TERM1 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. POST OVERVIEW 12
   3.1 SERVER 12
   3.2 IPHONE APP 12
4. POST DESIGN 13
   4.1 SERVER 13
      4.1.1 Server-Side System Architecture 13
      4.1.2 Database 14
      4.1.3 ER Diagram 14
      4.1.4 Schema 15
      4.1.5 Sequence Diagram 16
   4.2 PHONE APP 18
      4.2.1 Flow Chart 18
      4.2.2 Modules 20
         4.2.2.1 MVC 21
         4.2.2.2 Class Diagram 23
5. POST IMPLEMENTATION 25
   5.1 SERVER 25
      5.1.1 Retrieving messages 25
      5.1.2 Post Message 27
   5.2 IPHONE APP 29
      5.2.1 UI Design 29
         5.2.1.1 Map View Annotations 29
         5.2.1.2 Map View Category Menu 30
         5.2.1.3 Post Message View 33
         5.2.1.4 Message Display View 35
         5.2.1.5 Range slider bar for selecting time period 37
         5.2.1.6 Message Display 39
      5.2.2 Modules 40
         5.2.2.1 Implementation of Location Services: 41

5.2.2.2 Implementation of CLLocation and CLHeading: 42
5.2.2.3 Implementation of Longitude-Latitude Distance 43
5.2.2.4 Implementation of MapViewController. 45
5.2.2.5 Implementation of postMsgViewController 49
5.2.2.6 implementation of segue 50
5.2.2.7 Implement of Danmaku(Message Display) 52
5.2.2.8 Implementation of RangeSlider 54
6. LIMITATIONS AND DIFFICULTIES 56
6.1. LIMITATION OF CLCIRCULARREGION: 56
6.2. LIMITATION OF DEFAULT SEGUE: 58
6.3. CHANGING IN SWIFT VERSION AND IOS VERSION 58
6.4. LIMITATION IN POSITIONING SYSTEM 59
6.5. DIFFICULTIES IN CODING 59
7. FUTURE WORKS (MILESTONE II) 60
8. CONCLUSION 62
9. REFERENCE: 63
10. ACKNOWLEDGEMENT: 64

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.

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. With this two 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 work 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.**

3. POST 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

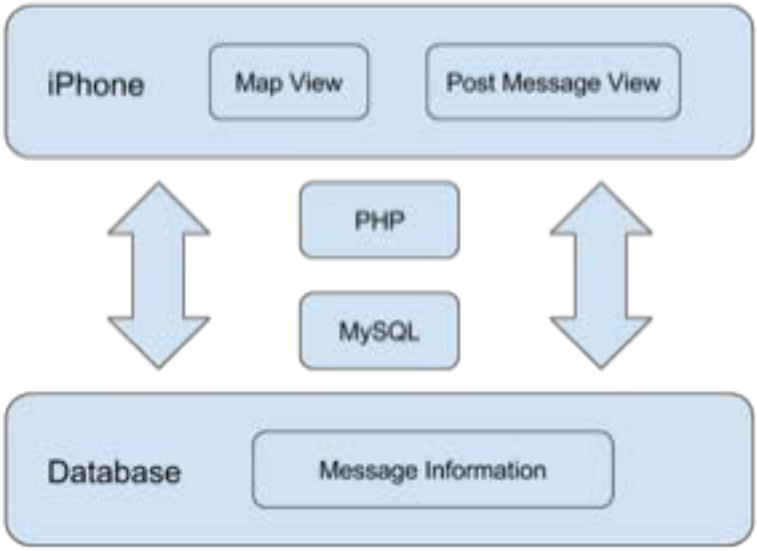
Our app has four major components: 1. Locate messages; 2. Post message; 3. Filter the messages; 4. Read the messages.

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 grep the location of the users, the user have to switch on the GPS 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. POST Design

4.1 Server

4.1.1 Server-Side System Architecture

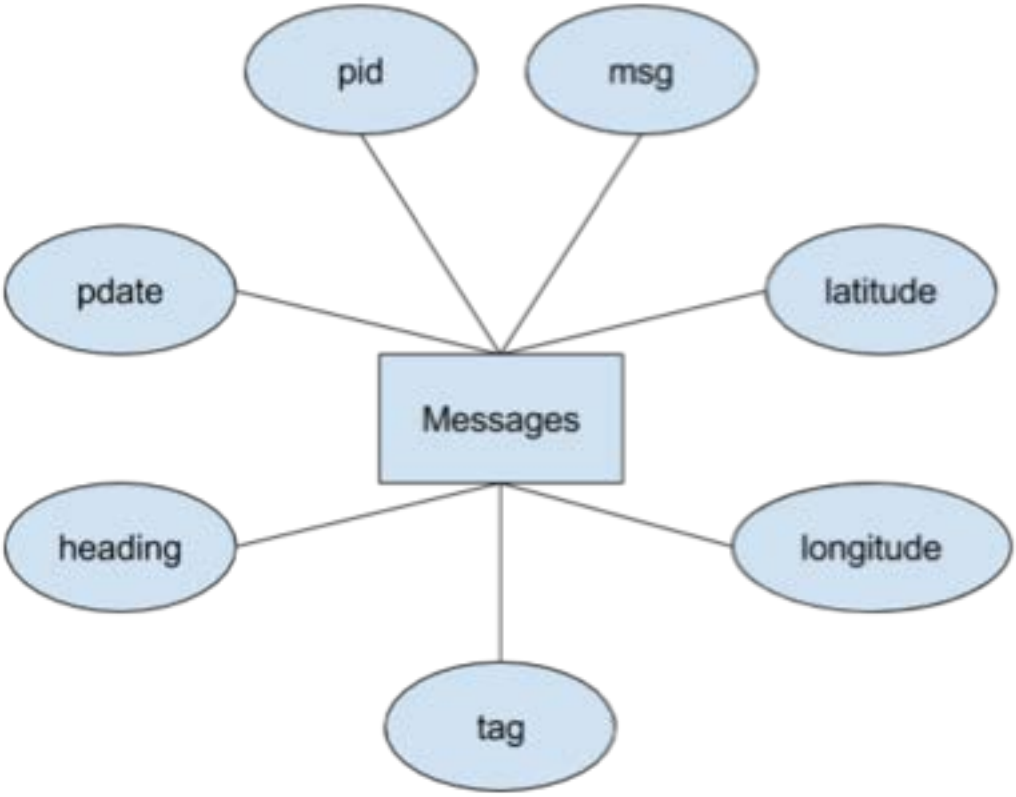


The above diagram is showing the server side system architecture. The iPhone application retrieves and uploads messages to database via PHP and MySQL. The data transfer is mainly the message content and message information.

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.

4.1.3 ER Diagram



The pid of each message is unique.

All message are identified by the pid.

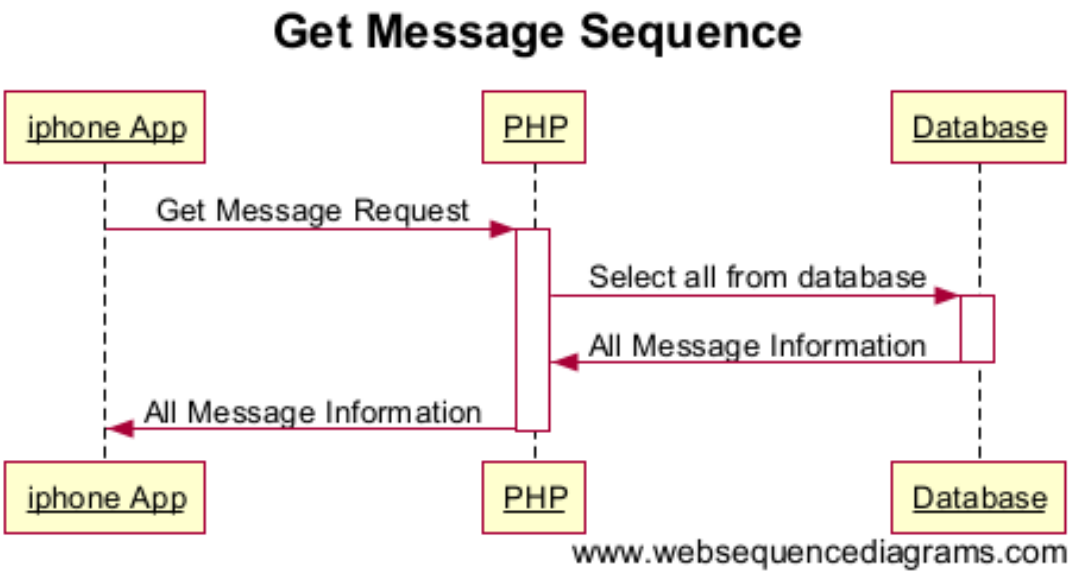
Besides pid, others attributes can be the same.

4.1.4 Schema

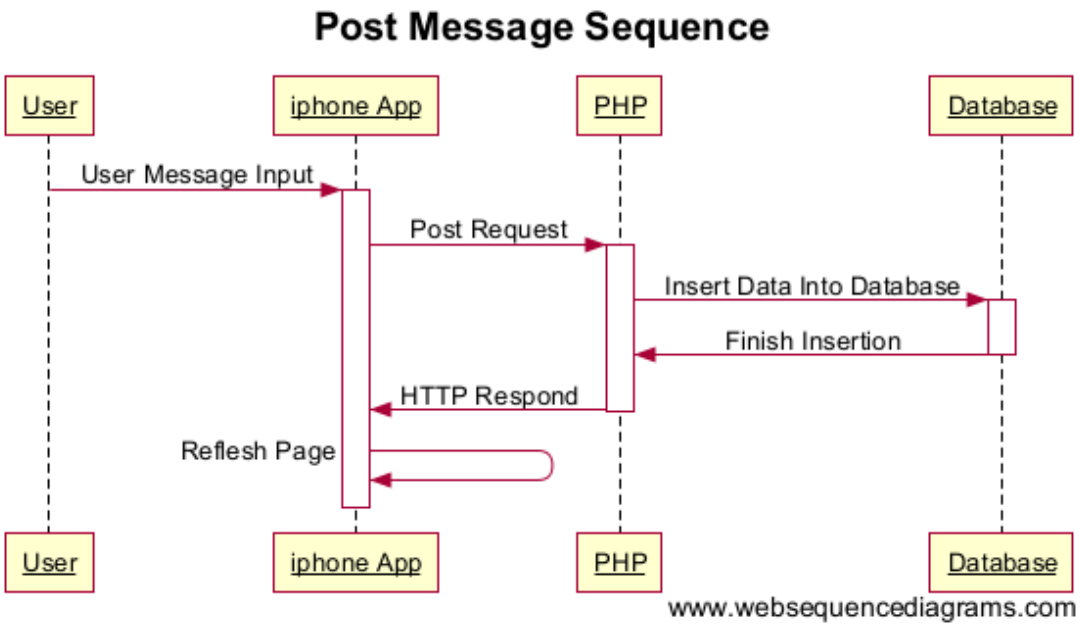
There is a record for each message posted on virtual notice board. Each message has its pid, longitude, latitude, heading, message content, tag, date of post.

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 Sequence Diagram



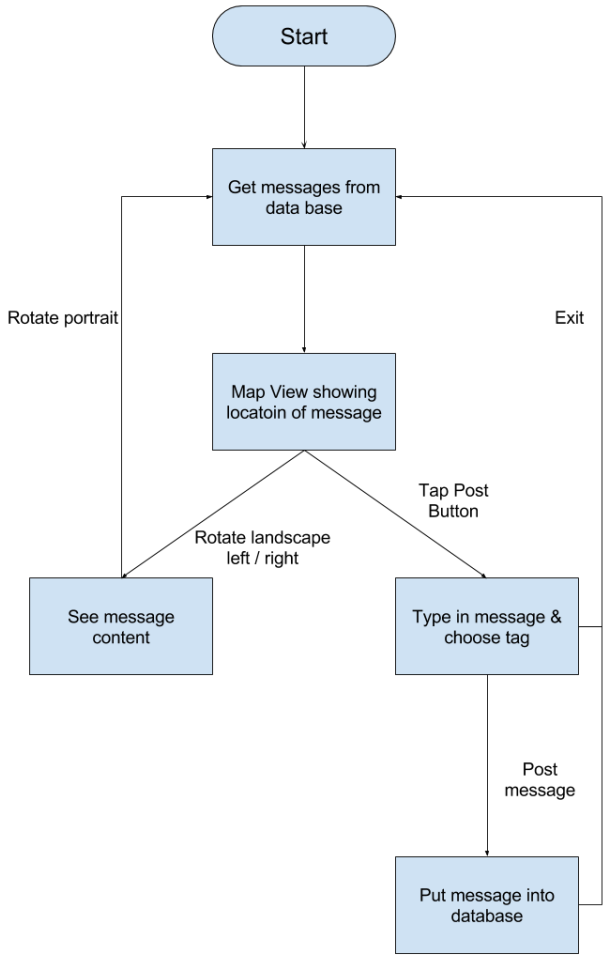
The above sequence diagram is showing how the message is get from the database. The app first sends a http request to server, via the PHP send a SQL command to database to retrieve all the message information. The messages will send back to app via the PHP by an http response. The above get message procedure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



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 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

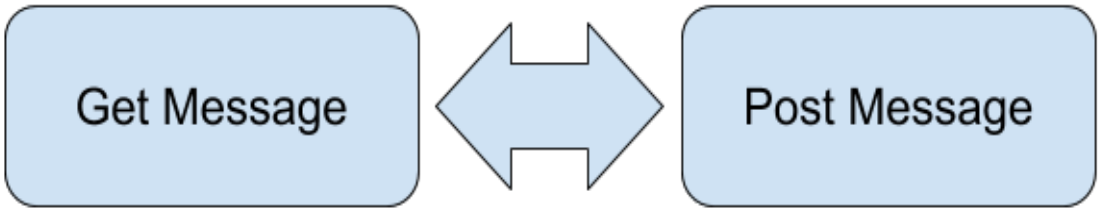
see where are the message posts before, filter the messages with category.

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.

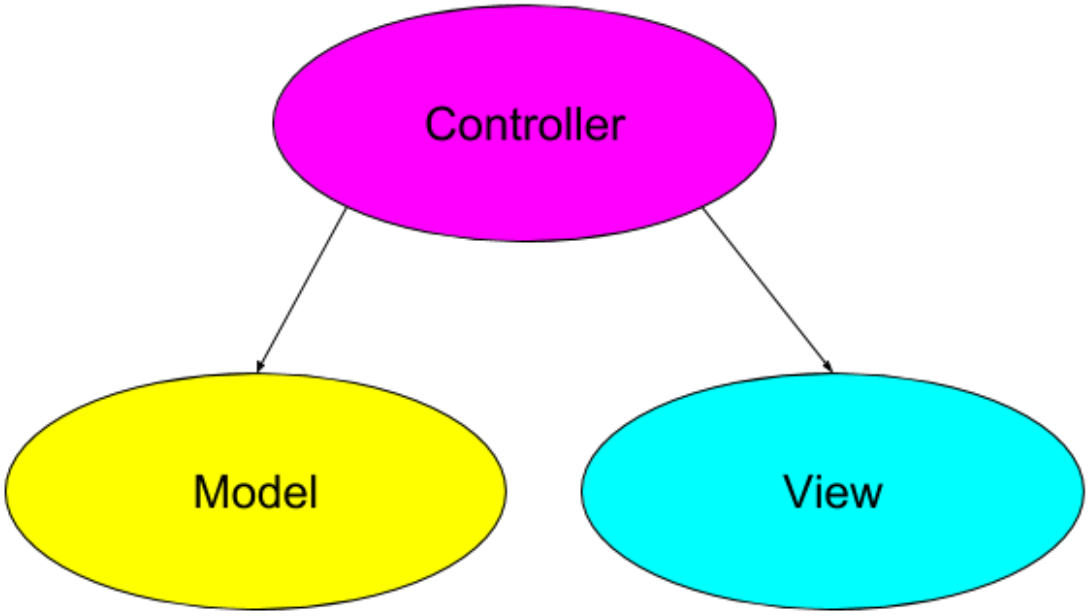
In the map view, user can tap on post button to switch to post message view. User can go back to map view, by clicking cancel button. Or user type in the message, post to server, then get update the information from database and back to map view.

4.2.2 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.

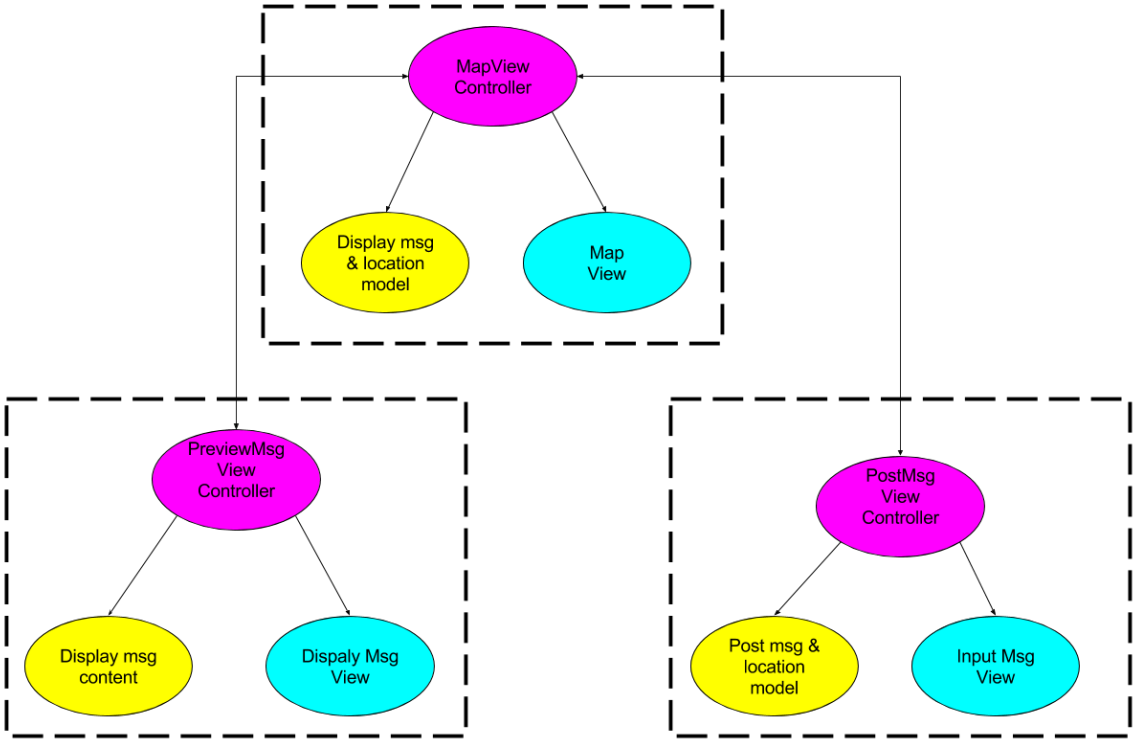


In IOS application development, the above modules can be further elaborate in a model-view-controller design pattern (MVC) which can much clearly show how the programs and the whole structure behind.



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.

4.2.2.1 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.

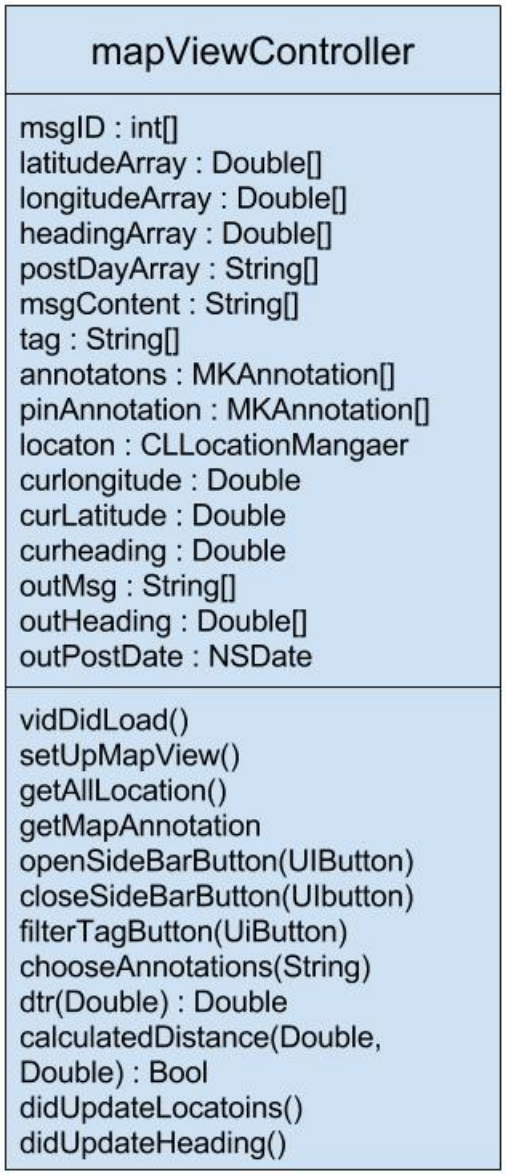
MapViewController is the entry of the app. It is a map view showing the location of messages. Behind the scene, it is getting message from database, storing the information of each data including the message identifier, content, position. Also, it allows user to filter message. At the same time, it is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

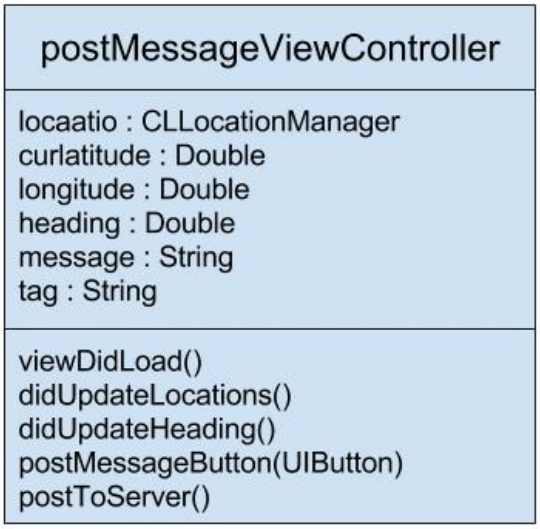
PostMsgViewController is for posting message to the database. User input the message content from view, via the controller the message is passed the post msg & location model to do the encoding with current location, then send to database.

4.2.2.2 Class Diagram

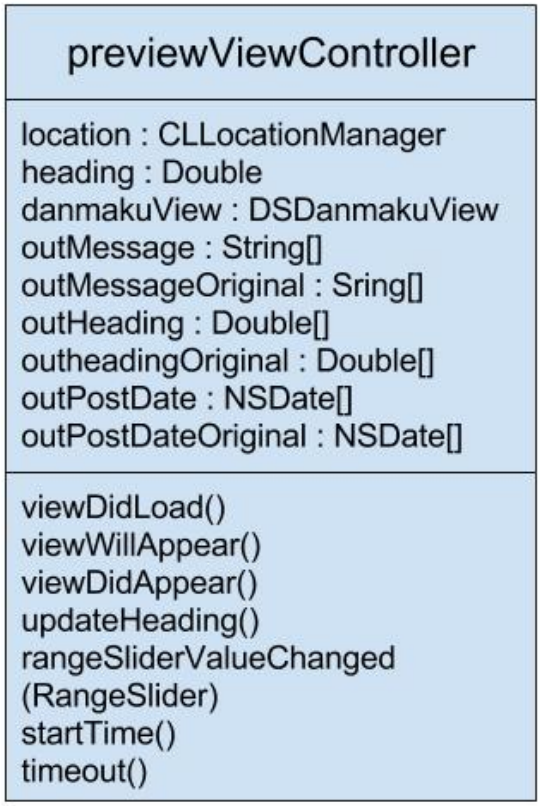
The class diagram of mapViewController



The class diagram of postMessageViewController



This is the class diagram of previewViewController



5. POST Implementation

5.1 Server

5.1.1 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.

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 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.

5.2 IPhone App

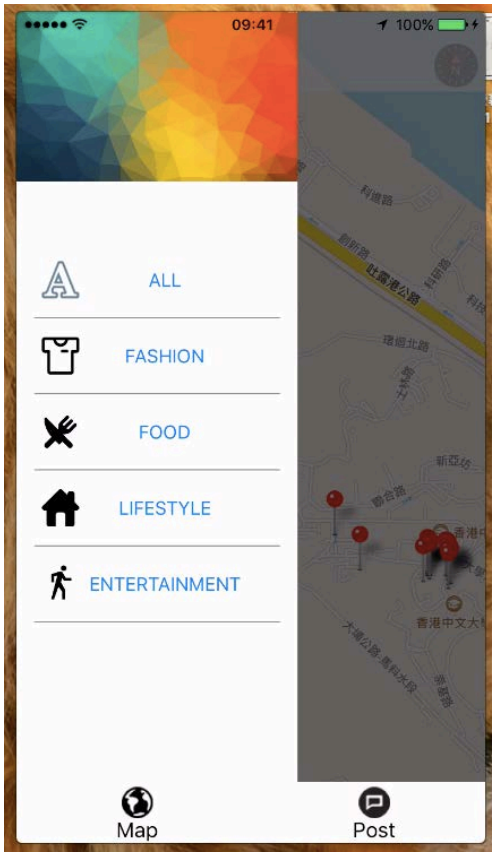
5.2.1 UI Design

5.2.1.1 Map View Annotations

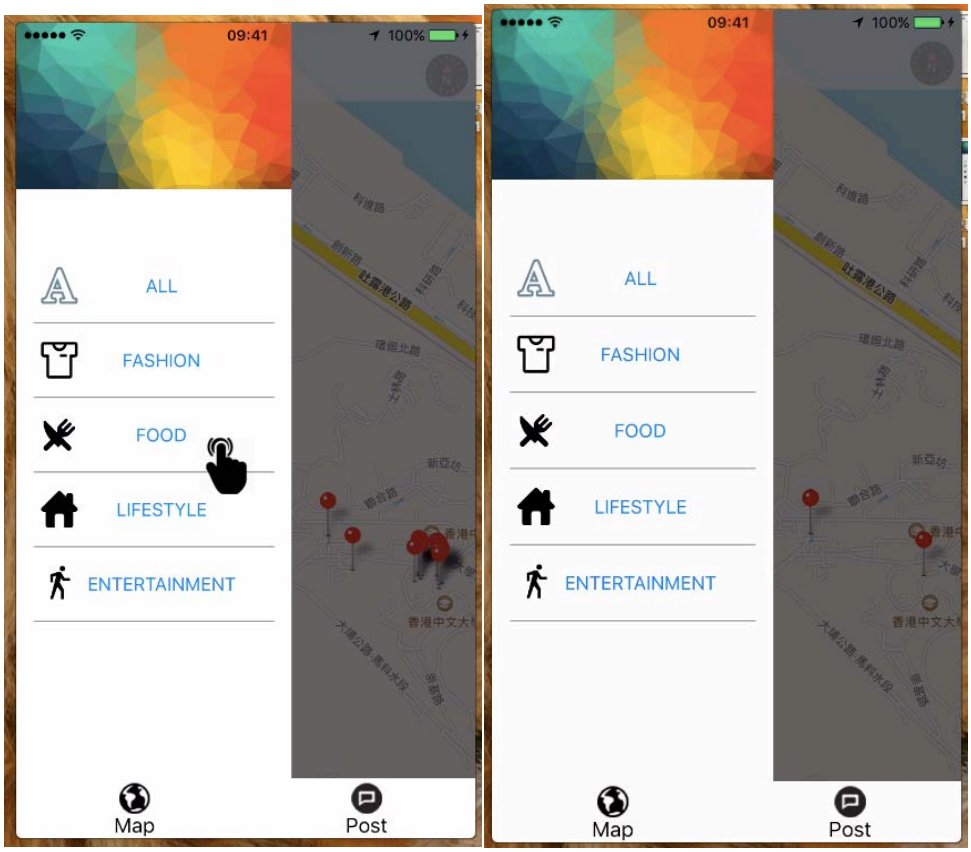
The map view is the main view of the app. It is the first page we can see when user open our app. It will show a map of user's current location. If there is a message, the map will display annotations that indicate the actual position of that message. Also, the user can know which type the message is 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.



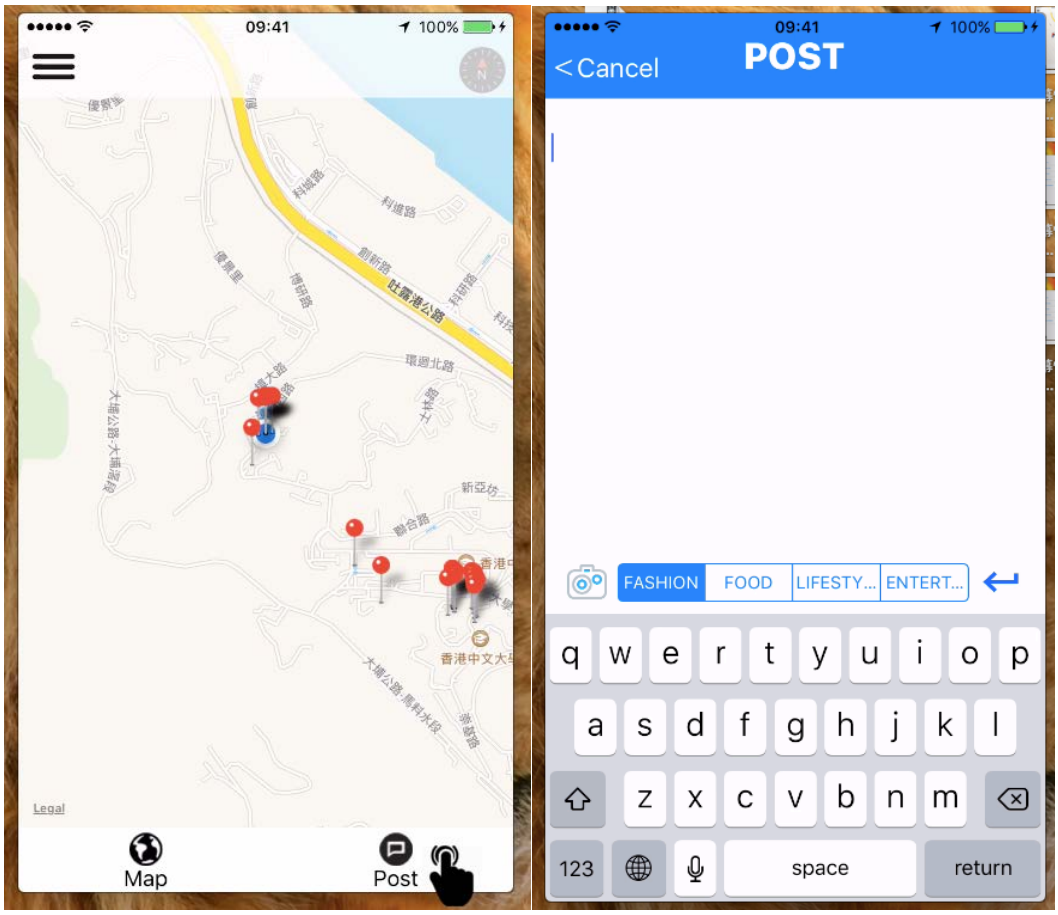
When the menu bar pops up, the background will be darker than the normal in order to make the sense of multi-level. And when the users tap on the dark region, the app will return to the map view.



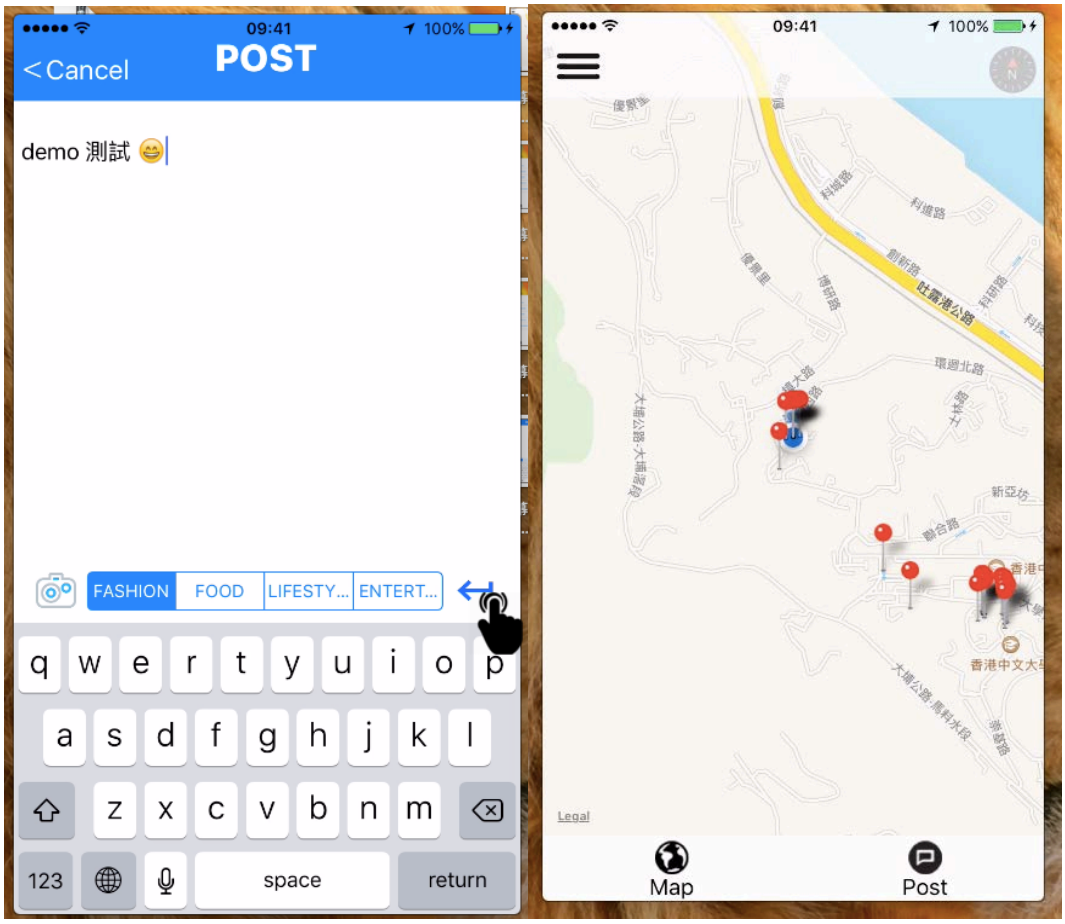
There are four categories for the messages, fashion, food, lifestyle and entertainment. User can choose a category message they are interested in.

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



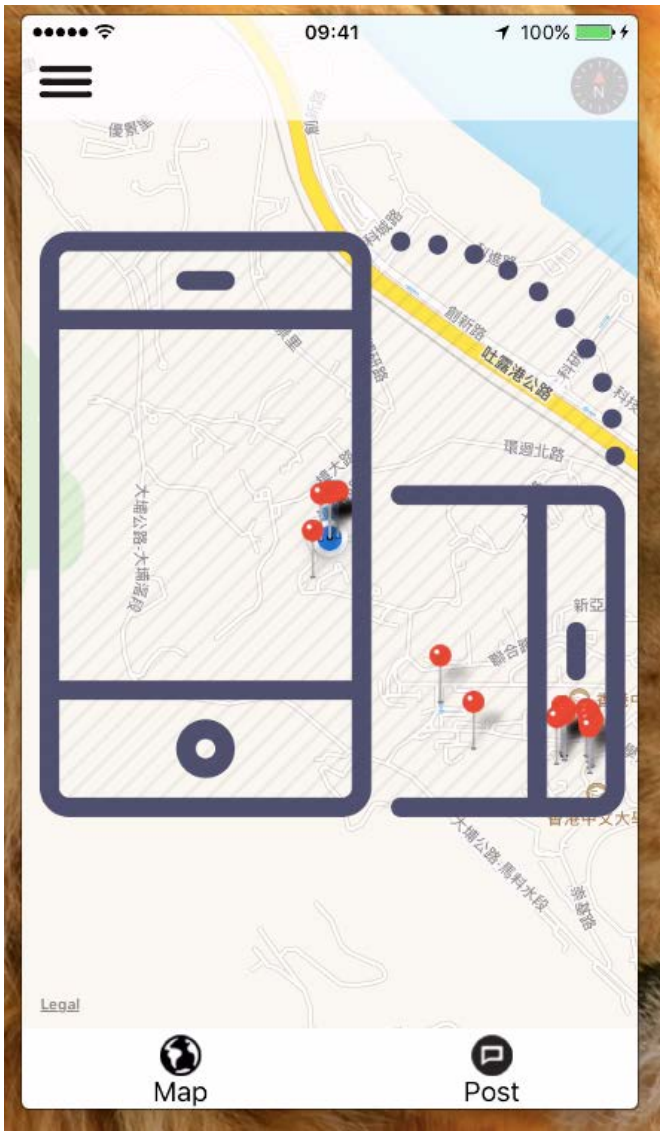
On map view, there is a Post button. By clicking the post button, it will switch to postMsgView, user can write their comment about this place at the postMsgView.



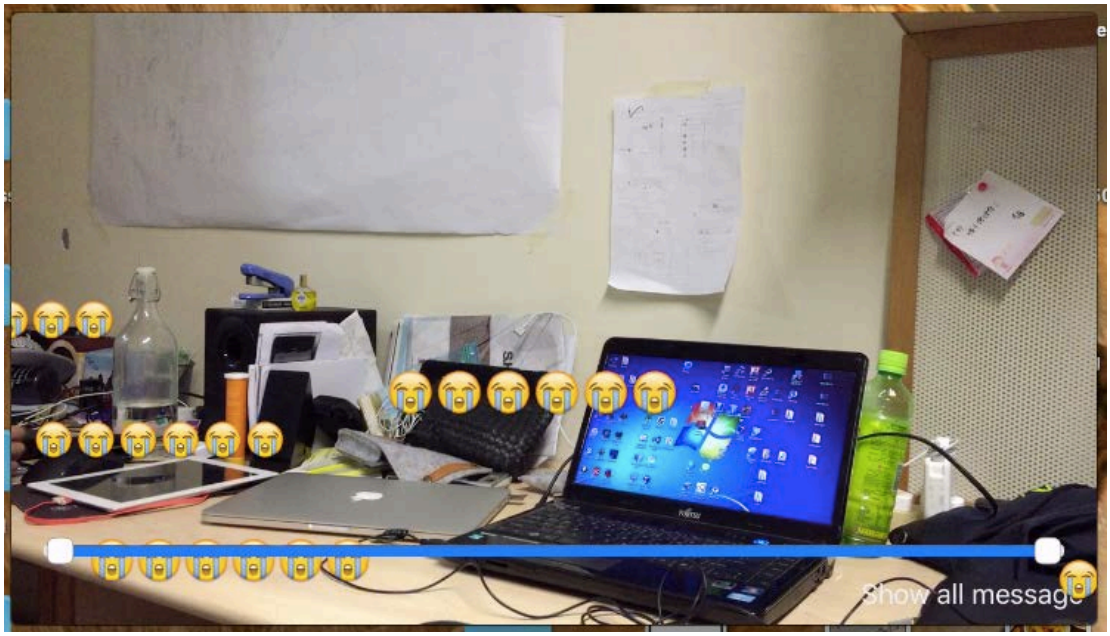
At the postMsgView, there is a white text field allow user to type in message content. The segment right below is to choose the tag category that is related to their message content. After finish typing, click on the enter button on the right hand side, the message will be uploaded to server and route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

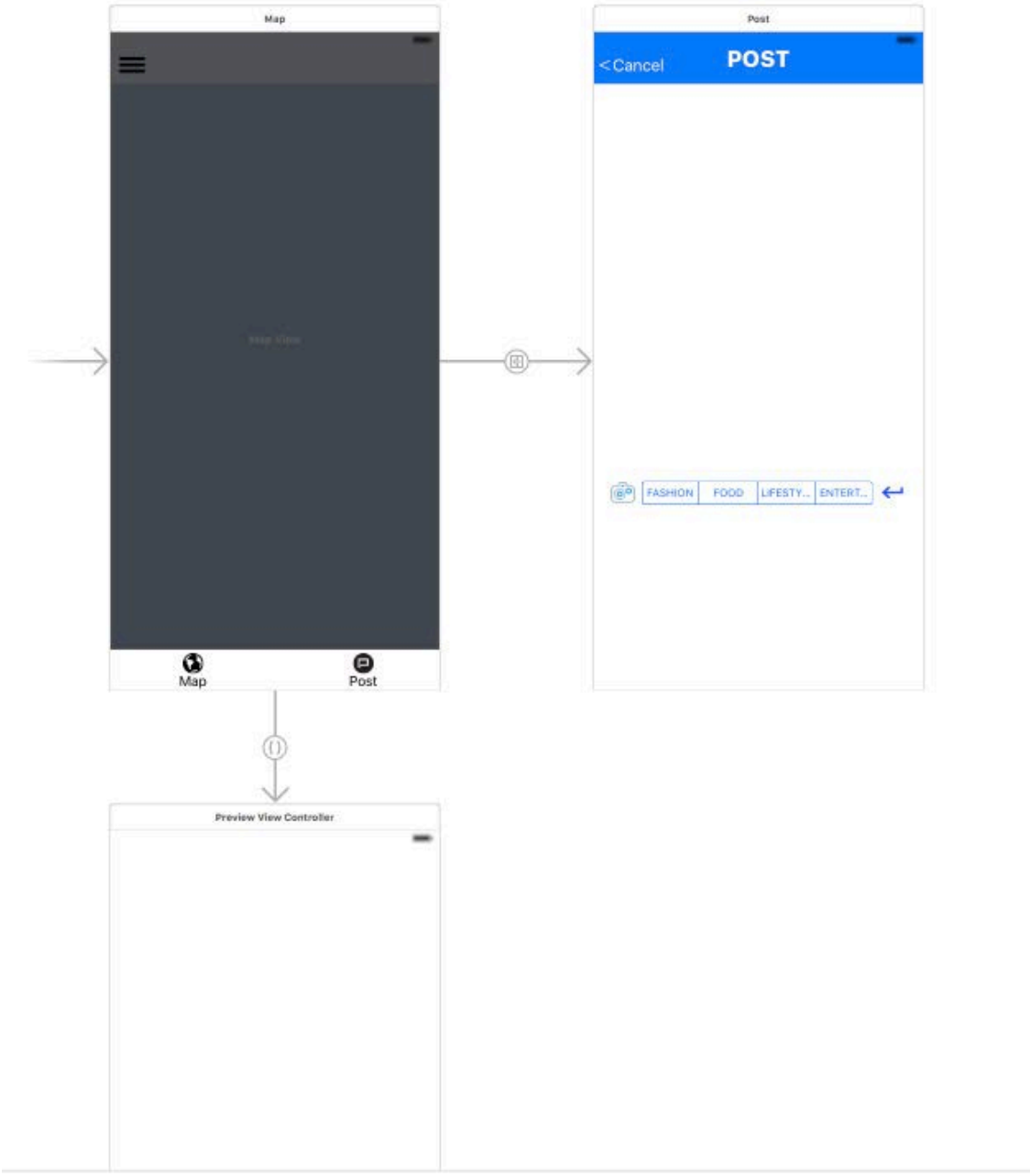
At the bottom of the message checking view, there is a range slider bar. Users can use that 2-sided slider bar to select the messages that post on that interval. The above example shows that i have select all the messages between today to 1 month ago. The camera shows the crying emoji that is the content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


```
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.

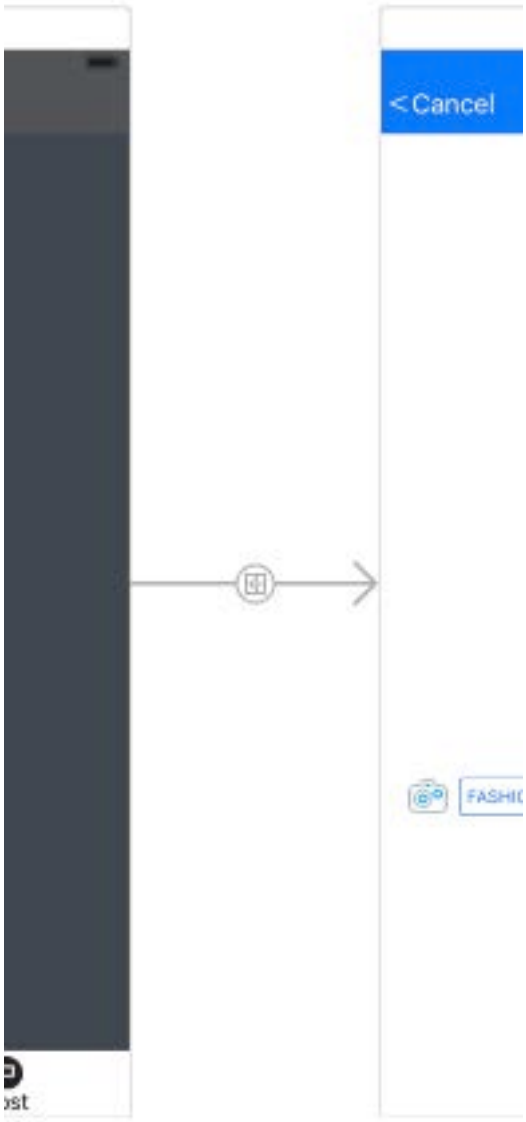
5.2.2.5 Implementation of postMsgViewController

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)
}
```

6. Limitations and Difficulties

6.1.Limitation of CLCircularRegion:

In the testing phase, we have tried CLCircularRegion for locating the messsages position. However, the region monitoring service (CLCircularRegion) provided by IOS API have 150 to 200 meters' variant that is not accurate enough for notifying any message nearby the user.

We first defined a region in Shaw College Student Hostel Two with 20 meters radius, although the GPS location is showing we are near the specific location, the region service cannot determine we are inside or outside the specified region or not. When we arrive at Residences 3 & 4, the region services eventually work by showing we are outside the defined region that is about 170 meter away from student Student Hostel Two. The performance of CLCiruclarRegion really disappoints us. We have to find another way to build a better region effect for our use.

```
let location = CLLocationManager()
let place1 = CLCircularRegion(center: CLLocationCoordinate2D(latitude: 22.422948, longitude: 114.201315), radius:
    20, identifier: "hostel 2")

override func viewDidLoad() {
    super.viewDidLoad()
    // Do any additional setup after loading the view, typically from a nib.
    //request the USAGE of GPS, add info.plist
    self.location.requestWhenInUseAuthorization()
    self.location.delegate = self
    self.location.desiredAccuracy = kCLLocationAccuracyBestForNavigation
    self.location.startUpdatingLocation()
    self.location.startUpdatingHeading()
    self.location.startMonitoringForRegion(self.place1)
}
```

```
func locationManager(manager: CLLocationManager, didEnterRegion region: CLRegion) {
    print("Enter hostel two")
}
func locationManager(manager: CLLocationManager, didExitRegion region: CLRegion) {
    print("fail enter")
}
func locationManager(manager: CLLocationManager, didDetermineState state: CLRegionState, forRegion region:
    CLRegion) {
    print("determ")
    switch state{
        case .Unknown:
        print("unknow");
        case .Inside:
        print("inside");
        case .Outside:
        print("outside");
    }
}
```

Finally, we have located the problem of the IOS API CLCiruclarRegion. This API only generate events only when boundary crossing, which means that if the user's location is already inside the region at registration time, the location manager doesn't automatically generate an event to notify the users. Users will only notify until crossing the region boundary. Which means users's inside that particular region cannot be updated immediately, that does now fulfill our expectation of updating messages in fast real time process. Also, each application can only monitor at most 20 regions at the same time. But our app is allowing users to leave messages in everywhere, it will encounter a big problem when there is huge number of messages and each single message is representing one region. Moreover, although the CLCircularRegion allow us to declare the radius from few meters to hundred meters, it only works well in approximately 200 meter and require at least 20 seconds for a notification to report. The respond of CLCiruclarRegion is quite slow, it does not meet requirement of quick respond of location information.

6.2. Limitation of default segue:

As mentioned before, IOS has it's own set of segue for transection. There are several problems in using the default segue.

- The major problem is that it does not have a segue for rotation.
- Although this can be done in changing the layout, it will mix the two modules get message and display into one. We think that separating into more modules is more convenient for further development.

6.3. Changing in Swift version and IOS version

At the testing phase, we have build location service function in Swift1.0 and IOS8 version. However, Xcode forced us to update to Swift 2.0. There are some changes in programming syntax and the API. In addition, when our devices upgrade to IOS9, all of our testing program immediately crash. We have to build most of the testing program again and acquire the changes of the Swift version. In addition, it is hard to find sample in Swift or detail usage report in Swift. As Swift is newly developed programming language, it is still not widely used, the resources in Swift as well as the Swift 2.0 is not enough. It takes us long time in studying the documentation of Swift.

6.4. Limitation in positioning System

As mentioned before, we aim at building a virtual notice board work well in both indoor and outdoor. At this stage we are using GPS for positioning. It works excellent is outdoor, but we found that it work with great error when the device come into an indoor. It is a must in improving the positioning system, so that this app can be more widely used in every scenario.

6.5. Difficulties in Coding

As it is our first time in building a mobile application, we are not familiar with way of building the program. We take a long time try and look for a better way in coding or using the API. We found that we still have a lot of space to improve the coding and programming efficiency to make the code more readable and clear.

7. Future Works (Milestone II)

So far milestone 1 is completed successfully, all the basics function required by a notice board is built and work well. In the coming semester, we will focus on enhancing the positioning system and interface design.

1. Supporting indoor positioning. The app is now adopting GPS to locate the position which with high accuracy in outdoor positioning. However, when it comes to an indoor scenario, GPS is really not suitable. We want virtual notice can be post message everywhere without any limitation. So it is necessary to enhance the positioning system that supports indoor usage.

   We decided to implement beacon technology that is now widely used in indoor positioning system.

2. Fancy UI in displaying messages content. A great user interface is very important in attracting user keep on using our app. In milestone 1, we have decided new way of checking message by rotating the phone. But it is still a along 2D plain text, we want more funny and interesting way in reading the message. Instead of displaying the message on 2D plain, we are going to build 3D displaying engines. Our target is via Google Cardboard to build a virtual reality for displaying message in 3 dimensional spaces.

8. Conclusion

The is a very good experience in building a mobile application, from design to implementation are all do e by ourselves.

The is our first time to build a mobile application; we have learnt a lot of new skills and concepts in writing an IOS application that is really useful for our future. During the development, we have encounter many problem, for example update of Swift and IOS version, get loss in how get start the whole development, little knowledge about IOS API, we still can stick on our first term target.

In this semester, we have built a virtual notice board with basic functions. It can read or write a message according to user position. Additionally, we have implement a filtering system for users to look the message that they. Also, we have design a segue for view switching so that display messages is can be done by a simple rotation of iPhone. Furthermore, we have built a server for storing all the messages post by a user.

With great experience from this semester, we believe that our self-learning skills and problem-solving skills have improved a lot. We are confident that we can do better in the coming semester. The next semester our target is improving and enhance the whole application. Internally, we are going to improve the positioning system so that POST can work well in both indoor and outdoor events. Externally, we will try to make the displaying view in 3D, as we want reading messages to be more fun and interesting.

9. 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

10. 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.