PHP PART ONE

XAMPP SET UP

PHP is a free download from php.net. However, along with PHP itself, you'll likely need a web server (such as Apache, Nginx or ISS) and then want to install a database such as MySQL. These installations can all be done individually but most developers will use a 'Stack solution' that will install all the bits 'n' pieces for you.

There are many development setups available that allow for the quick set up of a LAMP (Linux, Apache, MySQL, PHP) stack for development purposes. One of the most popular is XAMPP which stands for:

X – Can be installed on Windows, Mac or Linux operating system.

A – Apache, the web server technology.

M – MySQL, the relational database server.

P – PHP, the server-side scripting language.

P – phpMyAdmin, a web-based database administration tool used in conjunction with MySQL.

Other stack solutions include MAMP and WAMP Server, or alternatively there are containerization solutions with Docker.

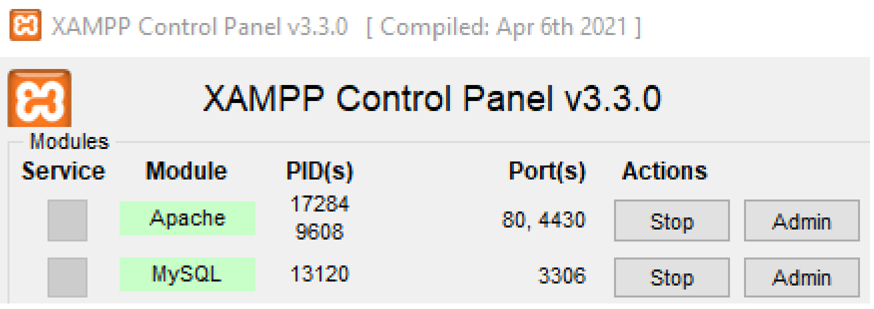
This lab requires the use of XAMPP. Review the video on the Blackboard site as to how to install XAMPP on a Windows machine. The video covers installation on the Windows 11, Virtual Machines.

If installing yourself, find the appropriate installation at:

https://www.apachefriends.org/

XAMPP CONTROL PANEL

When successfully installed and launched the XAMPP Control Panel will appear. Use this to stop/start the Apache server (the webserver which has PHP installed). The main reason to use a server-side technology like PHP, is that it enables the integration of a database with a web application. However, PHP does not provide the database itself this comes from a different database application. PHP will work with a range of databases, but the most common combination is to use it with the open-source MySQL database. In XAMPP, the M is the MySQL. It is the next item to switch on in the XAMPP control panel.



XAMPP Control Panel v3.3.0 [Compiled: Apr 6th 2021]

XAMPP Control Panel v3.3.0

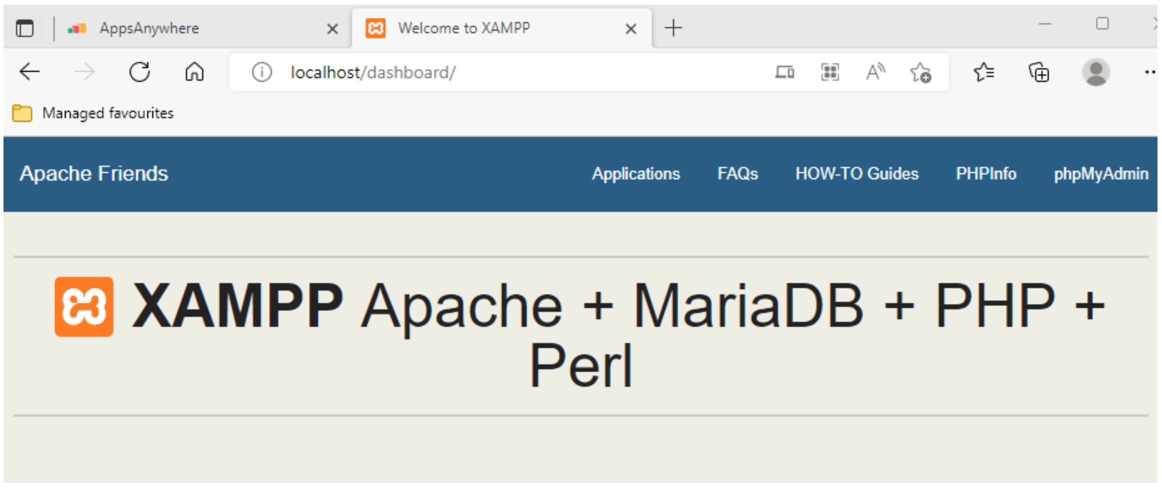
Modules

Service	Module	PID(s)	Port(s)	Actions	
	Apache	17284 9608	80, 4430	Stop	Admin
	MySQL	13120	3306	Stop	Admin

We can now test our application is running by opening a web browser and visiting:

http://localhost/

You should see the following:



AppsAnywhere | Welcome to XAMPP

localhost/dashboard/

Managed favourites

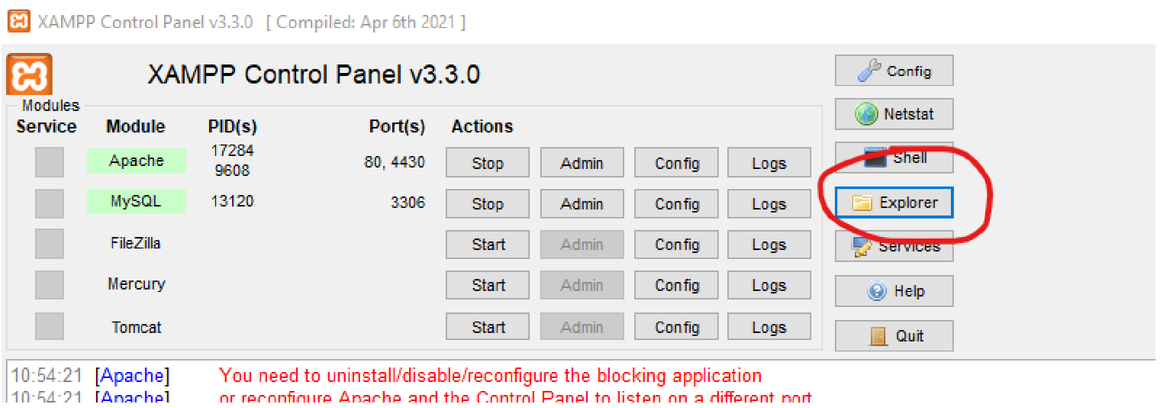
Apache Friends — Applications | FAQs | HOW-TO Guides | PHPInfo | phpMyAdmin

XAMPP Apache + MariaDB + PHP + Perl

This indicate that PHP is running as expected.

LOCATING THE FILES

In the XAMPP control panel select the File Explorer button:


XAMPP Control Panel v3.3.0 [Compiled: Apr 6th 2021]
XAMPP Control Panel v3.3.0
Config
Netstat
Shell
Explorer
Services
Help
Quit

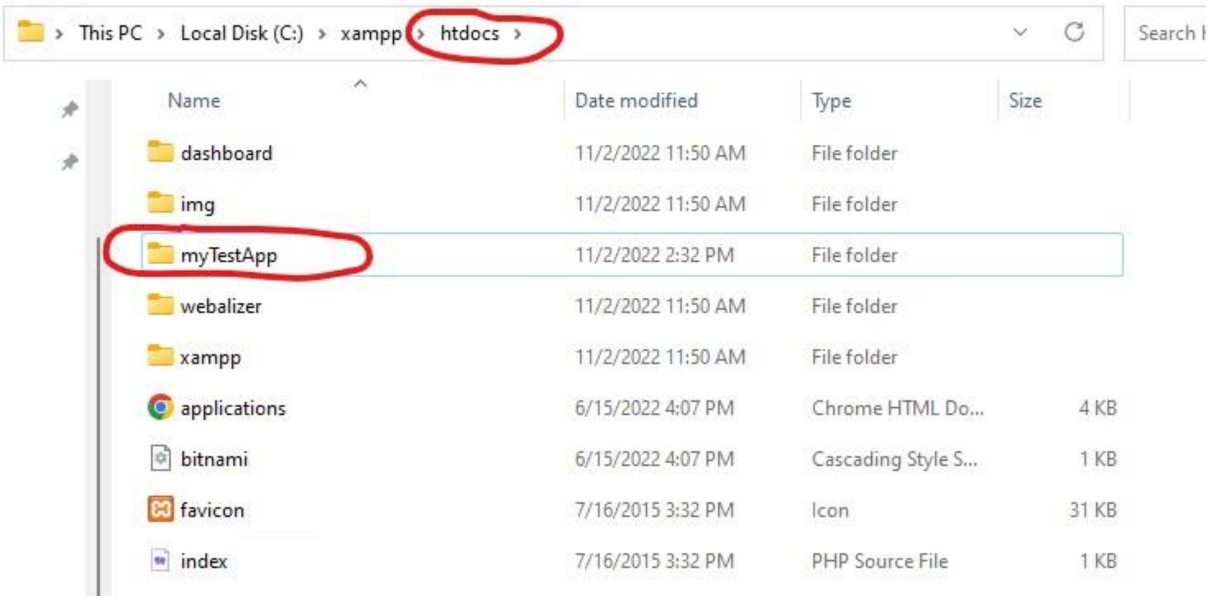
Service	Module	PID(s)	Port(s)	Actions			
	Apache	17284 9608	80, 4430	Stop	Admin	Config	Logs
	MySQL	13120	3306	Stop	Admin	Config	Logs
	FileZilla			Start	Admin	Config	Logs
	Mercury			Start	Admin	Config	Logs
	Tomcat			Start	Admin	Config	Logs

```
10:54:21  [Apache]     You need to uninstall/disable/reconfigure the blocking application
10:54:21  [Apache]     or reconfigure Apache and the Control Panel to listen on a different port
```

This will show us the location of the files used by XAMPP. XAMPP will have installed a range of folders and files:

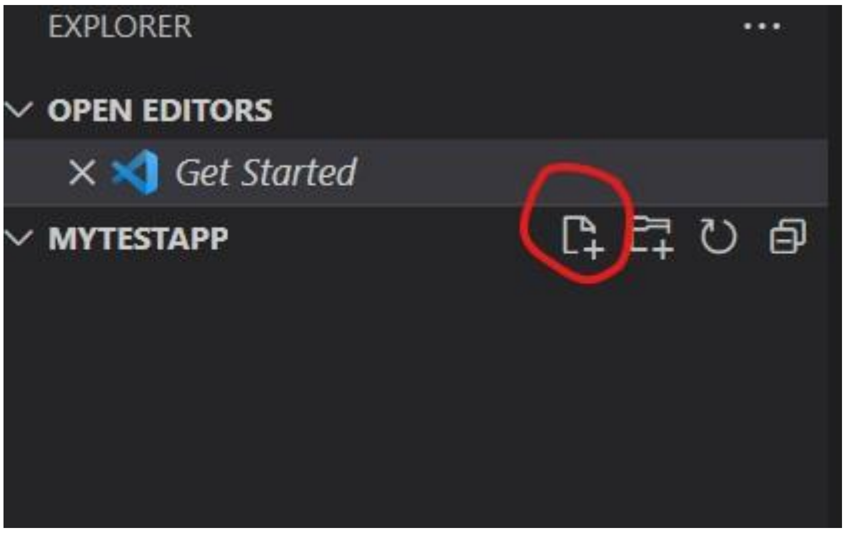
It is the *htdocs* folder that is of particular interest as this is the root of the webserver. The root folder of the web server is the point at which files are visible to users of the web server ie *localhost* is equal to *htdocs*.

To demonstrate how *htdocs* works create a new folder called *myTestApp*.

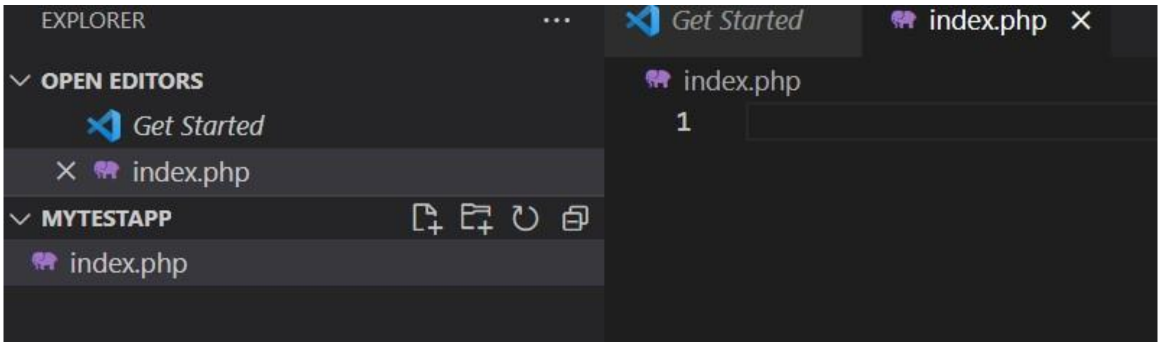


Name	Date modified	Type	Size
dashboard	11/2/2022 11:50 AM	File folder	
img	11/2/2022 11:50 AM	File folder	
myTestApp	11/2/2022 2:32 PM	File folder	
webalizer	11/2/2022 11:50 AM	File folder	
xampp	11/2/2022 11:50 AM	File folder	
applications	6/15/2022 4:07 PM	Chrome HTML Do...	4 KB
bitnami	6/15/2022 4:07 PM	Cascading Style S...	1 KB
favicon	7/16/2015 3:32 PM	Icon	31 KB
index	7/16/2015 3:32 PM	PHP Source File	1 KB

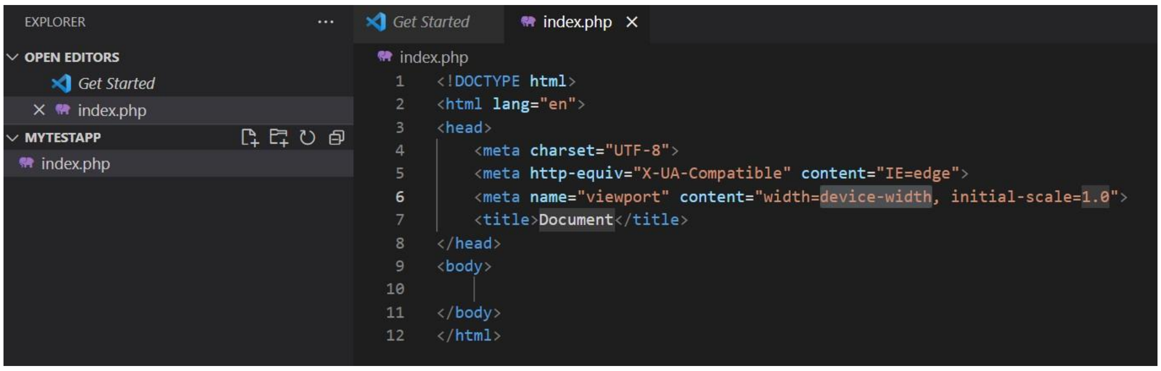
In Visual Studio Code and drag your newly created *myTestApp* folder into the editor. In Visual Studio Code create a new file called *index.php*.



Open it in the editor and it will appear blank.



In the editor type an exclamation mark (!) and then the TAB key. The skeleton of a HTML page is created for you (through a feature called Emmet):



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

</body>
</html>
```

Inside the `<body>` tag add some basic HTML ie:

```
<h1>Hello</h1>
```

Save your file (this removes the circle next to the file name) and view your page in the browser at:

http://localhost/myTestApp

You should see the "Hello" message.

ADDING PHP

To add PHP we use the `<?php …. ?>` syntax. Try editing your *index.php* as follows:

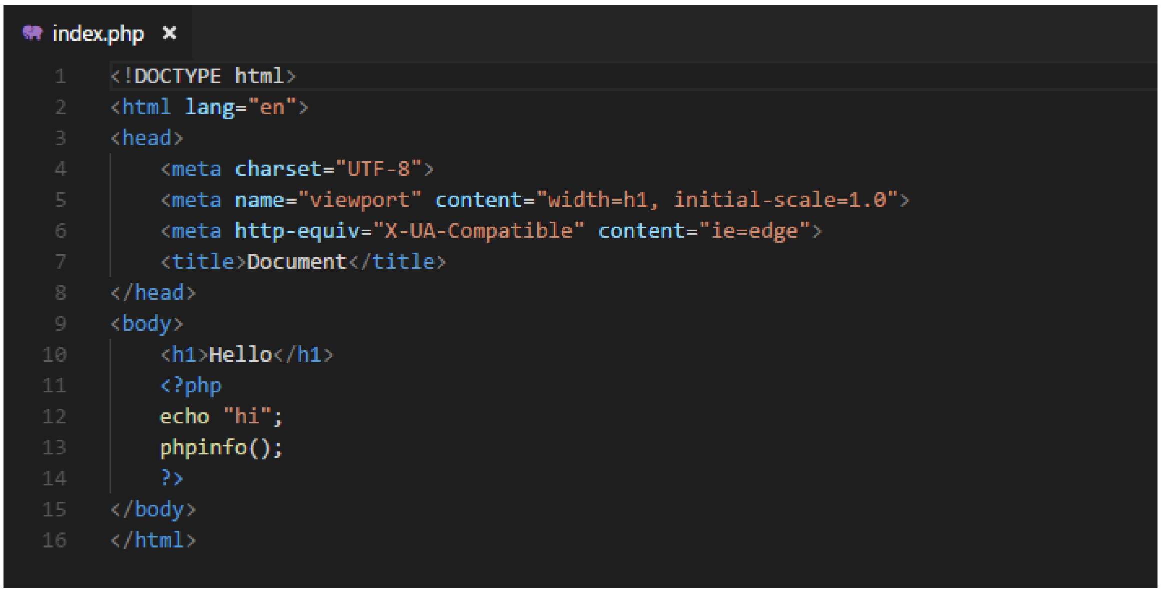
```
<?php

echo "hi";

?>
```

Save and test your file through the browser. This is just a basic example of how text can be written to the HTML page from PHP.

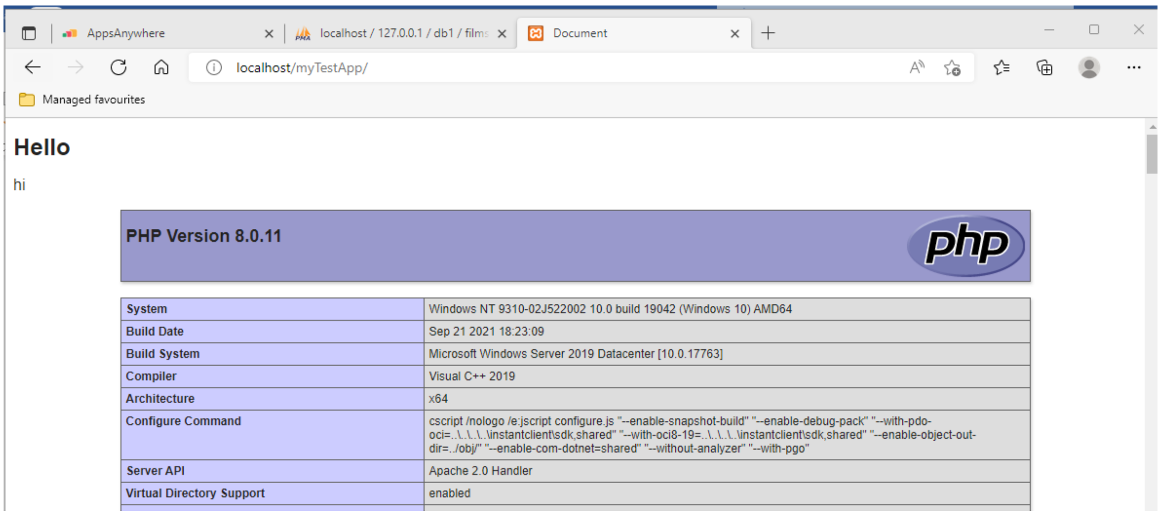
A final test would be to add the PHP method `phpinfo()` as below.



index.php

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=h1, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>
<body>
    <h1>Hello</h1>
    <?php
    echo "hi";
    phpinfo();
    ?>
</body>
</html>
```

Again, save and test your page in the browser and you should see the results of the `phpinfo()` method (which provides a breakdown of the PHP install):



Hello

hi

PHP Version 8.0.11

System	Windows NT 9310-02J522002 10.0 build 19042 (Windows 10) AMD64
Build Date	Sep 21 2021 18:23:09
Build System	Microsoft Windows Server 2019 Datacenter [10.0.17763]
Compiler	Visual C++ 2019
Architecture	x64
Configure Command	cscript /nologo /e:jscript configure.js "--enable-snapshot-build" "--enable-debug-pack" "--with-pdo-oci=..\..\..\..\instantclient\sdk,shared" "--with-oci8-19=..\..\..\..\instantclient\sdk,shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet=shared" "--without-analyzer" "--with-pgo"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled

Notice that we are using PHP Version 8.0.11. PHP 8 was released in November 2020, so this represents an up-to-date version of the software.

CLONE THE PROJECT FILES

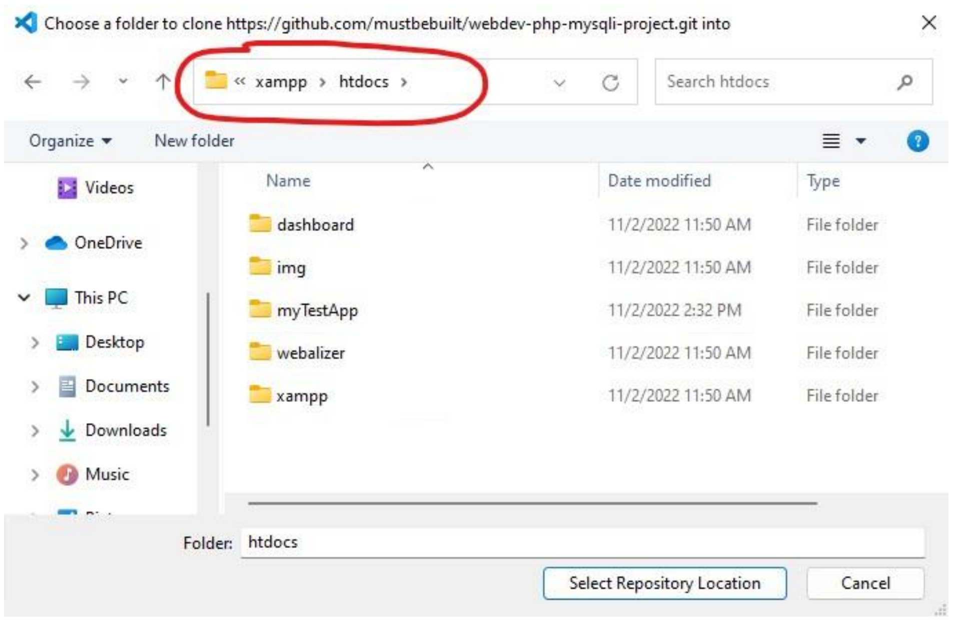
Close the folder from above and we'll work on a fuller project example to build a website around a films database.

Fork the Github repo found at:

https://github.com/mustbebuilt/webdev-php-mysqli-project

Clone this into Visual Studio Code. Be careful that you clone the repo into the *htdocs* folder so that files are placed within the Apache server (the 'A' of XAMPP).

For example, clone the repo into c:/xampp/htdocs/


Choose a folder to clone https://github.com/mustbebuilt/webdev-php-mysqli-project.git into

« xampp › htdocs ›

Name	Date modified	Type
dashboard	11/2/2022 11:50 AM	File folder
img	11/2/2022 11:50 AM	File folder
myTestApp	11/2/2022 2:32 PM	File folder
webalizer	11/2/2022 11:50 AM	File folder
xampp	11/2/2022 11:50 AM	File folder

Folder: htdocs

The project files should now be viewable in Visual Studio Code at the file location of:

```
"C:\xampp\htdocs\webdev-php-mysqli-project"
```

With XAMPP running you should be able to view the application through the apache webserver at:

```
http://localhost/webdev-php-mysqli-project/
```

WORKING WITH INCLUDES

Now we have a PHP supporting server we can make use of includes. Includes enable a snippet of code to be shared across multiple pages. This allows for a modular approach to the taken when putting pages together.

For example, our site should have a standard footer that appears on every page. We can create this as an include file, that just has the snippet of code needed for the footer, and then add it to a page where required.

It is good practice to group the include files together, thus the include folder in the project files. Open and review the file *includes/footer.php*.

```
<div class="footerContainer">
<footer>
      <nav>
        <menu>
          <li><a href="#">Terms</a></li>
          <li><a href="#">FAQ</a></li>
          <li><a href="#">Facebook</a></li>
          <li><a href="#">Twitter</a></li>
        </menu>
      </nav>
      <div>&copy;</div>
    </footer>
</div>
```

In the PHP files (*index.php, catalogue.php, contact.php, film-details.php, search.php, thankyou.php*) where prompted with a comment add:

```
<?php
include("includes/footer.php");
?>
```

As an include the *footer.php* file can now be edited centrally. We'll add the year used in the copyright statement automatically via PHP. Add the following to the top of the *includes/footer.php* file:

```
<?php
$year = date("y");
?>
```

This will create a date object in PHP that solely represents the current year.

Then amend the copyright statement with:

```
<div>&copy; <?php echo $year;?></div>
```

The year will now automatically appear at the bottom of each page.

Notice that the header information (the `<header>` and `<nav>`) has already been set up as an include of *include/header.php*. There is currently no link to the *contact.php* page. Add this to the include with:

```
<li><a href="contact.php">Contact Us</a></li>
```

When saved, all the files will now pick up the new link.

WORKING WITH $_GET AND $_POST

In the newly linked *contact.php* there is a HTML form. Notice that this form has a **method** of **post**. This sets the http method used to send the data from the form. The data from the form will be sent to the **action** of the form ie *thankyou.php*.

Open the *thankyou.php* page and add the PHP under `<section class="twoColumn">`:

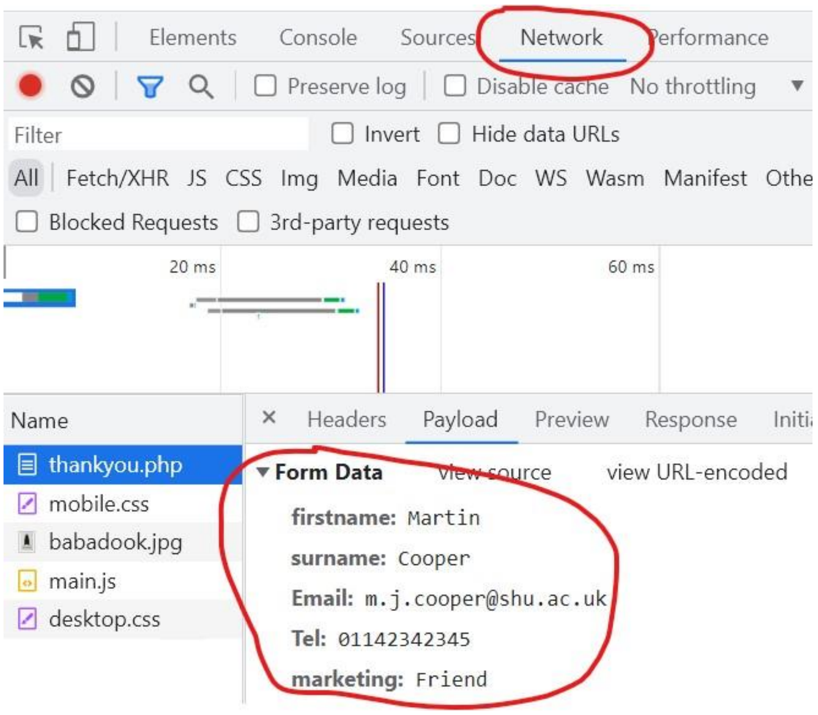
```
<?php
      echo "<div>";
      echo $_POST['firstname'];
      echo "</div>";
 ?>
```

This uses the `$_POST` PHP superglobal (an array of the values sent via the form). The value of *firstname* relates to the *name* set in the HTML form ie:

```
<div>
   <label for="firstname">First name</label>
   <input type="text" name="firstname" id="firstname" placeholder="Your First
Name"/>
</div>
```

Try adding other values from the form.

Notice these values are coming via the HTTP method POST. This can be seen in Google Chrome by looking at the Network tab under Payload and Form Data.



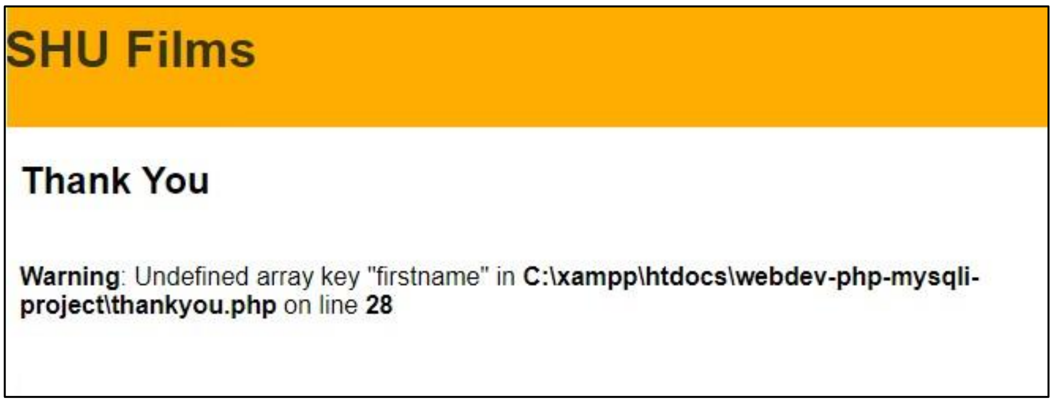
Form Data
firstname: Martin
surname: Cooper
Email: m.j.cooper@shu.ac.uk
Tel: 01142342345
marketing: Friend

Remember the names of these values are set in the HTML form using the *name* attribute.

In the *contact.php* try changing the form method to *get* ie:

```
<form action="thankyou.php" method="get">
```

Refresh and fill the contact form again and submit it. This time you will see an error:



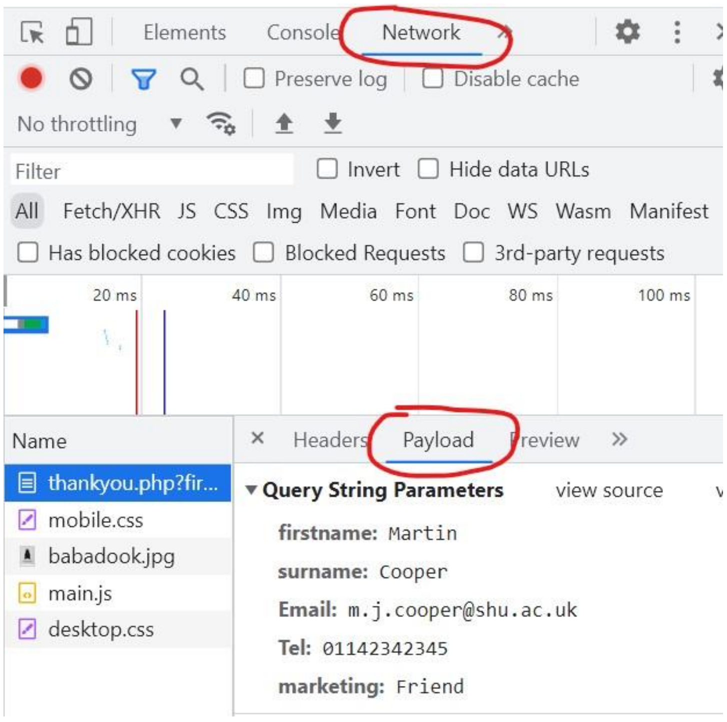
SHU Films

Thank You

Warning: Undefined array key "firstname" in **C:\xampp\htdocs\webdev-php-mysqli-project\thankyou.php** on line **28**

The data won't appear this time as the data is now sent via HTTP GET.

This can be seen in the Chrome network tab under Payload this time listed as query string values.



thankyou.php?fir...
mobile.css
babadook.jpg
main.js
desktop.css

Query String Parameters

firstname: Martin
surname: Cooper
Email: m.j.cooper@shu.ac.uk
Tel: 01142342345
marketing: Friend

To display this data in the *thankyou.php* page change the use of the `$_POST` superglobal to use `$_GET` instead. The `$_GET` superglobal looks in the payload to find the name/value pairs as query strings.

Notice as well that with `$_GET`, the values entered also appear in the URL ie:

```
http://localhost/webdev-php-mysqli-
project/thankyou.php?firstname=Martin&surname=Cooper&Email=m.j
.cooper%40shu.ac.uk&Tel=01142342345&marketing=Friend
```

Sending values this way is not very secure (don't use for sensitive data) but is perfectly acceptable for non-sensitive data. A classic example would be:

```
https://www.google.co.uk/search?q=cats
```

This technique is known as a query string where we have name/value pairs (above ***q*** is the ***name*** and ***cats*** is the ***value***).

Both `$_GET` and `$_POST` are arrays. As such they can be looped. Change the *contact.php* form method back to **post**. In the *thankyou.php* page, remove the previous code used to display the values and add the following:

```
<div>
      <table>
        <?php
     foreach($_POST as $key => $value){
       echo "<tr>";
       echo "<td class=\"uc\">";
       echo $key;
       echo "</td>";
       echo "<td>";
       echo $value;
       echo "</td>";
       echo "<tr>";
     }
     ?>

    </table>
      </div>
```

This code makes use of the fact that `$_POST` (and indeed `$_GET`) are arrays, as such we can loop around them to extract the values they contain.

Both `$_POST` and `$_GET` are associate arrays where the key is a string rather than a number. The key is the name of the field from the form, or the name of in the name/value pair of the query string.

These two methods for sending data from one page to another are important part of web development that you will see in all server side technologies.

In summary:

HTML forms with method of `post`, send values in the HTTP payload as form data and are accessed via `$_POST`.

HTML forms with method of `get`, send values in the HTTP payload as a query string and are accessed via `$_GET`.

Finally, note that with query string you can amend a link with `?` and then add name/values pairs as you see fit, as a way to send data between pages.

This technique will be used in the next lab to allow users to 'drill deeper' into the data ie:

```
http://localhost/webdev-php-mysqli-project/film-
details.php?filmID=6
```

… can be used to send a query string name/value pair of, filmID and '6'.