TRiGlobe

# PRODUCTION DATA – Our Own WS iGlobe

In this tut we will use a helper class called SantiappsHelper, which contains the code to run these connections to the web. A standalone class in such case is usually called a library, which takes care of those processes. If you only require one instance of such a class, as is our case, you create a Singleton pattern. You only want one instance of the connection because you don’t want many instances of the connection class creating, receiving and disconnecting. That could end up in multiple connections to the same resource at different times, which could confuse you or the server.

Ok, iGlobe was a game I created a while back which was meant to engage users by creating a competition which egged you on to get more points in order to beat your competitors and at the end the winner would get a money pot! Nice huh?

Basically we want the user to be able to tag a place or a person, get points because of that. In order to do this, the app has to interact with a web service we will create. What will we need our service to be able to do:

Store username information for each player

Receive users points

Present users points

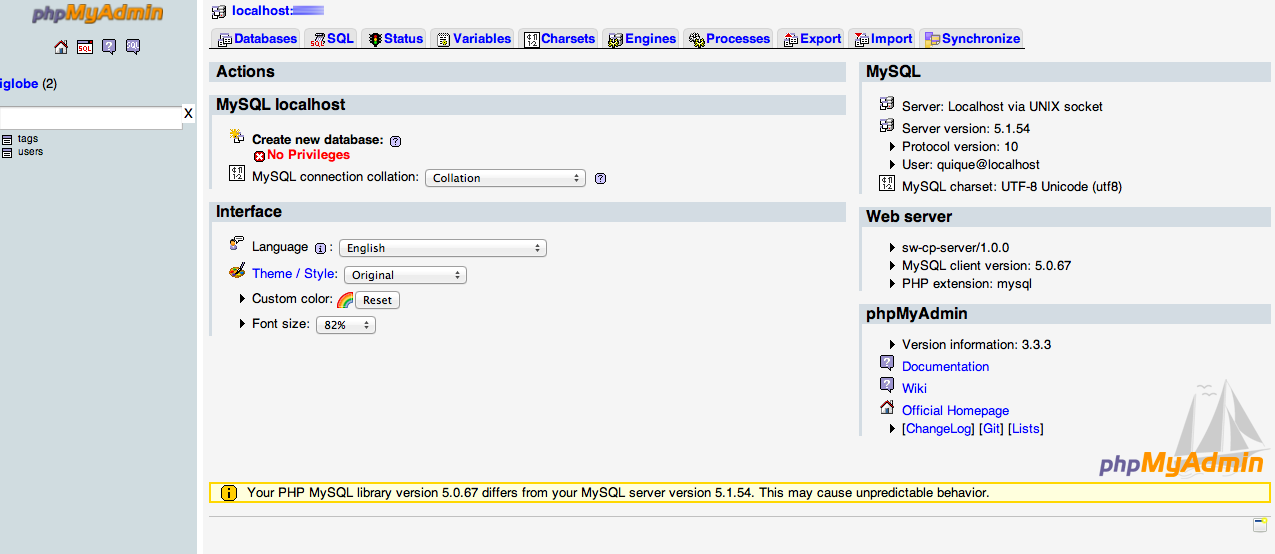
Sounds simple enough. Web services are usually large databases of information, so our database will need a table to store all this information we mentioned above. Databases come in a lot of flavors and the way we interact with them; Create, Read, Update and Delete (CRUD) data from them is just as varied. So let’s detour for a minute and talk a bit about databases, their structure and how we interact with them. Sounds simple but we should map out a route just in case:

1. **Create web database**
2. Create web service back end
3. Create iOS front end (Storyboard)
4. Fetch Data
   1. NSURLConnection
   2. GCD & Completion Blocks
5. Add Bump API
6. Throw in Social

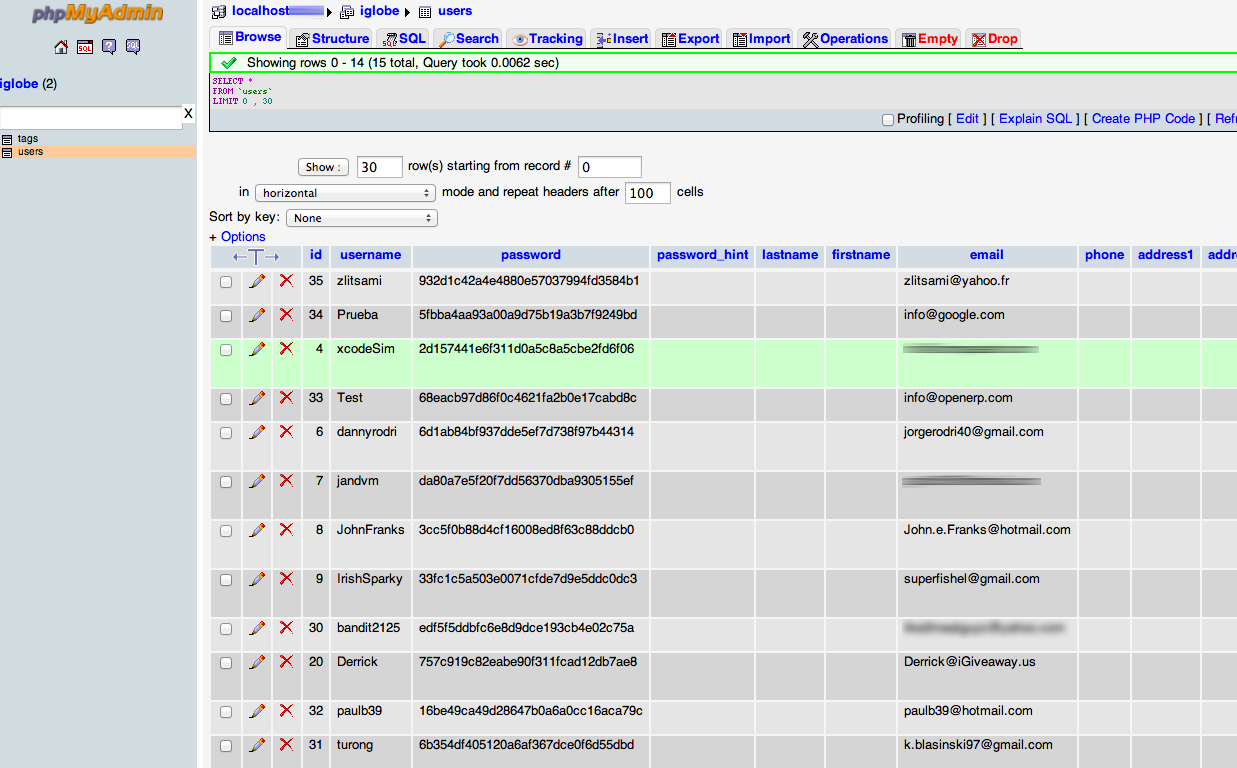
# Databases

Databases are simply information stores. Information stores can be written as files (such as word or powerpoint), but the information in such files has a predetermined structure that Word and Powerpoint know how to read and access in order to present you what you want and let you edit it and store it again. The problem is only Word will read a docx file and only powerpoint will read a pptx file. The great advantage of databases is that they store information in a very compact way. And they can be read by many different interfaces. The simpler the database is, the more interfaces can read it.

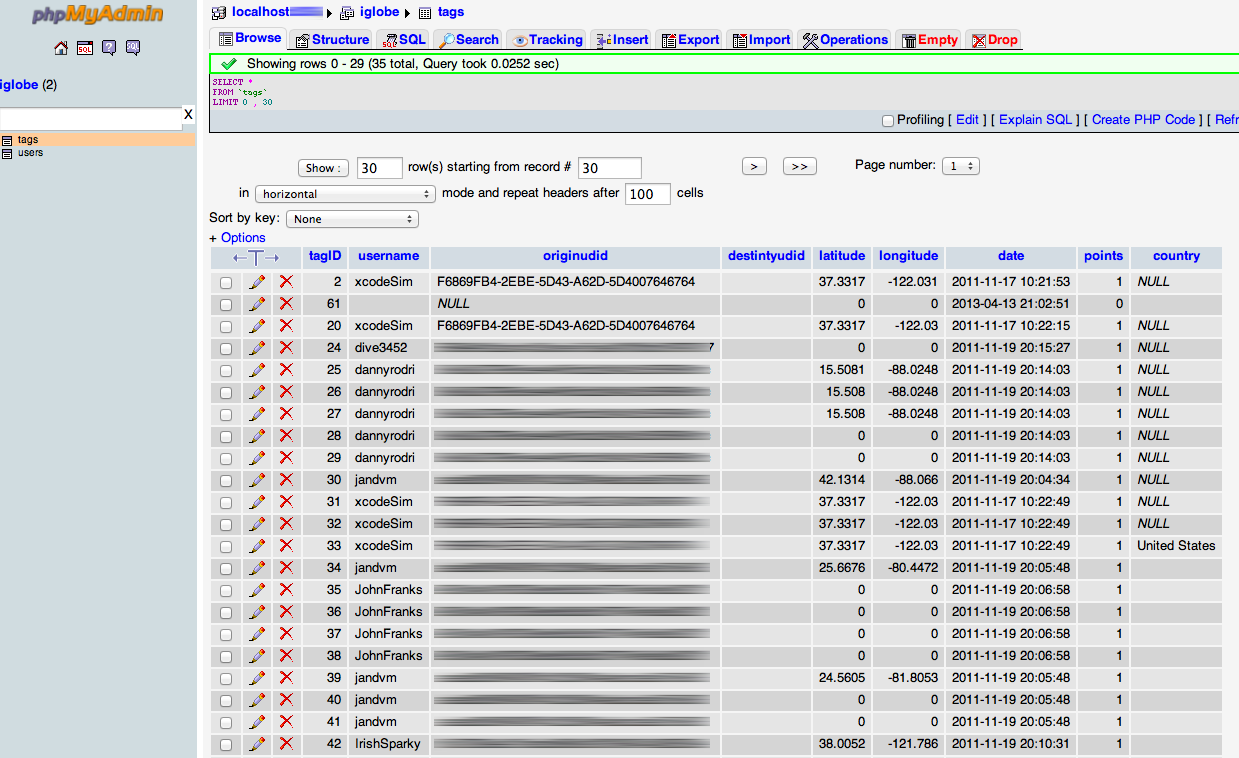
Ok so we will be using a database that is usually available for free in most web hosting services. My web hosting service has phpMySQL, which comes included with a free package. If you want other databases such as MySQL you would need a paid service. This is what my database management interface looks like:



As you can see we have a database named iglobe on localhost with 2 tables; a users and tags table. The users table looks like this:



It contains a primary key with a username, a password, password hint, first and lastname as well as an email, phone address and such regular stuff. If we take a look at the tags table:



This table also has its own primary key (tagID), the corresponding username, an identifier, the tag’s latitude and longitude as well as the date it was created and how many points its worth to that user. There is also a country field, which was implemented later as this project progressed. As you can see its been in the works since 2011.

The idea for our webservice will be to read from these tables and write to them whatever data users request or post to them. So this part requires you to know some PHP, which is basically a scripting language that is used on server side applications. Let’s start simply by looking at what the code to read a table looks like.

<?php

include\_once("JSON.php");

$json = new Services\_JSON();

$link = mysql\_pconnect("localhost", "user", "pass") or die("Could not connect");

mysql\_select\_db("iglobe") or die("Could not select database");

$arr = array();

$rs = mysql\_query("SELECT \* FROM users");

while($obj = mysql\_fetch\_object($rs)) {

$arr[] = $obj;

}

Echo $json->encode($arr);

?>

First we include the JSON.php file in order to access JSON files on your server. Make sure your webserver or host provides you with the latest version of PHP, at least 5.2. We then make a connection to the database using the database user and password as well as the database host. Now we create an array object so once we execute the mysql\_query where all entries from the users table are collected into $rs, we can put that object into our $arr[] object. Finally we encode the $arr into $json and echo it onto the screen.

Once this code is up and ready along with your database (including some records), you can direct your browser to this file (which I called myserver.com/getusers.php) and I get the following result:

[{"id":"35","username":"zlitsami ","password":"932d1c42a4e4880e57037994fd3584b1","password\_hint":"","lastname":"","firstname":"","email":"zlitsami@yahoo.fr","phone":"","address1":"","address2":"","city":"","state":"","zip":"","country":"","url":"","permissions":"1","udid":"9","userCreated":"2013-01-01 14:27:22","time\_queued":null,"time\_sent":null}, {another}, {another}]

This is an array, which has many elements in it. Each element is a users table entry. Each entry is a dictionary of key value pairs. Look familiar? ☺

Ok so now we know how to read info from our database, let’s create the code for writing to the database.

<?php

$con = mysql\_connect("localhost","user","pass");

if (!$con)

{

die('Could not connect: ' . mysql\_error());

}

mysql\_select\_db("iglobe", $con);

$sql="INSERT INTO tags (username, latitude, longitude, country,destintyudid,points) VALUES ('$\_POST[sender]','$\_POST[latitude]','$\_POST[longitude]','$\_POST[country]','$\_POST[receiver]','$\_POST[points]')";

if (!mysql\_query($sql,$con))

{

die('Error: ' . mysql\_error());

}

echo "1 record added to tags";

mysql\_close($con)

?>

Ok so we connect to our database again, we create a sql statement with values to insert (these values come from a form either online or on a mobile device). Finally we execute that sql statement with our connection and echo the results for verification to the user. I called this file writephp.php.

Now before we move onto iOS, let’s test our service online. Create an HTML file called Writeform.html and save this code to it:

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

<head>

</head>

<body>

<form action="writephp.php" method="post">

<table style="text-align: left; width: 100%;" border="0"

cellpadding="2" cellspacing="2">

<tbody>

<tr>

<td>Name:</td>

<td><input name="sender" type="text"></td>

<td></td>

</tr>

<tr>

<td>UDID(unnecessary):</td>

<td><input name="udid" type="text"></td>

<td></td>

</tr>

<tr>

<td>Latitude:</td>

<td><input name="latitude" type="text"></td>

<td></td>

</tr>

<tr>

<td>Longitude:</td>

<td><input name="longitude" type="text"></td>

<td></td>

</tr>

<tr>

<td>Country</td>

<td><input name="country" type="text"></td>

<td></td>

</tr>

<tr>

<td>Receiver</td>

<td><input name="receiver" type="text"></td>

<td></td>

</tr>

<tr>

<td></td>

<td></td>

<td><input type="submit"></td>

</tr>

</tbody>

</table>

&nbsp;&nbsp;&nbsp;&nbsp;<br>

</form>

</body>

</html>

Now load up the form on your web browser and submit some data to your database. Pretty simple huh? Ok so let’s do one more thing. I don’t want to make this webservice too complicated because I want to keep your attention on the iOS side. So let’s create a form to eventually read points from our webservice for a particular user. Create another html file called Testform.html and save this code to it:

<HTML>

<head>

<form action="readpoints.php" method="post">

User: <input type="text" name="userNa"/>

<input type="submit" />

</form>

</head>

</HTML>

And create its php counterpart:

<?php

include\_once("JSON.php");

$json = new Services\_JSON();

$link = mysql\_pconnect("localhost", "user", "pass") or die("Could not connect");

mysql\_select\_db("iglobe") or die("Could not select database");

$username = $\_POST["userNa"];

$result = mysql\_query("SELECT username, SUM(points) AS PUNTOS FROM tags WHERE username='$username' GROUP BY username");

// THIS RETURNS ARRAY NOT READ PROPERLY BY iOS JSON

$resultado = array();

while($obj = mysql\_fetch\_object($result)) {

$resultado[] = $obj;

}

Echo $json->encode($resultado);

?>

We will use this last bit of code later once we get some more data into the db.

Before moving on, let’s just think of something real quick here which has to do with performance. So far we have a resource that returns the points for a particular user, readpoints.php. This is what’s called a web service endpoint. Web services can have many endpoints. In a game or app, we might want to get a lot of user’s points at once, to fill up a leaderboard for example. We might want to fetch a lot of transactions from an invoice database instead of one by one, right? So let’s just get ahead of ourselves and create an endpoint to manage a set of input data. In our case, we must be able to pass the web service a set of users. Our file would look something like this:

<?php

include\_once("JSON.php");

$json = new Services\_JSON();

//1. PROCESS RECEIVED ARRAY

$handle = fopen("php://input", "rb");

$http\_raw\_post\_data = '';

while (!feof($handle)) {

$http\_raw\_post\_data .= fread($handle, 8192);

}

fclose($handle);

//1.1 Just decode to see what kind of object it is

$post\_data = json\_decode($http\_raw\_post\_data,true);

if (is\_array($post\_data))

$response = array("status" => "ok", "code" => 0, "original request" => $post\_data);

else

$response = array("status" => "error", "code" => -1, "original\_request" => $post\_data);

//2. CALL DB QUERY

$link = mysql\_pconnect("localhost", "quique", "sod123") or die("Could not connect");

mysql\_select\_db("iglobe") or die("Could not select database");

//3. CREATE FINAL ARRAY TO RETURN

$arrayToReturn = array();

//4. CYCLE THROUGH USERS

foreach ($post\_data as $value)

{

//CREATE QUERY

$result = mysql\_query("SELECT username, SUM(points) AS PUNTOS FROM tags WHERE username='$value' GROUP BY username");

//EXECUTE QUERY & ADD EACH USER/POINTS DICTIONARY TO $resultado ARRAY

$resultado = array();

while($obj = mysql\_fetch\_object($result)){

$arrayToReturn[] = $obj;

}

}

Echo $json->encode($arrayToReturn);

?>

Ok, this is some basic php code that just takes the passed in array as we mentioned and loops through the database to get the points for each user. This is important because we save the app a lot of trips to the web server database.

Following our roadmap, we will now work on our iOS Storyboard of front end. We will then hardcode data and finally webfecth data. The reason is that this way we can see what our frontend will require in terms of data models and then we can fetch web data and replace our data in those data models. We will also be learning 2 ways of fetching data: inline, messy code and neat and tidy coding. So here is our roadmap:

1. Create web database
2. Create web service back end
3. **Create iOS front end (Storyboard)**
4. Fetch Data
   1. NSURLConnection
   2. GCD & Completion Blocks
5. Add Bump API
6. Throw in Social

# Storyboard

First let’s create a new Empty project using Storyboards, ARC, iPhone and NO Core Data. Now go to the storyboard and drag a UITableViewController onto the grid. Now let’s create a class called UsersListViewController and in Storyboard, select the scene and in the Identity Inspector make our scene of type UsersListViewController from the dropdown list. Ok let’s run a quick test to make sure our tvc is working. Build & Run and you should get an empty tableview. Great! Now let’s review what we will do in this section. Here are the quick steps:

* Add an array property to your .m file.
* Prefill that array in viewDidLoad
* Eliminate the pesky warning lines
* Make tableview return 1 section
* Make tableview return array count
* Make tableview cell return array objects

This should be second nature to you by now so Ill blaze through the specifics.

Here is the property code:

@property (nonatomic, strong) NSArray \*testArray;

Here is the viewDidLoad code:

- (void)viewDidLoad

{

[super viewDidLoad];

self.testArray = [[NSArray alloc] initWithObjects:@"me", @"you", @"them", nil];

NSLog(@"array %d", [self.testArray count]);

}

Here is the return array count code:

return [self.testArray count];

And here is the cFRAIP code:

- (UITableViewCell \*)tableView:(UITableView \*)tableView cellForRowAtIndexPath:(NSIndexPath \*)indexPath

{

static NSString \*CellIdentifier = @"Cell";

UITableViewCell \*cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier forIndexPath:indexPath];

// Configure the cell...

cell.textLabel.text = [self.testArray objectAtIndex:indexPath.row];

return cell;

}

Before you build and run, select the UITableViewCell in Storyboard and in the Attributes Inspector make sure you use Cell as the Reuse Identifier. Now your app should work fine.

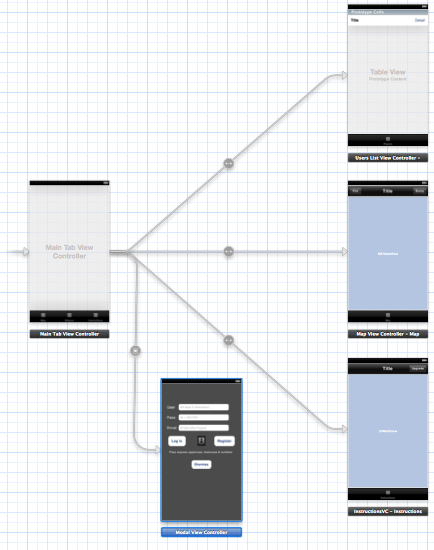
If you Build & Run now you should get your users displayed in the tableview. Cool! That’s what we are going to want to do, display a list of users in a tableview and then add in the points, like a score table.

So let’s look at what a mock up of our app will look like. In essence we will have a tab bar controller manage 3 views; a Users view, a Map View and an Instructions View. We will also throw in a Login view as the app launches.

This should give you an idea of what kinds of tasks we will need to perform in order to accomplish this:

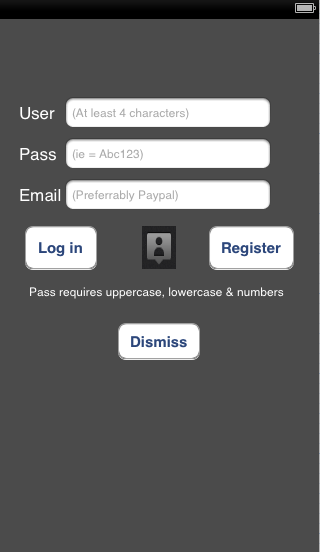
1. Present a login view controller
2. Save user and pass information
3. Fetch user data from the web service
4. Plot points on a map
5. Display instructions in a view

Ok so here is what the mock up looks like:



You should be able to re-create this in your storyboard. Here are the basic steps:

1. Select your existing UsersViewController scene and from the Editor menu select Embed In->Tab Bar Controller. You should have a Scene and a Class for UsersViewController and the scene should be set to its Class Type in the Identity Inspector.
2. Now clear out the second scene that was added when you embedded your tableview scene in a tab bar. By clear out I mean just make sure it doesn’t have any labels or other controls in it. Now drag a UIMapvView into it. Add a UINavigationBar to the top and 2 buttons on either side. One is the Plot button and the other is the Bump button. Create a MapViewController class for it and set its type. Finally add a MKMapView IBOutlet property and 2 UIBarButtonItem IBOutlet properties and connect them. Add the MKMapViewDelegate.
3. For the last view add another UIViewController and drag a UIWebView and a UINavigationBar into it. Create its class file and name it InstructionsVC. Add a UIWebView IBOutlet property and connect it. Also add the UIWebViewDelegate and don’t forget to set its scene type.
4. Last thing will be to add a UIViewController and call it ModalViewController. This what mine looks like:



Go ahead and create all the IBOutlet properties for it. That’s 4 labels with static text (User, Pass, Email and Pass requires…). Then there are 3 UITextFields with placeholder text to guide the user. There are 3 UIButtons for different actions and finally that person icon is actually a button with a Background Image set to the image. It will be the button the users will use to upload their image to the web server.

1. There are other class files we could create right now which are the TagListController & Tag/Users Model & Annotation/PlacemarkVC…

# NSURLConnection

Moving on to Part 2 of TRiGlobe.

Ok now we are going to connect to our webservice and fetch some actual data. First let’s add some properties to our UsersListViewController.m:

@interface UsersListViewController () <NSURLConnectionDelegate>

@property (nonatomic, strong) NSMutableArray \*testArray;

@property (nonatomic, strong) NSMutableData \*buffer;

@property (strong, nonatomic) IBOutlet UIActivityIndicatorView \*spinner;

@property (nonatomic, strong) NSURLConnection \*myConnection;

Now we are going to use NSURLConnection to create a web connection and fetch some data, so let’s change our viewDidLoad method to look like this:

- (void)viewDidLoad{

[super viewDidLoad];

// Animate the spinner

[self.spinner startAnimating];

// Create the URL & URLRequest

NSURL \*myURL = [NSURL URLWithString:@"http://www.yourserver.com/iglobe/getusers.php"];

NSURLRequest \*myRequest = [NSURLRequest requestWithURL:myURL];

// Create the connection

self.myConnection = [NSURLConnection connectionWithRequest:myRequest delegate:self];

//Test to make sure the connection worked

if (self.myConnection){

self.buffer = [NSMutableData data];

[self.myConnection start];

}else{

NSLog(@"Connection Failed");

}

}

Ok now NSURLConenction has 4 delegate methods you must implement. So first, let’s make sure to add the <NSURLConnectionDelegate> to our @interface line for UsersListViewController.h (or in .m – just make sure it’s the @interface line and not the @implementation line). Now let’s add the first method:

# pragma NSURLConnection Methods

- (void)connection:(NSURLConnection \*)connection didReceiveResponse:(NSURLResponse \*)response{

[self.buffer setLength:0];

}

This method is called when the app receives a response from the server. We will simply reset the buffer length. Now we must deal with each time the app receives data from the server:

- (void)connection:(NSURLConnection \*)connection didReceiveData:(NSData \*)data{

[self.buffer appendData:data];

}

We simply append the received data to the existing buffer data. Let’s handle any error response from the server as well:

- (void)connection:(NSURLConnection \*)connection didFailWithError:(NSError \*)error {

// Do cleanup

self.myConnection = nil;

self.buffer = nil;

// Inform the user, most likely in a UIAlert

NSLog(@"Connection failed! Error - %@ %@",

[error localizedDescription],

[[error userInfo] objectForKey:NSURLErrorFailingURLStringErrorKey]);

}

Finally let’s implement the method that is called when the connection finishes loading data from the web server:

- (void)connectionDidFinishLoading:(NSURLConnection \*)connection {

NSLog(@"Succeeded!");

//Create a queue and dispatch the parsing of the data

dispatch\_async(dispatch\_get\_global\_queue(DISPATCH\_QUEUE\_PRIORITY\_DEFAULT, 0), ^{

// Parse the data from JSON to an array

NSError \*error = nil;

NSArray \*jsonString = [NSJSONSerialization JSONObjectWithData:\_buffer options:NSJSONReadingMutableContainers error:&error];

// Return to the main queue to handle the data & UI

dispatch\_async(dispatch\_get\_main\_queue(), ^{

//Check if error or not

if (!error) {

//If no error then PROCESS ARRAY

self.testArray = [[NSMutableArray alloc] initWithCapacity:50];

for (NSDictionary \*tempDictionary in jsonString) { // Extract each dictionary’s username & put it into our array

[self.testArray addObject:[tempDictionary objectForKey:@"username"]];

}

// Call reload in order to refresh the tableview

[self.tableView reloadData];

}else{

NSLog(@"ERROR %@", [error localizedDescription]);

}

//Stop animating the spinner

[self.spinner stopAnimating];

// Do cleanup

self.myConnection = nil;

self.buffer = nil;

});

});

}

Well that’s quite a bit of code but its actually pretty simple. The important bit is that we have seen the web response logged in our console thus we know its an array. We also know the array has dictionaries at each index. Therefore we must loop or iterate through each NSDictionary entry in the array and fetch the “username” key’s value or object. We add that object to our self.testArray with each new iteration. In the end we refresh our tableview to use the newly populated self.testArray with our usernames.

Great so we can read information from our webservice. As we will see later, reading more complex data from the webservice is just a matter of creating a more complex request string on the iOS side and putting it together with some logic in the php server side. This is one way to fetch data, using NSURLConnection directly in a viewDidLoad. It’s better than calling NSURLConnection on the main thread, that’s for sure. But we want to make sure our code is re-usable, particularly the web fetch code since it’s probably code we will want to use again in the future for many projects to come. That’s the reason we created the SantiappsHelper class.

So let’s revisit our roadmap:

1. Create web database
2. Create web service back end
3. Create iOS front end (Storyboard)
4. Fetch Data
   1. **NSURLConnection**
   2. GCD & Completion Blocks
5. Add Bump API
6. Throw in Social

# HelperClass – GCD & Completion Blocks

So we want to make our code a bit neater and more portable. We will achieve this by creating our SantiappsHelper Class.

This helper class will concentrate on fetching data from the web. Its very similar to other class files you have worked with before but basically it has only Class Methods.

#import <Foundation/Foundation.h>

#import "Tag.h"

typedef void (^Handler)(NSArray \*users);

typedef void (^Handler2)(NSArray \*points);

typedef void (^Handler3)(NSArray \*usersPointsArray);

@interface SantiappsHelper : NSObject {

}

+(void)fetchUsersWithCompletionHandler:(Handler)handler;

+(void)fetchPointForUsersArray:(NSArray\*)usersArray WithCompletionHandler:(Handler3)handler;

+ (BOOL)postNewTag:(Tag\*)passingObject;// from gamebumpconnector

@end

Ok these 3 methods do the following: (1) fetchUsersWithCompletionHandler will basically fetch the list of users in the game, (2) fetchPointForUsersArray: WithCompletionHandler: will fetch the points for all users and (3) postNewTag will create a new location tag for a particular user.

The idea here is that the user will exchange tokens or tags with another user. Physically the two users at the same location (obviously) will bump phones to initiate a tag worth 2 points. As the data is exchanged, each user account will post a 2-point value tag to the database. Originally the game was built to create tags individually at locations but you cant very well call it a game of Tag if you don’t tag a second user, can you? ☺

Let’s review what these methods do:

// THIS METHOD FETCHES USER ARRAY

+(void)fetchUsersWithCompletionHandler:(Handler)handler {

NSString \*urlString = [NSString stringWithFormat:@"http://www.myserver.com/myApp/getusers.php"];

NSURL \*url = [NSURL URLWithString:urlString];

NSMutableURLRequest \*request = [NSMutableURLRequest requestWithURL:url cachePolicy:NSURLRequestReloadIgnoringLocalAndRemoteCacheData timeoutInterval:10];

[request setHTTPMethod: @"GET"];

\_\_block NSArray \*usersArray = [[NSArray alloc] init];

dispatch\_async(dispatch\_get\_global\_queue(DISPATCH\_QUEUE\_PRIORITY\_BACKGROUND, 0), ^{

// Peform the request

NSURLResponse \*response;

NSError \*error = nil;

NSData \*receivedData = [NSURLConnection sendSynchronousRequest:request

returningResponse:&response

error:&error];

if (error) {

// Deal with your error

if ([response isKindOfClass:[NSHTTPURLResponse class]]) {

NSHTTPURLResponse \*httpResponse = (NSHTTPURLResponse\*)response;

NSLog(@"HTTP Error: %d %@", httpResponse.statusCode, error);

return;

}

NSLog(@"Error %@", error);

return;

}

NSString \*responseString = [[NSString alloc] initWithData:receivedData encoding:NSUTF8StringEncoding];

//NSLog(@"responseString fetchUsers %@", responseString);

usersArray = [NSJSONSerialization JSONObjectWithData:[responseString dataUsingEncoding:NSASCIIStringEncoding] options:0 error:nil];

//Returns handler

if (handler) {

dispatch\_async(dispatch\_get\_main\_queue(), ^{

handler(usersArray);

});

}

});

}

//Fetches points for users array

+(void)fetchPointForUsersArray:(NSArray\*)usersArray WithCompletionHandler:(Handler3)handler{

NSError \*error = nil;

NSData \*data = [NSJSONSerialization dataWithJSONObject:usersArray options:0 error:&error];

if (error)

NSLog(@"%s: JSON encode error: %@", \_\_FUNCTION\_\_, error);

// create the request

NSURL \*url = [NSURL URLWithString:@"http:/ /www.myserver.com/myApp/readpointsforarray.php"];

NSMutableURLRequest \*request = [NSMutableURLRequest requestWithURL:url];

[request setHTTPMethod:@"POST"];

[request setValue:@"application/json; charset=utf-8" forHTTPHeaderField:@"Content-Type"];

[request setHTTPBody:data];

\_\_block NSArray \*pointsArray = [[NSArray alloc] init];

dispatch\_async(dispatch\_get\_global\_queue(DISPATCH\_QUEUE\_PRIORITY\_BACKGROUND, 0), ^{

// Peform the request

NSURLResponse \*response;

NSError \*error = nil;

// issue the request

NSData \*returnData = [NSURLConnection sendSynchronousRequest:request returningResponse:&response error:&error];

if (error) {

// Deal with your error

if ([response isKindOfClass:[NSHTTPURLResponse class]]) {

NSHTTPURLResponse \*httpResponse = (NSHTTPURLResponse\*)response;

NSLog(@"HTTP Error: %d %@", httpResponse.statusCode, error);

//return;

}

NSLog(@"Error %@", error);

//return;

}

NSString \*responseString = [[NSString alloc] initWithData:returnData encoding:NSUTF8StringEncoding];

//NSLog(@"asyncrhonous: %@",responseString);

pointsArray = [NSJSONSerialization JSONObjectWithData:[responseString dataUsingEncoding:NSASCIIStringEncoding] options:0 error:nil];

//NSLog(@"pointsArray %@", pointsArray);

if (handler){

dispatch\_async(dispatch\_get\_main\_queue(), ^{

handler(pointsArray);

});

}

});

}

// Called from MKViewController, creates shared tags points=2

+ (BOOL)postNewTag:(Tag\*)passingObject{

//1. Log the tag for verification first

NSLog(@"passingObject:%@,%@,%@,%@,%@",passingObject.sender, passingObject.receiver, passingObject.rglatitude, passingObject.rglongitude, passingObject.rgcountry);

//NSLog(@"tagReceived:%@,%@,%@,%@",tagReceived.originUdid, tagReceived.destintyUdid, tagReceived.rglatitude, tagReceived.rglongitude);

//2.REBUILD status string from passingObject

NSString \*s1 = [[NSString alloc] initWithFormat:@"sender=%@&latitude=%@&longitude=%@&country=%@&receiver=%@&points=2",passingObject.sender,passingObject.rglatitude,passingObject.rglongitude,passingObject.rgcountry,passingObject.receiver];

//3. Post tag to cloud

NSData \*postData = [s1 dataUsingEncoding:NSUTF8StringEncoding allowLossyConversion:YES];

NSString \*postLength = [NSString stringWithFormat:@"%d", [postData length]];

NSMutableURLRequest \*request = [[[NSMutableURLRequest alloc] init] autorelease];

NSURL \*url = [NSURL URLWithString:[NSString stringWithFormat:@"http:/ /www.myserver.com/myApp/writephp.php"]];

[request setURL:url];

[request setHTTPMethod:@"POST"];

[request setValue:postLength forHTTPHeaderField:@"Content-Length"];

[request setValue:@"application/x-www-form-urlencoded" forHTTPHeaderField:@"Content-Type"];

[request setHTTPBody:postData];

NSURLResponse \*response;

NSError \*error;

// We should probably be parsing the data returned by this call, for now just check the error.

[NSURLConnection sendSynchronousRequest:request returningResponse:&response error:&error];

NSLog(@"success!");

[s1 release];

return (error == nil);

}

fetchUsersWithCompletionHandler:

If you noticed at the beginning of our helper class we defined 3 typedefs. This just means we are defining a type, which we called Handler. We will see in a minute what these are used for. So basically this method creates a GET type request, which will be used to GET our users from the webservice we created. It creates a block array to be used inside the block to store our returned data. Before executing the block, which contains the NSURLConnection method call, we dispatch the operation to the background queue and execute it there. This means that a slow or busy server will not block our main thread. The main thread is responsible for drawing operations and user interaction. If a heavy-duty operation such as fetching data from a web server or processing images and video were to be run on the main thread, the user would not be able to interact with the app until that task was complete.

The handler is the typedef that is created once the method has completed. Therefore we check to see if it exists and once it does exist, we call back to the main thread, returning the now populated usersArray. It is that usersArray that will be used to fill our list of users in our tableview.

fetchPointForUsersArray: WithCompletionHandler

This method is basically responsible for sending in an array of users and fetching their points back. It talks to the last php file we created and returns the usersArray, which contains a dictionary with users and their points.

postNewTag

The final method is in charge of simply logging a newly created tag and then posting it to the database. So if you step back a little bit, you can see we have one method for writing data to the database (postNewTag) and two methods for reading data from the database (fetchUsers… and fetchPointForUsersArray…). We read and write points (or Tags) and read users. We must have a way to write users. We will cover this later on in a class called ModalViewController which will accept a login from a user o create a new user. So let’s review where we are:

1. Create web database
2. Create web service back end
3. Create iOS front end (Storyboard)
4. Fetch Data
   1. NSURLConnection
   2. GCD & Completion Blocks
5. **Add Bump API**
6. Throw in Social

Before we bump phones and exchange tags we must be able to create tags. So let’s create a Tag Class and its ViewController and plot them on the map view. Once we have that we will then save our user info to our app and be ready to bump!

LoginVC **-** Save User

Tag Class

Tag VC

MapView & Annotations

Plot method

NSUserDefaults is a pretty simple concept. An iOS app basically stores certain data locally. We will start by going to our ModalVIewController Class and adding some properties and methods. Then we will work on some php code for our backend.

First import the Santiapps Helper class created above into our .h file. While you’re there add some protocols, properties and methods:

#import <UIKit/UIKit.h>

#import "SantiappsHelper.h"

@interface ModalViewController : UIViewController <UINavigationControllerDelegate, UIImagePickerControllerDelegate>{

}

@property (nonatomic,retain) NSURLConnection \*myConnection;

@property (nonatomic,retain) NSMutableData \*incomingPostData;

@property (nonatomic,retain) IBOutlet UIButton \*registerUser;

@property (nonatomic,retain) IBOutlet UIButton \*loginUser;

@property (nonatomic,retain) IBOutlet UIButton \*dismissModal;

@property (nonatomic,retain) IBOutlet UITextField \*userName;

@property (nonatomic,retain) IBOutlet UITextField \*userPass;

@property (nonatomic,retain) IBOutlet UITextField \*userEmail;

-(IBAction)userWillDismiss;

-(IBAction)userWillLogIn;

-(IBAction)userWillRegister;

-(NSDictionary\*)addUser:(NSString \*)usuario withPass:(NSString \*)clave withAddress:(NSString\*)direccion;

-(NSString\*)checkUserLogin:(NSString \*)loginUsuario withPass:(NSString \*)loginClave;

@end

The navigation controller delegate is required along with the UIImagePickerControllerDelegate for picking images and we will use them for when the user uploads his picture.

As for properties, we will revisit our old friend, NSURLConnection, for which we require NSMutableData. Finally connect the buttons and textfields to their controls.

The methods will be used for when the user dismisses the login screen. Its always important to add a way out of a login screen. Even if that means the user will have limited access to the app. The login method will do just that whereas the register method will create a new user. Finally we will use the other methods for checking a login request.

Let’s review what these look like:

#import "ModalViewController.h"

@implementation ModalViewController

-(IBAction)userWillLogIn{

//Call checklogin.php to compare user/pass and return value

NSString \*loginSuccess = [self checkUserLogin:self.userName.text withPass:self.userPass.text];

if ([loginSuccess isEqualToString:@"SUCCESS"]) {

NSLog(@"We did it...");

//If all is well, then store userDefaults

NSUserDefaults \*prefs = [NSUserDefaults standardUserDefaults];

[prefs setObject:self.userName.text forKey:@"storedUser"];

[prefs setObject:self.userPass.text forKey:@"storedPass"];

[prefs synchronize];

[self dismissViewControllerAnimated:YES completion:nil];

} else {

NSLog(@"Authentication Failed..");

//Add UIAlertView

UIAlertView \*alertView = [[UIAlertView alloc] initWithTitle:@"Oops..." message:loginSuccess delegate:nil cancelButtonTitle:@"OK" otherButtonTitles:nil];

[alertView show];

[alertView release];

}

//if checklogin.php = OK then dismiss

}

-(IBAction)userWillRegister{

// Create uniqueIdentifier

//UIDevice \*device = [UIDevice currentDevice];

//NSString \*uniqueIdentifier = [device uniqueIdentifier];

//1. Get the LOGIN errorsDictionary

NSDictionary \*errorsDict = [self addUser:self.userName.text withPass:self.userPass.text withAddress:self.userEmail.text];

//2. if dictcount = 1 then dismiss && = 5926, then OK

if ([[errorsDict objectForKey:@"code1"] intValue] == 5926) {

NSLog(@"yeay");

//If all is well, then store userDefaults

NSUserDefaults \*prefs = [NSUserDefaults standardUserDefaults];

[prefs setObject:self.userName.text forKey:@"storedUser"];

[prefs setObject:self.userPass.text forKey:@"storedPass"];

[prefs synchronize];

//Dismis ModalVC after users first login

[self dismissViewControllerAnimated:YES completion:nil];

} else {

NSLog(@"Errors");

// extract nsdict

NSMutableString \*resultString = [NSMutableString string];

for (NSString\* key in [errorsDict allKeys]){

if ([resultString length]>0)

[resultString appendString:@"&"];

[resultString appendFormat:@"%@=%@", key, [errorsDict objectForKey:key]];

}

NSLog(@"rS:%@",resultString);

//Add UIAlertView

UIAlertView \*alertView = [[UIAlertView alloc] initWithTitle:@"Check your user,pass or email" message:resultString delegate:nil cancelButtonTitle:@"OK" otherButtonTitles:nil];

[alertView show];

[alertView release];

}

}

-(IBAction)userWillDismiss{

[self dismissViewControllerAnimated:YES completion:nil];

}

//registering user----moved to modalVC

-(NSString\*)checkUserLogin:(NSString \*)loginUsuario withPass:(NSString \*)loginClave{

//CREATE URL TO SEND

NSString \*urlString = [NSString stringWithFormat:@"username=%@&password=%@",loginUsuario,loginClave];

NSLog(@"login string:%@",urlString);

//POST THE STRING

NSData \*postData = [urlString dataUsingEncoding:NSUTF8StringEncoding allowLossyConversion:YES];

NSString \*postLength = [NSString stringWithFormat:@"%d", [postData length]];

NSMutableURLRequest \*request = [[[NSMutableURLRequest alloc] init] autorelease];

NSURL \*url = [NSURL URLWithString:[NSString stringWithFormat:@"http://www.yourserver.com/iGlobe/login2/checklogin.php"]];

[request setURL:url];

[request setHTTPMethod:@"POST"];

[request setValue:postLength forHTTPHeaderField:@"Content-Length"];

[request setValue:@"application/x-www-form-urlencoded" forHTTPHeaderField:@"Content-Type"];

[request setHTTPBody:postData];

NSURLResponse \*response = nil;

NSError \*error = nil;

// We should probably be parsing the data returned by this call, for now just check the error.

NSData \*myData = [NSURLConnection sendSynchronousRequest:request returningResponse:&response error:&error];

//NSError \*outError = NULL;

//NSDictionary \*tempDict = [NSDictionary dictionaryWithJSONData:myData error:&outError];

NSString \*string=[[NSString alloc] initWithData:myData encoding:NSUTF8StringEncoding ];

//NSLog(@"string es:%@, %i, %C",string,[string length],[string characterAtIndex:7]);

NSLog(@"string ess:%@",string);

//NSLog(@"Dict of errors:%@",tempDict);

return string;

}

-(NSDictionary\*)addUser:(NSString \*)usuario withPass:(NSString \*)clave withAddress:(NSString\*)direccion{

//CREATE URL TO SEND

NSString \*urlString = [NSString stringWithFormat:@"username=%@&password=%@&email=%@",usuario,clave,direccion];

NSLog(@"user registration string:%@",urlString);

//POST THE STRING

NSData \*postData = [urlString dataUsingEncoding:NSUTF8StringEncoding allowLossyConversion:YES];

NSString \*postLength = [NSString stringWithFormat:@"%d", [postData length]];

NSMutableURLRequest \*request = [[[NSMutableURLRequest alloc] init] autorelease];

NSURL \*url = [NSURL URLWithString:[NSString stringWithFormat:@"http://www.yourserver.com/iGlobe/login2/user\_add\_save.php"]];

[request setURL:url];

[request setHTTPMethod:@"POST"];

[request setValue:postLength forHTTPHeaderField:@"Content-Length"];

[request setValue:@"application/x-www-form-urlencoded" forHTTPHeaderField:@"Content-Type"];

[request setHTTPBody:postData];

NSURLResponse \*response = nil;

NSError \*error = nil;

// We should probably be parsing the data returned by this call, for now just check the error.

NSData \*myData = [NSURLConnection sendSynchronousRequest:request returningResponse:&response error:&error];

NSError \*outError = NULL;

NSDictionary \*tempDict = [NSJSONSerialization JSONObjectWithData:myData options:NSJSONReadingMutableContainers error:&outError];

//dataWithJSONObject:myData options:NSJSONWritingPrettyPrinted error:&outError];

//dictionaryWithJSONData:myData error:&outError];

//NSString \*string=[[NSString alloc] initWithData:myData encoding:NSUTF8StringEncoding ];

NSLog(@"Dict of errors:%@",tempDict);

return tempDict;

}

///////////////////////////////////----------------PHOTO UPLOAD-----

-(IBAction)uploadPhoto:(id)sender{

NSLog(@"picker");

UIImagePickerController \*picker = [[UIImagePickerController alloc] init];

picker.delegate = self;

picker.sourceType = UIImagePickerControllerSourceTypePhotoLibrary;

[self presentViewController:picker animated:YES completion:nil];

}

-(void)imagePickerController:(UIImagePickerController \*)picker didFinishPickingMediaWithInfo:(NSDictionary \*)info{

NSLog(@"pickerPicked");

UIImage \*profile\_image = [info objectForKey:UIImagePickerControllerOriginalImage];

[self uploadImage:UIImageJPEGRepresentation(profile\_image, 1.0) filename:@"globe57.png"];

[picker dismissViewControllerAnimated:YES completion:nil];

}

-(void)imagePickerControllerDidCancel:(UIImagePickerController \*)picker

{

[picker dismissViewControllerAnimated:YES completion:nil];

}

- (BOOL)uploadImage:(NSData \*)imageData filename:(NSString \*)filename{

NSLog(@"uploading");

NSString \*urlString = @"http://www.yourserver.com/iGlobe/photos/uploadPhoto.php";

NSMutableURLRequest \*request = [[[NSMutableURLRequest alloc] init] autorelease];

[request setURL:[NSURL URLWithString:urlString]];

[request setHTTPMethod:@"POST"];

NSString \*boundary = @"0xKhTmLbOuNdArY";

NSString \*contentType = [NSString stringWithFormat:@"multipart/form-data; boundary=%@",boundary];

[request addValue:contentType forHTTPHeaderField: @"Content-Type"];

NSMutableData \*body = [NSMutableData data];

[body appendData:[[NSString stringWithFormat:@"\r\n--%@\r\n",boundary] dataUsingEncoding:NSUTF8StringEncoding]];

//Set the filename

[body appendData:[[NSString stringWithString:[NSString stringWithFormat:@"Content-Disposition: form-data; name=\"userfile\"; filename=\"%@\"\r\n",filename]] dataUsingEncoding:NSUTF8StringEncoding]];

[body appendData:[@"Content-Type: application/octet-stream\r\n\r\n" dataUsingEncoding:NSUTF8StringEncoding]];

//append the image data

[body appendData:[NSData dataWithData:imageData]];

[body appendData:[[NSString stringWithFormat:@"\r\n--%@--\r\n",boundary] dataUsingEncoding:NSUTF8StringEncoding]];

[request setHTTPBody:body];

NSData \*returnData = [NSURLConnection sendSynchronousRequest:request returningResponse:nil error:nil];

NSString \*returnString = [[[NSString alloc] initWithData:returnData encoding:NSUTF8StringEncoding] autorelease];

NSLog(@"returningOKString");

return ([returnString isEqualToString:@"OK"]);

}

//////////////////////////////

- (void)didReceiveMemoryWarning {

// Releases the view if it doesn't have a superview.

[super didReceiveMemoryWarning];

// Release any cached data, images, etc. that aren't in use.

}

- (void)viewDidUnload {

[super viewDidUnload];

// Release any retained subviews of the main view.

// e.g. self.myOutlet = nil;

}

- (void)dealloc {

[super dealloc];

}

@end

This is a lot of code so we will review it method by method. Here we go, method by method:

1. userWillLogin. This method makes a call to the web fetch method that sends the submitted data to the server and runs a check (checkUserLogin). If authentication was successful then we store the user data in NSUserDefaults so that the user doesn’t have to log in again. Otherwise we display an alert view with the server error response.
2. checkUserLogin. This method actually makes the web fetch and returns the server response.
3. userWillRegister. This is the method that calls to create a new user when the user taps on the register button. It calls the addUser:withPass:withAddress:.
4. addUser:withPass:withAddress:. This method actually makes the web post to add that user to our web database.
5. userWillDismiss is pretty self explanatory ☺
6. uploadPhoto is in charge of calling the UIImagePicker so that the user can select his or her image to upload.
7. imagePickerController:didFinishPickingMediaWithInfo. This method returns the image selected by the user and passes it to uploadImage. However if you notice, for simplicity sake we are not using the picked image, instead we have hardcoded a placeholder image just to make things move along more smoothly.
8. uploadImage:filename. This is the method that actually makes the web post to upload the image to the server.

Now we need the backend to respond to these 2 requests. Specifically we need 2 php files (checklogin.php & user\_add\_save.php) and their helper files. Here is checklogin.php:

<?php

include"login\_config.php";

//Connection String Variables\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// connect to the server

mysql\_connect( $db\_host, $db\_username, $db\_password )

or die( "Error! Could not connect to database: " . mysql\_error() );

// select the database

mysql\_select\_db( $db )

or die( "Error! Could not select the database: " . mysql\_error() );

//IBM suggested scrub for URL request

$urlun = strip\_tags(substr($\_REQUEST['username'],0,32));

$urlpw = strip\_tags(substr($\_REQUEST['password'],0,32));

$cleanpw = md5($urlpw);

//echo"Cleanpw: $cleanpw<br>";

//$sql="SELECT \* FROM agents WHERE username='$urlun' and password='$urlpw'";

$sql="SELECT \* FROM users WHERE username='$urlun' and password='$cleanpw'";

$result=mysql\_query($sql);

// Mysql\_num\_row is counting table rows

$count=mysql\_num\_rows($result);

// If result matches $myusername and $mypassword, table row must be 1 row

//echo"Count:$count<br>";

if($count==1){

// Register $myusername and redirect to file designated success file

$cookie\_name ="$cookiename";

$cookie\_value ="$urlun";

//set to 24 hours

$cookie\_expire ="86400";

setcookie($cookie\_name,$cookie\_value,time() + (86400),"/", $cookie\_domain);

header("location:$successful\_login\_url");

}else{

header("location:$failed\_login");

}

?>

and user\_add\_save.php:

<?

include"master\_inc.php";

//--------------------------------------------------------------------------RECEIVE LOCAL VARIABLES FROM FORM

$lastname = strip\_tags(substr($\_POST['lastname'],0,32));

$firstname = strip\_tags(substr($\_POST['firstname'],0,32));

$phone = strip\_tags(substr($\_POST['phone'],0,32));

$password\_hint=$\_REQUEST['password\_hint'];

$noERROR=1;

$udidposted = 9;

//---------------------------------------------------------------------------CHECK IF USERNAME IS LONG ENOUGH

$username = strip\_tags(substr($\_POST['username'],0,32));

if(trim($username)!=='' && strlen(trim($username)) >= 4){

//---------------------------------------------------------------------------IF LONG ENOUGH THEN RUN A QUERY GETTING ALL DATA FROM THAT USER

$sql="SELECT \* FROM users WHERE username='$username'";

$result=mysql\_query($sql);

$count=mysql\_num\_rows($result);

//---------------------------------------------------------------------------IF $sql GOTTEN HAS ROW COUNT > 1 THEN USER ALREADY EXISTS----------------SET EXISTING USER 104 FLAG

if($count>0){

$username\_already\_in\_use = 3141;

}

//---------------------------------------------------------------------------ELSE USERNAME IS TOO SHORT?!?!-------------------------------------------------------SET USER TOO SHORT 104 FLAG

}else{

$username\_too\_short = 3142;}

//---------------------------------------------------------------------------EMAIL FORMAT CHECK

$email\_raw = $\_REQUEST['email'];

if(eregi("^[\_a-z0-9-]+(\.[\_a-z0-9-]+)\*@([a-z0-9-]{2,3})+(\.[a-z0-9-]+)\*(\.[a-z]{2,3})$", $email\_raw))

{

$email = $email\_raw;

}else{

//---------------------------------------------------------------------------IF INVALID EMAIL THEN----------------------------------------------------------------------SET INVALID EMAIL 104 FLAG

$bad\_email=3143;

}

//email unique?

$sql="SELECT \* FROM users WHERE email='$email'";

$result=mysql\_query($sql);

$count=mysql\_num\_rows($result);

if($count>0){

//---------------------------------------------------------------------------IF SQL FOR EMAIL RETURNS A ROW THEN------------------------------------------------SET EMAIL 104 FLAG

$email\_already\_in\_use=3144;

}

//Secure Password Format Checks

$pw\_clean = strip\_tags(substr($\_POST['password'],0,32));

if (preg\_match("/^.\*(?=.{4,})(?=.\*[0-9])(?=.\*[a-z])(?=.\*[A-Z]).\*$/", $pw\_clean, $matches)) {

}else{

//---------------------------------------------------------------------------IF PW NOT IN FORMAT THEN-----------------------------------------------------------------SET PW 104 FLAG

$pw\_insecure = 3145;

}

//---------------------------------------------------------------------------IF ERROR FLAGS ARE SET THEN LOG HEADERS----------------------------

if($username\_already\_in\_use==3141 OR $email\_already\_in\_use==3144 OR $pw\_insecure==3145 OR $bad\_email==3143 OR $username\_too\_short==3142){

header(

"location:user\_add\_errors.php?pw\_insecure=$pw\_insecure&email\_already\_in\_use=$email\_already\_in\_use&username\_already\_in\_use=$username\_already\_in\_use&bad\_email=$bad\_email&username\_too\_short=$username\_too\_short");

die();

}

else {header("location:user\_add\_errors.php?noERROR=$noERROR");}

//End Error Checks\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//-------------------------------------------------------------------INSERT INTO SQL

//Encrypt Password

$encrypted\_pw = md5($pw\_clean);

$query = "INSERT INTO `users` (`username`,

`password`,

`lastname`,

`firstname`,

`email`,

`phone`,

`password\_hint`,

`udid`,

`userCreated`)

VALUES

(

'$username',

'$encrypted\_pw',

'$lastname',

'$firstname',

'$email',

'$phone',

'$password\_hint',

'$udidposted',

now())";

// save the info to the database

$results = mysql\_query( $query );

// print out the results

if( $results )

{

if($username\_too\_short==3142){echo"ShortUser=".$username\_too\_short;}

if($username\_already\_in\_use==3141){echo"UserTaken=".$username\_already\_in\_use;}

if($email\_already\_in\_use==3144){echo"EmailTaken=".$email\_already\_in\_use;}

if($pw\_insecure==3145){echo"ShortPass=".$pw\_insecure;}

if($bad\_email==3143){echo"BadEmail".$bad\_email;}

//echo( "<font size='2' face='Verdana, Arial, Helvetica, sans-serif'>Your changes have been made sucessfully. <br><br><a href='login.php'>Back to login</a></font> " );

}

else

{

die( "Trouble saving information to the database: " . mysql\_error() );

}

//--------------------------------------

$sql="SELECT \* FROM users";

$result=mysql\_query($sql);

$count=mysql\_num\_rows($result);

if($count==1){

$query = "UPDATE `users` SET `permissions`='5' WHERE `email`='$email'";

//---------------------------------------SAVE the info to the database

$results = mysql\_query( $query );

//---------------------------------------JUST PRINT CODE

if( $results )

{ echo( "ADMINCREATED" );

}

else

{

die( "ERRORSAVINGADMIN" . mysql\_error() );

}

}

?>

We won’t review all of what these files do. You can download them here\_\_\_\_\_! But checklogin basically queries the database for the user and password submitted. Whereas user\_add\_save.php verifies the user and password submitted for registration, as well as the email, and then returns a result accordingly.

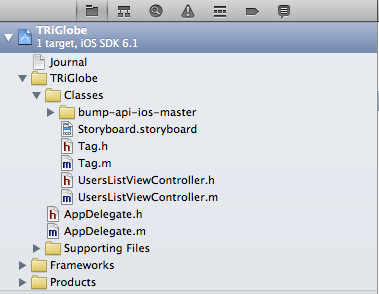
You must configure your database, user and password accordingly. This is done in the login\_config.php file. We wont go too much into the php files but another important one is the user\_add\_errors.php. This file takes care of any errors during registration such as too short a username or too simple a password or invalid email or user already exists.

asd

# Bump

Download the Bump for iOS v3 and import it into your XCode Project. Also add CoreLocation.framework, CFNetwork and AudioToolbox.framework. (NOTE: If you get a compiler error saying you have duplicate symbols its probably because you imported the Bump test project as well. That project itself contains a main.m file, which XCode attempts to process along with your TRiGlobe project’s main.m file. So just comment everything out in that other main.m file or simply remove the project. But leave the other files in there.

Your project navigator should look like this:



Don’t worry about the Tag Class file for now. I just happened to have it in there but we will create it later.

Ok so we have our user’s list in a tableview. Users must be able to post data to the tag table. This is where the Tag Class comes in. Create a new NSObject subclassed file and name it Tag. Now replace its code with the following:

#import <Foundation/Foundation.h>

@interface Tag : NSObject {

NSNumber \*rglatitude;

NSNumber \*rglongitude;

NSString \*originUdid;

NSString \*destintyUdid;

NSString \*rgcountry;

NSString \*sender;

NSString \*receiver;

}

@property(nonatomic,copy)NSString \*destintyUdid;

@property(nonatomic,copy)NSNumber \*rglatitude;

@property(nonatomic,copy)NSNumber \*rglongitude;

@property(nonatomic,copy)NSString \*originUdid;

@property(nonatomic,copy)NSString \*rgcountry;

@property(nonatomic,copy)NSString \*sender;

@property(nonatomic,copy)NSString \*receiver;

-(id)initWithOriginUdid:(NSString\*)oudid

rglatitude:(NSNumber\*)lati

rglongitude:(NSNumber\*)longi;

-(id)initWithSender:(NSString\*)senderi

receiver:(NSString\*)receiveri

rglatitude:(NSNumber\*)lati

rglongitude:(NSNumber\*)longi

rgcountry:(NSString\*)rgcountri;

-(id)initWithOriginUdid:(NSString\*)oudid

destintyUdid:(NSString\*)dudid

rglatitude:(NSNumber\*)lati

rglongitude:(NSNumber\*)longi

rgcountry:(NSString\*)rgcountri;

@end

#import "Tag.h"

@implementation Tag

-(id)initWithOriginUdid:(NSString\*)oudid

rglatitude:(NSNumber\*)lati

rglongitude:(NSNumber\*)longi

{

NSLog(@"TAG INIT");

if ( (self = [super init]) == nil )

return nil;

self.rglatitude = lati;

self.rglongitude = longi;

self.originUdid = oudid;

return self;

}

-(id)initWithSender:(NSString\*)senderi

receiver:(NSString\*)receiveri

rglatitude:(NSNumber\*)lati

rglongitude:(NSNumber\*)longi

rgcountry:(NSString\*)rgcountri

{

NSLog(@"TAG INIT OF TYPE SENDER RECEIVER");

if ( (self = [super init]) == nil )

return nil;

self.receiver = receiveri;

self.rglatitude = lati;

self.rglongitude = longi;

self.sender = senderi;

self.rgcountry = rgcountri;

return self;

}

-(id)initWithOriginUdid:(NSString\*)oudid

destintyUdid:(NSString\*)dudid

rglatitude:(NSNumber\*)lati

rglongitude:(NSNumber\*)longi

rgcountry:(NSString\*)rgcountri

{

NSLog(@"TAG INIT");

if ( (self = [super init]) == nil )

return nil;

self.destintyUdid = dudid;

self.rglatitude = lati;

self.rglongitude = longi;

self.originUdid = oudid;

self.rgcountry = rgcountri;

return self;

}

@end

Ok so we have a class that creates Tag objects and we have given it different initializers. So every time a user wants to create a tag they will have to use one of these methods. Users will create tags by bumping phones. This means each time users bump their phones they will be able to exchange tags. That is where the Bump API comes in.

There is ample documentation you can read through on their site, but basically you must activate the Bump Awareness by tapping a button. This will present a dialog telling the user to bump their phones. Only then will a bump work. Otherwise your phone could be trying to exchange tags with keys or other things in your pockets ☺.

Go ahead and open up the bump-api-ios-master folder group and navigate to the BumpAppDelegate.m. Find the configureBump method, which has pretty much everything we will need. You need to register for a Bump API key so go ahead and get that now: <http://bu.mp/company/apiagree>.

Now let’s:

Make sure libBump.a library to your project

Add BumpClient.h to your UsersListViewController.h

Add NSUserDefaults to store your user

Configure the bump method. Let’s review the provided example:

- (void) configureBump {

*// userID is a string that you could use as the user's name, or an ID that is semantic within your environment.* ***Here we will add the storedUser***

[BumpClient configureWithAPIKey**:**@"**your\_api\_key**" andUserID**:**[[UIDevice currentDevice] name]];

[[BumpClient sharedClient] setMatchBlock**:^**(BumpChannelID channel) {

NSLog(@"Matched with user: %@", [[BumpClient sharedClient] userIDForChannel**:**channel]);

[[BumpClient sharedClient] confirmMatch**:**YES onChannel**:**channel];

}];

[[BumpClient sharedClient] setChannelConfirmedBlock**:^**(BumpChannelID channel) {

NSLog(@"Channel with %@ confirmed.", [[BumpClient sharedClient] userIDForChannel**:**channel]);

[[BumpClient sharedClient] sendData**:**[[NSString stringWithFormat**:storedUser**] dataUsingEncoding**:**NSUTF8StringEncoding]

toChannel:channel];

}];

[[BumpClient sharedClient] setDataReceivedBlock**:^**(BumpChannelID channel, NSData **\***data) {

**NSLog(@"Data received from %@: %@",**

[[BumpClient sharedClient] userIDForChannel**:**channel],

[NSString stringWithCString**:**[data bytes] encoding**:**NSUTF8StringEncoding]);

}];

*// optional callback*

[[BumpClient sharedClient] setConnectionStateChangedBlock:^(BOOL connected) {

if (connected) {

NSLog(@"Bump connected...");

} else {

NSLog(@"Bump disconnected...");

}

}];

*// optional callback*

[[BumpClient sharedClient] setBumpEventBlock**:^**(bump\_event event) {

switch(event) {

case BUMP\_EVENT\_BUMP:

NSLog(@"Bump detected.");

break;

case BUMP\_EVENT\_NO\_MATCH:

NSLog(@"No match.");

break;

}

}];

}

The 4 red bold sections above correspond to, in order of appearance; (1) we will pass the storedUser on the application which we must first save upon launching the app, (2) the api key you got from Bump, (3) the data sent will be the actual storedUser and (4) we will log the results just to make sure it worked.

Copy methods to our MapViewController from testapp and import bumpclient.h & Connect the bump button to the configure bump method and add call to configureBump in startBumpButtonPress to test the bump…