

# Report – Critical Analysis

## Introduction

This assignment is to create a cloud web service that is available through a variety of technologies. This web service is formed of two projects, the back-end which contains the API routes and methods as well as the communication with the database, and the front-end which job is to make calls to the back-end project and display the data retrieved as the user requests.

## Content

The author has decided to implement the project using the PHP language instead of Java, since given his proficiency in building Java application, he wanted to challenge himself and learn something new using the programming knowledge and skills gained in previous years. The syntax used by PHP is very similar to the one adopted by Java, therefore this made the learning experience easier and more enjoyable.

Regarding the cloud based requirement, a multitude of providers offer the possibility to host a web service on their facilities, including Google, Amazon, Enomaly, AT&T and so on. Google has been chosen, since students have had the opportunity to try this platform in the laboratories at the university and therefore are more familiar with it.

Google offers a vast selection of services, however only Google App Engine (or GAE) and Google SQL were intended for this project, providing a place to store and run the web service and its database.

For the implementation of the application, GAE offers a building tool for PHP developers that provides a local server to run and test the code and functionalities to deploy the project to the cloud with just one click.

Once the project has been set up, it is time to decide what libraries and frameworks are going to be used for the visual content and API functionalities.

After researching, the author has decided to use Silex micro-framework to make the data accessible in a RESTful manner, since it is light, testable, provides all the necessary features for the creation of routes, handles requests and responses object, encourages the proper use of the HTTP methods and is compatible with the GAE.

REST has been chosen instead of SOAP because it is easier to learn, light-weight, concise and uses exclusively the HTTP protocol, where SOAP is more complex, extensible, heavier and mostly adopted for projects that have some security prerequisites and implement the use of various protocols (not just HTTP) (Spies, B. 2008).

Other tools this project is taking advantage of are the JQuery library for the AJAX interactions and event handling, and Bootstrap library which is the most used Javascript framework for designing web pages and organizing elements to create a clear and friendly user experience, compatible on different devices including smartphones, tablets and desktop PCs (JQuery, No Date).

However, JQuery itself provides a similar library to Bootstrap, which is called JQuery UI, but given the ease of using Bootstrap without the need of downloading and importing extra

archive files in your project, it simplifies things, very important factor if a project is growing in size and complexity.

The downside of making this assignment RESTful is the unfeasibility of creating a WSDL file. A WSDL is a self-describing file, typical of a SOAP web service that contains information on how the service works. This specification is not available in a RESTful service, but since this type of development is more diffused and easy to implement, the author has decided to use it and ignore SOAP for this occasion, with the aim of building a project that can be proudly shown to employers.

## API Routes

The APIs have been designed to provide the following options:

- Retrieve all the staff data stored in the database in a standardized format (JSON) or by requesting a specific one.
- Search for a member of staff by passing an ID or a forename and surname and retrieve this data in a specified format or JSON by default.
- Add a member of staff to the system.
- Remove a member of staff by passing the staff ID to the API.
- Update a staff record.

The web service routes along with the HTTP methods that achieve such purpose are listed below:

```
29 //API routes
30 $app->get('/', 'welcomePage');
31 $app->get('/staff', 'retrieveAllMembersNoFormat');
32 $app->get('/staff/format/{dataTypeRequested}', 'retrieveAllMembers');
33 $app->get('/staff/{forename}/{surname}', 'searchByNameNoFormat');
34 $app->get('/staff/formatFS/{forename}/{surname}/{format}', 'searchByName');
35 $app->get('/staff/{id}', 'searchByIdNoFormat')->assert('id', '\d+');//Accept only id with value > 0
36 $app->get('/staff/formatID/{id}/{format}', 'searchById');
37 $app->post('staff/addMember', 'addNewStaff');
38 $app->post('staff/delete', 'deleteStaffMember')->assert('id', '\d+');//Accept only id with value > 0;
39 $app->post('staff/update', 'updateStaffMember');
40
```

Figure 1 – Web Service Routes

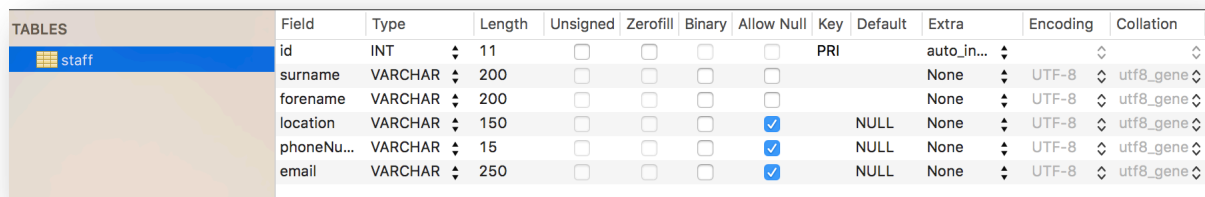
## How is the data accessed?

Each API relies on retrieving data from a Data Access Object class (DAO), which job is to establish a connection with the Google SQL table, execute queries and return the data. This data is then modified by the API which transforms it in the format requested by the user. Finally, this data contained in an object of the requested format is elaborated by the front-end methods which decides how to display the information to the end-user.

This technique of having a Model (database data and Staff class), a Viewer (front-end site) and a Controller (staff DAO and API) takes the name of MVC (Model View Control) where, each section is separated to allow scalability and group development (W3Schools, No Date). MVC is a design pattern that many professionals use to consent future developments and since the project is broken into smaller parts, is easier to understand its operations and structure.

### How is the data stored?

Staff data is kept in a relational database and its structure is shown here:



The screenshot shows a database table named 'staff' with the following structure:

Field	Type	Length	Unsigned	Zerofill	Binary	Allow Null	Key	Default	Extra	Encoding	Collation
id	INT	11	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	PRI		auto_in...		
surname	VARCHAR	200	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		None		UTF-8	utf8_gene
forename	VARCHAR	200	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		None		UTF-8	utf8_gene
location	VARCHAR	150	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>		NULL		UTF-8	utf8_gene
phoneNu...	VARCHAR	15	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>		NULL		UTF-8	utf8_gene
email	VARCHAR	250	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>		NULL		UTF-8	utf8_gene

Figure 2 – SQL Table Structure

When a new member of staff is added, the numerical value ID is automatically assigned by the system and is not editable by the user at any time. At registration time, a valid surname and forename must be entered, whereas location, phone number and email address are not compulsory fields and can be added later on.

### When and how is the data checked?

Final users cannot be trust, especially when there is sensitive data that can be modified, added or deleted.

For this reason, this assignment includes data checks at three stages:

- In the front-end project, when the data is inserted into text fields.
- In the back-end API file, when the data is passed from the front-end.
- In the back end DAO file, just before a SQL query is performed in the database.

The check in the text field controls only if any data has been inserted; if the input passes this check, a call to a specific API is made, which verifies that string values contains actual values, phone numbers are formed of only numbers and email addresses are valid addresses which means they must contain the @ symbol. When the data has passed step two, in case of numbers, it gets verified one more time, making sure the values entered are for example greater than zero in case of an ID.

The information validation at stage three could be avoided, however the author added this data verification in case the project is going to be reused in a future development and the API is rewritten. This will stop the introduction of errors and insertion of incorrect data in the SQL table.

### How is the data retrieved and returned to the front-end?

To allow the front-end developers to work on different data formats, the APIs can read in a requested type and if it corresponds to implemented output such as XML, JSON or Plain text, it retrieves the data from the DAO and formats it as requested for then sending it back. Consenting the retrieval of multiple types, induces the front-end developers to more flexibility, to retrieve the data in the format the person is more familiar with and more suitable to the undertaken project.

The screenshot shows how the data is retrieved from the page, wrapped into a Javascript object and sent along with the request via a JQuery method. The serialize function called on a 'Form' tag binds the data into an array assigning a key and a value for each elements contained in the Form as shown below:

```
63 //Action for when a new staff needs to be added to the system
64 $("#addNewStaffRequest").click(function(event){
65     // serialize the form and store input variable
66     // call the addStaff function, passing in the form containing the new staff data
67     var data=$("#addStaffForm").serialize();
68     addStaff(data,url);
69 });
```

Figure 3 –Serialize Data

In case more data is to be added to the serialized object, the following script will inject the additional information, in this occasion a parameter staffID and its value are inserted:

```
79 //Action for when a staff needs to be updated to the system
80 $("#updateStaffRequest").click(function(event){
81     //The form has not got an attribute ID that specifies the staff ID, therefore this will add it to the form before it is serialized;
82     var dataForm=$("#updateStaffForm").serialize()+'&'+$.param({ 'idUpdate': idUp });
83     //call the updateStaff function, passing in the form containing the updated details
84     updateStaff(dataForm,url);
85 });
```

Figure 4 – Serialize and Inject Data

The screenshot below shows the creation of a response object containing data of XML, JSON or Text format:

```
86 //Create a Response object that will contain the data in the right format to display to the final user
87 $responsePage=new Response();
88 if($staffList!=NULL){
89     $responsePage->setStatusCode(200);
90     //If the response needs to be formatted in XML
91     if($dataTypeRequested=='xml'){
92         $responsePage->headers->set('Content-Type', 'application/xml');
93         //Create a XML tag for each element found and add it to the response page
94         $xmlOutput = new SimpleXMLElement('<Staff/>');
95         foreach($staffList as $element){
96             $elementXML=$xmlOutput->addChild('StaffElement');
97             $elementXML->addChild('ID', $element->getID());
98             $elementXML->addChild('Surname', $element->getSurname());
99             $elementXML->addChild('Forename', $element->getForename());
100             $elementXML->addChild('Location', $element->getLocation());
101             $elementXML->addChild('PhoneNumber', $element->getPhoneNumber());
102             $elementXML->addChild('Email', $element->getEmail());
103         }
104         //Remove reference to $staffList
105         unset($element);
106         $responsePage->setContent($xmlOutput->asXML());
107     }
108     //If the response needs to be contain JSON objects
109     else if($dataTypeRequested=='json'){
110         $responsePage->headers->set('Content-Type', 'application/json');
111         $arrayTemp= array();
112         //Each element in the db response needs to be in array format for then be encoded to JSON
113         foreach($staffList as $element){
114             array_push($arrayTemp, $element->toArray());
115         }
116         //Remove reference to $staffList
117         unset($element);
118         $responsePage->setContent(json_encode($arrayTemp));
119     }
```

```
120 //If the response needs to be formatted in plain text
121 else if($dataTypeRequested=='plainText'){
122     $responsePage->headers->set('Content-Type', 'text/html');
123     $arrayTemp= array();
124     //Create a string to identify each object find in the db
125     foreach($staffList as $element){
126         array_push($arrayTemp, $element->toString());
127     }
128     //Remove reference to $staffList
129     unset($element);
130     $responsePage->setContent(json_encode($arrayTemp));
131 }
```

Figure 5 – Handling Formats

### How are the HTTP calls performed?

To communicate efficiently with the back-end API, the front-end takes advantage of a Javascript library named JQuery, which brings to life website transforming static to dynamic web pages. This external library uses AJAX to make the site dynamic (no need of a page reload to display new information) and also allows calls to HTTP functions.

The HTTP protocol defines eight methods, from which, five of them are mostly used for manipulating information on a system:

- GET -> retrieve
- POST -> add
- PUT -> modify
- DELETE -> remove
- OPTIONS -> return the methods allowed by the service

Despite these methods define precise actions, some of them are exchangeable; for example, it is possible to request to update or delete an existing record by using a POST request.

This technique is discouraged by the HTTP Protocol, but is widely used in development since it is easier for programmers to implement it and the request will perform successfully anyway (W3Schools, No Date).

For this project only GET and POST methods have been used, since the invocation of PUT and DELETE requests were blocked in the GAE and therefore cannot be executed by the front-end. Two examples of GET and POST requests used in this assignment are shown below.

```
269 function addStaff(data,urlReq) {
270     //Extract the surname and forename passed, since they will be used again in this section
271     var surnameTemp=$("#input[name=surnameAdd]").val();
272     var forenameTemp=$("#input[name=forenameAdd]").val();
273     $.ajax({
274         type: 'POST',
275         url: urlReq+'addMember',
276         data: data,
277         //When the post request is successfull, display the data inserted, calling the search method passing
278         //the forename and surname previously extracted
279         success: function(result){
280             //Show result of insertion
281             alert(result);
282             //Empty text fields
283             $("#forenameAdd").val('');
284             $("#surnameAdd").val('');
285             $("#locationAdd").val('');
286             $("#phoneAdd").val('');
287             $("#emailAdd").val('');
288             //Display the new member and set up the appropriate view
289             $("#staffListOutput").empty();
290             $("#searchStaff").show();
291             $("#addStaff").hide();
292             $("#updateStaff").hide();
293             searchFSStaff(forenameTemp,surnameTemp,urlReq);
294         },
295         //Display this message in case of error, to remind the user about the rules for a valid email address and phone number
296         error: function(error){
297             //console.log(error);
298             alert('Staff Details Incorrect or incomplete-->\n\nRemember:\n'+
299                 '->A valid email address contains the @ symbol\n'+
300                 '->A phone number contains only numbers, omit the + for the cuntry code, Ex: for UK(+44), use only 44');
301         }
302     });
303 }
```

Figure 6 – POST Request

```
169 function searchFSStaff(fName,sName,urlReq) {
170     //Validate the input
171     if(fName!='' && sName!=''){
172         $.ajax({
173             type: 'GET',
174             url: urlReq + fName + '/' + sName,
175             success: function(result){
176                 //Extract the JSON object from the response page
177                 if(result.ID!=null){
178                     //Set the Staff ID to not editable, in case the user decides to remove it
179                     $("#idSearch").prop("readonly", true);
180                     $("#idSearch").val(result.ID);
181                     $("#forenameSearch").val(result.Forename);
182                     $("#surnameSearch").val(result.Surname);
183                     $("#locationSearch").val(result.Location);
184                     $("#phoneSearch").val(result.PhoneNumber);
185                     $("#emailSearch").val(result.Email);
186                 }
187                 //In case the object returned does not contain a valid id, this means no such object with the
188                 //given forename and surname exists in the system, therefore return this message
189                 else{
190                     alert('No Staff Found!');
191                 }
192             },
193             timeout: 10000,
194             error: function(error){
195                 //Display error message
196                 alert('Error: '+error.statusText);
197             }
198         });
199     }
200     //Ask the user to re-insert a valid forename and surname
201     else{
202         alert('Insert a valid forename and surname');
203     }
204 }
```

Figure 7 – GET Request

### How are comments written?

Comments are very important because they help explaining your solution to a problem to others. For this assignment, the author followed the professional Java Doc standards, which is an international standard for commenting code written in Java, PHP, Python and many other languages.

It is important to use this technique when the code is used by other developers to make them understand the procedures and algorithms implemented. It is also implemented in most professional environments to make the code reusable.

### Have any Design Pattern been used?

Design patterns are techniques adopted by developers to make the code reusable and adaptable to various situations. For this assignment the Singleton design pattern has been used when a database connection is requested by any of the methods inside the DAO class. The implementation allows the creation of only one connection at a time to avoid concurrency and data manipulation errors. This technique is already part of the PDO database connection management in PHP in a way that the connection is automatically closed by the PDO service once a query has been consumed and the execution exits the scope of the method. Moreover, the author destroys each connection object after every use, to decrease even more any minimal chance of having multiple opened connections at once.

### Problems Encountered

During the development of this assignment, the author incurred into some problems; most of them have been solved, however there is one issue that has not find solution yet.

Issue	Solution
<b>GAE blocks some of the PHP function for security reason, and it may happen that some of them are needed</b>	Write a php.ini file that contains a list of enabled methods, previously blocked. Once the project is deployed to the cloud, the PHP server will find and read the .ini file and change its setup to enable the requested services.
<b>A folder named Resources was created inside the main project folder to separate the html/php pages from the Javascript ones. However, this folder was not visible by the server and consequently the web site became static and unresponsive to button click and other events.</b>	An edit to the app.yaml file is essential to allow the server to see subfolders. The edit consists of adding this lines to the already existing code: <pre>12 - url: /Resources 13   static_dir: Resources 14</pre>
<b>When a project is deployed, the changes made do not take place straight away.</b>	This is a problem of cloud services. Servers are places around the globe and when a project is saved to the cloud, all the changes need to be rectified on the whole provider network. This may take from few minutes to hours, depending on the



	workload these machine need to perform. For this reason, the only solution is waiting.
<b>The author has decided to split this assignment into two separate projects, one that serves the back-end and one for the front-end. When the front-end tries to make HTTP calls to the API (back-end), the back-end returns a XMLHttpRequest error saying that the request to the server has been blocked because the request header does not contain a 'Access-Control-Allow-Origin' resource.</b>	<p>This happens when a web site tries to perform actions on another path, external to the one the website is using, this needs to be allowed by the external site.</p> <p>To do so, the back-end file that deals with requests and responses, the API.php file in this case, needs to have the following code to allow the front-end project to work:</p>
<pre>20 //This sets the header of any response sent back, with the aim of handling possible CORS issues 21 \$app-&gt;after(function (Request \$request, Response \$response) { 22     //All other domain to connect to this api 23     \$response-&gt;headers-&gt;set("Access-Control-Allow-Origin","https://mike-assignment-frontend.appspot.com"); 24     //Specify the methods accepted by this web service 25     \$response-&gt;headers-&gt;set("Access-Control-Allow-Methods","GET,POST,PUT,DELETE,OPTIONS"); 26     // \$response-&gt;headers-&gt;set("Access-Control-Allow-Headers", "X-Requested-With,Content-Type"); 27 });</pre>	
<b>For a correct use of the HTTP protocols, it is asked to make use of the 'PUT' and 'DELETE' methods in order to update and delete a records. However, even after this methods have been enabled in the header of any response in the API, the back-end keeps blocking such requests, returning a CORS (Cross-Origin Resource Share) error which it should have been solved with the previous problem solution.</b>	<p>The author has not find a solution to this problem so far, and the way around it was to change the 'PUT' and 'DELETE' requests to 'POST' requests. The 'POST' hides the parameters that are sent to the server API and accomplish the same result.</p> <p>However, this is not accepted by the HTTP Protocol Standards, even if it is used by many websites.</p>



Application Screenshots

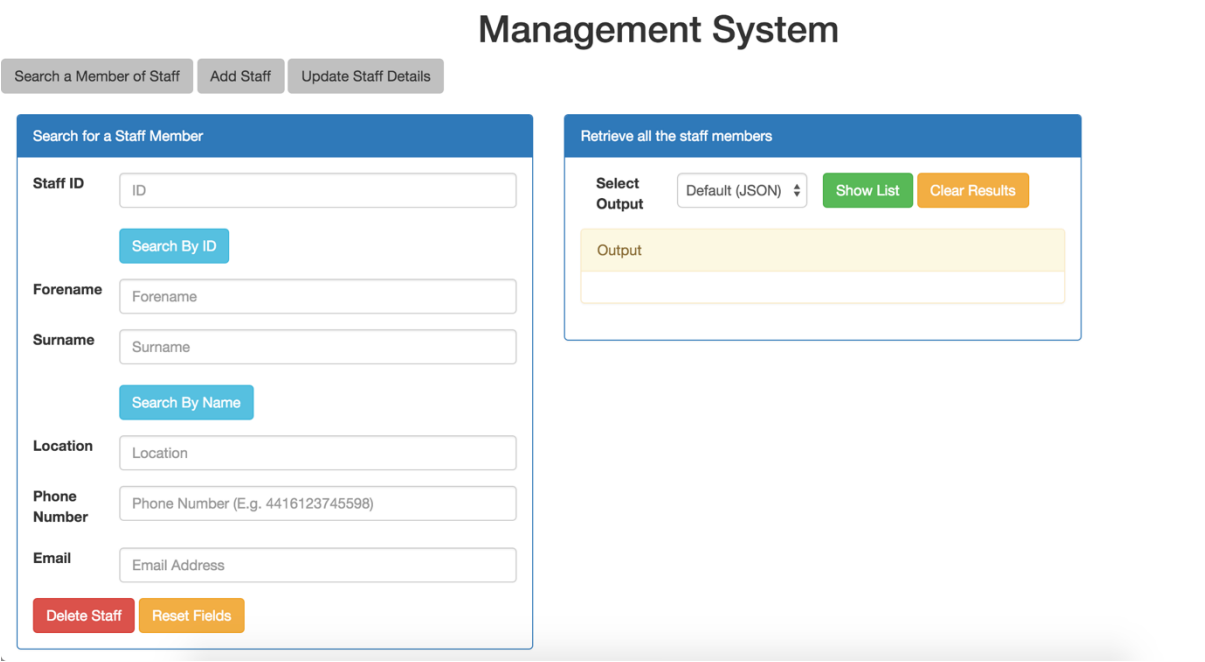


Figure 8 – Home Page

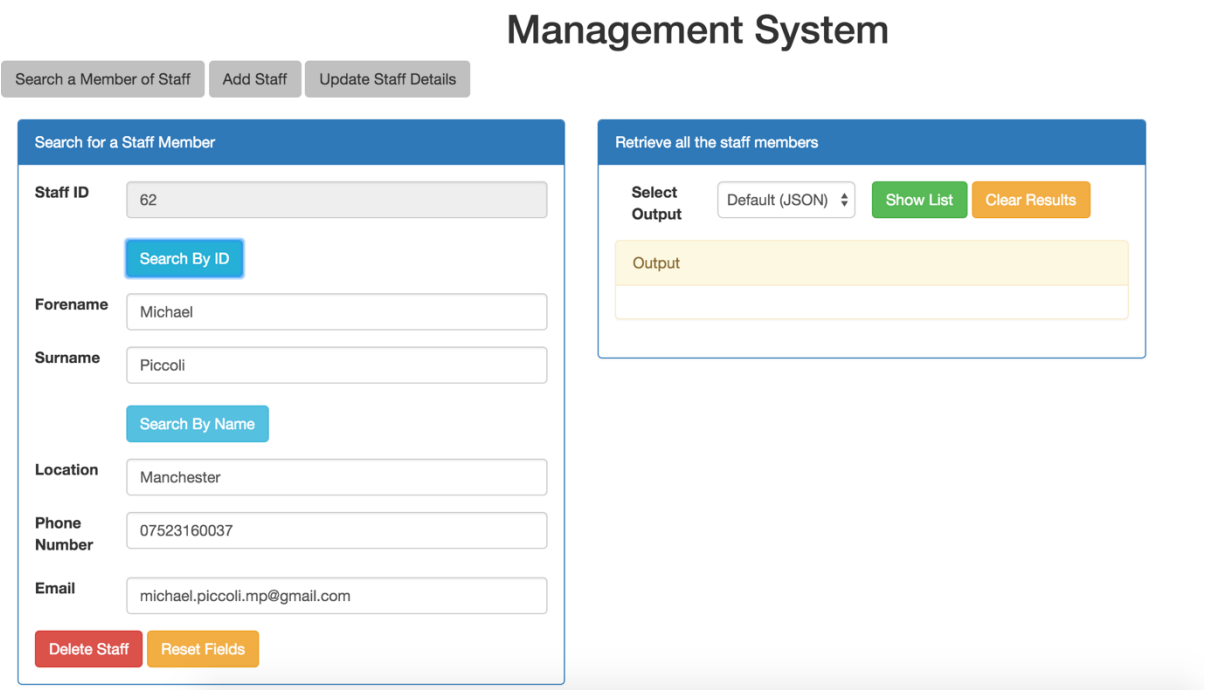


Figure 9 – Search Staff

## Management System

### Add a new Staff Member

Forename

Surname

Location

Email Address

Phone Number

Figure 10 – Add Staff

## Management System

### Update Staff Details

ID:

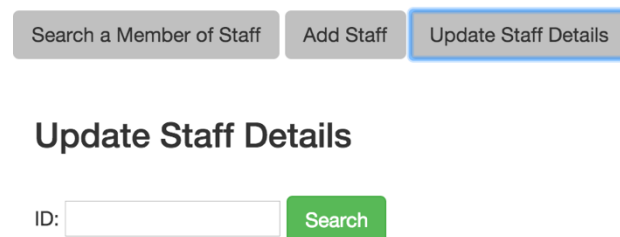


Figure 11 – Update Staff Data Part 1

## Management System

Search a Member of Staff

Add Staff

Update Staff Details

### Update Staff Details

ID: 62

Search

Forename

Michael

Surname

Piccoli

Location

Manchester

Email  
Address

michael.piccoli.mp@gmail.com

Phone  
Number

07523160037

Update

Undo Changes

Figure 12 – Update Staff Data Part 2

## Tests

Test N.	Name	Description	Expected Result	Test Result	Pass-Fail
1	Search Invalid ID	Search Staff by ID inserting an invalid ID	Error	As Expected	Pass
2	Search Valid ID	Search Staff by ID with valid ID	Staff Details filled in text fields	As Expected	Pass
3	Search Invalid Name	Search Staff by name with invalid name	Error	As Expected	Pass
4	Search Valid Name	Search Staff by name with valid name	Staff Details filled in text fields	As Expected	Pass
5	Delete Invalid ID	Delete Staff after searching invalid ID	Error	As Expected	Pass
6	Delete ID	Delete Staff stored in DB	Delete Completed	As Expected	Pass
7	Get JSON	Retrieve all member of Staff JSON	Display all Staff Data	As Expected	Pass
8	Get XML	Retrieve all member of Staff XML	Display all Staff Data	As Expected	Pass
9	Get Text	Retrieve all member of Staff PlainText	Display all Staff Data	As Expected	Pass
10	Add Invalid Staff	Add member of Staff with invalid email address and phone number	Error	As Expected	Pass
11	Add Valid Staff	Add a valid Staff to the system	Insertion Completed and new Staff displayed in the search section	As Expected	Pass
12	Update Invalid Staff	Update a staff details with incorrect email address and phone number	Error	As Expected	Pass
13	Update Valid Staff	Update a staff with valid email and phone number	Update Completed and new Staff displayed in the search section	As Expected	Pass

## Conclusion

Even though, this was a challenging assignment, the developer found it particularly enjoyable since it taught him new technologies, libraries and programming language that he can now implement in his own personal projects outside university.

This is a good project since it performs flawlessly, bugs have been fixed and the front-end is easy to use and it achieves everything it has been requested with extra features added to it such as, editing the staff information or search a member of staff by ID.

The tests have helped to find bugs the author did not meet during the implementation and he believes it is a good project to show to employers and insert to a personal portfolio.

Features that could be improved are:

- The use of "DELETE" and "PUT" methods instead of only "POST".
- A designed Front-end.
- Produce more test cases using the appropriate PHPUnit framework to disclose hidden bugs.

## References

- 1) OData. (2015) *URI Convention (OData Version 2.0)*. [Online][Accessed on 4<sup>th</sup> January 2016] Available from: <http://www.odata.org/documentation/odata-version-2-0/uri-conventions/>
- 2) TrinityTuts. (2015) *Build Your First Web Service with PHP, JSON and MySQL*. [Online][Accessed on 4<sup>th</sup> January 2016] Available from: <https://trinitytuts.com/build-first-web-service-php/>
- 3) CodeDiesel. (2014) *PHP application with Google App Engine*. [Online][Accessed on 5<sup>th</sup> January 2016] Available from: <http://www.codediesel.com/php/php-applications-on-google-app-engine/>
- 4) Wurzer, E. (2012) *Why You should be using PHP's PDO for Database access*. [Online][Accessed on 5<sup>th</sup> January 2016] Available from: <http://code.tutsplus.com/tutorials/why-you-should-be-using-phps-pdo-for-database-access--net-12059>
- 5) MDN. (2015) *HTTP Access Control (CORS)*. [Online][Accessed on 5<sup>th</sup> January 2016] Available from: [https://developer.mozilla.org/en-US/docs/Web/HTTP/Access\\_control\\_CORS](https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS)
- 6) Kirby, M. (2013) *Creating an efficient REST API with HTTP*. [Online][Accessed on 5<sup>th</sup> January 2016] Available from: <http://mark-kirby.co.uk/2013/creating-a-true-rest-api/>
- 7) Devaublanc, B. (2013) *Silex Simple REST*. [Online][Accessed on 6<sup>th</sup> January 2016] Available from: <https://github.com/vesparny/silex-simple-rest/blob/master/src/app.php>
- 8) Symfony. (No Date) *Routing*. [Online][Accessed on 6<sup>th</sup> January 2016] Available from: <http://symfony.com/doc/current/book/routing.html>
- 9) W3Schools. (No Date) *The MVC Programming Model*. [Online][Accessed on 7<sup>th</sup> January 2016] Available from: [http://www.w3schools.com/aspnet/mvc\\_intro.asp](http://www.w3schools.com/aspnet/mvc_intro.asp)
- 10) SoapUI. (No Date) *SOAP vs REST challenges*. [Online][Accessed on 10<sup>th</sup> January 2016] Available from: <http://www.soapui.org/testing-dojoworld-of-api-testing/soap-vs--rest-challenges.html>

- 11) Static Apps. (2014) *Cross-Domain Requests with CORS*. [Online][Accessed on 5<sup>th</sup> January 2016] Available from: <https://staticapps.org/articles/cross-domain-requests-with-cors/>
- 12) Fote, W. (2013) *What Is AJAX and Where it is Used in Technology?* [Online][Accessed on 10<sup>th</sup> January 2016] Available from: <http://www.seguetech.com/blog/2013/03/12/what-is-ajax-and-where-is-it-used-in-technology>
- 13) JQuery. (No Date) *What is JQuery?* [Online][Accessed on 10<sup>th</sup> January 2016] Available from: <https://jquery.com/>
- 14) Spies, B. (2008) *Web Services Part 1: SOAP vs REST*. [Online][Accessed on 11<sup>th</sup> January 2016] Available from: <http://www.ajaxonomy.com/2008/xml/web-services-part-1-soap-vs-rest>
- 15) W3Schools. (No Date) *HTTP Method Definitions*. [Online][Accessed on 8<sup>th</sup> January 2016] Available from: <https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html>

## Table of Content

<b>INTRODUCTION</b>	<b>1</b>
<b>CONTENT</b>	<b>1</b>
<b>API ROUTES</b>	<b>2</b>
<b>HOW IS THE DATA ACCESSED?</b>	<b>2</b>
<b>HOW IS THE DATA STORED?</b>	<b>3</b>
<b>WHEN AND HOW IS THE DATA CHECKED?</b>	<b>3</b>
<b>HOW IS THE DATA RETRIEVED AND RETURNED TO THE FRONT-END?</b>	<b>3</b>
<b>HOW ARE THE HTTP CALLS PERFORMED?</b>	<b>5</b>
<b>HOW ARE COMMENTS WRITTEN?</b>	<b>7</b>
<b>HAVE ANY DESIGN PATTERN BEEN USED?</b>	<b>7</b>
<b>PROBLEMS ENCOUNTERED</b>	<b>7</b>
<b>APPLICATION SCREENSHOTS</b>	<b>9</b>
<b>TESTS</b>	<b>12</b>
<b>CONCLUSION</b>	<b>13</b>
<b>REFERENCES</b>	<b>13</b>
<b>TABLE OF CONTENT</b>	<b>14</b>