GROUP ASSIGNMENT

TECHNOLOGY PARK MALAYSIA

AAPP009-4-2-WDT

Web Development

UCDF2005-ICT(SE)

Group member(s):

(pseudonym) j5ylim

Lecturer:

En. Firdaus Che Abdul Rani

Table of Contents

1.0: Gantt Chart

(Chart has been split in half to maintain legibility, due to its width.)

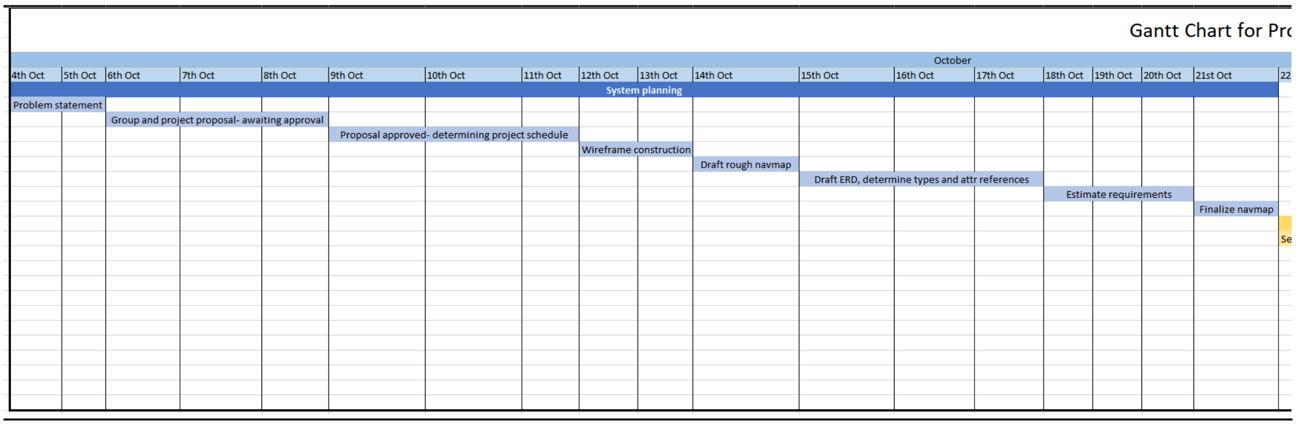


Figure 1: The "System Planning" half of the project plan.

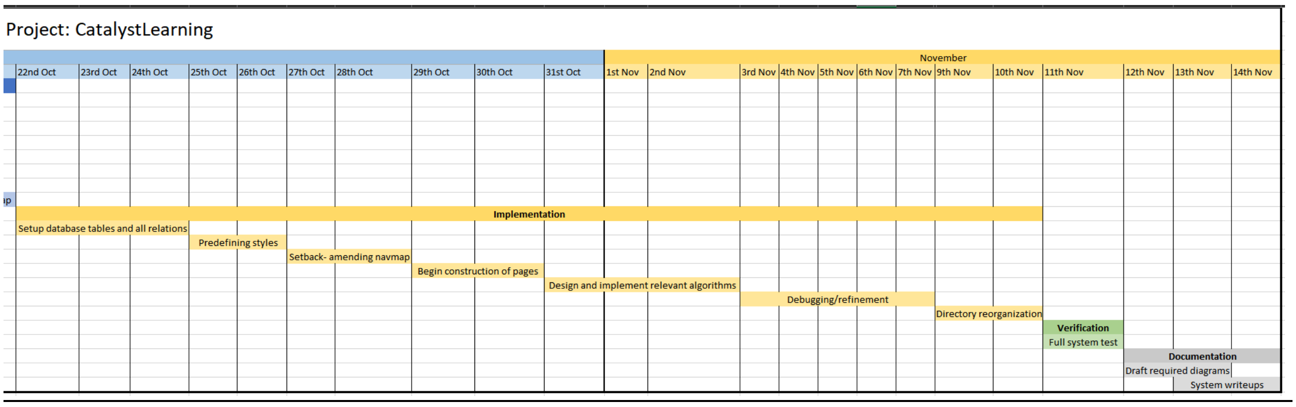


Figure 2: The "Implementation", "Verification" and "Documentation" parts of the project plan.

2.0: Introduction

The mandatory lockdowns enforced by governments worldwide in combatting the recent pandemic has negatively impacted traditional means of education, whose alternative being virtual learning platforms which had, in my opinion, been ill-equipped both in terms of reliability, design and infrastructure to deal with the sudden demand, expectations and sole reliance placed on said method of education.

Figure 3: An example of an interactive learning system developed locally and sanctioned by the government, VLE Frog

It is due to witnessing first-hand the shortcomings of said systems that led to my design and construction of an interactive learning system, despite my unfamiliarity with the topic. My approach to this topic, albeit an admittedly myopic, rushed, and likely incomplete iteration, addresses as many perceived problems as possible while offering significant improvements in performance, speed, design choice and accessibility over present options. This web application is thus named CatalystLearning.

An “interactive learning system”, which thus is my topic of choice for this assignment, is vaguely defined as “the exchange of information that can occur either in real-time or later through technology” (IGI Global, 2021). Along this definition, the site I built has met the definition of information exchange by manner of “articles”, materials meant for self-study authored by teachers on the webpage (also interchangeably referred to as either “notes” or “memos” for the rest of this report), and “quizzes”, a multiple-choice-question style interface with no limitations on correct answer combinations, for students to test and evaluate their knowledge and understanding of various topics acquired at their own discretion.

2.1: Objectives

All objectives, both as laid out by the assignment requirements and self-imposed, have been successfully achieved. The application integrates with a local instance of XAMPP MySQL and associated tables. Operations of creation, read, update, deletion, basic registration and verification as well as CSS and JavaScript had been included.

Self-imposed objectives, such as reduced reliance on JavaScript, adequate provision of fallback from a scenario where JavaScript is disabled, no involvement of any framework, effort on aggressive optimization whenever appropriate, retaining maintainability to the best of my efforts, avoiding all forms of external aid and referencing aside from provided language manuals and W3Schools, and a generally performant site, had also been achieved.

Regarding general styling, I went for a 'clean' style with a dark palette, with minimal reliance on multimedia aside from a few images on the public-facing interface and SVGs for icons. Most user inputs to the page are emphasized by subtle but unobtrusive animations.

The rationale to this design choice is mainly for my own practice in UX and UI design, both of which I do not have any experience prior to this assignment. A 'clean' style reduces time taken for building relatively rudimentary designs and ease in determining color combinations, a desired characteristic mainly due to the nature of this assignment being rushed as a consequence of severely falling behind schedule.

The dark palette was settled upon due to its unintrusive nature across an exceptionally wide range of scenarios, which vary greatly in the context of online learning and the environment in which students carry it out.

My avoidance on frameworks, excessive styling and multimedia is in consideration of the demographic which this web application is directed at, which are both the students and teachers who will access this web application over a wide range of personal computers. While this application is designed mainly for desktop use, a large portion of devices accessing learning materials are low-end, such as Chromebooks, and would face difficulties in rendering webpages should they become too taxing on said system. With the lack of overhead that these frameworks usually require, potential points of failure that cause various technical issues can also be mitigated entirely.

On the topic of devices, as the web application is designed majorly for Chromebooks, who have a typical resolution of 1920x1080, the webpage has been styled accordingly for such devices. Other tested resolutions are 2560x1440 and 4096x2160.
SVGs are used in place of image files and other multimedia due to their scalability, allowing them to maintain legibility of their depictions under more extreme circumstances such as unusual screen resolutions or large zoom sizes.

The target audience for this web-facing application are for disparate groups of students seeking to self-study on various topics, as well as teachers or lecturers who aim to share their knowledge on topics of their expertise, with special consideration given to lower-end devices regardless of demographic. While being an interactive learning system, capabilities such as forum discussions or an online chat system are regrettably not implemented due to my lack of experience in such matters, difficulty of implementation and my lack of time to properly complete this assignment.

3.0: System design

3.1: Wireframes

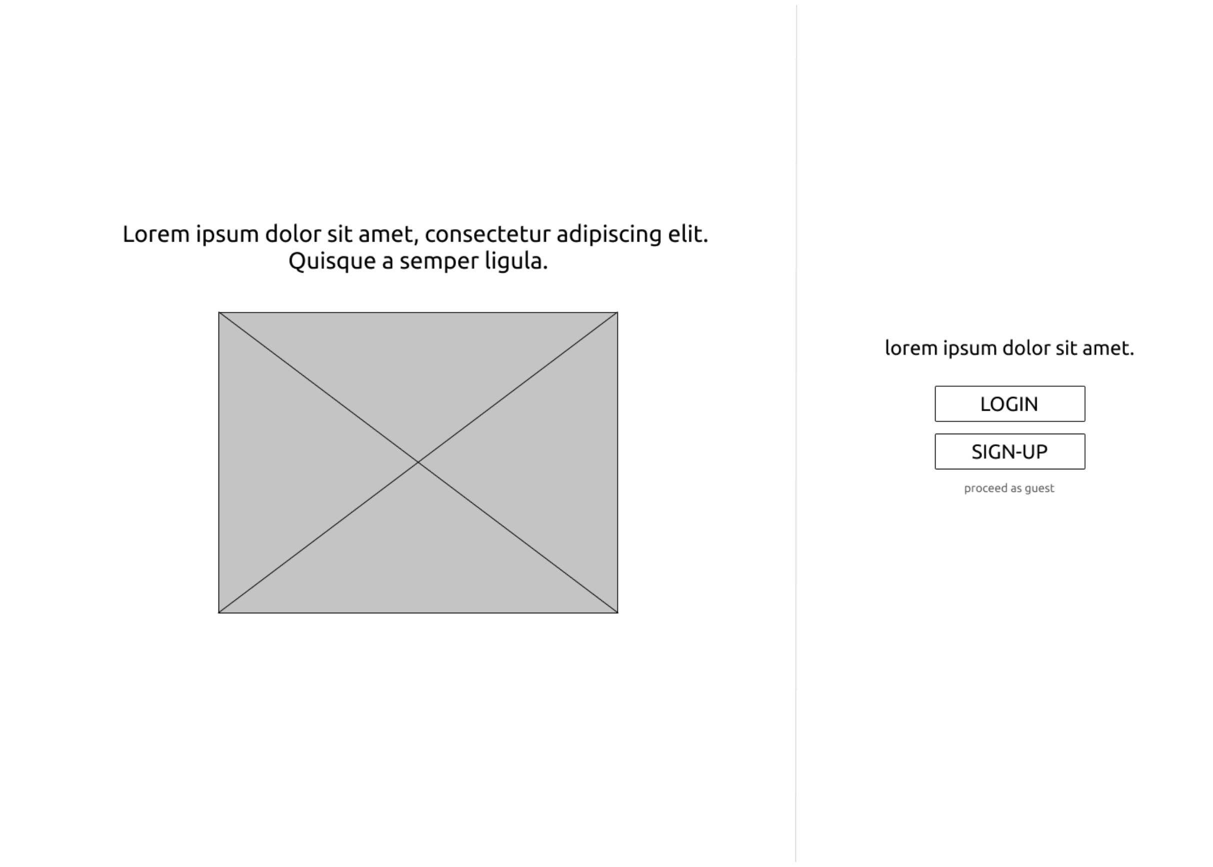

Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Quisque a semper ligula.
lorem ipsum dolor sit amet.
LOGIN
SIGN-UP
proceed as guest

Figure 4: Rough design of the visitor-facing introduction page.

Figure 4 shows the wireframe of what is intended to be the visitor-facing page, introducing the mission of the web application in a concise manner. It is a relatively standard layout where the majority of screen-space is dedicated to the splash text and appropriate imagery. To the right portion of the screen the visitor is offered to either log in to access contents of the web application, or register for an account.

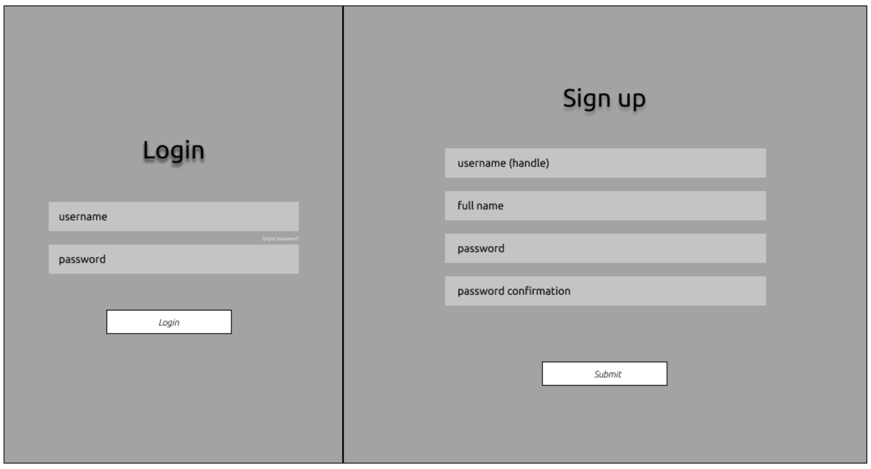


Figure 5: Wireframe of the combined register/login screen

Figure 5 shows the wireframe for a combined registration and log-in screen as planned out. With the demographic of students with relatively disadvantaged hardware in mind, the concept of a single-load page that accomplishes two basic functionalities, the log-in and registration features would be desirable for the sake of convenience.

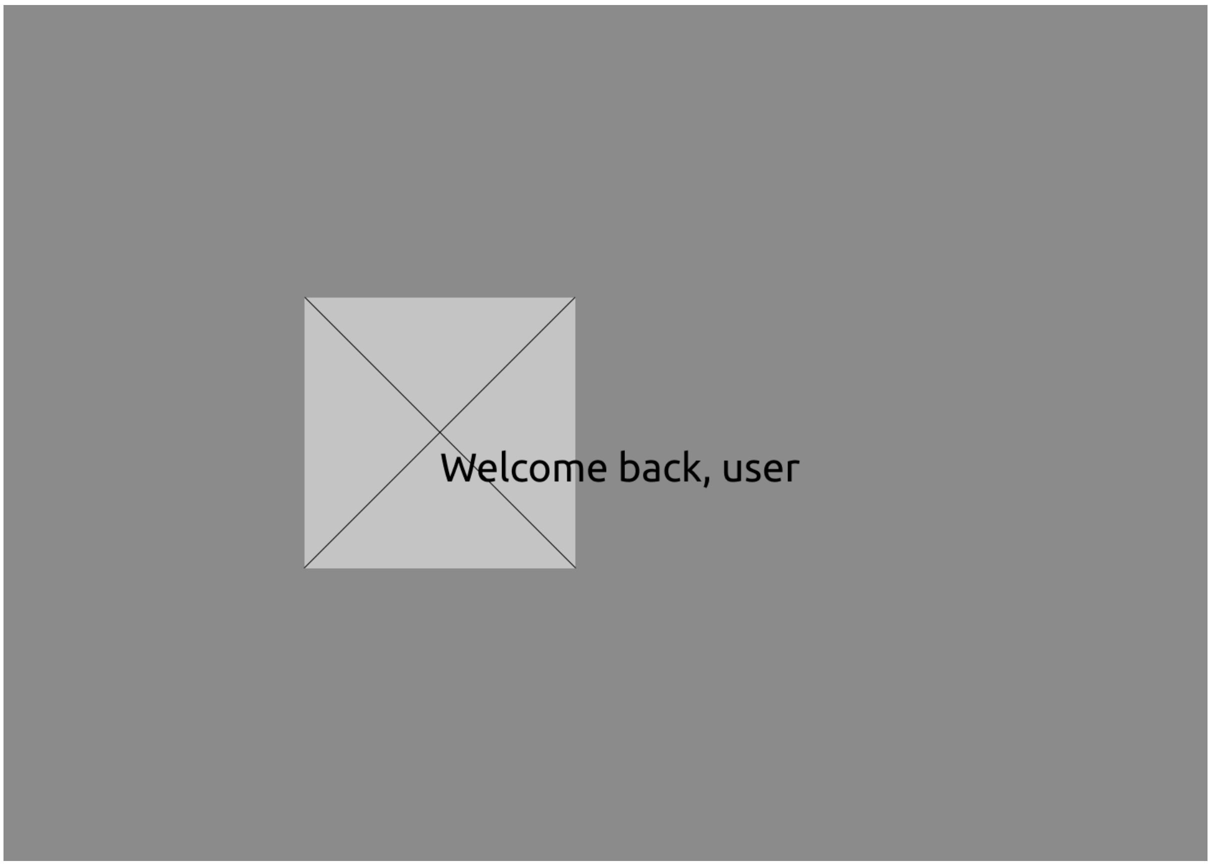


Figure 6: Wireframe of the main user interface.

The diagram shows the main user interface, that is only shown to users who have logged into the system. It is a standard "clean-cut" page, with text floating over an appropriate illustration. Further aesthetic improvements can be achieved with the aid of CSS scroll snapping, as more functionalities of the application are progressively revealed to the user by scrolling down from this screen.

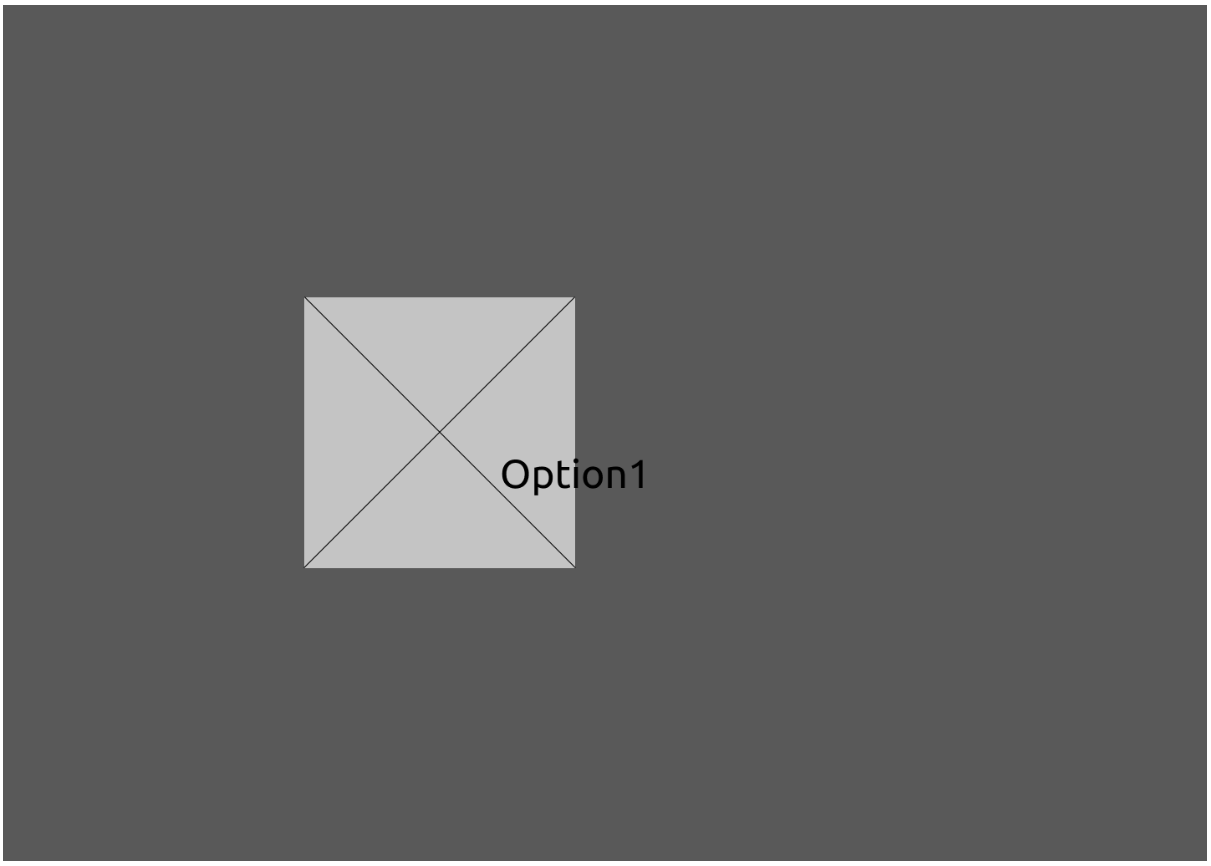


Figure 7: A "page" in the main user interface, revealed as the user scrolls down.

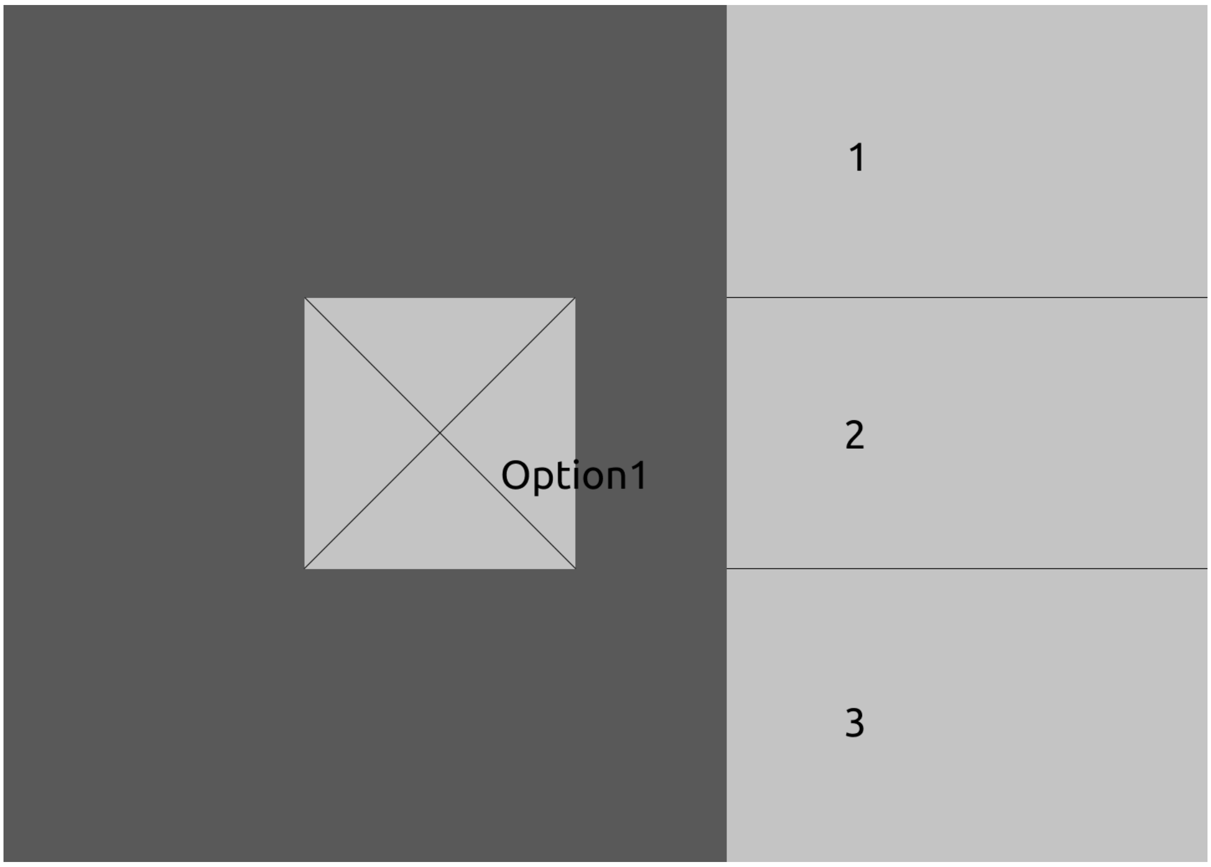


Figure 8: The "page" in the user interface showing additional options when hovered over.

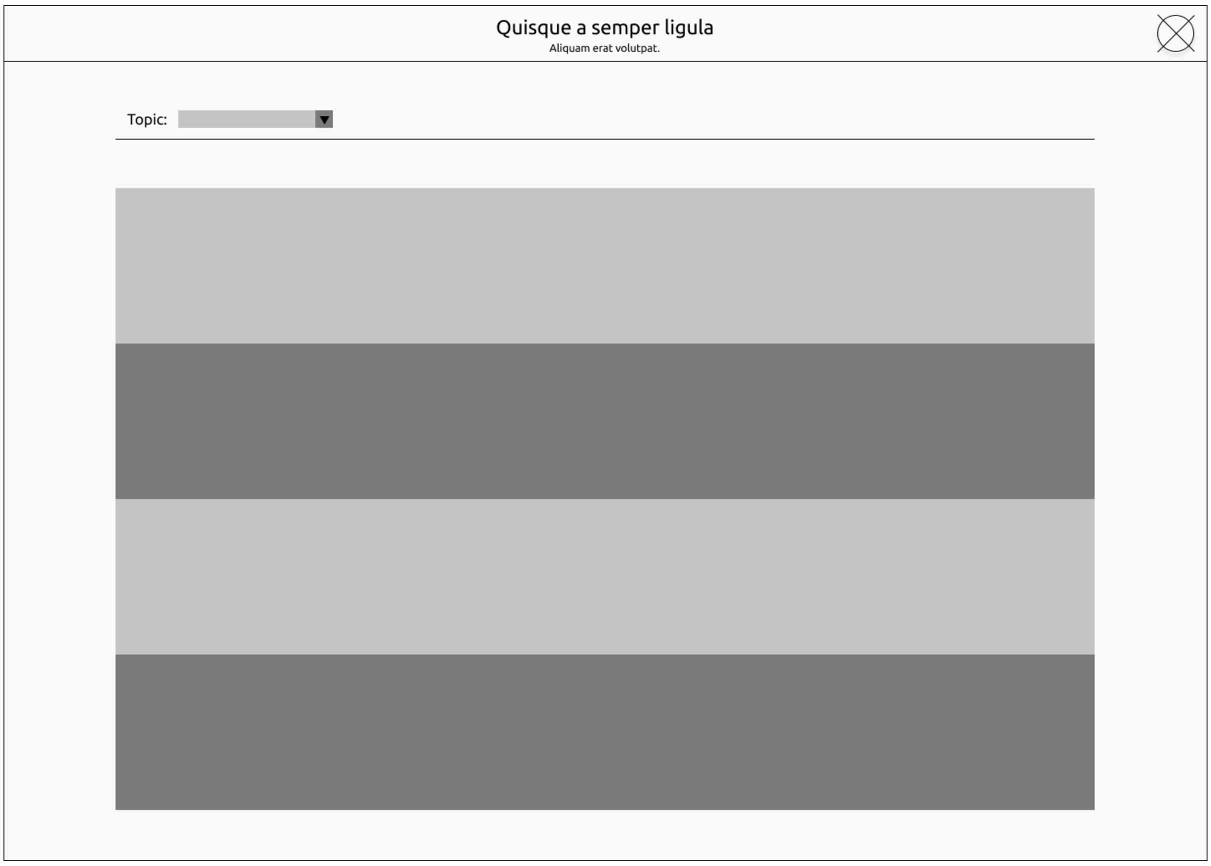
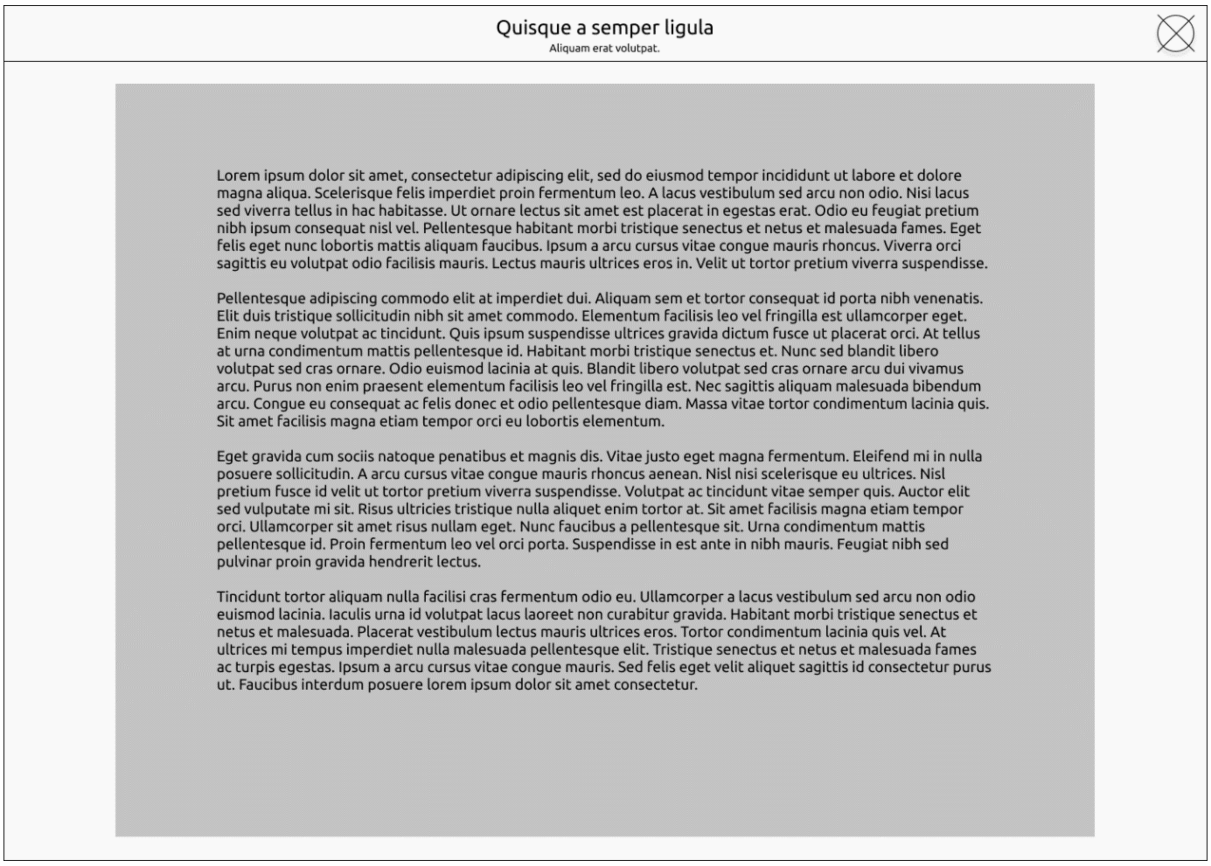


Figure 9: A page where users use to view content. An identical layout with the topic selection menu omitted is used for the interface during quizzes.

Figure 9 depicts the finalized design for the "view" page, which handles the viewing of content such as profiles, quizzes and memos. Further refinement of the query can be done by the user, by selecting a topic of choice from the 'Topic:' drop-down menu. The page then automatically reloads, displaying only content relevant to the topic.



Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Scelerisque felis imperdiet proin fermentum leo. A lacus vestibulum sed arcu non odio. Nisi lacus sed viverra tellus in hac habitasse. Ut ornare lectus sit amet est placerat in egestas erat. Odio eu feugiat pretium nibh ipsum consequat nisl vel. Pellentesque habitant morbi tristique senectus et netus et malesuada fames. Eget felis eget nunc lobortis mattis aliquam faucibus. Ipsum a arcu cursus vitae congue mauris rhoncus. Viverra orci sagittis eu volutpat odio facilisis mauris. Lectus mauris ultrices eros in. Velit ut tortor pretium viverra suspendisse.

Pellentesque adipiscing commodo elit at imperdiet dui. Aliquam sem et tortor consequat id porta nibh venenatis. Elit duis tristique sollicitudin nibh sit amet commodo. Elementum facilisis leo vel fringilla est ullamcorper eget. Enim neque volutpat ac tincidunt. Quis ipsum suspendisse ultrices gravida dictum fusce ut placerat orci. At tellus at urna condimentum mattis pellentesque id. Habitant morbi tristique senectus et. Nunc sed blandit libero volutpat sed cras ornare. Odio euismod lacinia at quis. Blandit libero volutpat sed cras ornare arcu dui vivamus arcu. Purus non enim praesent elementum facilisis leo vel fringilla est. Nec sagittis aliquam malesuada bibendum arcu. Congue eu consequat ac felis donec et odio pellentesque diam. Massa vitae tortor condimentum lacinia quis. Sit amet facilisis magna etiam tempor orci eu lobortis elementum.

Eget gravida cum sociis natoque penatibus et magnis dis. Vitae justo eget magna fermentum. Eleifend mi in nulla posuere sollicitudin. A arcu cursus vitae congue mauris rhoncus aenean. Nisl nisi scelerisque eu ultrices. Nisl pretium fusce id velit ut tortor pretium viverra suspendisse. Volutpat ac tincidunt vitae semper quis. Auctor elit sed vulputate mi sit. Risus ultricies tristique nulla aliquet enim tortor at. Sit amet facilisis magna etiam tempor orci. Ullamcorper sit amet risus nullam eget. Nunc faucibus a pellentesque sit. Urna condimentum mattis pellentesque id. Proin fermentum leo vel orci porta. Suspendisse in est ante in nibh mauris. Feugiat nibh sed pulvinar proin gravida hendrerit lectus.

Tincidunt tortor aliquam nulla facilisi cras fermentum odio eu. Ullamcorper a lacus vestibulum sed arcu non odio euismod lacinia. Iaculis urna id volutpat lacus laoreet non curabitur gravida. Habitant morbi tristique senectus et netus et malesuada. Placerat vestibulum lectus mauris ultrices eros. Tortor condimentum lacinia quis vel. At ultrices mi tempus imperdiet nulla malesuada pellentesque elit. Tristique senectus et netus et malesuada fames ac turpis egestas. Ipsum a arcu cursus vitae congue mauris. Sed felis eget velit aliquet sagittis id consectetur purus ut. Faucibus interdum posuere lorem ipsum dolor sit amet consectetur.

Figure 10: A conceptual layout for article display.

This wireframe displays the layout for the page handling the reader view for articles.

3.2: Navigational structure

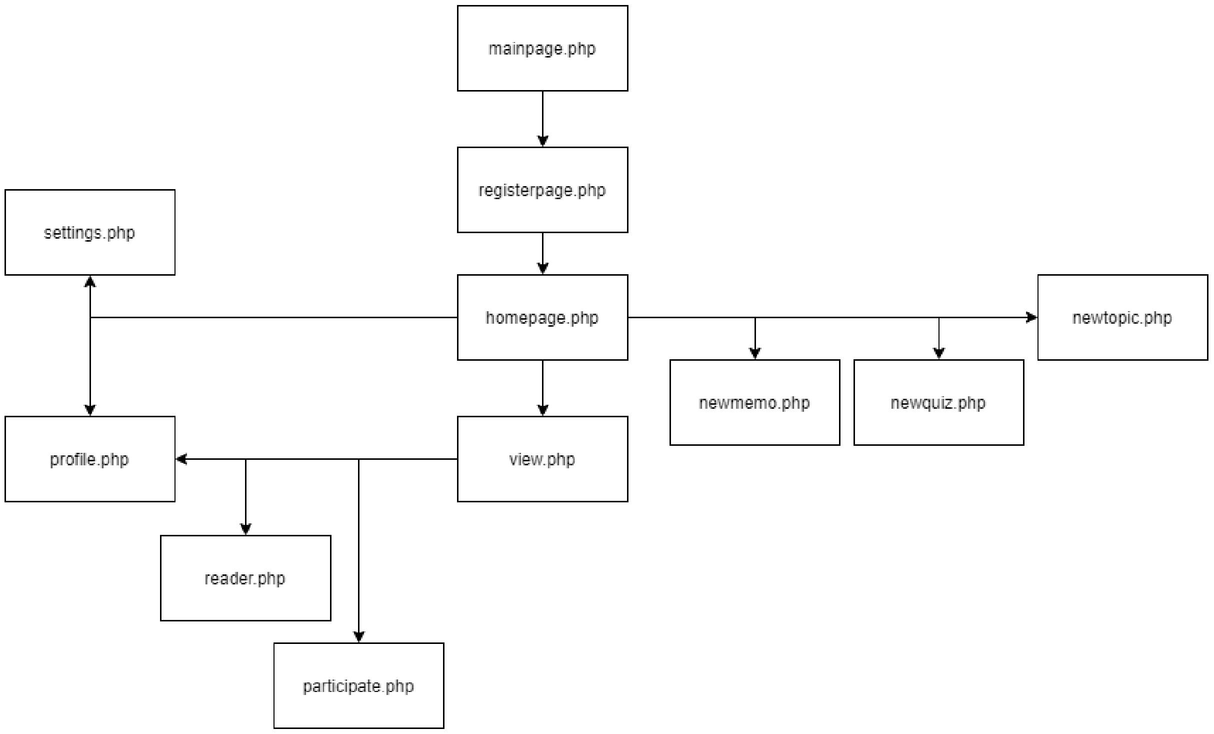


Figure 11: The intended navigational structure for the web application.

3.3: Activity diagrams

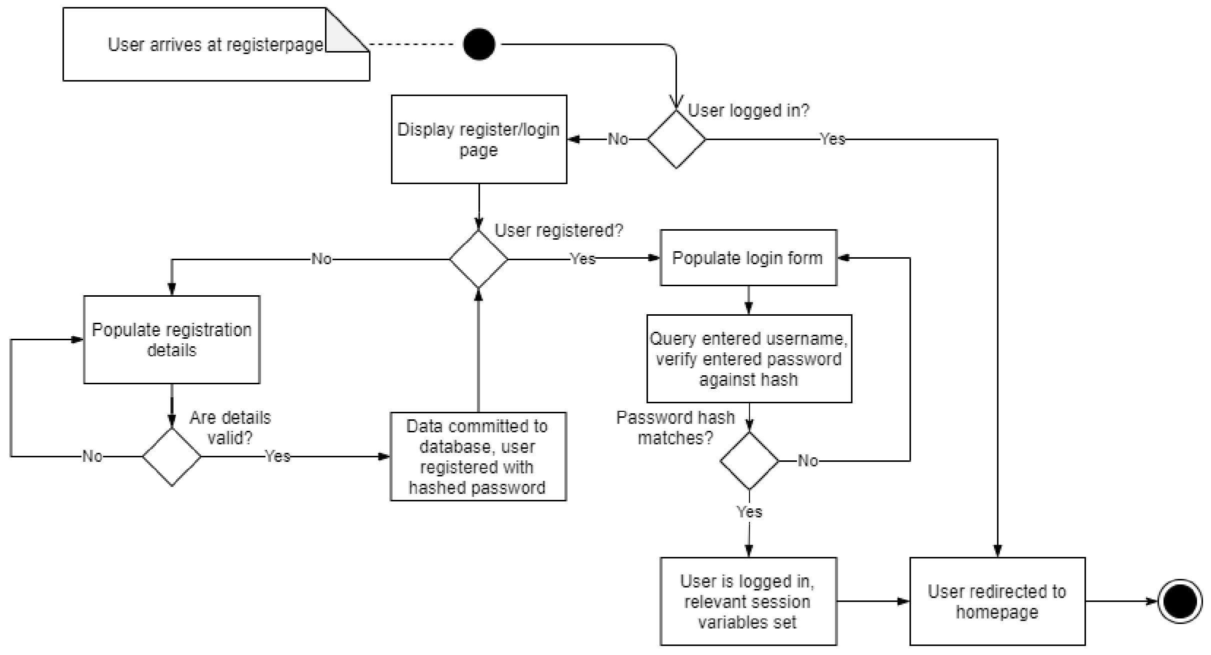

User arrives at registerpage
User logged in?
No
Yes
Display register/login page
User registered?
No
Yes
Populate login form
Populate registration details
Query entered username, verify entered password against hash
Are details valid?
No
Yes
Data committed to database, user registered with hashed password
Password hash matches?
No
Yes
User is logged in, relevant session variables set
User redirected to homepage

Figure 12: Activity diagram for the combined login-registration system

Figure 12 shows an abstraction for the registration and log-in system for the web application. The page starts by checking if an index in the session variable corresponding to a username is set, which is synonymous to a user having logged into the site at some point in time prior. In that case, the user is redirected to the homepage, and if not, the combined login and registration page would be displayed.

Registration follows the standard steps of verification of entered data, where the user would be rejected in the event of invalid data or an incorrectly confirmed password. A valid registration involves the system building a query for an addition of new user data to the database, after the password string given by the user has been encrypted. Validation of input data such as password length can be accomplished by *RegEx* patterns on the client-side, with the help of the latest HTML standards.

Logging in only involves the submission of a filled log-in form, which is then queried against the database. The provided password is hashed and compared against the recorded has within the database, rejecting the user if it does not match. In the event of a successful verification, relevant session variables are set and the user is redirected to the homepage.

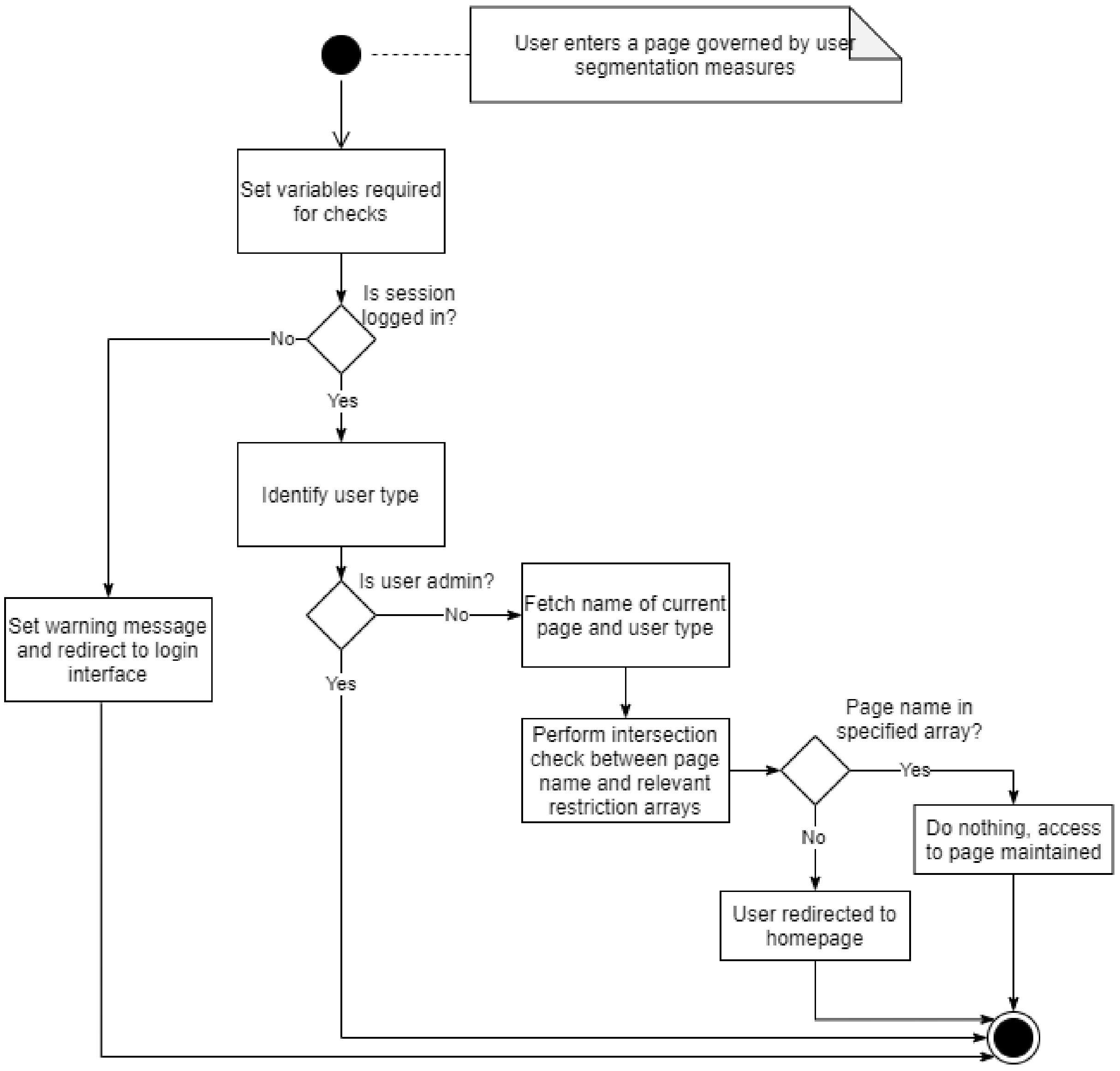

User enters a page governed by user segmentation measures
Set variables required for checks
Is session logged in?
No
Yes
Identify user type
Is user admin?
No
Yes
Set warning message and redirect to login interface
Fetch name of current page and user type
Perform intersection check between page name and relevant restriction arrays
Page name in specified array?
Yes
No
Do nothing, access to page maintained
User redirected to homepage

Figure 13: Diagram for the user segmentation/"privilege" system

Figure 13 shows a simplification for a user segmentation system, designed to limit certain pages to certain groups of users of different permission levels, while simultaneously controlling access of visitors or unregistered users to portions of the site. This system is intended to prevent the hypothetical situation of a student account being able to access administrator functionality, as an example.

Arrays named after the 3 privilege levels used within the system, *student*, *teacher* and *admin*, are initialized, while pages where each user-type is blocked from are manually defined within code. This paradigm makes it easier to implement this manner of content blocking in code, as inbuilt functions can then be used against data laid out in this manner, significantly improving code conciseness.

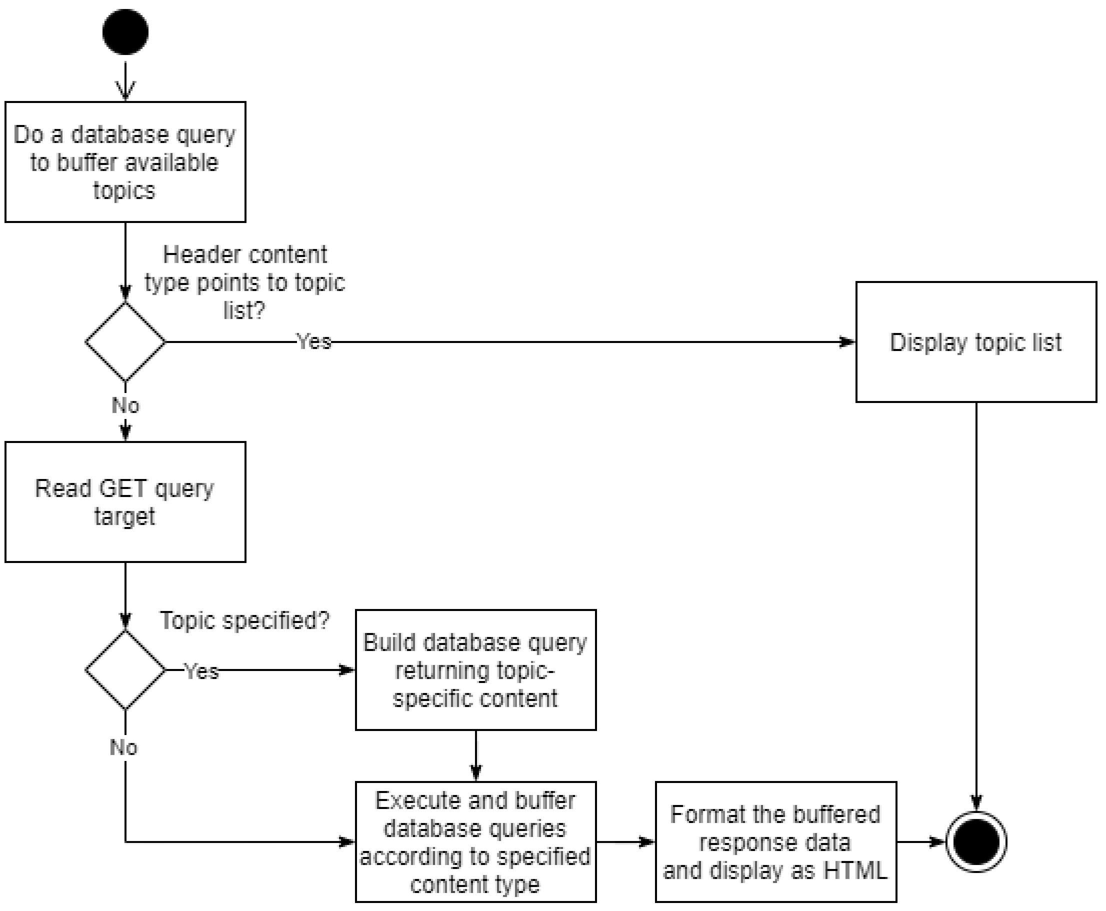

Do a database query to buffer available topics
Header content type points to topic list?
Yes
Display topic list
No
Read GET query target
Topic specified?
Yes
Build database query returning topic-specific content
No
Execute and buffer database queries according to specified content type
Format the buffered response data and display as HTML

Figure 14: Diagram for the dynamic content viewing handler

The system also involves a dynamic content viewing handler to build around the various types of content served by the web application. It is relatively simple, as it only involves querying the database with relevant query strings and buffering records to be formatted into appropriate HTML at the end of the process.

While being conceptually simple, the implementation in code is comparatively verbose and cluttered due to the differing HTML styling and queries required corresponding to each view mode. This plan, however, makes potential extensions to the system as easy as adding additional checking criteria corresponding to the additional content.

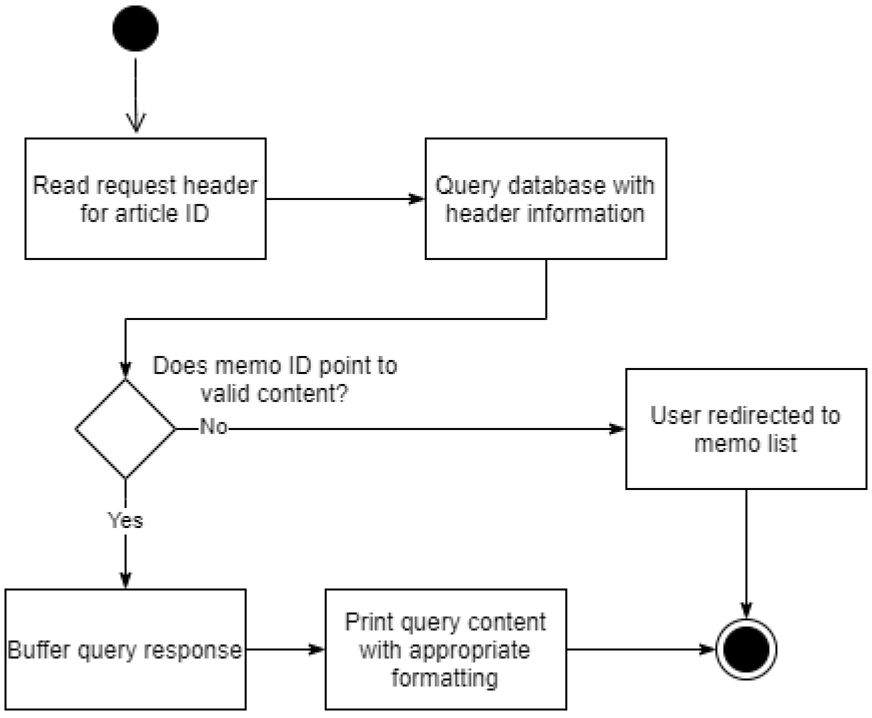
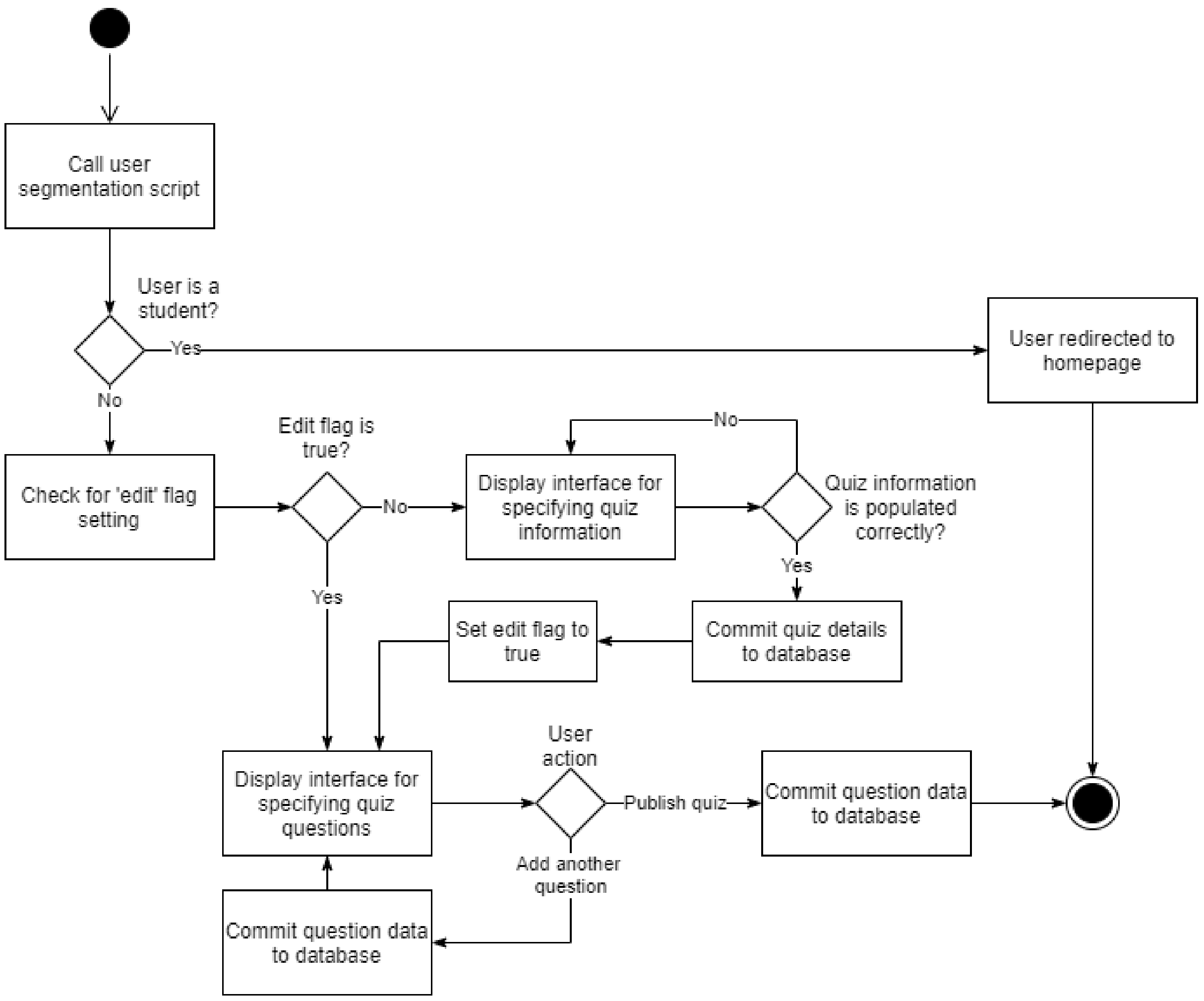

Read request header for article ID
Query database with header information
Does memo ID point to valid content?
No
User redirected to memo list
Yes
Buffer query response
Print query content with appropriate formatting

Figure 15: Diagram for a relatively simple article displaying system

Articles, also referred to as memos in Figure 15, while being critical content hosted on this web application, are however relatively simple to display.
This is due to their nature of being easily addressable from query return arrays, which reduces the complexity of the implementation. The only steps involved are obtaining the query string pointing to the article ID, which is then fed into the database query that returns data for a single article.
Thus, no further processing is required due to the nature of an article ID being unique to each article.



Call user segmentation script
User is a student?
Yes
User redirected to homepage
No
Check for 'edit' flag setting
Edit flag is true?
No
Display interface for specifying quiz information
No
Quiz information is populated correctly?
Yes
Commit quiz details to database
Set edit flag to true
Yes
Display interface for specifying quiz questions
User action
Publish quiz
Commit question data to database
Add another question
Commit question data to database

Figure 16: Diagram showing the quiz addition system

The other main feature of this web application is to aid teachers in facilitating quizzes. This system thus handles the creation of a new quiz. Users which have insufficient privileges, such as students, are denied access to the page, while teachers and admins first get checked if a Boolean “edit” flag is set to true, a session variable used within this system. Users who have just finished creating a quiz, or have not created a quiz prior, will be directed to the quiz information form where they can fill in the details of their quiz. Submitting valid quiz information inverts the “edit” flag, which reloads this page with a different layout. Completing and submitting this form commits the entered quiz details to the database and reloads the same interface, where the users can add another question to the quiz. They can hence start writing quiz content in the page with no limit on question quantity or correct options, which unlike standard Multiple-Choice Questions, can range from having no answer at all to 4 simultaneous correct choices.

When the user has finished composing the quiz, they can click on a separate form to complete authoring of the quiz and return to homepage.

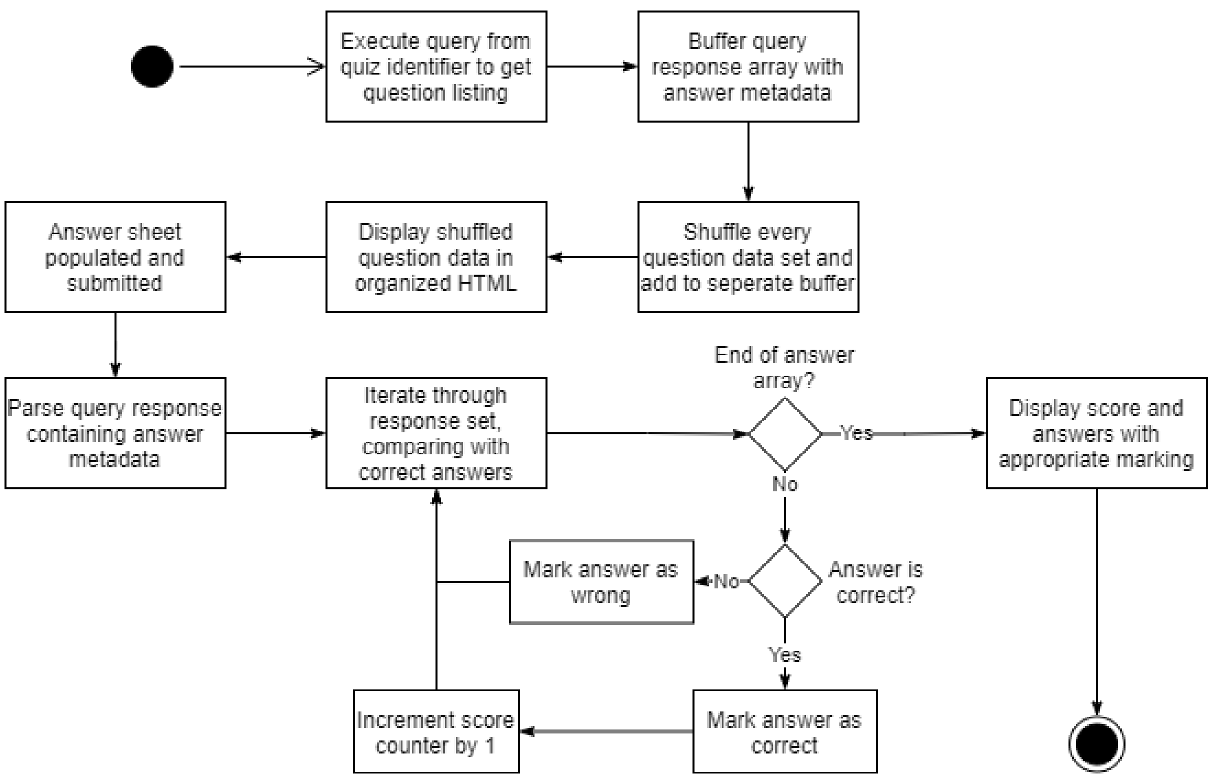

Execute query from quiz identifier to get question listing
Buffer query response array with answer metadata
Shuffle every question data set and add to seperate buffer
Display shuffled question data in organized HTML
Answer sheet populated and submitted
Parse query response containing answer metadata
Iterate through response set, comparing with correct answers
End of answer array?
Yes
Display score and answers with appropriate marking
No
Answer is correct?
No
Mark answer as wrong
Yes
Mark answer as correct
Increment score counter by 1

Figure 17: An abstraction explaining the quiz display, submission and marking system

The last feature of note, being the most complex to implement, is the question serving, submission, and marking system. The displaying of quiz content is the simplest, being a database query with the quiz identifier, whose response is then shuffled and buffered into an array for display as HTML in a form on the user-end.

The user taking the quiz, would then populate the form and submit it. The submitted form data would be split into arrays for special processing, marking questions right or wrong based on the answers provided, while incrementing the score accordingly.

If the system has reached the end of the question array, a page showing the mistakes, markings by the system, recorded answers and the score would be displayed.

While quiz answer choices are shuffled, the quiz sequence is intentionally left undisturbed due to some teachers finding quiz question sequence critical in certain scenarios, such as a succeeding question referencing information from questions prior.

Adhering to my own motive of creating a platform allowing for students to appropriately self-study, the feature of allowing variable “correct-answer” choices has led to an interesting problem to implement in code during the marking procedure, as standard “compare” operations proved inaccurate in the marking of the quiz due to quiz answer shuffling in prior operations.

The response page displays the score and markings for the quiz-taker, where wrong or missed choices would be highlighted in red, and correct answers highlighted green. The issue of accurate marking is solved by manner of combinations of array operations, operating on answer sets based on the “classification” of the error found.

3.4: Entity Relationship Diagram

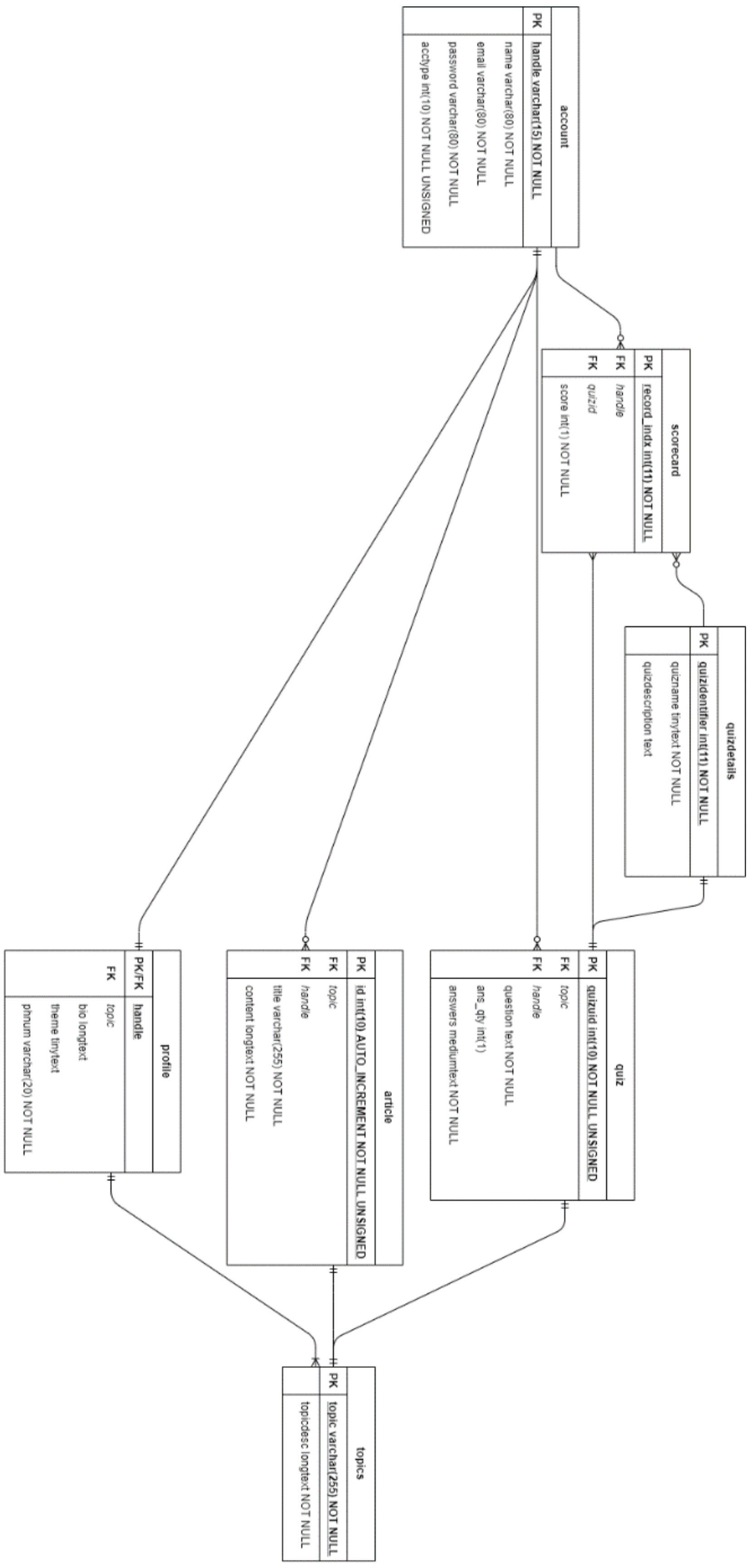
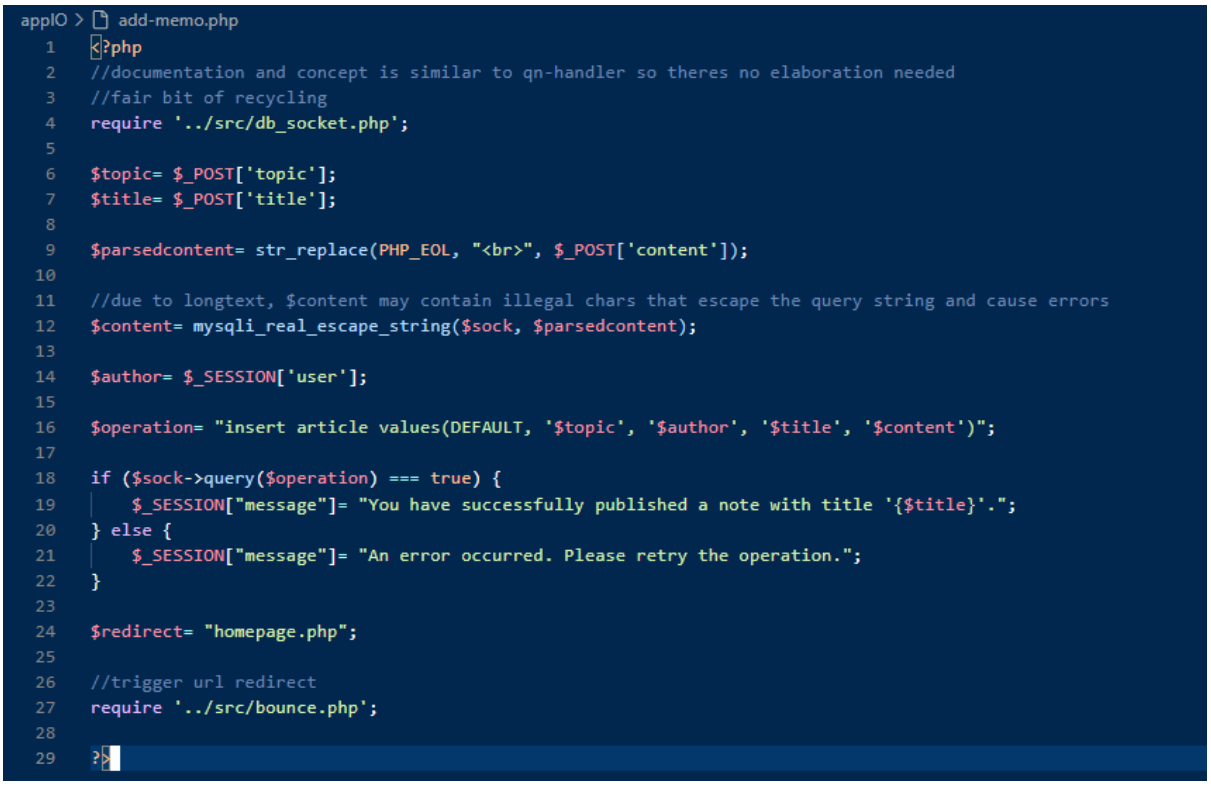

account
PK handle varchar(15) NOT NULL
name varchar(80) NOT NULL
email varchar(80) NOT NULL
password varchar(80) NOT NULL
acctype int(10) NOT NULL UNSIGNED
scorecard
PK record_indx int(11) NOT NULL
FK handle
FK quizid
score int(1) NOT NULL
quizdetails
PK quizidentifier int(11) NOT NULL
quizname tinytext NOT NULL
quizdescription text
profile
PK/FK handle
FK topic
bio longtext
theme tinytext
phnum varchar(20) NOT NULL
article
PK id int(10) AUTO_INCREMENT NOT NULL UNSIGNED
FK topic
FK handle
title varchar(255) NOT NULL
content longtext NOT NULL
quiz
PK quizuid int(10) NOT NULL UNSIGNED
FK topic
FK handle
question text NOT NULL
ans_qty int(1)
answers mediumtext NOT NULL
topics
PK topic varchar(255) NOT NULL
topicdesc longtext NOT NULL

Figure 18: The ERD, showing the layout, datatypes and relations of the finalized database

4.0: Implementation

4.1: Create

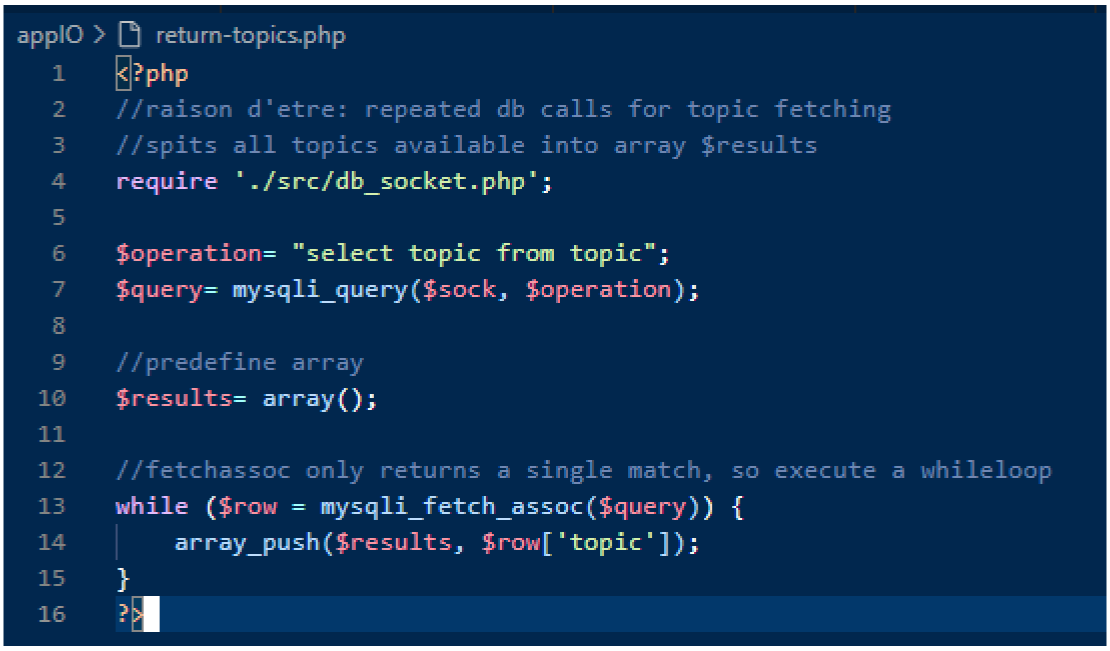


```
applO > add-memo.php
<?php
//documentation and concept is similar to qn-handler so theres no elaboration needed
//fair bit of recycling
require '../src/db_socket.php';

$topic= $_POST['topic'];
$title= $_POST['title'];

$parsedcontent= str_replace(PHP_EOL, "<br>", $_POST['content']);

//due to longtext, $content may contain illegal chars that escape the query string and cause errors
$content= mysqli_real_escape_string($sock, $parsedcontent);

$author= $_SESSION['user'];

$operation= "insert article values(DEFAULT, '$topic', '$author', '$title', '$content')";

if ($sock->query($operation) === true) {
    $_SESSION["message"]= "You have successfully published a note with title '{$title}'.";
} else {
    $_SESSION["message"]= "An error occurred. Please retry the operation.";
}

$redirect= "homepage.php";

//trigger url redirect
require '../src/bounce.php';

?>
```

Figure 19: Occurrence of a Create operation

The example for a **Create** operation used in this project is the add-memo script. It is a PHP script called on the event of an article submission by a user. In the context of this operation, the raw article text data passed to this script is unsafe if directly passed within an SQL query due to the possibility of arbitrary query execution by escape sequences. Hence, the function *mysqli_real_escape_string()* is used to sanitize the raw string data to prevent such attacks. The preceding *str_replace()* function serves only for the article to maintain newline formatting, as the raw article string data will be lost if passed through *mysqli_real_escape_string()* verbatim. After sanitization, the *INSERT* query is built and executed, returning a feedback message depending on the outcome. The outcome of the operation, in this case the publishing of an article, will hence be displayed to the user in the *holdingpage.php* file.

4.2: Read



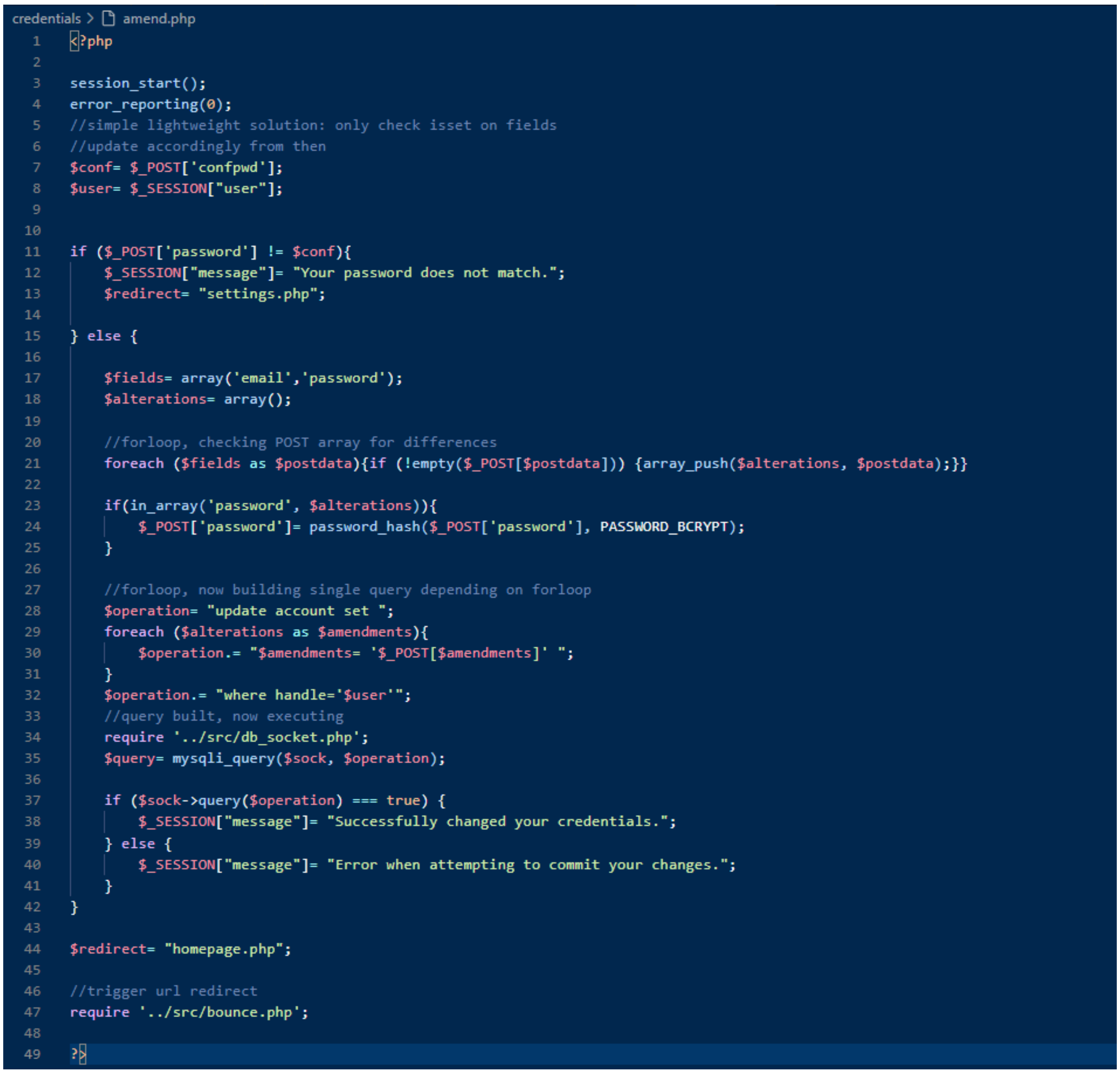
appIO > return-topics.php

```php
<?php
//raison d'etre: repeated db calls for topic fetching
//spits all topics available into array $results
require './src/db_socket.php';

$operation= "select topic from topic";
$query= mysqli_query($sock, $operation);

//predefine array
$results= array();

//fetchassoc only returns a single match, so execute a whileloop
while ($row = mysqli_fetch_assoc($query)) {
    array_push($results, $row['topic']);
}
?>
```

Figure 20: Occurrence of a Read operation

An example for a **Read** operation is the relatively simple *return-topics* script. This script simply queries the database table *topic* to fetch all available topics, funnelling it into the array *$results* for easy usage by other parts of the page with a *require* statement.

4.3: Update



credentials > amend.php

```php
<?php

session_start();
error_reporting(0);
//simple lightweight solution: only check isset on fields
//update accordingly from then
$conf= $_POST['confpwd'];
$user= $_SESSION["user"];


if ($_POST['password'] != $conf){
    $_SESSION["message"]= "Your password does not match.";
    $redirect= "settings.php";

} else {

    $fields= array('email','password');
    $alterations= array();

    //forloop, checking POST array for differences
    foreach ($fields as $postdata){if (!empty($_POST[$postdata])) {array_push($alterations, $postdata);}}

    if(in_array('password', $alterations)){
        $_POST['password']= password_hash($_POST['password'], PASSWORD_BCRYPT);
    }

    //forloop, now building single query depending on forloop
    $operation= "update account set ";
    foreach ($alterations as $amendments){
        $operation.= "$amendments= '$_POST[$amendments]' ";
    }
    $operation.= "where handle='$user'";
    //query built, now executing
    require '../src/db_socket.php';
    $query= mysqli_query($sock, $operation);

    if ($sock->query($operation) === true) {
        $_SESSION["message"]= "Successfully changed your credentials.";
    } else {
        $_SESSION["message"]= "Error when attempting to commit your changes.";
    }
}

$redirect= "homepage.php";

//trigger url redirect
require '../src/bounce.php';

?>
```

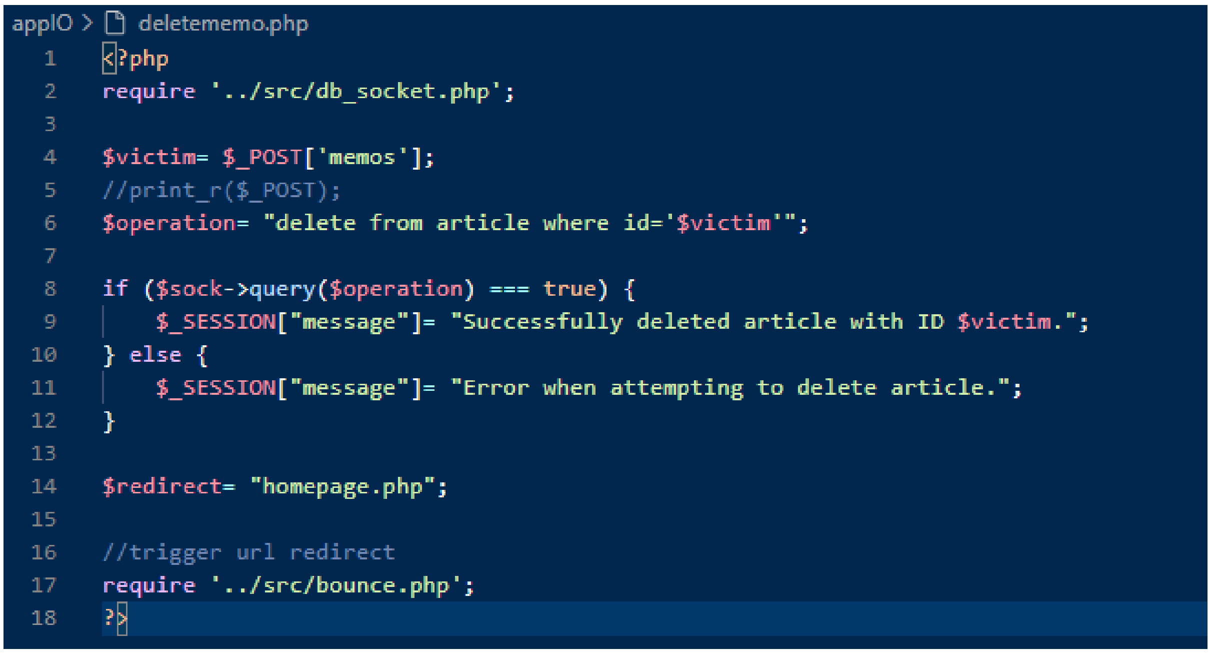
Figure 21: Example of an Update operation

An occurrence of an **Update** operation in this web application is in the *amend.php* script, called when a user accesses the *Settings* option, verifies himself and makes any changes to personal data.

The script checks the "*confpwd*" field (Confirm new password) within the POST headers for a match with the given new password if it were to be amended by the user. If the check passes, the script proceeds to do a check for amendments within the POST headers, hashing the provided password and buffering it if it were to be given by the user. The script then does a variable string append operation to procedurally build the query, due to the nature of the SQL *UPDATE* statement being verbose in nature.

The script ends by execution of the concatenated query with the feedback message corresponding to query status before the user is redirected back to the homepage.

4.4: Delete



appIO > deletememo.php

```php
<?php
require '../src/db_socket.php';

$victim= $_POST['memos'];
//print_r($_POST);
$operation= "delete from article where id='$victim'";

if ($sock->query($operation) === true) {
    $_SESSION["message"]= "Successfully deleted article with ID $victim.";
} else {
    $_SESSION["message"]= "Error when attempting to delete article.";
}

$redirect= "homepage.php";

//trigger url redirect
require '../src/bounce.php';
?>
```

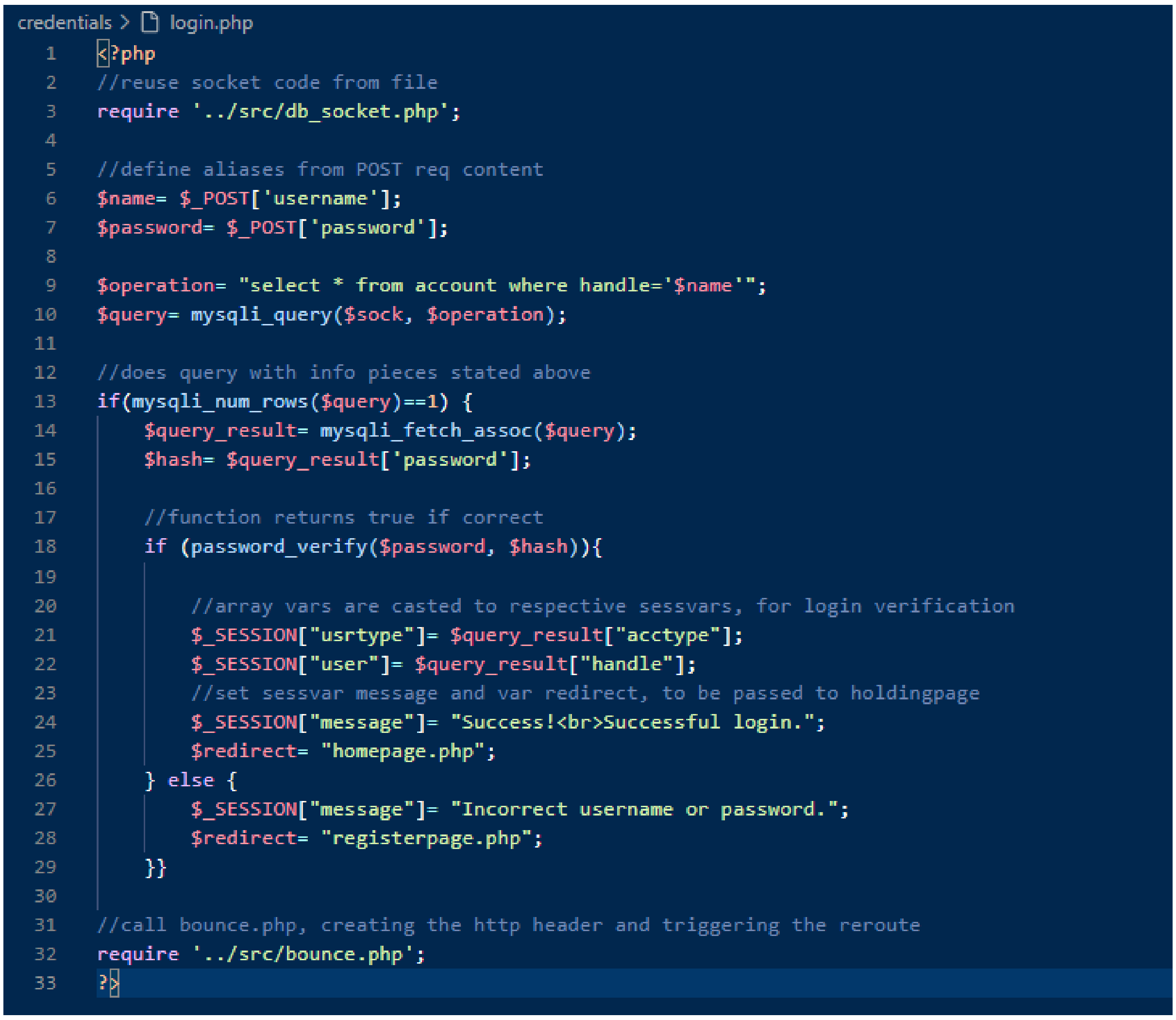
Figure 22: Example of a Delete operation

An instance of the **Delete** operation takes place during the *deletememo.php* script, called as an administrative function to remove an offending article from the database. It is just as simple as the **Read** operation, where the script receives the target article by ID to be deleted.

In the context of this script, the variable *$victim* is a simplification for the POST header value, keyed as *memos*. This header contains the ID of the memo to be deleted, where the SQL statement *DELETE* is called with a *WHERE* clause. The *WHERE* clause specifies the specific memo to be deleted, by ID. This ensures that even in the event of an accidental execution by any circumstance, the query will not delete the entire *article* database.

Similar to past examples, a feedback message would be set depending on the outcome of the SQL operation, and the user is then redirected accordingly.

4.5: Login



credentials > login.php

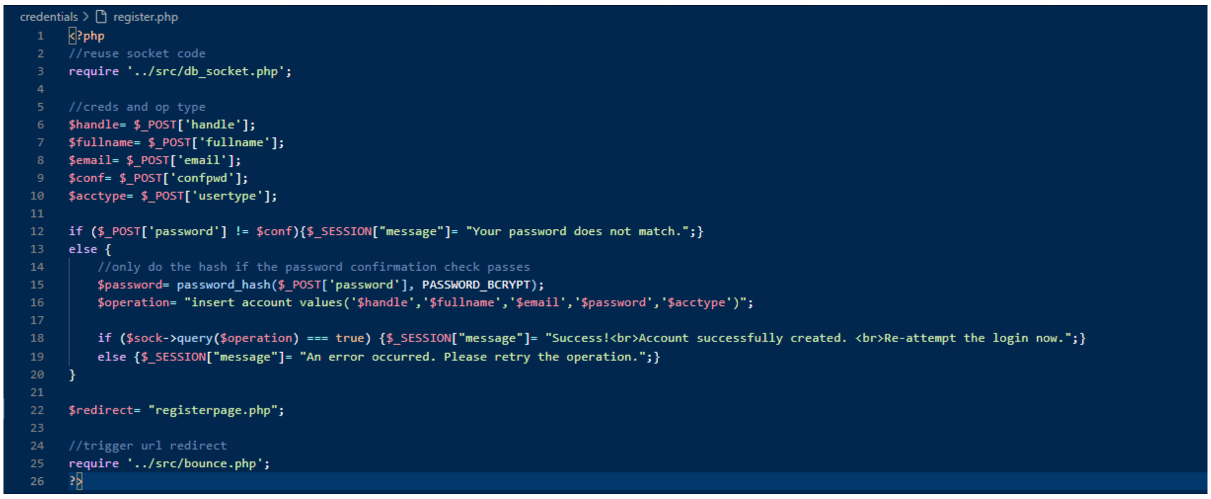
```php
<?php
//reuse socket code from file
require '../src/db_socket.php';

//define aliases from POST req content
$name= $_POST['username'];
$password= $_POST['password'];

$operation= "select * from account where handle='$name'";
$query= mysqli_query($sock, $operation);

//does query with info pieces stated above
if(mysqli_num_rows($query)==1) {
    $query_result= mysqli_fetch_assoc($query);
    $hash= $query_result['password'];

    //function returns true if correct
    if (password_verify($password, $hash)){

        //array vars are casted to respective sessvars, for login verification
        $_SESSION["usrtype"]= $query_result["acctype"];
        $_SESSION["user"]= $query_result["handle"];
        //set sessvar message and var redirect, to be passed to holdingpage
        $_SESSION["message"]= "Success!<br>Successful login.";
        $redirect= "homepage.php";
    } else {
        $_SESSION["message"]= "Incorrect username or password.";
        $redirect= "registerpage.php";
    }}

//call bounce.php, creating the http header and triggering the reroute
require '../src/bounce.php';
?>
```

Figure 23: Code snippet for the login system

The login operation involves the calling of the *db_socket* script in order to fetch database credentials. This script, which handles the authentication process is invoked when the log-in form is submitted.

It first defines alias variables for ease of reference to certain POST array variables later. The database query searches for the inputted username, or *handle* of the user logging in, whose aim is to fetch the password hash for the user computed during the registration process prior. The hash of the inputted password is then verified against the hash obtained from the database. On a failure to verify, the user is redirected back to the authentication interface. On a successful verification, session variables governing the user type and authentication state is set, and the user is redirected to the main page.

4.6: Signup



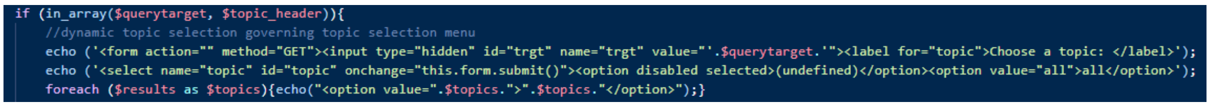
credentials > register.php

```php
<?php
//reuse socket code
require '../src/db_socket.php';

//creds and op type
$handle= $_POST['handle'];
$fullname= $_POST['fullname'];
$email= $_POST['email'];
$conf= $_POST['confpwd'];
$acctype= $_POST['usertype'];

if ($_POST['password'] != $conf){$_SESSION["message"]= "Your password does not match.";}
else {
    //only do the hash if the password confirmation check passes
    $password= password_hash($_POST['password'], PASSWORD_BCRYPT);
    $operation= "insert account values('$handle','$fullname','$email','$password','$acctype')";

    if ($sock->query($operation) === true) {$_SESSION["message"]= "Success!<br>Account successfully created. <br>Re-attempt the login now.";}
    else {$_SESSION["message"]= "An error occurred. Please retry the operation.";}
}

$redirect= "registerpage.php";

//trigger url redirect
require '../src/bounce.php';
?>
```

Figure 24: Code snippet for the signup operation

The signup process is comparatively simple due to it only requiring the creation of a query string. The *db_socket* script is invoked, and alias variables allowing ease of access to POST array variables are set. The preliminary check verifies if the “confirm password” field from the registration screen matches that of the main password input. A mismatch redirects the user back to the registration interface to refill the form, while a match proceeds with the system hashing the password. The query is then built and submitted accordingly, with feedback messages displayed based on query status.

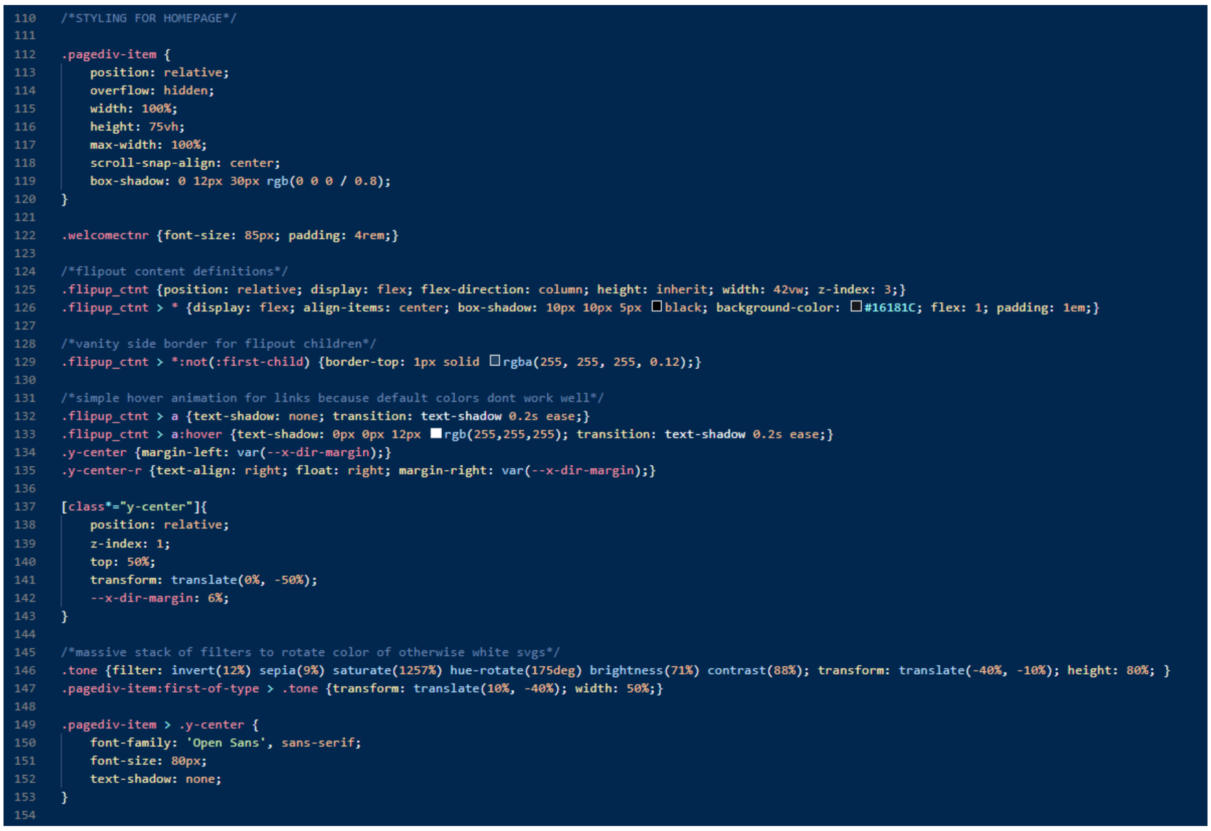
4.7: CSS and JavaScript



```
if (in_array($querytarget, $topic_header)){
    //dynamic topic selection governing topic selection menu
    echo ('<form action="" method="GET"><input type="hidden" id="trgt" name="trgt" value="'.$querytarget.'"><label for="topic">Choose a topic: </label>');
    echo ('<select name="topic" id="topic" onchange="this.form.submit()"><option disabled selected>(undefined)</option><option value="all">all</option>');
    foreach ($results as $topics){echo("<option value=".$topics.">".$topics."</option>");}
```

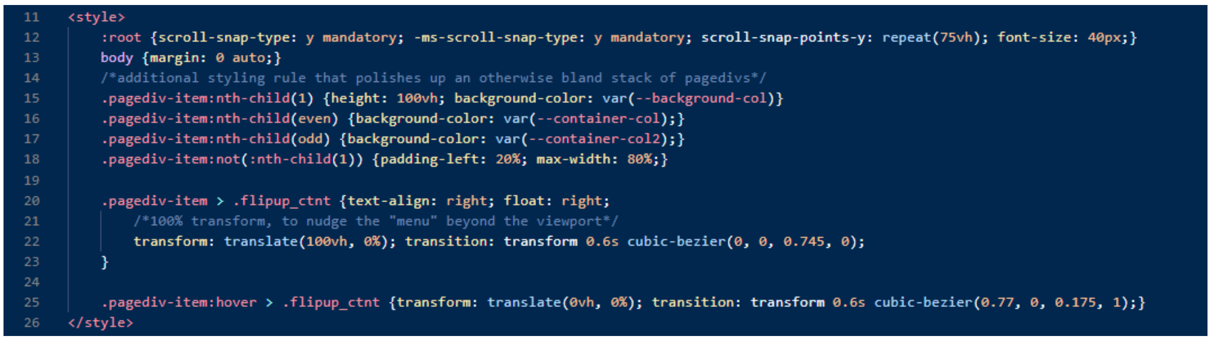
Figure 25: Example of JavaScript usage, embedded in an ***onchange*** *attribute*

In the development of this web application, I have seldom found use for JavaScript due to most of its potential usecases being replaced by innate capabilities introduced by HTML and CSS, such as the *pattern* attribute for input validation, or CSS animations being more performant than JavaScript (MDN contributors, 2021). It has, however, been useful in specific cases such as hot-reloading pages upon selection of an item from a drop-down menu. The figure above shows a fragment of embedded JavaScript, executed by the attribute *onchange* within the *select* tag on the 4th line of the figure.



```css
/*STYLING FOR HOMEPAGE*/

.pagediv-item {
    position: relative;
    overflow: hidden;
    width: 100%;
    height: 75vh;
    max-width: 100%;
    scroll-snap-align: center;
    box-shadow: 0 12px 30px rgb(0 0 0 / 0.8);
}

.welcomectnr {font-size: 85px; padding: 4rem;}

/*flipout content definitions*/
.flipup_ctnt {position: relative; display: flex; flex-direction: column; height: inherit; width: 42vw; z-index: 3;}
.flipup_ctnt > * {display: flex; align-items: center; box-shadow: 10px 10px 5px black; background-color: #16181C; flex: 1; padding: 1em;}

/*vanity side border for flipout children*/
.flipup_ctnt > *:not(:first-child) {border-top: 1px solid rgba(255, 255, 255, 0.12);}

/*simple hover animation for links because default colors dont work well*/
.flipup_ctnt > a {text-shadow: none; transition: text-shadow 0.2s ease;}
.flipup_ctnt > a:hover {text-shadow: 0px 0px 12px rgb(255,255,255); transition: text-shadow 0.2s ease;}
.y-center {margin-left: var(--x-dir-margin);}
.y-center-r {text-align: right; float: right; margin-right: var(--x-dir-margin);}

[class*="y-center"]{
    position: relative;
    z-index: 1;
    top: 50%;
    transform: translate(0%, -50%);
    --x-dir-margin: 6%;
}

/*massive stack of filters to rotate color of otherwise white svgs*/
.tone {filter: invert(12%) sepia(9%) saturate(1257%) hue-rotate(175deg) brightness(71%) contrast(88%); transform: translate(-40%, -10%); height: 80%; }
.pagediv-item:first-of-type > .tone {transform: translate(10%, -40%); width: 50%;}

.pagediv-item > .y-center {
    font-family: 'Open Sans', sans-serif;
    font-size: 80px;
    text-shadow: none;
}
```

Figure 26: Example of a segment of CSS use in a stylesheet, usage aimed at the homepage part of the application.



```html
<style>
    :root {scroll-snap-type: y mandatory; -ms-scroll-snap-type: y mandatory; scroll-snap-points-y: repeat(75vh); font-size: 40px;}
    body {margin: 0 auto;}
    /*additional styling rule that polishes up an otherwise bland stack of pagedivs*/
    .pagediv-item:nth-child(1) {height: 100vh; background-color: var(--background-col)}
    .pagediv-item:nth-child(even) {background-color: var(--container-col);}
    .pagediv-item:nth-child(odd) {background-color: var(--container-col2);}
    .pagediv-item:not(:nth-child(1)) {padding-left: 20%; max-width: 80%;}

    .pagediv-item > .flipup_ctnt {text-align: right; float: right;
        /*100% transform, to nudge the "menu" beyond the viewport*/
        transform: translate(100vh, 0%); transition: transform 0.6s cubic-bezier(0, 0, 0.745, 0);
    }

    .pagediv-item:hover > .flipup_ctnt {transform: translate(0vh, 0%); transition: transform 0.6s cubic-bezier(0.77, 0, 0.175, 1);}
</style>
```

Figure 27: Inline CSS definition within the <script> tags, used to aid in per-page styling overrides

CSS is heavily used in the styling of the web application. The design and development of this website's interfaces heavily relies on the class attribute in HTML, where CSS styling can be applied from there. "Main" styles are used in the *style.css* file at the root of the site, allowing for effortless reuse of predefined styles once the stylesheet is linked to a page. As CSS is "cascading" in nature, styles can be explicitly defined at the lower-level, allowing for style overrides whenever needed. The application, especially the homepage heavily uses pseudoselectors, such as *:hover*, *:nth-child()* and *:not* to achieve more complex effects, such as animating a glow on option selection, or pan option menus into view when hovered over.

5.0: Main section

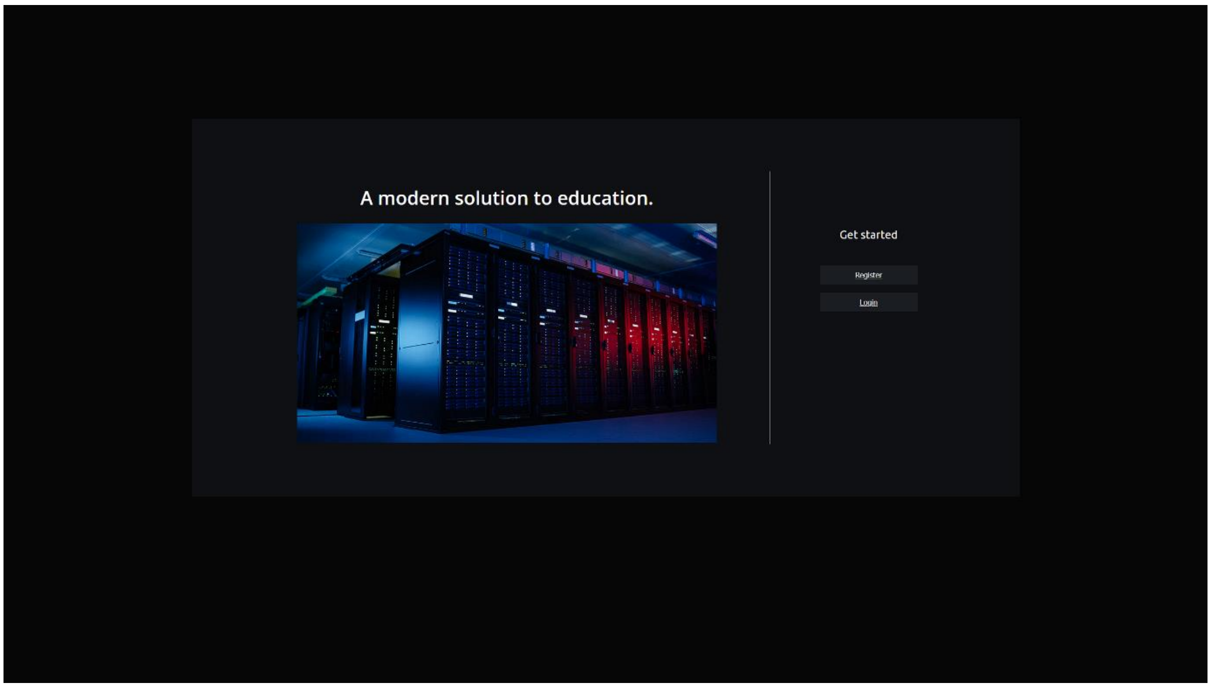
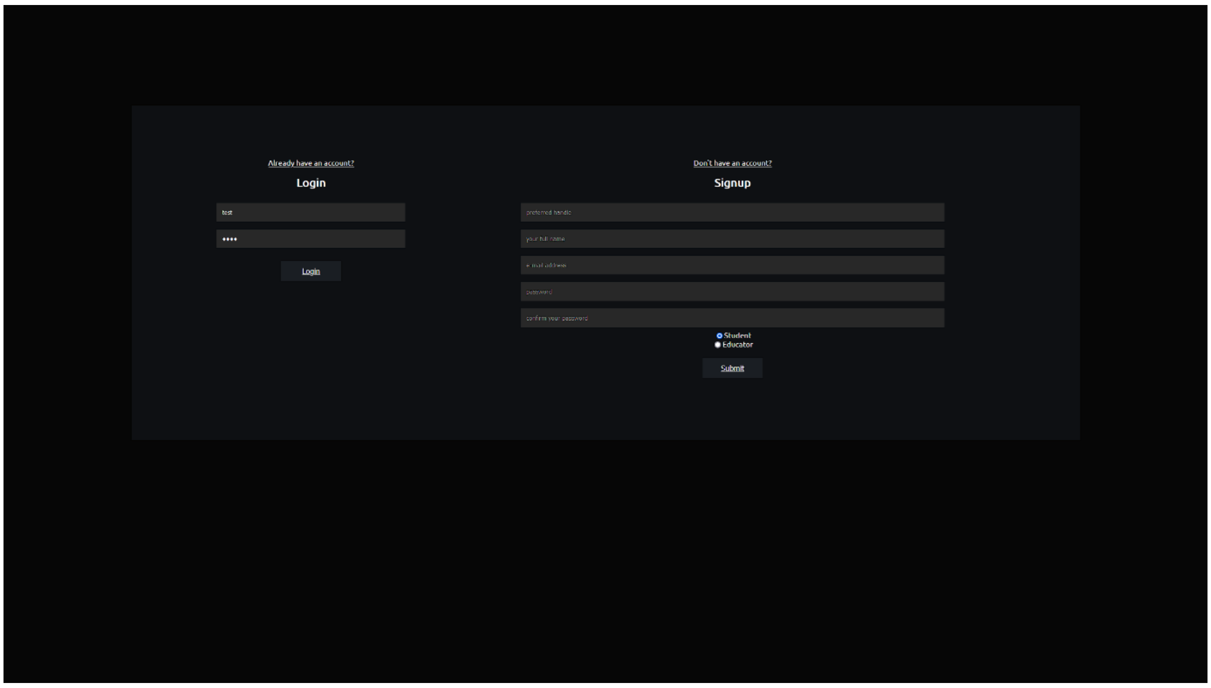


Figure 28: The visitor-facing introduction page



test
••••

preferred handle
your full name
e-mail address
password
confirm your password
[✓ Student]
Educator

Figure 29: The combined register and login page

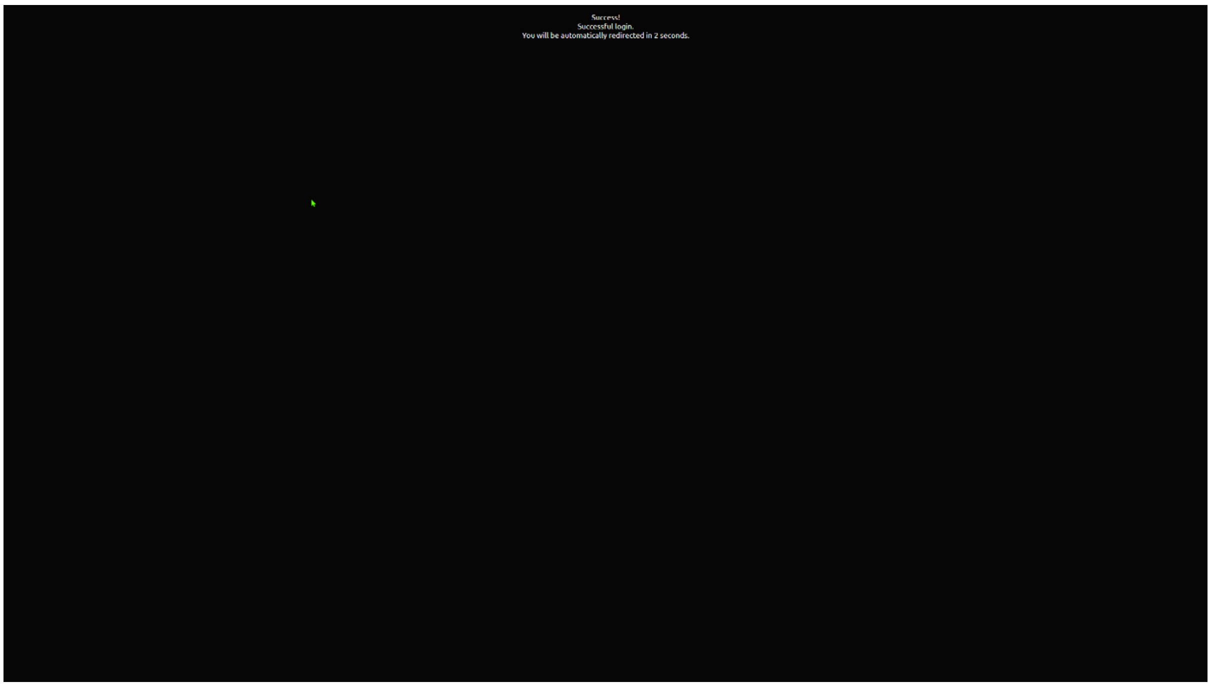

```
Success!
Successful login.
You will be automatically redirected in 2 seconds.
```

Figure 30: An example of holdingpage.php, in this case displaying confirmation of a successful login.

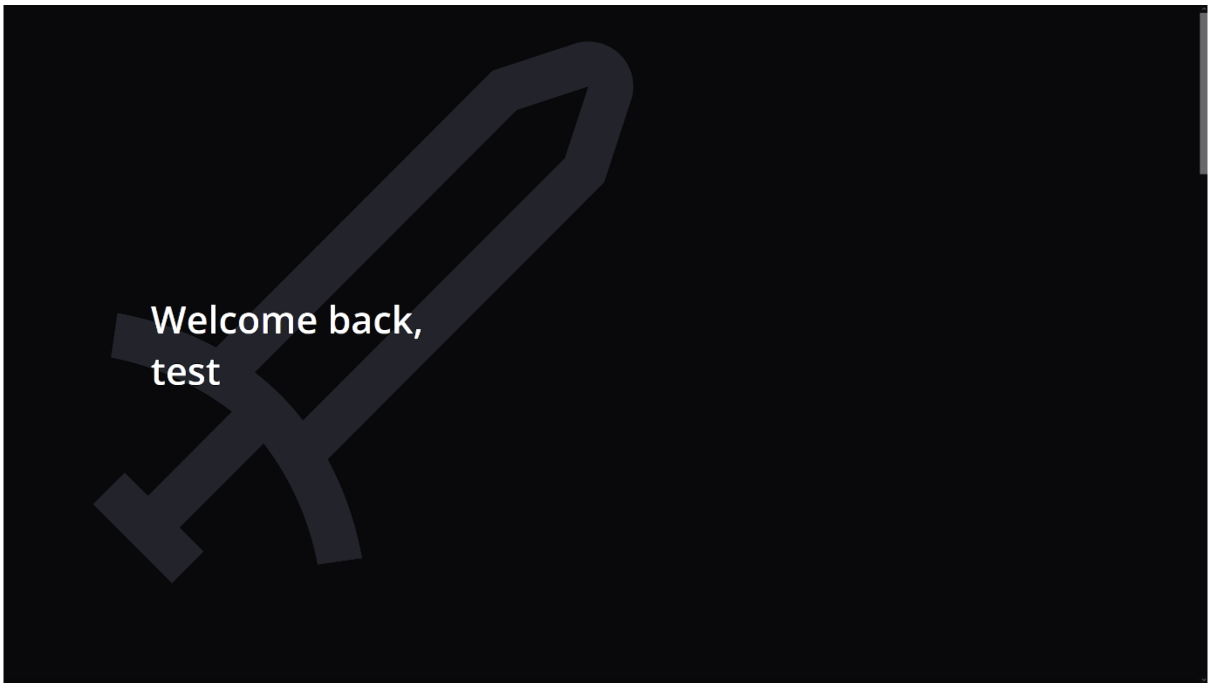


Figure 31: An example of homepage.php, displaying for administrator accounts.

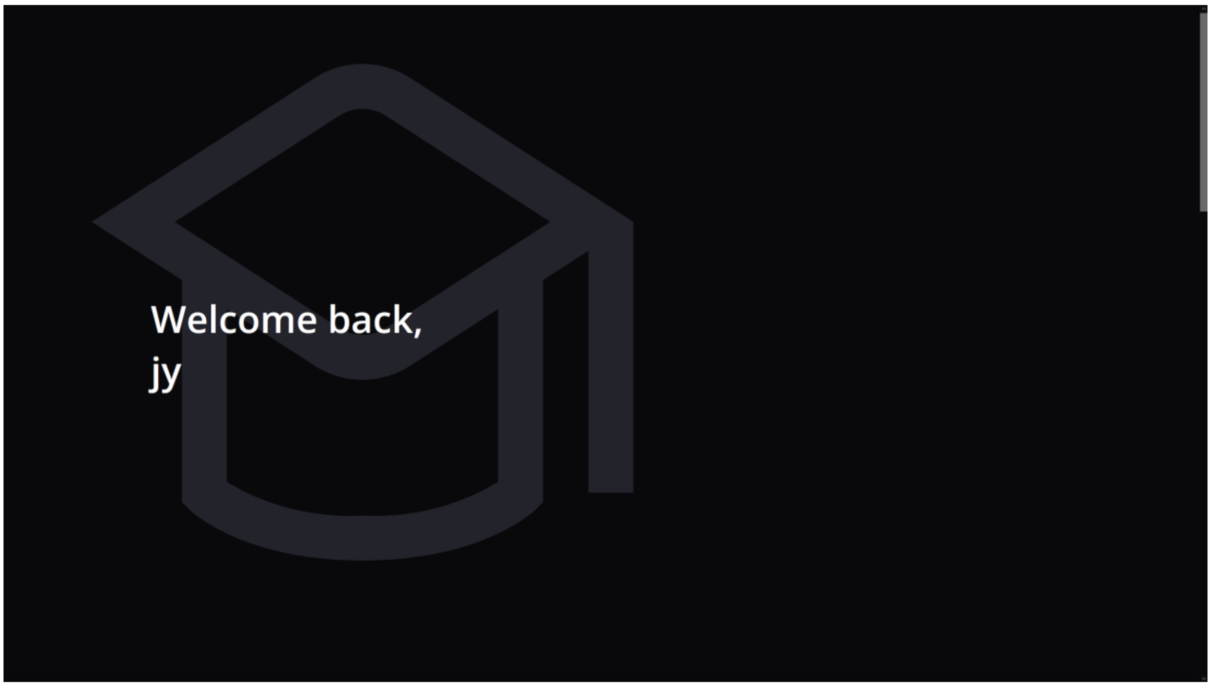


Figure 32: The same homepage.php, but displayed for student accounts.

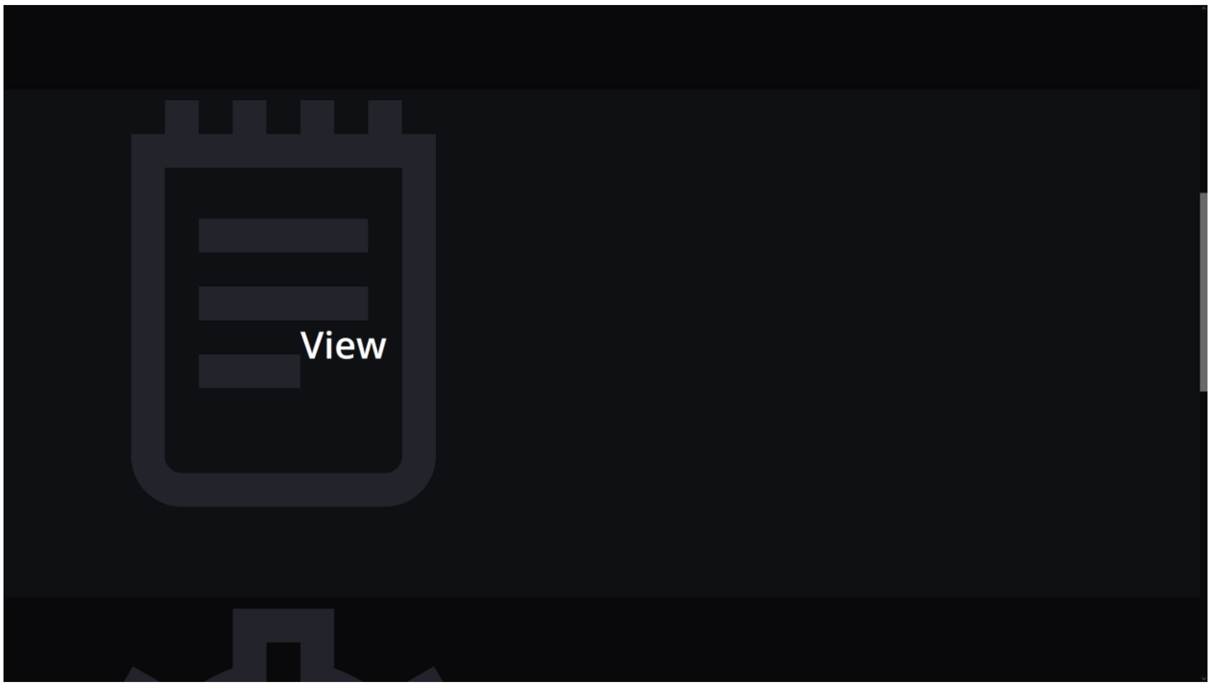


Figure 33: An example of a menu item in homepage.php, after scrolling down once from the welcome screen.

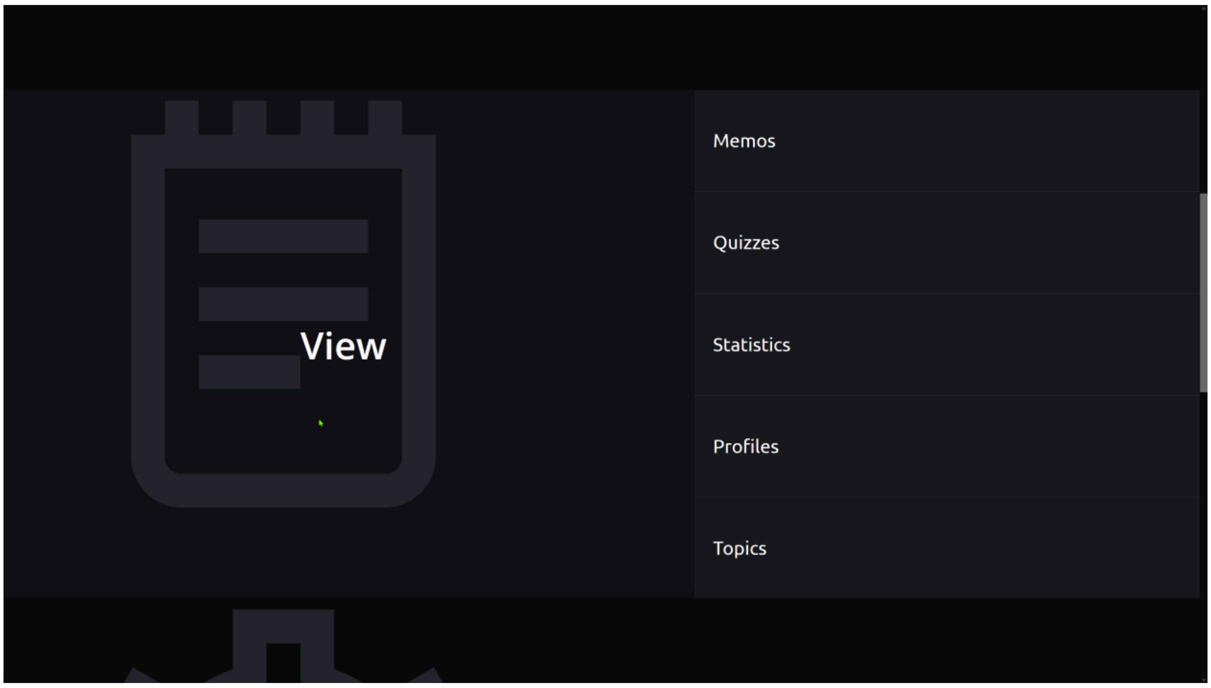
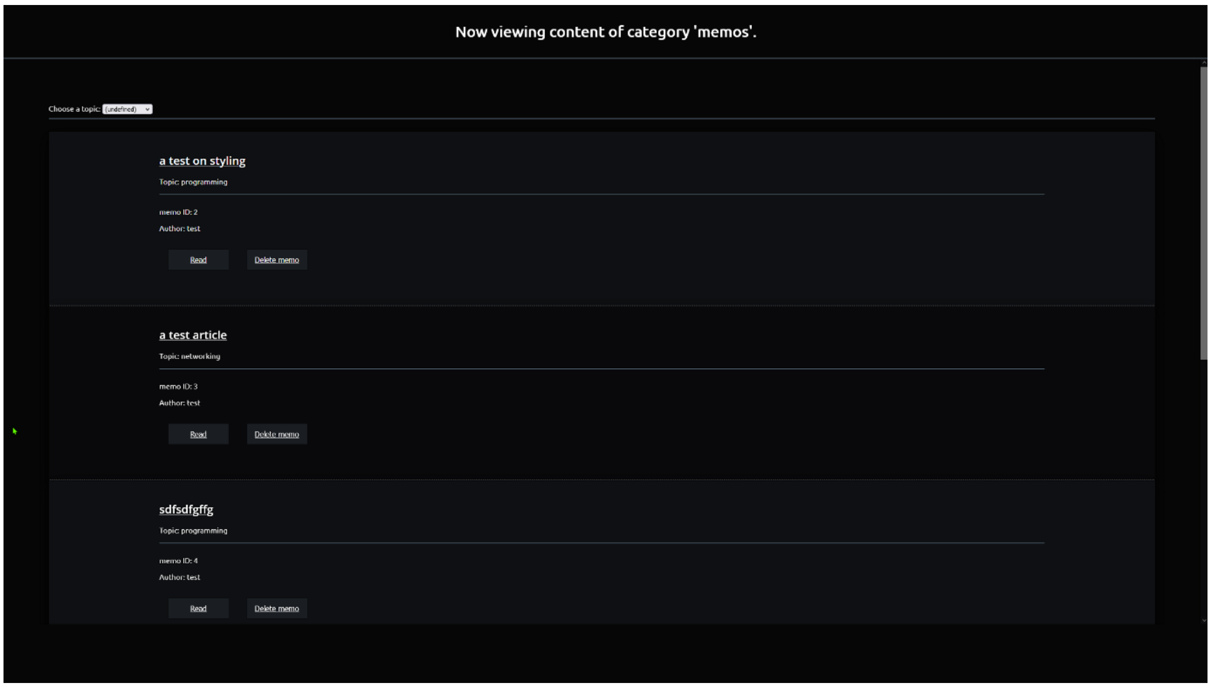


Figure 34: The same menu item as prior, but moused over.



Now viewing content of category 'memos'.

Choose a topic: (undefined)

a test on styling

Topic: programming

memo ID: 2

Author: test

a test article

Topic: networking

memo ID: 3

Author: test

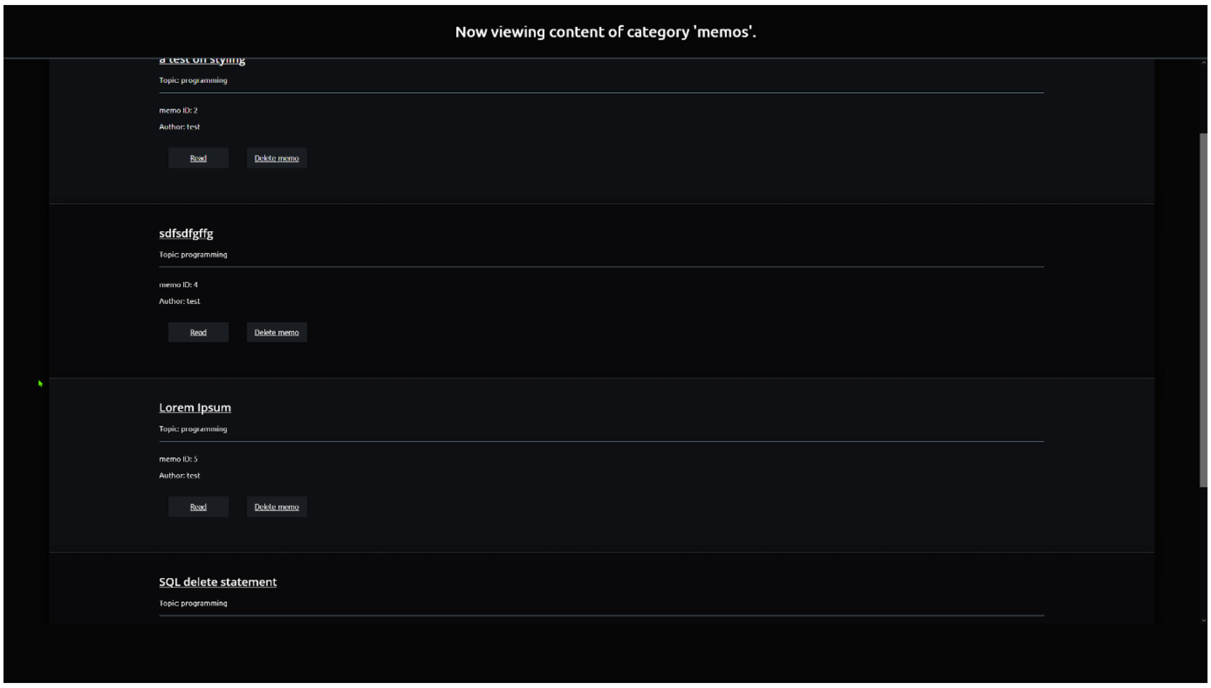
sdfsdfgffg

Topic: programming

memo ID: 4

Author: test

Figure 35: A view upon clicking the menu option 'memos' as an administrator. Note the filter option on the top-left of the screen.



Now viewing content of category 'memos'.

a test on styling

Topic: programming

memo ID: 2

Author: test

sdfsdfgffg

Topic: programming

memo ID: 4

Author: test

Lorem Ipsum

Topic: programming

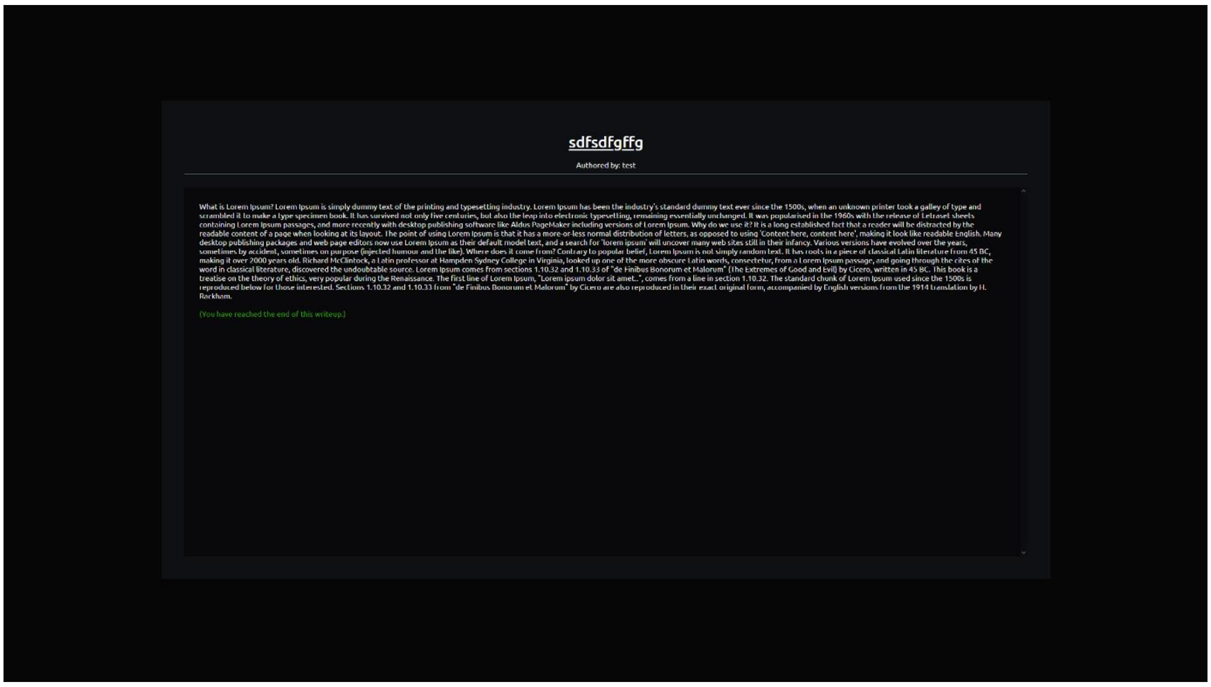
memo ID: 5

Author: test

SQL delete statement

Topic: programming

Figure 36: The same 'memos' view but filtered to only include memos with the 'programming' topic.



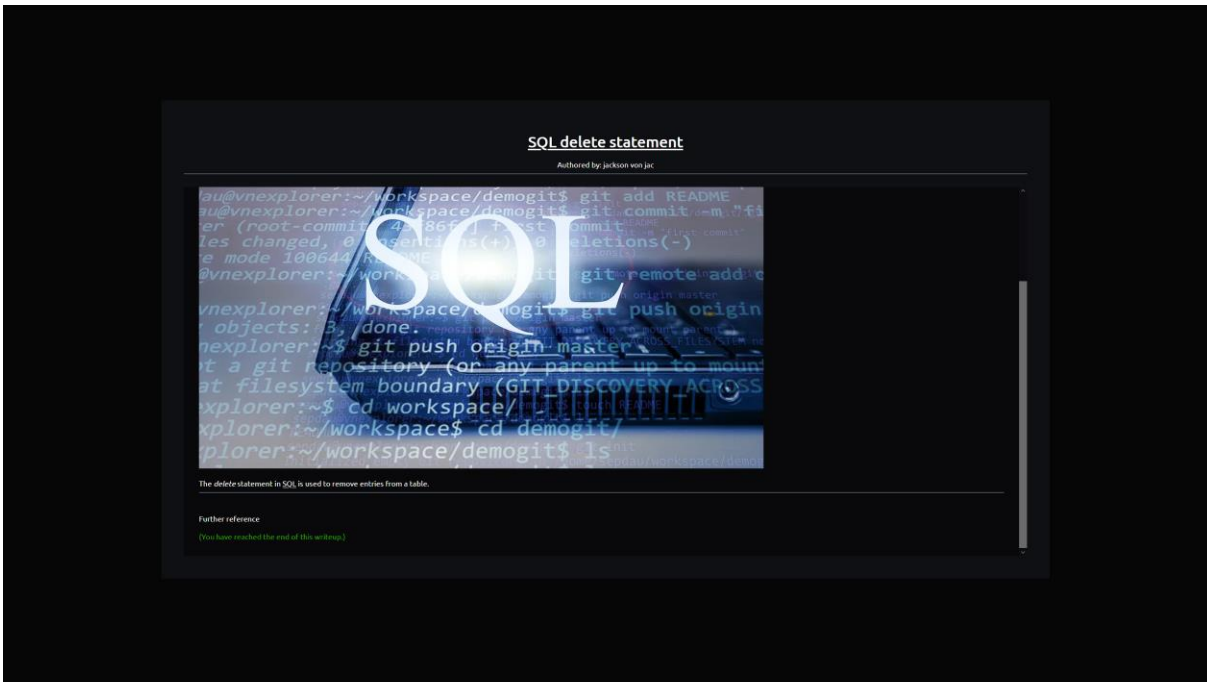
sdfsdfgffg

Authored by: test

What is Lorem Ipsum? Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum. Why do we use it? It is a long established fact that a reader will be distracted by the readable content of a page when looking at its layout. The point of using Lorem Ipsum is that it has a more-or-less normal distribution of letters, as opposed to using 'Content here, content here', making it look like readable English. Many desktop publishing packages and web page editors now use Lorem Ipsum as their default model text, and a search for 'lorem ipsum' will uncover many web sites still in their infancy. Various versions have evolved over the years, sometimes by accident, sometimes on purpose (injected humour and the like). Where does it come from? Contrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undoubtable source. Lorem Ipsum comes from sections 1.10.32 and 1.10.33 of "de Finibus Bonorum et Malorum" (The Extremes of Good and Evil) by Cicero, written in 45 BC. This book is a treatise on the theory of ethics, very popular during the Renaissance. The first line of Lorem Ipsum, "Lorem ipsum dolor sit amet..", comes from a line in section 1.10.32. The standard chunk of Lorem Ipsum used since the 1500s is reproduced below for those interested. Sections 1.10.32 and 1.10.33 from "de Finibus Bonorum et Malorum" by Cicero are also reproduced in their exact original form, accompanied by English versions from the 1914 translation by H. Rackham.

(You have reached the end of this writeup.)

Figure 37: The reader interface, displaying the full-text of a selected memo in the prior interface.



SQL delete statement

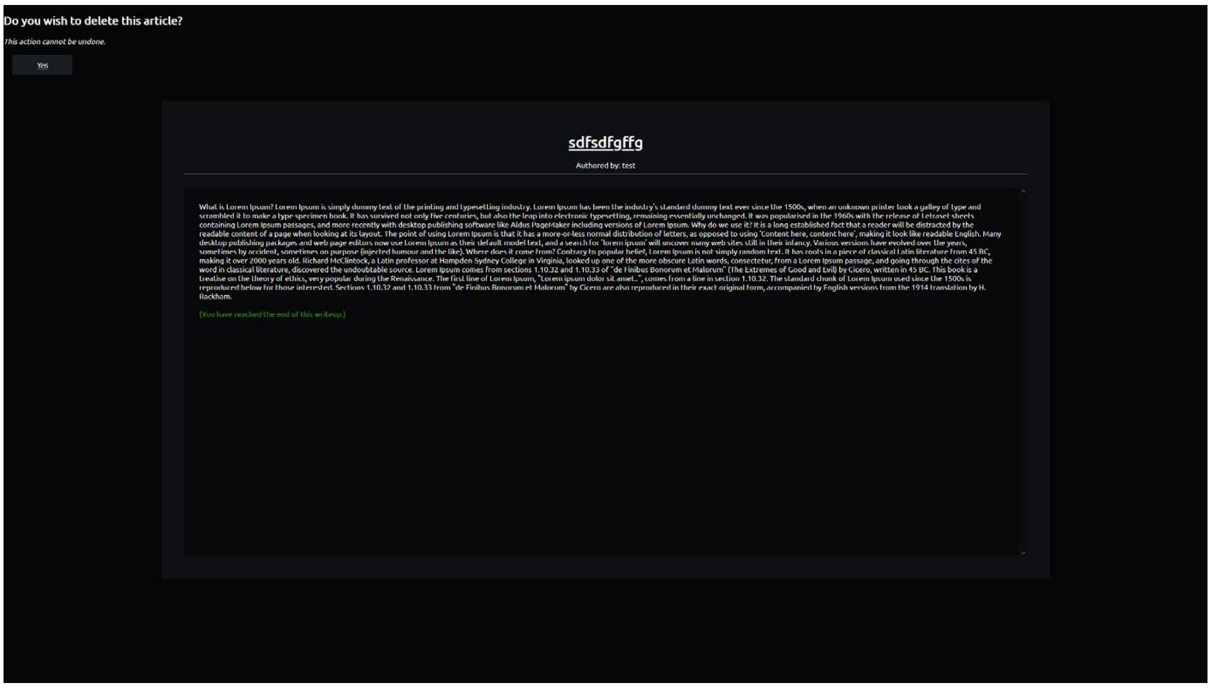
Authored by: jackson von jac

The *delete* statement in SQL is used to remove entries from a table.

Further reference

(You have reached the end of this writeup.)

Figure 38: The reader interface supports content formatting in the form of HTML tags and even <img> embeds.


Do you wish to delete this article?

This action cannot be undone.

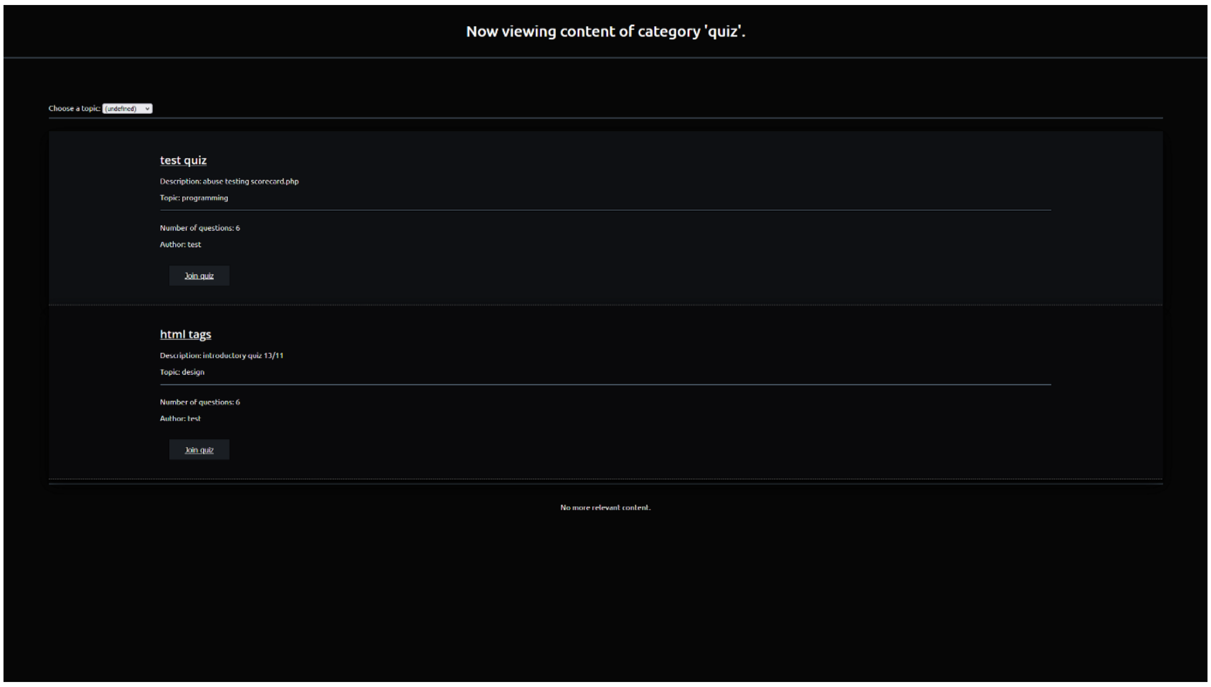
sdfsdfgffg

Authored by: test

What is Lorem Ipsum? Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum. Why do we use it? It is a long established fact that a reader will be distracted by the readable content of a page when looking at its layout. The point of using Lorem Ipsum is that it has a more-or-less normal distribution of letters, as opposed to using 'Content here, content here', making it look like readable English. Many desktop publishing packages and web page editors now use Lorem Ipsum as their default model text, and a search for 'lorem ipsum' will uncover many web sites still in their infancy. Various versions have evolved over the years, sometimes by accident, sometimes on purpose (injected humour and the like). Where does it come from? Contrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undoubtable source. Lorem Ipsum comes from sections 1.10.32 and 1.10.33 of "de Finibus Bonorum et Malorum" (The Extremes of Good and Evil) by Cicero, written in 45 BC. This book is a treatise on the theory of ethics, very popular during the Renaissance. The first line of Lorem Ipsum, "Lorem ipsum dolor sit amet..", comes from a line in section 1.10.32. The standard chunk of Lorem Ipsum used since the 1500s is reproduced below for those interested. Sections 1.10.32 and 1.10.33 from "de Finibus Bonorum et Malorum" by Cicero are also reproduced in their exact original form, accompanied by English versions from the 1914 translation by H. Rackham.

(You have reached the end of this writeup.)

Figure 39: The reader interface, with a floating delete confirmation if 'Delete memo' is clicked on the memo entry as an administrator.


Now viewing content of category 'quiz'.

Choose a topic: (undefined)

test quiz

Description: abuse testing scorecard.php

Topic: programming

Number of questions: 6

Author: test

html tags

Description: introductory quiz 13/11

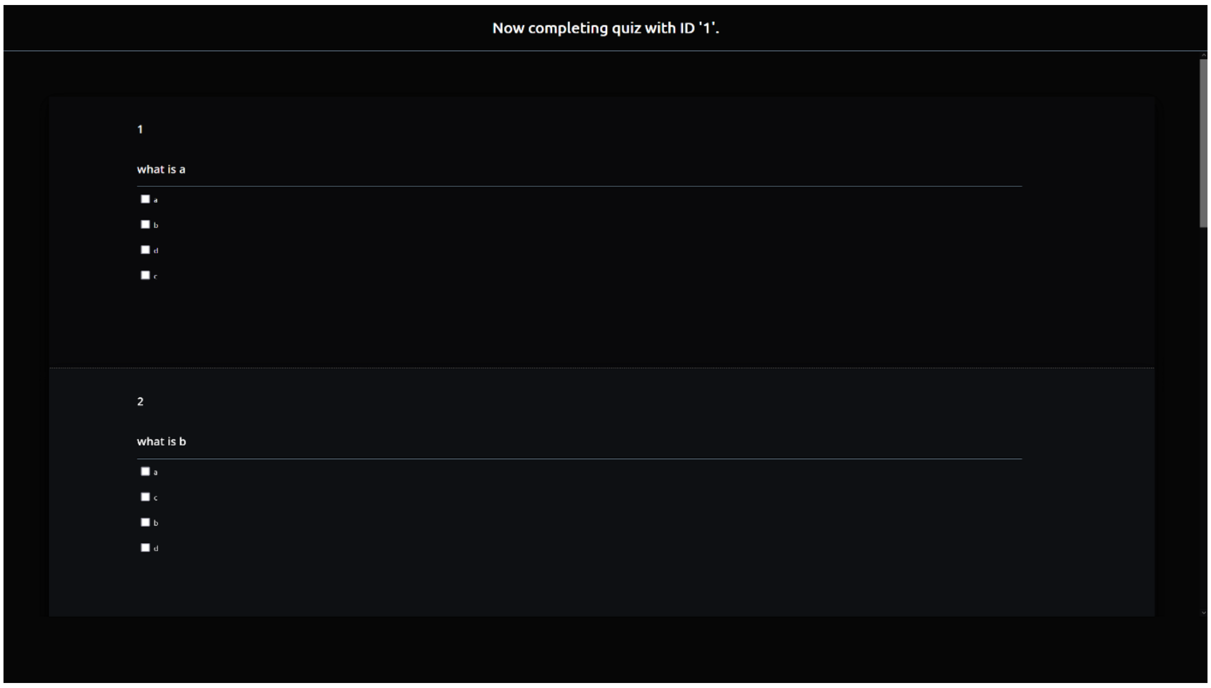
Topic: design

Number of questions: 6

Author: test

No more relevant content.

Figure 40: The view interface is reusable for other content types, such as quizzes.


Now completing quiz with ID '1'.

1

what is a

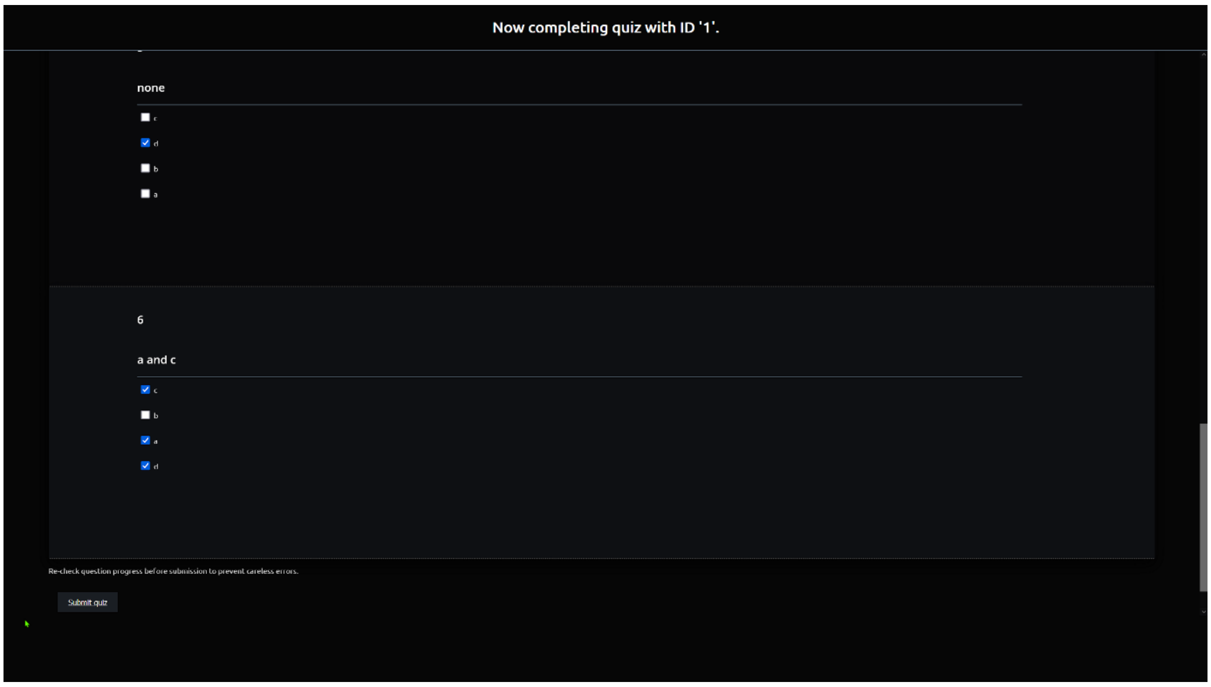
- [ ] a
- [ ] b
- [ ] d
- [ ] c

2

what is b

- [ ] a
- [ ] c
- [ ] b
- [ ] d

Figure 41: The quiz view if a quiz in the listing is joined.


Now completing quiz with ID '1'.

none

- [ ] c
- [✓] d
- [ ] b
- [ ] a

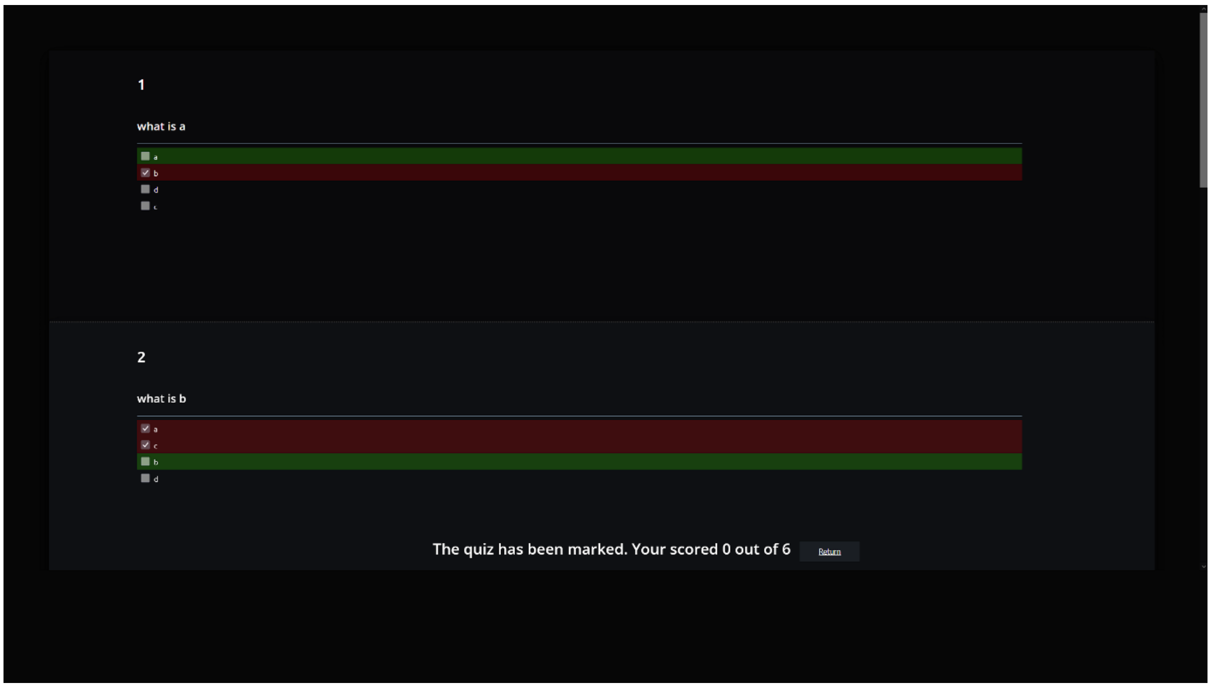
6

a and c

- [✓] c
- [ ] b
- [✓] a
- [✓] d

Re-check question progress before submission to prevent careless errors.

Figure 42: The option to submit the quiz is located at the bottom of the page.



1

what is a

- [ ] a
- [✓] b
- [ ] d
- [ ] c

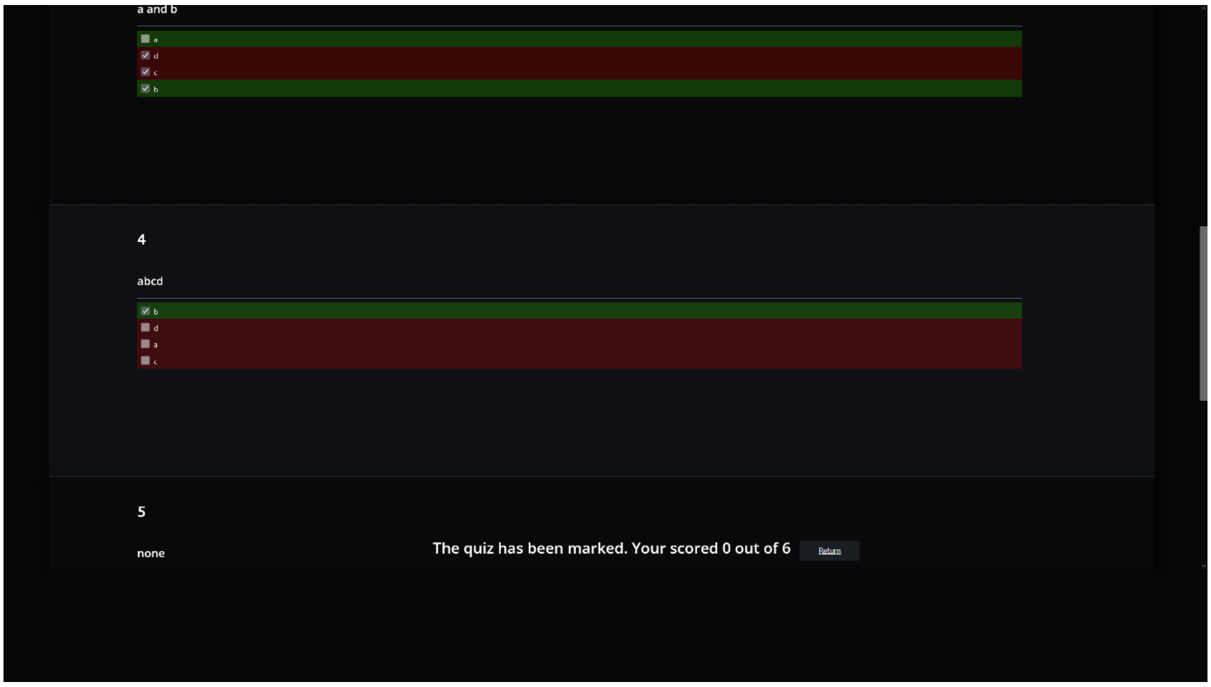
2

what is b

- [✓] a
- [✓] c
- [ ] b
- [ ] d

The quiz has been marked. Your scored 0 out of 6

Figure 43: The marking results for the prior quiz. All questions answered wrongly to demonstrate marking accuracy.



a and b

- [ ] a
- [✓] d
- [✓] c
- [✓] b

4

abcd

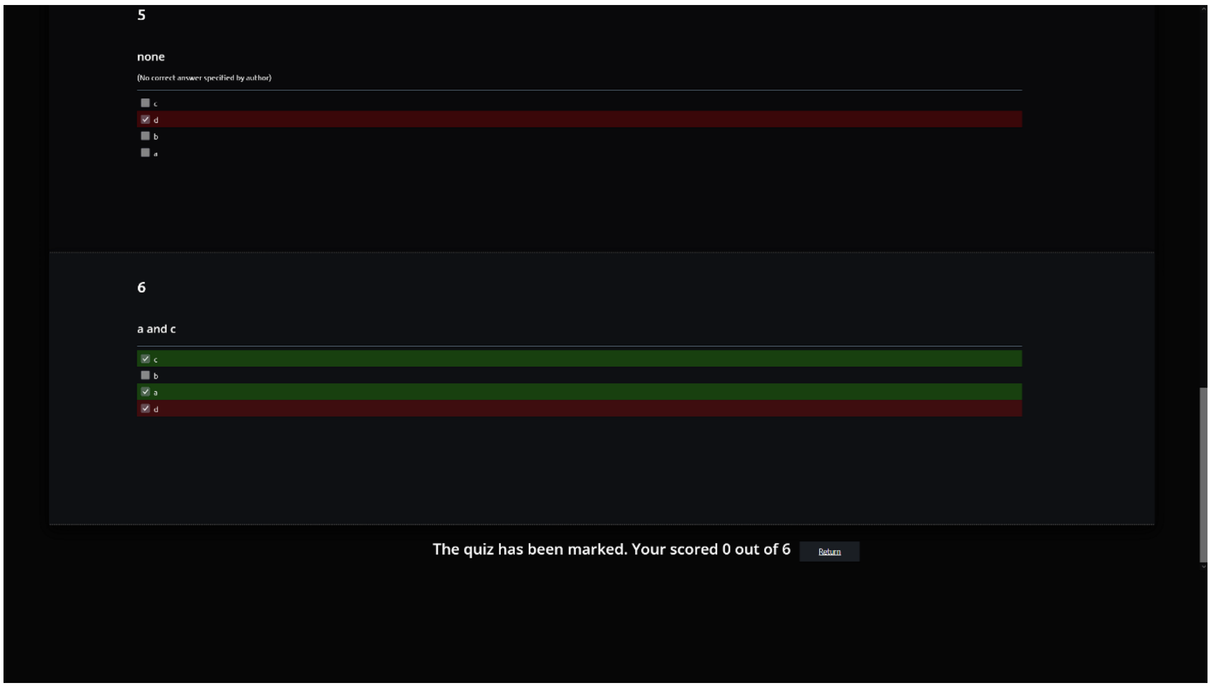
- [✓] b
- [ ] d
- [ ] a
- [ ] c

5

none

The quiz has been marked. Your scored 0 out of 6

Figure 44: Marking for prior quiz.


5

none

(No correct answer specified by author)

- c
- ✓ d
- b
- a

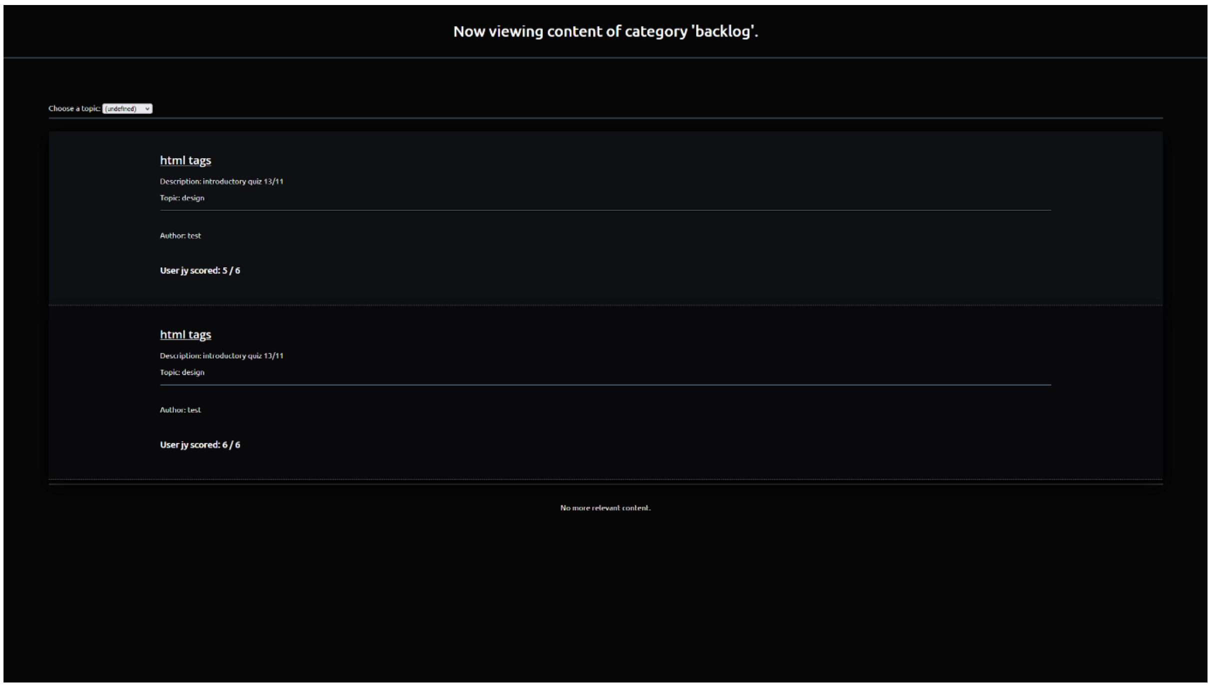
6

a and c

- ✓ c
- b
- ✓ a
- ✓ d

The quiz has been marked. Your scored 0 out of 6

Figure 45: Marking for prior quiz.


Now viewing content of category 'backlog'.

Choose a topic: (undefined)

html tags

Description: introductory quiz 13/11

Topic: design

Author: test

User jy scored: 5 / 6

html tags

Description: introductory quiz 13/11

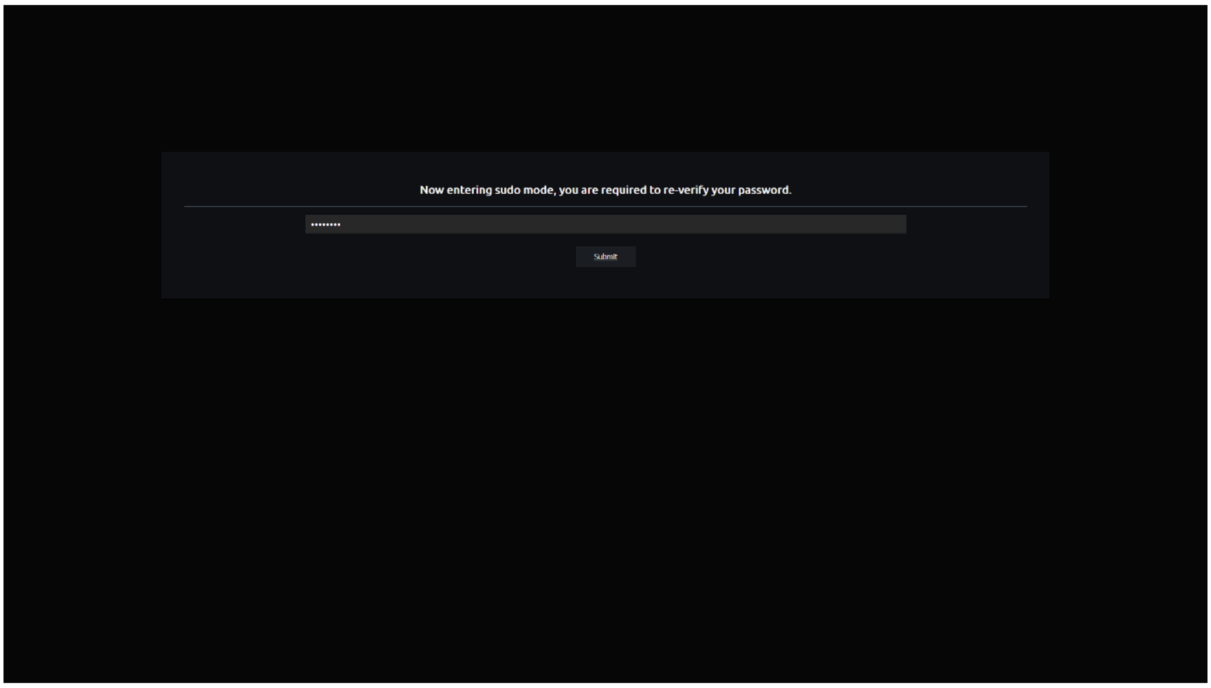
Topic: design

Author: test

User jy scored: 6 / 6

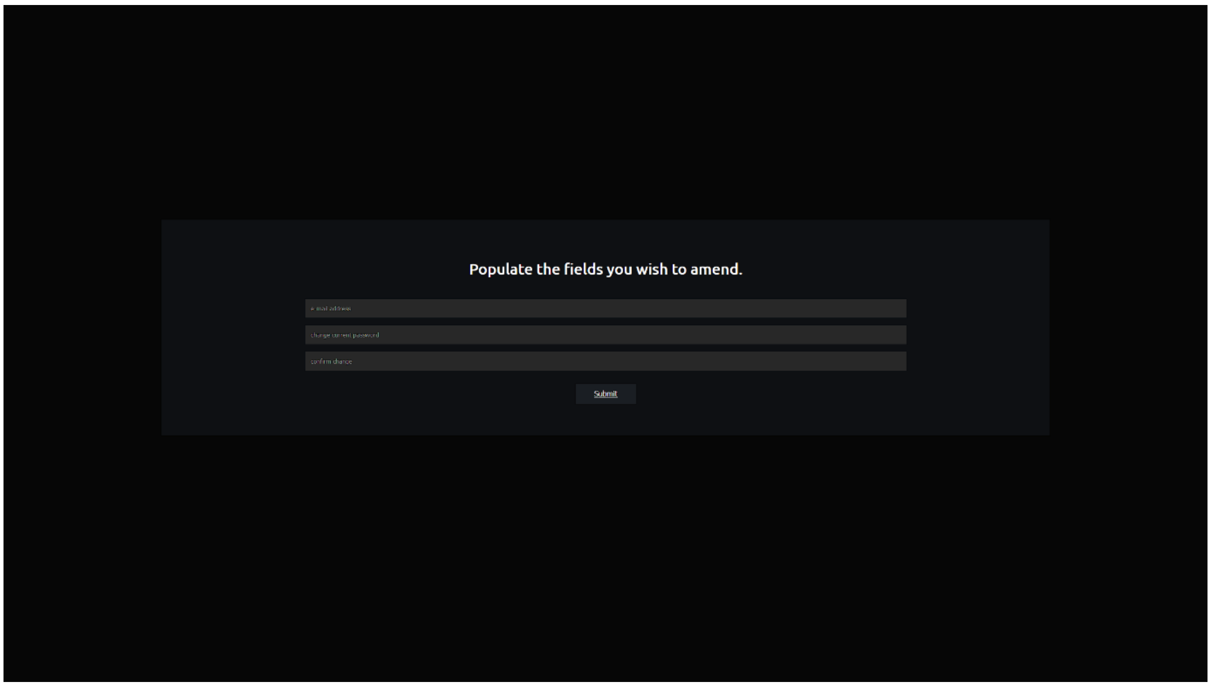
No more relevant content.

Figure 46: Students have the option to view their own marking history. Answering records are however unavailable.



Now entering sudo mode, you are required to re-verify your password.

Figure 47: A double-authentication page used when attempting to access settings, or when performing administrative acts.



Populate the fields you wish to amend.

Figure 48: The settings screen used by both student and teacher accounts, after an authentication round. Settings can be accessed by all users by navigating down the homepage.

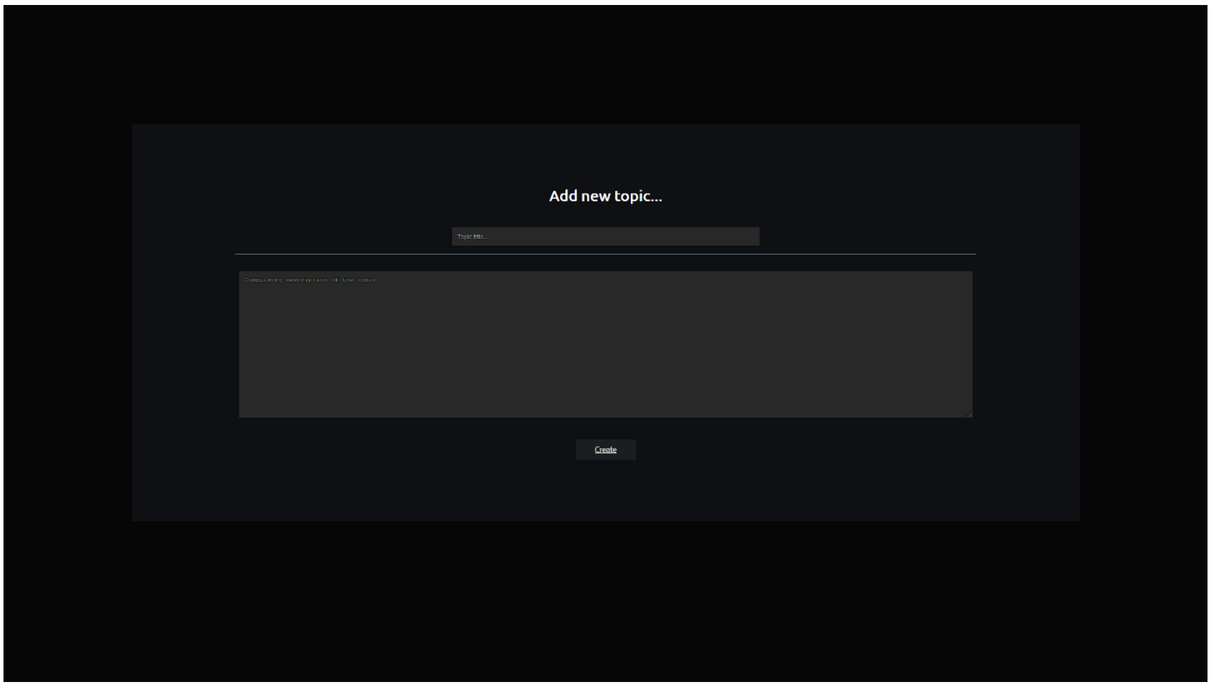


Figure 49: An admin-only interface used in adding new topics to be used by the system.

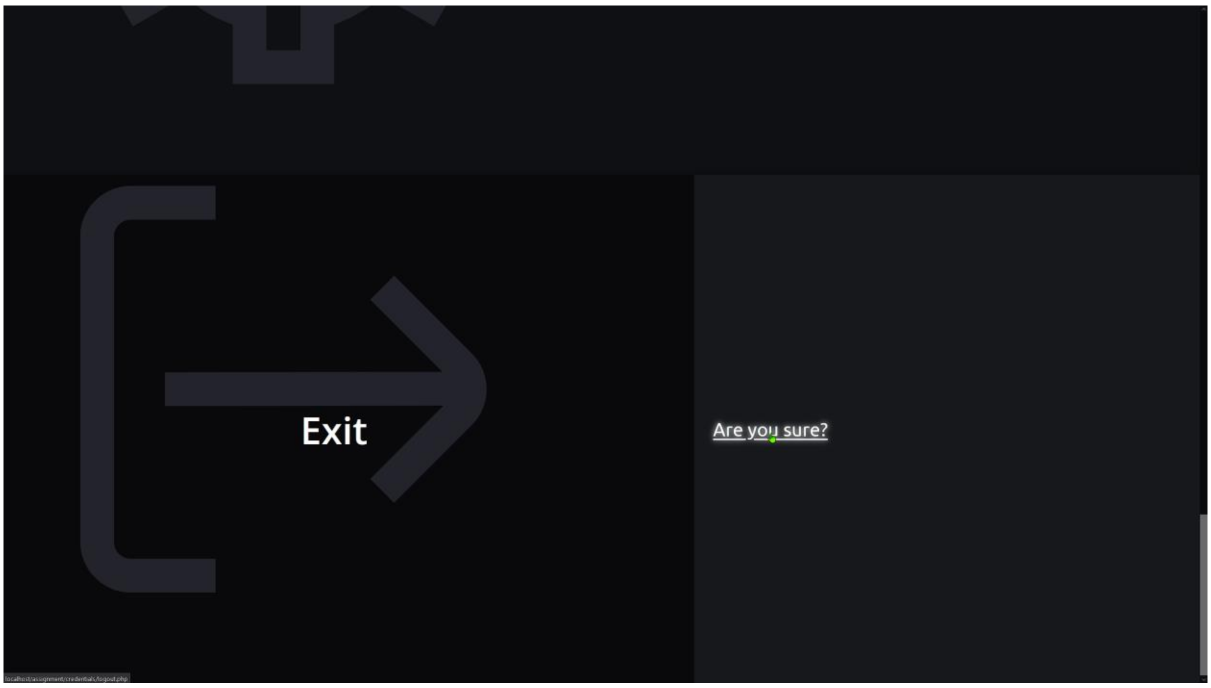


Figure 50: A log-out option is provided at the bottom of the homepage for all users. Click the pop-up to execute the log-out sequence.

6.0: References

The PHP Documentation Group (2021) *PHP: PHP Manual - Manual.* [Online]
Available from: https://www.php.net/manual/en/ [Accessed 24th October 2021]

W3Schools (2021) *CSS Pseudo-classes* [Online]
Available from: https://www.w3schools.com/css/css_pseudo_classes.asp [Accessed 24th October 2021]

Nicolas, Z. (2014) *PHP: How to use array_filter() to filter array keys?: StackOverflow.* [Online]
Available from: https://stackoverflow.com/a/22100877 [Accessed 2nd November 2021]

MDN contributors (2021) *CSS and JavaScript animation performance - Web Performance | MDN* [Online]
Available from:
https://developer.mozilla.org/en/US/docs/Web/Performance/CSS_JavaScript_animation_performance [Accessed 13th November 2021]