TERM2 FYP REPORT

Prepared for: Dr. LYU Rung Tsong, Dr. Sun Hanqiu
Prepared by: LAM Chi Kit, Wong Ka Lam
FYP Title: Virtual Notice Board

Table of Contents

1. INTRODUCTION 4

1.1 OVERVIEW 4

1.2 MOTIVATION 6

1.3 OBJECTIVE 8

1.4 OS PLATFORM 9

2. MILESTONE 10

3. MSGCRAFT OVERVIEW 14

3.1 SERVER 14

3.2 IPHONE APP 14

4. MSGCRAFT DESIGN 15

4.1 SERVER 15

4.1.1 Server-Side System Architecture 15

4.1.2 Database 16

4.1.3 Term 1 ER Diagram 17

4.1.4 Term 1 Schema 18

4.1.5 Term 2 ER Diagram 19

4.1.6 Term 2 Schema 20

4.1.7 Sequence Diagram 22

4.2 PHONE APP 24

4.2.1 Term 1 Flow Chart 24

4.2.2 Term 2 Flow Chart 26

4.2.3 Modules 28

4.2.3.1 Term 1 MVC 29

4.2.3.2 Term 1 Class Diagram 31

4.2.3.3 Term 2 MVC 33

4.2.3.4 Term 2 Class Diagram 36

5. MSGCRAFT IMPLEMENTATION 37

5.1 SERVER 37

5.1.1 Term 1 Retrieving messages 37

5.1.2 Term 1 Post Message 39

5.1.3 Term 2 Retrieving messages 41

5.1.4 Term 2 Post Message 41

5.2 TER 1 IPHONE APP 43

5.2.1 UI Design 43

5.2.1.1 Map View Annotations 43

5.2.1.2 Map View Category Menu 44

5.2.1.3 Post Message View 47

5.2.1.4 Message Display View 49
5.2.1.5 Range slider bar for selecting time period 51
5.2.1.6 Message Display 53
5.2.2 Modules 54
5.2.2.1 Implementation of Location Services: 55
5.2.2.2 Implementation of CLLocation and CLHeading: 56
5.2.2.3 Implementation of Longitude-Latitude Distance 57
5.2.2.4 Implementation of MapViewController. 59
5.2.2.5 Implementation of postMsgViewController 63
5.2.2.6 implementation of segue 64
5.2.2.7 Implement of Danmaku(Message Display) 66
5.2.2.8 Implementation of RangeSlider 68
5.3 Term 2 IPhone App 70
5.3.1 UI Design 70
5.3.1.1 previewMessage View 70
5.3.1.2 preiewMessage View GPS and Beacon 71
5.3.1.3 previewMessage View Category Menu 73
5.3.1.4 previewMessage View Refresh Button 75
5.3.1.5 previewMessage View Content / Emoji Mode 76
5.3.1.6 postMessage View 80
5.3.1.7 previewMessage View Navigation 83
5.3.2 Modules 84
5.3.2.1 Implementation of Beacon Services 85
5.3.2.2 Implementation of 3D Rendering 87
5.3.2.3 Implementation of postMessageView 90
6. LIMITATIONS AND DIFFICULTIES 93
6.1. LIMITATION OF BEACON: 93
6.2. LIMITATION OF SCREEN SIZE: 93
6.3. DIFFIFULTIES OF BUILDING 3D OBJECT 94
7. CONCLUSION 95
8. REFERENCE: 97
9. ACKNOWLEDGEMENT: 98

1. Introduction

1.1 Overview

Notice Board is a platform that let people post public message. It can be used in many purposes. We can use it to advertise goods that for sale, to promote events/activities, to announce information or to express personal feelings and so on. The traditional Notice Boards are often made of cork to make the message adding and removing easily. At the university, there is a well known notice board call Democracy Wall which let student express their feeling of themselves; At the canteen or supermarket, we can often see that there is also a notice board inside to let the customer leave their opinions in order to improve the quality of the shop. But the traditional notice board is not that popular nowadays. It is mainly because the inconvenience of leaving messages and the duration the messages can last. Indeed, when people want to post some messages on the notice board, they have to print/write a note first. Then use some sticker to stick that post on the board. Indeed, there is too much things to prepare. Because of the inconvenience of using notice board, traditional notice board is not that popular nowadays.

Apart from the traditional notice board, there is an electronic version of notice board, Internet forums. Although using the Internet forums is more convenient, it lost the location characteristic of it. Say, if there is a physical notice board inside a canteen, the information on the notice board may contain the promotion of that canteen, others' feedback and so on. Those characteristics cannot be replaced by online discussion forums.

1.2 Motivation

Although a traditional board is not that popular now, we still think that it has its values. But a notice board has limitations. In case if there is a lot of post, all the post cannot be put on the board at the same time because of the boundary of size and the lack of space. We find that there are always some notes on the board cover the others note. Moreover, if someone sees a post that they don't like, maybe related to the complainant of an event, the haters may try to damage it in order to make some post disappear. In addition, traditional notice board is not environmental friendly as many paper and stickers are used for leaving messages.

The second motivation that we want to achieve is to keep record of the post. In the previous year, there is Lennon Wall created during Umbrella Movement, located at Central Government Complex. There is a large-scale notice board (the wall) that full of colorful post-it notes with many people written message on the universal suffrage and democracy. However, with the end of the Umbrella Movement, the Hong Kong government cleared the notes that stick on the wall very soon. And the colorful mosaic wall returned to an empty grey wall. We observe that many people express their feel and

opinion through the notes on the wall. But that kind of physical wall can be removed or destroyed easily.

The third motivation is that after adding both outdoor positioning system (GPS) and indoor-positioning system (Beacon), virtual notice board can perform their post and read function very well with accurate position in everywhere. Virtual notice not only can let people to express their feelings, it can also perform navigation by specifying the direction of some specific buildings so that viewers can know which direction should go in order to get to that place. For example, in Ho-Sin Hang Engineering Building, there has a board to show the room number of each professor and name of each room, however, there is no direction sign in each floor to indicate the route to each room. Student may take a long time to search for the room. By using the virtual notice board student can easily know where should go. In addition, the room availability of a room can also be see so that student can know is it free to use lab.

So we consider making the notice board on an electronic way. On the one hand, we focus on how to keep the notice board's characteristic and enhance it to break out its current limitation; Moreover, we hope this app can act as a history book. When the user use this app, they can know what have been happen on the past, from the discount of a shop, the changing of taste of food to the student campaign the year before. And that's why we have this project.

1.3 Objective

The goal of our project is to create a virtual notice board using a locational-based approach. We have set the following objectives for our app to achieve the goal.

The app should

1. Keep the advantage of traditional notice board. The beauty of traditional notice board is that the message is open to public. Everyone can see the message on the board. And the information of the post almost related to the things nearby.
2. Enhance the functionality of a notice board. The notice board has its boundary of size. We have to consider about how to handle the huge numbers of post the user post using the app
3. Has good user-experience and user-interface. User may feel inconvenient they have bad user-experience that made user stop using our app. A good user-experience and user interface may make the user addicted to use the app.

1.4 OS Platform

In our project, we have decided to build an IOS application in Swift.

This is our first time to build a mobile application. We found that IOS has a more uniform app development platform; it may be easier for us to get started on building an app then Android application. Moreover, with a uniform development platform, standardized environment and detail documentation of IOS, it is more convenient for searching suitable development tools, as well as future development. Another reason is that Apple provides CoreLocation Framework, which contains CLLocation for outdoor positioning and CLBeacon for indoor positioning. In addition, Apple also provides a 3D framework game technology, SceneKit, which is easy to use 3D framework, therefore we can build a 3D message reading and posting with this API. With this three API, we believed that our app can be built in more efficient and can work well in both indoor and outdoor positioning.

2. Milestone

In the term 1, we are going to implement the basic function of a virtual notice board. They are the functionalities of read and write a note. The following are the problems and consideration we going to solve in order to implement a virtual notice board.

1. Leaving a location based message. Since a notice board is a location-based object that people can leave a message in a particular location, such as inside a student hostel, a restaurant or a supermarket. So our app will record the current location of the user who wants to leave a post. GPS outdoor positioning system will be used to tackle this problem. GPS is widely used most of the location-based application, as it works all over the world with accurate positioning.
2. Locating all post. Since our target is making a virtual notice board over the world. And the board can be placed in everywhere. Then it comes with a problem - how can we know the place where having messages? Based on the location record we stored when the user post a message. We decided to make a collection of messages' location and point it out on the map to make a clear view of location of each message.
3. Filtering message. The number of message the user post will be more and more over the time. Assume there is add up over 10 thousand of message in the map. How the user find the post that they are interested in or they want to see. The filtering process

becomes important when there are a huge number of messages. Posting message with a category is a way to filter the messages and it can help the user know what type of messages they are reading in a particular place. And the second filtering method we going to use is showing message by time period. We decided to have a range selection bar that allow user to choose messages within a period of time.

4. Reading message. A good user interface is very important. We have mentioned before, a well-designed interface can made the user addict on our app. We found that most of the messenger application has a very similar user interface. It make user feel they are the same and not creative. We want to have some breakthrough in displaying or reading the messages. At this stage we decided to make application into a reading message view with a much more simple gesture - rotate the iPhone to landscape. User does not need even a click to switch the screen.

In term 2, our target is to improve the app built in the last semester which have the basic functionality, so that our apps can work well in every scenario and have a better performance. There are several areas that we aim to improve.

1. Positioning System. In term 1, GPS is adopted as the only positioning system in our app to locate user location. It works excellent in outdoor, but we found that it works with great error when the the device come into an indoor due to blocking and interference of the signal. It brings a grate limitation to our app which only work in outdoor scenario. So it is necessary to enhance the positioning system that supports both indoor and outdoor usage. We want our app can be more widely used in every scenario. So our first target in term 2 is making our app supporting indoor positioning as well. We decided to implement beacon technology that is now widely used in indoor positioning. It helps device determine their approximate location in indoor.
2. Simplify User Interface. Some feedback says that rotating the phone still complicated and not easy to use in interchanging the post message view and preview message view. User experience is one of our major concern in this project, so more simplified user interface is needed. We decided to fix our app in landscape mode, so that no more rotating is needed in using the app. In addition, the interchanging of posting view and reading view is simply clicking one button. In addition, as the app is going to support both indoor and outdoor by two different positioning

system, a symbol will be added to notify the user which positioning system the app is using.

3. More fancy user interface. A great user interface is very important in attracting user keep on using our app and give user good user experience. In term 1, we have decided a new of checking message by rotating the phone, then showing the message in a Danmaku view. But it is still a 2D plain text, we want a funnier and interesting way in reading the messages. Instead of displaying the message on 2D plain, we are going to build a 3D displaying engines. Our target is posting and reading a message in 3D space. Moreover, instead of displaying the full content of the message, symbol or emoji can be displayed. So that users can easily identify the category of each message. And clicking the symbol will show the full content of the message.
4. More filtering category. Originally there is only 4 kind of category, they are fashion, food, lifestyle and entertainment. We found that four categories may not enough. Therefore, more category will be added. For example, navigation category will be added. We believe that with well positioning system, the message can perform direction sign function.

3. MsgCraft Overview

POST is a social app that user can use it to leave message to perform as a virtual notice board.

3.1 Server

The server is used to store and manipulate the information about the location, category of the message and the message content. In this project, our server is put inside CSE department. It can be easy to setup the environment by using CSE and have a reliability server.

3.2 IPhone app

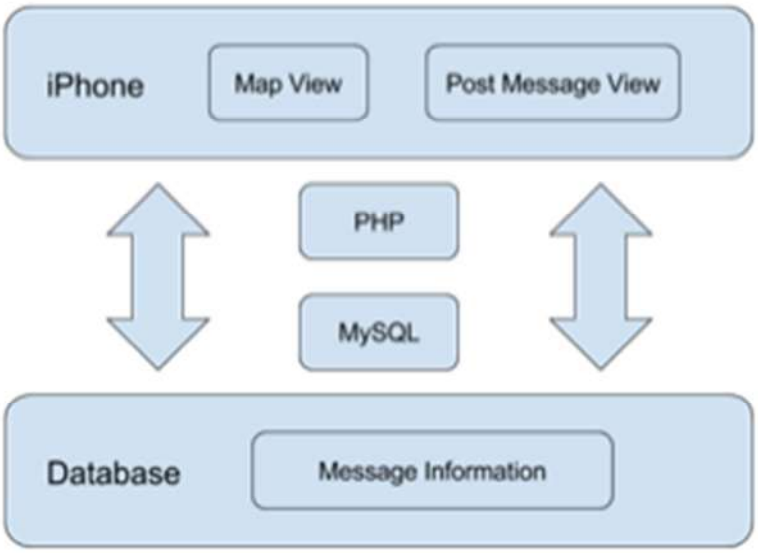
Our app has four major components: 1. Locate messages; 2. Post message; 3. Filter the messages; 4. Read the messages.

The message will locate in the place that the user visited such as lecture room, canteen, student hostel, etc. The user can leave message in the current location. To get the location of the users, the user has to switch on the GPS and blue-tooth for searching beacon before they post and read the messages. The app will record the latitude and longitude information with the direction together and send to the database.

4. MsgCraft Design

4.1 Server

4.1.1 Server-Side System Architecture



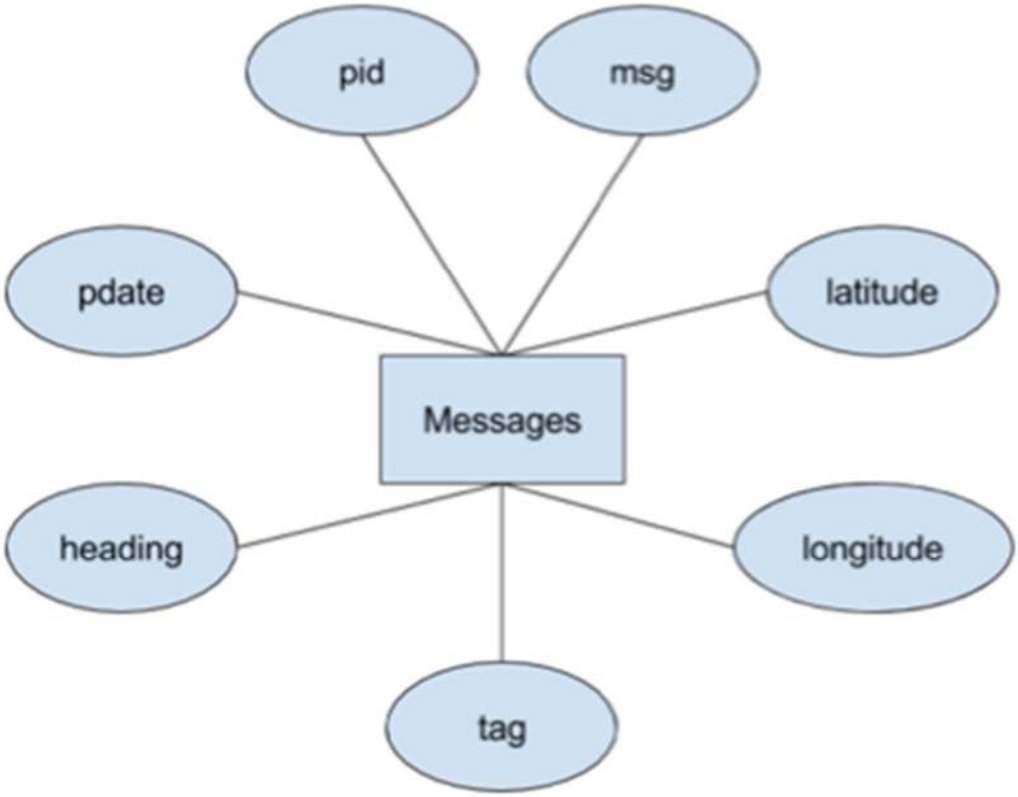
The above diagram is showing the server side system architecture. The iPhone application retrieves and uploads messages to database via PHP and MySQL. The data transfer is mainly the message content and message information.

4.1.2 Database

The database for this app at this stage is storing all the information related to the message and corresponding location information. Including message content and tag, position (longitude, latitude, heading), date of post. As the data is not complex, a single table is good enough to handle all the data.

In term 2, the database has to store six more attributes to differentiate beacons and record the orientation of the device. UUID, major and minor are used to identify beacons. Roll, pitch and yaw are used to indicate the orientation of the device as the message will be post along a 3D space.

4.1.3 Term 1 ER Diagram



The pid of each message is unique.

All message are identified by the pid.

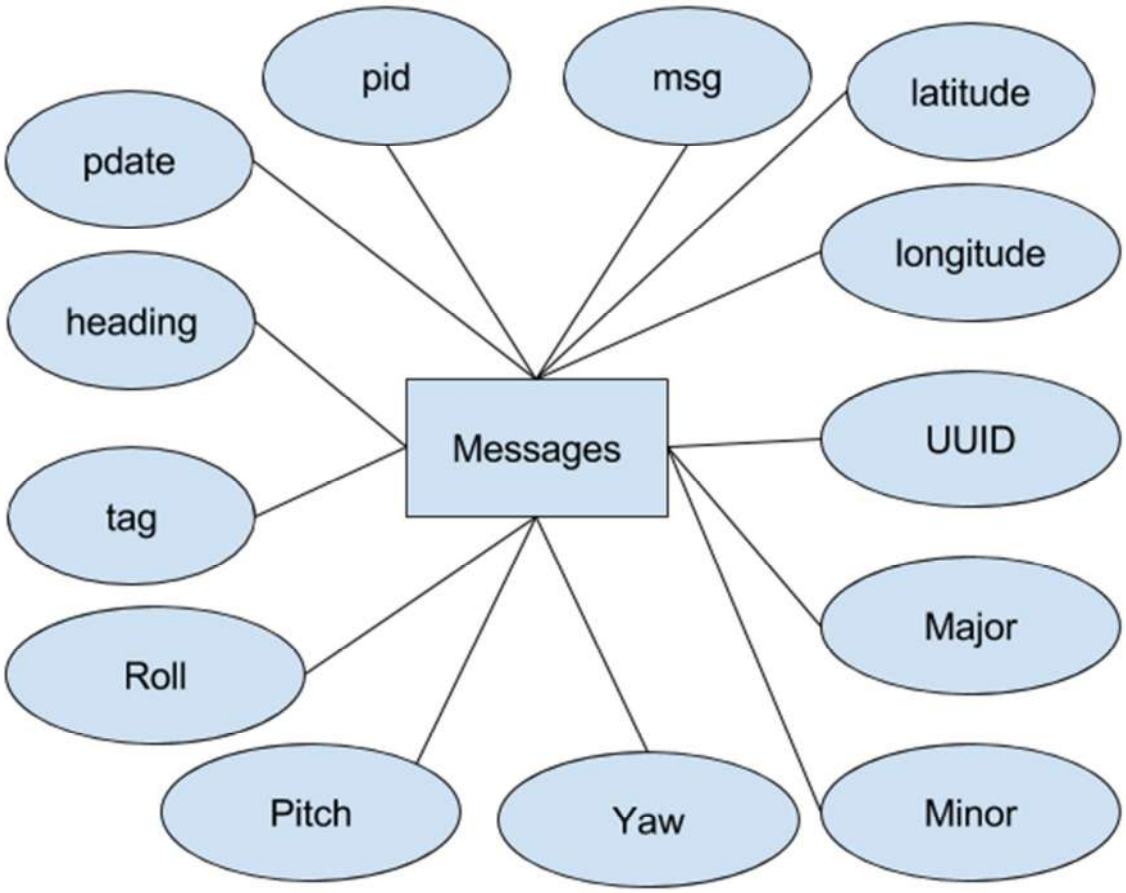
Besides pid, others attributes can be the same.

4.1.4 Term 1 Schema

There is a record for each message posted on virtual notice board. Each message has its pid, longitude, latitude, heading, message content, tag, date of post.

Attribute	Format	Description
pid	Non-empty positive integer	A unique identifier for message
longitude	Non-empty real number	The longitude of the message when it post
latitude	Non-empty real number	The latitude of the message when it post
heading	Non-empty real number	The heading of the message when it post
msg	Non-empty text	The message content
tag	Non-empty string with at most 80 character	The message tag
pdate	Non-empty data in the format of YYYY-DD-MM	The date of message post

4.1.5 Term 2 ER Diagram



The pid of each message is unique.

All messages are identified by the pid.

Besides pid, others attributes can be the same.

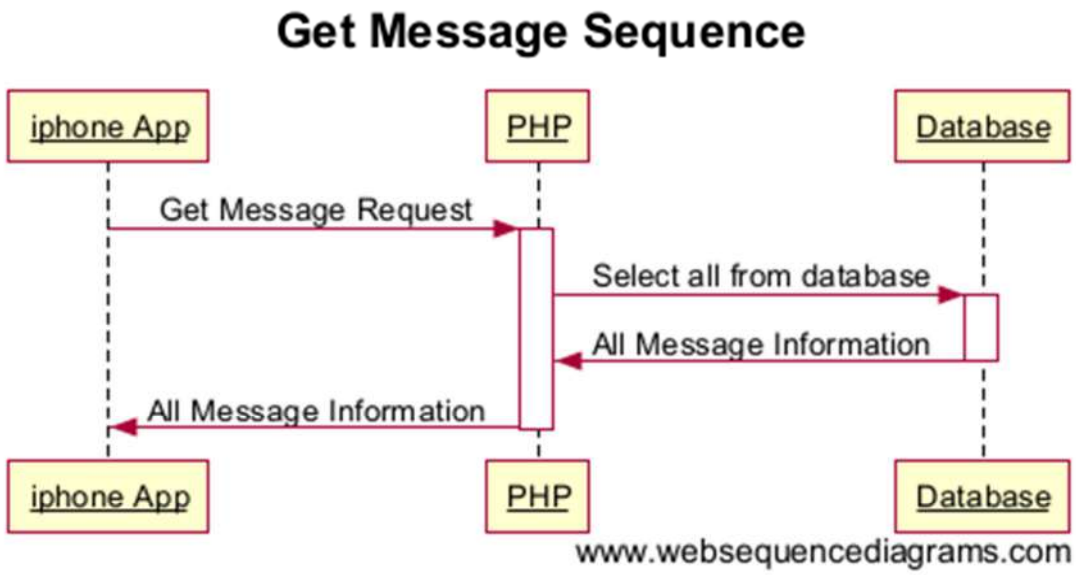
4.1.6 Term 2 Schema

There is a record for each message posted on virtual notice board. Each message has its own pid, longitude, latitude, heading, message, content, tag, date, UUID, major, minor, raw, pitch and yaw post.

Attribute	Format	Description
pid	Non-empty positive integer	A unique identifier for message
longitude	Non-empty real number	The longitude of the message when it post
latitude	Non-empty real number	The latitude of the message when it post
heading	Non-empty real number	The heading of the message when it post
msg	Non-empty text	The message content
tag	Non-empty string with at most 80 character	The message tag
pdate	Non-empty data in the format of YYYY-DD-MM	The date of message post

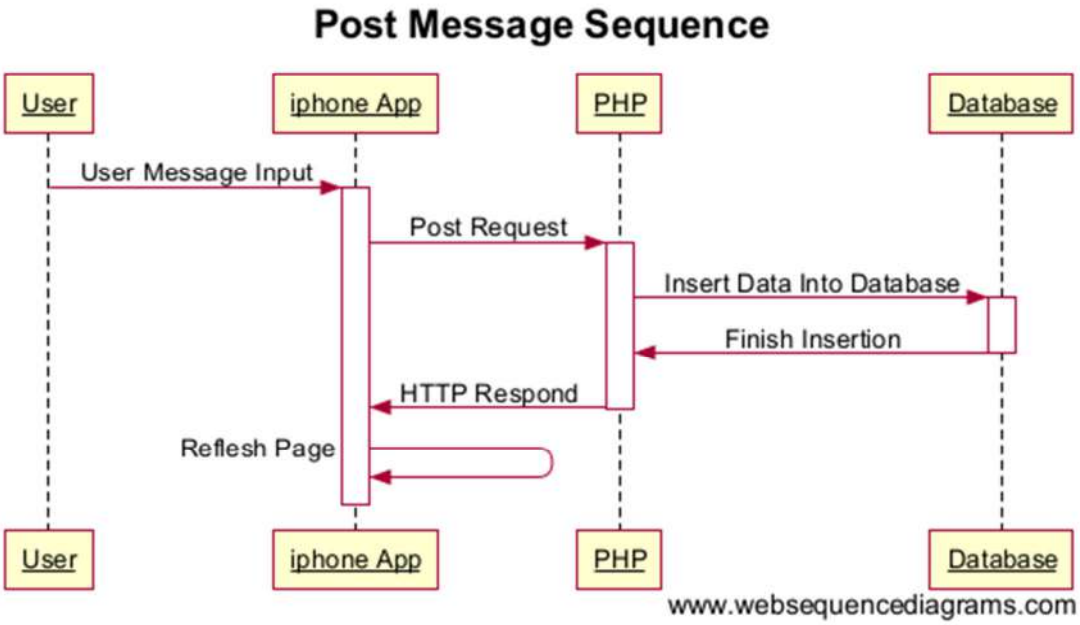
UUID	Non-empty string	The UUID of a beacon
Major	Non-empty positive integer	The major value of a beacon
Minor	Non-empty positive integer	The minor value of a beacon
Roll	Non-empty real number	The roll value of device orientation
Pitch	Non-empty real number	The pitch value of device orientation
Yaw	Non-empty real number	The yaw value of device orientation

4.1.7 Sequence Diagram



The above sequence diagram is showing how the message is get from the database. The app first sends a http request to server, via the PHP send a SQL command to database to retrieve all the message information. The messages will send back to app via the PHP by an http response. The above get message procedure will be invoked when map view is pull to the foreground.

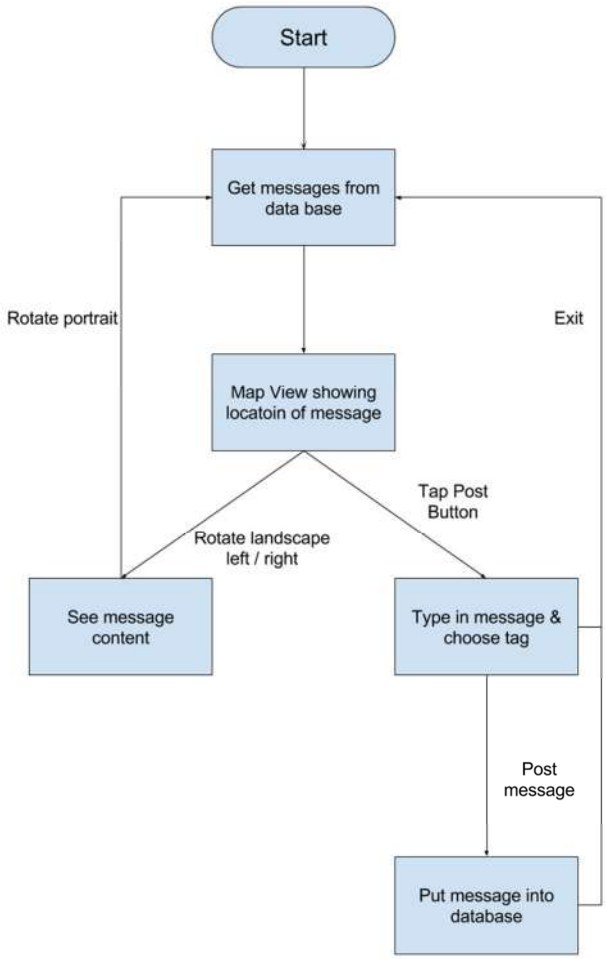
In term 2, the above sequence diagram is still used. Moreover, there will be 3 case to automatically invoke this procedure. A timer is added to invoke the get message procedure again, when there is no message received. Another case, is there is a change of beacons. The last case is user have move 50m away from the previous location. So that user does not need to keep on pressing the refresh button to check the existence of message. The message will be retrieved from time to time which is more user friendly.



The above sequence diagram is show how the message being post to database. User input message and choose tag, after pressing the submit button, the app send a http post request with message information to the server, via the PHP a SQL insert command is send to database. Then the server will send back http respond to indicate the message is successfully upload. The app then refreshes the page to map view, the get message procedure will immediately invoked to update message information.

4.2 Phone App

4.2.1 Term 1 Flow Chart



Mobile app may be suspended or interrupted by various event at any time, for example incoming phone, user pressing home button etc. In the above program flow diagram, the suspension and interruption problem is ignored. It only focuses on how the app works in foreground mode.

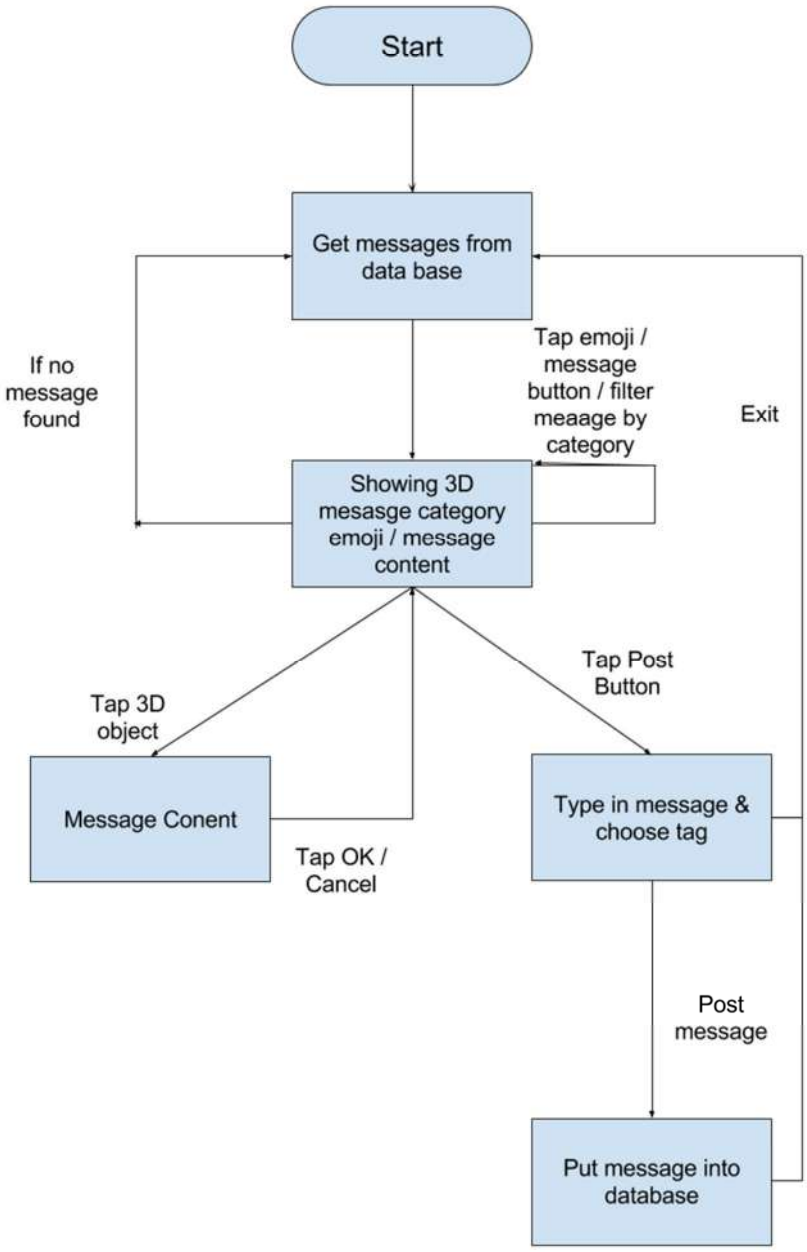
When the app begins, it get all the messages information from the database, then a map view with message location will shown. User can

see where are the message posts before, filter the messages with category.

When rotate to landscape mode, it goes the preview message mode users can see the filtered message content. User can also filter message at this mode by selecting time zone. When rotate back to portrait mode, the app make request to server to get update information.

In the map view, user can tap on post button to switch to post message view. User can go back to map view, by clicking cancel button. Or user type in the message, post to server, then get update the information from database and back to map view.

4.2.2 Term 2 Flow Chart



In term 2, the flow of the app has a few changes.

First, map view is deleted. After retrieving the message from database, it immediately showing the 3D message content or the 3D emoji to indicate the message type.

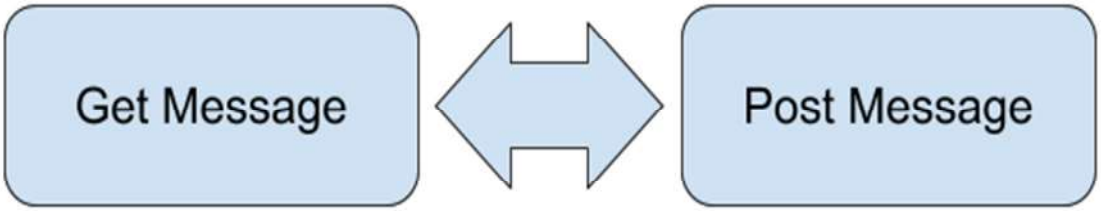
Second, the phone is fixed to landscape mode, there will be no change of view by rotation. All the views, will packed in a singe landscape view.

The rotation event will be replaced by a buttons. And the first incoming view is already the preview message view.

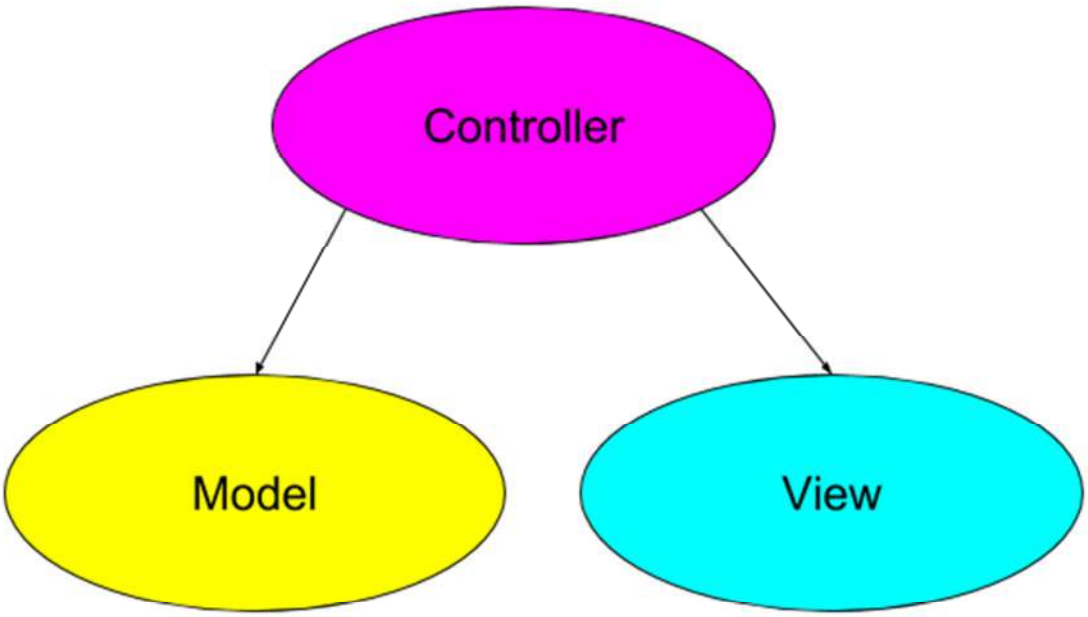
So now, after getting all the messages information from the database, then a preview message view will be shown. This view will be camera background with 3D object showing the content of messages or emoji indicating the category of messages. There will be a button to allow user to inter-switching the 3D emoji to 3D message content. Tap on the 3D object, a dialog box will be shown to show the full message content. Then, clicking the OK button will back to view message and the corresponding 3D object will be deleted. If cancel is clicked, it will back to view message but the corresponding 3D object will still appear. Moreover, if there are no messages retrieved, the preview messages view will still be shown. Then the get message from database procedure will be invoked from time to time in order to automatically get the message from database according to user location. So that the user does not need to click the refresh button again and again.

4.2.3 Modules

Basically there are two modules, get message and post message. Get message modules is getting messages from database and showing message to user. Post message is getting input from user and put into database. The whole program actually is the interchanging of these two modules.

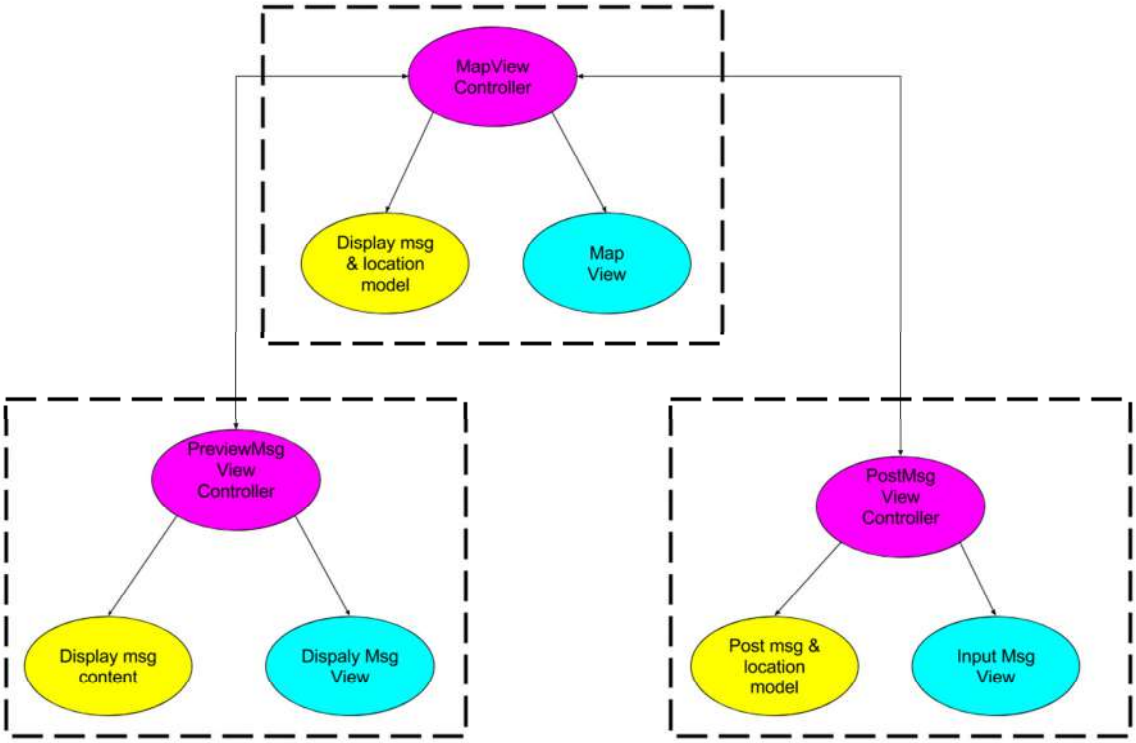


In IOS application development, the above modules can be further elaborate in a model-view-controller design pattern (MVC) which can much clearly show how the programs and the whole structure behind.



MVC is an object-oriented design pattern. Model object is defining the data and logic that manipulates that data. View object is presenting information and getting user input. Controller is performing set-up and coordinating tasks among model object and view object.

4.2.3.1 Term 1 MVC



The above diagram is showing the MVC model of our app. They are divided into three modules, Map View, Preview Message View and Post Message View. Each controller is having different or same model object to serve for the purpose of that view.

Switching amount screen display / view is actually switching from one controller to another controller. The switch mechanism in fact is a stack; the root view controller is the entry view that is the mapView.

Switching to another view is pushing the view controller into a stack. And only top element will be shown. Switch back to previous controller is pushing the top element out from the stack.

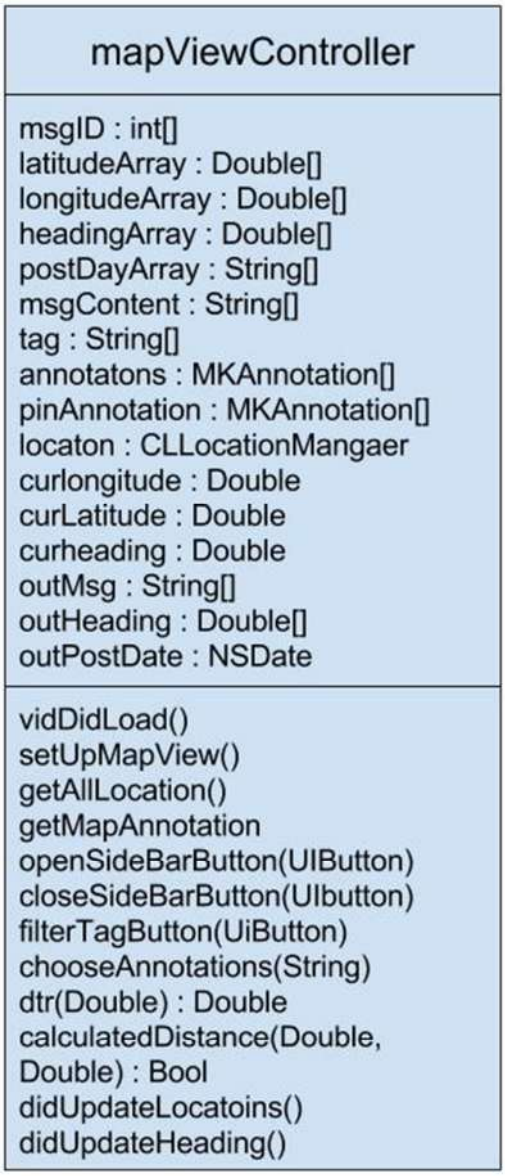
MapViewController is the entry of the app. It is a map view showing the location of messages. Behind the scene, it is getting message from database, storing the information of each data including the message identifier, content, position. Also, it allows user to filter message. At the same time, it is providing the location service to find the exact location of user and determine which message user is nearby.

PreviewMsgViewController is for displaying the message content. It contains function for filtering the message by a time selection bar and function to handle how the message content will be shown on screen.

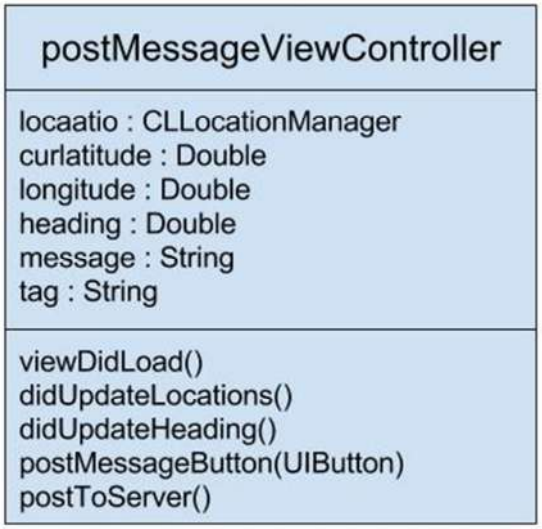
PostMsgViewController is for posting message to the database. User input the message content from view, via the controller the message is passed the post msg & location model to do the encoding with current location, then send to database.

4.2.3.2 Term 1 Class Diagram

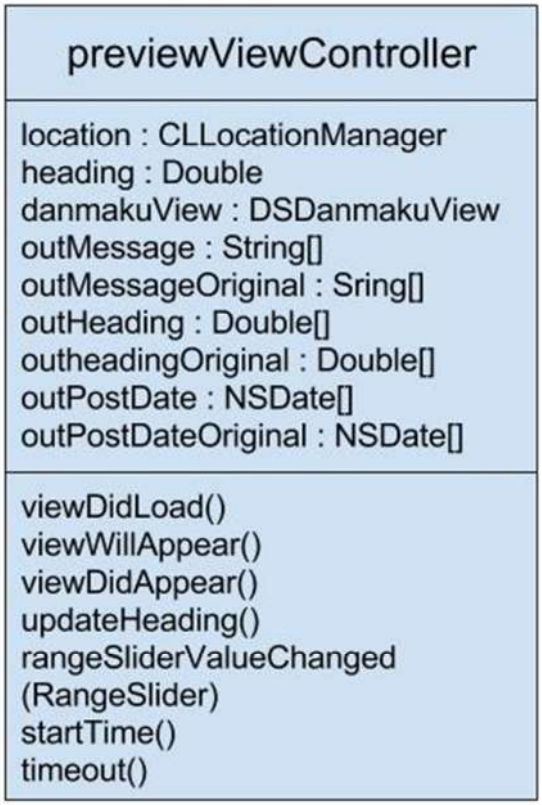
The class diagram of mapViewController



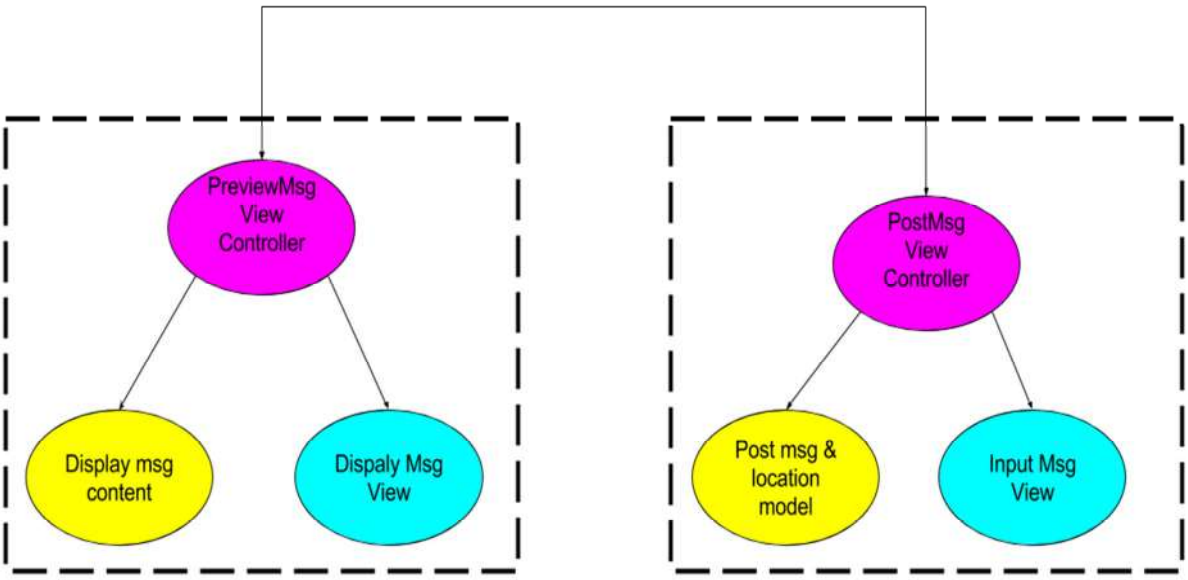
The class diagram of postMessageViewController



This is the class diagram of previewViewController



4.2.3.3 Term 2 MVC



The above diagram is showing the MVC model of term 2 app. The map view is deleted. There will be mainly two modules, Preview Message View and Post Message View.

Preview Message View will be the entry view of the app. It is showing the near by message with a 3D message content or 3D emoji. It first gets message from database, storing the information of each data and display suitable messages to users. It contains function for filtering the message by category, switching from viewing 3D message content or 3D emoji, showing which positioning system is using and providing orientation and location information of device, post messages to database.

Post Message View is for posting message to the database. After user input the message content and choosing the category. It will send those information to preview message view for uploading.

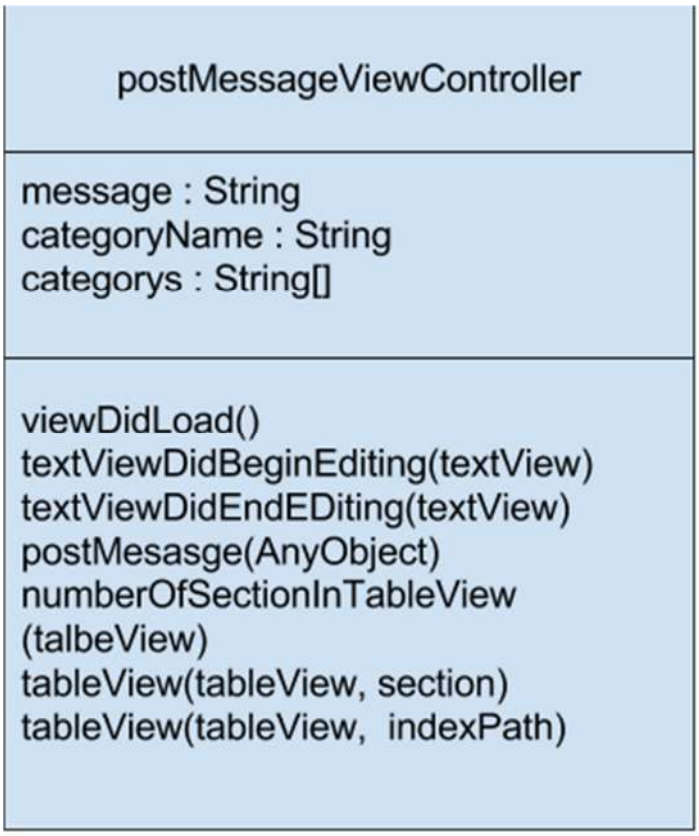
4.2.3.4 Term 2 Class Diagram

The class diagram of previewViewController

preViewMessageView	
currHeading : Double curLongitude : Double CurrrLatitude : Double currBeacon : CLBeacon currYaw : Double currPitch : Double currRoll : Double oldLat : Double oldLong : Double oldBeacon : CLBeacon firstHeading : Double counter : Integer messages : messageInfo[] messageInfo: ID : Integer latitude : Double longitude : Double tag : String content : String heading : Double data : String uuid : String major : Integer minor : Integer	roll : Double pitch : Double yaw : Double cameraNode : SCNode scene : SCNScene movementManager : CMMotionManager messageToPost : String messageCaegpry : String post3DSCNView : SCNView post3DScene : SCNScene
	randomColor() viewDidLoad() viewDidAppear() didReceiveMemoryWaring() supportedInterfaceOrientations() checkRefresh() initMenuBar() menuButtonTap(AnyObject) closeMenuButtonTap(AnyOjbect) buttonOn() captureButtonTap(AnyObject) changeMode(AnyObject) refresh(AnyObject) refreshView()

```
remvoeAllMessage()
postButtonTap(AnyObject)
initLocation()
locationManager(didUpdataLocation)
locationManager(didUpdatingHeading)
locatoinManager(didRangeBeacons)
getAllMessage()
filterButtonTap(AnyObject)
dtr(degree)
calculateDistance(lat1, lon1, lat2, lon2)
filterMessage()
initCamView()
init3DText()
renderMessageWithTitle()
renderMessageWithIcon()
handeTap(gestureRecognize)
initPost3D()
unwindForSegue(unwindSegue)
renderPost3D(messge, category)
postAction(UIButton)
postToServer()
corrHeading(currHeading)
```

The class diagram of postMessageViewController



5. MsgCraft Implementation

5.1 Server

5.1.1 Term 1 Retrieving messages

The app gets and retrieves message from database through PHP. First, it have to add a key in info.pist which require authorization from user to allow connection to internet as well as connect to database server.

▼ App Transport Security Settings	Dictionary	(2 items)
Allow Arbitrary Loads	Boolean	NO
▼ Exception Domains	Dictionary	(1 item)
▼ cse.cuhk.edu.hk	Dictionary	(3 items)
NSIncludesSubdomains	Boolean	YES
NSTemporaryExceptionAllow...	Boolean	YES
NSTemporaryExceptionMini...	String	TLSv1.1

To retrieve messages, an http request with post method is made to server.

```
let request = NSMutableURLRequest(URL: NSURL(string: "http://appsrv.cse.cuhk.edu.hk/~cklam4/getAll.php")!)
request.HTTPMethod = "POST"
let task = NSURLSession.sharedSession().dataTaskWithRequest(request) {
    data, response, error in
    if error != nil {
        print("error=\(error?.description)")
        return
    }
    let responseString = NSString(data: data!, encoding: NSUTF8StringEncoding)!
    let responseArray = responseString.componentsSeparatedByString("#")
```

The getAll.php is to get all the messages from the database. The retrieved information is stored in responseString and each information entry is separated by a special character “#”. So the actual data can be decoded more efficiently.

```php
<?php
    // MySQL connection
    $con=mysqli_connect("appsrvdb.cse.cuhk.edu.hk", "viewtech", "G5XYcFhy", "viewtech");
    if (mysqli_connect_errno ($con))
    {
      echo "Failed to connect to MySQL: " . mysqli_connect_error();
    }

    $sql = "SELECT * FROM lyu1504_stdplace";
    $result = mysqli_query($con, $sql);
    //search for region
    if(mysqli_num_rows($result) > 0){
      while($row = mysqli_fetch_assoc($result)){
        echo $row["pid"] . "#" . $row["latitude"] . "#" . $row["longitude"] . "#" . $row["tag"] .
        "#" . $row["msg"] . "#" .
         $row["heading"] . "#" . $row["pdate"] . "#";
            }
    }
    else{
      echo "none rows";
    }

    mysqli_close($con);
?>
```

The getAll.php first make a connection to the database, and then create a SQL command to get all the data from database. As shown in the above coding, each data is separated by “#” and concatenate as a long string and send back to the app.

5.1.2 Term 1 Post Message

In order to post a message to server, another http request has to be made to server.

```
let todaysDate:NSDate = NSDate()
let dateFormatter:NSDateFormatter = NSDateFormatter()
dateFormatter.dateFormat = "yyyy-MM-dd"
let DateInFormat:String = dateFormatter.stringFromDate(todaysDate)

//print(DateInFormat)

let request = NSMutableURLRequest(URL: NSURL(string: "http://appsrv.cse.cuhk.edu.hk/~cklam4/
    msg_post02.php")!)
request.HTTPMethod = "POST"
let postString = "lat=\(latitude)&long=\(longitude)&msg=\(message)&tag=\(userInputTag.titleForSegmentAtIndex
    (userInputTag.selectedSegmentIndex)!)&heading=\(heading)&date=\(DateInFormat)"

request.HTTPBody = postString.dataUsingEncoding(NSUTF8StringEncoding)
```

After user tap the post, all of the information about this message will be encoded into a single string postString, and send to server via http.

```
echo $date;

$time = strtotime($date);
$date = date('Y-m-d', $time);

$sql = "INSERT INTO lyu1504_stdplace (latitude, longitude, name, msg, tag, heading, pdate) VALUES
 ('".$latitude."','".$longitude."', 0, '".$message."','".$tag."','".$heading."','".$date."')";

echo $date;

mysqli_query($con, $sql);

mysqli_close($con);
?>
```

```php
<?php
// MySQL connection
$con=mysqli_connect("appsrvdb.cse.cuhk.edu.hk", "viewtech", "G5XYcFhy", "viewtech");
if (mysqli_connect_errno ($con)){
 echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

$latitude = $_POST["lat"];
$longitude = $_POST["long"];
$heading = $_POST["heading"];
// $name = $_GET["name"];
$message = $_POST["msg"];
$tag = $_POST["tag"];
$date = $_POST["date"];

echo $date;

$time = strtotime($date);
$date = date('Y-m-d', $time);

$sql = "INSERT INTO lyu1504_stdplace (latitude, longitude, name, msg, tag, heading, pdate) VALUES
 ('".$latitude."','".$longitude."', 0, '".$message."','".$tag."','".$heading."','".$date."')";
```

The msg_post02.php first connect to the database, and then start decoding the http body, which is the postString. After that, it saves the message information to the corresponding variables. Then make a SQL insert command with these variables.

5.1.2 Term 2 Retrieving messages

The info.pist and http request is the same in term 2. No changes is needed for this two part. As well as the getAll.php, we simple retrieve all the information back to the device it self. So no changes is needed for this part also.

5.1.3 Term 2 Post message

The http request and php has to be changed because the message information is increased due to beacon information and device orientation, corresponding changes to the postString and query statement is needed.

```
let request = NSMutableURLRequest(URL: NSURL(string: "http://appsrv.cse.cuhk.edu.hk/~cklam4/msg_post02.php")!)
request.HTTPMethod = "POST"
let heading = corrHeading(locationInfo.currHeading)

var postString = ""
if locationInfo.currBeacon == nil{
    postString = "lat=\(locationInfo.currLatitude)&long=\(locationInfo.currLongitude)&msg=\(messageToPost)&tag=\
        (self.messageCategory)&heading=\(heading)&date=\(DateInFormat)&uuid=0&major=0&minor=0&title=0&roll=\(motionInfo.currRoll)&pitch=\
        (motionInfo.currPitch)&yaw=\(motionInfo.currYaw)"
}
else{
    postString = "lat=0&long=0&msg=\(messageToPost)&tag=\(self.messageCategory)&heading=\(heading)&date=\(DateInFormat)&uuid=\
        (locationInfo.currBeacon!.proximityUUID.UUIDString)&major=\(locationInfo.currBeacon!.major)&minor=\
        (locationInfo.currBeacon!.minor)&title=0&roll=\(motionInfo.currRoll)&pitch=\(motionInfo.currPitch)&yaw=\(motionInfo.currYaw)"
}
```

The request function need to check determine which positioning system is using by checking is there any existence beacon. By default, the app will adopt GPS, when there is beacon signal, it indicates it is in indoor or at specific location, then it will choose the closest beacon.

Then according to each positioning system, corresponding postString content need to be change.

If GPS is used, the value of uuid, major and minor need to set to zero to indicate it is GPS based message.

If beacon is used, the value of longitude and latitude need to set to zero to indicate it beacon based message.

5.2 Term 1 IPhone App

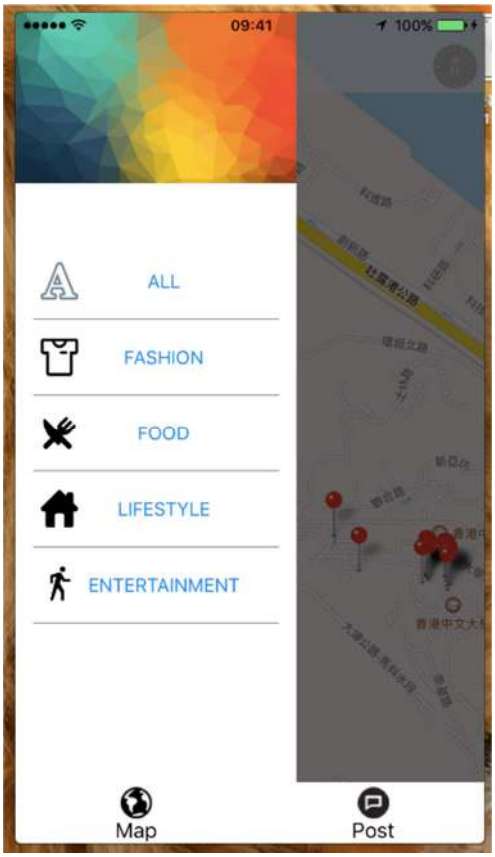
5.2.1 UI Design

5.2.1.1 Map View Annotations

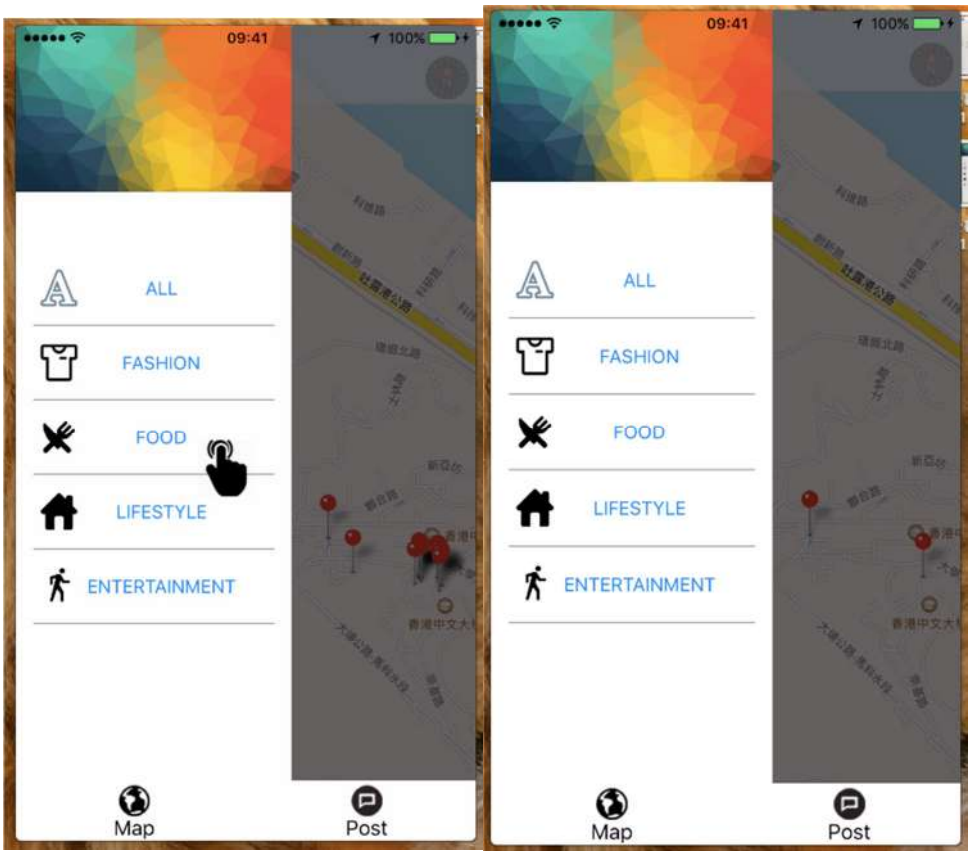
The map view is the main view of the app. It is the first page we can see when user open our app. It will show a map of user's current location. If there is a message, the map will display annotations that indicate the actual position of that message. Also, the user can know which type the message is by tapping on the annotation. There will be a pop up remark for the category of the message and the date of the message being post.

5.2.1.2 Map View Category Menu

When there are a huge number of messages. It is hard for the user to search message that they want to see. We have divided the messages into four category, fashion, food, lifestyle and entertainment. Each message has its own message type called tag. The app has a filtering function to help the user search for the message that they want to see at a quick manner. For the filtering process, the users have to tap the menu icon on the top left corner. And the category menu will pop up.



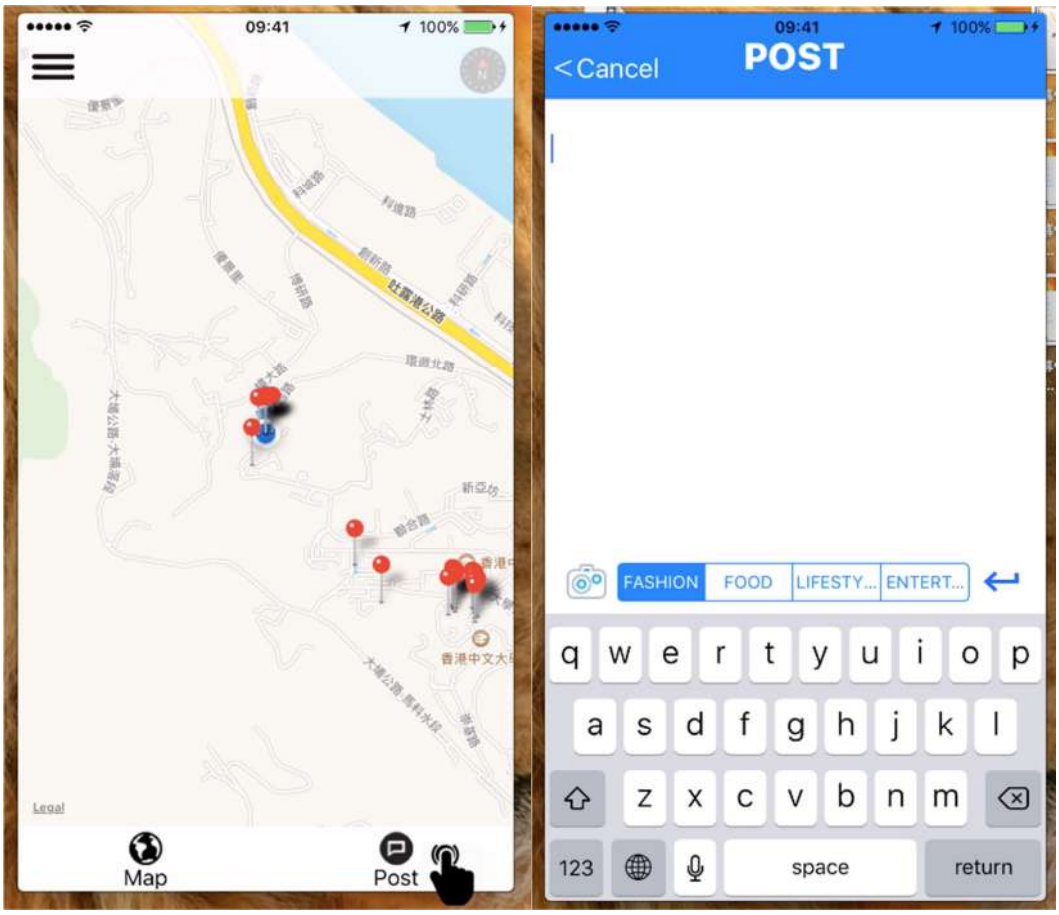
When the menu bar pops up, the background will be darker than the normal in order to make the sense of multi-level. And when the users tap on the dark region, the app will return to the map view.



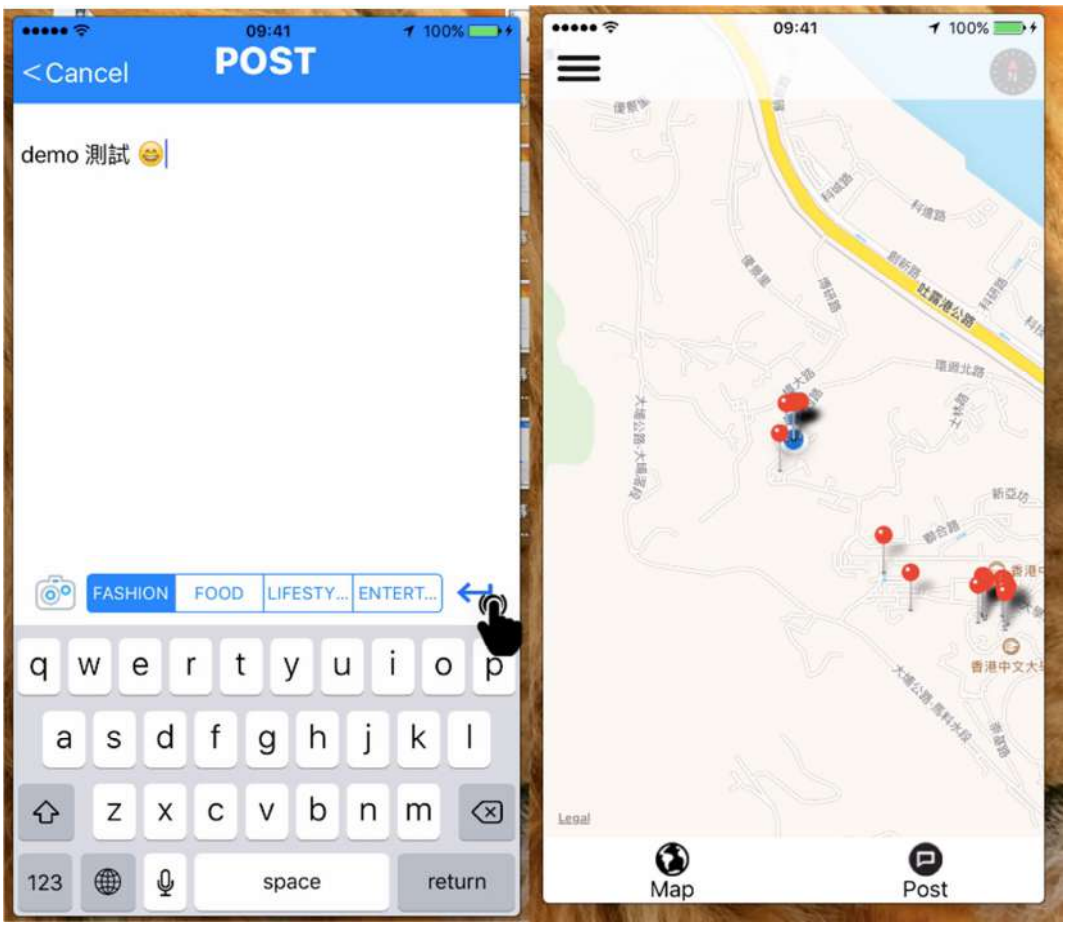
There are four categories for the messages, fashion, food, lifestyle and entertainment. User can choose a category message they are interested in.

As shown in the above diagram, it shows all kind of messages by default. After pressing, FOOD button, the number of annotation on the map is less than before, as it only shows the messages in food category.

5.2.1.3 Post Message View



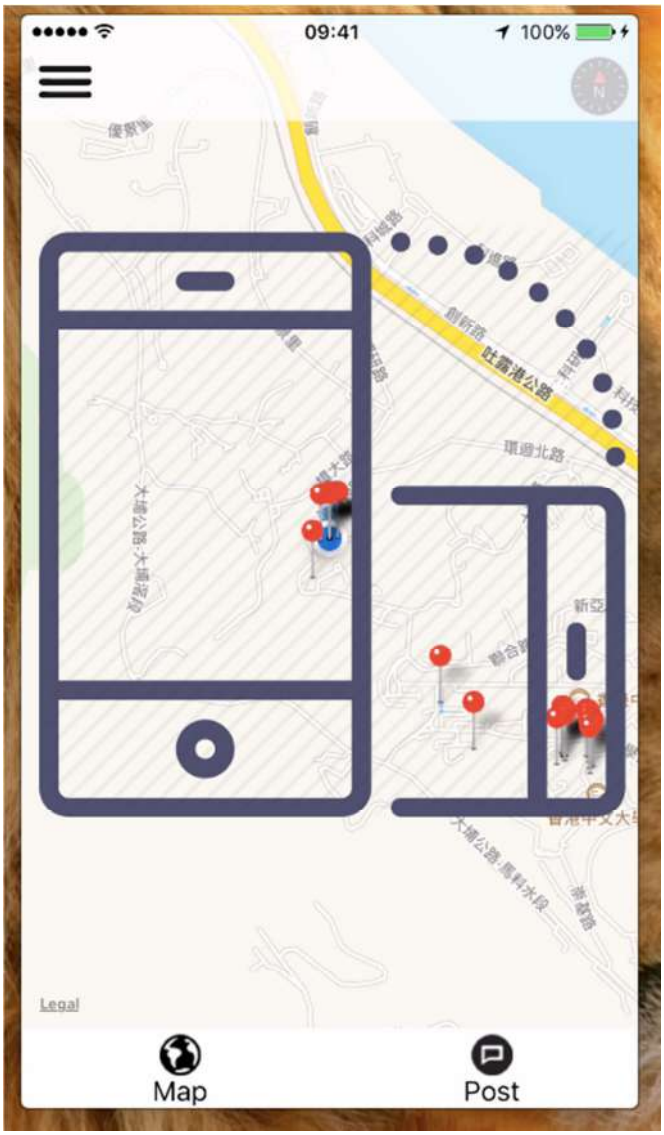
On map view, there is a Post button. By clicking the post button, it will switch to postMsgView, user can write their comment about this place at the postMsgView.



At the postMsgView, there is a white text field allow user to type in message content. The segment right below is to choose the tag category that is related to their message content. After finish typing, click on the enter button on the right hand side, the message will be uploaded to server and route back to mapView. The message just post will also be displayed on the map view.

Of course, user may want to get back to map view without posting anything, there is a cancel button at the top left hand corner to route to map view.

5.2.1.4 Message Display View



How to read a message is a very important objective of our project. In many existing messenger apps, almost all of them use the same way for message reading. It provides a table of messages. The users have to click on each message to see the details. We think it is a boring way for message checking using in a notice board. Our goal is to make a new and creative UI design. We have designed a fantastic way for the messages watching. The users just need to rotate their phone and scan the message out.

This is the message checking view. We use a camera view as a background. In the above example, we have left a message on the table and the message content is a crying emoji that expressing my feeling about too many things are on my desk. Then my roommate can read this crying face by rotating his phone in landscape mode and let the camera point to my desk. This is a new way for reading message that can give the user the new experience when they use the app. Also, then is an interesting method to show to messages. The message will keep moving from the right hand side to the left hand side in order to make a great effect.

5.2.1.5 Range slider bar for selecting time period

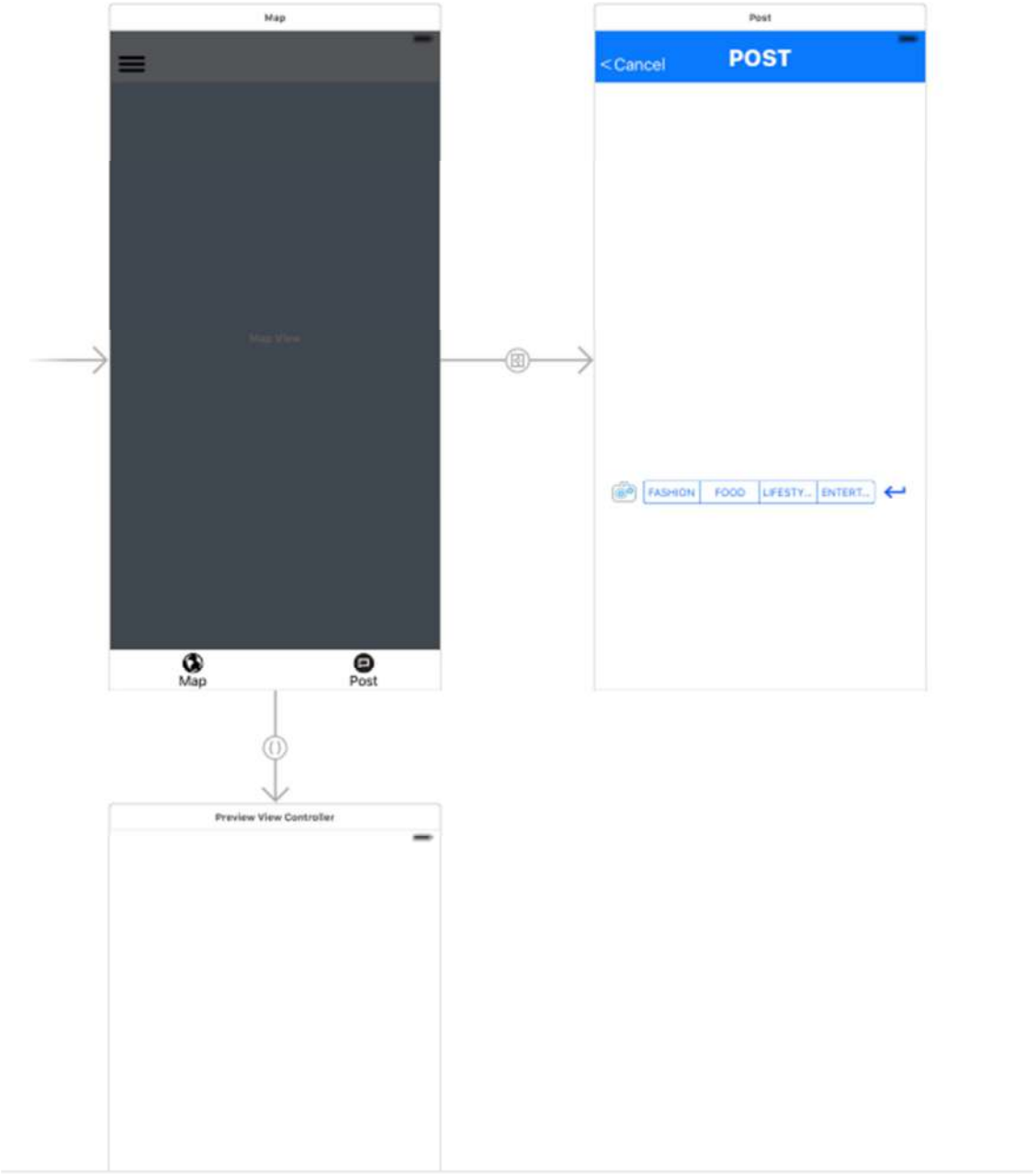
At the bottom of the message checking view, there is a range slider bar. Users can use that 2-sided slider bar to select the messages that post on that interval. The above example shows that i have select all the messages between today to 1 month ago. The camera shows the crying emoji that is the content of message 1 post the week before.

This is the second example. It shows that the user change the time interval, but still at the same position. The message checking view won't show the crying message. Instead of the crying message, it shows "There is no message" to indicate that there is no message in that particular time interval.

5.2.1.6 Message Display

The message will appear only when the user is at the right position and face with the right angle. When the user is standing in the right position but wrong angle, the view will not show the message. There will be two arrow at the left and right side to remind to user to turn around their phone, point it to the right angle to grep the messages.

5.2.2 Modules



By following the MVC discussed before, there are mainly 3 modules, the MapViewController, PostMsgViewController and PreviewViewController. The location service is repeatedly used among the three controllers. So the implementation of location service function will be discussed first.

5.2.2.1 *Implementation of Location Services:*

In IOS, there is a API CoreLocation service which provide most of the functions and methods for building a location-based application. For example, it provides GPS location services, region monitoring of a defineded location, the direction of the device pointing to. This app requires all of these services in order to get the accurate location for user to post or read a message in a specific region.

To use this API, first set up CLLocationManager class instance that is the central part for configuring the location related events. it is used to control the start and stop of delivery of location events. Moreover, it is used to retrieve most of the location data. Another function of CLLocationManager is to request the authorization to activate the GPS function of the iPhone from user and a key in the info.pist should also be added to get the usage of GPS.

NSLocationAlwaysUsageDescription	String	GPS

```
var location = CLLocationManager()
```

```
override func viewDidLoad() {
    super.viewDidLoad()
    // Do any additional setup after loading the view.
    NSNotificationCenter.defaultCenter().addObserver(self, selector: "rotated", name: UIDeviceOrientationDidChangeNotification, object: nil)

    mapView.delegate = self

    self.location.requestAlwaysAuthorization()
    self.location.delegate = self
    self.location.desiredAccuracy = kCLLocationAccuracyBestForNavigation
    self.location.startUpdatingLocation()
    self.location.startUpdatingHeading()

    closeSideBarOutlet.hidden = true
    setUpMapView()
}
```

5.2.2.2 Implementation of CLLocation and CLHeading:

A CLLocation object represents the location data generated by a CLLocationManager object. After implemented CLLocationManager, CLLocation can be implemented to get the actually location of the users in longitude and latitude by writing the event handler didUpdateLocations() to retrieve position data. It will be invoked whenever there is a slightly change in location as we have set the accuracy to the highest level via the CLLocatinManager. The CLLocation works really well, it locates user position with 10-meter variants and with a very fast respond to any changes in location.

```
func locationManager(manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {
    print(count++)
    print("locaton update")
    latitude = location.location!.coordinate.latitude
    longitude = location.location!.coordinate.longitude
    print("latitude = \(latitude)")
    print("longtitude = \(longitude)")
}
```

CLHeading is also implemented in a same way as CLLocation. didUpdateHeading() is to retrieve heading data that is computed values for magnetic north. It works extremely sensitive to slightly changes in direction of the device pointing to.

```
func locationManager(manager: CLLocationManager, didUpdateHeading newHeading: CLHeading) {
    headingInfo.currHeading = location.heading!.magneticHeading
}
```

5.2.2.3 Implementation of Longitude-Latitude Distance

To resolve the problems of declaring a region and have quick respond in notifying user's region. We decided to implement a longitude-latitude distance calculator. When message's location information is get updated, this calculator will be invoked and distance of user current location to each messages will be calculated immediately. Message with distance within the certain radius will be shown, the others will be ignored.

The equation we use for the calculator is Haversine fomula:

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

where

- *hav* is the haversine function:

$$\operatorname{hav}(\theta) = \sin^2\left(\frac{\theta}{2}\right) = \frac{1-\cos(\theta)}{2}$$

- *d* is the distance between the two points (along a great circle of the sphere; see spherical distance),
- *r* is the radius of the sphere,
- ϕ_1, ϕ_2: latitude of point 1 and latitude of point 2
- λ_1, λ_2: longitude of point 1 and longitude of point 2

Haversine formula is used to calculate the great-circle distance between two points by assuming the earth is spherical with radius 6371 km. Surely it has error due to the assumption (earth is very slightly ellipsoidal), but the accuracy is good enough for our application.

```
func dtr(degree:Double)->Double{
    return degree / 180 * M_PI
}

func calculateDistance(lat1 : Double, lon1 : Double) -> Bool{
    let R = 6371000.0 // metres
    let lat2 = latitude
    let lon2 = longitude

    let rlat1 = dtr(lat1)
    let rlat2 = dtr(lat2)
    let dlat = dtr(lat2-lat1)
    let dlon = dtr(lon2-lon1)

    let a = sin(dlat/2) * sin(dlat/2) + cos(rlat1) * cos(rlat2) * sin(dlon/2) * sin(dlon/2)
    let c = 2 * atan2(sqrt(a), sqrt(1-a))
    //var d = R * c;
    let d = R * c
    print(d)
    if d < 10{
        return true
    }else{
        return false
    }

}
```

calculateDistance() is implementation of Haversine formula programmatically , lat1 and lon1 is representing latitude, longitude of a message location, lat2 and lon2 is representing latitude, longitude of user current location. If the distance is within 10 meter, it returns true indicates it's within a region and message of that particular location will shown, otherwise return false and message of that location will be ignored.

5.2.2.4 Implementation of MapViewController.

MapViewController main function is to get the message information from database and show the location of each message on a map view. Besides this, it also responsible to gain the authorization of GPS usage and Internet access from user at the early beginning of the app. Moreover, it has to calculate user current position to the message and do filtering to determine which message will be displayed.

getalllocatons() is function for getting messages from database. The data get from the database will be stored in responseString and each data entry will be separated by "#" character. So by tokenizing the string with "#" character, all of the message information can be retrieved.

```
func getAllLocations(){
    print("***in getalllocations")

    let request = NSMutableURLRequest(URL: NSURL(string: "http://appsrv.cse.cuhk.edu.hk/~cklam4/getAll.php")!)
    request.HTTPMethod = "POST"
    let task = NSURLSession.sharedSession().dataTaskWithRequest(request) {
        data, response, error in
        if error != nil {
            print("error=\(error?.description)")
            return
        }
        let responseString = NSString(data: data!, encoding: NSUTF8StringEncoding)!
        let responseArray = responseString.componentsSeparatedByString("#")

        print("***responseArray")
        print(responseArray)

        for var i = 0 ; i < responseArray.count-1 ; i++ {
            if(i%7 == 0){
                self.msgID.append(Int(responseArray[i])!)
            }
            else if(i%7 == 1){
                self.latitudeArray.append(Double(responseArray[i])!)
            }
            else if(i%7 == 2){
                self.longitudeArray.append(Double(responseArray[i])!)
            }
            else if(i%7 == 3){
                self.tag.append(String(responseArray[i]))
            }
            else if (i%7 == 4){
                self.msgContent.append(String(responseArray[i]))
            }
            else if (i%7 == 5){
                self.headingArray.append(Double(responseArray[i])!)
            }
            else {
                self.postDateArray.append(String(responseArray[i]))
            }
        }

        self.annotations = self.getMapAnnotation(self.latitudeArray, long: self.longitudeArray, tag: self.tag,
            date: self.postDateArray)
        self.pinAnnotations = self.annotations

        let dateFormatter = NSDateFormatter()
        dateFormatter.dateFormat = "yyyy-MM-dd"

        for var i = 0 ; i < self.postDateArray.count ; i++ {
            self.formattedPostDateArray.append(dateFormatter.dateFromString(self.postDateArray[i])!)
        }
        print(self.postDateArray)
        print(self.formattedPostDateArray)

            self.mapView.addAnnotations(self.pinAnnotations)

        }
        task.resume()
        print("***after resume")
}
```

After retrieving each message information, every data entry will pass to getMapAnnotation() to prepare annotate that will show on map view. The location information is need for pinning the annotation. Also, in order to let the map view more interactive, message tag, message post

date is added as title and subtitle of annotation. When users tap on annotation, the title and subtitle will shown. After looping through the entire message, the annotation will be add to the map view, and location of each message can be clearly shown.

```
func getMapAnnotation(lat:[Double], long:[Double], tag:[String], date:[String]) -> [MKAnnotation]{
    var annotations = [MKAnnotation]()

    for var i = 0; i < lat.count; i++ {
        let annotation = MKPointAnnotation()
        annotation.coordinate = CLLocationCoordinate2DMake(lat[i], long[i])
        annotation.title = tag[i]
        annotation.subtitle = "From: \(date[i])"
        annotations.append(annotation)
    }

    return annotations
}
```

There is a sidebar in MapView as mention before to let user choose what category of message they want to display on map, in fact, it is remove and add of annotation.

```
//MARK: Choose Tag
@IBAction func filterTagButton(sender: UIButton) {
    switch sender.currentTitle!{
    case "ALL":
        mapView.removeAnnotations(pinAnnotations)
        pinAnnotations = annotations
        choosenTag = "ALL"
        mapView.addAnnotations(pinAnnotations)
    case "FASHION":
        mapView.removeAnnotations(pinAnnotations)
        pinAnnotations.removeAll()
        choosenTag = "FASHION"
        chooseAnnotations(choosenTag)
        mapView.addAnnotations(pinAnnotations)
    case "FOOD":
        mapView.removeAnnotations(pinAnnotations)
        pinAnnotations.removeAll()
        choosenTag = "FOOD"
        chooseAnnotations(choosenTag)
        mapView.addAnnotations(pinAnnotations)
    case "LIFESTYLE":
        mapView.removeAnnotations(pinAnnotations)
        pinAnnotations.removeAll()
        choosenTag = "LIFESTYLE"
        chooseAnnotations(choosenTag)
        mapView.addAnnotations(pinAnnotations)
    case "ENTERTAINMENT":
        mapView.removeAnnotations(pinAnnotations)
        pinAnnotations.removeAll()
        choosenTag = "ENTERTAINMENT"
        chooseAnnotations(choosenTag)
        mapView.addAnnotations(pinAnnotations)
    default:break
    }
    print(sender.currentTitle)
}

private func chooseAnnotations(chooseTag: String){
    for(var i = 0 ; i < annotations.count ; i++){
        if(tag[i] == chooseTag){
            pinAnnotations.append(annotations[i])
        }
    }
}
```

We first remove all the annotations on the map, then sort out the required by loop through the downloaded information again. Finally, add the required annotation back on map.

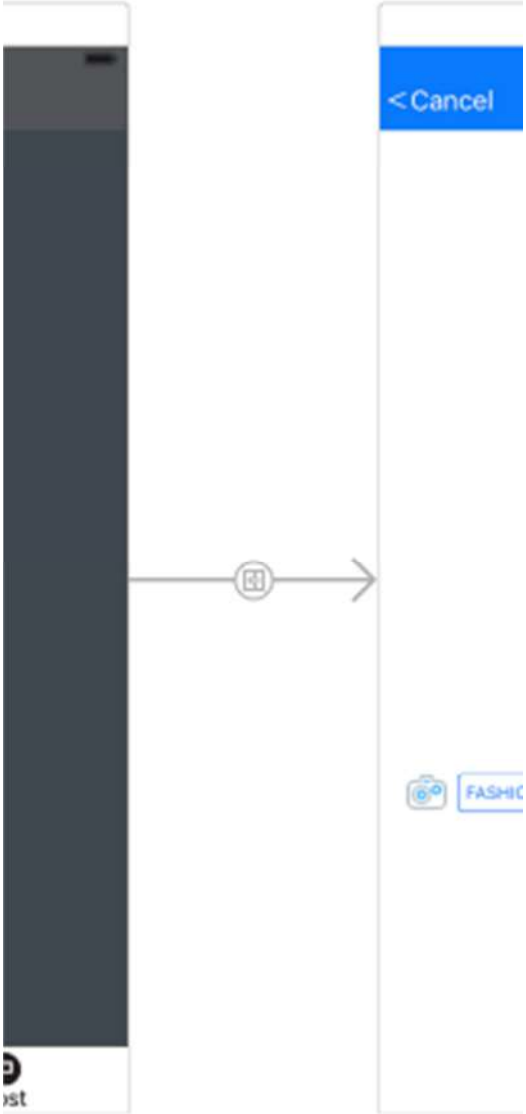
5.2.2.5 Implementation of postMsgViewController

postMsgViewController is a view for user to post message together with current location and heading information. It has implemented location service as mentioned before.

```
@IBAction func postMessageButton(sender: UIButton) {
    message = userInputText.text.stringByAddingPercentEncodingWithAllowedCharacters(NSCharacterSet.
        URLQueryAllowedCharacterSet())!
    print("detail of the")
    print("latitude = \(latitude)")
    print("longtitude = \(longitude)")
    print("content = \(message)")
    postToServer()

    performSegueWithIdentifier("exitPost", sender: nil)

}

func postToServer(){

    //get the current day
    let todaysDate:NSDate = NSDate()
    let dateFormatter:NSDateFormatter = NSDateFormatter()
    dateFormatter.dateFormat = "yyyy-MM-dd"
    let DateInFormat:String = dateFormatter.stringFromDate(todaysDate)


    let request = NSMutableURLRequest(URL: NSURL(string: "http://appsrv.cse.cuhk.edu.hk/~cklam4/msg_post02.php")!)
    request.HTTPMethod = "POST"
    let postString = "lat=\(latitude)&long=\(longitude)&msg=\(message)&tag=\(userInputTag.titleForSegmentAtIndex
        (userInputTag.selectedSegmentIndex)!)&heading=\(heading)&date=\(DateInFormat)"

    request.HTTPBody = postString.dataUsingEncoding(NSUTF8StringEncoding)
    let task = NSURLSession.sharedSession().dataTaskWithRequest(request) {
        data, response, error in

        if error != nil {
            print("error=\(error?.description)")
            return
        }

         let responseString = NSString(data: data!, encoding: NSUTF8StringEncoding)
        print("*****************@@@@@@@@*************")
        print(responseString)
    }
    task.resume()

}
```

When user click on post button, postToServer() function is invoked. It first get message content from the text field and tag from title segment, current position information. Then combine all these information into a string format and update to server.

5.2.2.6 implementation of segue



A segue in swift is for view transection. It controls how a user can go from one view to another view. In swift, it can be setup by just click and drag. It also allow programmer customizing his own segue. The default segue is not power enough for our use, so we have to implement the segue function by our own method instead of the given one.

```
import UIKit

class mySegue: UIStoryboardSegue {
    override func perform() {
        let sourceVC = self.sourceViewController
        let destinationVC = self.destinationViewController
        destinationViewController.view.tag = 100
        sourceVC.view.addSubview(destinationVC.view)
        print("add camera")
    }
}
```

mySegue class is putting a new sub-layer above the current layer. We implement it when we rotate our phone. This action will trigger the segue and bring the user from the mapView to previewView.

```
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if let identifier = segue.identifier{
        switch identifier{
        case "rotateSegue":
            if let vc = segue.destinationViewController as? previewViewController{
                vc.outMessage = outMsg
                vc.outMessageOriginal = outMsg
                vc.outHeading = outHeading
                vc.outHeadingOriginal = outHeading
                vc.outPostDate = outPostDate
                vc.outPostDateOriginal = outPostDate

                outMsg.removeAll()
                outHeading.removeAll()
                outPostDate.removeAll()
                  vc.message = msgContent
                  outMsg = ""
            }
        default: break
        }

    }
```

The function of prepareForSeque() is work before we transit from the mapView to the previewView. It prepare the datas that we want to pass to previewView and let us use it in the previewView.

5.2.2.7 Implement of Danmaku(Message Display)

The function addDanmakus() get an array of message as input. Then divide it one by one.

```
func addDanmakus(textsArray:[String], attrbute:DSSanmakuAttribute) {
    for text in textsArray {
        self.addDanmaku(text, attribute:attrbute)
    }
}
```

The function addDanmaku() get a message as input. Then it will count the available position for the message to avoid multiple messages appear on the same level causing overlapping.

```
func addDanmaku(text:String, attribute:DSSanmakuAttribute) {
    let danmakuLabel = DSDanmakuLabel(danmakuText:text, attribute:attribute

    let point = self.countAvailablePositionForDanmaku(danmakuLabel)
    danmakuLabel.setPosition(point)
    danmakuLabel.calculateReaminTime((Double)(CGRectGetWidth(self.frame)))

    self.addSubview(danmakuLabel)
    self.playDanmaku(danmakuLabel)
    self.addMovingDanmaku(danmakuLabel)
}
```

The above function set all the displaying condition. The function playDanmaku() is enable displaying the message. When finish playing the message, it will remove the message from the array to avoid repeating display the same message

```
func playDanmaku(danmaku:DSDanmakuLabel) {
    weak var wself = self
    danmaku.startPlay({(complete:Bool) -> Void in
        if (complete) {
            danmaku.removeFromSuperview()
            if let index = wself!.movingDanmaku.indexOf(danmaku){
                wself!.movingDanmaku.removeAtIndex(index)
            }
        }
    })
}
```

The function countAvailablePositionForDanmaku() is the function that finding the available position for displaying message. Our algorithm is randomly get a position first. Then check this position to see whether it is occupied by another message or not. If the position is free, then accept. If it is not free, then generate a new position and check for the availability again.

```
func countAvailablePositionForDanmaku(danmaku:DSDanmakuLabel) -> CGPoint {
    let x = CGRectGetWidth(self.frame)
    let y = CGFloat(rand()%(Int32)(CGRectGetWidth(self.frame) - 50))
    var rect = CGRectMake(x, y, CGRectGetWidth(danmaku.frame), CGRectGetHeight(danmaku.frame))
    var intersect:DSDanmakuLabel?
    var intersectY = CGFloat(0.0)
    repeat {
        if let value = intersect {
            rect.origin.y = intersectY
            if (rect.origin.y + CGRectGetHeight(danmaku.frame) > CGRectGetHeight(self.frame)) {
                rect.origin.y = 0.0
                rect.origin.x = CGRectGetMaxX(value.layer.presentationLayer()!.frame)
            }
            intersect = nil
        }
        var index:Int
        for index = 0; index < movingDanmaku.count; index++ {
            let existDanmaku = movingDanmaku[index]
            if let compareRect = existDanmaku.layer.presentationLayer()?.frame{
                if (CGRectGetMaxX(compareRect) < CGRectGetMaxX(self.frame)) {

                    continue
                }
                if (CGRectIntersectsRect(compareRect, rect)) {
                    if intersect == nil {
                        intersect = existDanmaku
                        intersectY = CGRectGetMaxY(existDanmaku.frame)
                    }
                    rect.origin.y = CGFloat(rand()%(Int32)(CGRectGetWidth(self.frame) - 50))
                }
            }
        }

    } while (intersect != nil)

    return rect.origin
}
```

5.2.2.8 Implementation of RangeSlider

A rangeSlider is a combine of two class. They are the

RangeSliderTrackLayer and the RangeSliderThumbLayer.

```
class RangeSliderTrackLayer: CALayer {
    weak var rangeSlider: RangeSlider?

    override func drawInContext(ctx: CGContext) {
        if let slider = rangeSlider {
            // Clip
            let cornerRadius = bounds.height * slider.curvaceousness / 2.0
            let path = UIBezierPath(roundedRect: bounds, cornerRadius: cornerRadius)
            CGContextAddPath(ctx, path.CGPath)

            // Fill the track
            CGContextSetFillColorWithColor(ctx, slider.trackTintColor.CGColor)
            CGContextAddPath(ctx, path.CGPath)
            CGContextFillPath(ctx)

            // Fill the highlighted range
            CGContextSetFillColorWithColor(ctx, slider.trackHighlightTintColor.CGColor)
            let lowerValuePosition = CGFloat(slider.positionForValue(slider.lowerValue))
            let upperValuePosition = CGFloat(slider.positionForValue(slider.upperValue))
            let rect = CGRect(x: lowerValuePosition, y: 0.0, width: upperValuePosition - lowerValuePosition, height: bounds.height)
            CGContextFillRect(ctx, rect)
        }
    }
}
```

```
class RangeSliderThumbLayer: CALayer {
    var highlighted: Bool = false {
        didSet {
            setNeedsDisplay()
        }
    }
    weak var rangeSlider: RangeSlider?

    override func drawInContext(ctx: CGContext) {
        if let slider = rangeSlider {
            let thumbFrame = bounds.insetBy(dx: 2.0, dy: 2.0)
            let cornerRadius = thumbFrame.height * slider.curvaceousness / 2.0
            let thumbPath = UIBezierPath(roundedRect: thumbFrame, cornerRadius: cornerRadius)

            // Fill
            CGContextSetFillColorWithColor(ctx, slider.thumbTintColor.CGColor)
            CGContextAddPath(ctx, thumbPath.CGPath)
            CGContextFillPath(ctx)

            // Outline
            let strokeColor = UIColor.grayColor()
            CGContextSetStrokeColorWithColor(ctx, strokeColor.CGColor)
            CGContextSetLineWidth(ctx, 0.5)
            CGContextAddPath(ctx, thumbPath.CGPath)
            CGContextStrokePath(ctx)

            if highlighted {
                CGContextSetFillColorWithColor(ctx, UIColor(white: 0.0, alpha: 0.1).CGColor)
                CGContextAddPath(ctx, thumbPath.CGPath)
                CGContextFillPath(ctx)
            }
        }
    }
}
```

The following functions are about touching the slider bar. They record the range that the user slide between the interval of user begin to touch and end with touch.

```
// MARK: - Touches
override func beginTrackingWithTouch(touch: UITouch, withEvent event: UIEvent?) -> Bool {
    previouslocation = touch.locationInView(self)

    // Hit test the thumb layers
    if lowerThumbLayer.frame.contains(previouslocation) {
        lowerThumbLayer.highlighted = true
    } else if upperThumbLayer.frame.contains(previouslocation) {
        upperThumbLayer.highlighted = true
    }

    return lowerThumbLayer.highlighted || upperThumbLayer.highlighted
}

override func continueTrackingWithTouch(touch: UITouch, withEvent event: UIEvent?) -> Bool {
    let location = touch.locationInView(self)

    // Determine by how much the user has dragged
    let deltaLocation = Double(location.x - previouslocation.x)
    let deltaValue = (maximumValue - minimumValue) * deltaLocation / Double(bounds.width - bounds.height)

    previouslocation = location

    // Update the values
    if lowerThumbLayer.highlighted {
        lowerValue = boundValue(lowerValue + deltaValue, toLowerValue: minimumValue, upperValue: upperValue - gapBetweenThumbs)
    } else if upperThumbLayer.highlighted {
        upperValue = boundValue(upperValue + deltaValue, toLowerValue: lowerValue + gapBetweenThumbs, upperValue: maximumValue)
    }

    sendActionsForControlEvents(.ValueChanged)

    return true
}

override func endTrackingWithTouch(touch: UITouch?, withEvent event: UIEvent?) {
    lowerThumbLayer.highlighted = false
    upperThumbLayer.highlighted = false
}
```

Finally, based on the range between two tumb. Then use function positionForValue() and function boundValue() to calculate the value inside the range.

```
func positionForValue(value: Double) -> Double {
    return Double(bounds.width - thumbWidth) * (value - minimumValue) /
        (maximumValue - minimumValue) + Double(thumbWidth / 2.0)
}

func boundValue(value: Double, toLowerValue lowerValue: Double, upperValue: Double) -> Double {
    return min(max(value, lowerValue), upperValue)
}
```

5.3 Term 2 IPhone App

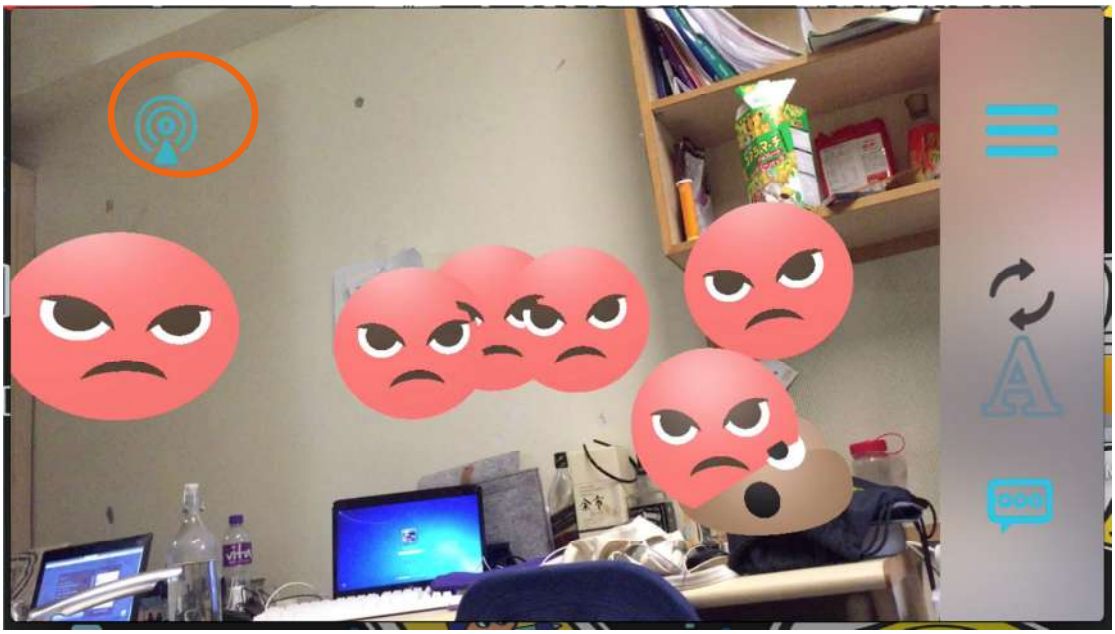
5.3.1 UI Design

5.3.1.1 previewMessage View

In term 2, the mapView showing the location of message is deleted. The previewMessage view become the first and main view of the app. This view by default is in emoji mode which is indicating the category of the messages. On the top left corner, the incon is showing the positioning system (GPS and beacon) the app is adopting, and the above symbol is showing that it is using GPS. On the right hand-side is a tool bar. From the top, ii is filtering category, when tap on this icon, a category menu bar will be pop up. The second one is refresh button that is to retrieve messages from database and refresh the whole preview message view. The third one is change the display mode, by default it is emoji mode. The last one is post message button. A post view will pop up to allow user type in the message content.

5.3.1.2 previewMessage View GPS and Beacon

The above view is showing that the GPS is adopted, when there is no registered beacon nearby, GPS will be used to locate user position that is mainly used for outdoor.

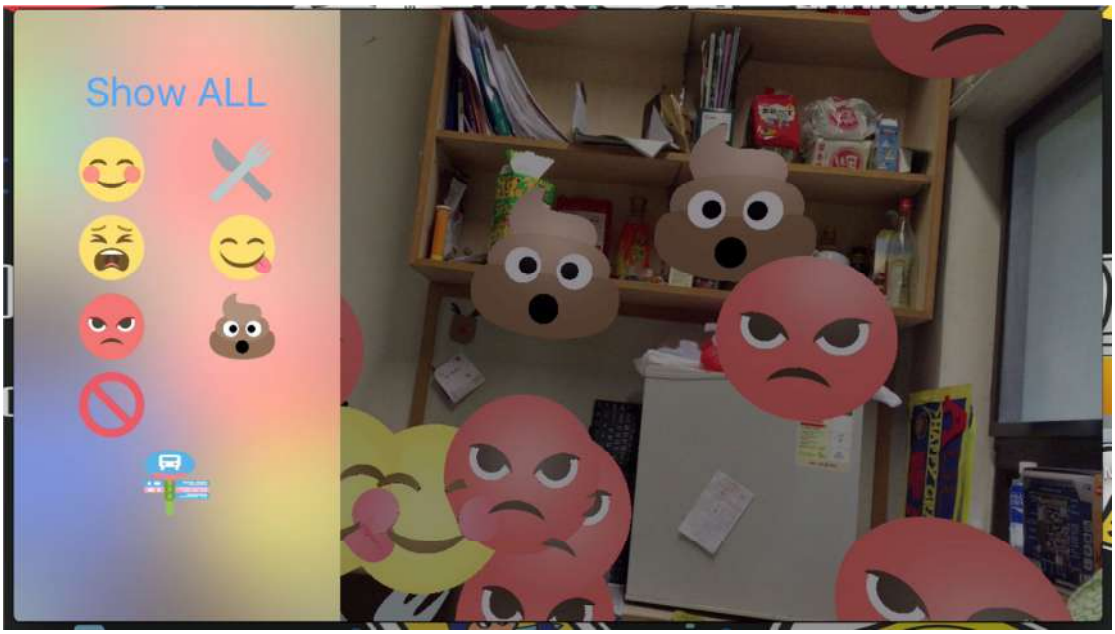


The above view is showing that beacon is adopted, when there is registered beacon beacon nearby, instead of using GPS, beacon will be immediately choosing as the position system. The transparency of icon is indication the signal strength of beacon. There is 3 level, far,

near and immediate. If signal strength is immediate, the icon transparency is low, the icon will be clearly shown, or vice versa. The above is immediate level signal.

5.3.1.3 previewMessage View Category Menu

Although the preview message view is displaying the category of each messages already, some users may want to see a specific type of message. So a category menu is provided for user to filter the type of messages.



After clicking the category menu button, the menu bar will pop. The tool bar will be disappeared. The background will be darker in order to make the sense of multi level. There are totally 8 categories, they are happy, sad, angry, warning, food, yummy, bad and navigation. After clicking the category icon, the messages will be filtered and return back to preview. Users can also tap on the dark region to back to preview.

5.3.1.4 previewMessage View Refresh Button

Tap on the refresh button, the app will delete the stored messages information immediately, then a http request will be sent to server to retrieve the message information again. After that the 3D icon will be render, an up-to-date message will be shown.

The refresh event will also be automatically invoked when there is a change in positioning system, change in registered beacon, there is no message and device moved about every 50m from previous position.

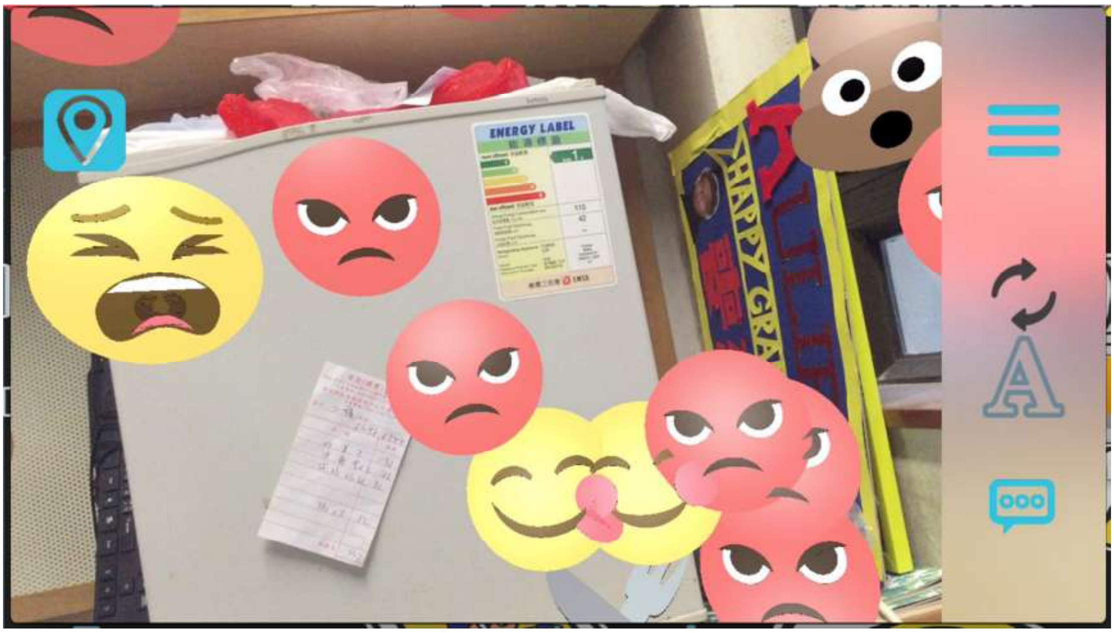
5.3.1.5 previewMessage View Content / Emoji Mode

Besides of displaying messages category emoji, by clicking the change mode button, full content of the messages can be shown.

The above view is the full content mode, we can see that all the messages content are shown on screen. Of course, by clicking the change mode button again, the view will change to emoji mode.

In addition, there is one more way to view the messages content. That is tap on the emoji, then a box will pop up showing full message.

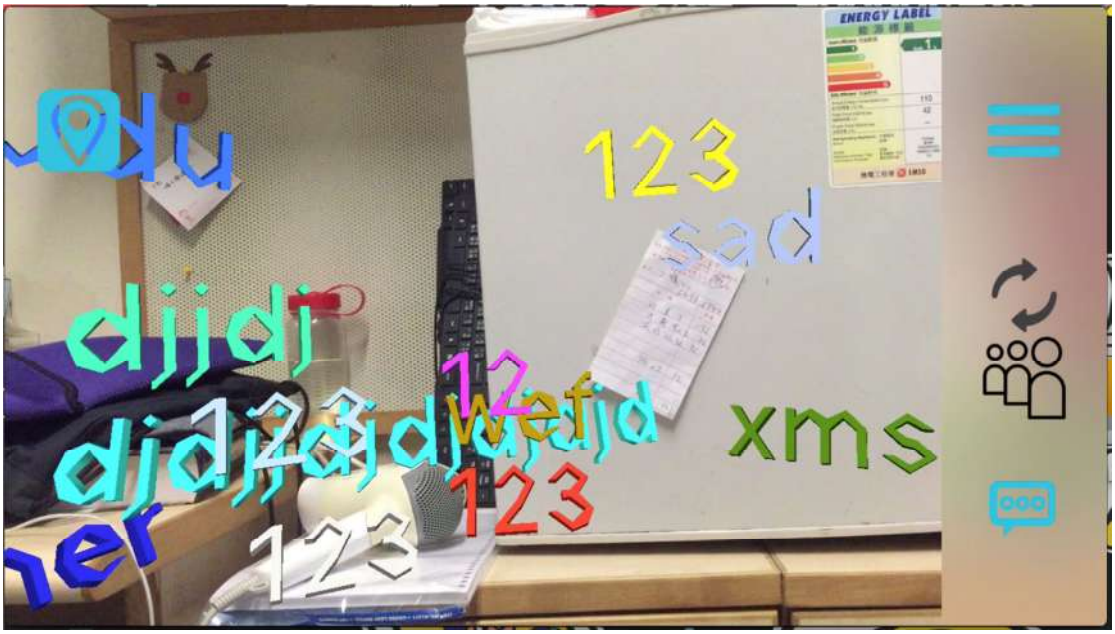
After clicking the bad emoji, a box has pop up showing the message ID and the message content. If clicking the cancel button, the box will disappear and return to preview. If clicking the seen button, it will back to preview, but the corresponding message emoji will be deleted, so that the users can omitte the message that have read already.



The above view is showing that after clicking the seen button, the emoji of that message will be disappeared

Similarly, in the content mode, users can also click on the message object.

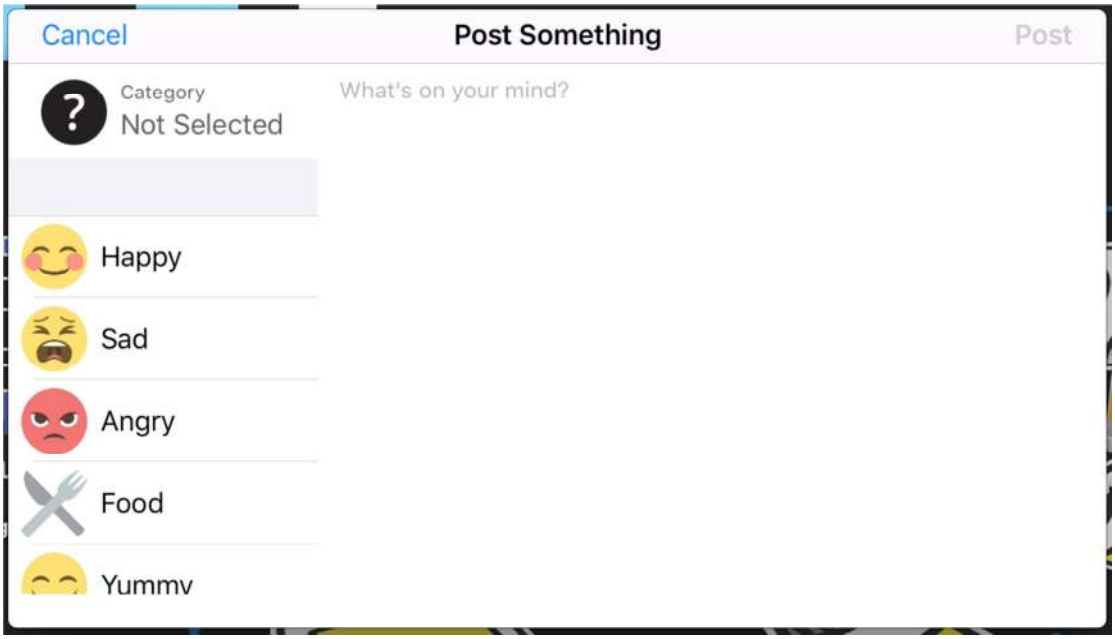
After clicking the message “no good”, a box is displayed to show the message ID and content. Clicking the cancel button will go back to preview. But if clicking the seen button, the message will disappear in preview.



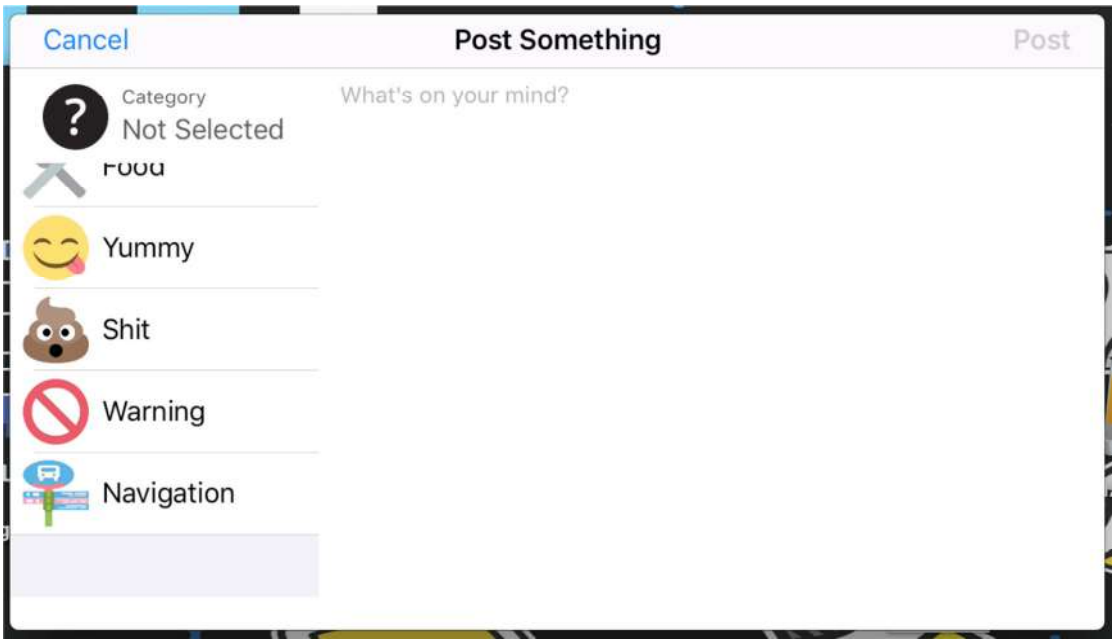
This view is showing that after clicking the seen the corresponding message object will be deleted in preview.

5.3.1.6 postMessage View

To post a message, tap on the bottom left icon.

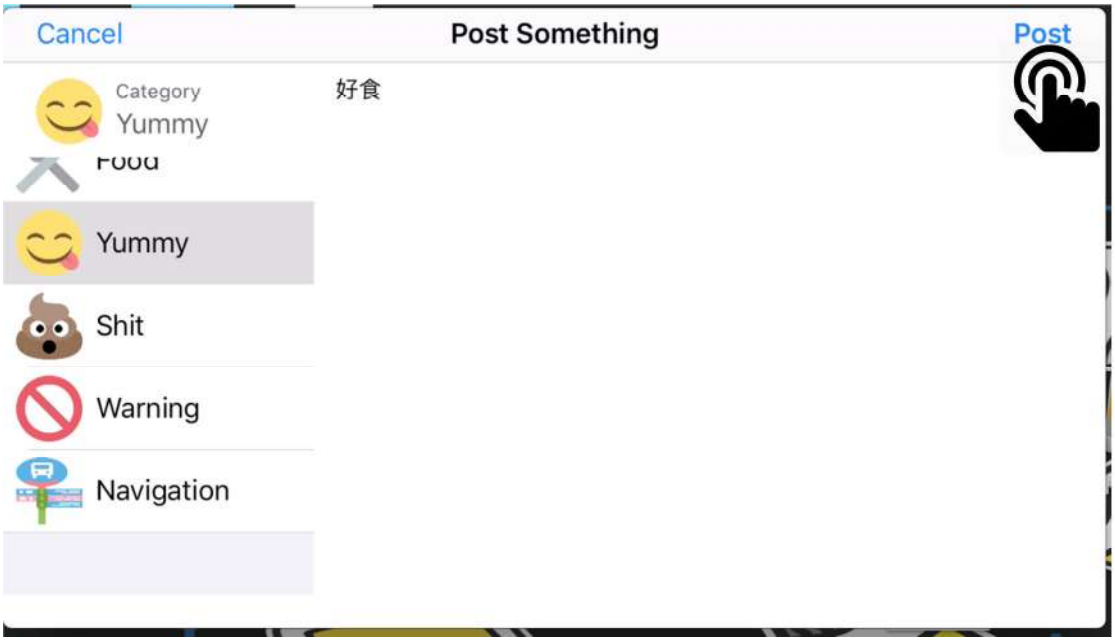


Then the postMessage view will pop up. On the left hand side, it shows which category has chosen by the user, as it just change to post view, no category is selected. Swipe up and down on the category table to chose the most suitable category to express users feeling.

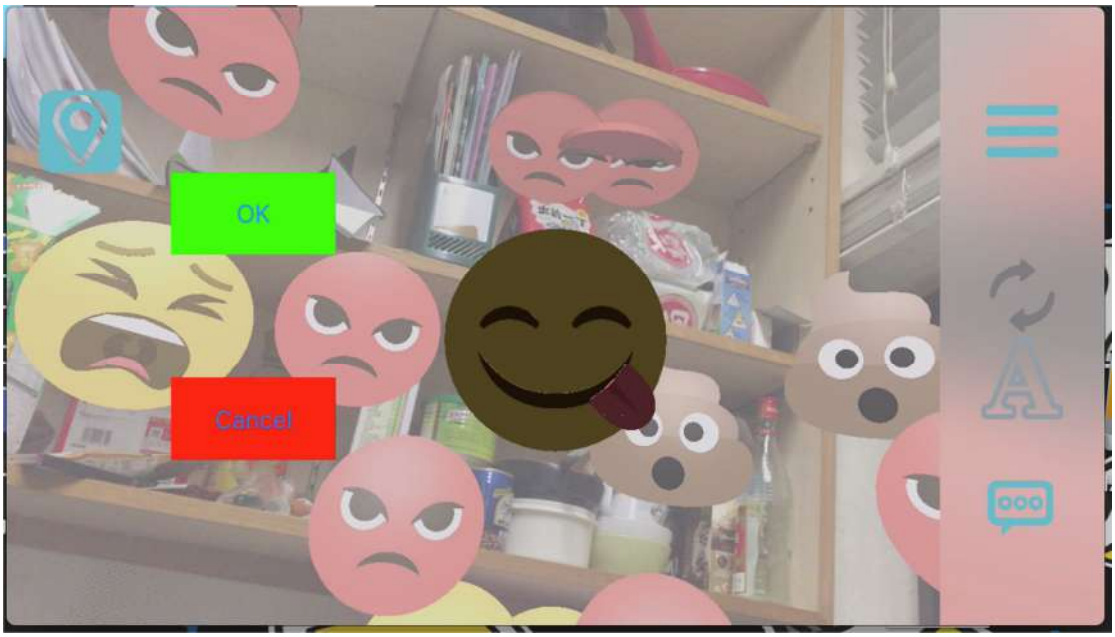


Double click on the text box to type in the message content.

For example, the message Yummy category is clicked and "好食" is the message content.



Then click the post button.



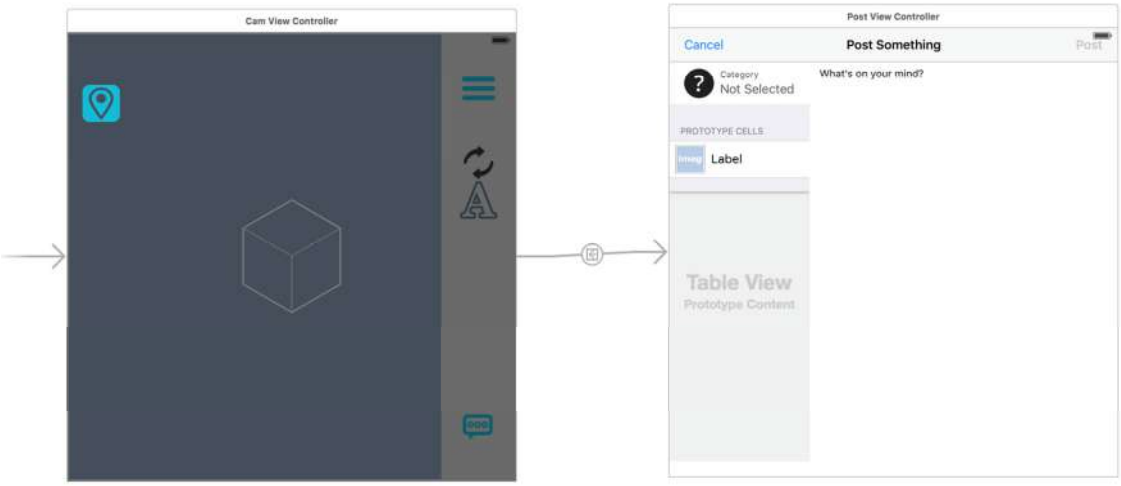
Then it will pop up a post preview mode, user can move their phone and put the message in the position they like. Click OK to confirm the position. Click cancel will back to the preview message view.

This view is showing after click the OK button, the Yummy emoji will be posted.

5.3.1.7 preivewMessge View Navigation

There is a special kind of category that is navigation. The navigation messages will render at the bottom part of the view. It does not have 3D emoji; it will have an arrow pointing toward the location direction with the messages content.

5.3.2 Modules



As mentioned before, we want a simpler user interface, so map view is deleted. PreviewMessage View will be the entry view and the main view of the app. Most of the function and device information is done in the view. The segue connecting preview and post view is a standard pop up segue.

Most the functionality is inherited from term 1, like location services, CLLocation, CLHeading, Longitude-Latitude Distance.

Although it has less modules, we have implemented few more to enhance the app.

In addition, a big change in user interface has been made in postMessageView.

5.3.2.1 Implementation of Beacon Services:

In this app, one of the biggest improvement is allowing indoor positioning that is using Beacon. In IOS, API CoreLocation services, it provides many methods supporting usage of beacon. This API is already imported in the first term, so we simply need to look for suitable function in the API and apply them in this app.

```
let regionB = CLBeaconRegion(proximityUUID: NSUUID(UUIDString:
    "6375686B-2E65-6475-2E68-6B2E30303031")!, identifier: "ViewLab-Beacons")
```

First, we have to specify the UUID of the Beacons that the app will monitor.

```
func locationManager(manager: CLLocationManager, didRangeBeacons beacons: [CLBeacon], inRegion region: CLBeaconRegion) {
    // print(region.identifier)
    // print(beacons)
    let knownBeacons = beacons.filter{ $0.proximity != CLProximity.Unknown }
    if( knownBeacons.count > 0){

        GPSImage.hidden = true
        beaconImage.hidden = false
        let closestBeacon = knownBeacons[0] as CLBeacon
        //BEACON ADD
        locationInfo.currBeacon = closestBeacon

        locationInfo.closeBeacon = closestBeacon.minor
          print("closebeacon: \(locationInfo.closeBeacon)")
        print("closebeacon: \(locationInfo.currBeacon!.proximityUUID.UUIDString)")
        let prox = closestBeacon.proximity.rawValue
        if( prox == 1){
            beaconImage.alpha = 1
        }
        else if ( prox == 2){
            beaconImage.alpha = 0.7
        }
        else{
            beaconImage.alpha = 0.2
        }
    }
    else{
        //BEACON ADD
        locationInfo.currBeacon = nil
        locationInfo.closeBeacon = 0
        GPSImage.hidden = false
        beaconImage.hidden = true
    }
      print("closeBeacon: \(closeBeacon)")
}
```

Then, didRangeBeacons function need to be added. This function will be automatically invoked when the device has detected beacon. As there may be more then one beacon nearby, in order to choose the strongest signal beacon, we need to sort the beacon array according to the strength and pick the first element. Then according to its

strength set the beacon transparency of beacon symbol and hide the GPS icon in preview view. If the beacon signal gone, this function will invoke again, then it hide the beacon icon and turn on the GPS again.

5.3.2.2 Implementation of 3D Rendering:

After retrieving all the messages from data base, it tokenizes the response as before to differentiate the message information. Then according to the messages information a 3D object is rendered to preview view.

```
let cameraNode = SCNNode()
let scene = SCNScene()
@IBOutlet weak var scnView: SCNView!
```

First add the cameraNode and scene that is to define the angle of viewing the 3D space and 3D space.

Then renderMessagwWithTitle() is called. This function is used to create 3D text in preveiew message view content mode.

```
func renderMessageWithTitle(){
    print("render message")

    for x in 0  ..< messages.count  {
          print(messages[x].content)

        let roll = Float(messages[x].roll)

        let z1 = -cos(-roll)
        let z12 = z1 * z1

        let lenght = sqrt(4900 - z12)
        let targetHeading : Float = GLKMathDegreesToRadians(Float(messages[x].heading))

        if messages[x].tag != "Navigation"{

            let temp = SCNText(string: messages[x].content, extrusionDepth: 2)
            temp.firstMaterial!.diffuse.contents = UIColor.randomColor()
            let tempNode = SCNNode(geometry: temp)

            tempNode.eulerAngles = SCNVector3Make(-roll, 0, -Float(targetHeading))

            tempNode.position = SCNVector3(x: lenght * sin(targetHeading), y: lenght * cos(targetHeading), z: 50 * z1)

            tempNode.name = "\(x)"

            var minVec = SCNVector3Zero
            var maxVec = SCNVector3Zero
            if tempNode.getBoundingBoxMin(&minVec, max: &maxVec) {
                let bound = SCNVector3(
                    x: maxVec.x - minVec.x,
                    y: maxVec.y - minVec.y,
                    z: maxVec.z - minVec.z)

                tempNode.pivot = SCNMatrix4MakeTranslation(bound.x / 2, bound.y / 2, bound.z / 2)

            }

            scene.rootNode.addChildNode(tempNode)
```

```
        } else {
            var postContent = ""
            let messageContent = messages[x].content
            //                let loopTimes = messageContent.characters.count/2 -1

            for _ in 0 ..< messageContent.characters.count/2 {
                postContent += " "
            }
            print(messageContent.characters.count)


            postContent += "⇧\n"
            postContent += messageContent

            let temp = SCNText(string: postContent, extrusionDepth: 2)
            temp.firstMaterial!.diffuse.contents = UIColor.randomColor()
            let tempNode = SCNNode(geometry: temp)

            tempNode.eulerAngles = SCNVector3Make(0, 0, -Float(targetHeading))

            tempNode.position = SCNVector3(x: 50 * sin(targetHeading), y: 50 * cos(targetHeading), z: -70)

            tempNode.name = "\(x)"

            var minVec = SCNVector3Zero
            var maxVec = SCNVector3Zero
            if tempNode.getBoundingBoxMin(&minVec, max: &maxVec) {
                let bound = SCNVector3(
                    x: maxVec.x - minVec.x,
                    y: maxVec.y - minVec.y,
                    z: maxVec.z - minVec.z)

                tempNode.pivot = SCNMatrix4MakeTranslation(bound.x / 2, bound.y / 2, bound.z / 2)

            }
            scene.rootNode.addChildNode(tempNode)
        }

    }

}
```

This function mainly divided into two part. The first part is checking the message category is navigation type or not. If yes, render the message on the bottom part of the 3D space according to message information(orientation of device). If not, simply render all the messages to its corresponding location.

Then, renderMessageIcon() is invoked. This method is used to render the 3D emoji in preview message view emoji mode.

```
func renderMessageWithIcon(){
    for x in 0  ..< messages.count  {

        let roll = Float(messages[x].roll)

        let z1 = -cos(-roll)
        let z12 = z1 * z1

        let lenght = sqrt(3600 - z12)


        let targetHeading : Float = GLKMathDegreesToRadians(Float(messages[x].heading))

        if messages[x].tag != "Navigation"{
            print("the cat is  ")
            print(messages[x].tag)
            let categoryScene = SCNScene(named: "art.scnassets/\(messages[x].tag).scn")!
            let object = categoryScene.rootNode.childNodeWithName("\(messages[x].tag)", recursively: true)!
            //object.geometry?.firstMaterial?.diffuse.contents = UIColor.randomColor()

            var tempNode = SCNNode()
            tempNode = object


              if messages[x].tag == "Angry"{
                  tempNode.eulerAngles = SCNVector3Make(-roll , 0, -Float(targetHeading))
              } else if messages[x].tag == "B" {
                  tempNode.eulerAngles = SCNVector3Make(-roll - Float(M_PI_2), 0, -Float(targetHeading))
              } else if messages[x].tag == "C" {
                  tempNode.eulerAngles = SCNVector3Make(-roll - Float(M_PI_2), 0, -Float(targetHeading))
              } else if messages[x].tag == "D"{
                  tempNode.eulerAngles = SCNVector3Make(-roll - Float(M_PI_2), 0, -Float(targetHeading) + Float(M_PI))
              }
            tempNode.eulerAngles = SCNVector3Make(-roll , 0, -Float(targetHeading))

            tempNode.position = SCNVector3(x: lenght * sin(targetHeading), y: lenght * cos(targetHeading), z: 50 * z1)

            tempNode.name = "\(x)"
            scene.rootNode.addChildNode(tempNode)


        } else {
            var postContent = ""
            let messageContent = messages[x].content
            //                let loopTimes = messageContent.characters.count/2 -1

            for _ in 0 ..< messageContent.characters.count/2 {
                postContent += " "
            }
            print(messageContent.characters.count)


            postContent += "⇧\n"
            postContent += messageContent

            let temp = SCNText(string: postContent, extrusionDepth: 2)
            temp.firstMaterial!.diffuse.contents = UIColor.randomColor()
            let tempNode = SCNNode(geometry: temp)

            tempNode.eulerAngles = SCNVector3Make(0, 0, -Float(targetHeading))

            tempNode.position = SCNVector3(x: 50 * sin(targetHeading), y: 50 * cos(targetHeading), z: -70)

            tempNode.name = "\(x)"

            var minVec = SCNVector3Zero
            var maxVec = SCNVector3Zero
            if tempNode.getBoundingBoxMin(&minVec, max: &maxVec) {
                let bound = SCNVector3(
                    x: maxVec.x - minVec.x,
                    y: maxVec.y - minVec.y,
                    z: maxVec.z - minVec.z)

                tempNode.pivot = SCNMatrix4MakeTranslation(bound.x / 2, bound.y / 2, bound.z / 2)

            }
            scene.rootNode.addChildNode(tempNode)
        }


    }
}
```

The mechanism of this function is similar to rendMessageWiithTitle().

5.3.2.3 Implementation of postMessageView:

postMessageView consist of two part, the table-view for choosing the category and text field.

The text field functions are inherited from last app. So this part will only focus on the table-view.

```
@IBOutlet weak var categoryTableView: UITableView!

@IBOutlet weak var categoryName: UILabel!
@IBOutlet weak var categoryImage: UIImageView!


    override func viewDidLoad() {
        self.messageToPost.delegate = self

        messageToPost.text = "What's on your mind?"
        messageToPost.textColor = UIColor.lightGrayColor()

        self.categoryTableView.dataSource = self
        self.categoryTableView.delegate = self
//        loadCategorys()

    }
```

In the view, it has to declare a categoryTableView and set the dataSource and delegate to be the postViewController.

```
import UIKit

class CategoryTableViewCell: UITableViewCell {

    @IBOutlet weak var categoryImageView: UIImageView!
    @IBOutlet weak var categoryLabel: UILabel!


    override func awakeFromNib() {
        super.awakeFromNib()
        // Initialization code
    }

    override func setSelected(selected: Bool, animated: Bool) {
        super.setSelected(selected, animated: animated)

        // Configure the view for the selected state
    }

}
```

Then, we need to design a framework of a cell in a table.

```
    func numberOfSections InTableView(tableView: UITableView) -> Int {
        // #warning Incomplete implementation, return the number of sections
        return 1
    }

    func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        // #warning Incomplete implementation, return the number of rows
        return categorys.count
    }


    func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {

        let cellIdentifier = "CategoryCell"
        let cell = tableView.dequeueReusableCellWithIdentifier(cellIdentifier, forIndexPath: indexPath) as! CategoryTableViewCell

        let category = categorys[indexPath.row]
        let image = images[indexPath.row]

//        cell.categoryLabel.text = category.name
//        cell.categoryImageView.image = category.photo
        cell.categoryLabel.text = category
        cell.categoryImageView.image = UIImage(named: image)

        return cell
    }

    func tableView(tableView: UITableView, didSelectRowAtIndexPath indexPath: NSIndexPath) {
        let category = categorys[indexPath.row]
        let image = images[indexPath.row]
        self.categoryName.text = category
        self.categoryImage.image = UIImage(named: image)


        // MARK: problem in the second condition
        if self.categoryName.text != "Not Selected" || self.messageToPost.text != "What's on your mind?" {
            postButton.enabled = true
        }
    }
```

Then according to cell framework, draw the whole table one by one, and input different category and its corresponding emoji.

6. Limitations and Difficulties

6.1Limitation of Beacon:

In IOS device, if the app has to detect beacons, the UUID of the beacons need to be register in the app. Therefore, only the specified beacons can be monitored by the app. This issue brings a big problem to our app. As our app highly depends on beacon to do indoor positioning, but we can only detect the beacons that we install. When there is unknown beacon, we cannot use that beacon to do indoor positioning. So that this app can only detect the beacons that we installed in CUHK engineering building. It highly limits the scalability of our apps.

6.2. Limitation of Screen Size:

The screen size is iPhone is small, so that the 3D objects need to in small scale. It leads to the 3D effect of emoji and text is not clearly shown. Moreover, when there is huge amount of messages, the 3D object has chance of overlapping. The clarity of messages is worse.

6.3. Difficulties of Building 3D objects:

In this app, we render the 3D objects according to the orientation of the device when posting the message. And the orientation of a device can be resolved into three components: roll, pitch and yaw. This coordinate system is defined when the phone is in portrait mode, but our app is fix to landscape mode. That means it need some calculation to change the coordinate system to landscape mode. But we take a long time to figure out this problem. Moreover, our teams do not have any knowledge in 3D graphics and editing, we need lots of time to acquire the related knowledge. In addition, the 3D object is not stable, when there is a sudden movement of the device, the position of the 3D cannot stay at its correct position. But the 3D objects should still at the original postion.

7. Conclusion

To conclude, the application that we developed this year is a funny product. Although the technique that we use is not that new, we try to combine lots of the existing things to produce an exciting experience that can surprise the user.The application make use of indoor positioning, outdoor positioning, and orientation of the phone to perform a stable platform to let the use say what they want to say.

Also, this app can be apply in many direction such us being a digital road signs over the world. It also can perform as a digital democracy wall instead of the one we are using in the 中大文廣. For the entertainment aspect, we also can apply the app in playing orienteering, etc.

Last but not least, since there are still many ways to apply such concept of the location based message. We hope that the user using our app can make use of these features that we made and will be able to discover more, having fun on our app.

8. Reference:

[1] Calculate distance, bearing and more between Latitude/Longitude points[Online] http://www.movable-type.co.uk/scripts/latlong.html

[2] Harversine Fomula, Wikipedia[Online] https://en.wikipedia.org/wiki/Haversine_formula#The_haversine_formula

[3] Apple Inc. iOS Developer Library[Online] https://developer.apple.com/library/ios/navigation/#section=Frameworks&topic=CoreLocation

9. Acknowledgement:

We would like to express our appreciation to our supervisor, Professor Lyu Rung Tsong Michael, for giving valuable guidance and comments.

In addition, we would like to thank Mr. Edward Yau in ViewLab for giving inspiring idea and advices.