# Section 2 – Design

## Part (A) – Methods, Variables, Classes & Parameters

### Methods

Due to the nature and lifecycle of my project (A website), while there are still a few helper functions, the majority of the code is in the form of method like segments of php code separated by HTML content. For this reason, I will treat relevant segments similar to anonymous (lambda) functions and include these in my method table along with the functions in the helper classes.

Method tables

*Note: All method tables subject to change*

*Note: Only page with methods have a method table*

index.php

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Parameters | Purpose | Output/Return | Possible error output |
| anon | None | Log user’s IP Address, time and date into the database | None | Connection error message  MySQLi error message |

connect.php

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Parameters | Purpose | Output/Return | Possible error output |
| connect | None | Reads database access tokens from server HDD and connects to the database using said access tokens | MySQLi database connection object | Connection error message  MySQLi error message |

downloadsearchq.php

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Parameters | Purpose | Output/Return | Possible error output |
| anon | q (POST from prev page) | Uses MySQLi LIKE clause to query the database for questions where the query POSTed to the page (q) is either in the title or question body. The returned questions are then outputted in a format where searchq.php can output them. | Title, id and number of votes from matching questions returned from the database | Connection error message  MySQLi error message |

question.php AT TIME OF WRITING I HAVE NO IDEA WHY SetCookies EXISTS AND WHAT ITS PURPOSE IS

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Parameters | Purpose | Output/Return | Possible error output |
| SetCookies | None | Store the current question id in a cookie called “current\_qid” and store a Boolean value in a cookie called “logged\_in” depending on if the user is logged in or not. Also creates a cookie called “current\_username” and sets it to the user’s username if the user is logged in.  However, if no question id is specified in the URL, redirect the user to error.php specifying the error URL parameter as noquestionid. | No output, but can create and set cookies and redirect the user to the error page. | Connection error message  MySQLi error message |
| Anon | None | Queries the database for the question with id specified in the URL parameter “id”. Stores the relevant data from the result in the database | No output, but creates 5 variables and stores data fetched from the database in them. | Connection error message  MySQLi error message |
| Anon | None | Creates an array of lines in the question by using the build in method explode with the parameters “\n” and $qContent.  Split the newly created array using the SplitLines function from questionFuncs.php.  Iterates through the lines in the $questionArray array using a for loop  Checks to see if the index of the iteration is in the $splitArray[0] array. If it is, output the value stored in $questionArray[$i] where $i is the current iteration.  If it isn’t, check if the index of the ireratopm is in the $splitArray[1] array. IF it is, create a variable called line and make it equal to the value stored in $questionArray[$i] stripped of starting and trailing whitespaces where $i is the current iteration of the loop. Whitespaces are removed using the building method trim().  Check if the length of the line is more than three and the first three characters of the line are equal to “```”. The length of the line is given by the build in method strlen() and a substring of the first three characters is given by the build in method substr(). If these conditions are both true, then that start of a code block must have been detected. Calls StartCodeBlock() function from questionFuncs.php  If the above condition is not true, and the lengh of the line (given by the built in method strlen()) is equal to 3 and the first three characters of the line (given by the built in method substr()) is equal to “```”, the end of a code block must have been detected, and the EndCodeBlock() function from questionFuncs.php is called.  If neither of these conditions are met, then the current line must be code that needs formatting using the code block markdown. This is handled by simply outputting the line, but first any “special characters” (characters that have significance in HTML) need to be converted to HTML entities, to prevent the browser interpreting the < symbol as the start of an HTML tag. This conversion is done using the built in method htmlspecialchars(). | Outputs text formatted as code using a primitive code markdown parser I wrote. | None |

questionFuncs.php

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Parameters | Purpose | Output/Return | Possible error output |
| UsrVoted | $id, $qID, $connection | Queries the database to work out if a user has voted on a question. It does this by preparing a MySQLi SELECT query to select the `id` attribute from the `votes` table where the `uID` attribute is equal to the $id parameter and the `qID` attribute is equal to the $qID parameter. It then checks the number of rows returned from the database. If the user with the user id stored in the parameter $id has voted on the question with the question id stored in the parameter $qID, one row with be returned from the database by the query. If said user has not voted on said question, then it will return false. | true/false | MySQLi error message  Connection error message |
| Upvoted | $id, $qID, $connection | Queries the database to work out if a user has votes a question up. As with the UsrVoted function, it does this by preparing a MySQLi SELECT query to select the `type` attribute from the `votes` table where the `uID` attribute is equal to the $id parameter and the `qID` attribute is equal to the $qID parameter. If it is then it checks if the returned value is equal to “u” (for up). If it is, then the function returns true. If not, the function returns false. | true/false | MySQLi error message  Connection error message |
| ShowVotedArrows | $qID, $qVotes, $connection | Checks to see if the question with the id, $qID, has been voted up by the user with the user id stored in the SESSION variable $\_SESSION[“id”]. It checks this by calling the Upvoted() function with the parameters $\_SESSION[“id”], $qID and $connection. If Upvoted returns true (if the user with the user id, $\_SESSION[“id” up voted the question with the id, $qID), then the function outputs a clickable green up arrow and a clickable grey down arrow. It also sets the onclick attribute to Up(“green”) and Down(“grey”) respectively. It also outputs the number of votes the question has (calculated by up – down)  If not, then it outputs a clickable grey up arrow and a clickable red down arrow (Because the ShowVotedArrows function is only called if the user has voted, so no upvote equals a downvote). As above, the onclick attribute is respectively set to Up(“grey”) and Down(“red”), as well as displaying the score of the question in between the arrows. | Outputs clickable arrows coloured to represent the way a user has voted, and the score of the question in between. | MySQLi error message  Connection error message |
| ShowGreyArrows | $qID, $qVotes, $connection | Similar to ShowVotedArrows, but without any processing. Only used to ouput clickable grey arrows with onclick functions and vote score. | Outputs clickable grey arrows and the score of the question in between. | MySQLi error message  Connection error message |
| GetComments | $qID, $connection | Uses a MySQLi SELECT statement to select everything from the `comments` table where the `qid` attribute is equal to $qID. It also uses the ORDER BY keyword (s) to order the results by highest score in descending order.  Then uses a while function to iterate through all the comments returned from the database and uses the Comment function to output them. The comment function takes the parameters $row and $connection. | Outputs the comments from a question in defencing order based on comment score. | MySQLi error message  Connection error message |
| Comment | $row, $connection | Separates the $row array into the individual sections of the comment (author, comment, commentID and qID) and outputs them, using an inline if statement to show an edit button if the user is logged in (Based on the SESSION variable $\_SESSION[“username”]). | Outputs and formats a comment. | MySQLi error message  Connection error message |
| getUserID | $connection, $username | Uses a MySQLi SELECT statement to select the `id` attribute from the `users` table where the `username` attribute equals the $username parameter. | Returns the user id of a given user | None. |
| SplitLines | $questionArray | Initialises two empty arrays, $normalLines and $codeBlockLines. Then uses a for loop to iterate through $questionArray. For each iteration, a variable called $line is created and set to the value of $questionArray[$i] where $i is the current iteration (between 0 and sizeof($questionArray))  Then, if the first character of the $line is “`”, add the line to the $codeBlockLines array. The first character is given by the build in substr() method. If not, the $line is added to the $normalLines array.  After finishing iterating through the lines, the function returns a multidimensional array of the two arrays. | A multidimensional array of the $normalLines array and the $codeBlockLines array. | None |
| StartCodeBlock | $language | 2nd Shortest function in questionFuncs.php. Opens a <pre> tag and opens a <code> tag using the $language parameter to specify the language of the code block. This is used by the prism-js library to format and style the code in accordance with its language. | Opens a <pre> and a <code> block. | None |
| EndClodeBlock | None | Shortest function in questionFuncs.php. Simply closes the <code> tag and <pre> tag. | Closes a <pre> and a <code> block. | None |
| isUsersComment | $connection, $username, $id | This function is used to check if the current logged in user is the owner of a supplied comment.  It does this by using a MySQLi SELECT statement to select the `author` attribute from the `comments` table, where the `author` attribute matches the $username parameter and the `id` attribute matches the $id parameter (comment id). As with the UsrVoted() and Upvoted() functions, if the number of rows returned from the database equals 1, then the function returns true, and if not the function return false. | Returns true if a specified comment was written by a specified user. Otherwise returns false. | MySQLi error message  Connection error message |
| GetComment | $connection, $id | Simple function that fetches the comment body from the database with a specified id. It does this by using a MySQLi SELECT statement to select the `comment` attribute (comment body) from the database where the `id` attribute matches the $id parameter supplied to the function. It then returns the comment body. | The comment body for a comment based on a specified comment id. | MySQLi error message  Connection error message |

downloadquestions.php

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Parameters | Purpose | Output/Return | Possible error output |
| Anon | None | Prepares the MySQLi statement for the three types of filter (new, top and hot) and stores the chosen one as a string. The “new” query simply works by selecting all the attributes from the first 10 rows in the `questions` table and ordering it by descending id. The “top” query works the same way as the “new” query but orders it by the `votes` attribute instead of the `id` attribute. The `hot` query is even simpler, as all the processing is done in a separate function, instead of both in the query and a separate function, like “new” and “top”. It simply selects all the attributes about the first 10 questions.  Then the query is run and the result is stored in an array called $result. If the type is not “hot”, then the notHot() function is ran. If not, the hot() function is ran. | None | MySQLi error message  Connection error message |
| hot | $result | The algorithsms uses in this function are modified from reddit’s “hot” algorithm.  Creates two empty arrays, $scoreArray and $array. Then, it iterates through the $result array. A $points variable is created and set equal to the number of votes question currently being iterated over has (given by $row[“votes”]). A variable called $order is also created. The value of $order is given by (loge(max(abs($points), 1, 10). Then, if the $points variable is bigger than 0, A variable called $sign is created and set to equal 1. If the $points variable is smaller than 0, A variable called $sign is created and set to equal -1. If neither of the two above conditions are met, a variable called $sign is created and set to equal 0.  Then, a variable called $seconds is created and set to the Unix time when the question was asked - the constant 1516221943 (Wednesday, 17 January 2018 20:45:43, the date the system was first implemented). Another variable, $score, is also created, and assigned the value of ($order + $sign \* $seconds / 45000) rounded to 7 decimal places. The rounding is done using the build in method round(). It then pushes $score to the $scoreArray array, and pushes a new array composed of $score, $row[“title”], $row[“id”] and $row[“votes”] to the $array array.  Then, after all the returned questions have been processed, it sorts the array by a user defined order as defined in the sortOrder() function. The function to sort an array by a custom sorting function is provided by the build in method usort().  Then, the contents of $array are outputted in a format that can be read by the scripts running on question.php. | The 10 “hottest” questions are outputted to the user, in a form that question.php can output it to the user in the questions table. | MySQLi error message  Connection error message |
| notHot | $result | Iterates through the array using a while loop and outputs the questions to in format which can be read by the scripts running on question.php in the questions table. | Outputs either the “top” 10 questions or the “newest” 10 questions in format which can be read by the scripts running on question.php, where they can be outputted to the user in the questions table. | None |
| sortOrder | $a, $b | Defines a sotring function to be used in the build in method usort(). The purpose of this function is to sort the array in order of the highest score attribute. | $b[“score”] - $a[“score”] | None |

downloadsearchq.php

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Parameters | Purpose | Output/Return | Possible error output |
| anon | None | Connects to the database and selects all the attributes from the `questions` table where the `title` attribute contains the search string (stored in the substring $q) or the `question` attribute contains the search string ($q again). This is done by using the MySQLi LIKE keyword and using $q as a wildcard to return any questions where the search string is either in the title or question.  The function then iterates through the results and outputs them in a format readable by the scripts running on searchquestion.php.  Finally, it closes the connection to the database. | Outputs the results of the search in format which can be read by the scripts running on question.php, where they can be outputted to the user in the questions table. | MySQLi error message  Connection error message |

processsignup.php

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Parameters | Purpose | Output/Return | Possible error output |
| anon | None | Initiates and populates variables from POST ($username, $password, $firstname, $lastname and $emailaddress), and generates a hash of $password and stores it in the variable $password\_hash. The hashing function is provided by the built in function password\_hash(), using the constant PASSWORD\_DEFAULT as the algorithm parameter. | None | None |
| anon | None | Gets the user’s IP Address from the session variable $\_SESSION[“IPADDR”] and stored it as a variable called $address, and prepares and runs a MySQLi query to select the `address` attribute from the `blocked\_ipaddr` table, where the `address` attribute equals $address (the user’s IP Address).  Then, it uses this query to check if the user has been banned from creating user accounts, by checking the number of rows returned (>0 equals banned) It then checks if all the fields are filled in by checking if the variables $username, $password\_hash, $firstname, $lastname and $emailaddress all contain data. This check is done by using the built in method isempty(). This check is done to stop the user entering blank value for the various account attributes. If this check fails then an error message is displayed to the user, along with a button to take the user back to the sign-up page.  Next, the function performs a simply email validation check to see if the value specified as an email address contains an @ symbol. This is far from fool proof, but at least performs so basic validation. This check is performed using the built in function, strops(), with the parameters emailaddress and “@”.  If this check fails then an error message is displayed to the user, along with a button to take the user back to the sign-up page.  If this condition is met, then an array containing the all the login attributes, called $paramsArray, is created. This is then passed into the function CreateAccount(), alongside the $connection object. These are stored in an array to keep the number of parameters passed down. This is not an elegant solution, so I will probably end up changing it. | None | MySQLi connection error message  Connection error message |
| CreateAccount | $connection, $paramsArray | Creates and initialises variables ($username, $password\_hash, $firstname, $lastname, $emailaddress) from the $paramsArray array, then creates a MySQLi query to select the `id` attribute from the `users` table where the `username` attribute equals the value stored in the $username variable. This query is assigned to the variable $query. The query is then run, and the result is stored in the variable $result. Another variable called $num\_rows is created and assigned the value of the number of rows returned by the query stored in $query. The number of rows is worked out using the built in function mysqli\_num\_rows(), with $result as the single parameter. This check is done to see if a user with the same username already exists. The other attributes are not checked, because multiple user can have the same value for the rest of the attribute, e.g. password, as multiple users can use the same password without error, but multiple users cannot share a username. | Creates a new user account in the database. | MySQLi connection error message.  Connection error message. |
| redirect | $url | This function is designed to redirect the user safely to another page. This is an alternative to using the build in function header() to change the location of the page. This is because you cannot modify headers after they have been sent, which means that I couldn’t modify any part of the page before redirecting, which meant I couldn’t display error message. The redirect function takes the desired url as a parameter ($url).  First, the function checks whether the headers have been sent or not. If they haven’t, it uses the previous method (the built in header() function, with $url as the url), and quits the loop. If they have (which they will have been in most cases), it outputs a JavaScript script using echo, which redirects the user to the url stored in the $url parameter. |  | HTTP 404 error code. |

processlogin.php

Doing these tables is really boring so I’m just going to stop and do it later

## Part (B) –Algorithms

This section is probably worse than the method tables so I’m going to skip this for now and do it when I can be bothered

## Part (C) – Usability

### Learnability

My website should be intuitive and easy to learn for the first time. I feel that this will give it an edge over more complicated sites such as StackOverflow. If a user is coming to my site in order to learn programming or find a fix for their code, they should not have to learn how to use my site first. With this sentiment in mind, I designed my website to be as easy to navigate as possible, by breaking it down into three distinct segments: ”Q&A”, “Careers” and “Tutorial zone”. These are all in their own box on the homepage along with a description. This allows a user to easily select the relevant section of the site for their specific needs.

When the user is on the correct section of the site, the layout is incredibly straight forward. The “Q&A” section of the website is mainly compromised of a search bar (with option to search for tags), an “Ask question!” button and a large table of questions which can be filtered by “hot”, “top” and new”. This simple layout makes it really easy for a user to either ask a question (Simply by clicking the large “Ask question!” button), or to access a previous question (By searching or by filtering using the table).

The tutorial zone is even more self-explanatory. It simply has a list of common languages and technologies. When you click on a language, it redirects you to a page with a few subheadings. These vary based on the language but often include a list of sample programs written by me to show how the theory works in practice and to provide examples, links to various 3rd party tutorials and official documentation, and a description of the language and what it’s used for in industry.

The careers section is broken down again into two sections to help users access the relevant information and features. These sections are “Employers” and “Looking for work”. As with the homepage, these will be in separate boxes with a description to point the user in the right direction. The “Employers” section …  
The “Looking for work section ...

Do this when I have a clue what these two sections will look like

I will be testing the learnability of my site by using several people who have never seen or used the site to perform a set task on it, such as apply for a job that meets some given parameters, or to create a new account and ask a question. I will then ask them to give me feedback. This will allow me to see if the site is as easy to learn as I believe it to be. If they all complete the task with ease and have good feedback, then I will consider my site to be easy to learn. I will take on board any negative feedback and implement anything relevant in order to improve the user experience and learnability of the site.