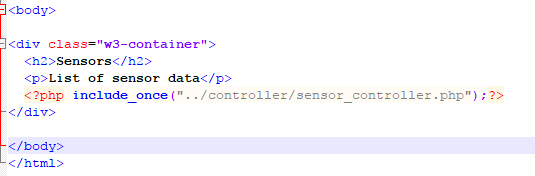
You should now have a Web API that produces data for many query types, e.g. data for a sensor for a date range.

If you have not achieved this so far, you will need to get that working first before continuing.

Ensure that the index.php file contains the following in the <body> section:

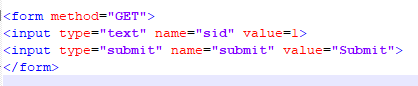


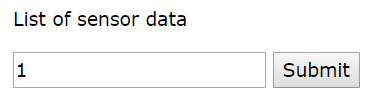
To add a form to your web page, e.g. a simple text box, you need to consider:

Field name:

Method – GET or POST (compare GET and POST form data)

For now, start with the following simple form in a file. Name the file form\_view.php and store it in the view directory:



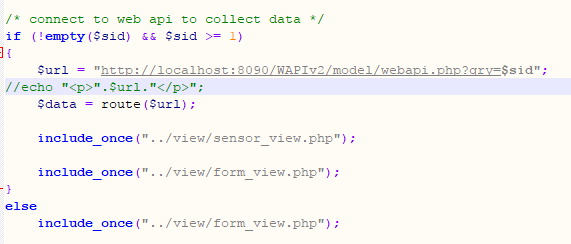


The form uses the URL (a GET request) to send the form data to the WebAPI via the controller.



The form is included by the sensor\_controller shown below.

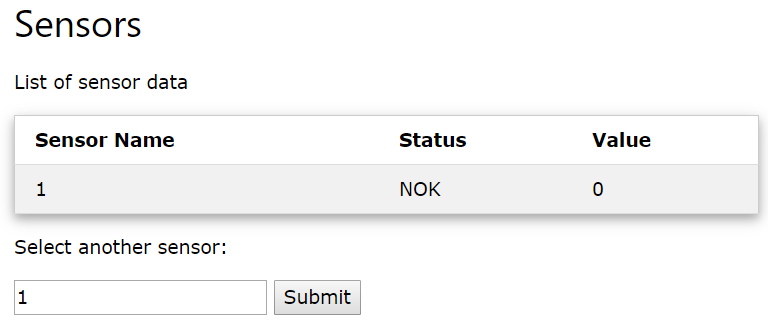
The function route($url) has been added to this example, to simulate a standard MVC framework, which routes requests to specific URLs. It is also wise to break up code into manageable, reusable code.



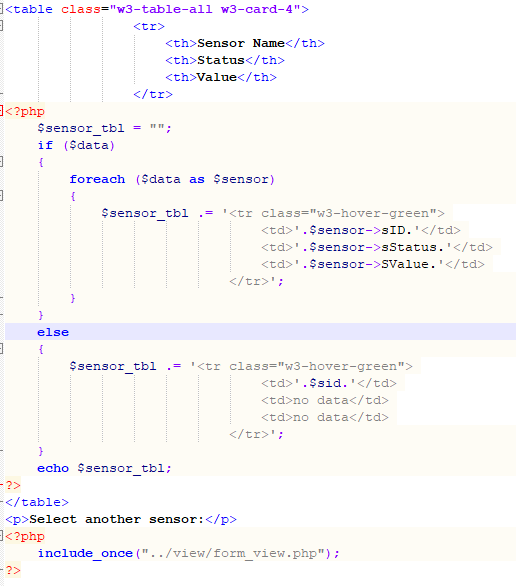
If the data obtained via the URL is present, e.g. if ($sid is not empty and contains a value greater than or equal to 1) then curl is used to connect to the Web API with the sensor id data sent as part of the query.

*Note: To this version, a new function, route(), has been created to connect to the Web API (passing the URL as a parameter).*

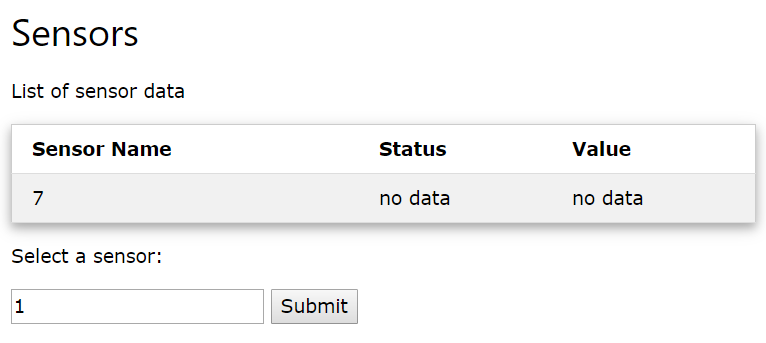
The controller includes the sensor\_view.php and form\_view.php or just form\_view.php depending upon whether the form has been submitted or not, e.g. $sid is empty. This is what you will see when you run the main index.php file:



Note: In this version, sensor\_view.php includes a table irrespective of whether data is returned or not:



A foreach loop in sensor\_view.php is used to create a table containing the returned data for the sensor. If no data was returned, then a simple table row is displayed indicating this.

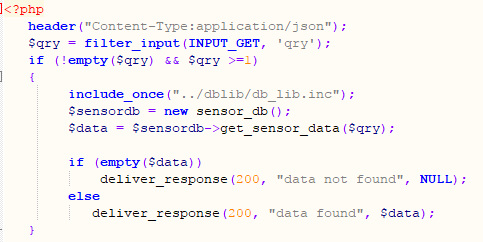


The form\_view is also redisplayed.

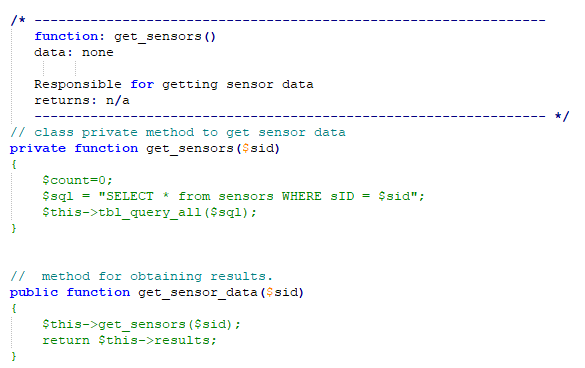
The controller sends the form data to the Web API via the URL:



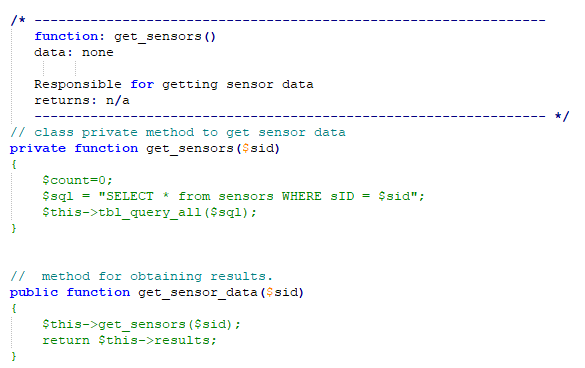
The Web API needs to pick up the data from the URL, using filter\_input(…), see below:



The data is stored in variable $qry, which is passed as a parameter to the get\_sensor\_data(…) function in db\_lib.inc. An extract of file db\_lib.inc is shown below:

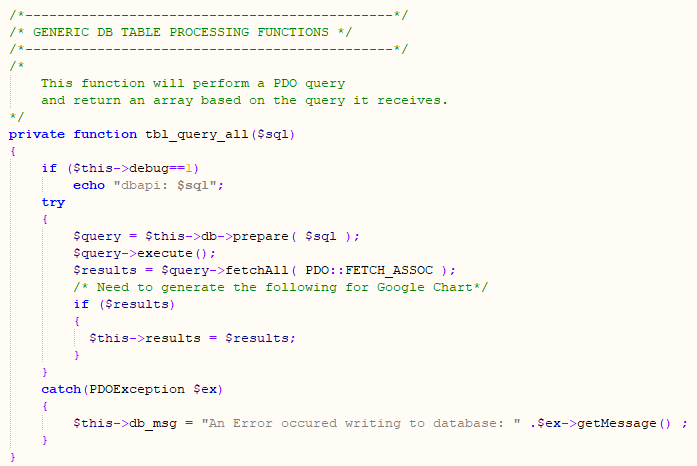


The private function (method) allows access to internal private class data. This will perform the database query based on the data send from the Web API:



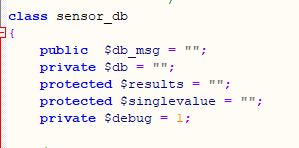
The above public method is called by the Web API. This is accessible from the Web API as it is a public method. It calls the private method get\_sensors which populates the class variable results. The public function the returns the class result data (array of sensor values for the specific sensor).

Remember, some generic functions were created to perform queries. They have now been made private – you should not need to alter the generic functions as they can run most queries, as long as you pass the SQL statement to them ($sql):

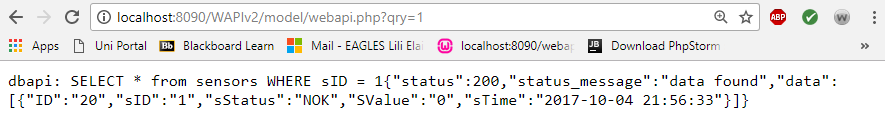


## Troubleshooting

If your SQL statement is not working properly, set the $debug variable at the top of the class to 1. Then run the Web API directly from the URL:



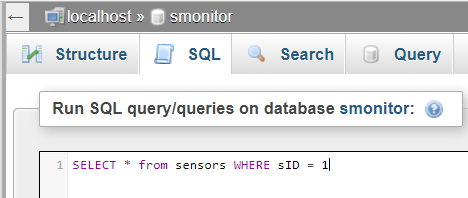
You will see that when testing the Web API with $debug set to 1, your queries will be shown:



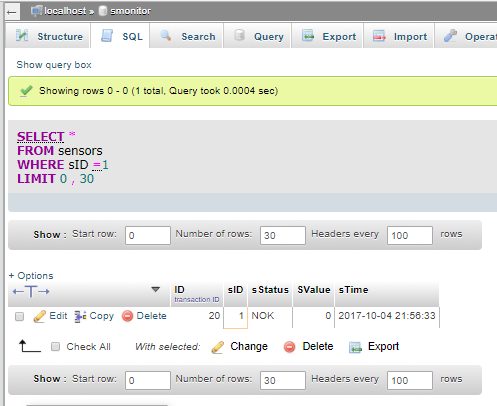
If there happened to be a problem with the data, e.g. you expected there to be data returned from the Web API and you didn’t receive it, start by testing the Web API with $debug set to 1.

Things to check if your site is not working as expected:

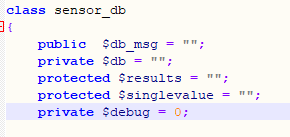
1. With $debug is set to 1, test the Web API to ensure that data is returned as expected
2. If the data is not accurate, copy the SQL from the web page and try it in PhpMyAdmin:



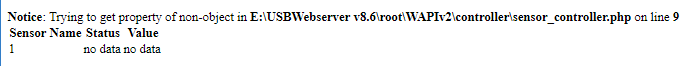
With a successful outcome such as that below, check the controller variables:



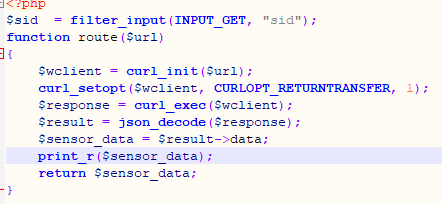
1. If the Web API appears to be correct and the SQL and data are as expected, ensure $debug is set to 0 in dbli.inc, then check the controller:



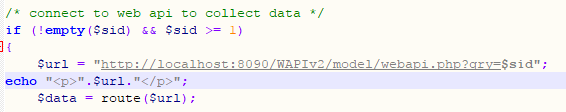
If you leave $debug as 1, you will see the following when you run the controller:



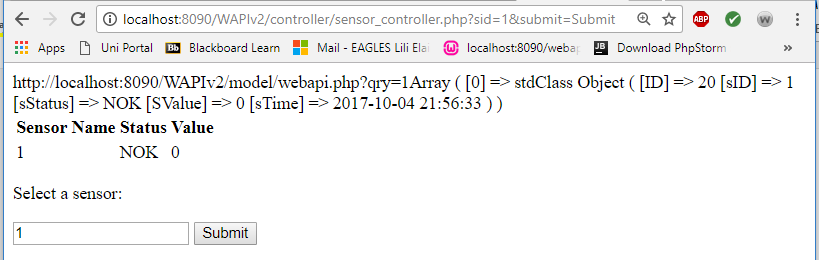
Uncomment the print\_r statement:



And uncomment or add the echo statement to show the URL:



You should see the following when you run the controller and submit the form:



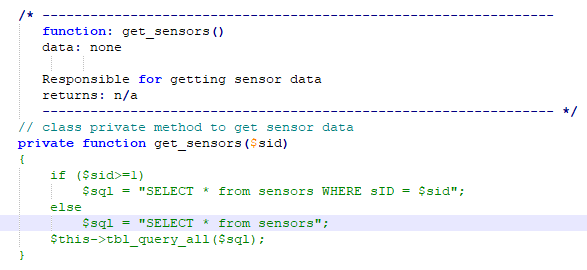
This shows the URL connected to and the data that was returned by the WebAPI.

Things to check:

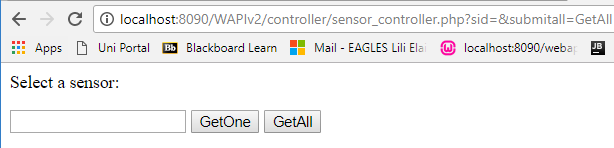
1. copy and paste the URL shown into the browser’s address bar and test.
2. Check the data corresponds with the database data.
3. Check that the ports are correct, e.g. 8090 is set on the web server:
4. Check that the directory is correct (i.e. WAPIv2/model…)

Querying all sensors:

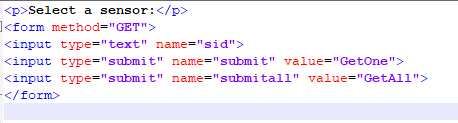
To be able to list all sensors you need to ensure that the dblib can perform a query that does not include a WHERE, e.g.:



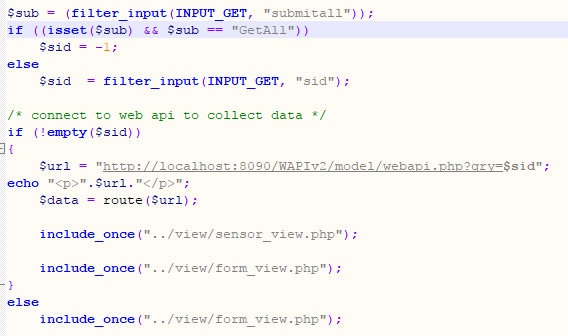
This means that you need a value to be passed in that is not greater than or equal to 1. Zero could cause problems due to null values and therefore, -1 could be used. The -1 could be sent when a certain button is pressed at the view. See amended form below:



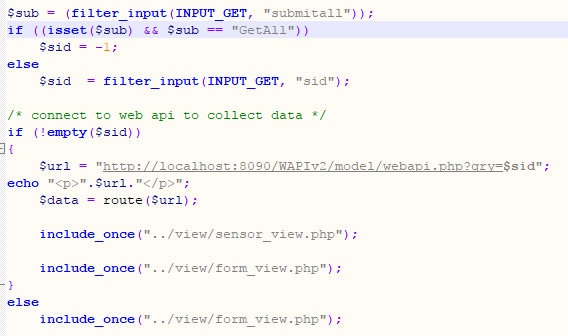
As HTML in form\_view.php:



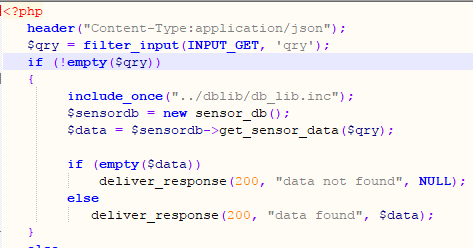
When GetAll is pressed, the text box data is checked at the controller, otherwise it is ignored. If the submit All button was pressed $sub is populated by filter\_input(…), otherwise it is empty. If it is set, then the $sid value is set to -1, otherwise it is set to the value passed in via the form’s text box.



Note that the check against $sid has been changed to remove the && $sid >= 1 condition.

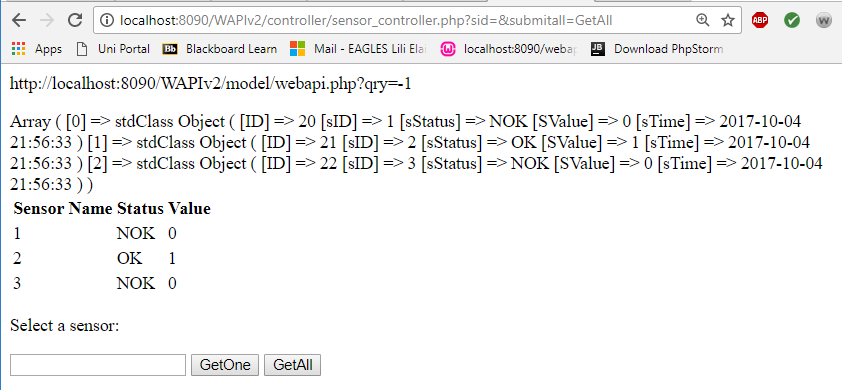


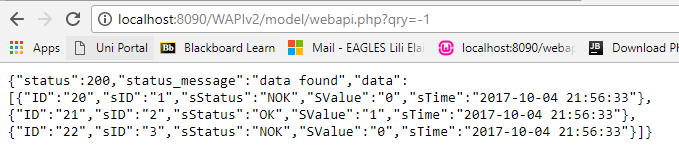
This has also been changed in the Web API:



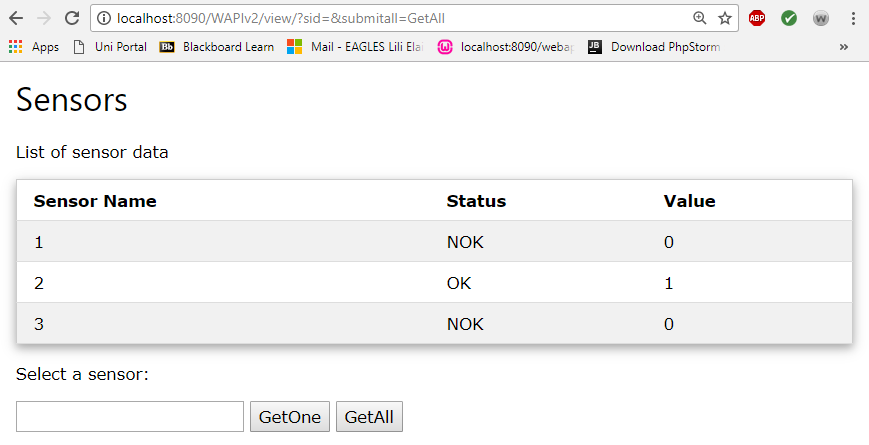
This allows the -1 to be processed to mean all sensors.

Using -1 might not be a great idea of course, as -1 is normally used in programming to indicate that there is an error. But, this is what you should see with those changes, when you run the controller directly:





Putting it all together and running the code via the index.php (with debugging and echo messages commented out), you will see:



Now experiment with different HTML Form elements, such as drop down boxes. See:

<https://www.w3schools.com/w3css/w3css_input.asp>