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.

NOTE: The initial design of my project was for a procedural style of PHP, but about half-way through the coding I swapped to an object-oriented style of programming as described later on. These method tables however were written while I was still planning on using the procedural style of PHP, so while the functionality and algorithms are the same, the method names, parameters, returns and overall structure of the final outcome are very different.

Method tables

Note: All method tables are subject to change in final build

Database.php

Name	Parameters	Purpose	Output/Return	Throws	Class
__construct	None	Class constructor. Initialises variables and Objects. Reads database configuration details from the ini file stored on the server outside the public web directory and attempts to create a new connection to the database.	None	Custom exception: “Error connecting to database”	Database
query	String: $query	Execute a MySQLi query against the current connected database.	Returns the result of the query if there was a result. No return if nothing returned from the database	Custom exception: “Error executing query”	Database

			either due to error or non-SELECT query.		
getResult	None	Getter function. Gets the value of the current query stored in memory. Returns the value stored in $this->result.	Returns the value stored in $this->result	None	Database
getConnection	None	Getter function. Gets the value of the MySQLi connection object stored in memory. Returns the object $this->connection	Returns the object $this->connection	None	Database
fetchAssoc	None	Create an associative array of the data returned from the last query. Runs the mysqli_fetch_assoc on the value stored in $this->result (The result of the last query executed against the database) and returns the result. Only runs the function if there is data in $this->result to avoid a null pointer exception.	Returns an associative array of the data stored in $this>result. No return value of $this->result is NULL.	None	Database
numRows	None	Gets the number of rows returned from the last query ran against the database. Runs the mysqli_num_rows function on $this->result to get the	Returns the number of rows returned from the database from $this>result.	None	Database

		number of rows of data returned from the database. Only runs the function if there is data in $this->result to avoid a null pointer exception.	No return value of $this->result is NULL.		

index.php

Name	Parameters	Purpose	Output/Return	Throws	Class
None	None	Set up some session variables if they don't already exist such as $_SESSION["IPADDR"] for storing the IP Address of the client's connection. Creates and initialises the Database class for use on the page. Collects information about the user such as the time and data they visited the site, as well as the page they visited and store it in the database using the Database->query() function. Include the universal header for the website from the header.html file.	None	Custom exception: "Error connecting to database" Custom exception: "Error executing query"	None

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	Title, id and number of votes from matching	Connection error message

		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.	questions returned from the database	MySQLi error message

question.php

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	Outputs text formatted as code using a primitive code markdown parser I wrote.	None

		$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().		

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	true/false	MySQLi error message

		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.		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	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

		respectively set to Up(“grey”) and Down(“red”), as well as displaying the score of the question in between the arrows.		
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	Returns the user id of a given user	None.

		equals the $username parameter.		
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	Returns true if a specified comment was written by a	MySQLi error message

		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.	specified user. Otherwise returns false.	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\`	None	MySQLi error message Connection error message

		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.		
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 $order is given by (log$_e$(max(abs($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	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

		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.		
notHot	$result	Iterates through the array using a while loop and outputs the questions to	Outputs either the "top" 10 questions or the	None

		in format which can be read by the scripts running on question.php in the questions table.	"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.	
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.	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

		Finally, it closes the connection to the database.		

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,	None	MySQLi connection error message Connection error message

		$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.		
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		HTTP 404 error code.

		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.		

core.php

Name	Parameters	Purpose	Output/Return	Throws	Class
__constructor	String: $username String: $password String: $first_name String: $last_name $email_address	Class constructor. Used to declare and initialise variables and objects. Checks if a password attribute has been provided and generates a hash of it using the User->generateHash(). This check is done because you cannot	None	None	User

	Note all parameters default to NULL so constructor can be called with any number of parameters	generate a hash of a password if you have no password.			
getPassword	None	Getter function to get current password for an instance of the User object.	$this->password	None	User
getUsernane	None	Getter function to get current username for an instance of the User object.	$this->username	None	User
getFirstName	None	Getter function to get current first name for an instance of the User object.	$this->first_name	None	User
getLastName	None	Getter function to get current last name for an instance of the User object.	$this->last_name	None	User
getEmail	None	Getter function to get current email address for an instance of the User object.	$this->email	None	User
generateHash	None	Uses the built-in method password_hash on the current password stored in an instance of the User object. Uses PASSWORD_DEFAULT as the hashing algorithm.	A hash of $this->password	None	User
verifyHash	String: $password	Takes a password as a parameter and checks if it matches the hash stored in $this->password_hash	Returns true if the supplied password matches the stored hash. Returns false if the supplied password does not match the stored hash	None	User
getID	Database: $database	Uses a reference to the Database class to perform a query against	Returns the ID attribute of $this (A	Custom exception: “Error	User

		the database to get the ID attribute of $this (A specific instance of the User object)	specific instance of the User object)	connecting to database” Custom exception: “Error executing query”	
allAttributesFilled	None	Calls Util::_isset on all the attributes in an instance of the User object ($username, $password, $first_name, $last_name, $email_address)	Returns true if all attributes in the class pass the isset() check. Returns false if one or more of the attributes do not pass the isset() check	None	User
create	Database: $database	Function used to add an instance of the User object to the database. First, generates a new hash of the password by setting $this->password_hash = $this->generateHash(); Then use $database->query to execute an INSERT INTO MySQL query into the database to add a new record for the user. Then use $database->query to perform a SELECT query against the database to retrieve the auto-incrementing id attribute assigned to the newly created user. Then stores the username attribute and the id attribute in	None	Custom exception: “Error connecting to database” Custom exception: “Error executing query”	User

		session variables to log the user in. Next, the built-in method session_write_close is called to allow the session variables to persist after HTTP headers are sent. Then call Util::redirect to redirect the user back to the homepage			
Redirect	String: $url	Function used to safely redirect the user to a different page. First, checks if HTTP headers have already been sent using the built-in method headers_sent(). If the HTTP headers haven't been sent, use the built-in header function to redirect the user by supplying Location: $url as the header. If the headers have already been sent, use PHP to output a HTML <script> tag containing some JavaScript which will be ran and used to redirect the user.	None	None	Util
isIPBlocked	String: $address Database: $database	Uses the $database->query function to query the database to select the address attribute from the database. If there are any results (Maximum of 1) then the address has been blocked.	Returns true if the IP Address has been blocked. Returns false if the IP Address hasn't been blocked	Custom exception: "Error connecting to database" Custom exception: "Error	Util

		If there are no results, then the user hasn't been blocked.		executing query"	
_isset	…$vars The … syntax allows for an unspecified number of parameters to be supplied to a function.	Uses a foreach loop to loop through all the variables supplied in …$vars and runs an isset() check on each variable. If a single variable fails to pass this check then the function returns false. After the loop the function will return true. This is because if the function doesn't return false earlier on then all the variables managed to pass the isset check and the function can safely return true.	Returns false if one or more of the variables doesn't pass the isset check. Returns true if all the variables pass the isset check.	None	Util
emailValid	String: $email	Uses the build in function filter_var using $email as the variable parameter and FILTER_VALIDATE_EMAIL as the filter.	Returns true if the supplied parameter is a valid email address. Returns false if the supplied parameter is not a valid email address	None	Util
Error	String: $message Boolean: $backbutton String: $backURL (= NULL to allow it to be optional)	Function to display an error message to the user. First, simply output the error message supplied followed by a line break HTML tag (<br/>) Next, if $backButton is true, output a <button> element with the onclick attribute equal to $backURL	None	None	Util

modifyvote.php

Name	Parameters	Purpose	Output/Return	Throws	Class
None	None	Create a new Database object and initialise it. Create and initialise variables used in the rest of the class. Get the question id, username and function attributes from the URL and stores them in variables Use a switch statement to run a different function based on the function parameter in the URL.	None	None	None
AlreadyVoted	Database: $database Int: $qID Int: $uID	Function to decide whether a user has voted on a question. Prepare and run a MySQLi query using the $database object to select the id attribute from the votes table where the $qID parameter matches the qID database attribute and the $uID parameter matches the uID database attribute. If there is a result from the query, the user must have voted on the question, so return true. If there are no results from the query then return false as the	Return true if the user with Id $uID has voted on the question with Id $qID. Return false if the user with Id $uID has not voted on the question with Id $qID.	Custom exception: “Error connecting to database” Custom exception: “Error executing query”	None

		user can't have voted on the question			
AddUpVote	Database: $database Int: $uID Int: $qID	Function to add an "up vote" to a question. First, run the AlreadyVoted function to check if the user has already voted on the question. If they haven't already voted on the question, use the $database object to insert a new record into the votes table with the qID attribute of $qID, the uID attribute of $uID and the type attribute of "u" to signify and "up vote".	None	Custom exception: "Error connecting to database" Custom exception: "Error executing query"	None
AddDownVote	Database: $database Int: $uID Int: $qID	Function to add an "down vote" to a question. First, run the AlreadyVoted function to check if the user has already voted on the question. If they haven't already voted on the question, use the $database object to insert a new record into the votes table with the qID attribute of $qID, the uID attribute of $uID and the type attribute of "d" to signify and "down vote".	None	Custom exception: "Error connecting to database" Custom exception: "Error executing query"	None

RemoveVote	Database: $database Int: $uID Int: $qID	Uses the $database object to run a query against the database to remove a record from the votes table where the qID attribute equals $qID and the uID attribute equals $uID	None	Custom exception: “Error connecting to database” Custom exception: “Error executing query”	None
getUserID	Database: $database String: $username	Uses the $database object to run a query against the database to select the if attribute from the users table where the username database attribute equals $username.	Returns the id attribute of the user with the username $username	Custom exception: “Error connecting to database” Custom exception: “Error executing query”	None

filterQuestions.js

Name	Parameters	Purpose	Output/Return	Throws	Class
Download	None	Function used to send an HTTP GET request to download the output of the downloadquestions.php script. First, gets the String contents of the select box on the page and uses this value to prepare the URL to query. Then, makes a new XMLHttpRequest object, sets its callback function to the Output function and opens and sends a request.	None	None	None
Output	None	Function used to format and output the results of the get request.	None	None	None

		First, checks the readyState and status of the XMLHttpRequest object. If they are both “ready” then store the response of the request. Next, create an array of the response by using the build-in split method using “<br/>” as the delimiter. After that, iterate through the table on the HTML page and delete all the nodes except from the table headings. Then, the function calls the createElements function with the response of the array and a reference to the HTML table as the parameters.			
createElements	Array: questionArray HTML element: table	Iterates through the supplied array and creates new HTML elements to append to the table using the document.createElement function and add text to said elements using the document.createTextNode function. The text comes from the supplied questionArray array. Finally, the function appends the created elements to the page using the appendChild and appendChildren function.	Outputs the contents of the array as part of the HTML table.	None	None
appendChildren	HTML element: Parent Html elements: …children	Function to append multiple child elements to a single parent element. Iterates over all the variables provided in …children and appends each one to the element	None	None	None

	(…var = JavaScript syntax for an unspecified number of variables)	provided in the parent parameter.			

filterSearchQuestions.js

Name	Parameters	Purpose	Output/Return	Throws	Class
Download	None	Function used to send an HTTP GET request to download the output of the downloadsearchq.php script. First, gets the String contents of the search box. Then, makes a new XMLHttpRequest object, sets its callback function to the Output function and opens and sends a request.	None	None	None
Output	None	Function used to format and output the results of the get request. First, checks the readyState and status of the XMLHttpRequest object. If they are both “ready” then store the response of the request. Next, create an array of the response by using the build-in split method using “<br/>” as the delimiter.	None	None	None

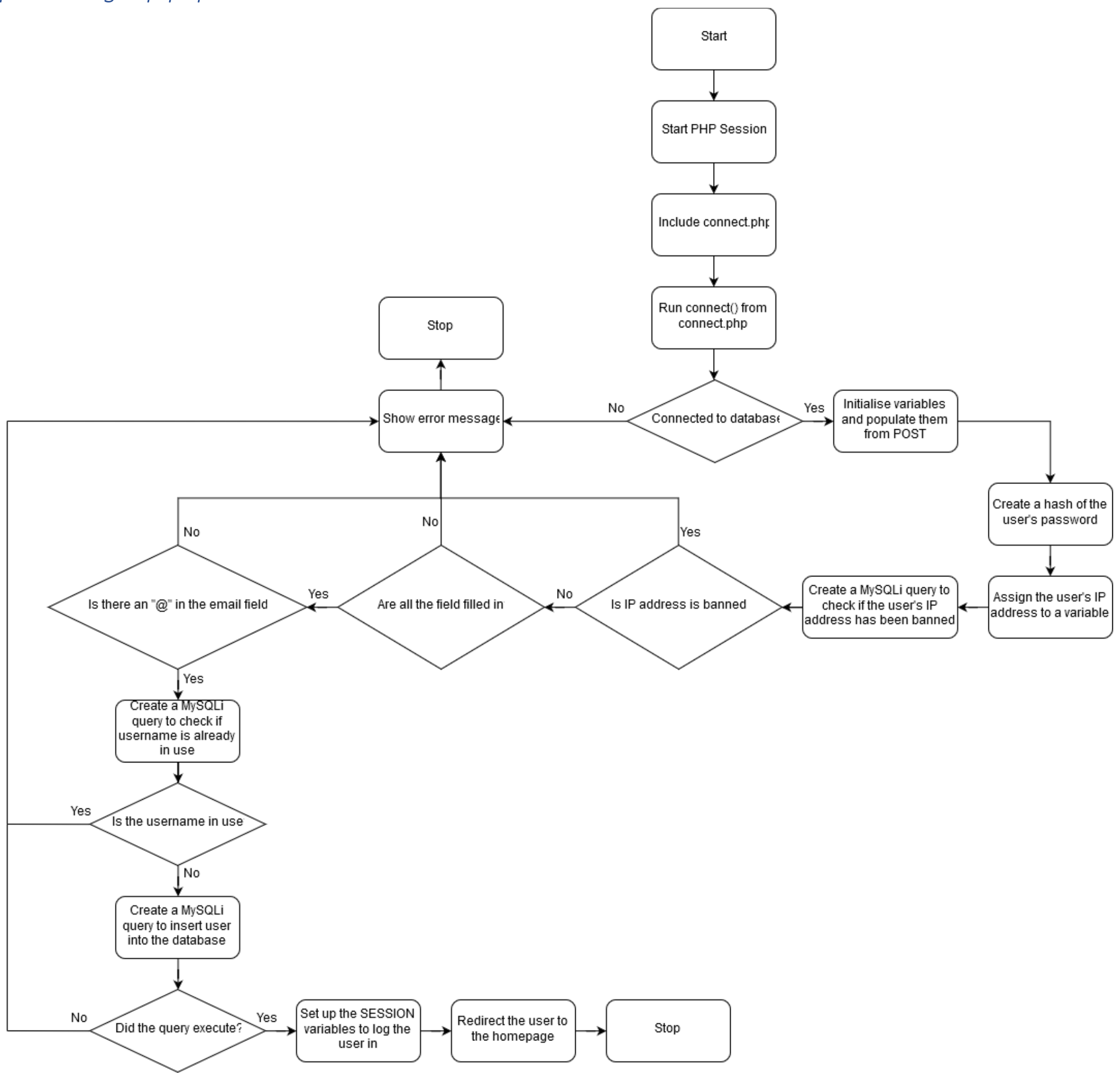
		Then, call the wipeTable function supplying the HTML table as the parameter. After that, if the length of the array returned from the XMLHttpRequest is less than 3, the createBlankResult function is called with the HTML table and an error message as the parameters. This is because the response from the server comes in three parts which combine together to make a single result. Therefore, if the array contains less than three items then no results have been fetched from the server. If the size of the response array is 3 or more, the createElements function is run, using the questionArray array and the HTML table reference as the parameters.			
wipeTable	HTML table element: table	Function to remove all the nodes from a HTML table. Uses a while loop to run the parent.removeChild function while the Boolean value of parent.hasChildNodes is true.	None	None	None

createElements	Array: questionArray HTML table element: table	Function used to create all the elements needed for results table. Iterates through the questionArray array, three items at a time, and append them to the table.	None	None	None
createBlankResult	HTML table element: table String: message	Function used to output a blank table to the user with a message saying no results found. Creates all the nodes for the table and appends it to the table	None	None	None
appendChildren	HTML element: parent …children (…var = JavaScript syntax for an unspecified number of variables)	Function to append multiple child elements to a single parent element. Iterates over all the variables provided in …children and appends each one to the element provided in the parent parameter.	None	None	None

Part (B) –Algorithms

Flowcharts:

As with the method tables, all flowcharts are subject to change in the finished solution *processsignup.php*


Start
Start PHP Session
Include connect.php
Run connect() from connect.php
Connected to database
No
Yes
Show error message
Stop
Initialise variables and populate them from POST
Create a hash of the user's password
Assign the user's IP address to a variable
Create a MySQLi query to check if the user's IP address has been banned
Is IP address is banned
Yes
No
Are all the field filled in
No
Yes
Is there an "@" in the email field
No
Yes
Create a MySQLi query to check if username is already in use
Is the username in use
Yes
No
Create a MySQLi query to insert user into the database
Did the query execute?
No
Yes
Set up the SESSION variables to log the user in
Redirect the user to the homepage
Stop

processsignup.php is the page that creates a new user and adds it to the database. Before any use is created, a number of factors must be checked. The flowchart starts by starting a new PHP session (To provide access to SESSION_ variables), and then including connect.php. connect.php contains the function connect() which returns a MySQLi connection object. This function is then run in the flowchart to create a way of interacting with the database.

Next, there is a decision to be made. This comes in the form of an if statement asking if the script is connected to the database. In other words, this step checks that the connect() function ran without errors and managed to establish a connection between the client and the server. If this decision returns false, then an error is shown to the user and the STOP block is reached. This signals the end of the script. However, if the decision returns true, the flowchart moves onto the next block.

This next block declares the necessary variables for creating a user (username, password, etc…) and initialises them using the POST data from the previous page (the signup form). These are declared as variables instead of simply being accessed from the POST_[] array for simplicity and readability.

In the next block, the program generates a hash of the user's password for storing in the database. Unlike T-Mobile Austria, I don't store passwords as plaintext!

(https://motherboard.vice.com/en_us/article/7xdeby/t-mobile-stores-part-of-customers-passwords-in-plaintext-says-it-has-amazingly-good-security). This increases the security of the website and the data it stores.

Next, the flowchart creates a new variable and uses it to store the user's IP Address. It then moves on and queries the blocked addresses table in the database against the value stored in the previous block. It stores the result of the query as a variable, then moves onto the next block.

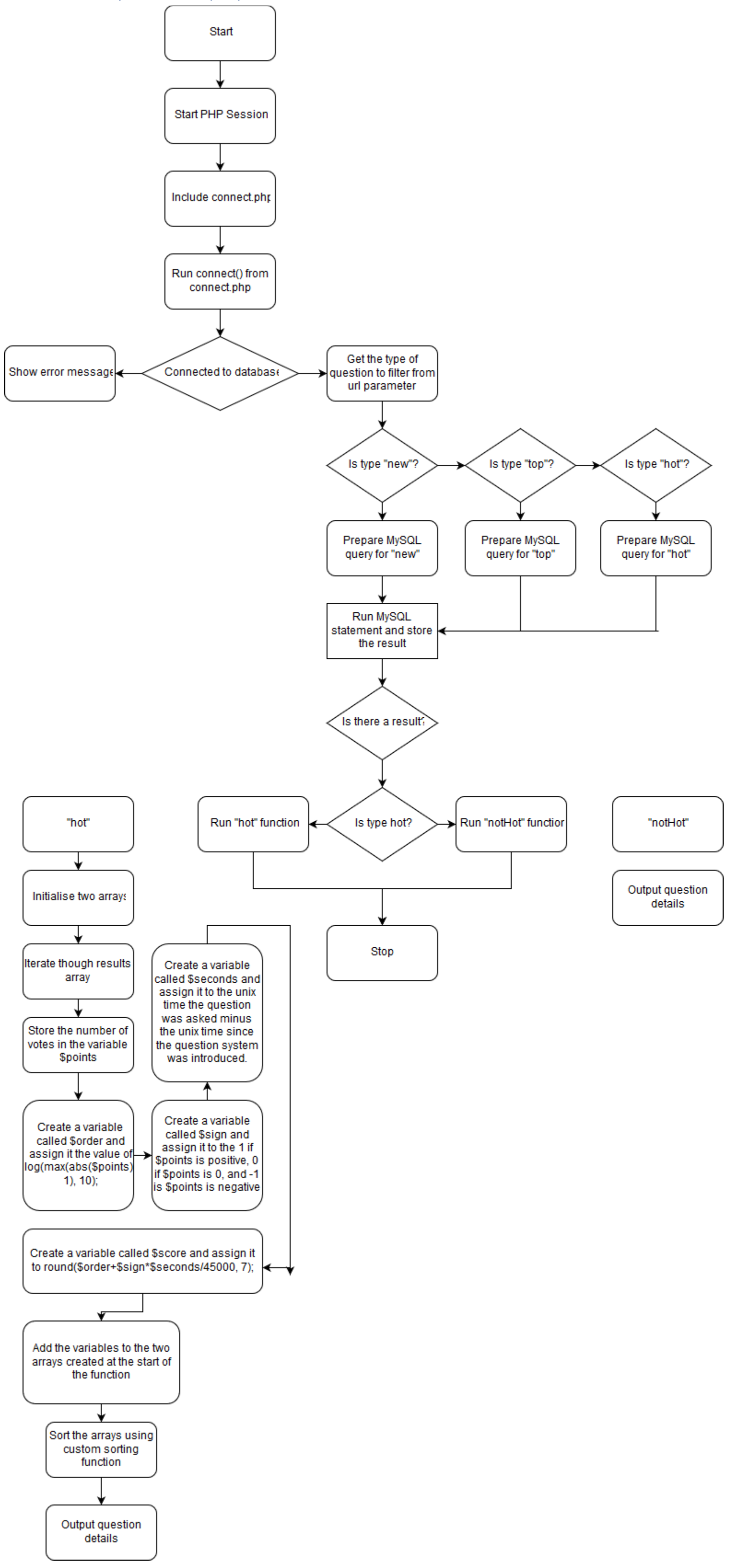
The next block is another decision. If the value stored in the previous block states that the user's IP Address has been blocked from creating user accounts, then the flowchart moves back to the "Show error message" block, then moves to the STOP block, ending the script. However, if the user isn't blocked, the flowchart continues to the next step.

The next two blocks in the flow chart are both decisions representing validation. The first checks that all the fields were filled in using an is empty check, and the second checks that the email address is valid. Note that in the final version of my solution the email validation will be performed using RegEx (Regular Expression) instead of simply checking for an "@" symbol for better results.

Next, the flowchart prepares another MySQLi query to check if the username is already in use. If it isn't, the flowchart moves onto the next block, but if it is, it moves onto the "Show error message" block and then onto the STOP block where the script stops executing.

Providing the username is deemed to be unique, the flowchart moves onto the next block, where it creates another MySQLi query to insert the new user into the database. It then runs this query and providing there were no errors, it logs the user in using the SESSION_ variables and performs a page redirect to the homepage. If there was an error, it moves again to the "Show error message" block before reaching the STOP and exiting.

downloadquestion.php


Start
Start PHP Session
Include connect.php
Run connect() from connect.php
Connected to database
Show error message
Get the type of question to filter from url parameter
Is type "new"?
Is type "top"?
Is type "hot"?
Prepare MySQL query for "new"
Prepare MySQL query for "top"
Prepare MySQL query for "hot"
Run MySQL statement and store the result
Is there a result?
Is type hot?
Run "hot" function
Run "notHot" function
Stop
"hot"
Initialise two arrays
Iterate though results array
Store the number of votes in the variable $points
Create a variable called $order and assign it the value of log(max(abs($points), 1), 10);
Create a variable called $sign and assign it to the 1 if $points is positive, 0 if $points is 0, and -1 is $points is negative
Create a variable called $seconds and assign it to the unix time the question was asked minus the unix time since the question system was introduced.
Create a variable called $score and assign it to round($order+$sign*$seconds/45000, 7);
Add the variables to the two arrays created at the start of the function
Sort the arrays using custom sorting function
Output question details
"notHot"
Output question details

The first four blocks on this flowchart are identical to the first four blocks in the processsignup flowchart so I won't write about them multiple times.

The first new block is the "Get type of question from URL parameter". This is a very simple block that creates a new variable called type and assigns it to $_GET["type"], which is the value of the URL parameter called "type".

Next come three decision blocks. These represent and "if->else if->else if" chain. The purpose of these blocks is to prepare a different MySQLi query for each of the three different question filters: "new", "top" and "hot". For "new" and "top", theses queries make up all the processing and filtering needed to sort the relevant questions by these categories. This is because of the MySQLi ORDER BY keywords, which allow me to ORDER BY \`id\` DESC for the "new" questions or ORDER BY \`score\` DESC for the "top" filter. The "hot" however needs processing in a separate function later on.

Next, the query is run, and the flowchart checks if it returns a result. If there isn't, the flowchart will exit. No error message is displayed because this is a processing page rather than a page the user will ever see. If there is a result, then the flowchart continues to the next block.

The next block is another check to see if the type variable contains the value "hot". The flowchart then proceeds to run either the "hot" function (if type = "hot") or the "notHot" function (if type ≠ "hot). This is the end of the main branch of the flowchart, with the rest of the output coming from the "hot" and "notHot" functions respectively.

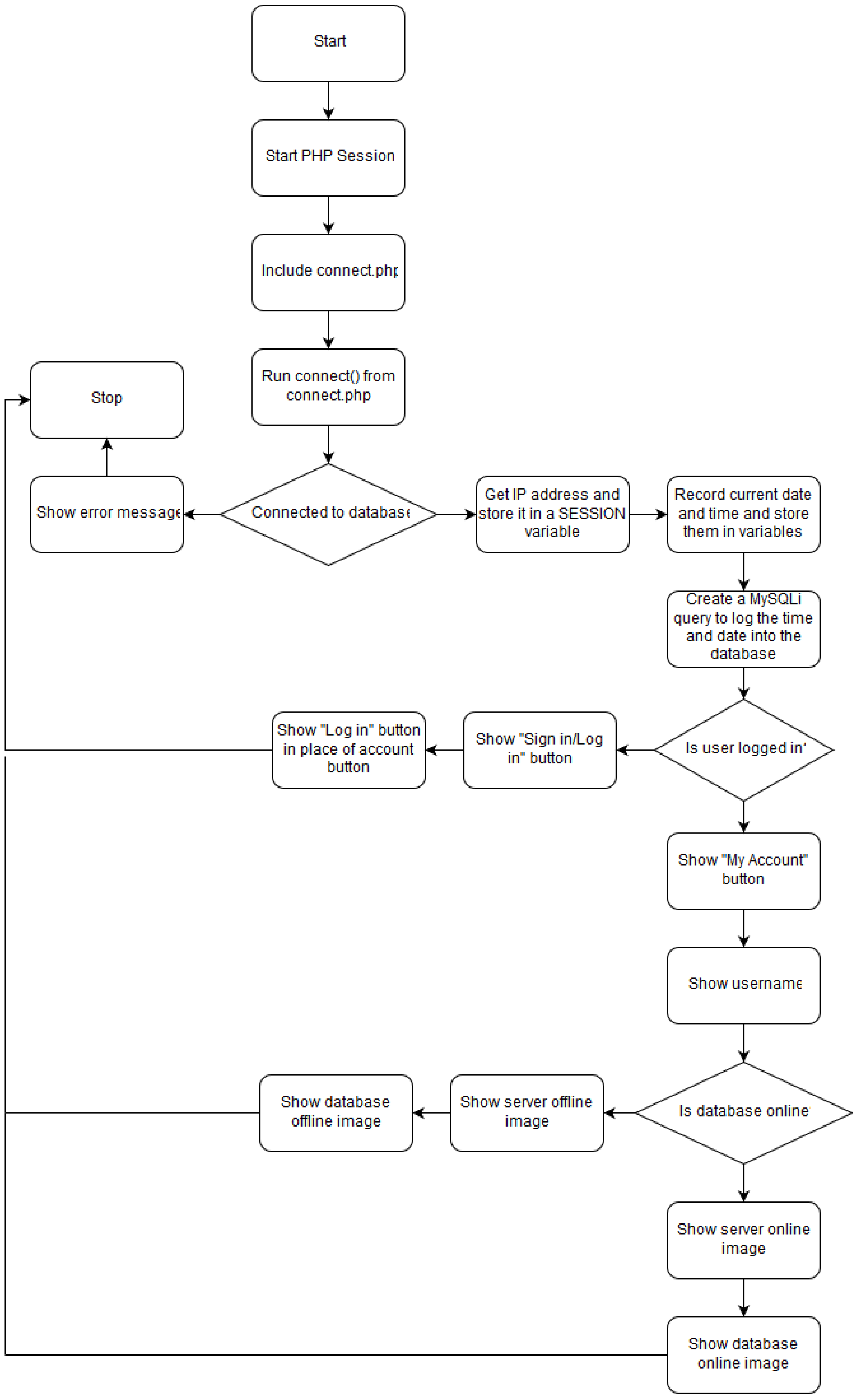
The "hot" function starts off by initialising two arrays in the first block. Next, the questions returned from the database are iterated over, and for each iteration, the following occurs:

- The number of votes is stored in a variable called $points
- A variable called $order is created and assigned the value of the formula (in PHP) log(max(abs($points), 1), 10).
- A variable called $sign is created and assigned to a representation of the sign of the number (1, 0, -1)
- A variable called $seconds is created and assigned the value of the UNIX time it was created minus the UNIX time the questions database was created. To give the number of seconds since the question was first asked.
- A $score variable is created and assigned to the value of the formula (in PHP) round($order+$sign*$seconds/45000, 7). This is used to order the questions, based on descending $score.
- Populate the arrays with all this data

Then, the "hot" function sorts the arrays based on the $score attribute calculated for each question. It then outputs the questions in a format readable by other scripts.

The "notHot" function requires no processing, and simply iterates over and outputs the already sorted questions in a format readable by other scripts.

index


Start
Start PHP Session
Include connect.php
Run connect() from connect.php
Connected to database
Show error message
Stop
Get IP address and store it in a SESSION variable
Record current date and time and store them in variables
Create a MySQLi query to log the time and date into the database
Is user logged in
Show "Sign in/Log in" button
Show "Log in" button in place of account button
Show "My Account" button
Show username
Is database online
Show server offline image
Show database offline image
Show server online image
Show database online image

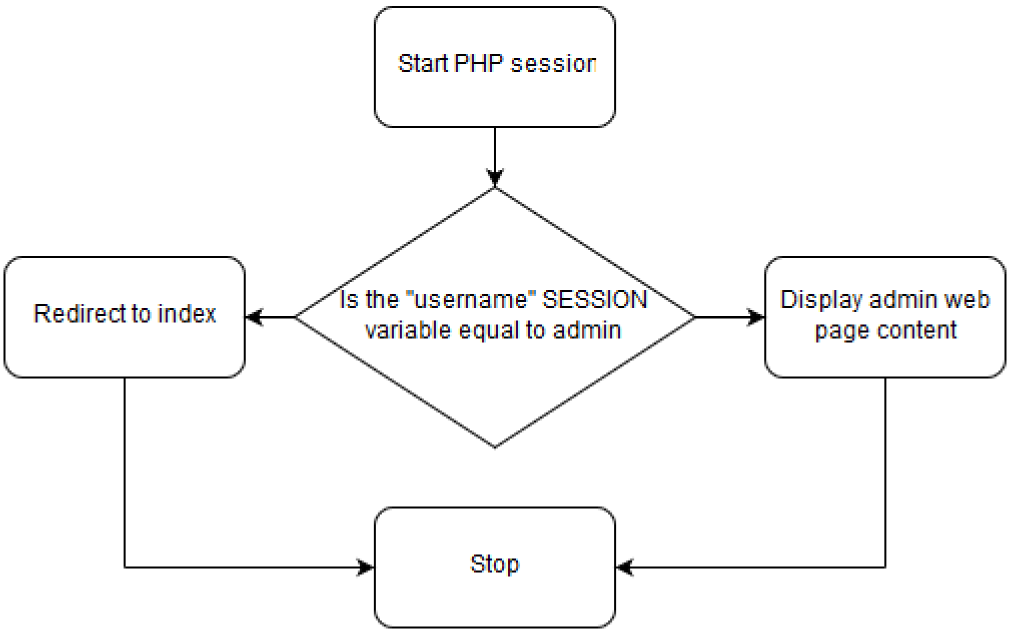
This flowchart represents the homepage of the website. Note that the final design will not include the server and database status images. As with most of the flowcharts, the first four blocks are repeated in most files and therefore are only described once, in the first flowchart.

The first unique block of the flowchart gets the IP Address of the client and stores it as a SESSION_ variable. The next block is also related, as it stores the current time and data of the client's connection as SESSION_ variables. Next, the flowchart prepares a MySQLi query to log the client's IP Address, the time and the data that were recorded in the previous block. This query is then executed.

Then, the flowchart reaches a decision block. If the user is logged in, it should display the sign out button on the navigation bar, but if no user is logged in the button should take the user to a sign up / log in page.

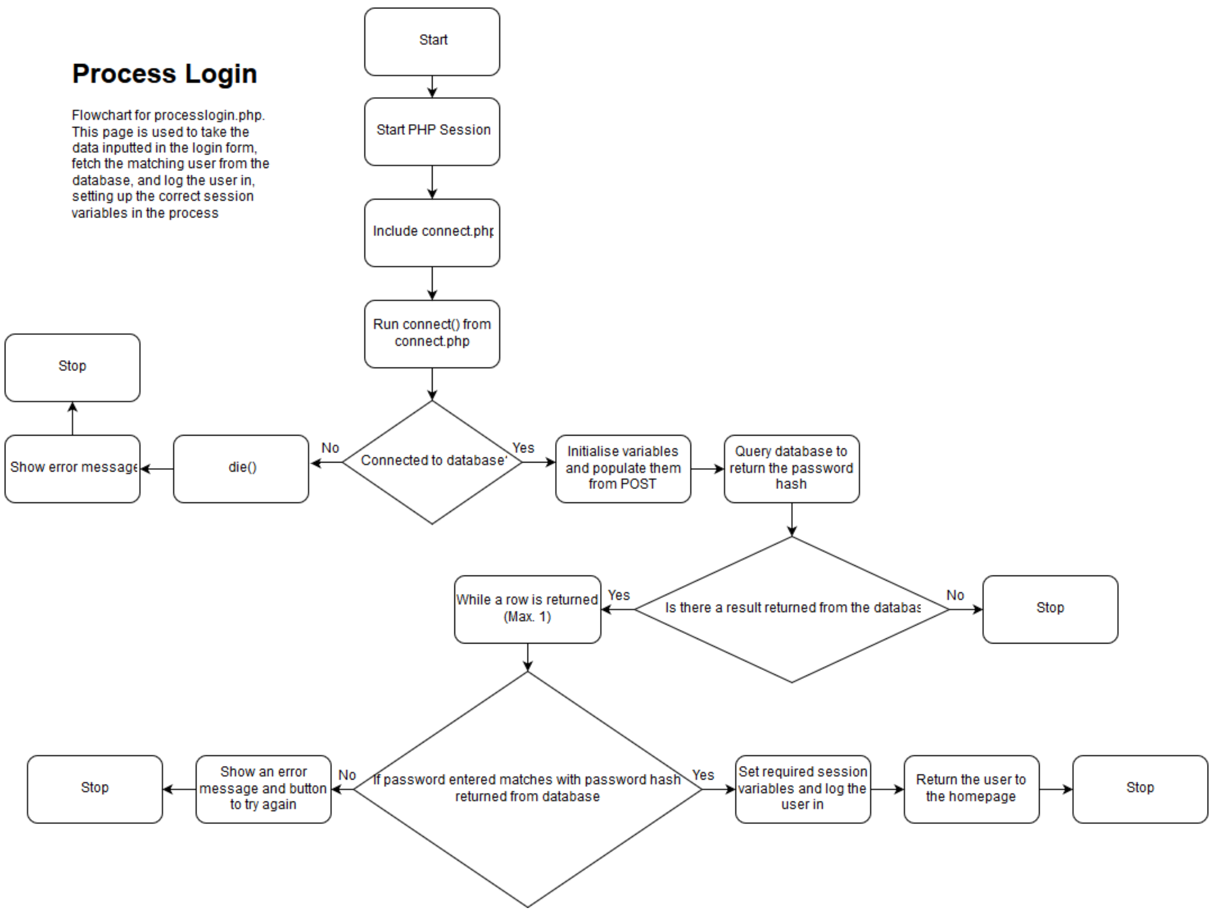
The final decision on the page only existed in an early draft design. The idea was to have status images on the footer displaying if the database was online. I removed this feature after a redraft, but the flowchart was already made. The flowchart states that if the database is online, it should draw the "database online" image and the "server online" image. If the database was offline it should draw the "database offline" image and the "server online" image. There was no "server offline" image because if the server was offline you wouldn't be able to connect to the website to see it.

admin


Start PHP session
Is the "username" SESSION variable equal to admin
Redirect to index
Display admin web page content
Stop

This is probably the simplest flowchart in the entire program. It simply checks if the current logged in user is the admin account and redirects the user if they're not. It does this by checking the value of the SESSION_["username"] SESSION variable. If the user is logged in, it will load the HTML content of the page.

processlogin


Process Login
Flowchart for processlogin.php. This page is used to take the data inputted in the login form, fetch the matching user from the database, and log the user in, setting up the correct session variables in the process
Start
Start PHP Session
Include connect.php
Run connect() from connect.php
Connected to database?
No
die()
Show error message
Stop
Yes
Initialise variables and populate them from POST
Query database to return the password hash
Is there a result returned from the database
No
Stop
Yes
While a row is returned (Max. 1)
If password entered matches with password hash returned from database
No
Show an error message and button to try again
Stop
Yes
Set required session variables and log the user in
Return the user to the homepage
Stop

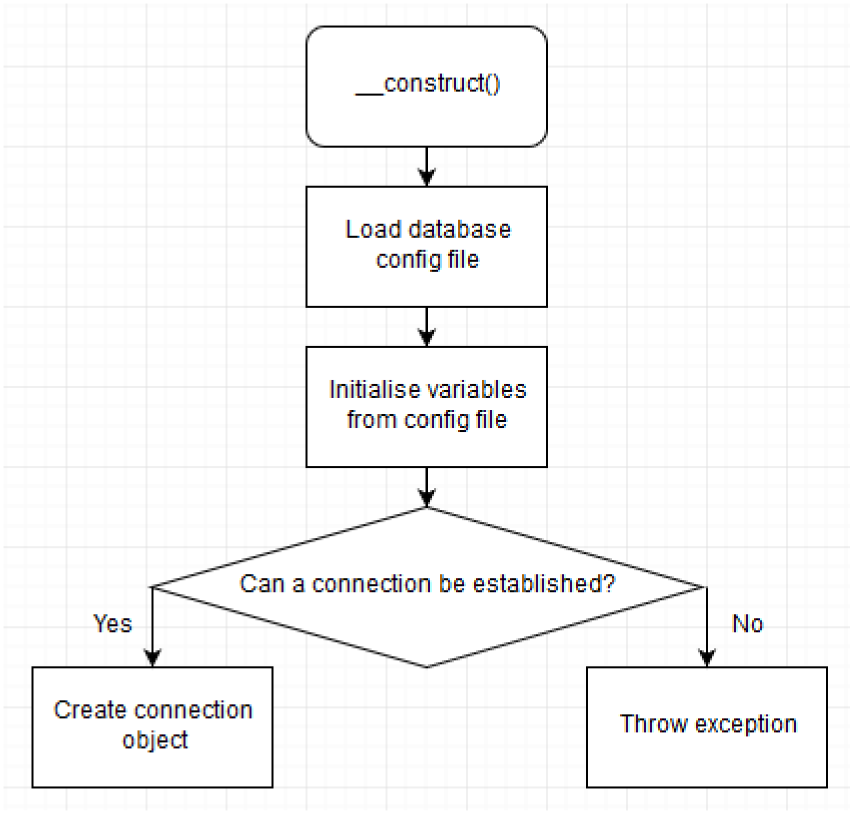
As with all the other flowcharts, the first four blocks appear in almost all pages as they are used to establish and test a connection to the database. These blocks are discussed in detail in an earlier flow-chart. If the connection to the database fails to be established, the flow-chart follows the simple path of calling the built-in die() function to exit the script, showing an error message in the process. However, if the connection is successful (which it should be), the first stage of the script is to initialise some variables used in the script and populate them with POST variables from the login form on the previous page. This is one way data can be passed between pages in PHP.

Next, the script prepares a MySQLi query to select the password_hash attribute from the database for a user matching the parameters entered in the login form on the previous page. If there are no matching results returned from the database, then the user's login details must be incorrect, and the flow-chart continues to the next stage, where is shows an error message, a button to go back to the previous page, and then exits. However, if any results are returned from the database (Up to a maximum of 1 result), the flow-chart called the passwordVerify function to check that the password the user attempted to log in with matches the hash returned from the database.

If the hash doesn't match, the script shows an error message to the user saying that their username/password combination doesn't exist and a button to redirect them back to the login page. However, if the hash matches the password the user entered is the correct password for their username and the script log them in. It does this by setting the value of some session variables such as SESSION_["username" and SESSION_["id"] in order to log the user in, then redirects them back to the homepage.

Database

__construct()


__construct()
Load database config file
Initialise variables from config file
Can a connection be established?
Yes
No
Create connection object
Throw exception

__construct is the constructor for the Database class. The class constructor is a function that is ran when the class is first initialised, e.g. when this a new Database is created as shown below, __construct is called.

Database $database = new Database();

The main purpose of a class constructor is to create and initialise any variables and objects that are needed by the rest of the class. The constructor for the Database class has the added job of loading the database connection variables from a separate file on the server that is not accessible to the public and is not on the webserver part of the server. This is because the file contains the verification details for the database which are used for connecting to and changing any aspect of it. Allowing anyone but me to access this file has serious security and privacy issues.

The first block of __construct is to load the config.ini file from an offline directory on the server, which in reality is only a single line function call to the parse_ini_file function with the path of the file as the parameter. This returns an associative array.

The next block however is slightly longer. The purpose of this next block is to read the individual properties from the config file and create variables to store them in. It does this by setting each variable to the value of the $config["property name"], where config is the associative array from the previous block, and the property name is the String name of the property as defined in the ini file.

After these two blocks, the flowchart moves onto a decision block. This checks to see if the script can create a MySQLi connection object using the variables loaded from the ini file in the previous blocks. It may seem counter productive to check the connection before creating it, but this is because the PHP representation of this block and the two possibilities (yes/no) is similar to:

```
if (!$this->connection = new mysqli($db_server, $db_username, $db_password, $db_database)) {
        throw new Exception("Cannot connect);
}
```

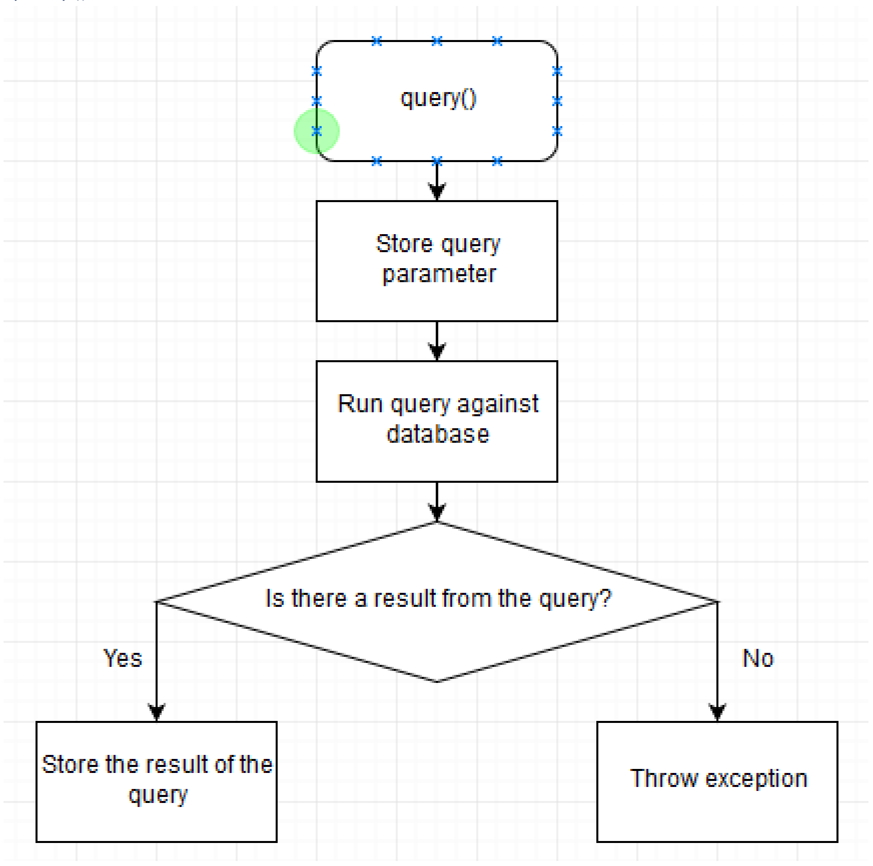
This works because in PHP you can create objects inside if statements. The flow of this if statement is as follows:

Attempt to create a MySQLi connection object in $this->connection, returning true if successful or false if unsuccessful.

If this is successful, the if statement then resembles if (!true) which evaluates to if (true != true) which it doesn't so the MySQLi connection object has been created and the clause of the if statement is just ignored.

If it wasn't successful, the if statement then resembles if (!false) which evaluates to if (false != true) which resolves to if (true) and therefore runs the if clause, throwing a connection failure exception.

query()


query()
Store query parameter
Run query against database
Is there a result from the query?
Yes
No
Store the result of the query
Throw exception

query() is one of the most important functions in the Database class, but is also incredibly simple. The purpose of query() is to execute a MySQL query against the database. It has one parameter, a String called $query, which contains the MySQL query to be executed. It is called by running:

database->query("…….");

where *database* is a Database object that has already been created and initialised (because query is not a static function).

The first block of this function simply stores the query provided in the $query parameter in the instance of the class so that it can be accessed later on. The second block of this function simply calls *$this->connection->query($query);* where *$this* is the instance of the class, *connection* is the MySQLi connection object created in __construct and *query()* is the build in MySQLi->query() function.

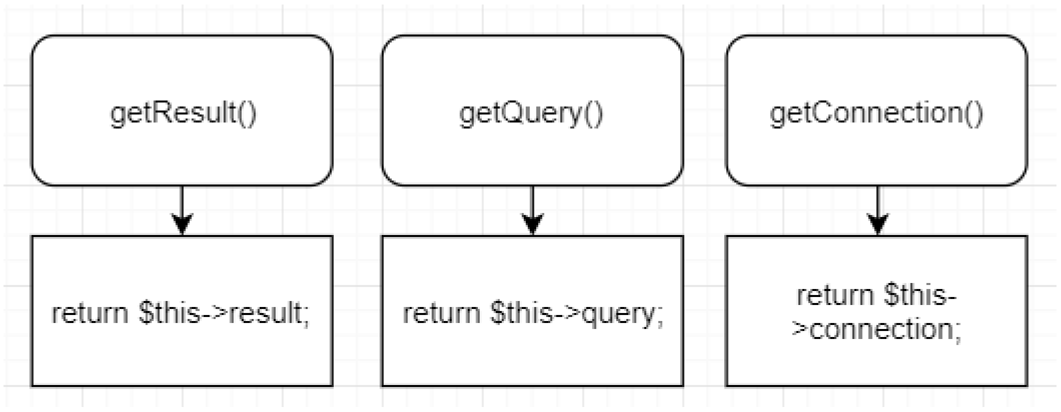
It may seem unnecessary so write a whole class that simply extends pre-existing built-in functions, but the purpose of this class is to provide a simple library tailored towards the needs of my project for interfacing with the database. The class provides error handling, creating a connection, and many more useful functions that remove duplicating code in multiple files and re-implementing error in every class.

The next block of the flowchart is a decision, to run a different branch if the query that just executed returned a result or not. The purpose of this block is to gracefully handle any errors that come from running the query and store the result of the query if it executed successfully. As well as storing the result of the query, the function also needs to return the result if the query was successful. This allows querying the database and getting the result to be a simple one-line call from the rest of the website:

$result = $database->query("….");

Where $database is a pre-initialised Database object and "…" is a MySQL query.

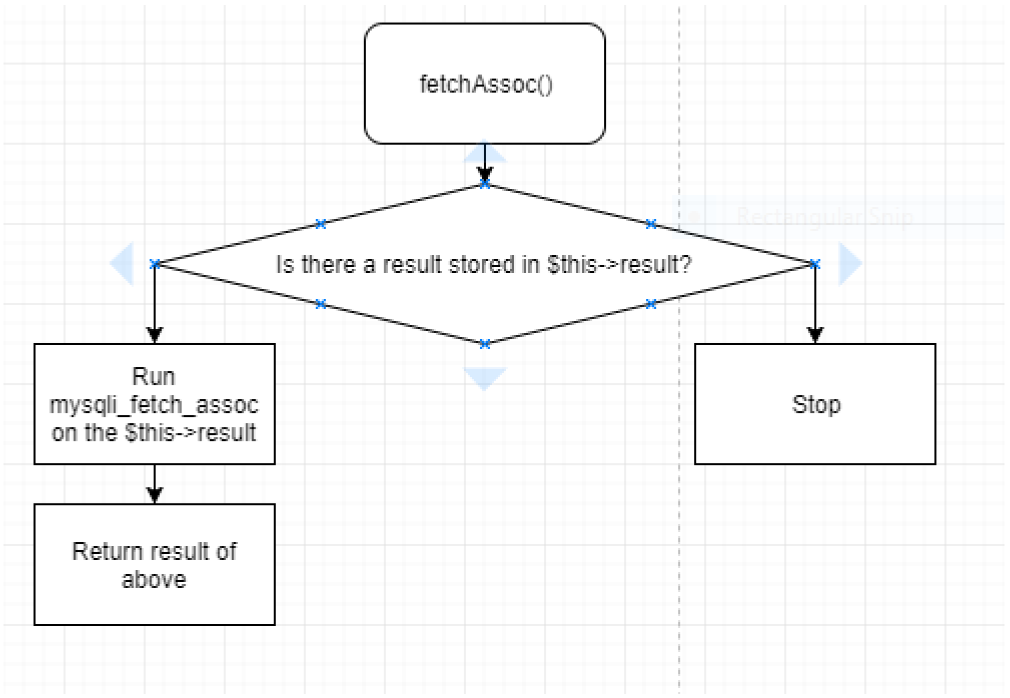
getResult, getQuery and getConnection



getResult, getQuery and getConnection are both a special type of function known as a Getter function. The purpose of Getter functions is to provide an OOP (Object Oriented Programming) way of accessing a variable from an instance of a class. Getter functions allow the programmer to call *$myObject->getName()* to get the name variable held within that instance of the object's class. This is different to a more procedural style of programming, where the programmer would call *Object.name* to get the name of the object. This however doesn't allow for multiple instances of the object, and although it can be refactored to allow multiple instances, it is best practice to use OOP in modern PHP.

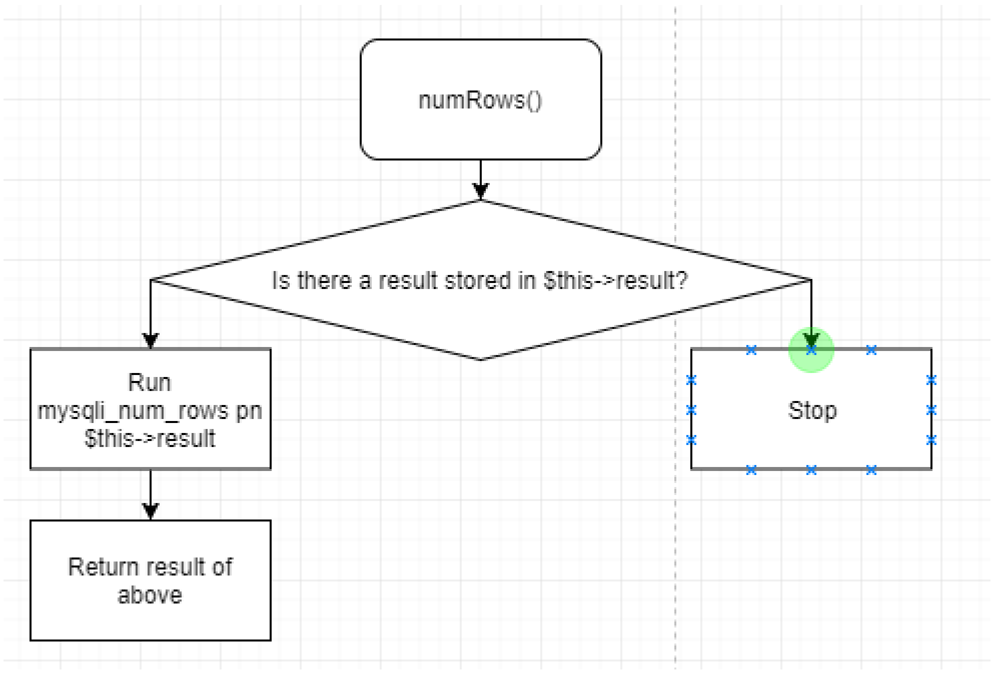
These are the only getter functions used in the Database class, although other functions such as User have a lot more. getResult simply returns the last query result from an instance of the Database class (or NULL if there was an error/no query has been performed yet), getQuery returns the last query executed against the database for an instance of the class, and getConnection returns the actual MySQLi connection object used by an instance of the class.

fetchAssoc


fetchAssoc()
Is there a result stored in $this->result?
Run mysqli_fetch_assoc on the $this->result
Return result of above
Stop

fetchAssoc is a function used to provide an interface to the PHP function mysqli_fetch_assoc. The purpose of the said function is to produce an associative array of the data returned from a MySQL query. An associative array in PHP is an array where the data in the array can be accessed by an identifier. The resulting associative array returned by this function uses the database table field names as the identifiers. fetchAssoc first checks if there is a result stored in the $this-result. If there is then it runs the function and returns the result. If not, then there is no return output from the database. This is in order to stop the function crashing or throwing an exception if the parameter is null.

numRows


numRows()
Is there a result stored in $this->result?
Run mysqli_num_rows pn $this->result
Return result of above
Stop

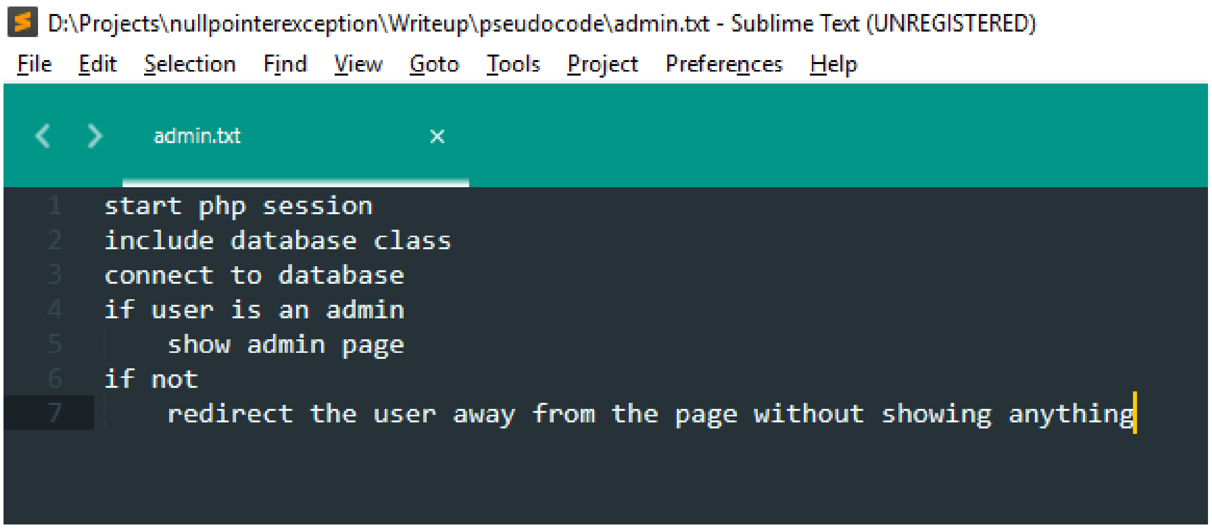
This is the final function in the Database class. As with fetchAssoc, it provides an interface to a MySQLi function, mysqli_num_rows, on the query result stored in $this->query. The purpose of this function is to return a count of the number of rows returned from a database query. Like fetchAssoc it also first performs a null check to prevent the function crashing or throwing an exception if no data is supplied. Knowing the number of items returned from the database is important for functions that either iterate over the data (although this could be negated by simply using a foreach loop), or as cleaner alternative to the built in size_of() function. As with fetchAssoc, it is not strictly necessary for the finished program to function, but as the Database class is designed to provide an interface/wrapper for a collection of the MySQLi class, it makes sense to include this incredibly simple function in the Database class.

Pseudocode

Alongside the flowcharts I also used pseudocode to mock-up most of the scripts. This can be seen bellow:

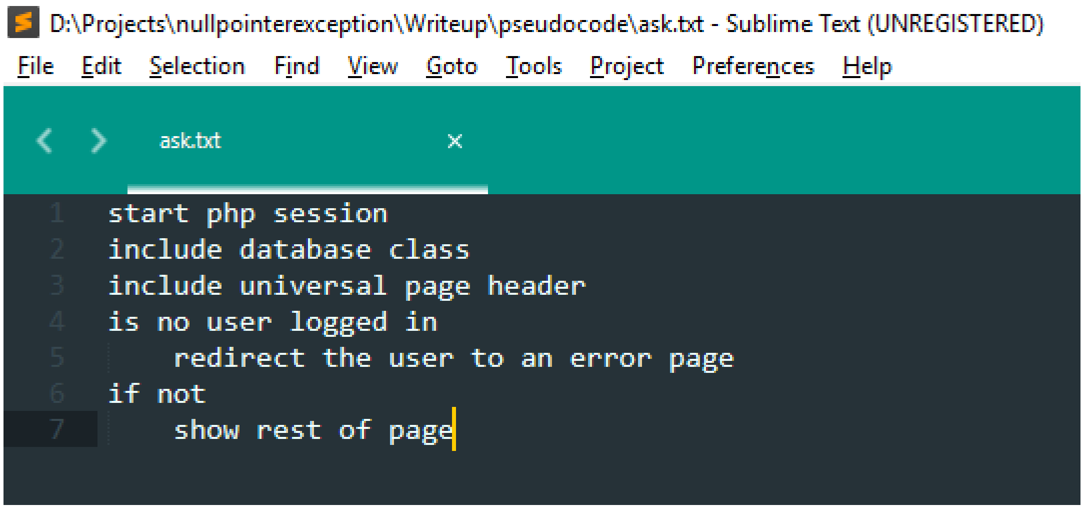
NullPointerException - Your questions. Answered.

admin


D:\Projects\nullpointerexception\Writeup\pseudocode\admin.txt - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
admin.txt

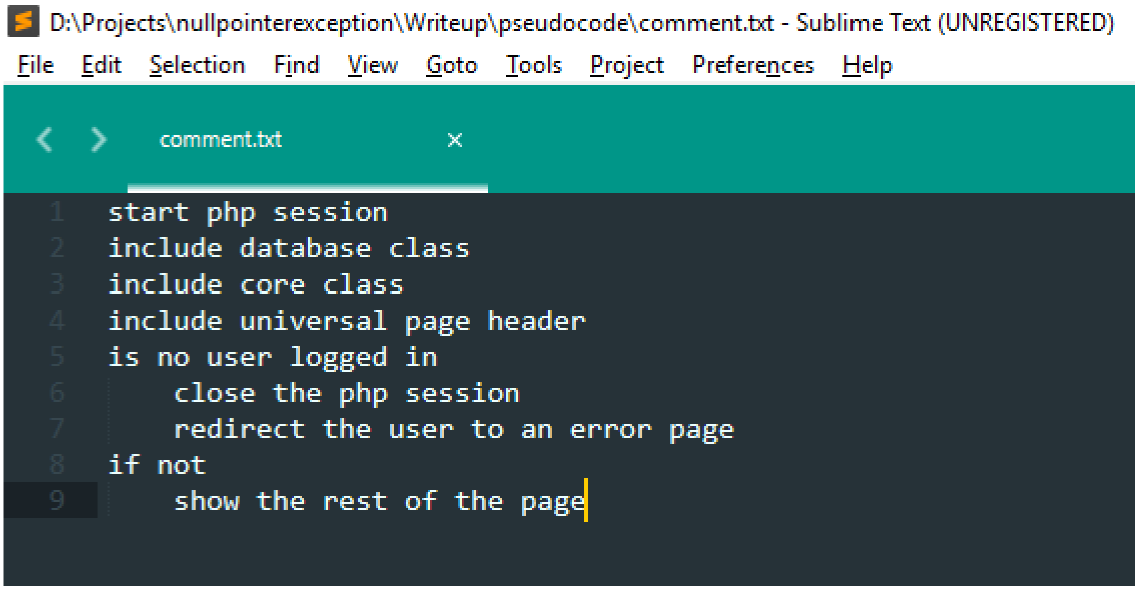
```
start php session
include database class
connect to database
if user is an admin
    show admin page
if not
    redirect the user away from the page without showing anything
```

ask


D:\Projects\nullpointerexception\Writeup\pseudocode\ask.txt - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
ask.txt

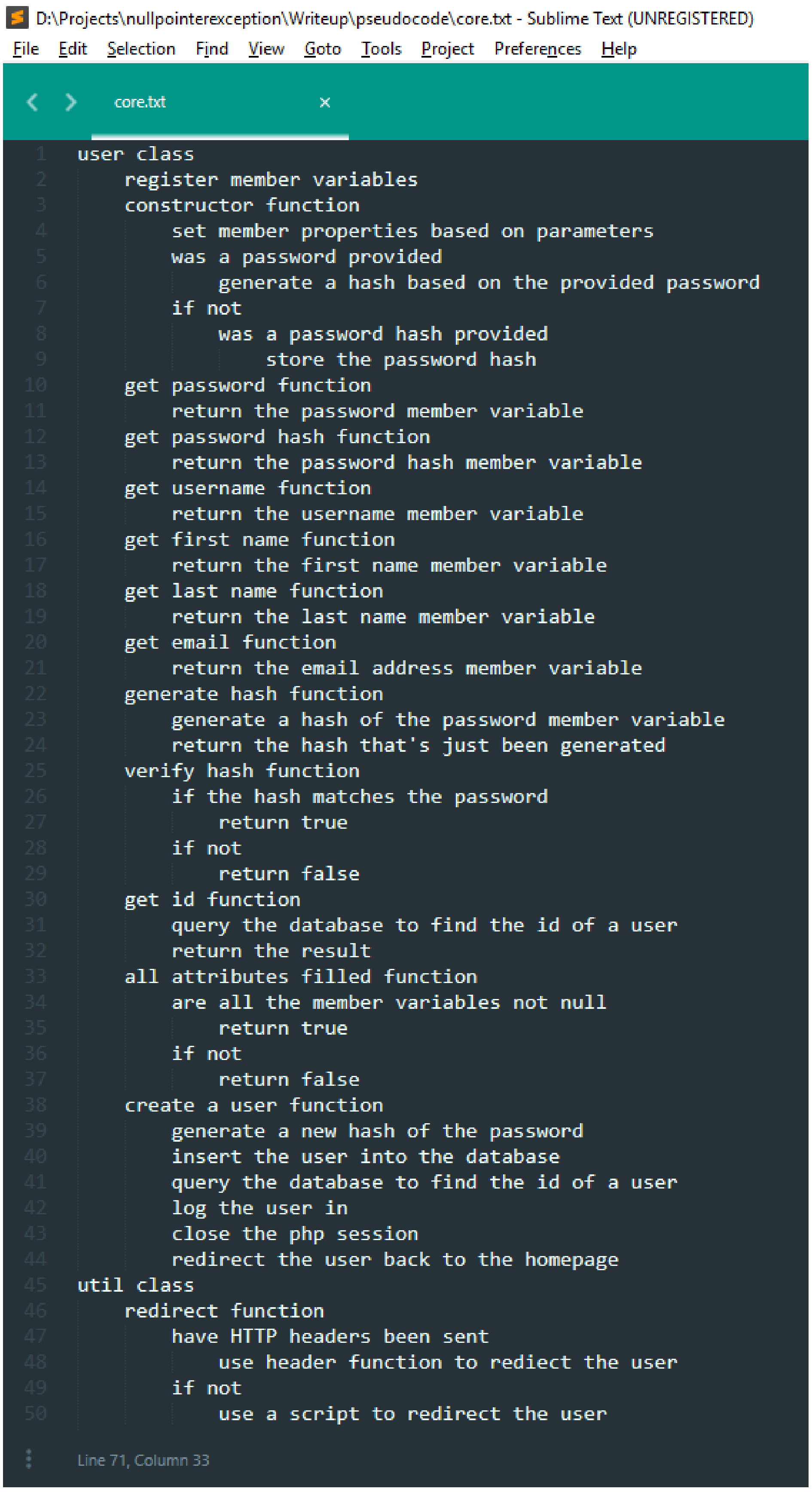
```
start php session
include database class
include universal page header
is no user logged in
    redirect the user to an error page
if not
    show rest of page
```

comment


D:\Projects\nullpointerexception\Writeup\pseudocode\comment.txt - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
comment.txt

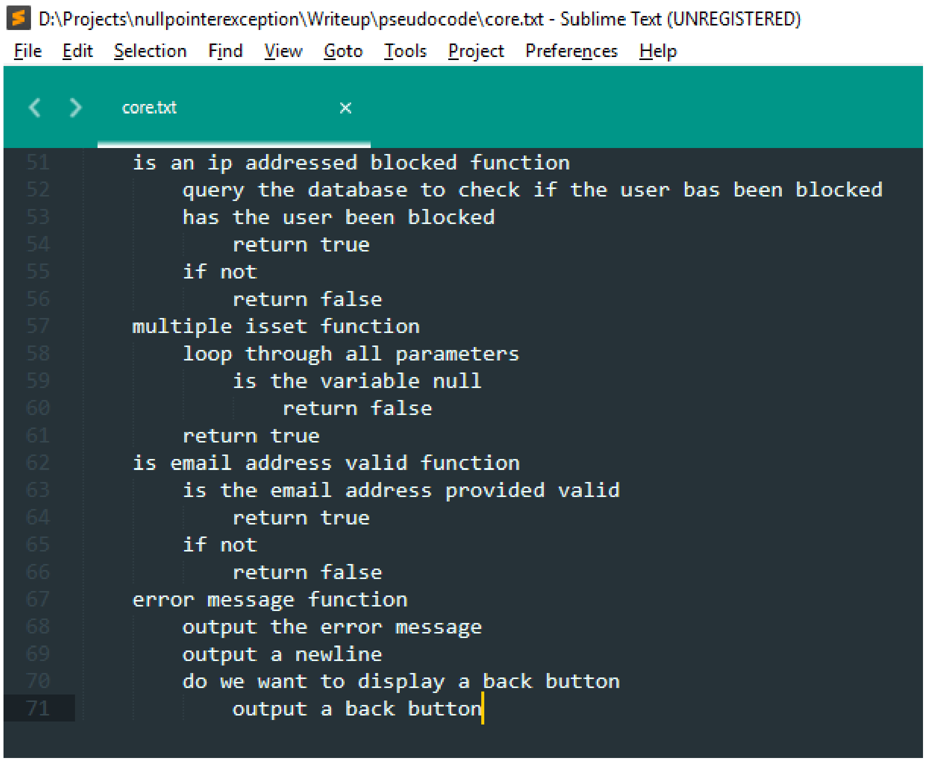
```
start php session
include database class
include core class
include universal page header
is no user logged in
    close the php session
    redirect the user to an error page
if not
    show the rest of the page
```

core (part one)


D:\Projects\nullpointerexception\Writeup\pseudocode\core.txt - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
core.txt

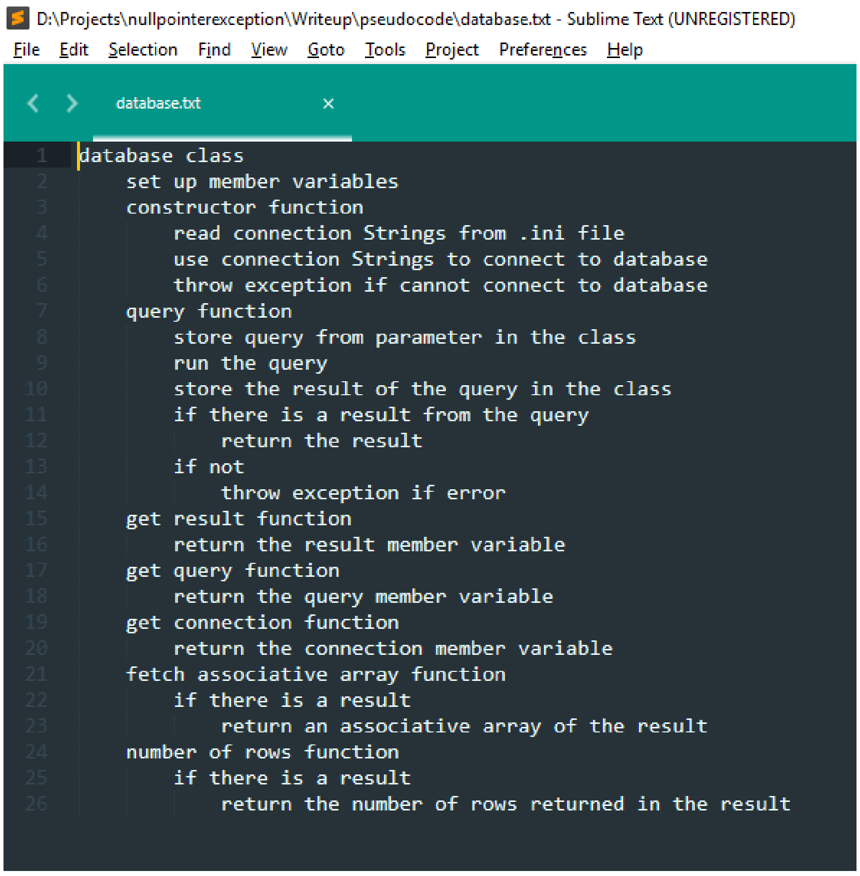
```
user class
    register member variables
    constructor function
        set member properties based on parameters
        was a password provided
            generate a hash based on the provided password
        if not
            was a password hash provided
                store the password hash
    get password function
        return the password member variable
    get password hash function
        return the password hash member variable
    get username function
        return the username member variable
    get first name function
        return the first name member variable
    get last name function
        return the last name member variable
    get email function
        return the email address member variable
    generate hash function
        generate a hash of the password member variable
        return the hash that's just been generated
    verify hash function
        if the hash matches the password
            return true
        if not
            return false
    get id function
        query the database to find the id of a user
        return the result
    all attributes filled function
        are all the member variables not null
            return true
        if not
            return false
    create a user function
        generate a new hash of the password
        insert the user into the database
        query the database to find the id of a user
        log the user in
        close the php session
        redirect the user back to the homepage
util class
    redirect function
        have HTTP headers been sent
            use header function to rediect the user
        if not
            use a script to redirect the user
```

core (part two)


D:\Projects\nullpointerexception\Writeup\pseudocode\core.txt - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
core.txt

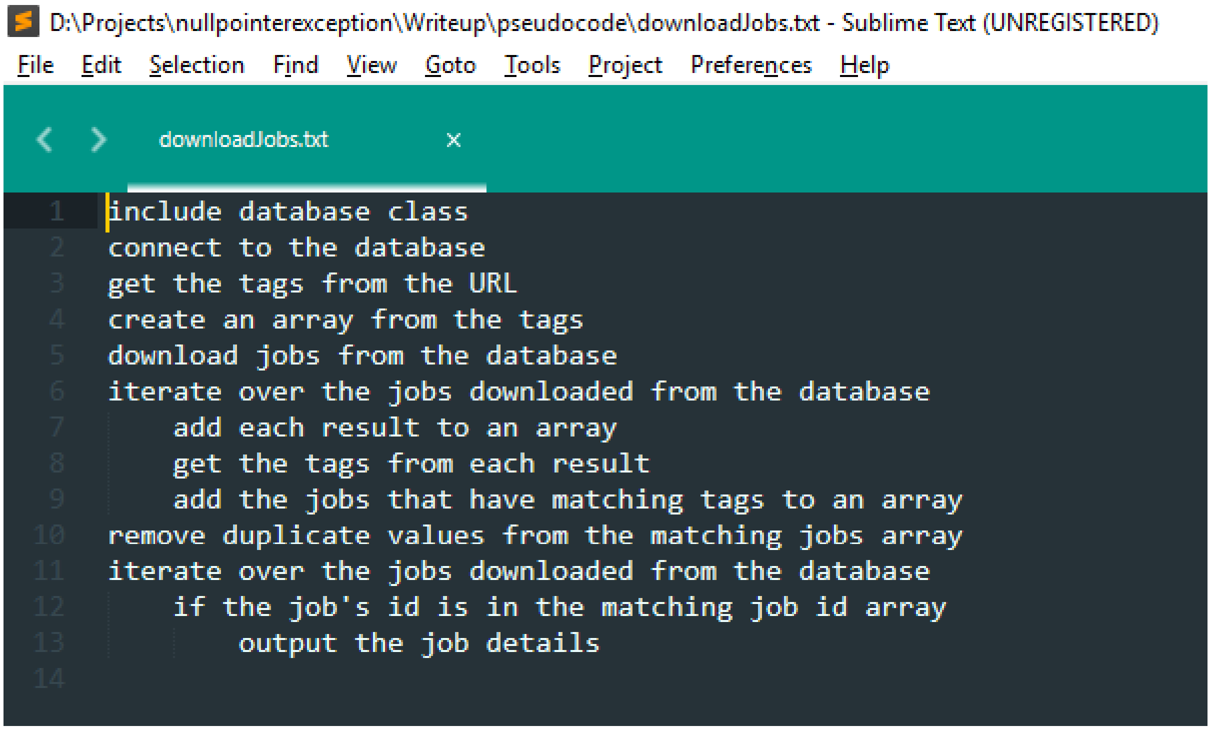
```
    is an ip addressed blocked function
        query the database to check if the user bas been blocked
        has the user been blocked
            return true
        if not
            return false
    multiple isset function
        loop through all parameters
            is the variable null
                return false
        return true
    is email address valid function
        is the email address provided valid
            return true
        if not
            return false
    error message function
        output the error message
        output a newline
        do we want to display a back button
            output a back button
```

database


D:\Projects\nullpointerexception\Writeup\pseudocode\database.txt - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
database.txt

```
database class
    set up member variables
    constructor function
        read connection Strings from .ini file
        use connection Strings to connect to database
        throw exception if cannot connect to database
    query function
        store query from parameter in the class
        run the query
        store the result of the query in the class
        if there is a result from the query
            return the result
        if not
            throw exception if error
    get result function
        return the result member variable
    get query function
        return the query member variable
    get connection function
        return the connection member variable
    fetch associative array function
        if there is a result
            return an associative array of the result
    number of rows function
        if there is a result
            return the number of rows returned in the result
```

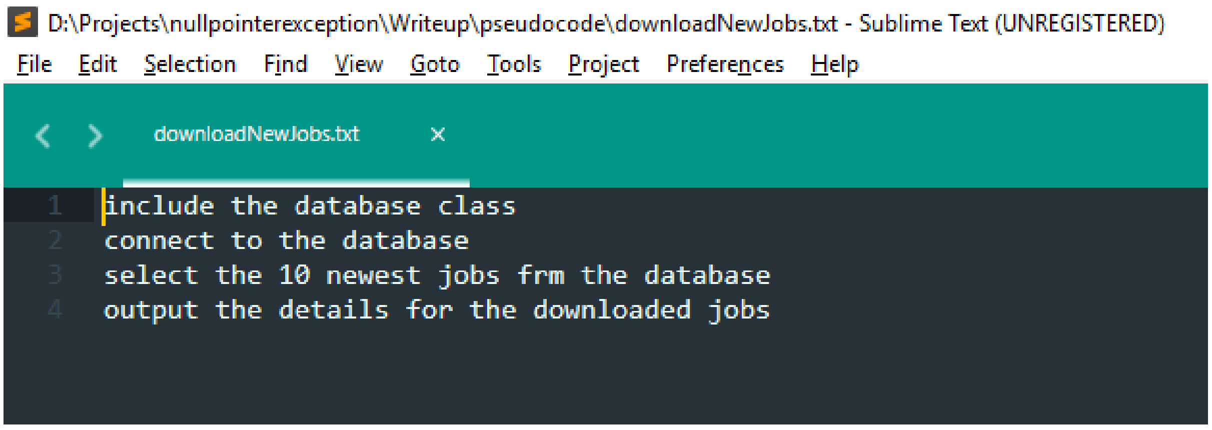
downloadJobs


D:\Projects\nullpointerexception\Writeup\pseudocode\downloadJobs.txt - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
downloadJobs.txt

```
include database class
connect to the database
get the tags from the URL
create an array from the tags
download jobs from the database
iterate over the jobs downloaded from the database
    add each result to an array
    get the tags from each result
    add the jobs that have matching tags to an array
remove duplicate values from the matching jobs array
iterate over the jobs downloaded from the database
    if the job's id is in the matching job id array
        output the job details
```

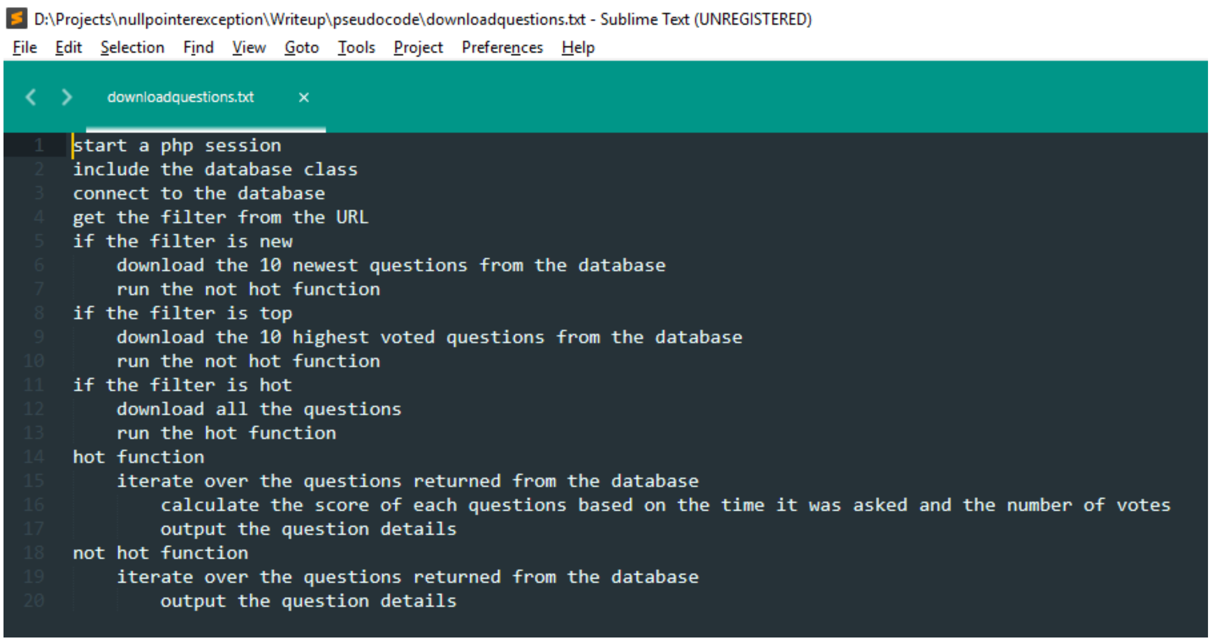
NullPointerException - Your questions. Answered.

downloadNewJobs


D:\Projects\nullpointerexception\Writeup\pseudocode\downloadNewJobs.txt - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
downloadNewJobs.txt

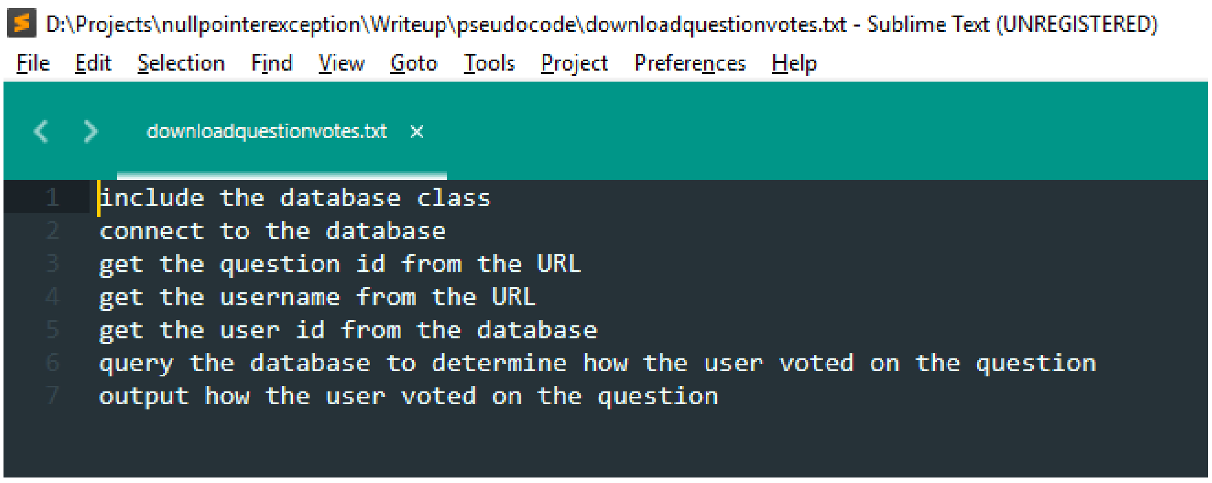
```
include the database class
connect to the database
select the 10 newest jobs frm the database
output the details for the downloaded jobs
```

downloadquestions


D:\Projects\nullpointerexception\Writeup\pseudocode\downloadquestions.txt - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
downloadquestions.txt

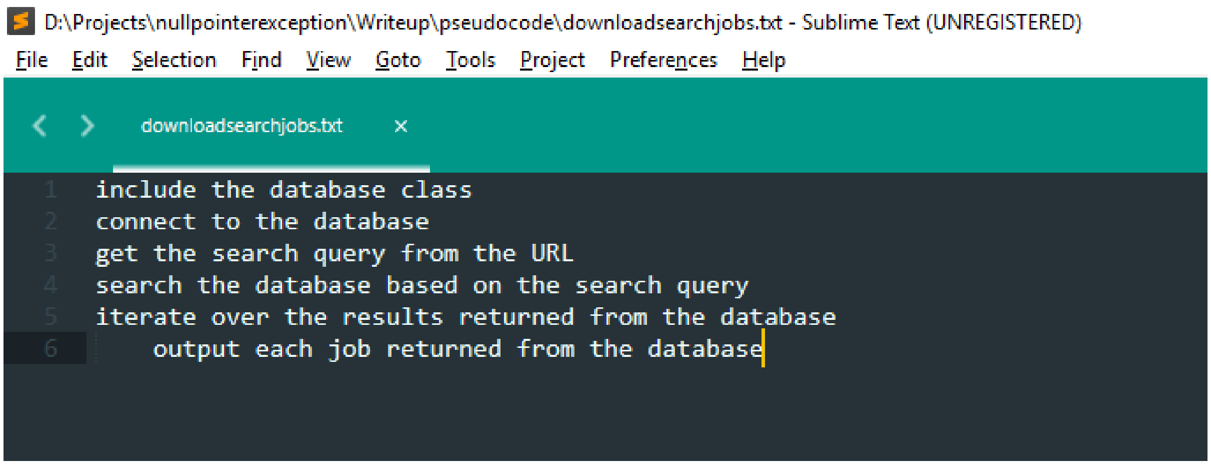
```
start a php session
include the database class
connect to the database
get the filter from the URL
if the filter is new
    download the 10 newest questions from the database
    run the not hot function
if the filter is top
    download the 10 highest voted questions from the database
    run the not hot function
if the filter is hot
    download all the questions
    run the hot function
hot function
    iterate over the questions returned from the database
        calculate the score of each questions based on the time it was asked and the number of votes
        output the question details
not hot function
    iterate over the questions returned from the database
        output the question details
```

downloadquestionvotes


D:\Projects\nullpointerexception\Writeup\pseudocode\downloadquestionvotes.txt - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
downloadquestionvotes.txt

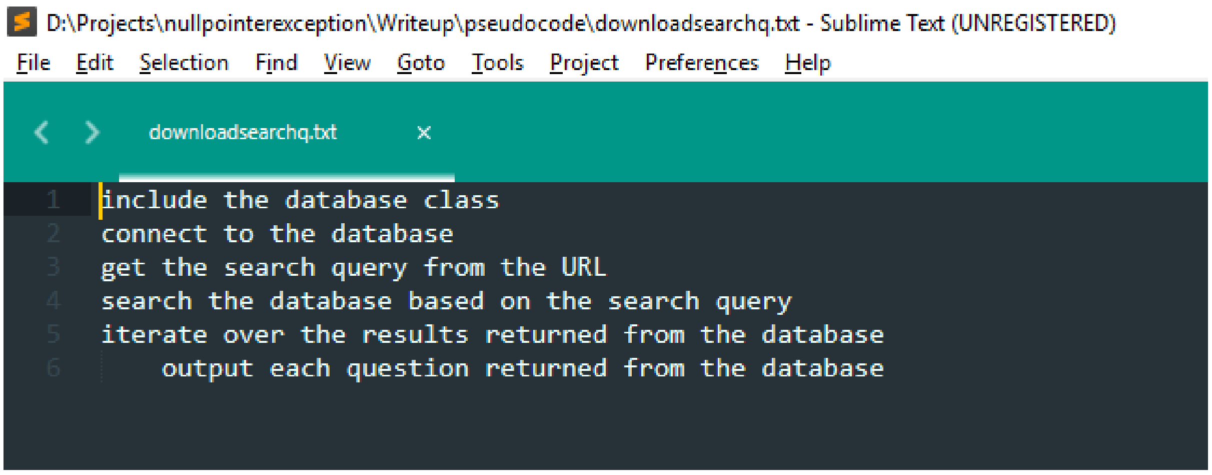
```
include the database class
connect to the database
get the question id from the URL
get the username from the URL
get the user id from the database
query the database to determine how the user voted on the question
output how the user voted on the question
```

downloadsearchjobs


D:\Projects\nullpointerexception\Writeup\pseudocode\downloadsearchjobs.txt - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
downloadsearchjobs.txt

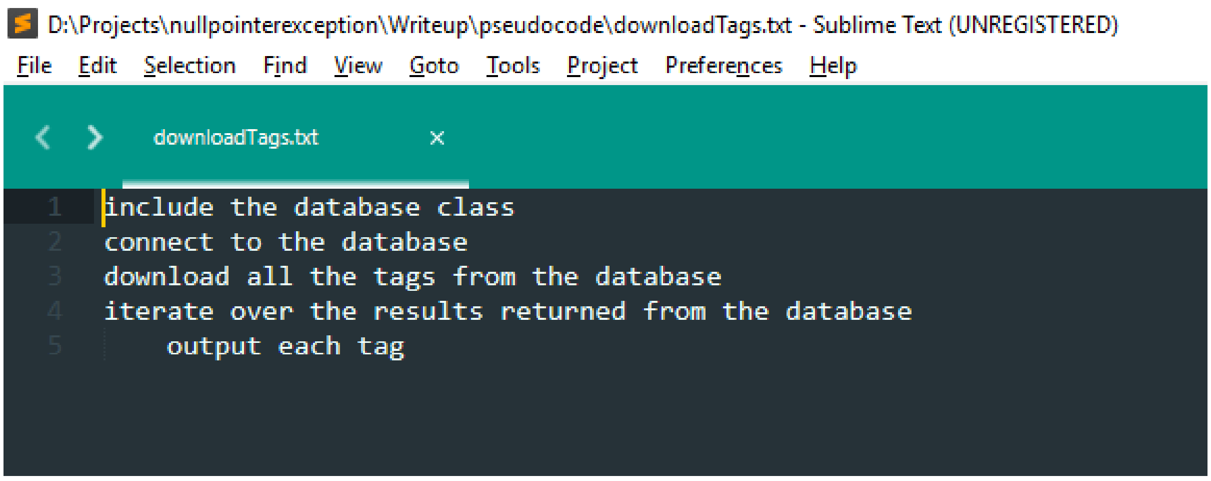
```
include the database class
connect to the database
get the search query from the URL
search the database based on the search query
iterate over the results returned from the database
    output each job returned from the database
```

downloadsearchq


D:\Projects\nullpointerexception\Writeup\pseudocode\downloadsearchq.txt - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
downloadsearchq.txt

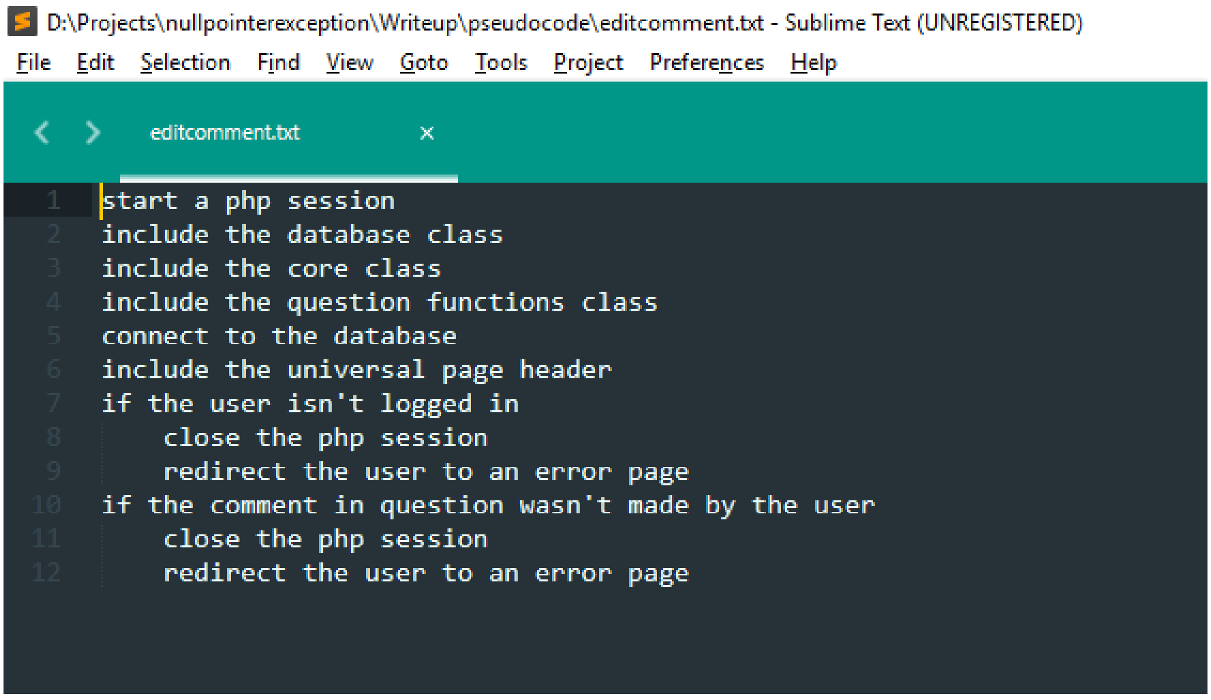
```
include the database class
connect to the database
get the search query from the URL
search the database based on the search query
iterate over the results returned from the database
    output each question returned from the database
```

downloadTags


D:\Projects\nullpointerexception\Writeup\pseudocode\downloadTags.txt - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
downloadTags.txt

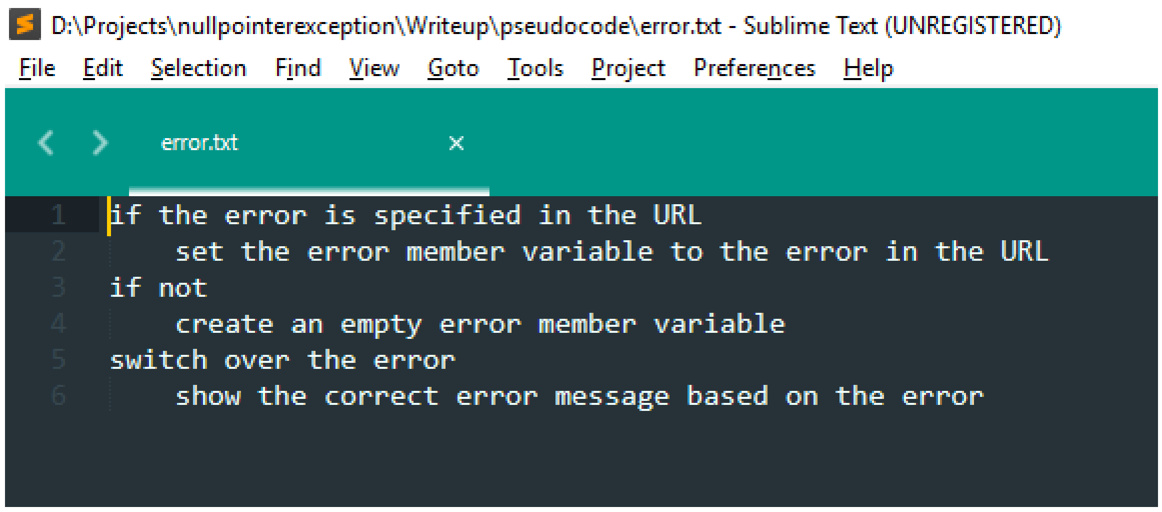
```
include the database class
connect to the database
download all the tags from the database
iterate over the results returned from the database
    output each tag
```

editcomment


D:\Projects\nullpointerexception\Writeup\pseudocode\editcomment.txt - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
editcomment.txt

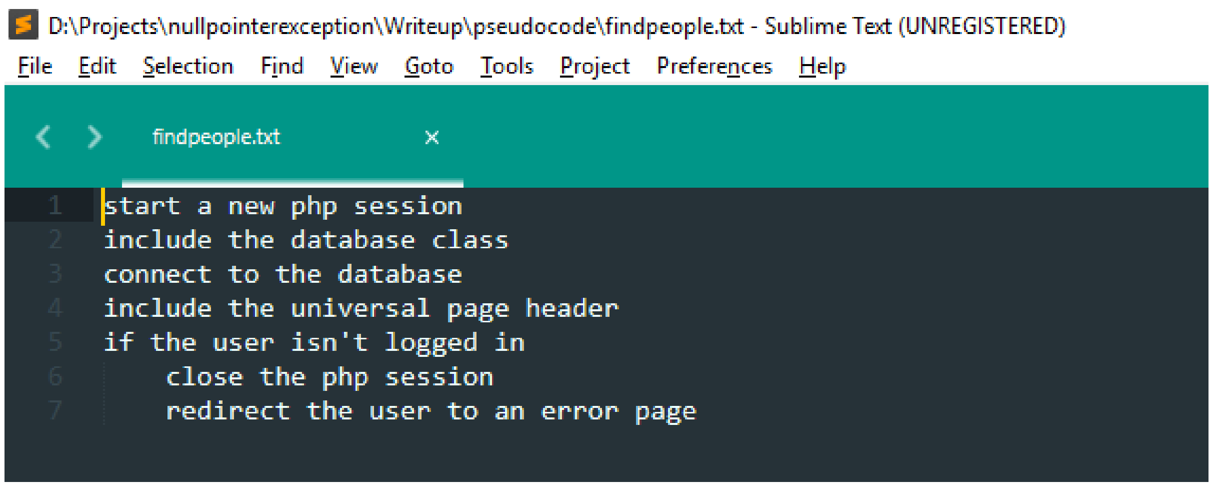
```
start a php session
include the database class
include the core class
include the question functions class
connect to the database
include the universal page header
if the user isn't logged in
    close the php session
    redirect the user to an error page
if the comment in question wasn't made by the user
    close the php session
    redirect the user to an error page
```

error


D:\Projects\nullpointerexception\Writeup\pseudocode\error.txt - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
error.txt

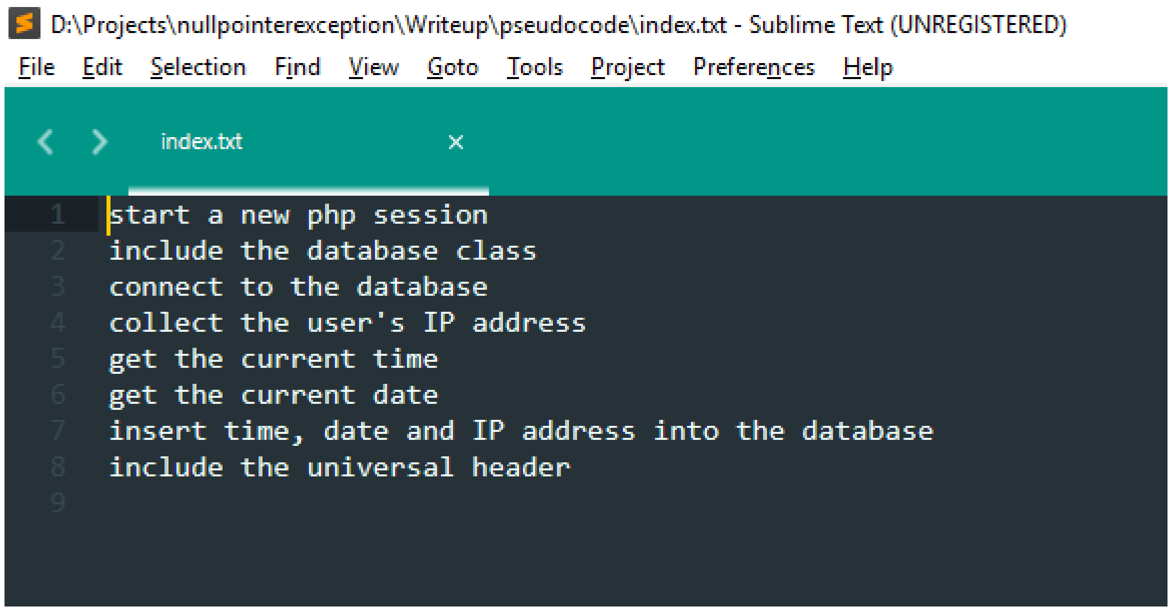
```
if the error is specified in the URL
    set the error member variable to the error in the URL
if not
    create an empty error member variable
switch over the error
    show the correct error message based on the error
```

findpeople


D:\Projects\nullpointerexception\Writeup\pseudocode\findpeople.txt - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
findpeople.txt

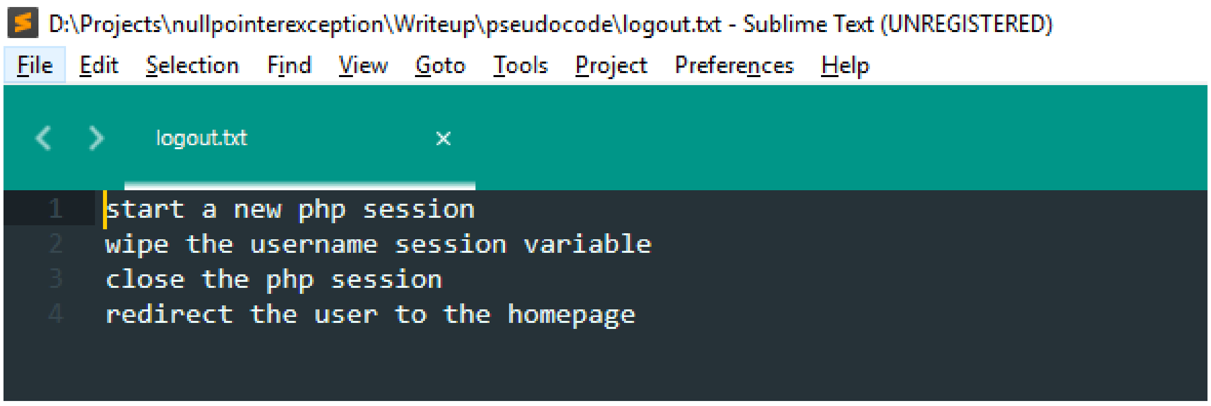
```
start a new php session
include the database class
connect to the database
include the universal page header
if the user isn't logged in
    close the php session
    redirect the user to an error page
```

index


D:\Projects\nullpointerexception\Writeup\pseudocode\index.txt - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
index.txt

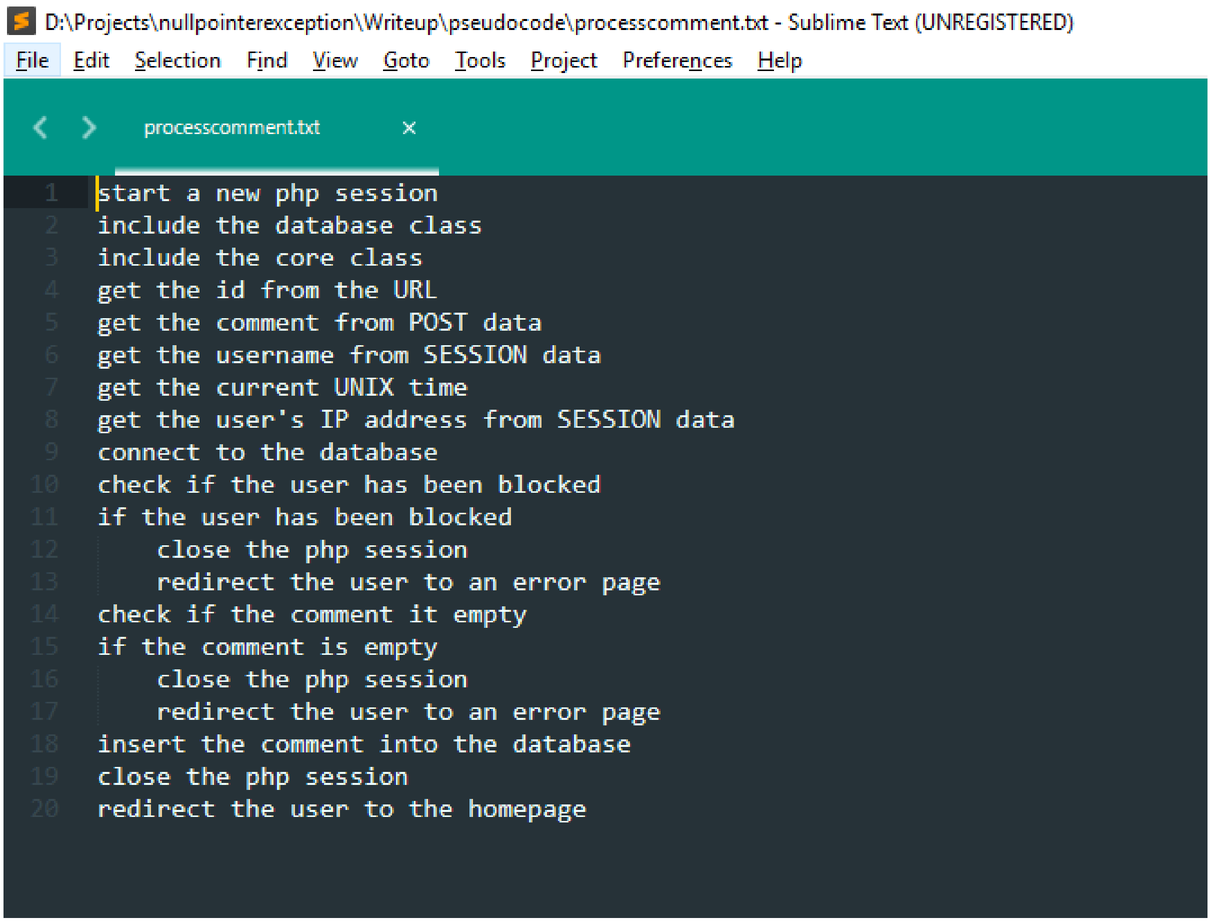
```
start a new php session
include the database class
connect to the database
collect the user's IP address
get the current time
get the current date
insert time, date and IP address into the database
include the universal header
```

logout


D:\Projects\nullpointerexception\Writeup\pseudocode\logout.txt - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
logout.txt

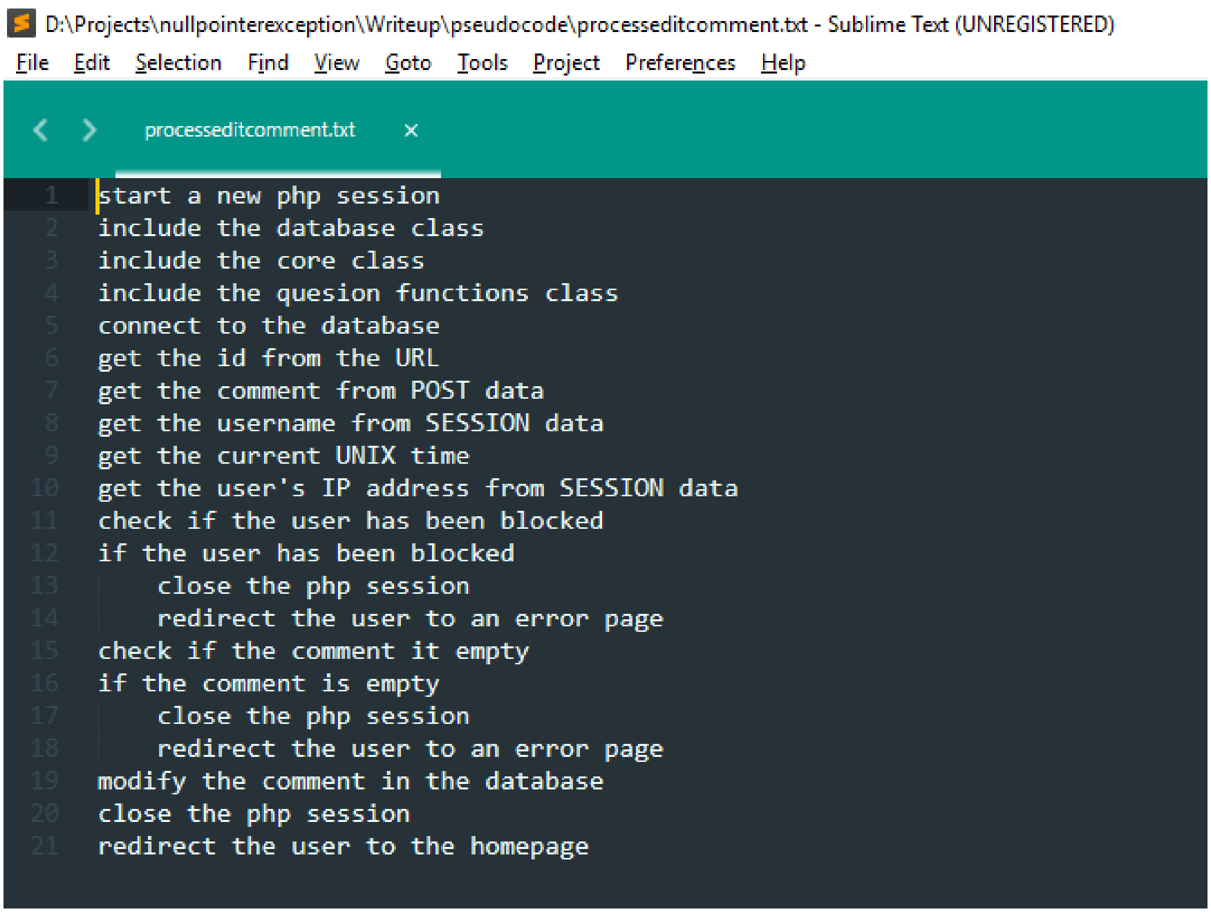
```
start a new php session
wipe the username session variable
close the php session
redirect the user to the homepage
```

processcomment


D:\Projects\nullpointerexception\Writeup\pseudocode\processcomment.txt - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
processcomment.txt

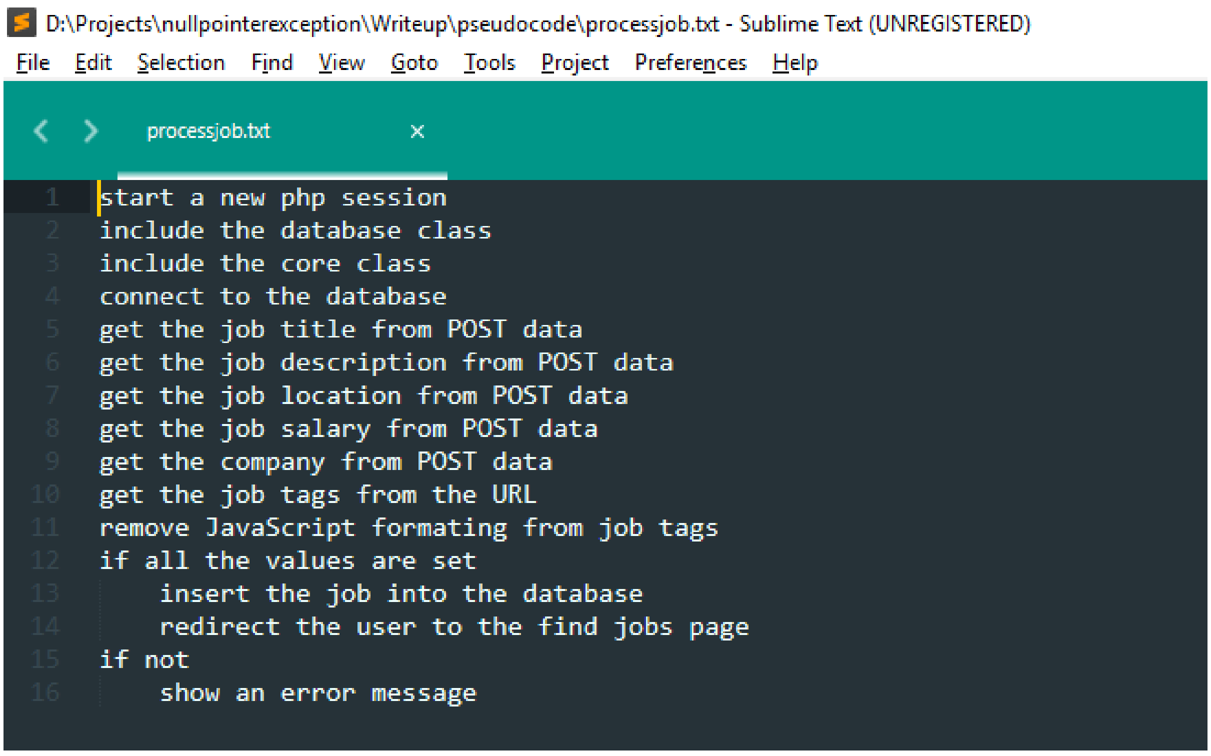
```
start a new php session
include the database class
include the core class
get the id from the URL
get the comment from POST data
get the username from SESSION data
get the current UNIX time
get the user's IP address from SESSION data
connect to the database
check if the user has been blocked
if the user has been blocked
    close the php session
    redirect the user to an error page
check if the comment it empty
if the comment is empty
    close the php session
    redirect the user to an error page
insert the comment into the database
close the php session
redirect the user to the homepage
```

processeditcomment


D:\Projects\nullpointerexception\Writeup\pseudocode\processeditcomment.txt - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
processeditcomment.txt

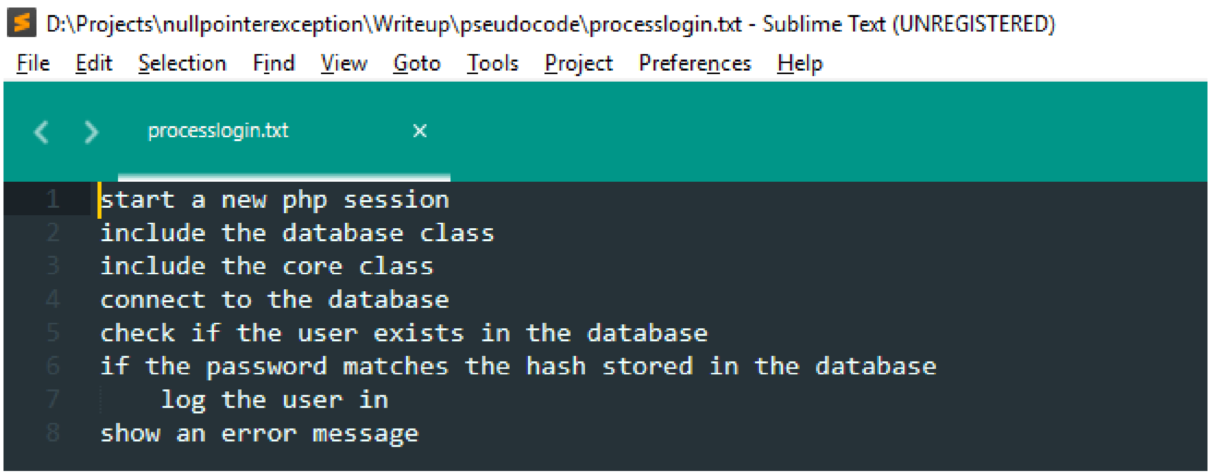
```
start a new php session
include the database class
include the core class
include the quesion functions class
connect to the database
get the id from the URL
get the comment from POST data
get the username from SESSION data
get the current UNIX time
get the user's IP address from SESSION data
check if the user has been blocked
if the user has been blocked
    close the php session
    redirect the user to an error page
check if the comment it empty
if the comment is empty
    close the php session
    redirect the user to an error page
modify the comment in the database
close the php session
redirect the user to the homepage
```

processjob


D:\Projects\nullpointerexception\Writeup\pseudocode\processjob.txt - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
processjob.txt

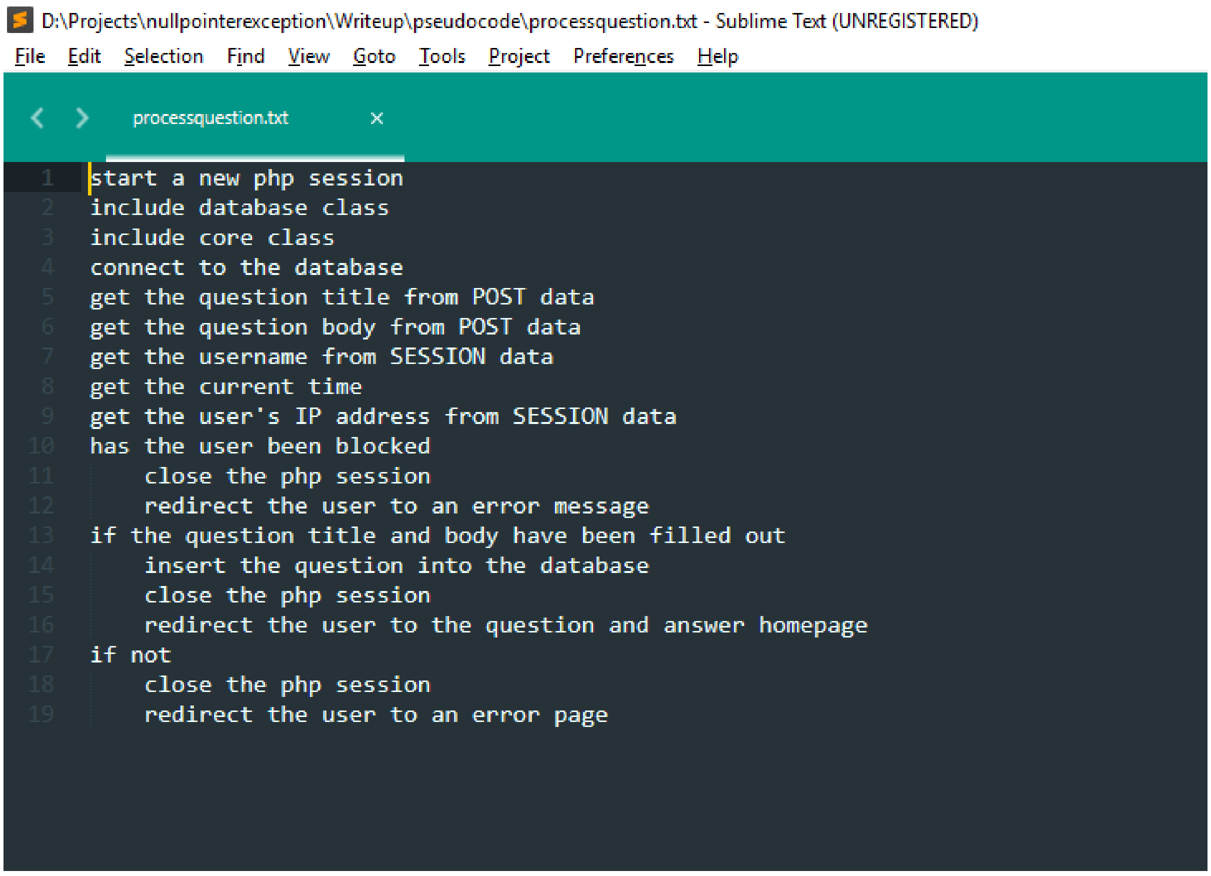
```
start a new php session
include the database class
include the core class
connect to the database
get the job title from POST data
get the job description from POST data
get the job location from POST data
get the job salary from POST data
get the company from POST data
get the job tags from the URL
remove JavaScript formating from job tags
if all the values are set
    insert the job into the database
    redirect the user to the find jobs page
if not
    show an error message
```

processlogin


D:\Projects\nullpointerexception\Writeup\pseudocode\processlogin.txt - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
processlogin.txt

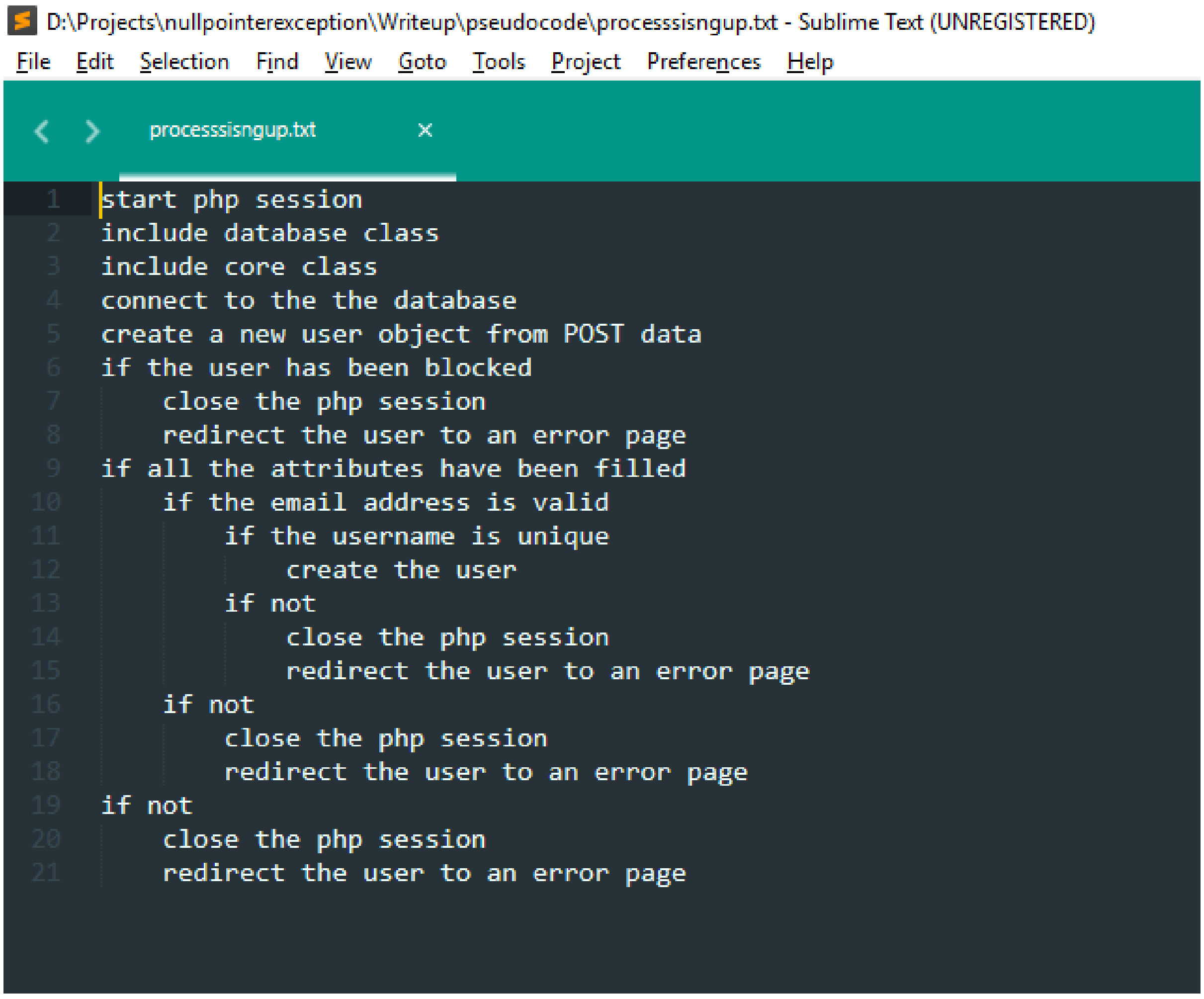
```
start a new php session
include the database class
include the core class
connect to the database
check if the user exists in the database
if the password matches the hash stored in the database
    log the user in
show an error message
```

processquestion


D:\Projects\nullpointerexception\Writeup\pseudocode\processquestion.txt - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
processquestion.txt

```
start a new php session
include database class
include core class
connect to the database
get the question title from POST data
get the question body from POST data
get the username from SESSION data
get the current time
get the user's IP address from SESSION data
has the user been blocked
    close the php session
    redirect the user to an error message
if the question title and body have been filled out
    insert the question into the database
    close the php session
    redirect the user to the question and answer homepage
if not
    close the php session
    redirect the user to an error page
```

processsignup


D:\Projects\nullpointerexception\Writeup\pseudocode\processsisngup.txt - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
processsisngup.txt

```
start php session
include database class
include core class
connect to the the database
create a new user object from POST data
if the user has been blocked
    close the php session
    redirect the user to an error page
if all the attributes have been filled
    if the email address is valid
        if the username is unique
            create the user
        if not
            close the php session
            redirect the user to an error page
    if not
        close the php session
        redirect the user to an error page
if not
    close the php session
    redirect the user to an error page
```

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 is for employers looking to fill vacancies or recruit new talent for their businesses. This means that there is a high chance that the clients using this section of the site will not be programmers or IT professionals, and therefore this section needs to be as simple and easy to learn as a badly designed system will drive customers away.

One feature designed to make the "employers" section easy to learn is an option to search using tags as well as queries. For example, instead of searching for "Rust systems programmer", they could search for tags such as "Rust" and "systems programmer" or "embedded" to make the searching process more visual and provide hints. As with the rest of the solution, one key feature is the real time searching and filtering features, as described in the Q&A section. Combined with the tag searching system, this allows users search very intuitively, and get almost instant feedback as their query changes.

As with the rest of the site, the "employers" section will have a simple and easy to navigate design as well as well labelled buttons. This means the information the user is trying to find or the task they are trying to perform is as painless as possible and doesn't involve searching through multiple pages, as this would increase frustration.

The "Looking for work" section of the project is designed to be used by IT professionals and programmers, so technical vocabulary and job titles shouldn't be a problem. Therefore, the main ways of increasing learnability on this section of the site are the easy to use and navigate user interface and the low number of clicks it takes to perform a simple task. This is in contrast to some existing solutions (as mentioned in the analysis section) where a user has to navigate though a lot of pages and form in order to achieve that they are trying to do. This increases frustration and could put people off using the site.

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.

Efficiency

As with all programming Q&A sites, NullPointerException has the potential for lots of repeat users, using the site to find quick and efficient solutions to common programming issues. Because of the repeat nature of potential users, my website will provide shortcuts for more experienced user's. One way of doing this would be to allow users to specify the languages and technologies that they are interested in and filter out anything else. For example, a C++ developer working on a GUI project would only see results for C, C++ and Win32 API. This would allow them to filter out anything unrelated to their work and allow them to be more productive.

Another system for optimising the experience of a repeat user would be to provide quick links on their profiles. These could point towards specific pages on the site, such a list of the top Kotlin tagged questions, on an Android developer's page. This would allow the developers to "bookmark" specific pages or questions to their profile. This means that if a developer wants to look back on a previous solution found on the site, they can have it easily accessible from their homepage. This could also apply to certain pages of the tutorial zone, for example a guide on string manipulation in C++.

Experienced users will be able to customise their experience via a settings page on their profile. This will allow them to do anything from changing the colour of their profile, to setting their favourite languages and uploading their C.V. Some of these settings are likely to lead to the user being more efficient, such as turning off popups and tooltips, intended to help beginner users. This should slightly increase efficiency as there is less to distract the user.

The final factor that will increase the efficiency a user is just the amount of time they spend on the site. The simple design means that as users become more experienced with the layout, they will be able to do certain tasks quicker. The design of the website means that the information is always in a predictable place and will not move around, so the user knows where to find information without wasting too much time looking around.

The efficiency of the site is harder to test than the learnability of the site as it would require a user to test the site over a number of days, which is unlikely to happen outside of a production environment. The next best thing would be to show people the efficiency tools after they have just tested the learnability of the site, and then ask them to complete the same tasks as before after they are familiar with the tools. This test is not conclusive because these tools are meant for long term productivity but can help to give me a basic idea of their effectiveness.

Memorability

If a user decided to take a break from the site, for example if they take a year off work or have a long holiday, they will have to be able to resume use of the site without having to relearn too much, in order to maintain productivity. One way I can ensure that users don't have to spend much time relearning the site after a period of absence is to ensure that the location and process of accessing of a piece of information don't change. For example, the process of searching for a question shouldn't change significantly when the site reaches production. This ensures that users don't have to relearn how to use a feature of the site.

The main way of ensuring that a user can come back to the site after a break and still be familiar with the site, is to avoid doing a major redesign of the site after release. Providing the finished site is modern and met my success criteria and had good user feedback, the site shouldn't ever need a redesign. With a much larger user base in on the production build, I would not be averse to small changes based on overwhelming user feedback, but the main look and feel of the site cannot change throughout its existence to ensure that users can continue to use the site after a period of absence.

As with the efficiency of the site, memorability is a hard property to test, because it involves someone learning the site, then attempting to use it after several weeks. This is unlikely to happen outside of a production environment.

Errors

NullPointerException is a website, which means that it has a user interface. This means you don't have to memorise commands and arguments like you would with a command line program. However, having a user interface doesn't prevent users from entering data in the wrong place, or in the wrong format, and so steps must be taken to prevent this from happening.

In order to prevent users from entering data in the wrong format, HTML input elements have a type attribute which specifies the type of data which is allowed. The form will not send its data unless the information is in the right format. There is also a required attribute that ensures that the form is not empty. However, in the interests of security and robustness, this is not enough. Users can modify the HTML content of the website to remove the type attribute or can create their own network request to the site containing whatever data they want. This is fixed by checking the data in the PHP script the form is processed by. The process script checks the data is not empty and is in the required format. If these conditions are not met the script attempts the handle the error gracefully instead of crashing. This ensures the security of the website and improves the user experience, as a PHP stack trace on a programming help website would be very embarrassing.

However, this is still not enough input validation. Some of the user input is uploaded to the database, which leaves it vulnerable to SQL injection. SQL injection is where a user can manipulate user input to run SQL code on the database. This be listing the passwords or simply deleting the database. I cannot prevent all forms of SQL injection due to time restrictions, but I can at least try and sanitize any user input to reduce the risk.

When it comes to preventing the user clicking the wrong link or entering the wrong data, all I can really do is build an intuitive site with plenty of resources in place to help users learn the site. In terms of entering the wrong data, as long as the inputs are all clearly labelled there isn't much I can do. The website will assume all data is correct and process it accordingly. My main focus instead is to ensure that any data entered incorrectly has no security implications and any errors are handled gracefully.

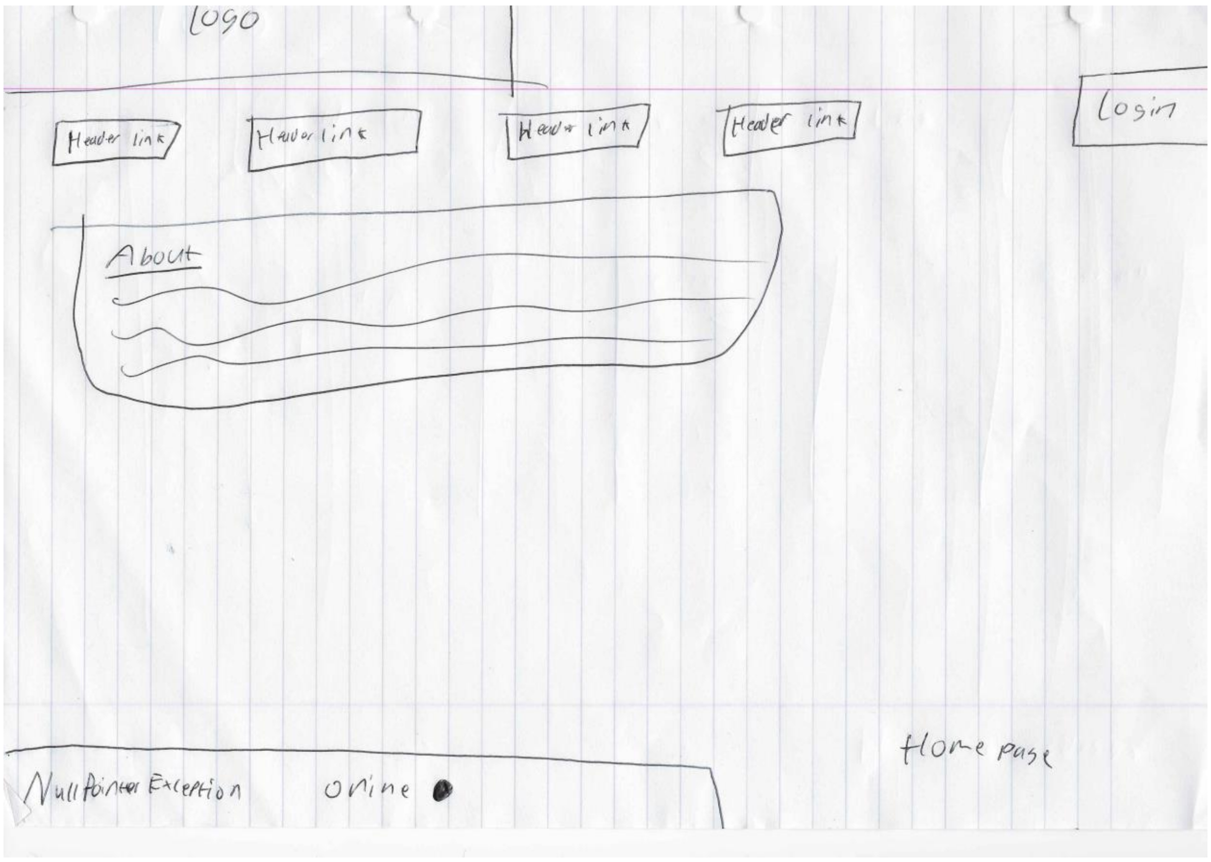
Satisfaction

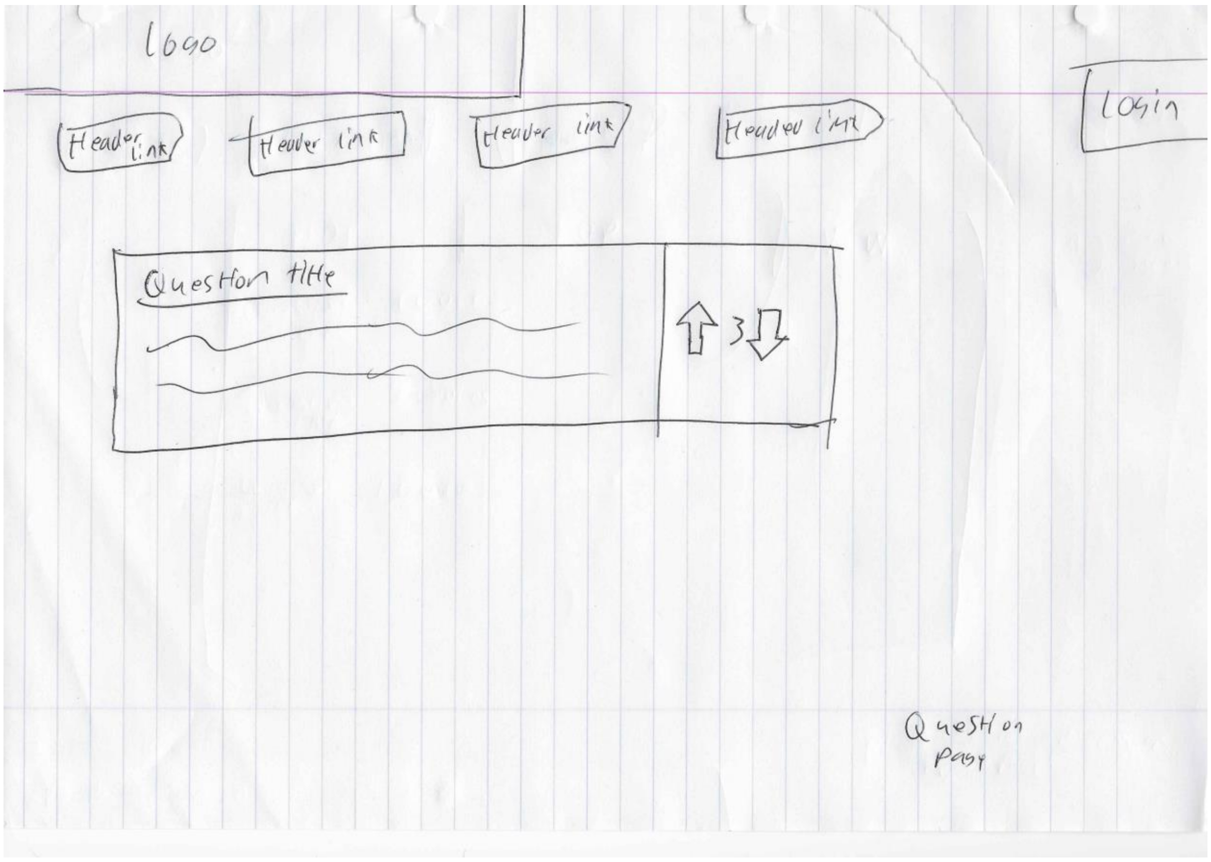
For a website, the design and user experience as very important. However, design is subjective and what I like might not be what anyone else likes. Because of this, I'm continually asking for feedback from friends while designing the site, in order to find a look that works for everyone. This feedback has led to a complete redesign and a new UI framework. After the website is finished, I will create some form of survey to ask my testers for feedback about the design and UI. I will use this to quantify how successful my design is. Fundamentally though, I believe that a functioning website is better than a pretty one, although I will strive for both.

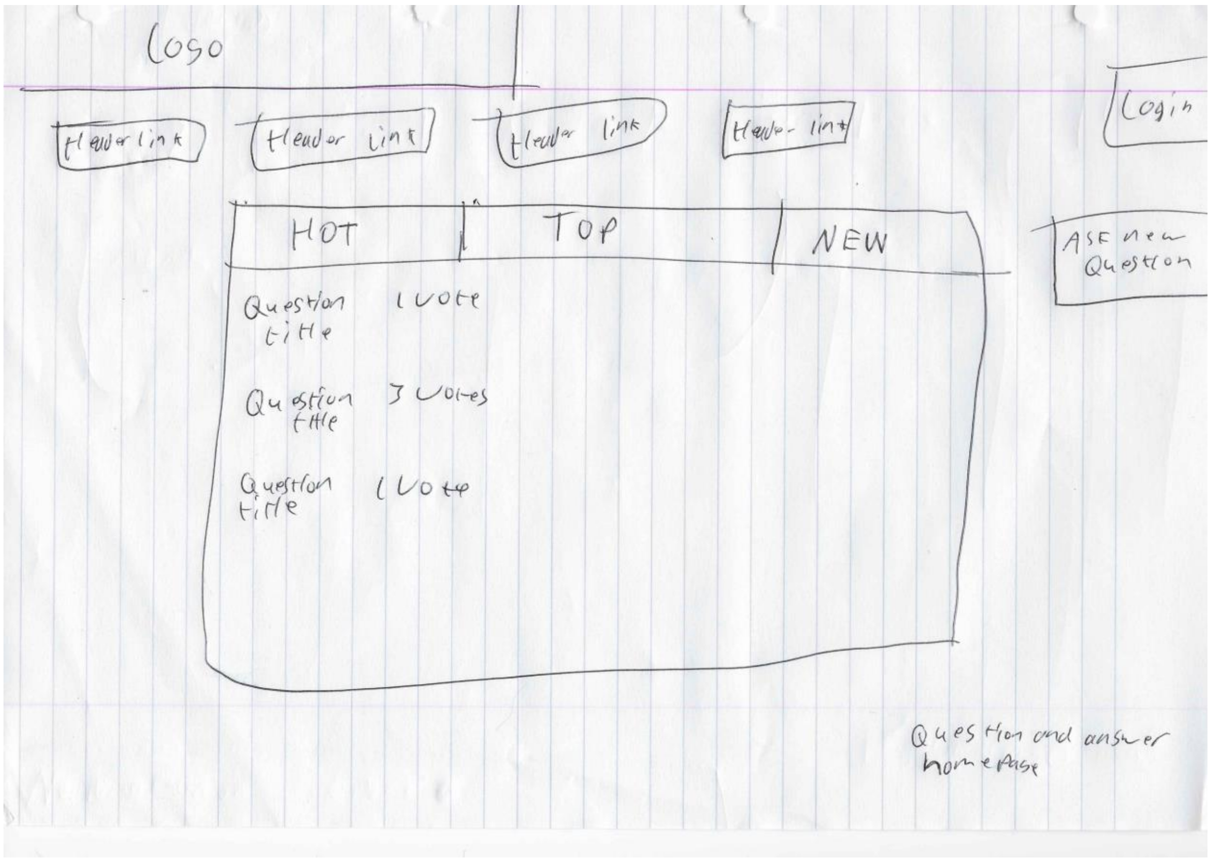
Design drawings

I planned the layout and user interface of my website using pen and paper to make the programming easier. I made two sets of design drawings: the initial design, and then a redesign during the development phase. I've included the redesign here as it's also design. The quality of the design drawings is inconsistent because they were scanned onto my computer by an unreliable printer.

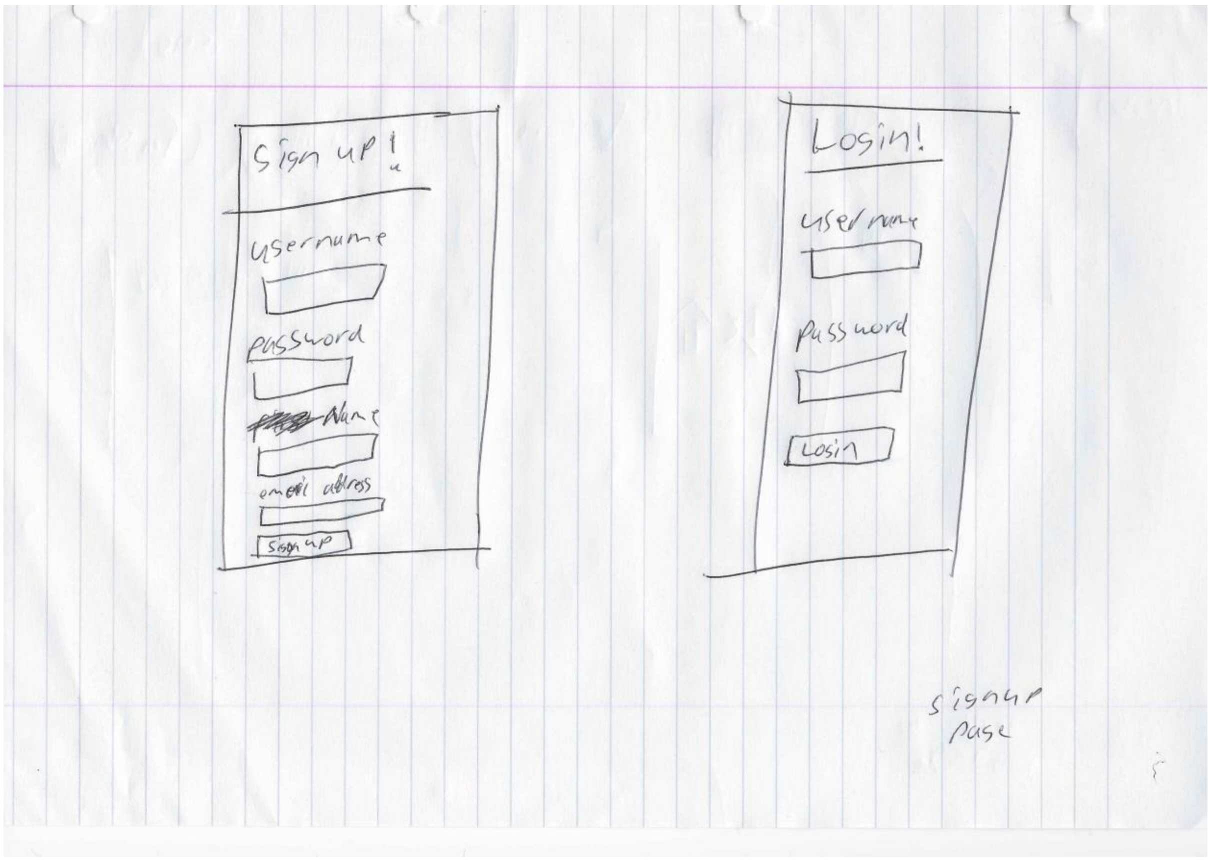
Initial design drawings:



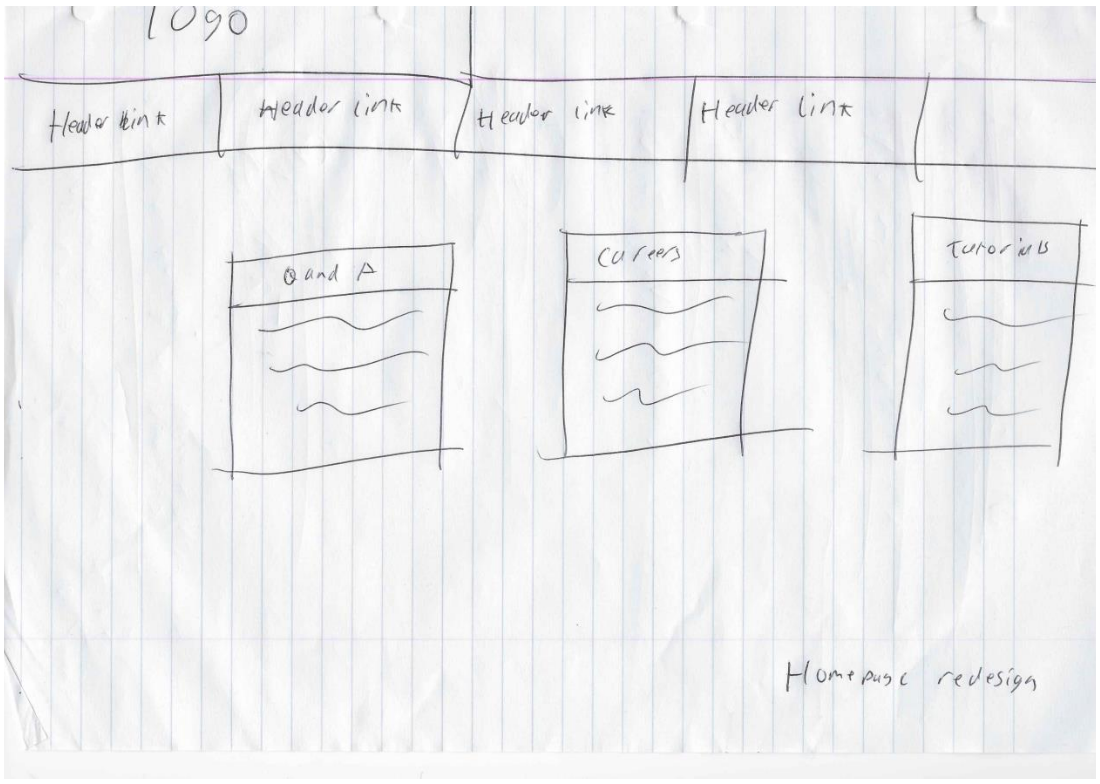

logo
Login
Header link
Header link
Header link
Header link
Question title
3
Question page


logo
Login
Header link
Header link
Header link
Header link
HOT
TOP
NEW
Ask new Question
Question title 1 vote
Question title 3 Votes
Question title 1 Vote
Question and answer homepage

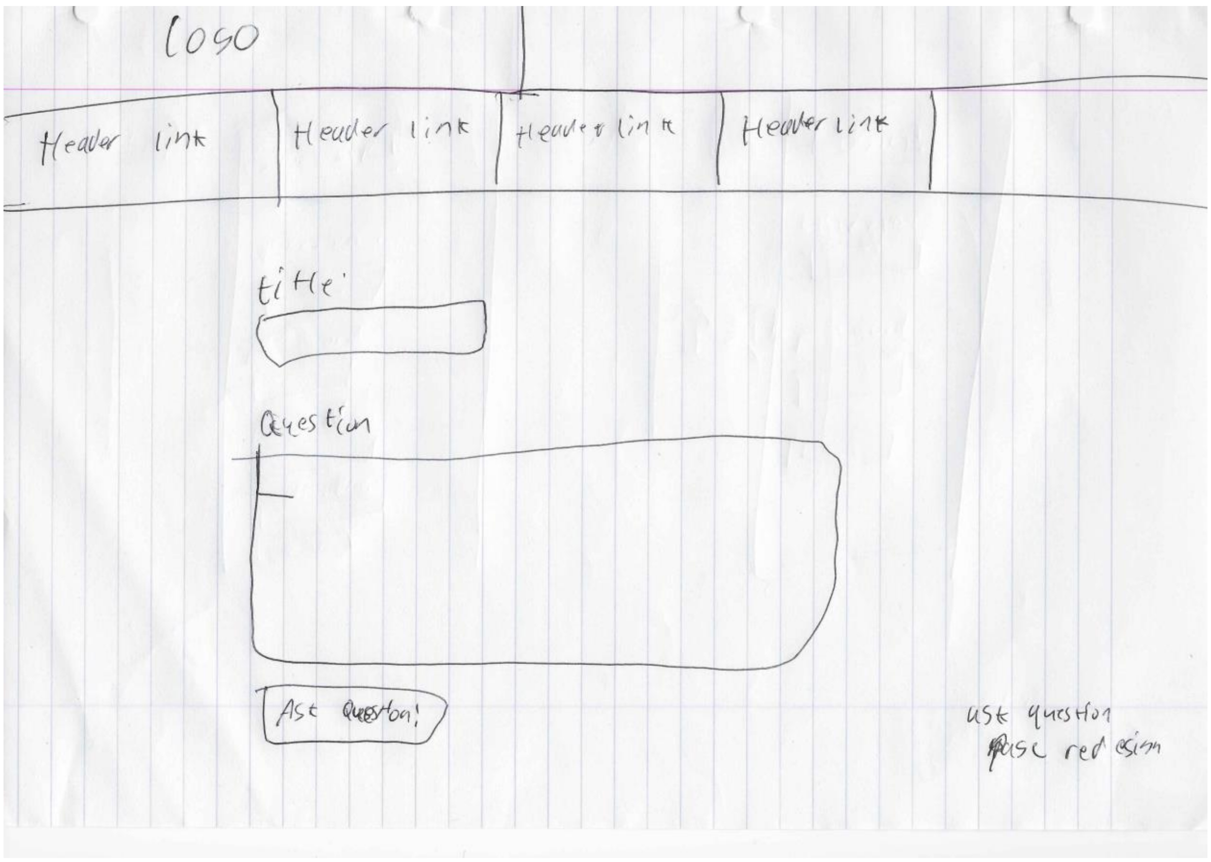
NullPointerException - Your questions. Answered.

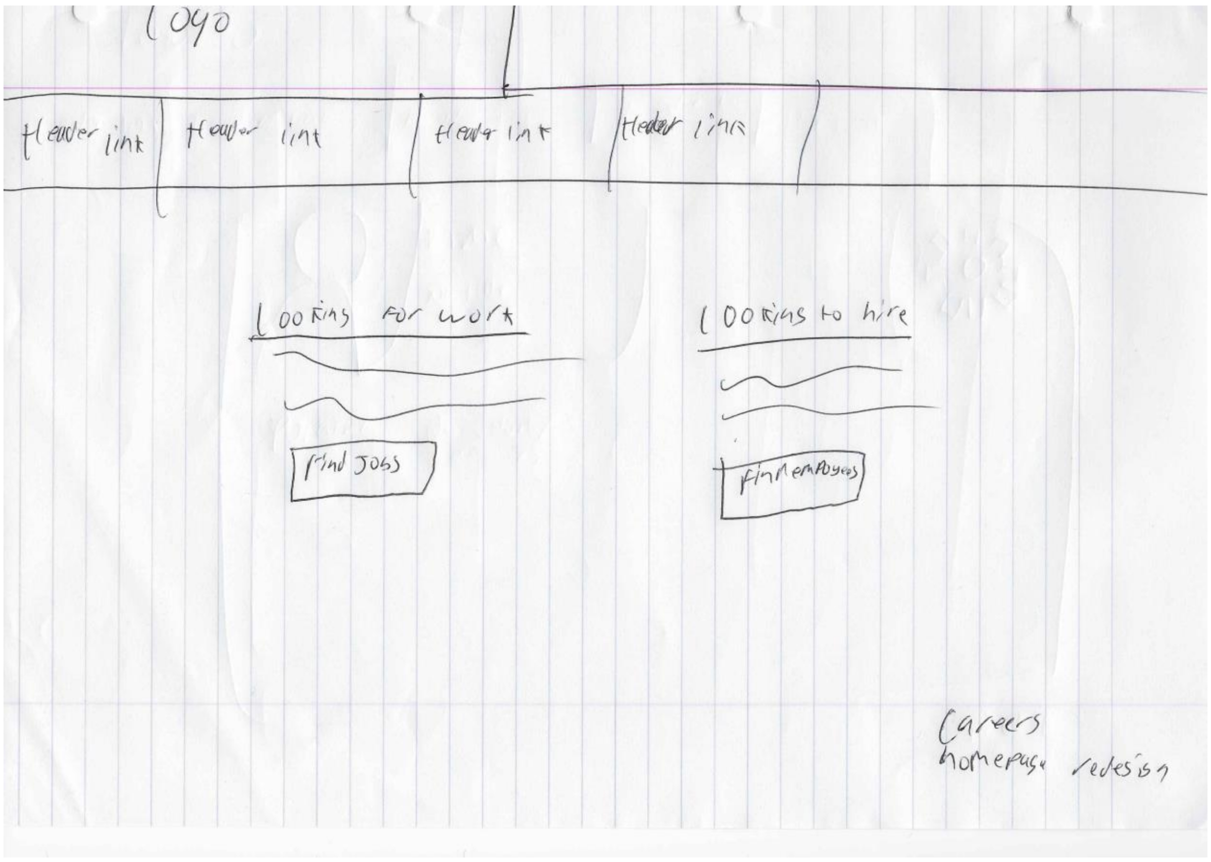


Redesign drawings:

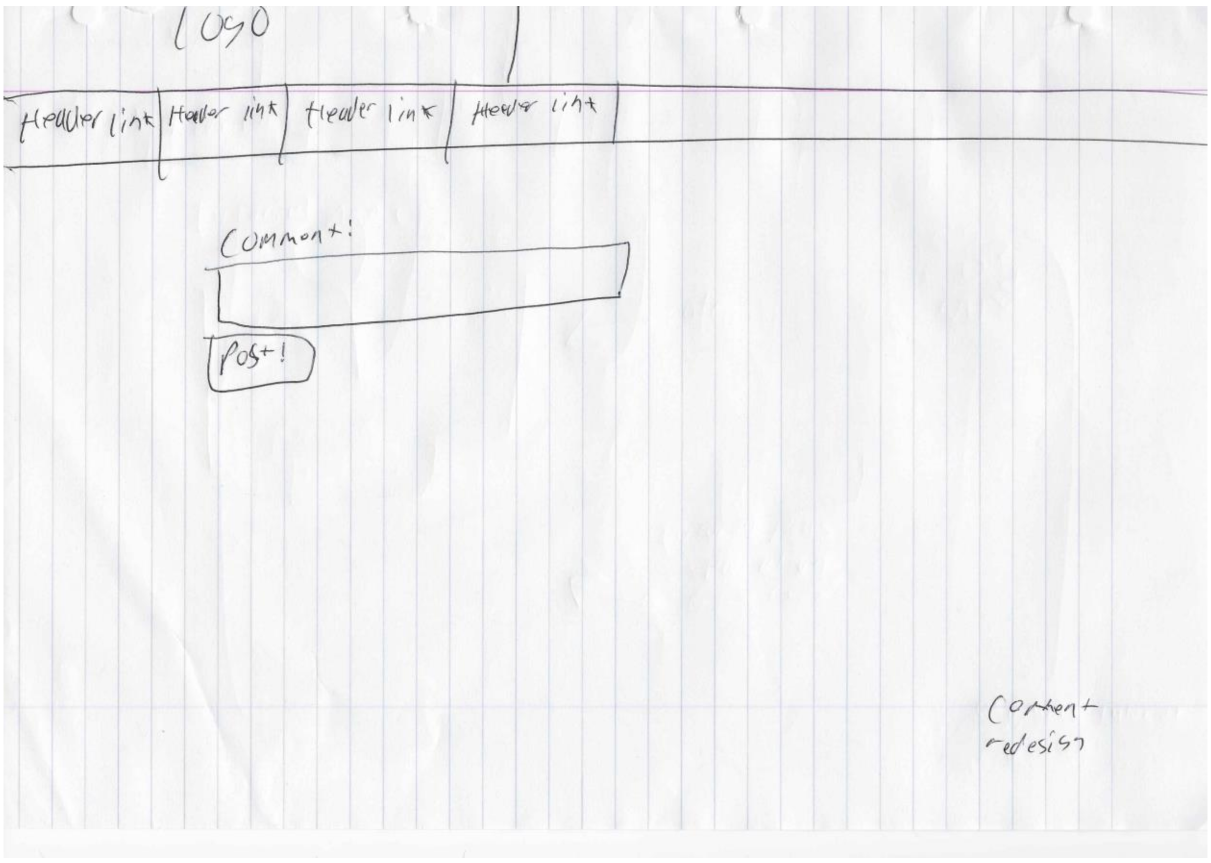


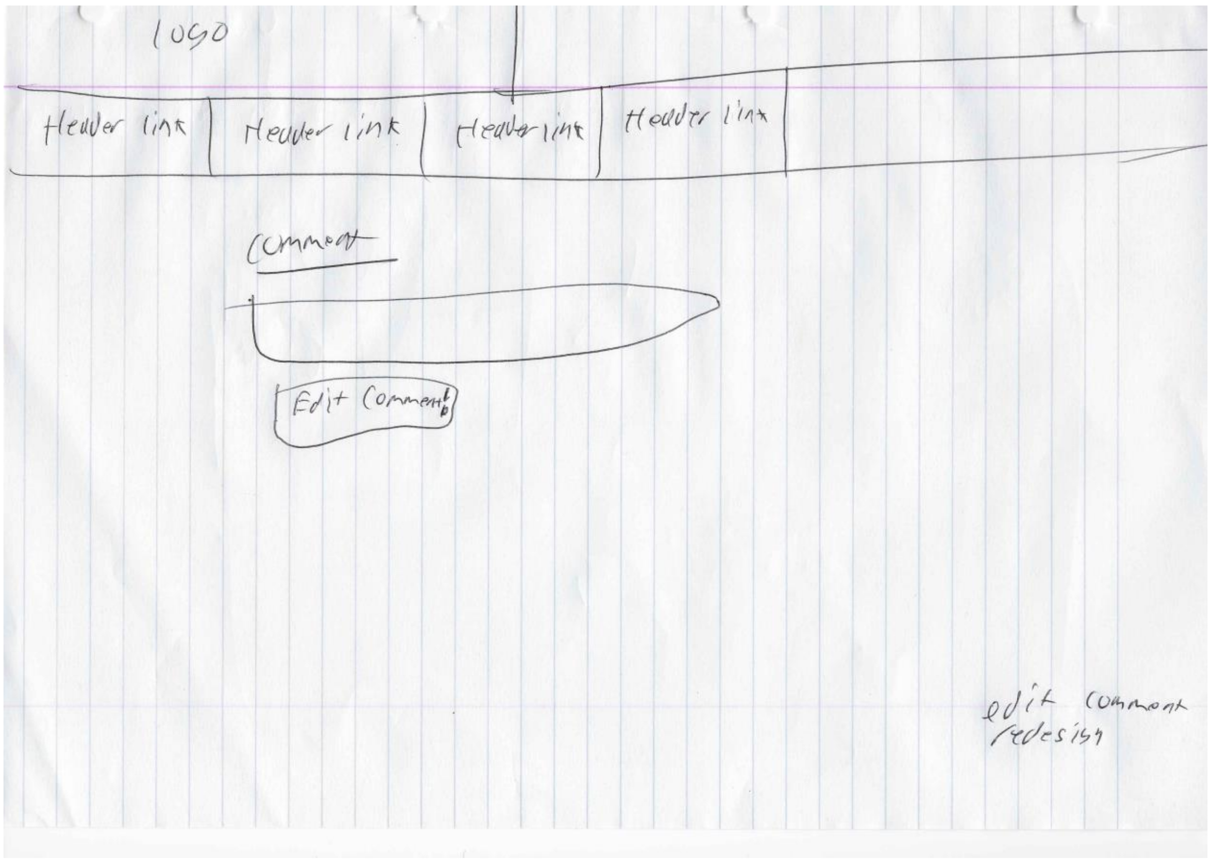
NullPointerException - Your questions. Answered.

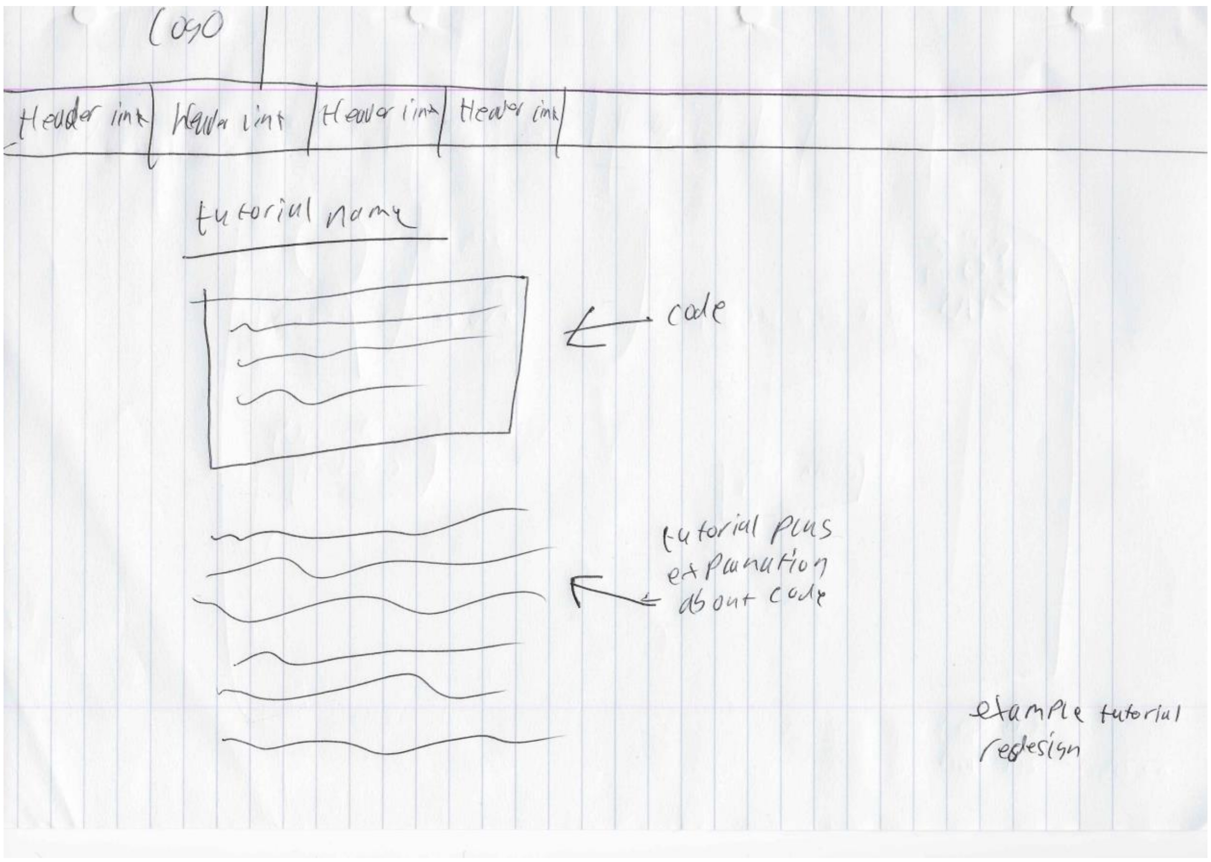


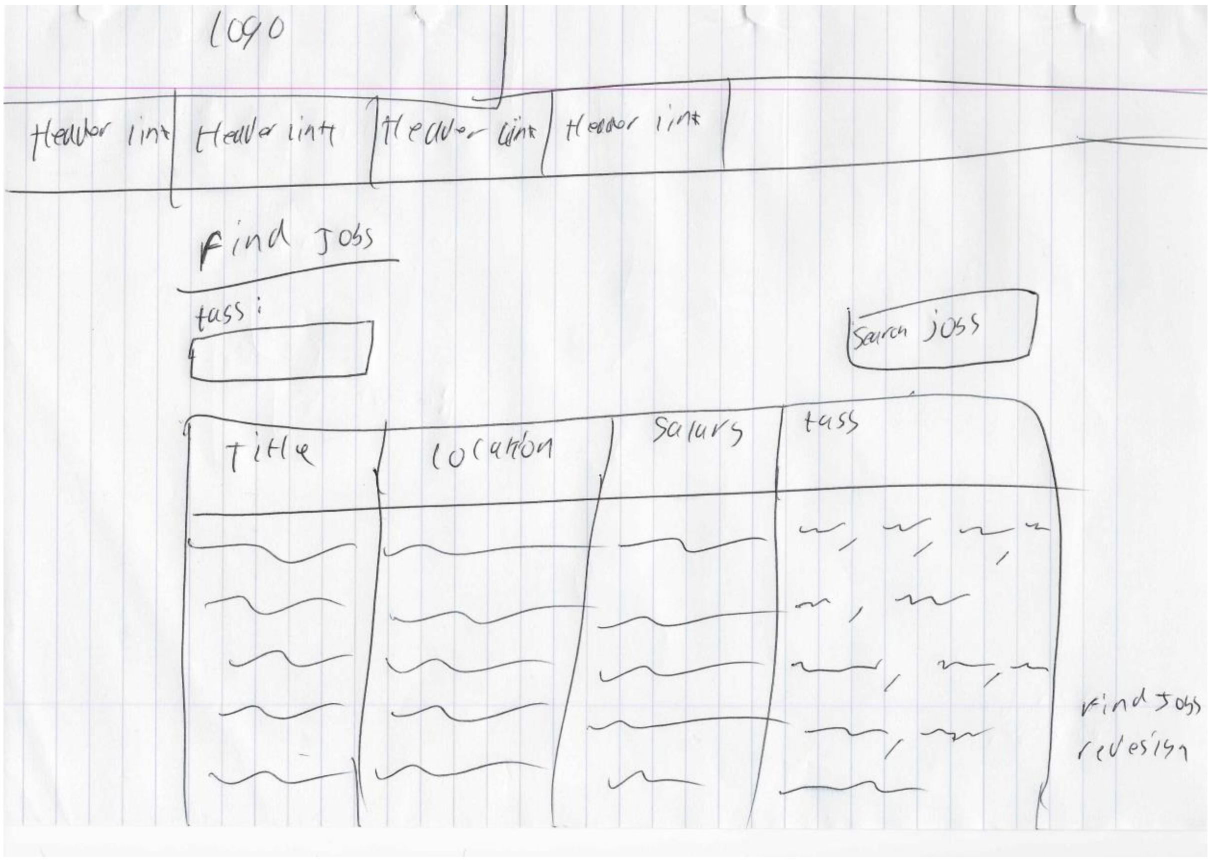


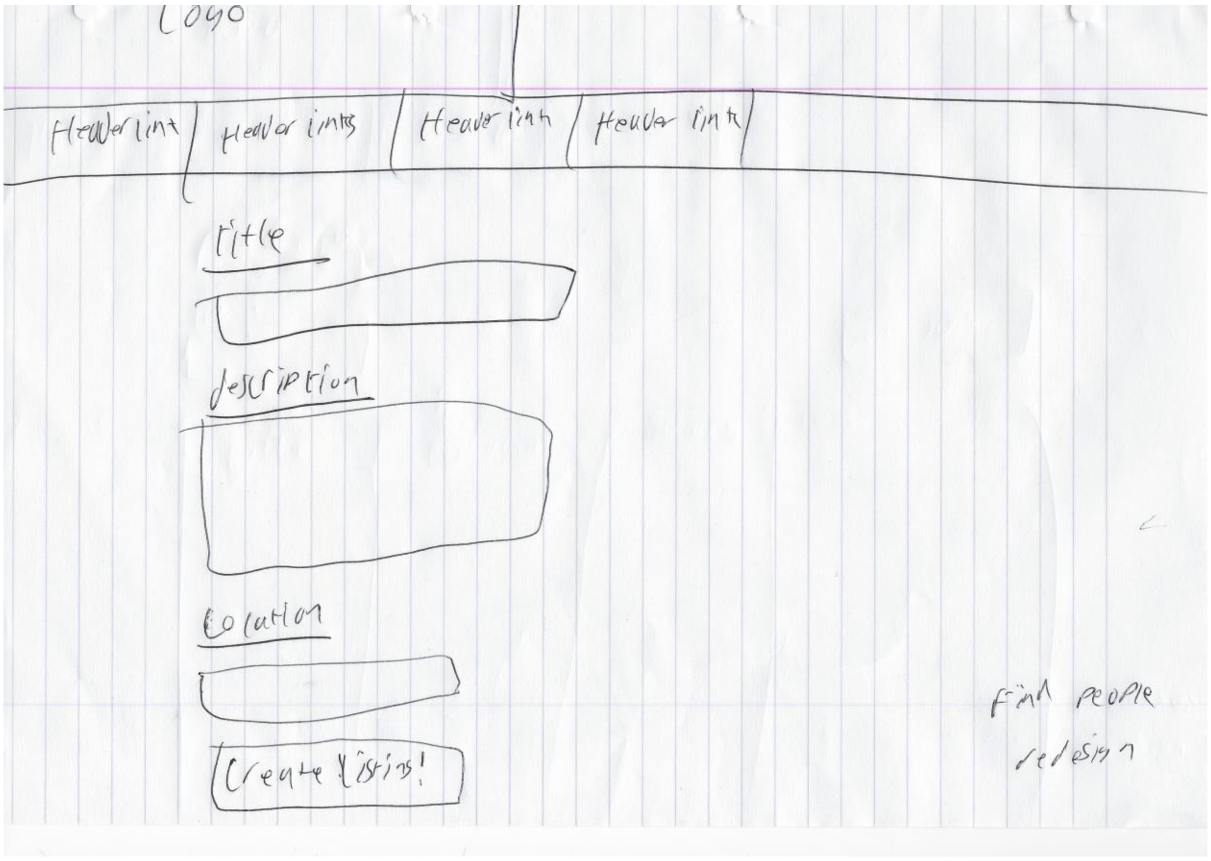
NullPointerException - Your questions. Answered.

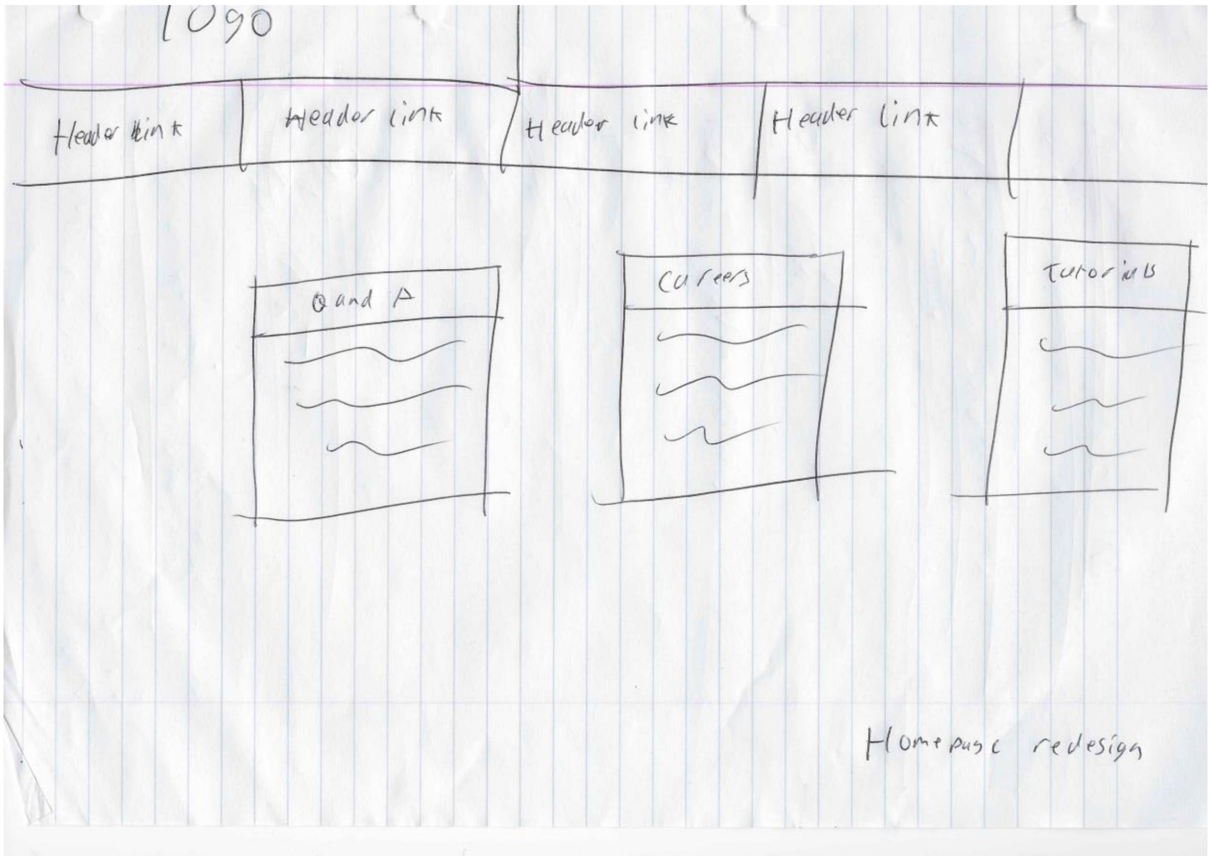




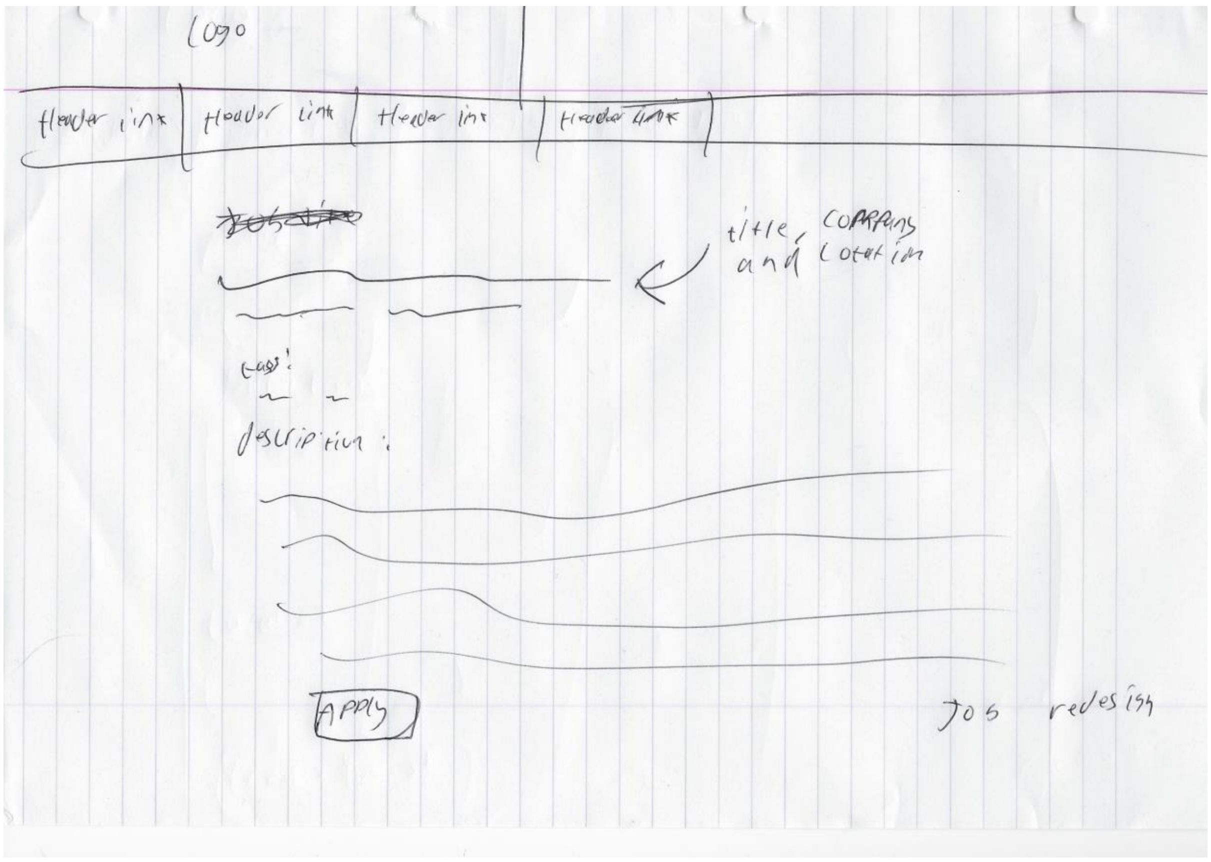

logo
Header link
Header link
Header link
Header link
tutorial name
code
tutorial plus explanation about code
example tutorial redesign

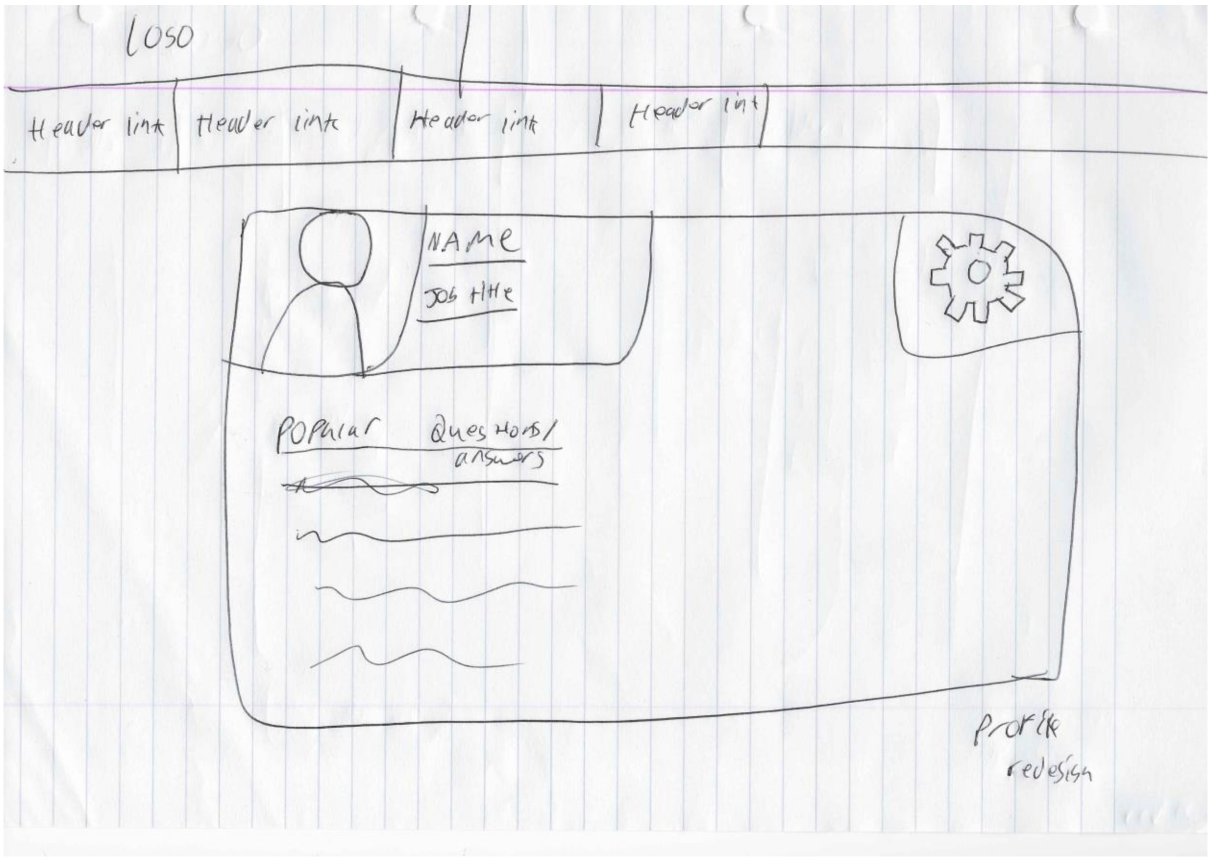

logo
Header link
Header link
Header link
Header link
Find Jobs
tags:
Search jobs
Title
location
Salary
tags
Find Jobs redesign


Logo
Header link
Header links
Header link
Header link
title
description
Location
Create listing!
Find people
redesign

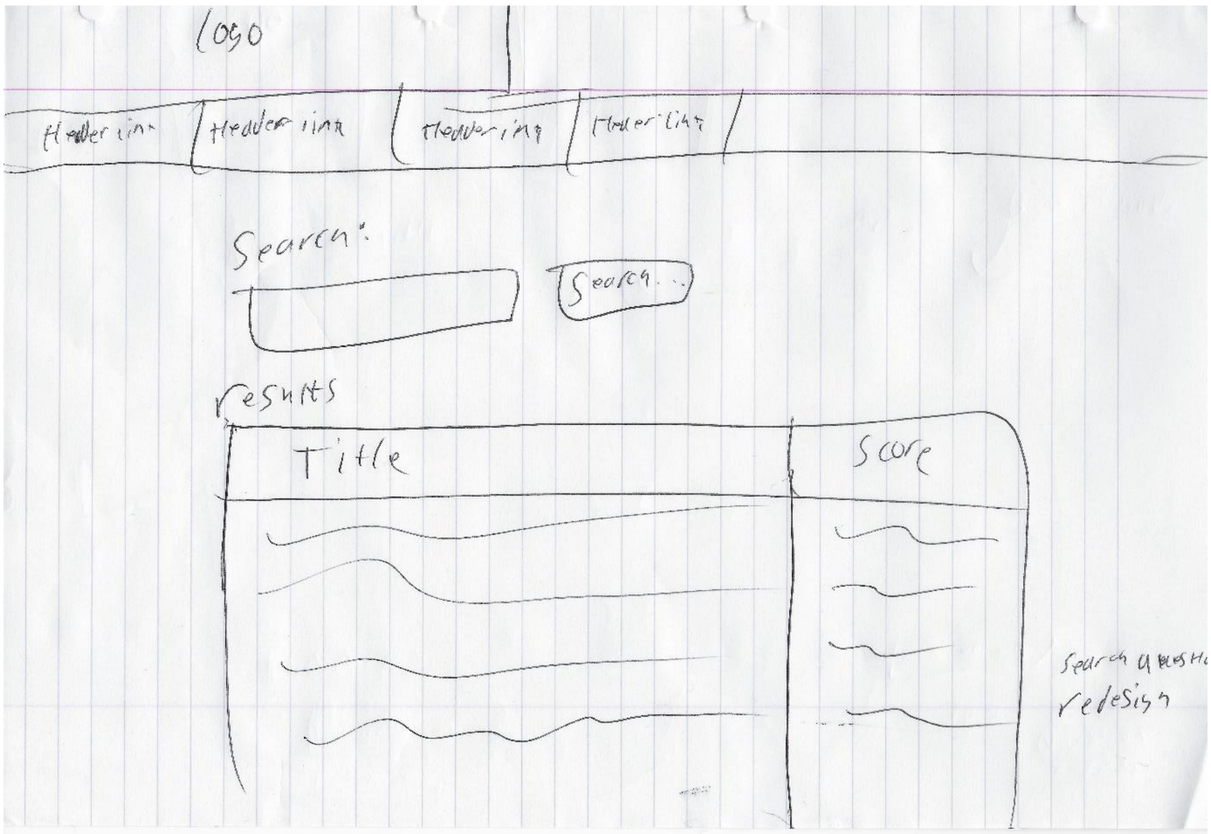

Logo
Header link
Header link
Header link
Header link
Q and A
Careers
Tutorials
Homepage redesign

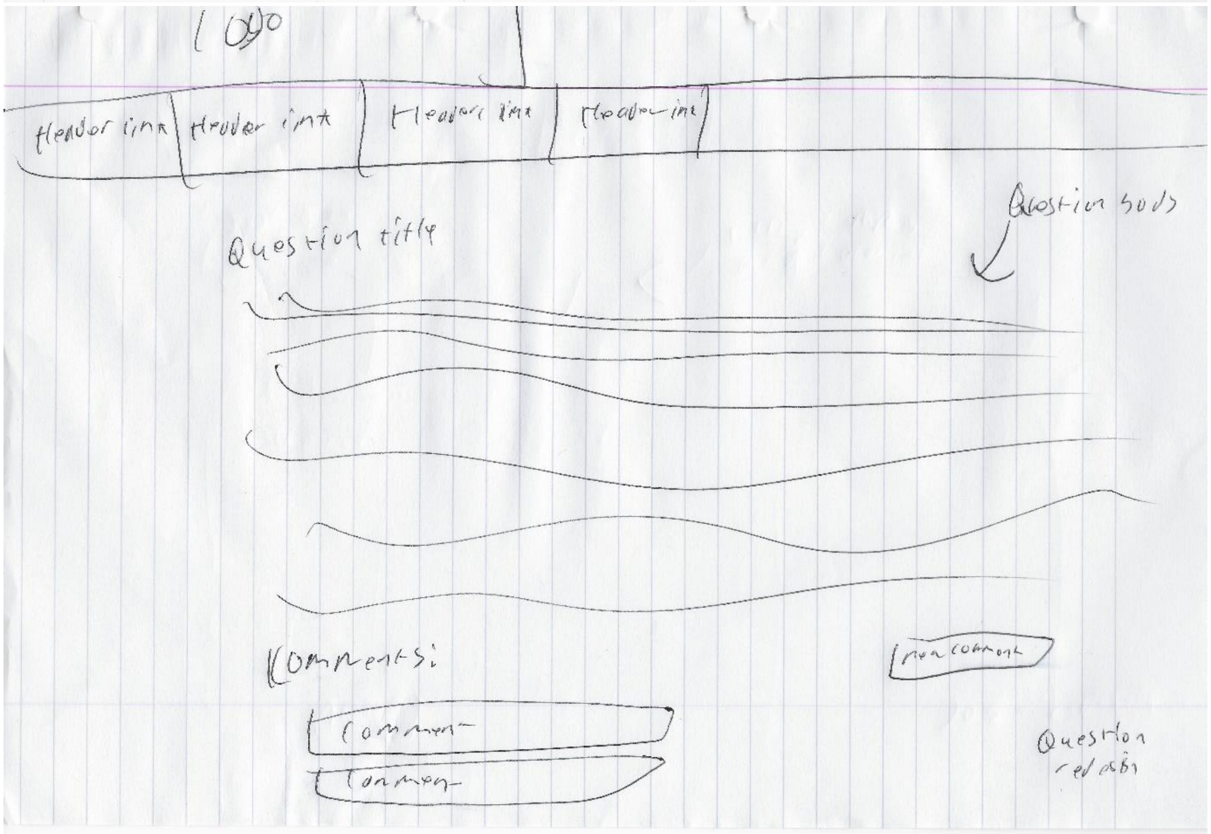
NullPointerException - Your questions. Answered.



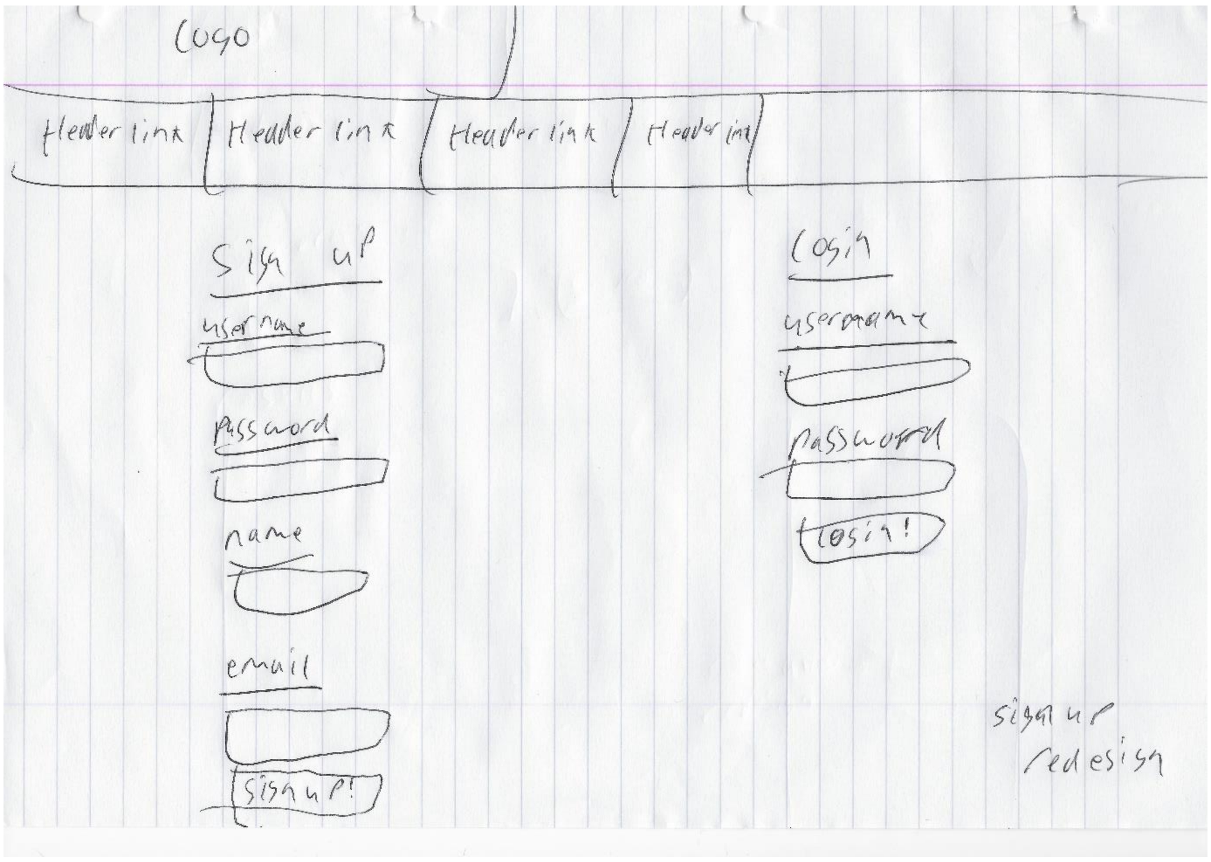


NullPointerException - Your questions. Answered.





NullPointerException - Your questions. Answered.



Part (D) – Validation

Error prevention

Methods of preventing a user from inputting invalid data:

- HTML inputs with fixed type
- HTML inputs with required attribute
- Analysing input in PHP script

HTML input elements have an attribute called type, which specifies the type and format of the data that the user can enter. For example, setting the type attribute to “date” will only allow the user to input a date, and only in a specified format. On some browsers it will also bring up a calendar for the user to select the date from. In most cases this method is enough to prevent users from entering data in the wrong format.

HTML input elements also have a required attribute, which states that the input must be filled in before a form can be sent. This prevents users from accidental forgetting to enter any data for an input. This, coupled with the type attribute, makes it very difficult to accidentally break the program.

While the two previous methods are both effective, client-side input validation is never enough on its own. Client-side validation can be abused or ignored by malicious users in order to disrupt, compromise or break a service. For example, required input can be ignored by simply sending a network request to the site without including certain fields. To increase security, I implemented server-side validation to the website, as part of the PHP scripts used to process and store input. This validation includes checking that all required fields contain a value and using regular expression to validate email addresses.

Error correction

If a user enters invalid data, two things must happen. The error must be handled gracefully, and the input must be ignored. The data must be ignored in the event of an error to ensure that incorrect or invalid data doesn't get sent to the database. Handling the error gracefully is also very important because no one wants to use a website with error messages and stack traces all over it. Stack traces can also compromise the security of a website as they can contain passwords and other confidential data.

Graceful error handling is when instead of crashing, a program attempts to intercept and error before crashing and either move on invisibly or display a formatted message to the user informing them of the issue. This has two benefits, it keeps the program or service running, and it controls what the user sees, leading to a better user experience.

Part (E) - Testing during iterative development

Input test data:

For the purpose of testing, I'm splitting the input data into two categories: expected and unexpected. Expected data will be any inputs that the website would normally receive in from a user. This includes entering a valid email address when signing up with a new account, and logging in as an existing user when signing in. This type of data allows me to test the functionality of my website and that the correct output is shown in all instances. Unexpected data will be any data not normally entered by a user, such as an email address in an incorrect format, or a username comprised of unexpected characters. This will allow me to test the robustness of the program and see how well it responds to unexpected inputs without crashing.

I will also be performing tests without input data, such as a load test to stimulate a large number of simultaneous active users.

Types of test I'm going to use:

- White box testing (All pages and scripts)
- Black box testing (All pages but one)
- Destructive testing (All pages but one)
- Unit testing (All PHP functions and scripts)

White box testing:

White box testing is when someone experienced with the application systematically goes through page by page and performs tests against expected output, with reference to the code and internal structure. This will be done by me, during and after development, as I have the most experience with both the code and the actual website, having spent so much time developing it. The benefits of white box testing are that as the developer understands and has access to the code, they can provide inputs specifically designed to cause a problem based on the code and see the result. This is different to black box testing where the tester has no knowledge of the systems underneath the UI and is simply testing inputs against an expected output. The drawbacks to white box testing are that it takes a lot of time, and the tests can be complex meaning that I can't enlist anyone else to help me with the white box testing.

I will be white box testing all the webpages and scripts that make up the website, regardless of their complexity.

Black box testing:

Black box testing is when a tester with no knowledge of or access to the internal systems and code tests the site to find missing features or unexpected output. This will be done by a class of younger students, as the more people carrying out the test, the thorough the test will be. One advantage of black box testing is that it is done form the point of view of a user, so can find issues that a user would otherwise have found. They are also unbiased, as the testers are unaffiliated with the developer/s. However, black box testing is less thorough that white box testing, meaning that white box testing should be done first.

The class of younger students will be black box testing all of the webpages apart from the admin page I built for submitting IP Addresses for blacklisting.

Destructive testing

Destructive testing is when you test a system with the intent of breaking it, such as supply extreme load (for a server/website) or using an obscure resolution to check that the design scales accordingly. This test will be done by a class of younger students, as it requires no knowledge or experience, and would benefit from a large number of testers. The benefit of destructive testing is that it often finds bugs that you wouldn't otherwise find, due to the large number of people testing, however not all of the site will be tested thoroughly.

As with the black box testing, the class of younger students will be testing all of the webpages apart from the admin page I built for submitting IP Addresses for blacklisting.

Unit testing

Unit testing is when a developer writes tests to supply input into a program and test the output against a supplied value. Providing the main structure of the program doesn't change too much, the main benefit of unit testing is that a test case can be written early on in the program's development, and don't have to be modified, so you can ensure a feature that worked a year ago still works today. Unit tests can also be ran in a batch, so you can see all your test cases either passing or failing automatically. For example, you could run all your tests automatically when you merge dev->prod to ensure that all your tests are passing before any changes to the application are deployed to production. I will be creating the unit tests for NullPointerException.

I will be writing unit tests for all the PHP functions and scripts on the site. There is no reason to write unit tests for static HTML pages or UI as there is no return value or output to check.