diagram of displaying location process

To display a location, user will first make a request to location's permalink, such as /locations/[id]-[location_name]. It will calls LocationController's show action, which will load a corresponding location from Location model. After that, LocationsController will render that show action with an API call to Google Maps API. Then the browser will issue the call to API and renders the map with its corresponding region on the page.

F.2.3 Search for locations

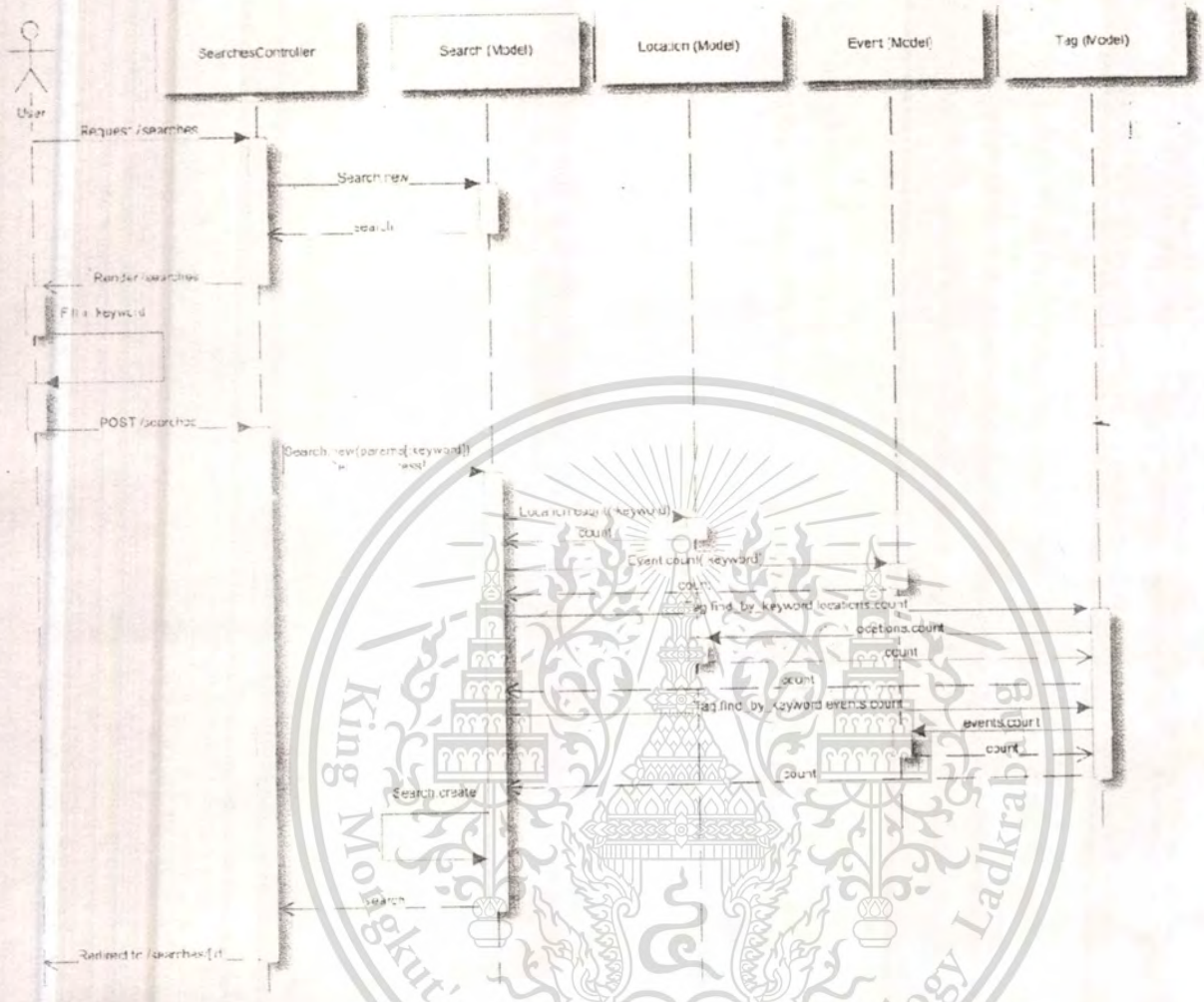

Figure F.4: Sequence diagram of search process

To search for a location, user first make a GET request to /searches, which will calls index action in SearchesController. That index action then will create a new search model, and renders a form back to user.

After that, user will enter the search term and submit a POST request to /searches. This will make a call to SearchesController's create action, and it will search for data by following this procedure:

- Search for locations and events based on the keyword, and store the id of those objects along with number of occurance of the keyword in an array, which will be used for sorting the result based on occurance.

- Lookup on Tag for a tag that has the same name as the keyword, store that tag for later use.
- If a tag is found, then load locations and events that associates with that tag, along with number of occurance of that keyword in those objects. Then store those data into the array.
- Union those arrays created from step 1 and step 3 to remove the duplicate data from the array.
- Sort those data based on the number of occurance, created date, and last updated date.
- Store those arrays into the database

After the search process is done, those results will be stored in the database as an array. Then SearchesController will redirect that user to the search result.

F.2.4 Add new feedback

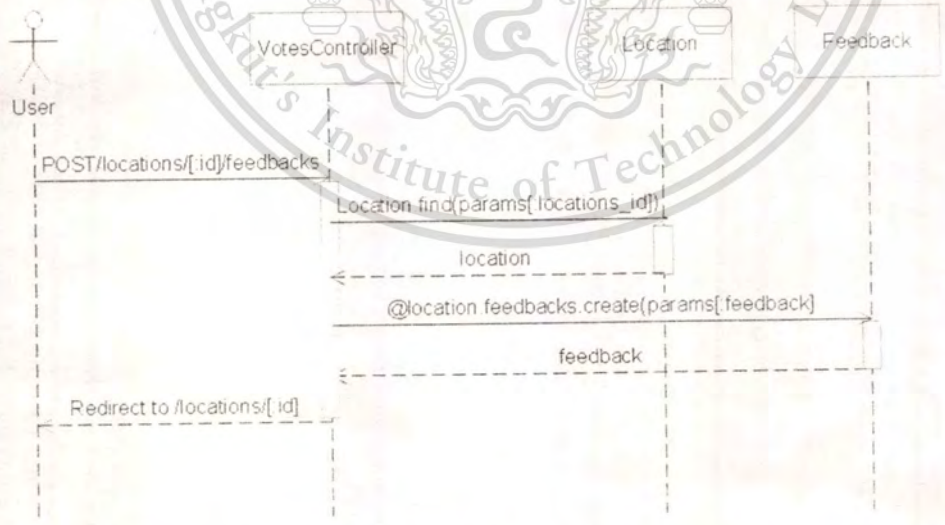

Figure F.5: Sequence diagram of add feedback process

To add a new feedback, user will input his or her feedback into the form, then submit the form to /locations/[:id]/feedbacks. This data will be sent to create action in FeedbacksController,

which will first load the Location from Location model, then creates a new feedback on it using create method on feedbacks collection. After the feedback is created, user will be redirected back to the location page, or event page, depends on his origin.

F.2.5 Add new tag

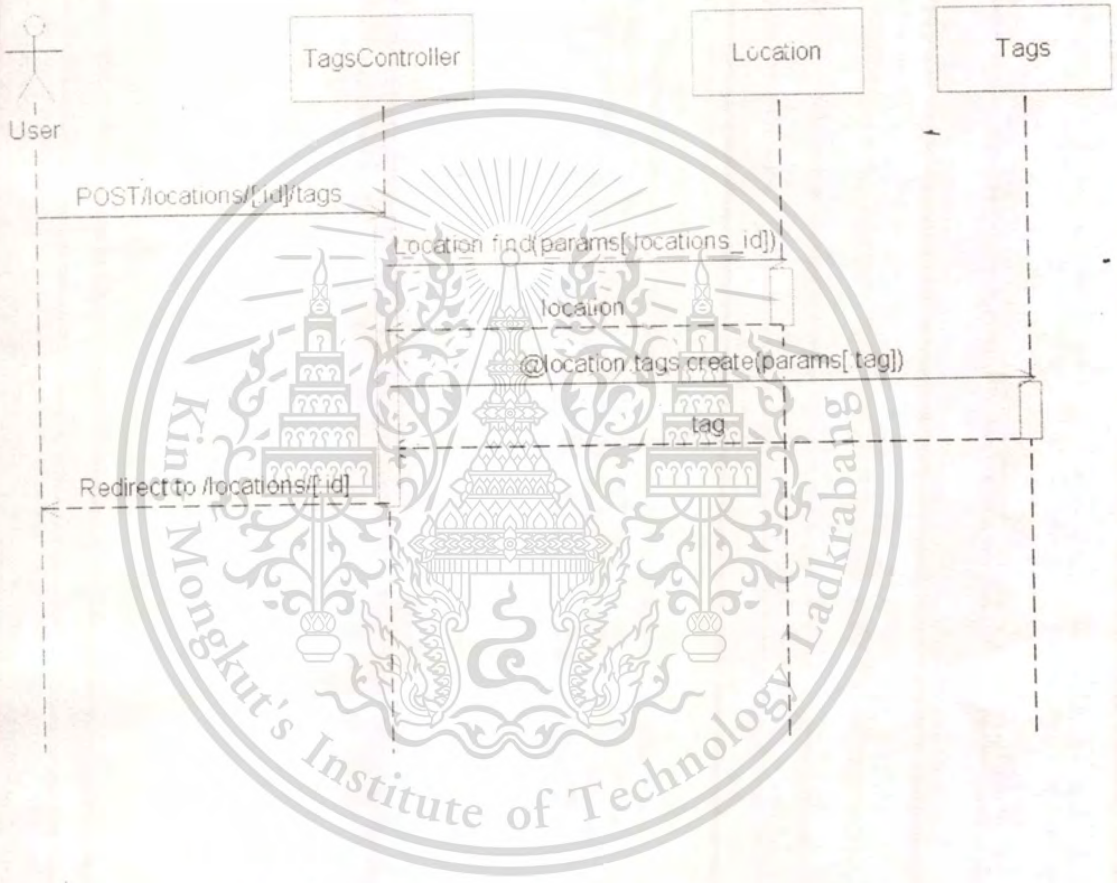

Figure F.6: Sequence diagram of add new tag

To add a new tag, user would have entered it in the form on the location or event page. After user submits the form, it will sent a POST request to /locations/[:id]/tags, which will calls create action of TagsController. In that create action, we would load the location and store it as instance variable first, then call a create method on the tags collection to create and add the tags. A new tag will be returned, and we would redirect user to the location or event page.

F.2.6 Add New Event

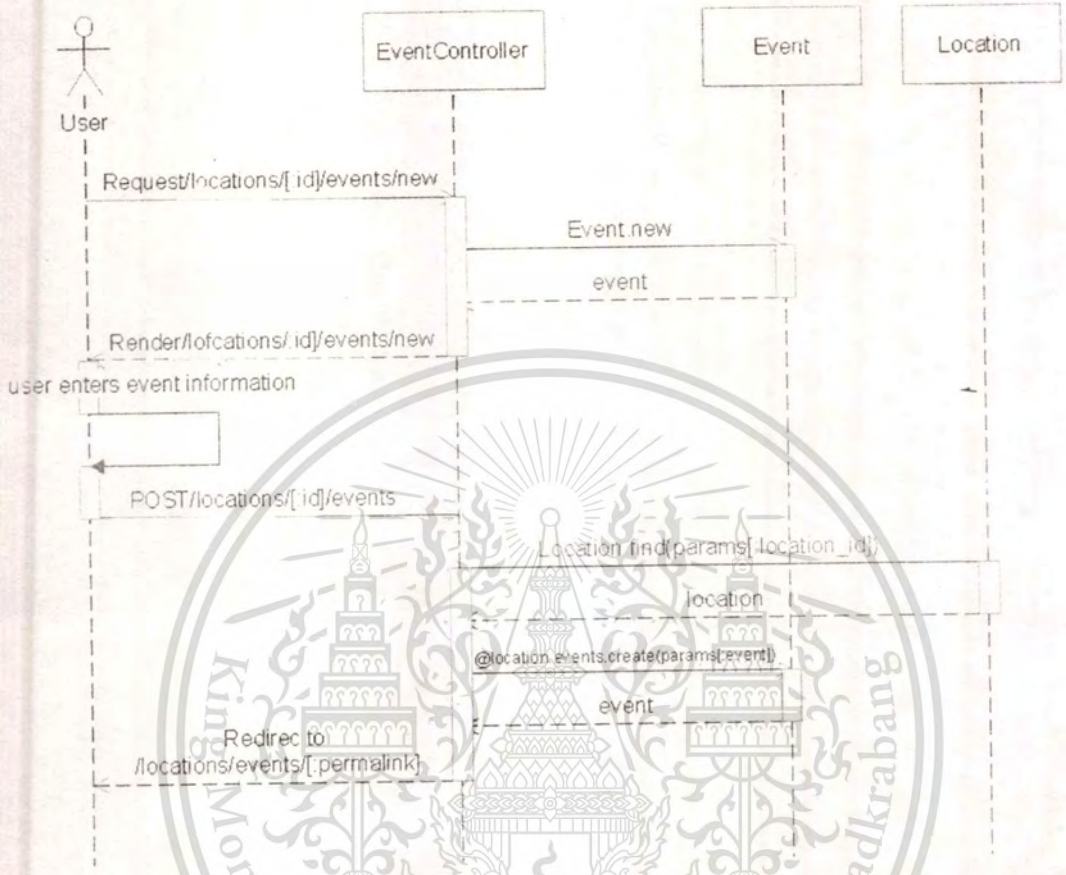

Figure F.7: Sequence diagram of add new event process

To add new event to a location, first user would navigate to /locations/[:id]/events/new, which will call new action on EventsController. The controller will create a new instance of event object, and render a form back to user. After user complete the form, he or she will post the form to /locations/[:id]/events, which will call the create action in EventsController. In this action, first the controller will load the location into an instance variable, then call the create method on the collection of events. After that, it will redirect user to newly created event.

F.2.7 Add Rating

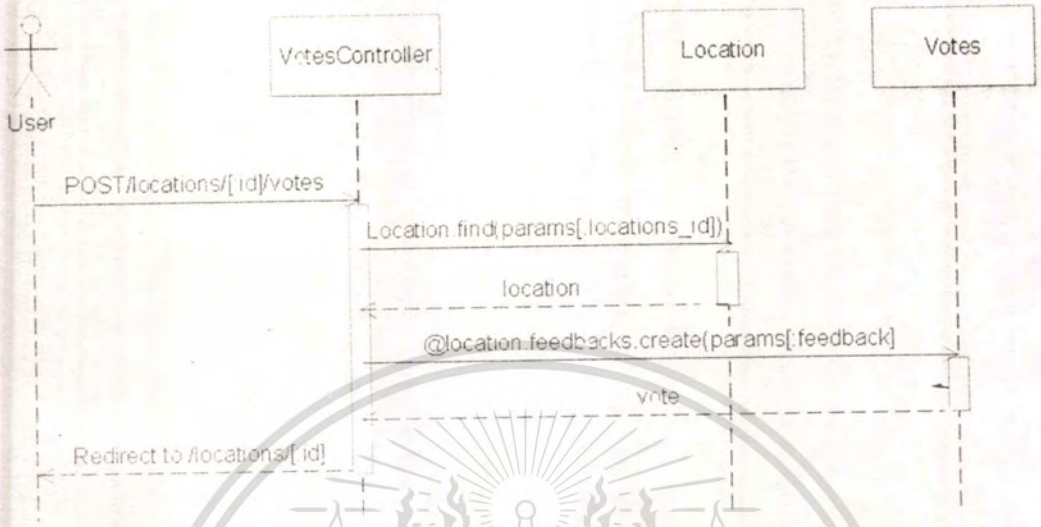

Figure F.8: Sequence diagram of add new rating

F.2.8 Browse by Category

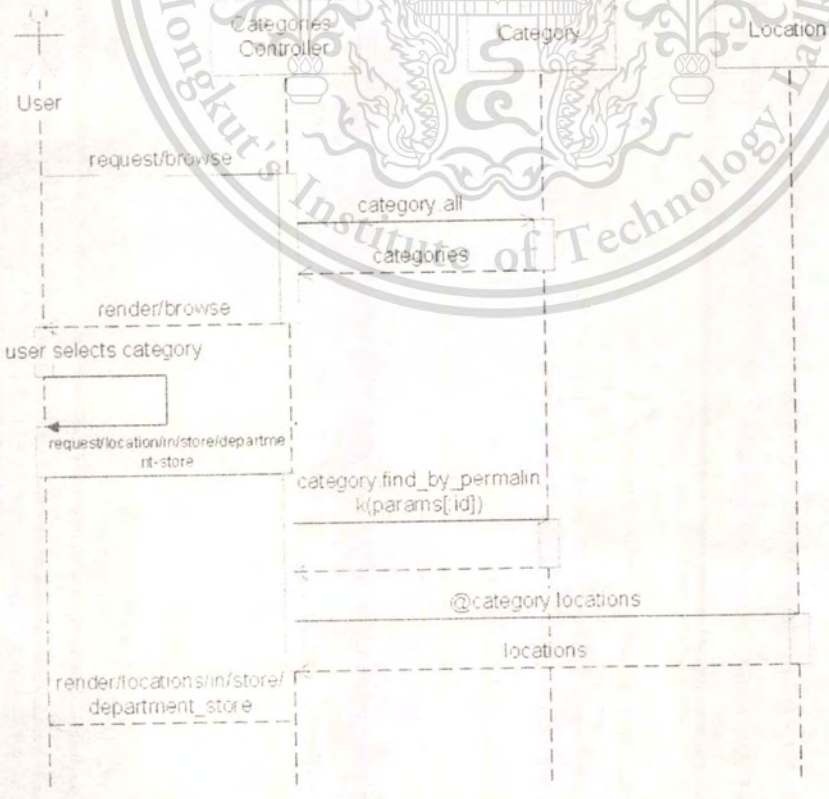

Figure F.9: Sequence diagram of browse by category process

To browse by category, first user would navigate to /browse, which will calls index action on CategoriesController. That index action will call Category.all method to fetch all of the categories and list them as categorized, then render those categories listing back to user. User would select a category on the page, then make a request to category page, such as /locations/in/store/department-store. This will call show action in CategoriesController. In this action, the controller will load the category into an instance variable first, and then use the locations association collection to retive locations that's in this category. This action will then render the location listing back to user.

F.2.9 Browse by Event

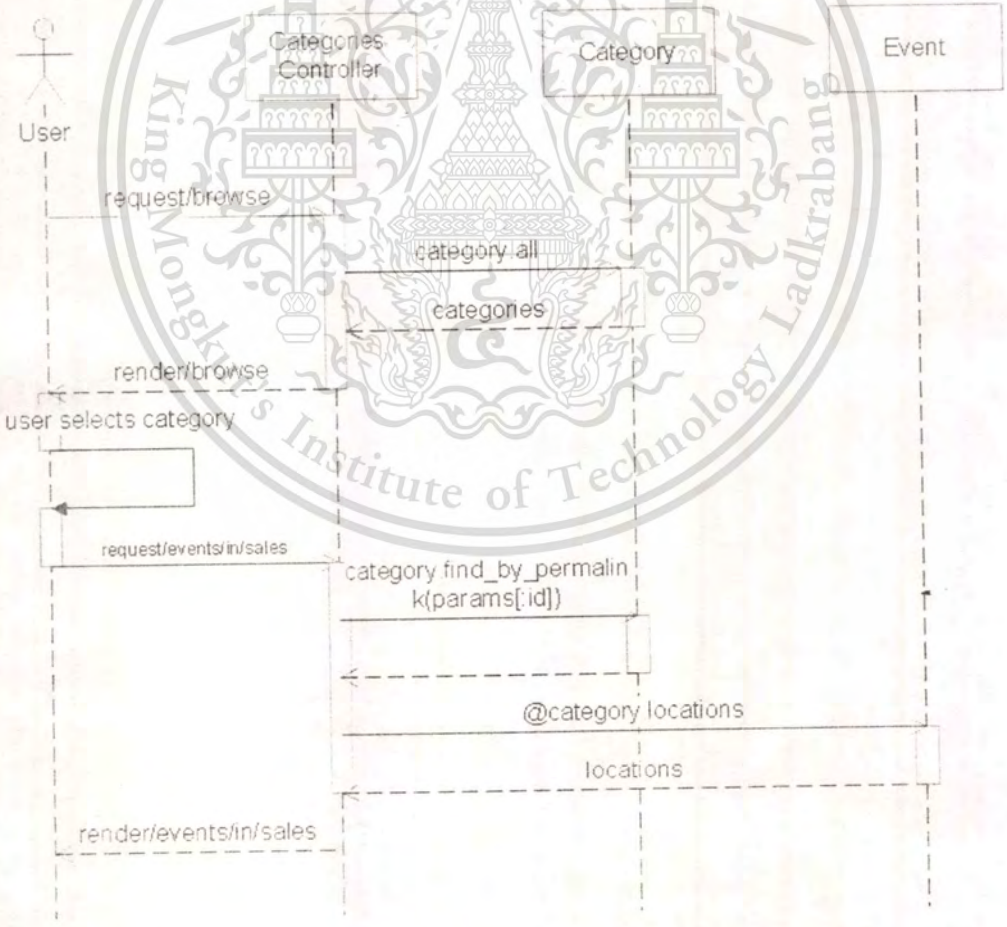

Figure F.10: Sequence diagram of browse by event process

F.2.10 Browse by Tag

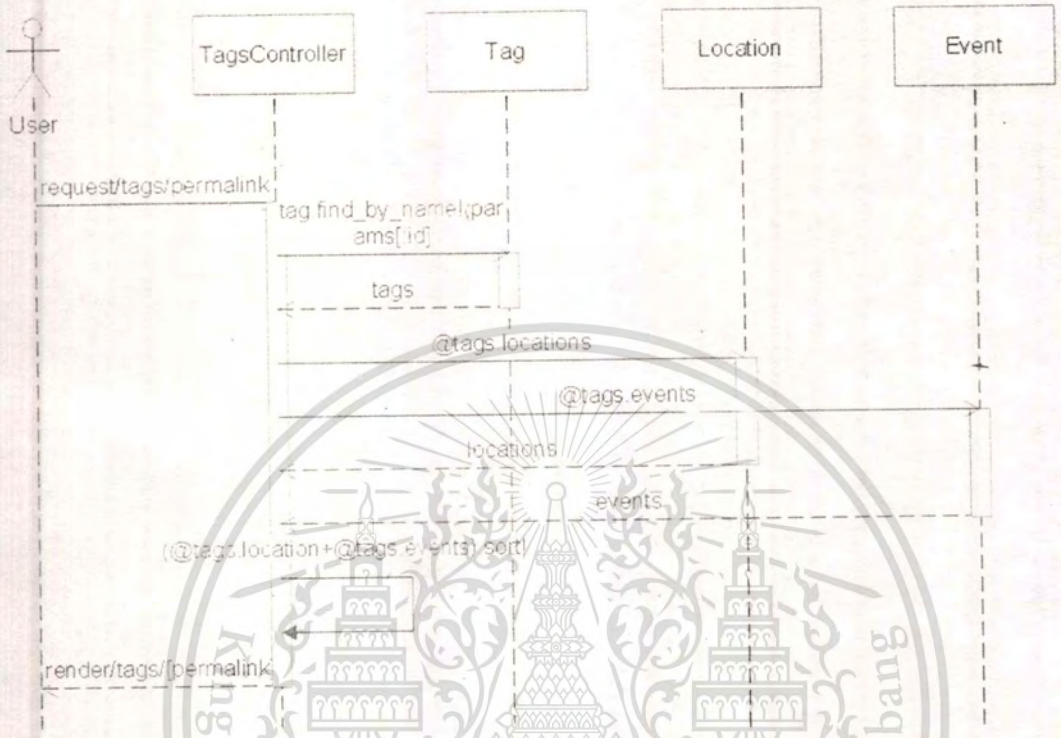

Figure F.11: Sequence diagram of browse by tag process

This process is almost the same as browse by category process, except that the result set must be added together and then sort according to sorting option.

F.2.11 Edit Information

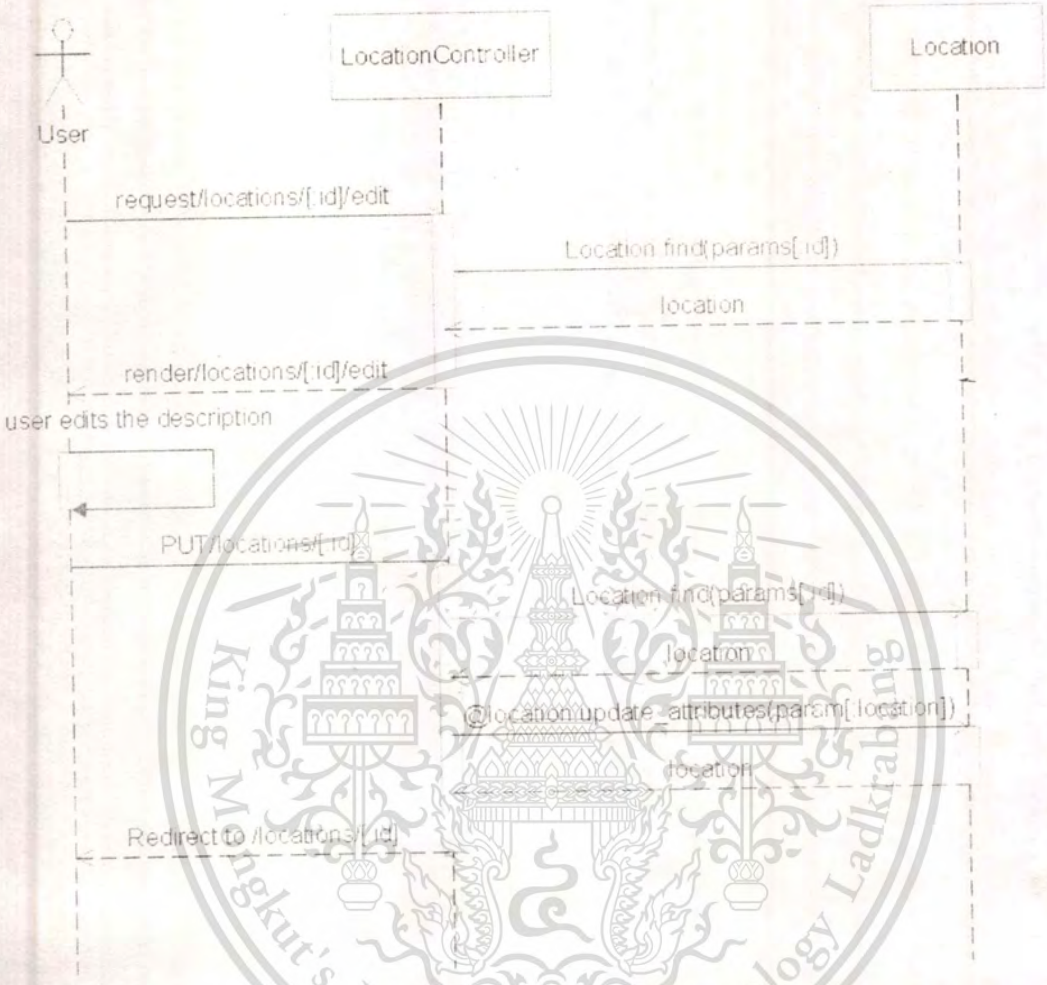

Figure F.12: Sequence diagram of edit information process

In this process, first the user will navigate to /locations/[:id]/edit, which will call new action on the LocationsController. The action will loads up the location data from the database, and display a form for user to edit. After that, user will submit the form, result in PUT request to /locations/[:id]. This will call update action on the location, which will load up the location and call update_attributes to save changes made by user to the database. Then the user will be redirected to location page.

For event, the same process will be applied except that instead of calling method of Location object, it would call the method on Event object

F.2.12 Send Report

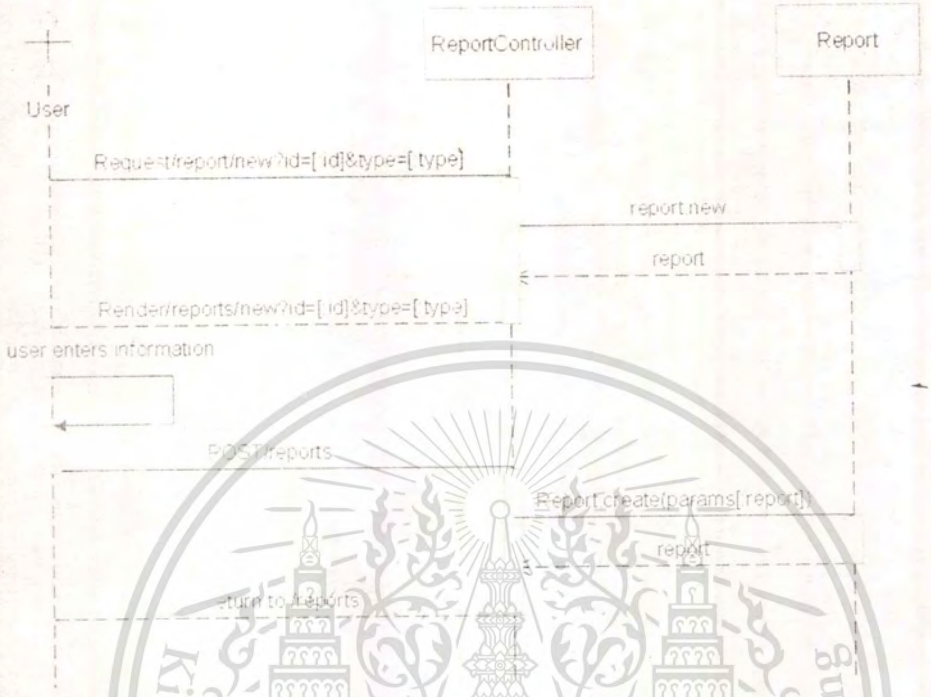

Figure F.13: Sequence diagram of send report process

F.2.13 Send to Friend

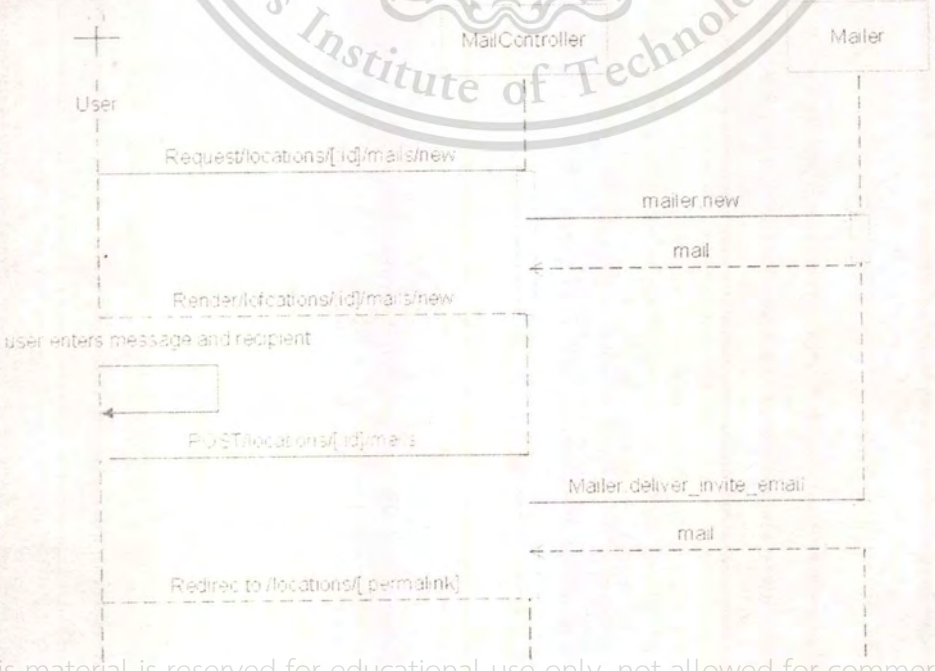

Figure F.14: Sequence diagram of send to friend process

F.2.14 Show Location

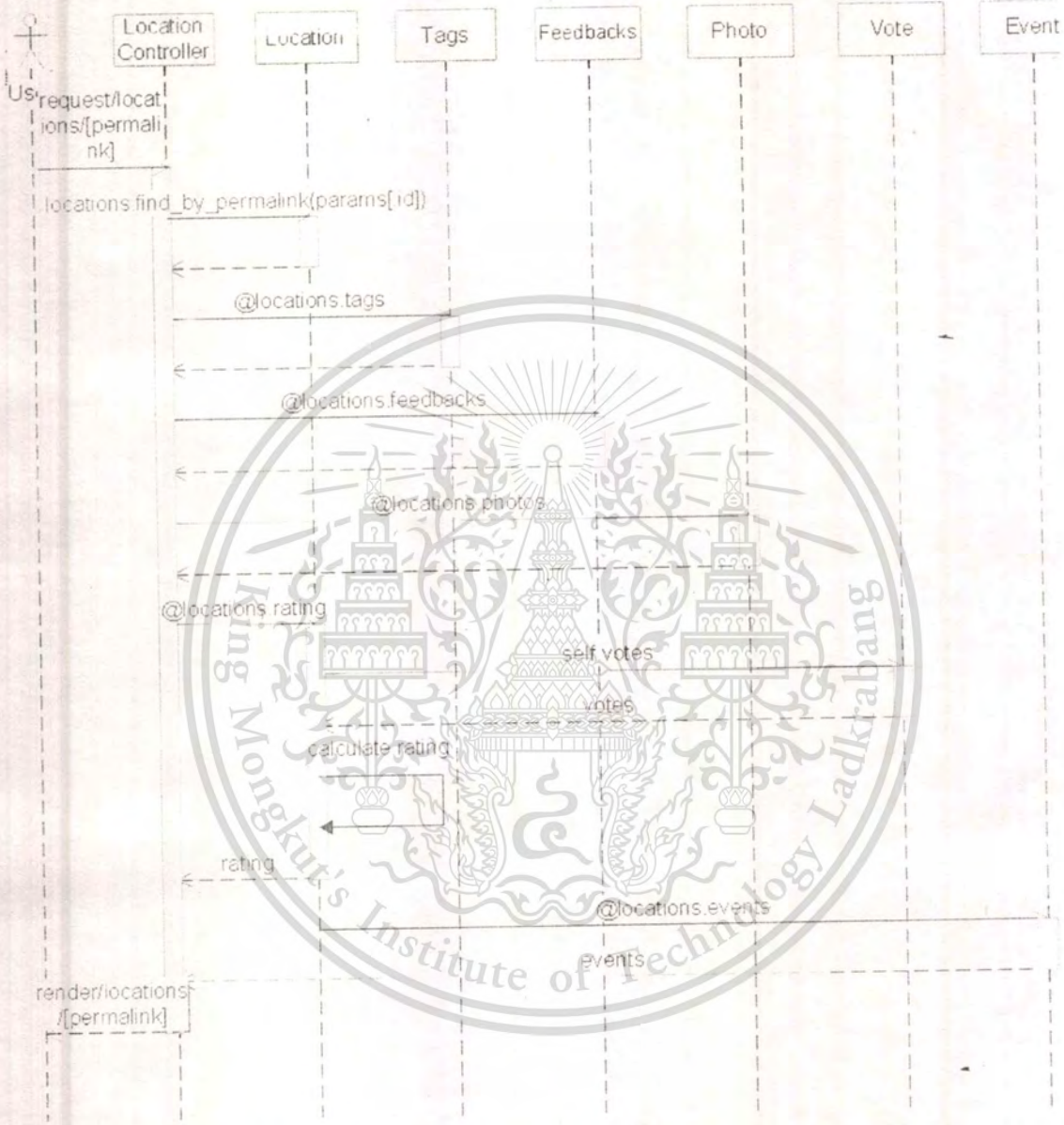

Figure F.15: Sequence diagram of show location process

To display location page, after user requests /locations/[:permalink], a show action will be called on LocationsController, which will load the location into database and also calls lots of association collection methods to form the data in order to display the page.

F.2.15 Add Photo

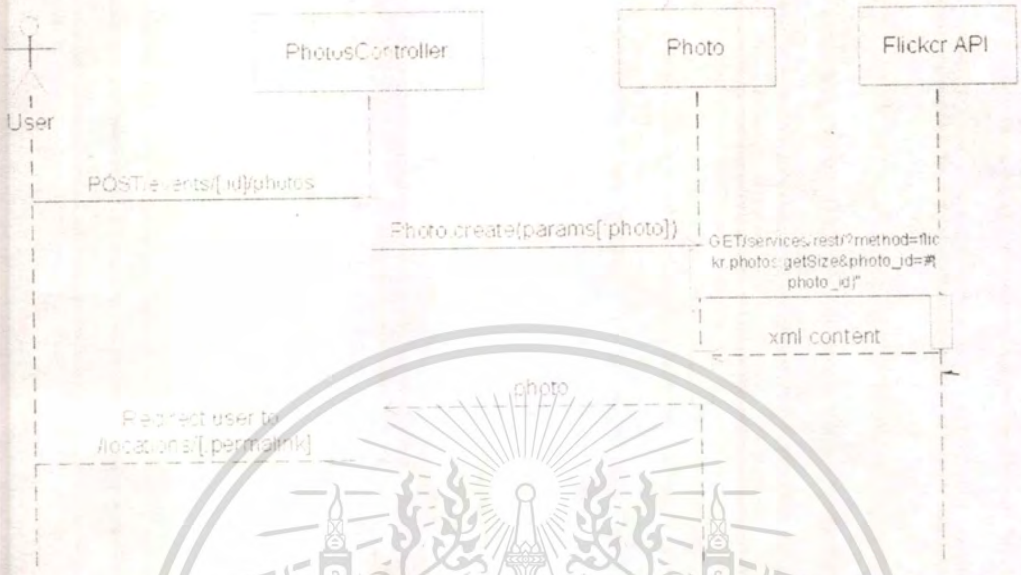

Figure F.16: Sequence diagram of add new photo

The add new photo process is almost the same as adding anything else, except that upon adding the URL the Photo model needs to contact Flickr API to retrieve list of photos in various size. It will then store the URL of thumbnail version into the object.

F.2.16 Watch this event

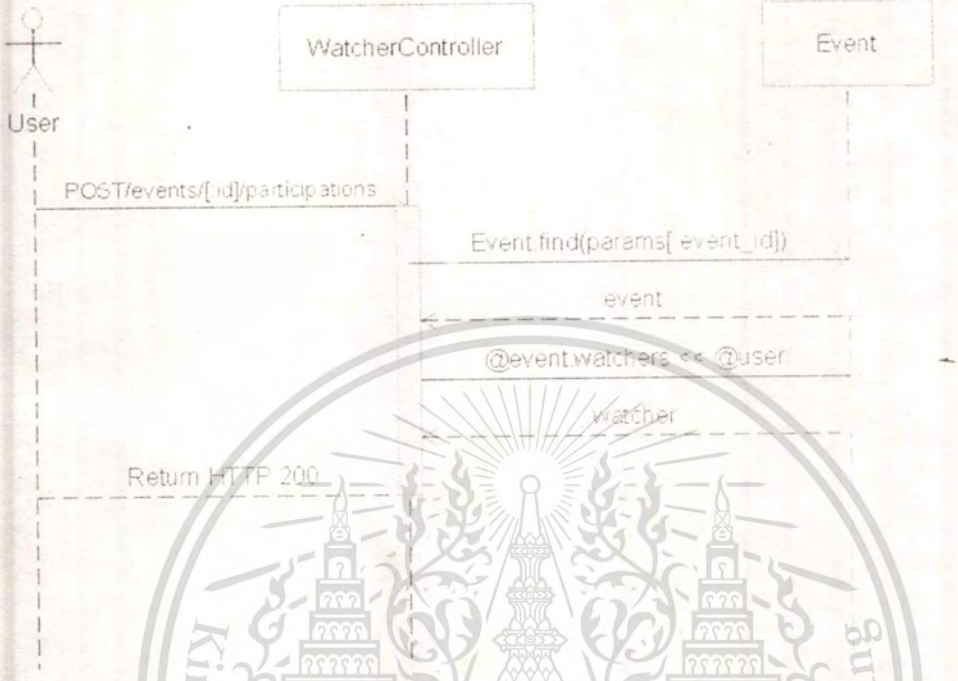

Figure F.17: Sequence diagram of 'watch this event' process

F.2.17 I'll participate this event

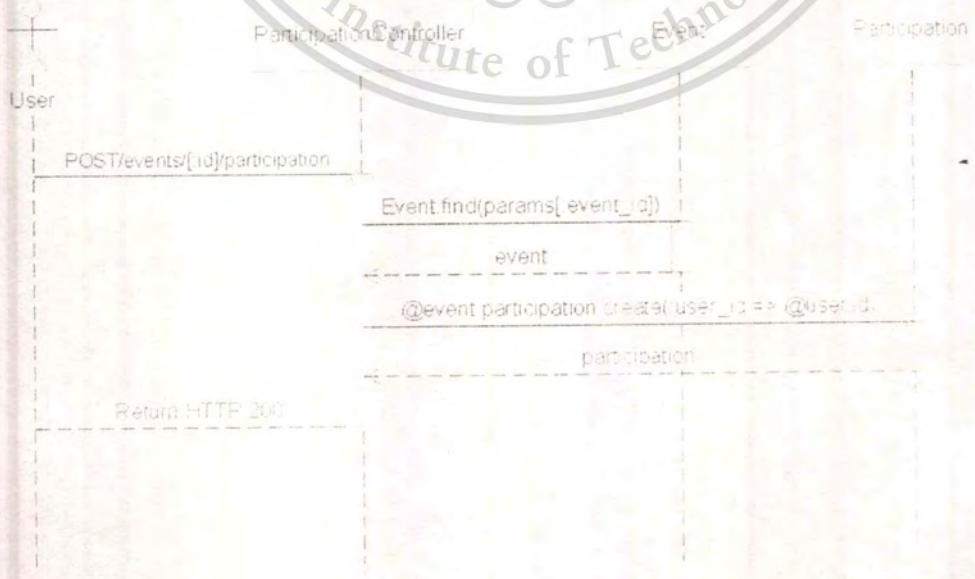

Figure F.18: Sequence diagram of 'I'll participate this event' process

Author Biography

Name	Mr. Sittidej Thongkhum
Date of Birth	May 22th, 1987
Place of Birth	Bangkok, Thailand
Education	1998 - 2004, High School and Junior High School
	Mattayomwatbungthonglang School
	Bangkok, Thailand

Name	Mr. Prem Sichanugrist
Date of Birth	July 2nd, 1986
Education	1998 - 2004, High School and Junior High School
	Samsen Wittayalai School
	Bangkok, Thailand