**Appendices R**

**Website**

As explained in the report, for this development I decided to develop and implement a website to allow MSc Properties to advertise services they provide through the Internet, and enable potential customers to submit a service request, notifying MSc Properties of the customer interest, and as explained the appendices is broken down in to two areas:

* Managing Communication with MySQL Server
* Advertise MSc Properties Services
* Service Requests

However, before I explain the process in which I went through to develop and implement a website that connects with the MySQL Server, advertises MSc Properties services, and accept service requests from potential customers and route these to the required office, I needed to install and run web server software as outlined in the literature review within the report.

The web server will deal with processing any web pages I have developed in HTML, PHP and JavaScript, that sit within my web server directory, and will then distribute these to clients as and when they request a web page. Additionally, the web server will deal with the interaction with the MySQL server outlined in Iteration Cycle 2, which will enable me to extract system information such as office details and property details and advertise them.

As I am running a MySQL Server and am running my server on a Windows machine, I decided to install the windows web development environment, called Windows, Apache, MySQL and PHP (WAMP), and as highlighted by the name, adopts the Apache Web Server, the MySQL database management system, processes PHP server side script, and runs on a Windows machine, which meets all of the current requirements for the server set up required.

I downloaded the WAMP package from <http://www.wampserver.com/en/> and once installed, the WAMP package sets up a web server directory as shown in Fig. 1.

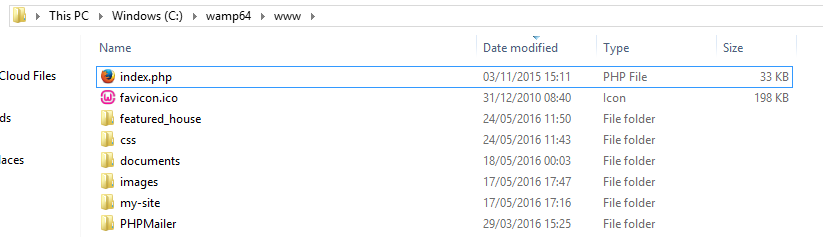


Fig. 1 – Web Server Development Area

Now that I have a web server set up and running on my machine (but it can be set up on a remote machine), I can then start developing my webpages, in HTML, PHP and JavaScript, ensuring that I place the webpages within the web server development area (inside the www folder), and ensuring that any references to webpages in different folders, use the correct path location, to ensure the webpage is able to be located.

1. **Managing Communication with MySQL Server**

As shown above in Fig. 1, upon installing the WAMP package, a web development area was created, which is where I will then place any HTML or PHP files I have developed for the website.

Firstly, as my website will interact with the MySQL database management system implemented in Iteration Cycle 2, I developed a PHP file called “MSc\_Properties - Common\_Functions.php”, which lists a number of functions written in PHP script and deals with a number of common functions such as:

* Connecting to MySQL Server
* Selecting data from MySQL Server
* Inserting, updating and deleting data from MySQL Server

This PHP file will then be included I other PHP files which will want to make use of any of the functions implemented within the common function PHP file. And as with any PHP script, the functions within the common function PHP file is enclosed within the opening (<?php) and closing (?>) tags.

* 1. **Connecting to MySQL Server**

As shown in Fig. 2, I declare a connectDatabase() function, where I invoke the mysqli\_connect() PHP function, and supply the database settings as parameter values, ‘localhost’, ‘ADMIN’, ‘Toxic9489!999’, and ‘msc\_propertieslive’, which are the host name, username, password, and database schema name for the database server respectively. By invoking mysqli\_connect() method, it will open a new connection to the MySQL server, and providing the information supplied is valid, will return an object representing the connection, otherwise an error.

Now I have an object representing the connection, I invoke mysqli\_connect\_errno, and pass the return value to an if statement, which will be false if there is no error encountered and will then return the object for the connection, otherwise, true, and then print an error message along with the actual error encountered.



Fig. 2 – Extract from Common Function PHP file – connectDatabase()

Now I have an object representing the connection to the MySQL server, I am then able to execute statements and queries on the database.

* 1. **Selecting data from MySQL Database**

As shown in Fig. 3, I declare a selectResults function which takes a variable called statement (which will be the SQL select statement, the client of Common Functions wishes to execute), and I firstly define three variables, of which one is an array and one is an empty String, and then invoke the connectDatabase() method defined earlier and assign the returned database connection object to the other variable.

I then check there is no errors connecting to the database as before, and if there is I add an error message to the array defined earlier and if not I then invoke mysqli\_query() and pass the statement variable passed to the selectResults method as parameter, along with the database connection variable. This then executes the query on the database and returns false if errors in statement and adds an error message to the array defined earlier, similarly to earlier, or because it is a select statement, returns the SQL results object, I then check to see if the results object is false, if so I produce an error message and print this error message, otherwise, I invoke mysqli\_num\_rows() and pass the results object returned from mysqli\_query() method, which returns the number of rows for the results object, I then invoke the PHP function array\_push() and add the number of results to the beginning of the array defined at the beginning of the function. I then use a for loop to traverse through the results object and invoke mysqli\_fetch\_array() and pass the results object, which returns the current row object, and is then added to the end of the array, so I end up with an array that has the number of rows as the first object within the array, and each row returned in the following array positions.

I then return the array object to the client invoking the selectResults() method, allowing the client PHP file to then use the results array to go through the results as shown in Fig. 5.

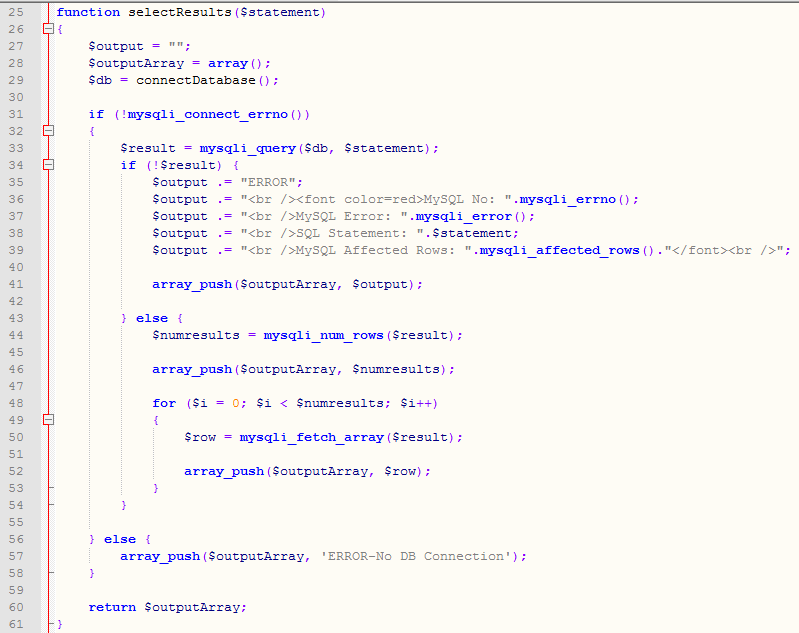


Fig. 3 – Extract from Common Function PHP file – selectResults()



Fig. 4 – Extract from Office Details PHP file – include common functions



Fig. 5 – Extract from Office Details PHP file

As you can see from Fig. 4, I firstly have to include the common functions php file, to enable me to invoke selectResults() function, I then build the SQL select statement I wish to execute on the SQL server, and then execute the selectResults function, passing the SQL select statement as parameter. As explained previously, the selectResults method will return an array either containing the number of rows and the results rows within each element of the array, or an error message within the first element of the array.

So I then check to see if the array element at position 0, contains an “ERROR”, if so then print an error message, if not then I then extract the information relating to the SQL statement, and in this instance, because there should only be 1 row of results, I am only concerned with the array element at position 1, and because it is a results row object, I am able to select the specific row information from the Select statement and add the field values to a variable, and in this instance I just want to pass the variables to a local printAddress() method.

* 1. **Inserting, Updating and Deleting data from MySQL Database**

As shown in Fig. 6, I declare a iduResults function which takes a variable called statement (which will be the SQL insert, delete or update query, the client of Common Functions wishes to execute), and I firstly define two variables, of which one is an empty string, and then invoke the connectDatabase() method defined earlier and assign the returned database connection object to the other variable.



Fig. 6 – Extract from Common Function PHP file – iduResults()

Again similarly with the selectResults function I then check there is no errors connecting to the database, and if there is I add an error message to the empty string variable defined earlier and if not I then invoke mysqli\_query() and pass the statement variable passed to the iduResults method as parameter, along with the database connection variable. The only difference is that instead of fetching rows from a results object returned from mysqli\_query(), I just need to return the number of rows affected by invoking mysqli\_affected\_rows().

I then return the string variable to the client invoking the iduResults() method, allowing the client PHP file to then use the string variable to identify if an insert, delete or update occurred as shown in Fig. 7.

When I first implemented this function, I was going to use it to insert a value into the database, however during the development stage it was decided that this will be implemented in a different way.

1. **Advertise MSc Properties Services**

Now that I have explained how I am able to interact with the MySQL server, I am now going to explain how the information from the business is then published as a HTML page to be viewed by potential customer.

For this development I developed 5 main pages (Home, Rent, Let, Careers, and About us), where I use a number of HTML display components to enable me to format the display of my website, and for users of the website to interact with the webpages, such as:

* Cascade Style Sheets
* Images
* Links
* Forms
* Tables
* Drop Down Box
* Buttons
* Text Entry Fields and Areas

Although I am not going to explain how I have implemented each of the elements within my website, I am going to explain the main process of advertising properties for MSc Properties, which will make use of some of the MySQL server communication functions outlined above.

To implement the view properties functionality within the website, I firstly needed to implement a form, which allows potential customers to enter search information, to allow the potential customer to filter the properties returned. For the website, I decided to implement a quick search form, where the potential customer can filter properties searched on using basic information, and I implemented a full search, to filter properties searched on using the full range of property details.

Firstly, to implement a HTML webpage, I firstly had to define a HTML DOCTYPE, which is outlined in the first line of the HTML/PHP file shown in Fig. 7, I then have to define a number of HTML tags which break up the document content into sections, which are:

* <html> section – This section describes a HTML document
* <head> - This section provides information about the document
* <title> - This section provides a title for the document
* <body> - This section provides the visible page content
* <h?> - This section defines text to be a heading, with the ‘?’ defining the header size
* <p> - This section defines text to be a paragraph

From the sections defined above, I am going to concentrate on explaining my implementation for the body section, as this is the main section of the HTML file, where the visible page content is defined. However, as you can see from Fig, 7, I define a DOCTYPE, which is for a HTML 4.01 file, I create a title for “MSc Properties” and lastly I define a link tag, which provides a link to an external style sheet, defined in CSS and called “msc\_properties.css”, saved within a folder called css, saved within the root web directory called www.

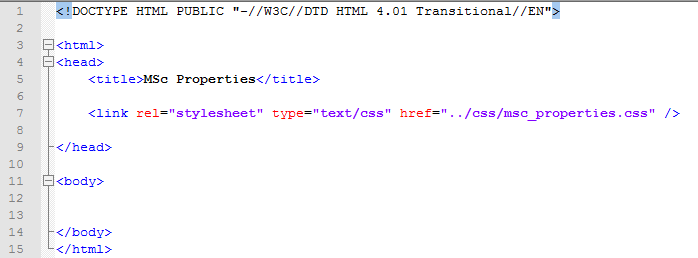


Fig. 7 – HTML file template

Additionally, to the template outlined above, I add two div.’s to the body of the HTML file, of which one is defined as toolbar, which provides the navigation to the main HTML/PHP files and is outlined in the CSS file, whereas the other div is to hold the image that is MSc Properties business logo.

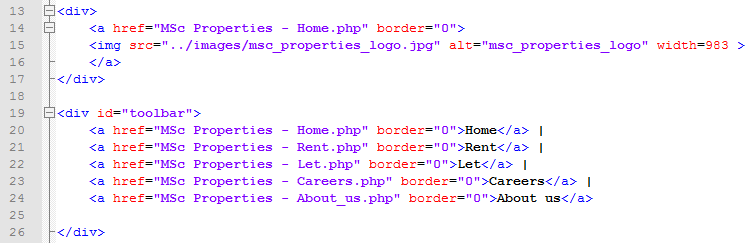


Fig. 8 – Extract from Home HTML file

As you can see from Fig. 8, I use the <a> tag within the anonymous div to define a link to another HTML/PHP file, where I use the href attribute to define the link’s destination, and the <img> tag to define an image source and size, of which both combined, produce an image (MSc Properties logo) that is a link to the Home, page, which allows users of the website to be directed to the Home page from wherever they are within the website.

I then use the <a> tag again within the toolbar div, to define 5 links to different HTML/PHP files, but instead of using an image as the component to be displayed to the user to act as the link, I use text to display the link to the user, as shown in Fig. 8.

Now that I have outlined the template of the HTML/PHP files I am to develop, I am now going to explain the process undertaken to implement the view properties functionality, which will deal with data entry from the user to provide some search criteria to search on, and then compiling a SQL statement from the data entered, and connecting to and compiling the SQL statement on the SQL server, and finally getting the returned result set from the SQL server and displaying the results back to the user through a webpage.

As explained, I implemented quick search and full search functionality to allow a user to search for properties, and as shown in Fig. 9, to do this I used a HTML form to enable the user to enter data, which will be collected by the browser. I then use the method attribute (POST) and supply the HTML/PHP file I would like to run upon submission of this form, of which I am running the View Properties 2 PHP file.

Now I have set up the form in which I want to collect the data from the user, I create a HTML table with no border, which will be used to structure my data entry components display, to ensure the components are displayed neatly.



Fig. 8 – Extract from Home PHP file

As you can see from Fig. 8, I use the <tr> and <td> tags to define the rows and data (cells) within the table, and implement a table with 3 rows, and 2 columns, of which the 1st column is displaying the name of the data to be entered, and the second column is displaying the HTML data entry component.

The first data entry component is the “City” of the property to search on, and I just enter plain text into the cell, and just amend text using the <strong> tag, and the actual data entry component (Drop Down Box) into the second cell. To implement the drop down box I use the HTML select tag, and supply a name, id and size for the drop down box. I then use the option tag, to create the drop down box list, and define the value and name of the list element for each list element. As the city drop down box contains content from the MySQL server, I have to connect use a function defined in common functions and shown in Fig. 9.

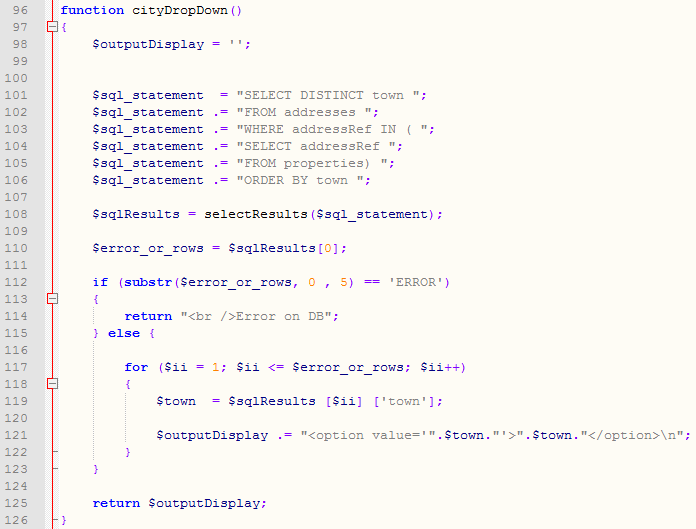


Fig. 9 – Extract from Common Functions PHP file – cityDropDown()

Firstly, I define an empty string variable called outputDisplay, and as shown previously, I compile the select statement I want to execute, which will return all of the different cities associated with a property as shown in Fig. 9.I then supply the SQL statement to the selectResults() method as a parameter, and store the returned array in an sqlResults variable.

I then extract the first element in the array and check to see if there are any errors from the executed statement, if so return an error message. If not, I then traverse through the array with a for loop extracting the town value, and produce an option tag, with the value and name of the element for the list defined by the value held within array, produce from the SQL statement executed on the MySQL server, where the option tag is then added to the outputDisplay variable, and then the outputDisplay is returned to the client invoking the cityDropDown() function, of which in the example I am discussing it is the Home PHP file, for the quickSearch div. shown in Fig. 8.

Moving back to the Home PHP file, I then print the return value from the cityDropDown() function, and carry out a similar task for the Area drop down component. I then create a beds drop down, but as this is just a drop down box from set values I just need to produce the drop down from hardcoded HTML and PHP (as I use a PHP for loop to produce a list of beds), I then close all tags as required.

Now that I have the table of display components implemented, I then need to implement a button to allow users to confirm their data entry and search for the properties, and to do this I use the <input> tag, and use the ‘submit’ type and assign the value name ‘Search’, which will give the button the visual name of ‘Search’. And for display purposes, I place the button inside a new div., inside the quickSearch div., and align the button to the right of the div., using the ‘align=right’ style.

So now I have a form implemented, that has data entry options for the user and a button to enable the user to submit the form and perform the search, I am going to move to the View Properties 2 PHP file (is the file I defined the form to link to when the form is submitted), and pick up from where the search results processing starts from.



Fig. 10 – Extract from View Properties 2 PHP file – viewProperties part 1

As you can see from Fig. 10, I define a div. viewproperties, and assign a header to the div. called Property Search. Again I display the information within a HTML table and define a border, cell padding, and col. width for both columns within the table. I then open up a PHP script section, using the opening PHP tag, and then use the PHP $\_POST function to return the town, area and beds variables defined by the user, and collected within the form in the Home PHP file. Additionally, as this is the quick search, I have to define the other elements within a full search criteria as not been selected using the ‘-’ character.

I then compile an SQL statement to return all of the ‘VOID’ properties within the MySQL database, execute it on the MySQL server using the selectResults() function, and store the returned results in a sqlResults variable, and go through the usual error checking.

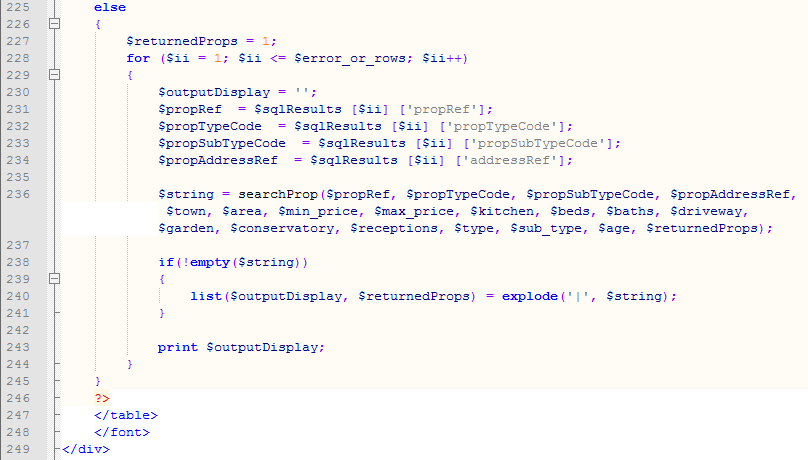


Fig. 10 - Extract from View Properties 2 PHP file – viewProperties part 2

Now I have the array results stored in sqlResults variable, I am then able to extract the required information from the array using the column names and store them in variables, and because the array of results may contain more than one row I perform a for loop to traverse over the elements using the number of rows stored in the error\_or\_rows variable, and extract property information with each traverse and invoke the searchProp function, supplying the property variables extracted from the results array, along with the 3 variables extracted from the Home PHP file, and the dummy variables to complete the full search details.

As you can see from Fig. 11 to Fig. 23 (searchResults() function implementation), I define an empty String called outputDisplay and a boolean value called true, I then construct a number of SQL statements to return a number of property related bits of information, and each time I return the property information from the MySQL server, I compare the results to check to see if they match those of the search criteria, if the information does not match then the returnProp boolean variable is assigned the value false, and if not I move on to comparing the next property related piece of information.



Fig. 11 – Extract from Common Functions – searchProp() part 1

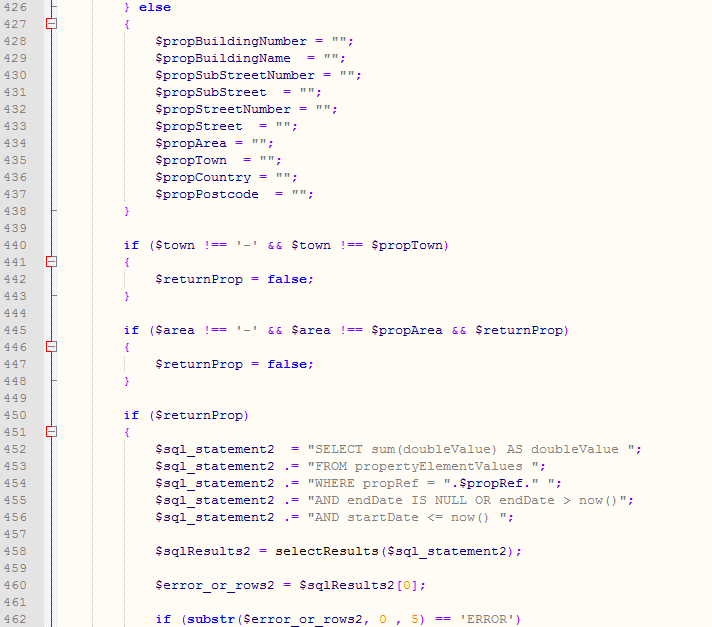


Fig. 12 – Extract from Common Functions – searchProp() part 2



Fig. 13 – Extract from Common Functions – searchProp() part 3



Fig. 14 – Extract from Common Functions – searchProp() part 4

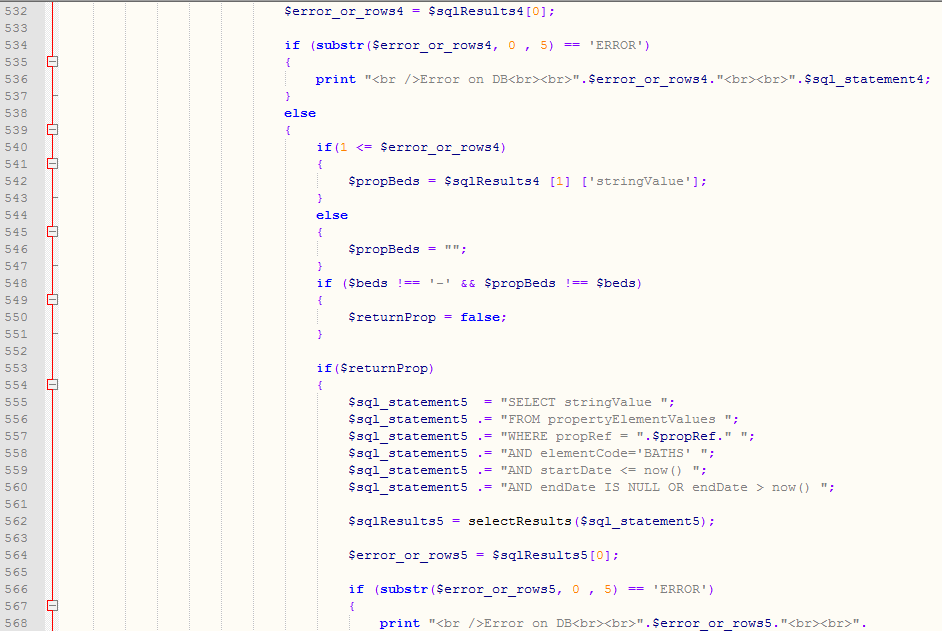


Fig. 15 – Extract from Common Functions – searchProp() part 5



Fig. 15 – Extract from Common Functions – searchProp() part 5



Fig. 16 – Extract from Common Functions – searchProp() part 6

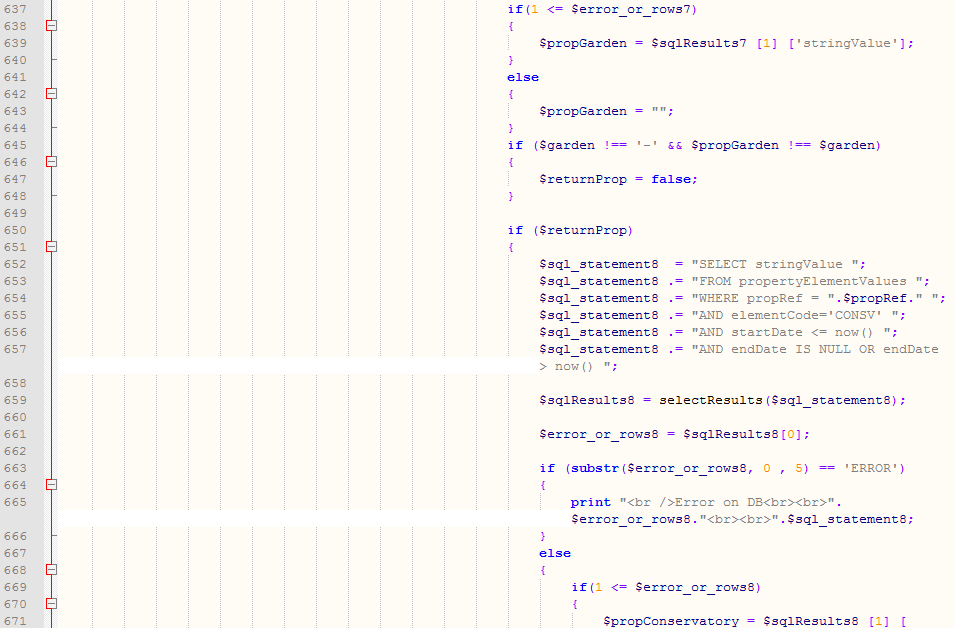


Fig. 17 – Extract from Common Functions – searchProp() part 7

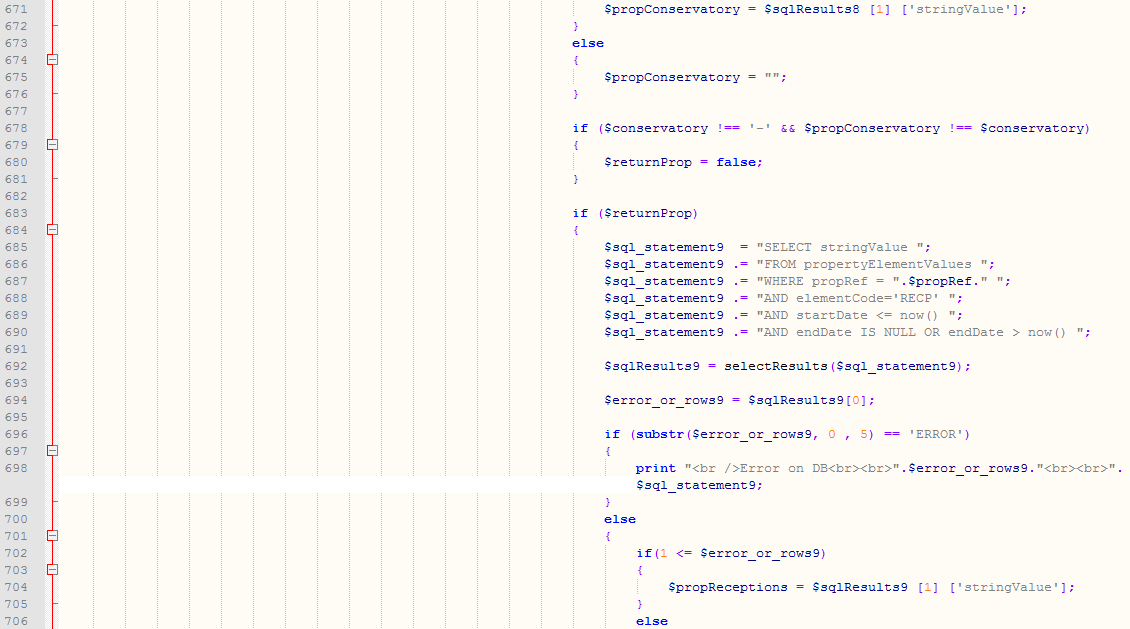


Fig. 18 – Extract from Common Functions – searchProp() part 8

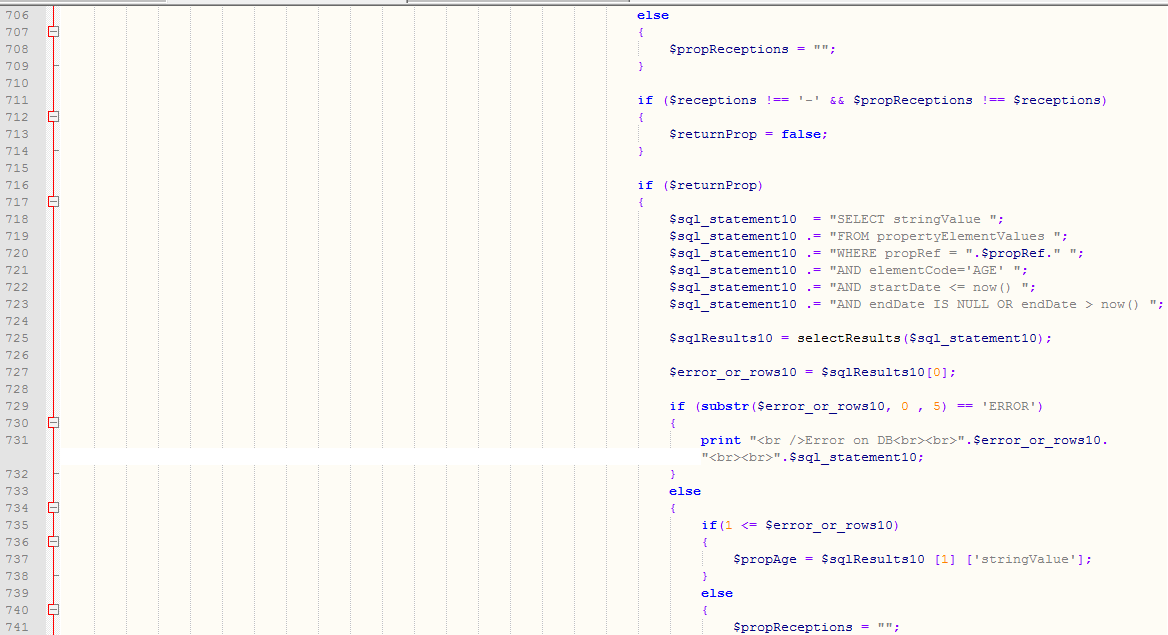


Fig. 19 – Extract from Common Functions – searchProp() part 9

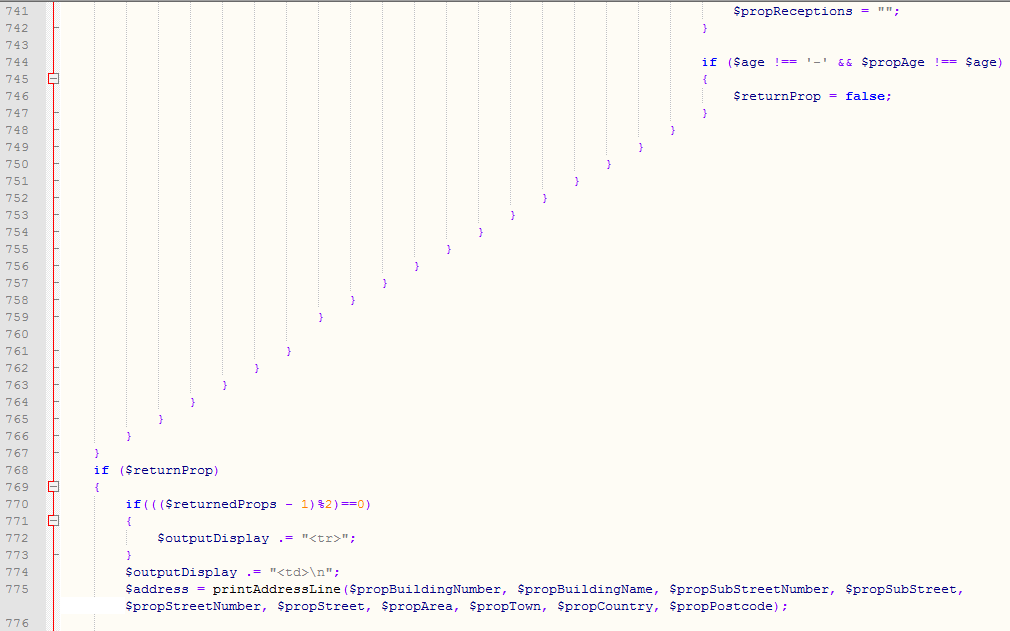


Fig. 20 – Extract from Common Functions – searchProp() part 10



Fig. 21 – Extract from Common Functions – searchProp() part 11

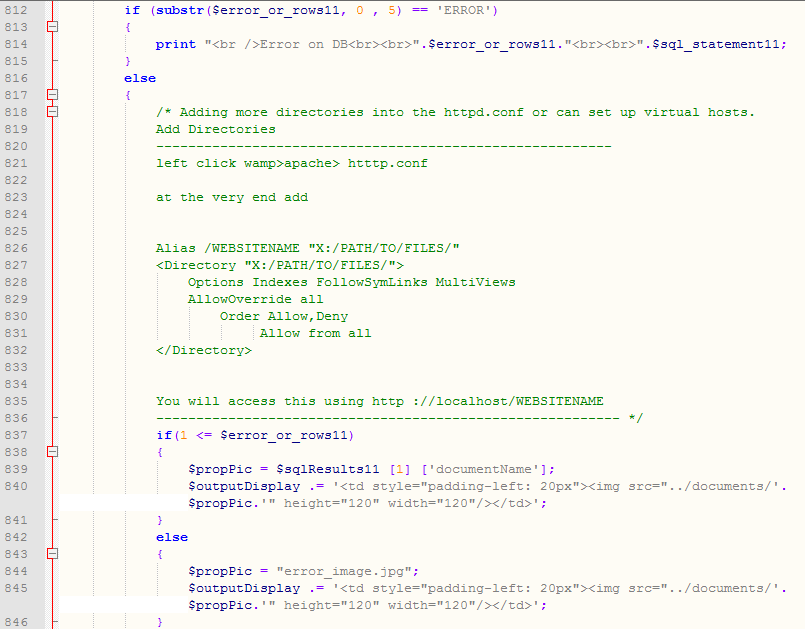


Fig. 22 – Extract from Common Functions – searchProp() part 12

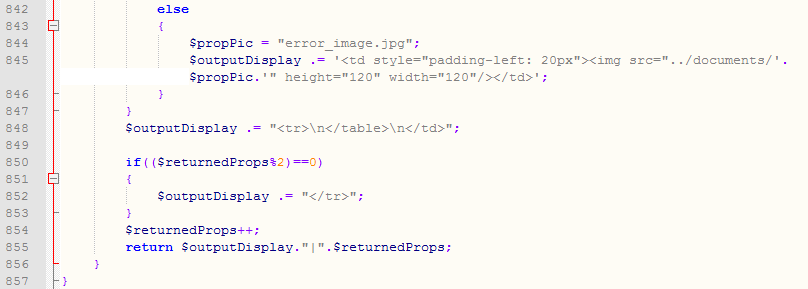


Fig. 23 – Extract from Common Functions – searchProp() part 13

Once I compared each piece of property information with the search criteria, and if the property information matches that defined by the search criteria, the returnProp variable defined will still be true, and I then check to see if the returnProp variable is true, if it is, I then check to see if the property I am currently returning information for is an even or odd number within the current properties to return (to determine if the property being added to the table within View Properties 2 PHP file is on a new row or not), and if so add a <tr> tag to the outputDisplay variable, and then add the <td> tag to the outputDisplay variable.

I then format the property information and add it to outputDisplay variable, and then add the closing <td> tag. I then compile a SQL statement which gets the first picture document path for the property (Document saved through the Document Management system outlined in Iteration Cycle 4), and execute the SQL statement on the SQL server and store the array result in sqlResults11 variable. I then carry out the usual error checking, and if there has been a property picture returned, I get the documentPath and use it to create an image within a table cell and add this to the outputDisplay, if there is no property picture returned, then a default property picture is supplied instead.

Finally, I add the closing tags for the columns and tables, and then check to see if the property is at the end of a row and needs to close the table row from the View Properties 2 PHP file. I then increment the returnProp variable passed from View Properties 2 PHP file and return the outputDisplay concatenated with a ‘|’ character and the returnProp.

Now moving back to Fig. 10, within the View Properties 2 PHP file, I return the string value that was just concatenated within common functions searchProp function, into a variable called string, I then check to see if string is empty, and if not I then invoke the explode() PHP function and pass the ‘|’ character and the string as parameters, and return the outputDisplay variable and returnProp variable into variables using the ‘|’ character to split the two strings, and the list() PHP function.

I then print the outputDisplay variable and continue traversing through the array of results with the for loop, invoking the same searchProp function, but with different property details, but the same search criteria, until all the properties have been search over and the property information printed.

1. **Service Requests**

The final area for the website I am going to discuss, is potential customer submitting a service request to an office, and as explained in the report, I implement email functionality, to enable the webserver to send an email to the office email address notifying them of the potential customer interest.

For me to implement this email functionality, I adopted similar data collecting functionality as to the previous example, as well as making use of the PHPMailer package downloaded from <https://sourceforge.net/projects/phpmailer/>. This package enabled me to generate and send an email from my WAMP server, without having to run my own mailing server.

As explained, firstly I have to collect the information from the user, such as contact details and possibly describing their service request, using techniques similar to shown previously. As there is two areas that potential customers can submit a service request for (Letting or Renting), there is two service request forms, one within the Let PHP file, and one within the Rent PHP file, I am going to go through the service request for the Rent PHP file as shown in Fig. 24 and Fig. 25



Fig. 24 – Extract from Rent PHP file

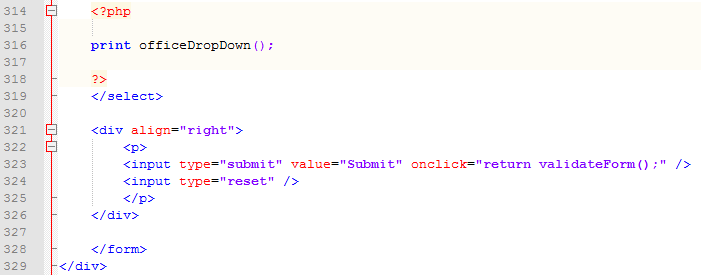


Fig. 25 – Extract from Rent PHP file

As you can see from Fig. 24, I create a header, and generate a form which links to another PHP file, and a table with two columns and 5 rows, similarly to before but instead I create text field inputs using the <input> tag but of type text and a text area input using the <textarea> tag, as well as drop down boxes shown before. The only other difference with this data entry form is that the submit button has an onclick attribute which is “return validateForm();”, and this refers to JavaScript defined at the top of the HTML/PHP file to ensure that the data entered is valid, and if it is not valid then the form will not link to the HTML/PHP file defined in the <form> tag, otherwise links to the HTML/PHP file.

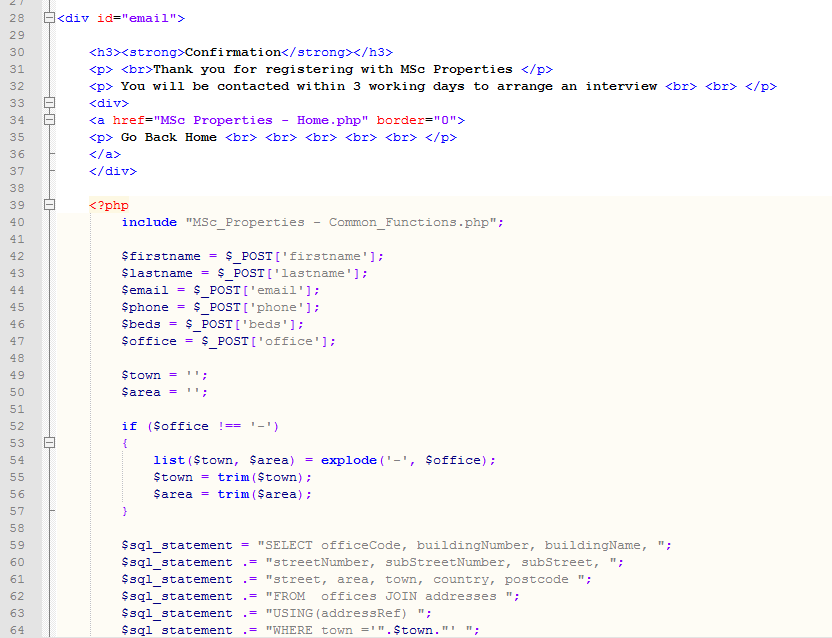


Fig. 26 – Extract from SendMail PHP file

As you can see from Fig. 26, I define a div. called email and create a heading and some confirmation text within a paragraph along with a link to go back to the home page. I then open a PHP script section and include the common functions PHP file and get the information collected from the form in the Rent PHP file, and store them in variables. I then extract the town and office from the office selected and compile and SQL statement to get the office details.

As you can see from Fig. 27, I then execute the SQL statement on the SQL server using searchResults function from common function PHP file, I then check for errors, if there is none I extract the officeCode from the array of results, and then invoke the getOfficeContact method which gets the email address of the office selected by the user. I then configure some PHPMailer settings for the email to be sent such as from email, subject and the message content, and then invoke the PHP require() function which adds the PHPMailer package to the SendMail PHP file, to allow me to invoke PHPMailer functions.

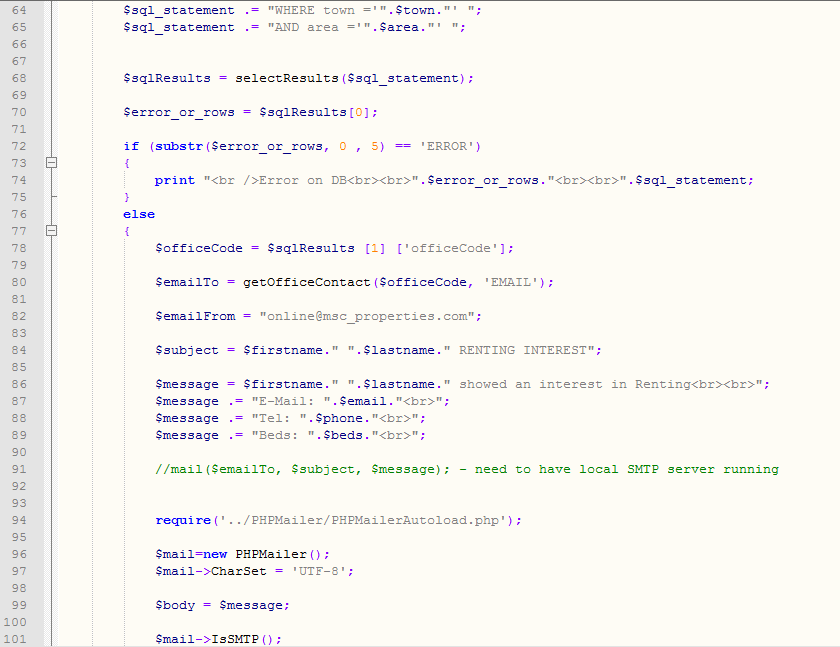


Fig. 27 – Extract from SendMail PHP file

I then create a new PHPMailer object and assign the object to a variable called mail. I then set a number of the PHPMailer object fields such as character set, if it is an SMTP connection, the mailing host, whether SMTP is secure, the port, SMTP debug value, if the SMTP connection is authenticated, the username and password combination, the from address, the reply to address, the subject, the actual message converted to HTML format and the to address. Once I have assigned these variables to the PHPMailier object, as shown in Fig, 28, I can then invoke the send() method, which will try to send the message created and print a response message.



Fig. 27 – Extract from SendMail PHP file