# Installation and Setup

1. Download and install PhoneGap (<http://phonegap.com/getstarted>/)
2. Follow their instructions to setup the default PhoneGap App on your desktop
3. Download the PhoneGap Developer Android/iOS app onto your mobile device, ensure it’s on the same local network as your development machine, and type in the IP address and Port and connect your device to PhoneGap
4. At this point you should have the default PhoneGap application on your phone showing it’s connected to the device (your desktop). A quick check can be done by making a text change to the index.html file of the App and seeing the change take place on your phone. If this isn’t the case, go back and check PhoneGap instructions again
5. Get a Drupal 7.x installation up and running
6. Install the following Drupal modules. [OAuth2 Server](https://www.drupal.org/project/oauth2_server) (Including the related PHP Library), [Libraries](https://www.drupal.org/project/libraries), [Entity API](https://www.drupal.org/project/entity), [Entity Reference](https://www.drupal.org/project/entityreference), [X Autoload](https://www.drupal.org/project/xautoload), [Services](https://www.drupal.org/project/services)
7. After enabling all of those, also enable the REST Server module
8. Open the Drupal Permissions and under OAuth2 Server, enable the permission of ‘Use OAuth2 Server’ for both Anonymous and Authenticated user
9. Add an OAuth2 Server (/admin/structure/oauth2-servers)
   1. Add a label
   2. Ensure that only ‘User credentials’ is checked
   3. On the Scopes tab, add a scope, nothing special here
   4. Click save
   5. On the Clients tab, add a client. Untick the ‘Require a client secret’ option, and include the external domain name of the website in the Redirect URI field
   6. Untick the other options
   7. Click save
10. Add a Service (/admin/structure/services)
    1. Provide a simple machine readable name
    2. In the Server field, select ‘REST’
    3. The Path to endpoint should be the URL you will request, so make sure it’s not the same as any possible content from the site, something like ‘api’ or ‘service’
    4. Select OAuth2 authentication
    5. Click save
    6. On the Server tab under Response formatters select JSON and JSONP
    7. Under Request parsing, select application/json, application/x-www-form-urlencoded and multipart/form-data
    8. Click save
    9. On the Authentication tab, select the OAuth server you created
    10. Click save
    11. On the Resources tab, select any option you want to expose as a service
    12. For any item you select, be sure to tick Require authentication
    13. Save
11. Now if you open your website with the Path to endpoint at the end, you will see a success message. For example, for a services endpoint called api, open: <http://example.com/api>
12. This will confirm that the service has been setup
13. In this example, I have enabled retrieve under the Node section of the Resources tab.  
    You can also test to ensure the Authentication by adding /api/node/(nid) to the domain, for example, [http://example.com/api/node/7](http://example.com/api/node/73) and it will return a set of square brackets
14. The URL is generated in the following format:  
    http://example.com/(Service End Point)/(Service Resource Name)/(Machine ID)  
    For example, to get a Node: <http://example.com/api/node/5>  
    To get a User: <http://example.com/api/user/4>
15. Now that the Drupal side of things is setup, we need to setup the PhoneGap Javascript code for it to interact with.

# Code Setup

1. We need to make an update to the Services module, in order to allow for Cross Domain requests.
2. Edit services.module in the Services module, and include the following on line 13:

drupal\_add\_http\_header('Access-Control-Allow-Origin', "\*");  
drupal\_add\_http\_header('Access-Control-Allow-Headers', "accept, authorization");  
drupal\_add\_http\_header('Access-Control-Allow-Methods', 'GET,PUT,POST,DELETE');

1. This ensures that we can access the content from other sources, as well as provide our Authorisation Key to keep the requests secure.  
   The reason we put it in the Services module, is so that it only gets invoked when we run those Services pages, and not on every single page for the website
2. Before we start, be sure to download jQuery 1.5+ and include that on all HTML pages
3. Next, open up the file system for the PhoneGap application, copy the index.html and rename it login.html
4. Open the new file, delete any unrelated Body fields. Add two text input fields, one for the Username and one for the Password, and a Submit button
5. For the Submit button, add in an onclick attribute which should contain the function name that we will use in Javascript to trigger the Login script.  
   For example, <button onclick="loginDrupal();">Login</button>
6. So, when that button is clicked, it will invoke the Javascript function loginDrupal(), so we need to create that below inside a <script> tag.  
   And first up, get the values from the input fields and save them as variables.

function loginDrupal(){  
 var user = $('#user-input').attr('value');  
 var pass = $('#pass-input').attr('value');  
};

1. Inside that function we need to include a set string to pass the Authorisation system.

var data\_string = 'grant\_type=password&client\_id=app';  
data\_string += '&username=' + encodeURIComponent(user);  
data\_string += '&password=' + encodeURIComponent(pass);

1. The client\_id variable needs to match the machine name of the Client listed in your OAuth2 configuration
2. And finally, the actual Ajax request which is the bulk of the functionality:

$.ajax({  
 url: "http://www.example.com/oauth2/token",  
 type: 'post',  
 data: data\_string,  
 dataType: 'json',  
 error: function(XMLHttpRequest, textStatus, errorThrown) {  
 alert($.parseJSON(XMLHttpRequest.responseText).error\_description);  
 },  
 success: function (data) {  
 localStorage.atoken = data.access\_token;  
 window.location = "index.html";  
 }  
});

1. Here is a breakdown of the above fields:
   1. **url** is fairly obvious, this ends with the location of the OAuth2 service
   2. **type** is the type of HTML request we are making to the server
   3. **data** is the additional data we are appending to the URL, in this case it’s the string that we created in Step 9
   4. **dataType** defines we are speaking JSON
   5. **error** and **success** are functions that contain the actions taken in those events
2. Error function
   1. The three variables in the function all help provide feedback based on which error happens
   2. Currently there is an Alert for a Failed login. By using the parseJSON function and passing in the return message we can display the actual error message to the user
3. Success function
   1. The data variable returned includes the Access Token, and some other useful variables that are returned when you login
   2. In order to make future requests, we need to re-use the Access Token, which is why it’s saved in the localStorage area
   3. And finally, we redirect to the index.html file
4. At this point, clicking the Login button will run the Ajax function, which sends a request off to the Drupal website, and the Access Token is returned to the Script. Once that is finished, we redirect the user to the index.html page where we provide most of the functionality
5. On the index.html page create a new input Button. As before, provide an onClick attribute and provide a new function to reference to.
6. This new function button will run a new request that will query a Node and return its fields.

$.ajax({  
 url: 'http://example.com/api/node/73',  
 method: 'GET',  
 crossDomain: true,  
 cache: true,  
 jsonp: true,  
 beforeSend: function (xhr) {  
 xhr.setRequestHeader ("Authorization", "Bearer " + localStorage.atoken);  
 },  
 success: function (returnData) {  
 returnData.title  
 },  
 error: function(){  
 $('#pageContent').html('fail');  
 }  
});

1. Here is a breakdown of the various options:
   1. **url** is the request URL as setup in the Services module back on line 14 of the first page
   2. **method**, just the same as on the previous request, but here we are using GET rather than POSTing
   3. **crossDomain** is important, as we are requesting from elsewhere of the domain
   4. Without **cache** set, the time in seconds would be appended to the request URL
   5. **jsonp** is a variant of JSON that is allowed to work Cross Domains
   6. **beforeSend** is a special function that adds an Authorisation HTTP Header that includes the stored ‘Access Token’ that was sent back by the Login script
   7. **success** is what is run on a successful login, noting you have access to the returnData variable. And this is a JSON object of the Node that I’m requesting. So for example, using returnData.title is the Title of the node. And for a field value returnData.field\_app\_url.und[0].value will get that
   8. **error** is the function that is run when the service call fails

There you go, that is the simple basics of Authorising an account with a Drupal installation, and then requesting a Node contents.

Using a combination of these calls, and the configuration options in the Services module, you should be able to read user field data by changing the URL to example.com/nodes/user/1.  
With ‘nodes’ being the name I gave to the Service, and ‘user’ being the resource value.

# How to access views

Now that everything is all setup, we can access Views results, which gives up the ability to not only access dynamic content, but also get Views to execute some PHP for us.

1. First up, install the Services Views module (<https://www.drupal.org/project/services_views>)
2. Open up the Services configuration, and open the Edit Resources tab
3. Scroll to the bottom, and enable Views and Retrieve. Also check the Requires Authentication box
4. Create a new view, and uncheck both ‘Create a page’ and ‘Create a block’
5. Once created, click the ‘+ Add’ button and add a new Service
6. Include a Path. While this isn’t used for the service call, it is mandatory
7. Include all the fields that you need, just like normal
8. If you need to add some PHP, it gets a bit tricky
   1. In the Output Code field, include the code:

<?php print\_r ($value); ?>

* 1. The majority of the code should be placed in the Value Code field. This can include code such as node\_load, or any data calculation or manipulation.
  2. In the example of getting a response code from a remote URL, I used the following:

$curling = node\_load($data->node\_field\_data\_field\_managed\_sites\_nid)->field\_app\_url['und'][0]['value'];  
$head = get\_headers($curling, 1);  
return explode(' ', $head[0])[1];

1. Now that your view is setup, we need to create an Ajax call in our Javascript to call the View
2. Below is the sample code I used, and after that I will go through each line:

function getSiteList(){  
 $.ajax({  
 url: 'http://example.com/api/views/users\_nodes?args='+localStorage.userID,  
 method: 'GET',  
 crossDomain: true,  
 cache: true,  
 jsonp: true,  
 beforeSend: function (xhr) {  
 xhr.setRequestHeader ("Authorization", "Bearer " + localStorage.atoken);  
 },  
 success: function (returnData) {  
 viewsData = $.parseJSON(returnData);  
 for (i in viewsData) {  
 tempVal = viewsData[i].site;  
 }  
 },  
 error: function(){  
 $('#pageContent').html('Failed to find users sites from View, try again.');  
 }  
 });  
};

* 1. The URL field is very much the same as other requests, except the resource is ‘views’ rather than Nodes or System
  2. The variable after that is the machine name of the View. NOT the Services Path, but the machine name. This can be found at the end of the URL when you’re editing the view.
  3. And finally, you can include ‘args’, ‘filter’, ‘limit’, etc…
  4. All the other sections are very similar to the previous requests
  5. Inside the Success function we run parseJSON to convert the string returned into actual JSON, and then we can use a for loop to iterate through each of the returned Rows from the View

# And here is an embedded version of the Index and Login files

Currently in my very much in development Panoptic App.

