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

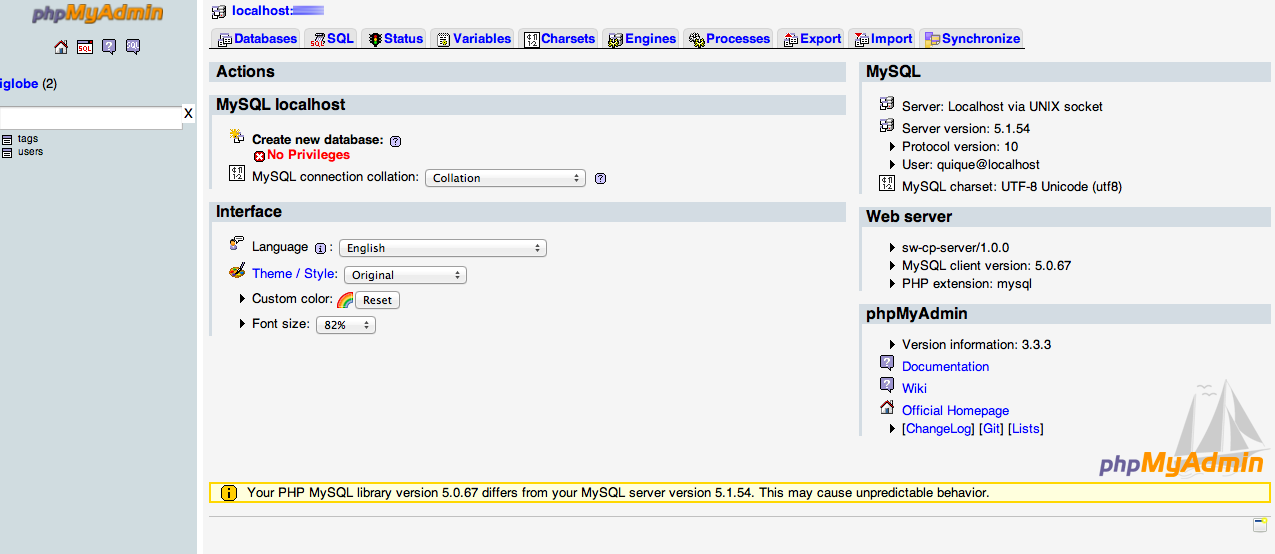
Sounds simple enough. Web services are usually large databases of information, so our database will need a table to store all this information we mentioned above. Databases come in a lot of flavors and the way we interact with them; Create, Read, Update and Delete (CRUD) data from them is just as varied. 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

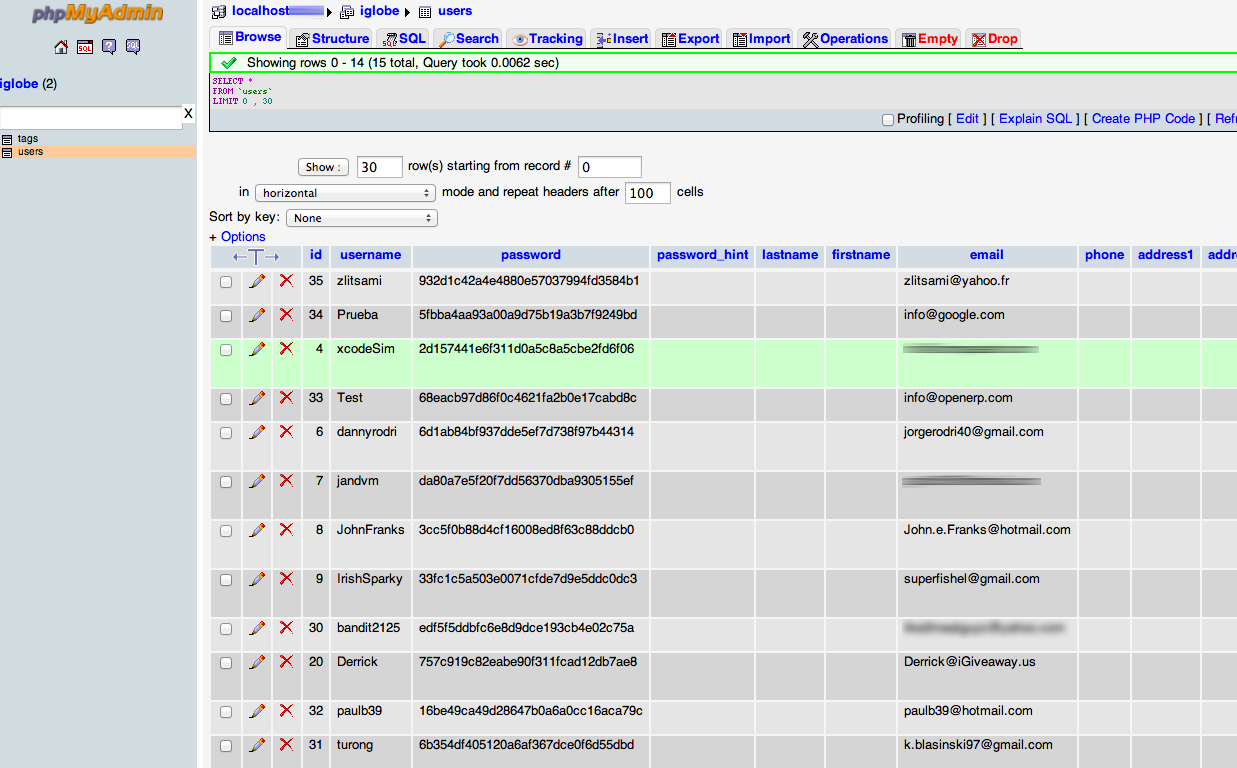
# Databases

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.

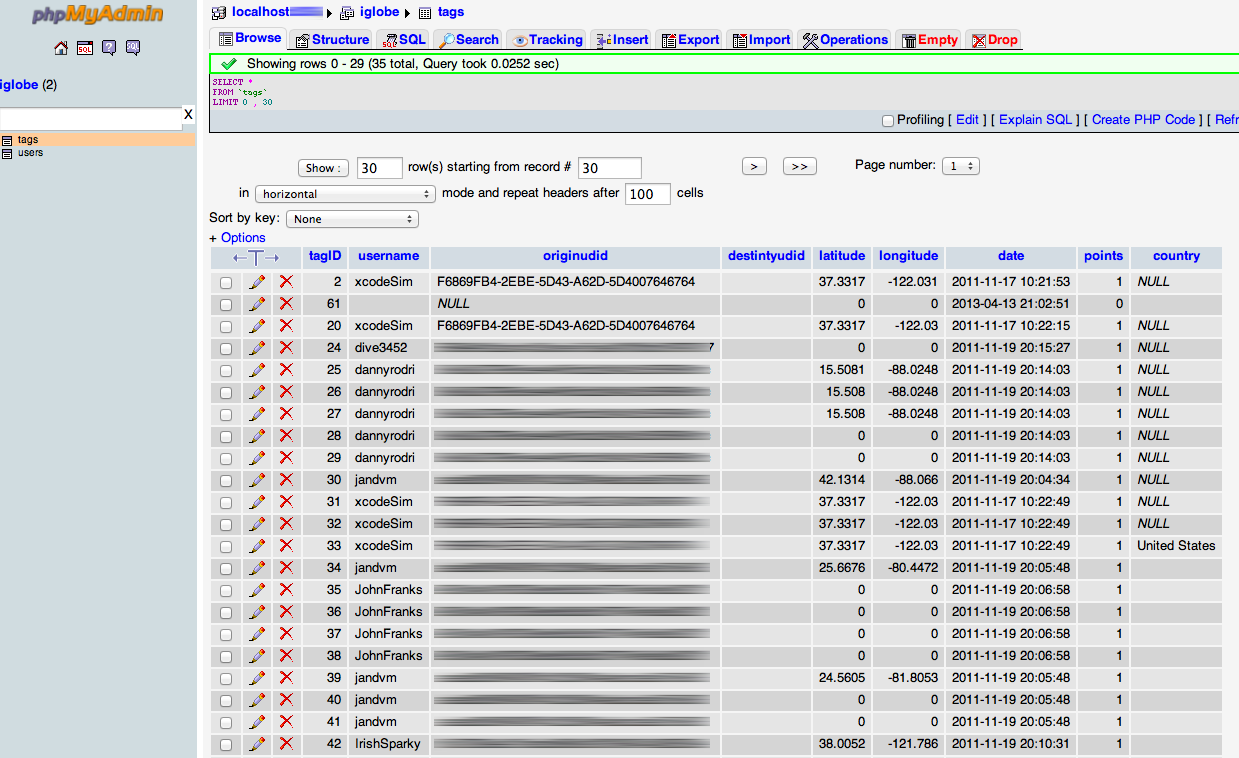
Ok so we will be using a database that is usually available for free in most web hosting services. My web hosting service has phpMySQL, which comes included with a free package. If you want other databases such as MySQL you would need a paid service. This is what my database management interface looks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

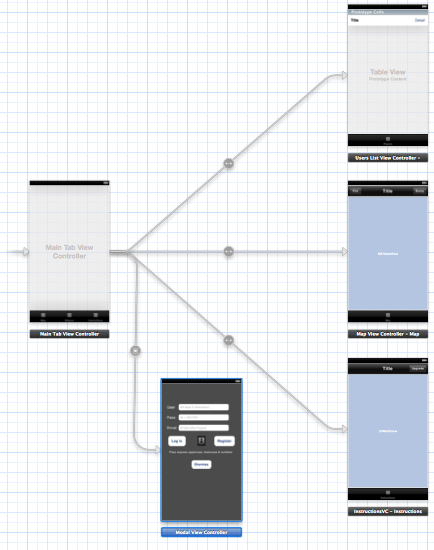
If you Build & Run now you should get your users displayed in the tableview. Cool! That’s what we are going to want to do, display a list of users in a tableview and then add in the points, like a score table.

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:

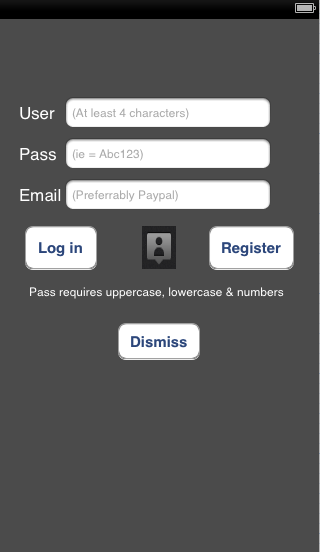
1. Present a login view controller
2. Save user and pass information
3. Fetch user data from the web service
4. Plot points on a map
5. Display instructions in a view

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…

Its time to take a break. While we are on the break however, let’s take a few minutes to visualize what the app layout will look like now that we have a better idea of where we are headed. Compare these results to the initial sketch you made of your app.

See you in Part 2!