1. FORMULATE THE ISSUE:

**Summary of what my client requested**

As part of the Abide Youth page, one of the functions was to have a way to be able to view page content eg. The gallery page contains private photos, in a manner that is only accessible while logged in.

For that I needed;

1. A way to read from a database
2. Compare data from a login form to the database
3. Allow access page if user is logged in

(Completed)

Other required functions relating to 1.3 was that administrators needed to be able to add new users to the database without having to use any external program or software.

To achieve this:

1. I set up a create user form
2. I made it so that only admins could access the create user form (to prevent random people from setting themselves up and getting access to the private information).

(Completed)

The function that will utilize the database is the ability to display the list of all users and emails. To enable Administrators to easily check and contact youth members.

To achieve this:

1. Whether someone is signed up and can access the data on the page.
2. Query the database for the list of users and order by ID
3. Display in a Compact format

(Completed)

Lastly, to assist with adding contacts to the youth leaders the server not only has a user for each youth to access the web page but each user will have the email associated to their account allowing the server to send reminders by email the day before the upcoming event[‘s].

1. To do this I would read all the emails from the database.
2. Then format for the email for the email request then send off the email with data for the event.

With my plan laid out, I now take action and create the first iterations of this project.

2. GATHER DATA

When I created the `*adduser.php*` in *1.4, 1.7 1,8* there was an online form for adding new youth allowing the data to be efficiently entered into the database. When the form is submitted by the user, it will update the dynamic database to further include this new user.

This input form will consist of:

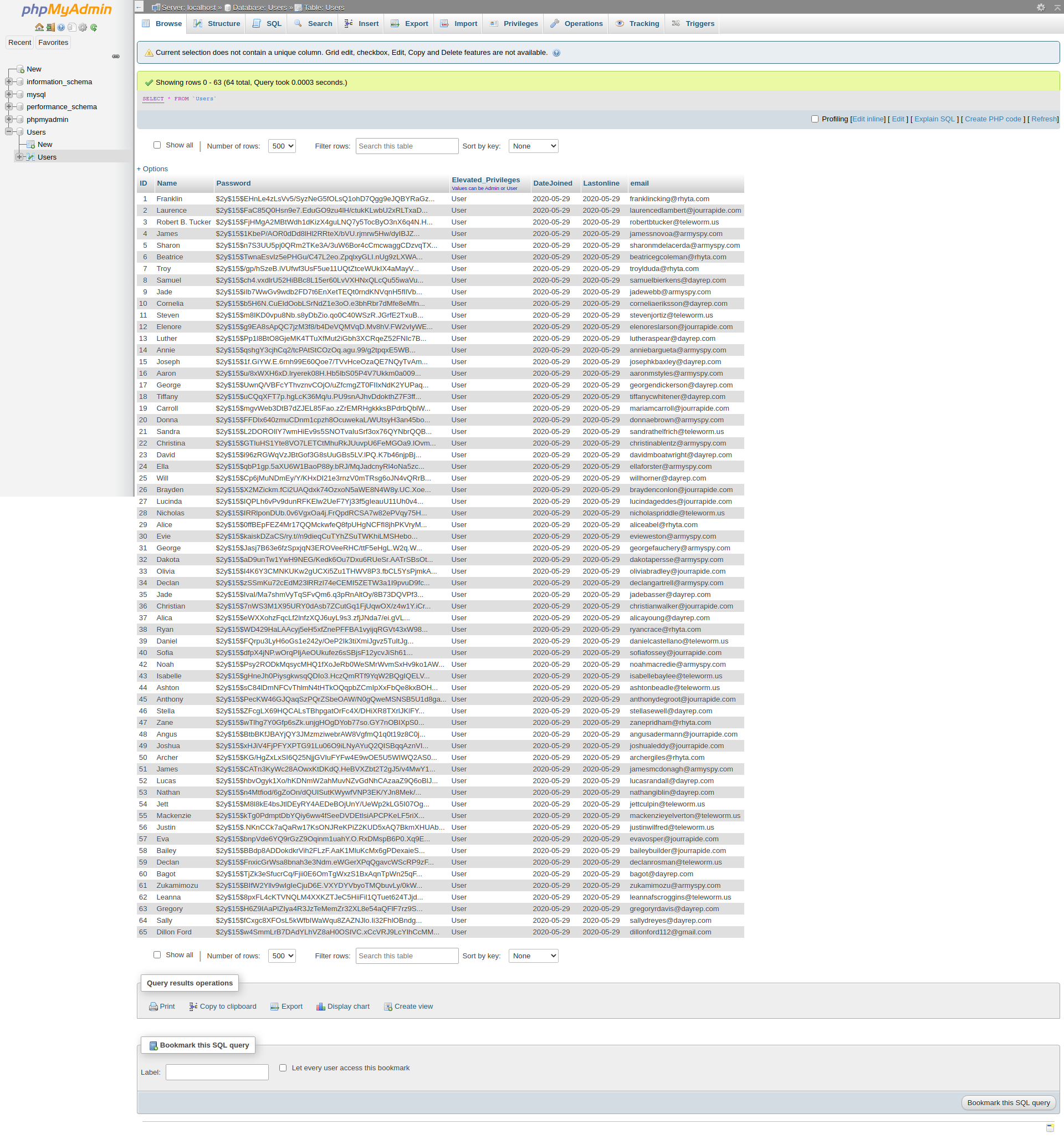
* Name:
* Email:
* Password:

The name and the password is needed to gain access to the restricted area of Abide Youth which is only for registered members.

The email is required for sending out notifications of events, and stuff that will be generally interested for everyone involved.

The process of gathering data for the full database is a matter of giving the signup link to the youth and in turn to create the following database. MYSQL is the choice of program.

After giving out the link to the signup page as a groups meet up everyone had an account with the abide youth website.   
  
To confirm that the data from everyone has been Successfully entered I checked on the server-side database portal. It can be seen 64 out of 64 entered were Successfully created



It can also be seen that from the only webportal that this is also the same data just with added filters to make the data easier to read to the human eye.



3. STRUCTURE: - DEVELOP A DATABASE

|  |
| --- |
| CREATE TABLE `Users`.`Users` ( `ID` INT NOT NULL AUTO\_INCREMENT , `Name` TEXT NOT NULL , `Password` TEXT NOT NULL , `Elevated\_Privileges` TEXT NOT NULL COMMENT 'Values can be Admin or User' ,`DateJoined` date NOT NULL,`Lastonline` Date NOT NULL,`email` TEXT NOT NULL, INDEX `ID` (`ID`)) ENGINE = InnoDB; |

^  
Final version of code to generate table for database

4. QUERY

Throughout this process a test sample of [oliver, josh] was used and then later expanding to the whole youth group.

5. PRESENTATION

See webpage/demonstrated video

6. IMPLICATIONS in this project

#### ● Cultural

There was a consideration in making versions with an alternative language[s]. In the end it was decided they could use the built-in translator in chrome or another browser they use.

#### ● Legal

To cover the legal side, I have primarily used open-source libraries within this project as well as leaving comments within the code giving credits to the creators.

#### ● Intellectual property

It is all my work, unless specified otherwise, as all work should be credited to their rightful owners.

#### ● Privacy

Personal details and passwords are normally the users biggest concerns when online. They are stored in a secure database private server with irreversible hashed passwords. Even if someone was to gain access somehow to this database, the hashed passwords would be useless to the hacker.

#### ● Accessibility

Range of users

Anyone with access to the interweb had the potential to view the web page but only people with an account can view any pages that contain sensitive information such as the location of where they are or private images, this links in with the implication of privacy.

#### ● Usability

The layout is made to be in line with the heuristics of external consistency and standards. This will help with proficiency so that new users can pick up the system with more ease.

#### ● Functionality

* See video of the functionality
* Adding User using SQL with input sanitization SQL, HTML and JS Inject
* Logging with input sanitization SQL, HTML and JS Inject
* Tracking of when the user last logged into the website
* Storing of emails
* MYSQL Database using PHPMyAdmin
* Admin only displays users in alphabetical order online on a webpage
* The adaptive display system for what people can see depends on viewing permissions linked to the user's account.

#### ● Aesthetics

Minimalistic and intuitive system design is setpot be used by all ages though primarily with teenagers What is the main Target in also links in with accessibility. This is so that the web page is not constricting of what the user can see and do do while maintaining functionality and Simplicity of layout.

#### ● Sustainability and Future Proofing

As for the future, one can set up the server on a site with Phpmyadmin, Apache2 installed and they will be in business as that is the only requirement to set up.

#### ● End-user Considerations

People’s passwords are hashed with blowfish so that they’re private passwords are kept safe From prying eyes. As the general security of data is a big thing these days, therefore this page is doing its part to protect people's data.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Version 1 (adding data)**

The code for this can be found in version 1 folder

This version consisted of me working out how to input data into SQL databases. The basic concept of this is as follows.

* I created a form with HTML
* I posted the data to a PHP page connected to an SQL database
* Using the SQL query called ‘*insert*’ This inputs the data into the desired database. For this to work I needed to make sure I knew the order of the columns prior so that I could input the data in the correct order.

Issues with **Version 1**

* It does not sanitize data input
* Input passwords are not hashed
* Code was in a messy format, therefore it hard to read
* There was no method to read data from a database

**Version 1.1 (Improved code Layout)**

The code for this can be found in version 1.1 folder

This version consisted of me refining the previous iteration so that the code was clearly laid out.

The changes of this layout, compared to the last one, are relatively small but make a big difference in the big picture of trying to debug this file.

Issues with **Version 1.1**

* Does not sanitize data input
* Input passwords are not hashed
* No method to read data from a database

**Version 2 (hashing and verification)**

The code for this can be found in version 2 folder

This version contains:

* A cost calculator for hashing with bcrypt. This is found in the file *`Cost4HashingwithBCRYPT.php`*
* A PHP script for testing of the password\_hash() This password hash function utilizes the cost calculated from the previous script and then takes an input from the test form, next I encrypted the string and compared it using password\_verify('password', $passwordhashed) What this does is compare the unhash string to the hash string and will return true or false depending on whether the string is matching or not. This is found in the file `*password\_check.php*`
* *`check.php`* is the predecessor to the *`verify.php`. `check.php`* contains my first fully functional login validation page. This page has the capability of reading, checking and also putting a password against the database and making sure that the password aligns with a user. It displays a whole bunch of information for debugging as well.

Issues with **Version 2**

* Does not sanitize data input
* Designed for use of developer-only (due to having debugging enabled)

**Version 2.1 (User list)**

The code for this can be found in version 2.1 folder

This versions purpose is to have a strong starting foundation for building off the `*Userlist.php*`

The main point of this was to have a starting place for just the functional code before adding styling to the page.

Issues with **Version 2.1**

* No styling for the page layout
* Lacks navbar
* No data output filter

**Version 3 (Integration)**

The code for this can be found in version 3 folder

`*loclaread .php*` was renamed to `*Userlist.php*` and now is styled, with navigation bar.

Steps for integration:

* Made so only the site admin for Abide Youth could access the administration page
* Tidy up the layout up
* Make this page read the user permission level (user, Admin)

This is used later in *`verify.php`* to get a $\_Session variable that will be used in providing a dynamic nav bar

* Integrated fully into Abide Youth web application

Now that SQL is integrated into the rest of the web application, there are several functions that need to be implemented rather than just listing the users .

* the ability to login and then compare the information from the login form against the database. (was successfully implemented.)
* The ability to add new users and hash the passwords before continuing to insert the data into the mysql database.

Log

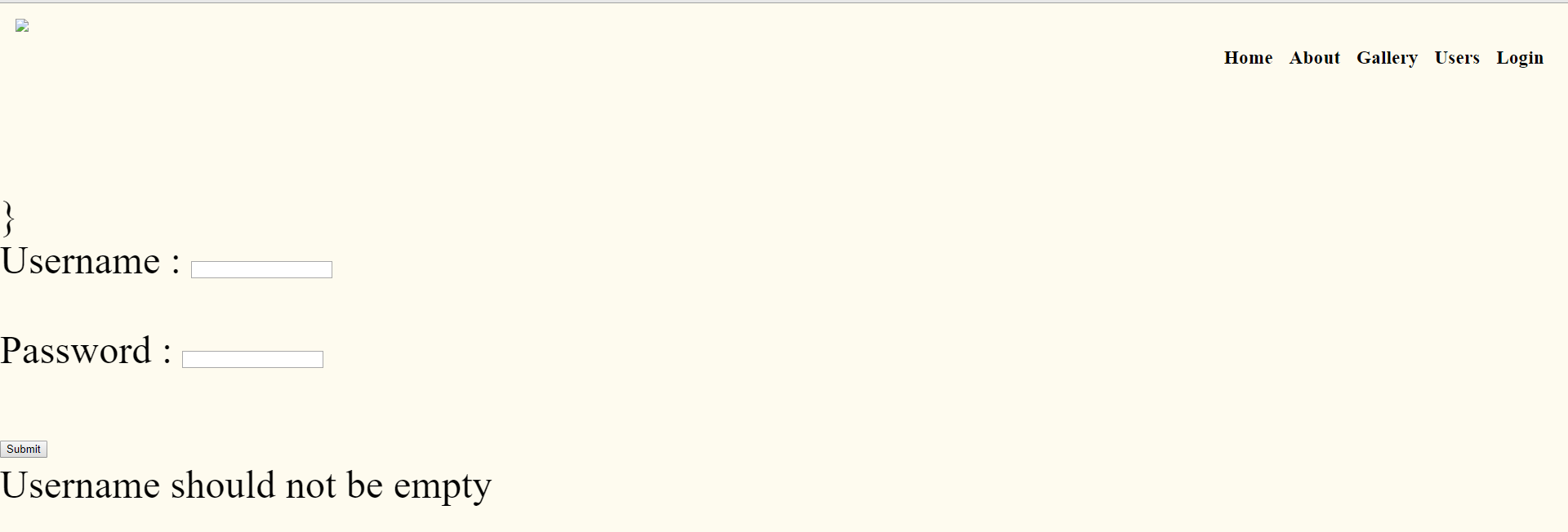
Hashing for the password storage on the database

*Getting the passwords from the database*

The storage of passwords for users are securely stored with Blowfish (*bcrypt*) hashing. That is a type of local SQL database (for the purpose of handing in this assignment I have left the password blank). Before my final version makes its way to the client this password will be changed to further prevent access to the data.

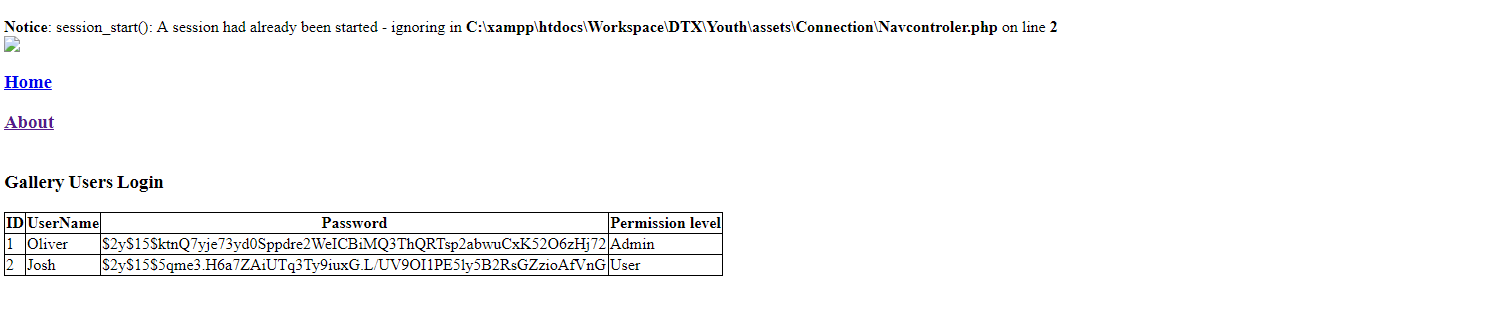
I have added one other admin only page which is for adding new users. It was a simple matter of copying the code from the list user page and pasting the code in.

The result looks as below;



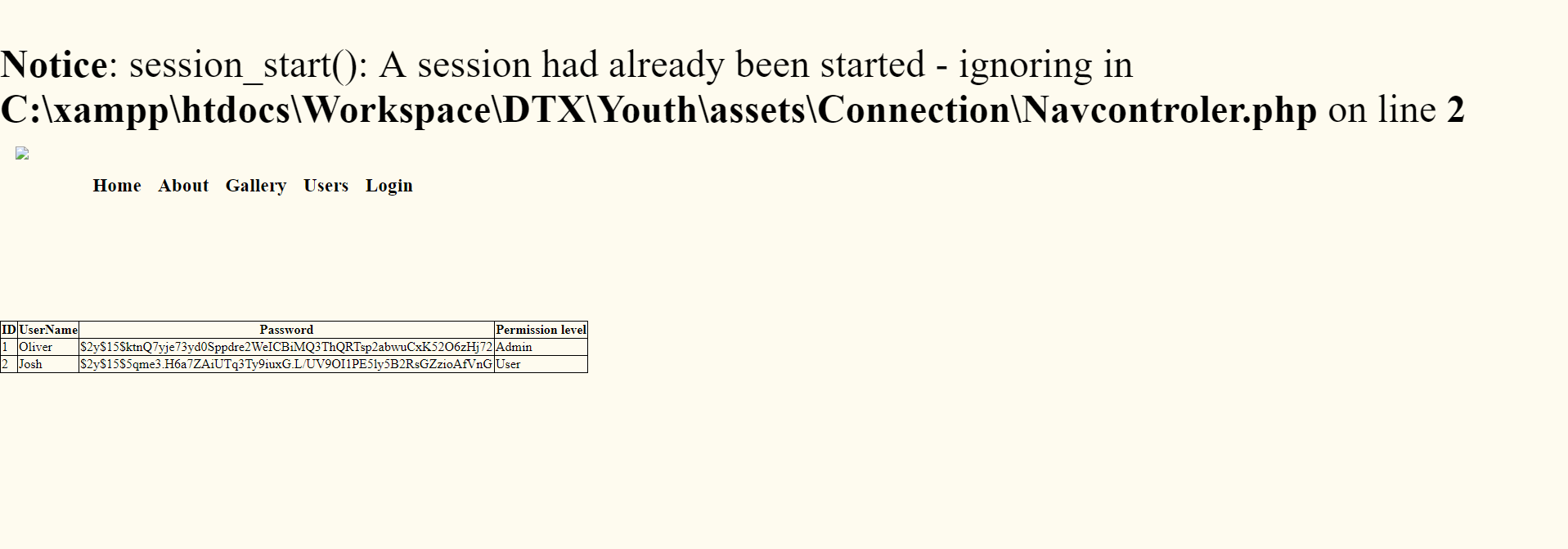
This page aids youth leaders in adding new users to the SQL database .

|  |
| --- |
| These admin pages have a priority of function over form. I could spend more time developing them and making them look really nice and stylish. The amount of time that would be spent using these pages is really minimal and would not warrant putting time into styling - especially as it is intended that I will be the primary admin in charge of the webpage. These admin pages are just there to assist with ease of management. Having the ability to segregate part of the website to only admins has allowed me to develop some more powerful functions without the need to worry about the admin user on this page abusing the power. I've restricted the ability to make new users to the admin permission role only. These tools also assist in making these new users by automatically formatting the information given to a format that can be inputted into a SQL database. This also allows ease of viewing information for the database and hopefully making it easier for someone else down the line to pick up the maintenance for this web site without necessarily knowing how SQL database work.  **Final outcome**    **WireFrame outcome** |

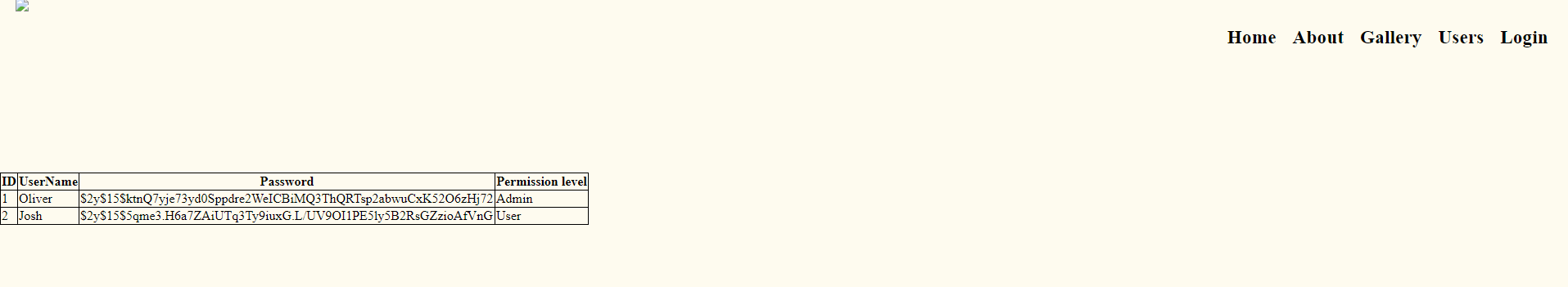


The navbar looks terrible so I am going to connect this page to my CSS page.

This was done by echoing out a reference script which tells the PHP file to add the CSS to HTML .The code for this looks like just the simple echo Echo '<link rel="stylesheet" type="text/css" href="../Styles.css">';'

I also had to deal with the fact that the dropdowns in the navigation bar did not initially work due to not having the required code. Actually it was in the CSS file but rather it had to be in the main file that is rendering the result. This was just a matter of simply copy and pasting the code from my main home file across resulting in an actual usable result.

The layout still was not correct, so I added a custom CSS property that only applies to this page and pushes the navbar across to the right. (also to add a piece of code which equalises the error message shown above by telling the server that this error is not important and to ignore use error\_reporting('none');)

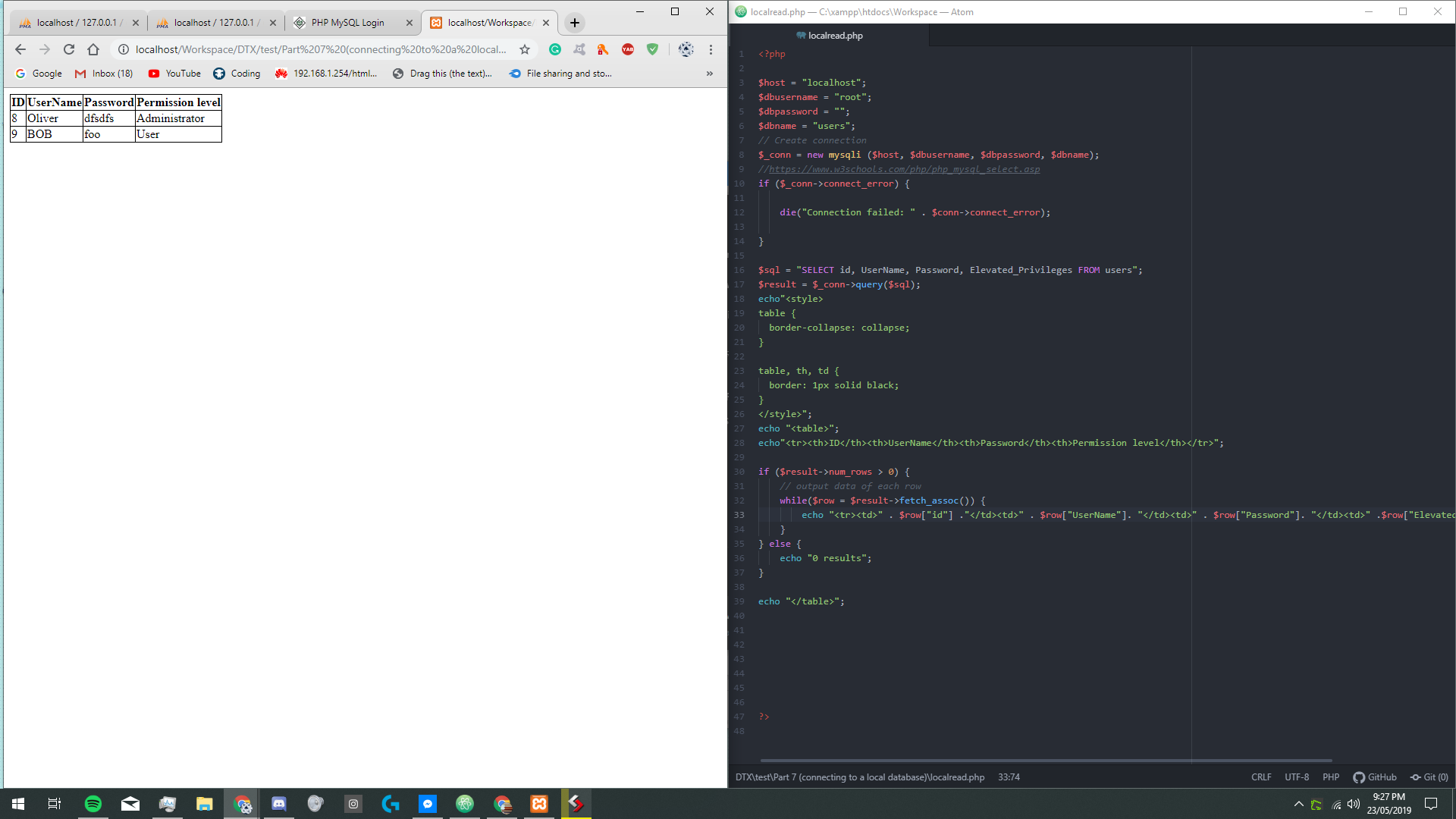


*Userlist.php* displays information about the current youth who are within the database as shown above

To start the login validation, I checked that I could read from the local database.

$host = "localhost";  
$dbusername = "root";  
$dbpassword = "";  
$dbname = "users";  
// Create connection  
$\_conn = new mysqli ($host, $dbusername, $dbpassword, $dbname);  
//https://www.w3schools.com/php/php\_mysql\_select.asp  
if ($\_conn->connect\_error) {  
  
 die("Connection failed: " . $conn->connect\_error);  
  
}  
//See localread.php

Once I proved that I could connect to a host local database, hosted with xampp, I formatted the data into a table layout. This involved repositioning the data along columns within a table.

****

This is using pseudo table data.

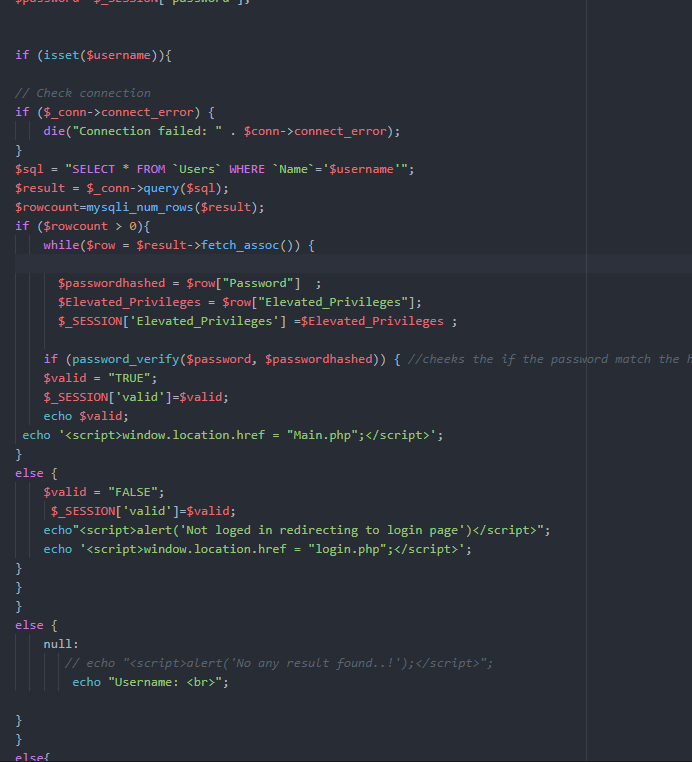
Also using the ORDER BY `Column name`

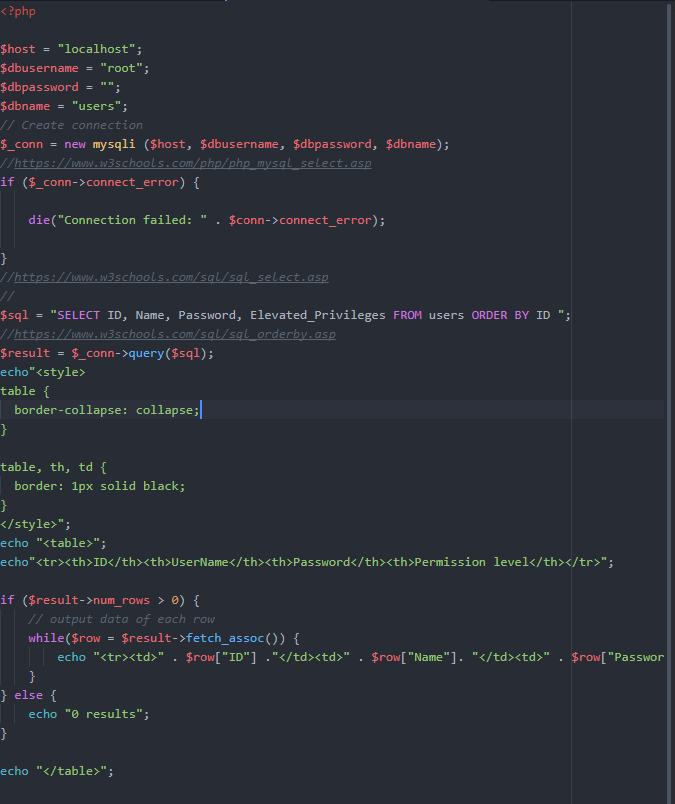
I was able to choose what order the data is laid out in .

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**19/05/27**

Added password and username verification to login .

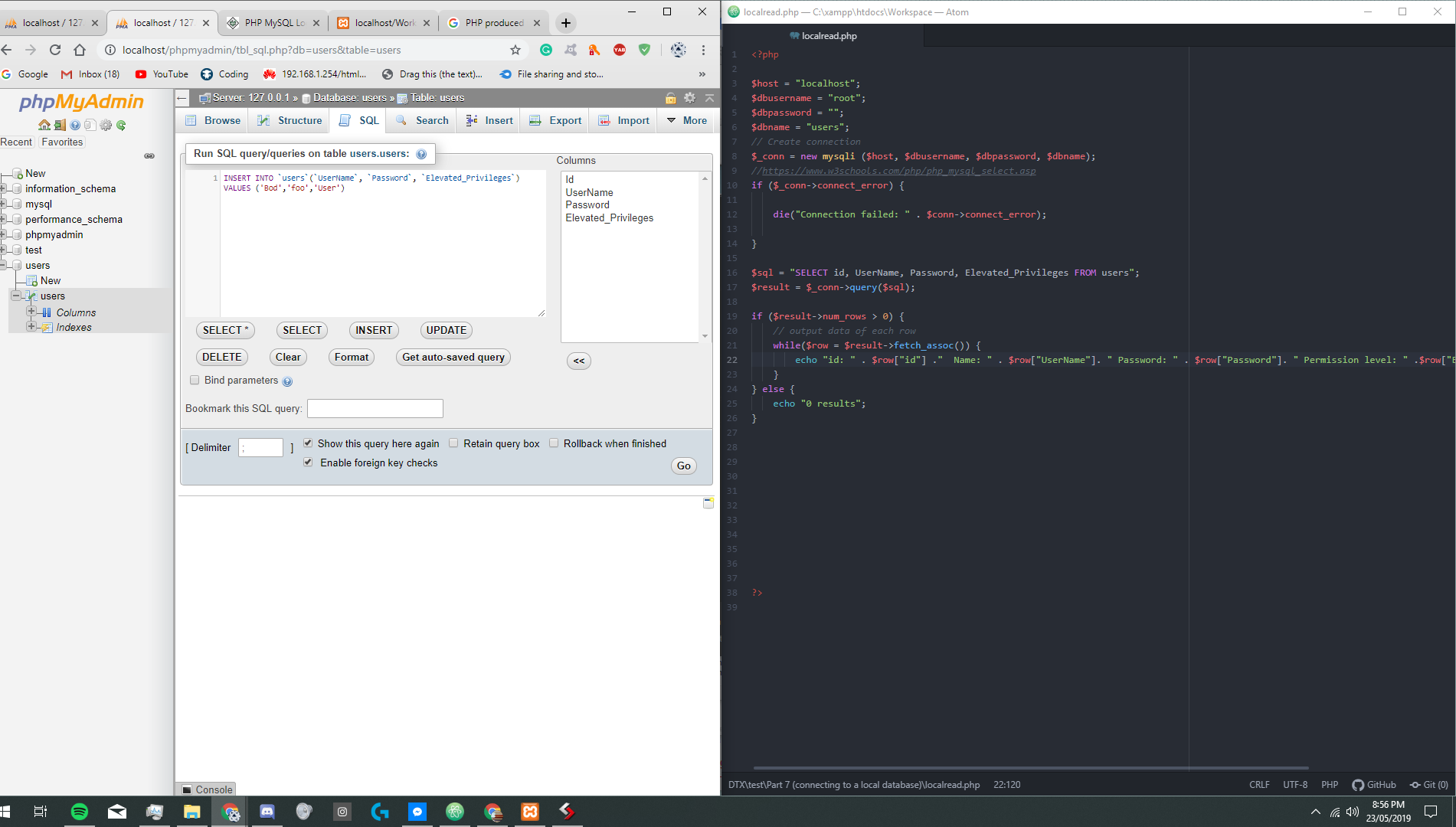




Adapted code from the original verify PHP code from term 1 when I was initially creating code snippets for PHP. It is so that it would be compatible with my current database and display.

Component 3: LOGGING-IN validation

I want the log-in validation process to be user-friendly and for inputted data from the log-in page to be matched with the SQL database. Set the php session equal to `valid` for pages such as the gallery page to compare and check if user is logged in.



Finishing setting up phpmyadmin.

Testing local reading of data and writing of data

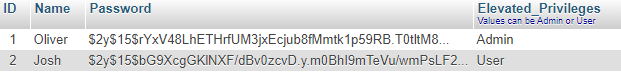
Localhost PHPMyAdmin using Xampp.

Issues with **Version 3**

* No data output filter

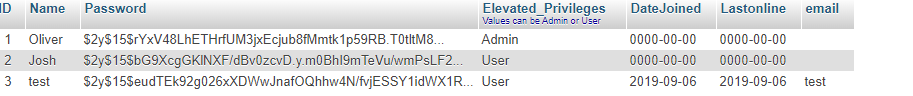
**Version 4 (Expansion)**

The code for this can be found in version 4 folder



This approach only deals with one of my requested features which is the ability for youth to login and then give them access to restricted pages.

But to be able to deal with the expansion for the addition of a mail list /date joined/when they last logged in.



Once these additional columns are added to the database the ability to store when the user account was created and last used will be completed and successful.

When the user account is created it is just a simple matter of getting the current date in an SQL friendly format (Year/Month/Day)

$datejoined = new DateTime();  
 $datejoined = $datejoined->format("Y-m-d");

Inspired from

https://tecadmin.net/get-current-date-and-time-in-php/

Parts to be built or improved/Issues from this iteration are:

* Making sure the data is formatted correctly before anything is imported into the database *(****HTML Form sanitization****)*
* Setting up a password confirm box so that encrypt password they're not accidentally submitted to database
* Check the emails are being submitted follow the correct email format

.

**Version 5 (HTML Form sanitization)**

The code for this can be found in version 5 folder

With all the correct data input setup I approached the task of initializing sanitization to clean up the data inputted from the forms completed by users.

The two places that would require input of the forms being cleaned are

* When creating a new user on `*Adduser.php*`
* Login form within the navigation bar.

The first issue I have decided to tackle is when creating a new user, is the possibility for code to potentially be injected through this form. As well as a duplicate user being created.

Fixing Inspiration:

<https://www.esecurityplanet.com/browser-security/prevent-web-attacks-using-input-sanitization.html>

<https://www.w3schools.com/php/php_ref_filter.asp>

<https://www.php.net/manual/en/filter.filters.php>

<https://www.w3schools.com/php/php_filter.asp>

<https://www.php.net/manual/en/filter.filters.sanitize.php>

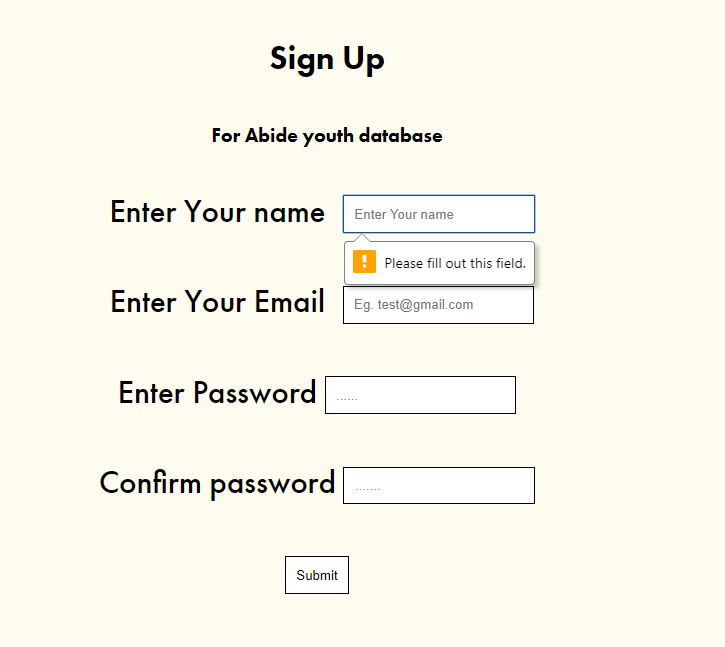
<https://www.ionos.com/community/hosting/php/basic-php-security/>

Armed with this knowledge, I took my first step in fixing the issue of scripts being able to be injected through the sign-up & login forms.

Taking a look at the signup form, I can see there there are 4 inputs:

* Name
* Email
* Password
* Confirm password

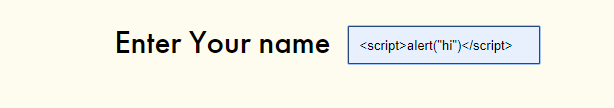
2/4 none of these details are left out when signing up, as I have added a required tag onto the <input> tag, resulting in an error being displayed if the user tries to complete the signup without filling those fields in.



Now with all data inputs being compulsory, I sanitise the name input with:

$username = filter\_var($username, FILTER\_SANITIZE\_STRING);

With the potentially malicious script of ⬇



FILTER\_SANITIZE\_STRING) is a method of [character escaping](https://www.w3.org/International/questions/qa-escapes) all the potentially malicious characters to ***ascii***(abbreviated from American Standard Code for Information Interchange)



This is what is needed for preventing any <script> tags from executing any malicious code, and also completely removes all **<** and **>** In contrast to htmlspecialchars() which would also just convert The <> Into ***ascii*** but what about the case where the code is meant to execute on the database that is being inserted into .

mysql\_real\_escape\_string() after first trying to use this to counteract a sql-injection i had no success.

So with further research the same desired outcome to be achieved with prepared statements within MySQL. This essentially means that everything is prepared and just requires an input. It automatically still has them but removes dangerous/malicious content (Relative to SQL Injection) Once again, the supposedly would have the desired outcome but I was unable to get the function to execute. Fortunately throughout my research of how to do the prepared statements for SQL and PHP I stumble across mysqli\_real\_escape\_string()

One letter does make an important difference from what I was trying to achieve earlier. The only difference being that this requires an active connection. That was not hard to achieve due to me already having the required code infrastructure setup.

With all malicious characters now removed from a user input, I now move onto cover the email input within the HTML form. I have already pre-defined the type for email meaning that it will only accept correctly structured data eg. [XXXXXXX@XXXX.XXX](mailto:XXXXXXX@XXXX.XXX) Meaning that the inputted data is already in that correct format ready for the following code. This prevents your average joe user from inputting an invalid email.

This does not prevent if somehow the data is imported from the server end manually, it will still filter it out using FILTER\_SANITIZE\_EMAIL to

|  |
| --- |
| **“**Remove all characters except letters, digits and *!#$%&'\*+-=?^\_`{|}~@.[]*.**”**php.net |

Then to be double safe, I also ran it through my filter for removing dangerous SQL characters . One final thing that was required to make these emails friendly for further purpose of being compared to an input is I used strtolower() To remove any differences with capitalisation. For example some entered *SUPERAWESOMEEMAIL@AWESOMEST.GEONET* It would be converted to *superawesomeemail@awesomest.geonet*

The last method of sanitisation I am using is for password input. The only sanitisation of the data is the Blowfish hash process that it was already run through. It will render any malicious input useless anyway.

The only other input that I have within the Abide Youth page is the drop down login box located in the navbar as I will be sanitizing the user input data with SQL and HTML Sanitisation.

What does each sanitizing process include?

* SQL sanitisation: consists of putting a **\** in front of any potentially malicious characters To SQL databases
* HTML sanitisation: consists of removing element tags eg <script> and more

Again, there is no need to do any sanitisation on a password due to the contents of the string being hash to a point where any imported tags or SQL queries would be rendered useless the hashing process

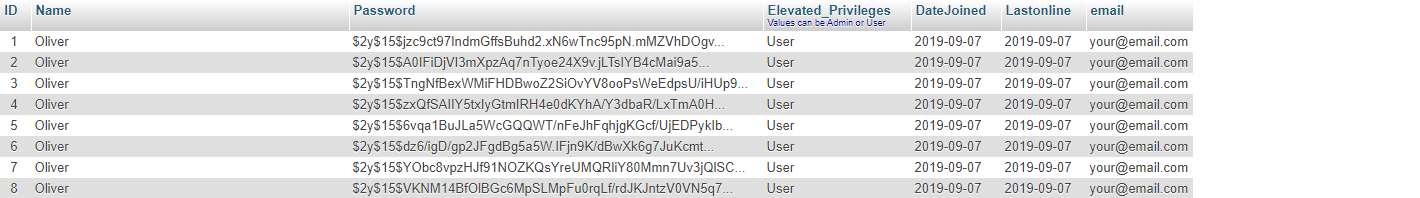
Issues with **Version 5**

* Does not protect against duplicate In Data entries when creating a new user

**Version 6 (duplicate data prevention )**

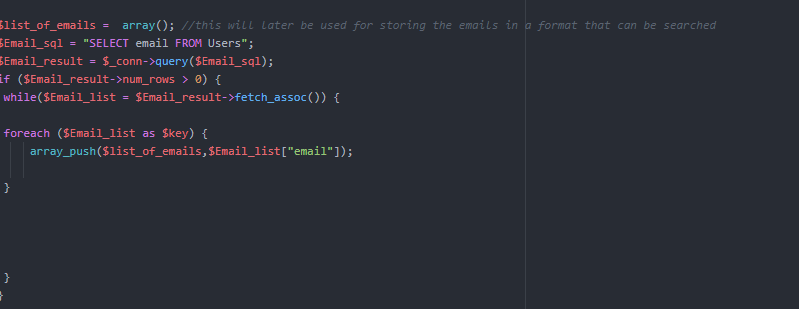
The code for this can be found in version 6 folder

With **version 5,** there was the issue when adding new users that the person inputting could be incompetent user and SOMEHOW enter duplicate information.



To validate this data, I will be comparing the entered email to all the emails stores in the database, and if the entered email matches any of the emails in the database. Then my planned code will return an error.

The first stage in preparing to remove duplication is to formulate a query with [SELECT](https://abideyouth.cf/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/select.html) `email` FROM `Users`.

This query gets a list of all the emails in the system and then I process the data into an array which is easier for me to search and compare.

With this I am now able to do a direct search.I can compare whether the entered email is equal to any of the emails inside the database . Any differences in capitalisation was fixed when the initial sanitisation was completed. The general rule for whether to trust or not trust data is that if you didn't enter it don't trust it until put on a verified format and sanitised and even then treated with caution.

With all the form inputs now being sanitised I have chosen to move on to the task of tracking when the user was last online.

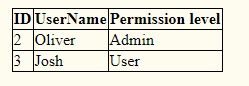
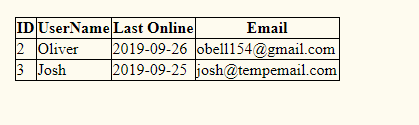
This will allow me to analyse the use of this web page allowing me later on to make and formulate statistical data about users.



The query above is run anytime a user logs on and updates the users last online time to the current date.

The next update for the user info page displays more useful data about registered users for page admins.

Old Layout | New layout

|

Giving the admins greater information on current users

Issues with **Version 6**

* Does send out emails
* Auto logout

**Version 7 (email setup)**

The code for this can be found in version 7 folder

But what use is it having people's email if they are not going to be used.

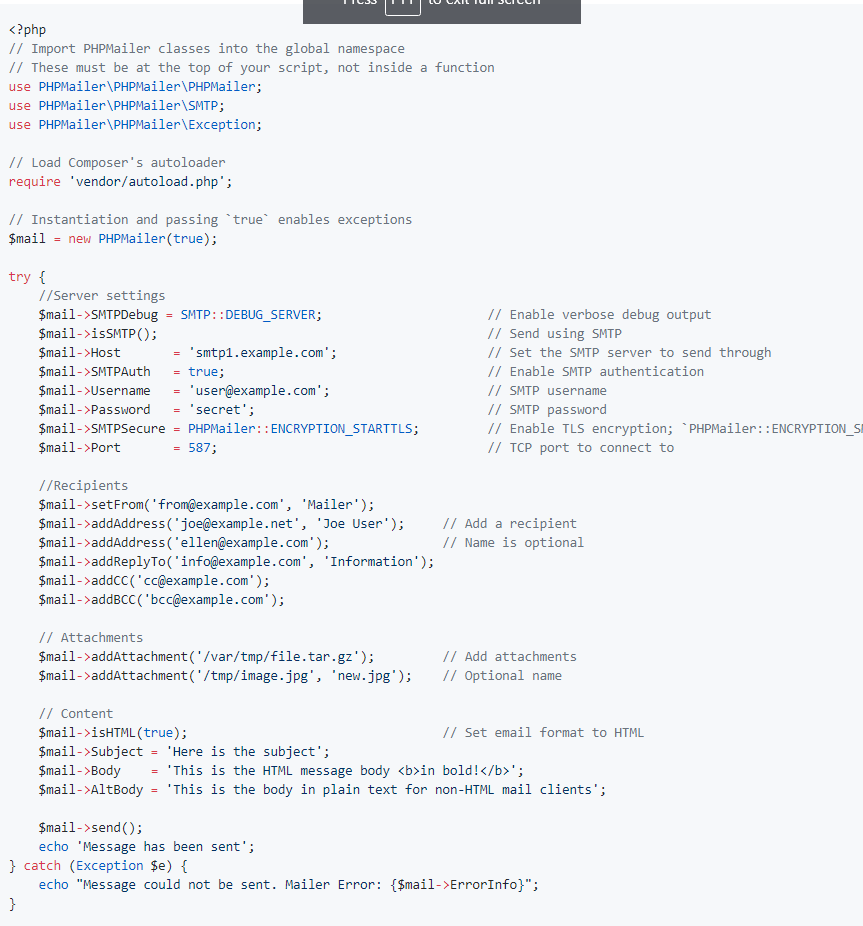
Therefore they are used to send out a reminder on the day before the event happens.

To send emails with php I could use the inbuilt mail() function, but that is very low level as you can’t send images or text formatting. After doing some research but I came across a PHP library called PHPmailer

<https://github.com/PHPMailer/PHPMailer>

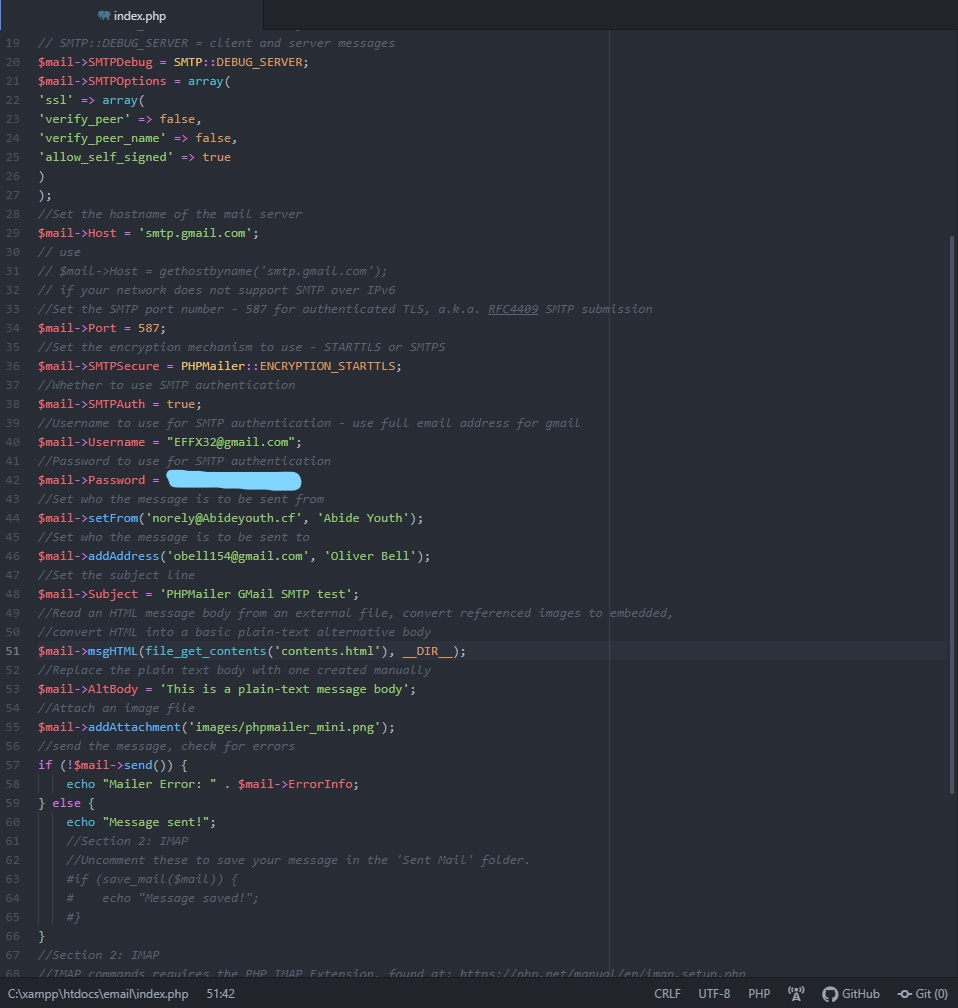
Which can use a SMTP internet based protocol to send html styled based emails

Code given on github repository:



I found that there was some bug in the code but after some research I found the fix on <https://pepipost.com/tutorials/phpmailer-smtp-error-could-not-connect-to-smtp-host/>

All what was need to be added was more SMTP settings.



Now with this updated code I was able to send email from php .

Issues with **Version 7:**

* The next stage is to make it so that the email will be sent out before the next upcoming event.

**Version 8 (Email integration)**

The code for this can be found in version 8 folder

This version approaches the issue of the last version. The last version was just a theory test of sending emails using SMTP.

The full implementation will consist of

(layout plan )

// get email list

// get next upcoming event from .txt file

//

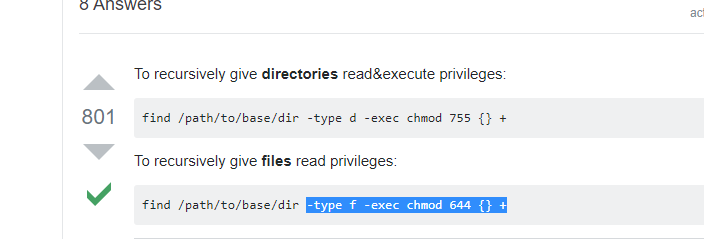
// add each email to Bcc

// Send email

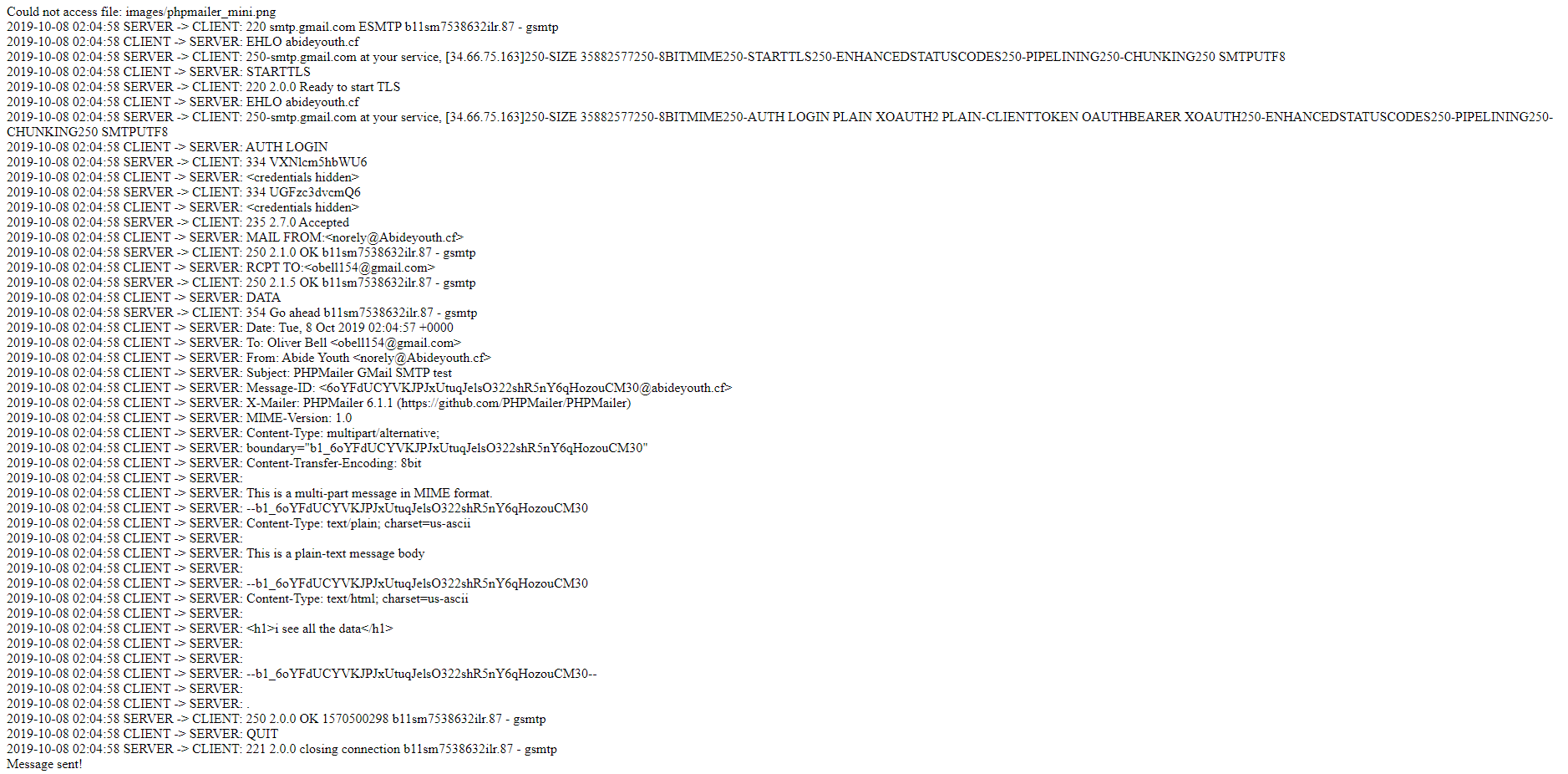
// update when this php file is to be next executed by editing the cron job

<https://superuser.com/questions/91935/how-to-recursively-chmod-all-directories-except-files>

When initially trying to execute any code in the new folder labelled email return the error pcfg\_openfile: unable to check htaccess file. This was due to files having incorrect permissions of 600 when they needed to have 644 and directories have 755.

To fix this I used 

Success, now that I am able to send emails from the server I'm utilising SMTP with application and database. Therefore I am now able to continue the integration to the database system.



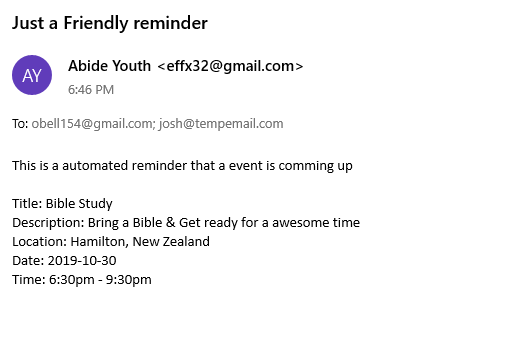
I concluded, to add the emails as I would just extract the emails from the database then run it through a while loop to tell the email function to send those emails.

$sql = "SELECT email FROM Users";  
//https://www.w3schools.com/sql/sql\_orderby.asp  
$result = $\_conn->query($sql);  
if ($result->num\_rows > 0) {  
 // output data of each row  
 $count = 0;  
 while($row = $result->fetch\_assoc()) {  
 foreach ($row as $count) {  
 $mail->addAddress($row["email"]);  
 $count++;  
 }  
 }  
} else {  
 echo "0 results";  
}

The next issue that needs to be overcome is the format of an email into a user friendly format email. Fortunately by the phpmailer library I am able to use HTML /markupformatting /to help style.

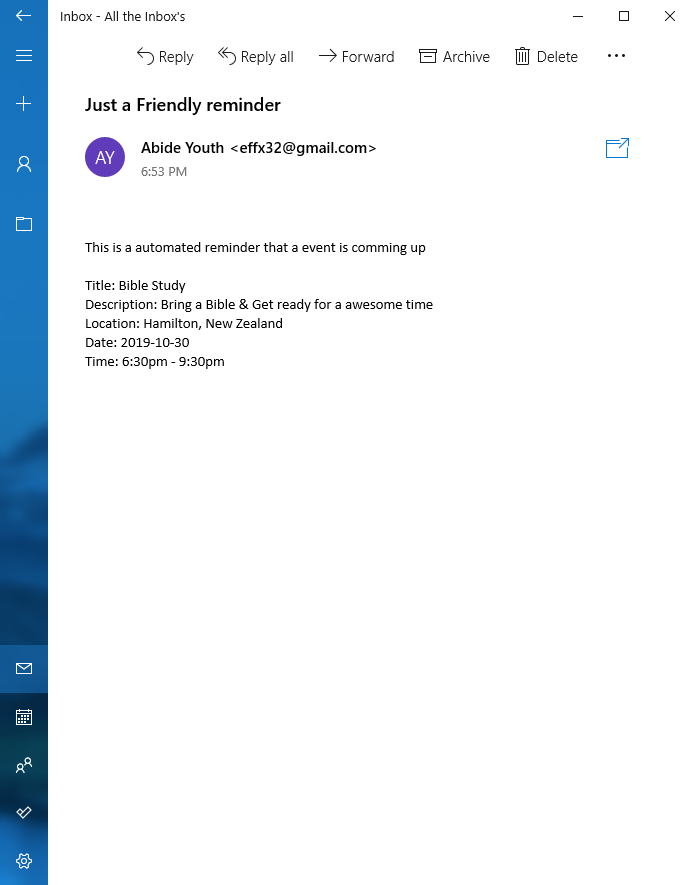
Reutilising the code previously used for displaying events text on the homepage I have adapted it so that it presents the information in a format that is friendly to sending emails.

After adding a short description of why they received the email the outcome is as shown.



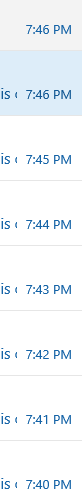
Unfortunately here I can see that people's addresses To fix this I'm going to go and change

$mail->addAddress($row["email"]);  
$mail->addBcc($row["email"]);



Now we can see we no longer have the implication security as private information that was stored in the database being linked to the Public/Youth.

Using [https://crontab.guru](https://crontab.guru/) is was to assist me with generating the crontab time for testing I have set it to run every 1 minute.



As we can see here, it was successful on activating on a regular basis as seen on the left.

In theory, it should be able to be extrapolated to once a week.

But unfortunately, crontab did not smoothly integrate with a dynamic system that sent out a reminder the day before the event was happening.

Rather than utilising the system that was designed on a regular schedule I have a solution that is more dynamic based.

The job at first is activated one day before the event

1. Clear all jobs from the job table there for making sure there is no double up on reminders
2. Then it sets another php file to run just after the actual time of the event
3. When the second job run it will update the Events.txt file from the google calender data base then will read what event is coming up next then set a job for that time

for i in `atq | awk '{print $1}'`;do atrm $i;done

Found from <https://unix.stackexchange.com/questions/53144/remove-all-at-jobs>

When the code above is executed to clear all queued jobs it works when executed from the terminal but returned errors when executed from the php file.

So I turned to <https://www.commandlinefu.com/commands/view/4618/remove-at-jobs> For more PHP code **atrm $(atq|cut -f1)**

With now clearing the job table complete, I update the events file with “update-events” (this is a custom linux cmd). Then set the main.php (main.php was call index.php) File to be run 24 hour before the next event.   
  
In summary,

Main.php runs  
> gets email from database   
> gets data of the event that is about to happen next.

> Send out an email with event data.

> Tell the server to run heartbeat.php an hour after the event.

Once heartbeat.php has finished:

> It clears all queued jobs (This is a precaution to protect from border cases)

> Update events.txt with the latest event

> Sets Main.php to run 24hours before the next event

After setting this all up I have run and tested some events that could be used. Although, I encountered the issue that if two events are closer than 24 hours between each other the cycle somehow gets broken causing the program to stop working as expected.

I am working on experimental code for dynamically changing the event so an email is sent out at a specified time before the event.

So far, I have started taking the time and date of the event now I have to check the difference between the two times in order to give a dynamic output of reminder time or to skip the reminder altogether.

I couldn't get this to work in the given time frame therefore I have simplified it a little bit.

I have just gone to set this program to work with a reminder being sent out 2 hours before. This means that there must be about 4 hours between each event this shouldn't be an issue as events tend to happen on a Friday from 6pm to 9pm or on Saturdays 9am at the earliest. Meaning that there is plenty of time in between events.

Issues with **Version 8**

* Cannot counter for whether or not there is a upcoming event or if the time schedules are just empty.

**Version 9 (Flexibility and Closure)**

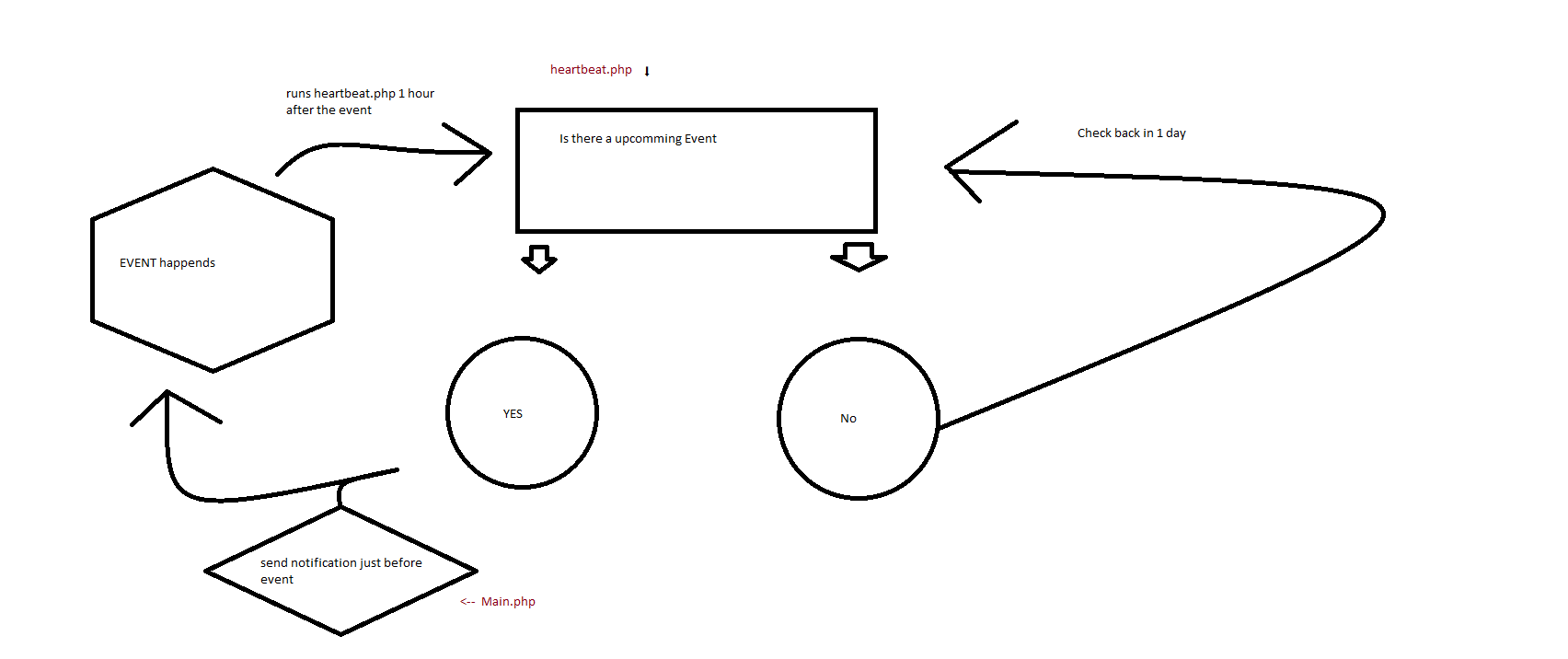
The code for this can be found in version 9 folder

Summary

Increase adaptability in each different use case. This is things such as board cases like what happens if there's no event present so the next day it will come back and check them. It should repeat this cycle daily.

First step was to make sure the server could handle what to do if there is no event that is upcoming. To achieve this I started my counting the amount of events yet to happen. This was achieved with size of() function within php, this lets me see the amount of upcoming events. The next step is utilising the information so that a system can be developed so the server can handle requests even when there is none and especially when no data for events is coming in

The goal is for the server is to end up with the process that the image below depicts:



So the next step that I took was to find a way to transfer the starting of the loop to my heartbeat.php. To complete this I decided to go with a persistent method and utilise a text file as a transfer medium as well as an offline buffer for redundancy. This was called `AT`

While sitting this up, I encountered several issues with file permissions of which are now resolved with the help of the chmod command. (Used for changing file permissions)

Now with the information being added to the heartbeat.php file, I can now create the if statement for whether an event is upcoming or not which it then loops back to the start for the next day.