**Milestone 4 Beta Launch**

**and**

**Final Project Reviews**

*CEN 4010 Principles of Software Engineering, Spring 2021*

**Group 8:** Connection During Social Distancing| PostHut

**James Kos:** jkos2018@fau.edu, Product Owner, Team Lead, Backend Lead

**Robert Blanchette:** rblanchettej2017@fau.edu, Development Team

**John Marder:**  jmarder2018@fau.edu, Scrum Master, Backend Dev.

**Michael Norberto:** mnorberto2017@fau.edu, Dev. Team, Github Master, Front End Lead

**Kamal Shrouder:** kshrouder2018@fau.edu, Development Team

**Documentation Date:** 9 April 2021

**Revision History Table:**

-Originally Created: 10 April 2021, 7:13PM

**Product Summary (Max ¾ Page)**

**PostHut:**

PostHut is a platform that focuses on bringing people together. So many social media platforms now serve only to divide us further apart, but PostHut takes a unique approach to content creation. In particular, we have a themed post page, which allows users to come together and vote on a daily theme for content. Once a theme is selected, users are able to create posts that relate to it. This allows users to come together and focus on their commonalities rather than their differences, and share their ideas, experiences, and more in a safe space.

* Users should be able to create a unique account by filling in the required fields.
  + System will not create an account if one or more of the fields is not filled in
  + System will not create an account if one already exists with the same email
  + Once the fields are correctly and validly filled in, the user is redirected to the login page
* Users can login by filling in the login fields with their registration information
  + System does not login if the information is incorrect
* When creating a post the system will check if the user is a valid user and will then proceed to the post creation page
* The system will not allow a post if the user has more than 500 characters in the text
  + The system will not allow a post if the image attachment is larger than 8Mb
* When a user presses the Nice! button, the total number of Nices! For that post increases by one
* When voting for a specific theme, the user should be able to vote once for a specific theme
  + Once a user submits a vote, the system should increase the total number of votes for that theme by one

**Unique Features:** PostHut has the unique feature of a section for daily themed posts based on the user’s votes. This section allows users to come together to pick a theme for the day and then post related content. The goal of this feature is to help focus on commonalities between people and to bring people closer together.

**URL:** <https://lamp.cse.fau.edu/~cen4010_s21_g08/>

**Usability Test Plan (Max 2 Pages)**

Select ONE major function (NOT login or registration) to be tested for usability. We recommend search or upload/post.Write a usability test plan for this selected function. Please consult class material on developing usability test plan and questionnaire. This test plan is to contain:

1)Test objectives: 0.5 page

2)Test plan: System setup, starting point, task to be accomplished, who is the intended user, completion criteria, URL of the system to be tested. 3/4 page

3)Questionnaire form: 3 Lickert scale questions, in a form easy to be used by reviewer (check class slides). 3/4 page

Your test plan must be formatted to be easy to read and use by usability testers, including the questionnaire. You can also ask your friends or team members to do the usability test.

**Test Objective:** Determine whether the search function is working as intended. Users should be able to search any term and PostHut will return the posts that contain the search term and omit all the posts that do not contain the search term. The posts will be ordered from those containing the most occurrences of the search term down to the ones with the least occurrences.

**Test Plan:**  The system setup will be achieved when PostHut is live on the LAMP server and users can access it from any computing device with Internet connection. After this, our starting point will be when a user has created an account and is logged in. A user will be able to navigate from the homePage to the themePage, or popularPostPage, and back again. On each page the search field will be available on the top menu between the page navigation options to the left and the userPage link on the right. The task to be accomplished by the user will be to type a search term into the search field and click “Go!” or press enter in order to locate posts that relate to the topic described by the search term. The intended user of the test will be any user of the PostHut system and the test completion criteria will be a successful search of the PostHut system for posts that relate to the search term while the posts that do not relate. A user will see the result of the search in the posts that are displayed to the page whether that page be homePage, themePage, or popularPostPage. The posts will be ordered with those containing the most occurrences of the search term at the top of the page down to the ones with the fewest occurrences at the bottom. There will be no posts which do not contain the search term. If a search locates no posts containing the search term, PostHut will generate a message indicating that no posts contain the search term entered by the user. Testers can conduct the usability test by going to the PostHut location on the world wide web <https://lamp.cse.fau.edu/~cen4010_s21_g08/index.php> and creating an account then logging in, or logging in immediately if they already have an account.

**Questionnaire form:**

Please use the below questionnaire to indicate your Agreement with the PostHut features by placing an “X” in the appropriate box.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Strongly Disagree | Disagree | Neutral | Agree | Strongly Agree |
| The search field was easy to locate and enter text in |  |  |  |  |  |
| It was simple to create a post |  |  |  |  |  |
| The search feature returned posts relevant to my search term |  |  |  |  |  |

**QA Test Plan (Max 2.5 Pages)**

For the same function you chose for the usability test, write AND execute a QA test plan (check class slides)

a) Create formal QA test plan (consult QA class material). Basically, it has to contain:

1)Test objectives: max 0.5 pages

2)Hardware and software setup: max 0.5 page

3)Feature to be tested: max 0.5 page

4)Actual test cases: 3 test cases and results of testing them on your system: 1 page

You must provide test plan and test summary in the format (e.g. form) allowing easy reading and analysis by management e.g. in a table format like presented in the lecture. Suggested format for QA Table columns are: test #; test title; test description; test input; expected correct output; test results (PASS or FAIL for each tested browser)

5)Perform the testing as per plan above and record the results in a form above. 6)Apply the above test on 2 browsers of different type and record it in the above table

**Objective:** PostHut will have many posts implemented on each page, and over time it will be extremely difficult to find a specific post. Therefore, a solution to this is possessing a search function. With the search function, finding a specific post can be simple by searching for the post. Hence, the objective is to test the functionality of searching for a post.

**Hardware and Software Setup:**

**Hardware**: There will be two different setups for hardware. One will be using a desktop, while another will be on a cellular device. This is because PostHut needs to work sufficiently on various amounts of hardware, so splitting tests on a desktop and a mobile device can fully test the function.

**Software**: For software, it will test certain operating systems, specifically Windows, IOS, and Android. The following QA test will examine certain internet browsers, such as Google Chrome, Safari, and Microsoft Edge.

An individual will need to contain a stable internet connection to access PostHut, as well as using the search function.

**Feature to be Tested:**

The feature that will be testing will be the **search** feature. If a user searches for a post, the database will look at each item and see if the post was uploaded. It will then show a list of the previous posts that had the ‘keyword’ in the search. If the keyword is not stored in the database, it will mention that there are no posts with that specific keyword.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test #** | **Test Title** | **Test Description** | **Input** | **Output** | **Browser** | **Results** |
| 1 | Successful Search | A user will search for a post that is already on PostHut. Once the user searches for the input, the output will display posts that contain the input. | Music | Once “Go!” is pressed, nothing happens, and the URL extends to ”...?search=music”. | Microsoft Edge  Google Chrome  Safari | FAIL  FAIL  FAIL |
| 2 | Unsuccessful Search | A user will search for a post that is **not** on PostHut. Once the user searches for the input, the output will state that they’re no posts containing the input. | Bird | Once “Go!” is pressed, nothing happens, and the URL extends to ”...?search=Bird”. | Microsoft Edge  Google Chrome  Safari | FAIL  FAIL  FAIL |
| 3 | Uncompleted Search | A user will search for a post but won’t fill in the complete word. For instance, if there is a post named “example,” the user will search for “examp” and will find no results. | Mus | Once “Go!” is pressed, nothing happens, and the URL extends to ”...?search=Mus”. | Microsoft Edge  Google Chrome  Safari | FAIL  FAIL  FAIL |

**Code Review**

By now you should have chosen a coding style. In the report state what coding style you chose. Choose the code (substantial portion of it) related to the feature you used for QA and usability test. You need to submit an example of the code (or part of it –2 pages or so MAX) for its function to be peer reviewed, and document this as follows:

1)One team member should submit code to other team member(s) for peer review.

2)Peer review should be performed by other group member(s) (1 review is OK).

3)Peer review is to be done by e-mail and comments are to be included in the code

4)Submit listing containing the peer review and commented code and `

communication related to this in your Milestone 4 document

Important: It is critical that code reviews are friendly and helpful, intended to help and education, and not to criticize. It is strongly suggested that you use peer review in the development of the whole system.

**Register.php - Peer Review (Original PHP file heavily edited to meet the milestone requirement of 2 pages) - Uses a Procedural Style**

<?php //I noticed that in this file, everything is spaced out so much, which makes it hard see exactly what's happening and what's grouped together. In the future, I'd recommend not having blank lines unless they represent a clear shift in what the program is doing. For example, since each thing is commented here, I've removed the empty lines and only kept a line before each comment.

//I've also kept a large amount of space for my comments so you can more easily distinguish between them and the comments that were already there.

// Include config file

require\_once "config.php";

// Define variables and initialize with empty values

$email = $username = $password = "";

$email\_err = $username\_err = $password\_err = "";

// Processing form data when form is submitted

if($\_SERVER["REQUEST\_METHOD"] == "POST"){

if(empty(trim($\_POST["username"]))){

$username\_err = "Please enter a username";

} elseif(strlen(trim($\_POST["username"])) < 3){

$username\_err = "Username should be more than 3 characters";

} else{

$username = trim($\_POST["username"]);

}

//The way you commented this part could hurt the readability. When making comments about individual lines, it would be better to include a comment at the end of each line rather than in the previous line (and taking up even more space due to having far too many blank lines that I removed).

// Validate username

if(empty(trim($\_POST["email"]))){

$email\_err = "Please enter your Email Address.";

} else {

// Prepare a select statement

$sql = "SELECT id FROM Users WHERE email = ?";

if($stmt = mysqli\_prepare($link, $sql)){

// Bind variables to the prepared statement as parameters

mysqli\_stmt\_bind\_param($stmt, "s", $param\_email);

// Set parameters

$param\_email = trim($\_POST["email"]);

//The indentation here is inconsistent. It may still work, but if maintenance ever needs to be done, it could be a pain dealing with this.

// Attempt to execute the prepared statement

if(mysqli\_stmt\_execute($stmt)){

/\* store result \*/

mysqli\_stmt\_store\_result($stmt);

if(mysqli\_stmt\_num\_rows($stmt) == 1){

$email\_err = "This email is already in use";

}

else{

$email = trim($\_POST["email"]);

}

} else{

echo "Oops! Something went wrong. Please try again later.";

}

// Close statement

mysqli\_stmt\_close($stmt);

}

}

<!--Your php code was good; I couldn't find any optimizations or ways to improve it. However, the formatting made it difficult to read at times. The only syntactic change I would recommend is inverting the condition of the first if statement. You're checking that the method is POST and executing the php if it is; however, it would make more sense to check that it isn't POST and exit the php early in that case rather than having the entire php in an if statement.-->

<!--It is not recommended to use <style> for this much css at once. You should move it to its own file and simply link to it.-->

<!DOCTYPE html>

<!--It is not necessary to have separators for each css like this. Additionally, it shouldn't be necessary to include this many css files. I believe most of the things you need these includes for could be done with the bootstrap css you have included. If you wrote the files yourself, you should consider moving them to one file.-->

<head>

<title>PostHut Create Account</title>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<!--===============================================================================================-->

<link rel="stylesheet" type="text/css" href="vendor/bootstrap/css/bootstrap.min.css">

<!--===============================================================================================-->

<link rel="stylesheet" type="text/css" href="fonts/font-awesome-4.7.0/css/font-awesome.min.css">

<!--===============================================================================================-->

<link rel="stylesheet" type="text/css" href="fonts/iconic/css/material-design-iconic-font.min.css">

<!--===============================================================================================-->

<link rel="stylesheet" type="text/css" href="vendor/animate/animate.css">

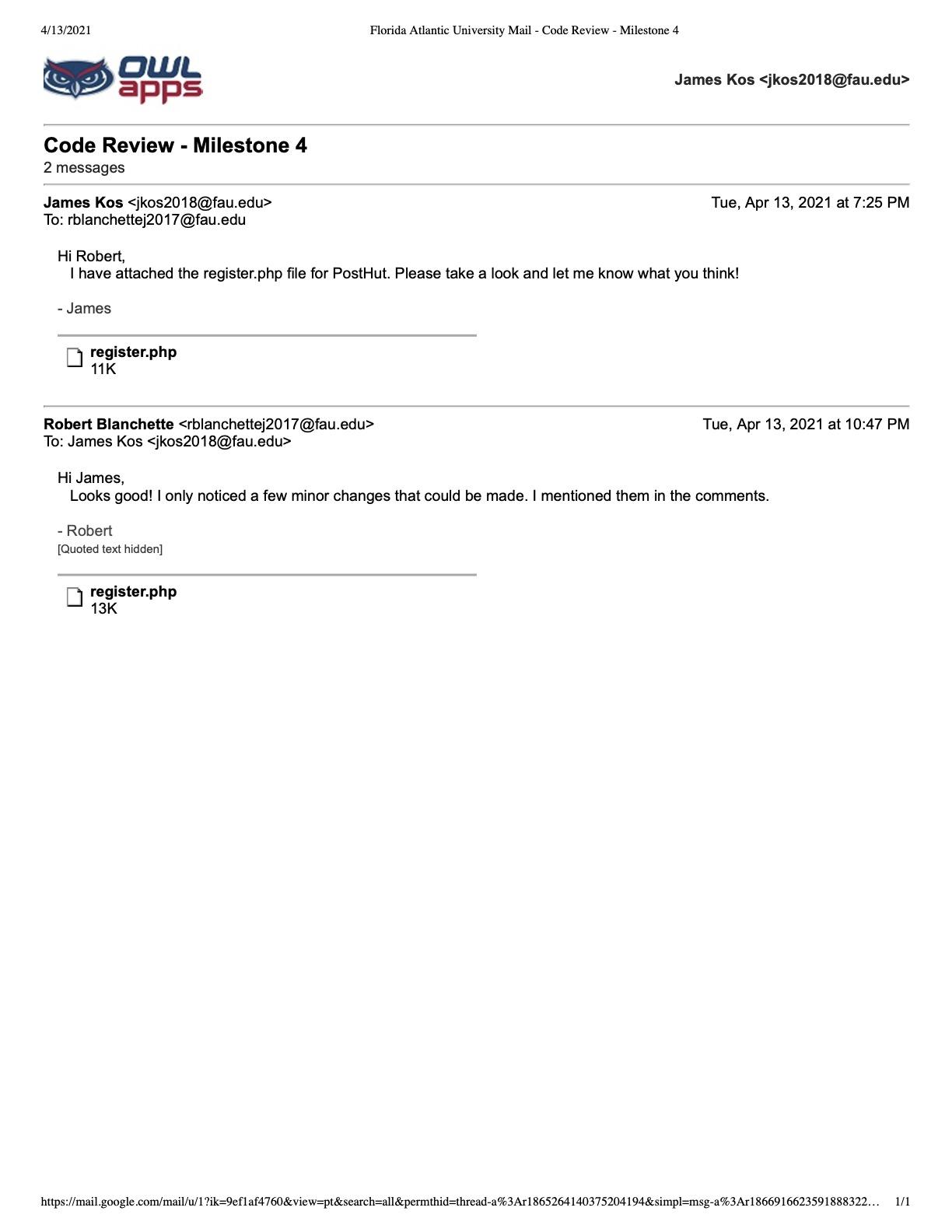
<!--===============================================================================================-->

<link rel="stylesheet" type="text/css" href="vendor/css-hamburgers/hamburgers.min.css">

<!--===============================================================================================-->

<!--Once again, it is not necessary to include many separators like this.-->

<!--Your html looks good as well aside from a few indentation inconsistencies. However, when testing it, I noticed that the way users are told their data is incorrect is inconsistent and not very user-friendly. No limitations for the username and password are specified at the start, but if you leave them blank, the form will only tell you the first one that needs to be fixed and the email will repeatedly say that the format is wrong until it's fixed. Only once you submit with something in all 3 boxes will the text for minimum username/password show up. Additionally, the wording for each is different and the warnings are in a different color and appear in the middle of the text areas, making it difficult for a user to tell what's going on (at first I thought the text boxes had actually been replaced with the error text) and generally not good-looking otherwise.-->

****

**Home.php - Peer Review - only using the portion of the code referencing the search functionality**

<?php

//I don't really have a comment for this section of the code

//but you can combine this php with the one above, and it would

//have a better flow

$query = mysqli\_query($link, "SELECT \* FROM Posts");

$resultPID = array();

$count = 0;

while($row = mysqli\_fetch\_array($query)) {

array\_push($resultPID, array($row['PostID']));

$count++;

}

?>

<script>

//This script here should be done with a link to a separate script file,

//doing it this way you can run into some issues

const SIZE = 256

let bmBcShift = new Array(SIZE); // Bad Char preprocessing table

var bmGsShift; //Good Suffix preprocessing table

function preBadChar (pattern, pSize, bcArr) {

var i;

for (i = 0; i < SIZE; i++) {

bcArr[i] = -1;

}

for (i = 0; i < pSize; i++) {

bcArr[pattern.charCodeAt(i)] = i;

}

}

//I do not really not much about the search algorithm you used,

//but this is an interesting implementation that you have, and

//I like the decomposition of functions, it makes it easier to

//read and follows what we learned

function suffixStrong (shift, bpos, pattern, size) {

let i = size;

let j = size + 1;

bpos[i] = j;

while (i > 0) {

while (j <= size && pattern[i-1] != pattern[j-1]) {

if (shift[j] == 0) {

shift[j] = j-i;

}

j = bpos[j];

}

i--;

j--;

bpos[i] = j;

}

}

function altSuffix (shift, bpos, pattern, size) {

var i;

var j;

j = bpos[0];

for (i = 0; i <= size; i++) {

if (shift[i] == 0) {

shift[i] = j;

}

if (i = j) {

j = bpos[j];

}

}

}

function preGoodSuffix(shift, pattern, size) {

var i;

let bpos = new Array(size +1);

for (i = 0; i < size + 1; i++) {

shift[i] = 0;

}

suffixStrong(shift, bpos, pattern, size);

altSuffix(shift, bpos, pattern, size);

}

function bmSearch () {

//variable declaration

//search pattern

let pattern = document.getElementById("search").value;

//I am wondering here, is the point of the window.alert to tell

//the user how many values were found, because that could use

//a bit better documentation

let aCount = <?php echo json\_encode($count); ?>;

//window.alert(aCount);

//loop through Post array to search the content of each post

//for the search pattern and record the number of times it is

//found within each post's content

//I see that for this function, you did relatively simple

//procedural code, and I like that, but it looks like

//your functions need more code, and obviously, for the actual

//product roll out, it should be that the functions are not commented out

let index = 0;

while (index < aCount) {

let output = <?php echo json\_encode($resultPID[0]); ?>;

//let outputRow = output[index];

let text = output;

let a = pattern.length;

let b = text.length;

//window.alert(text);

bmGsShift = new Array(a+1);

let c = 0;

var d;

let occur = 0;

//preprocessing

preBadChar(pattern, a, bmBcShift);

preGoodSuffix(bmGsShift, pattern, a);

//search

while (c <= (b-a)) {

d = a - 1;

while (d >= 0 && pattern[d] == text[c+d]) {

d--;

}

if (d < 0) {

occur++;

c += bmGsShift[0];

}

else {

c += Math.max(bmGsShift[d+1],d - bmBcShift[text.charCodeAt(c+d)]);

}

}

let result = "Found search text " + occur + " times";

//window.alert(result);

//output[index].push(occur);

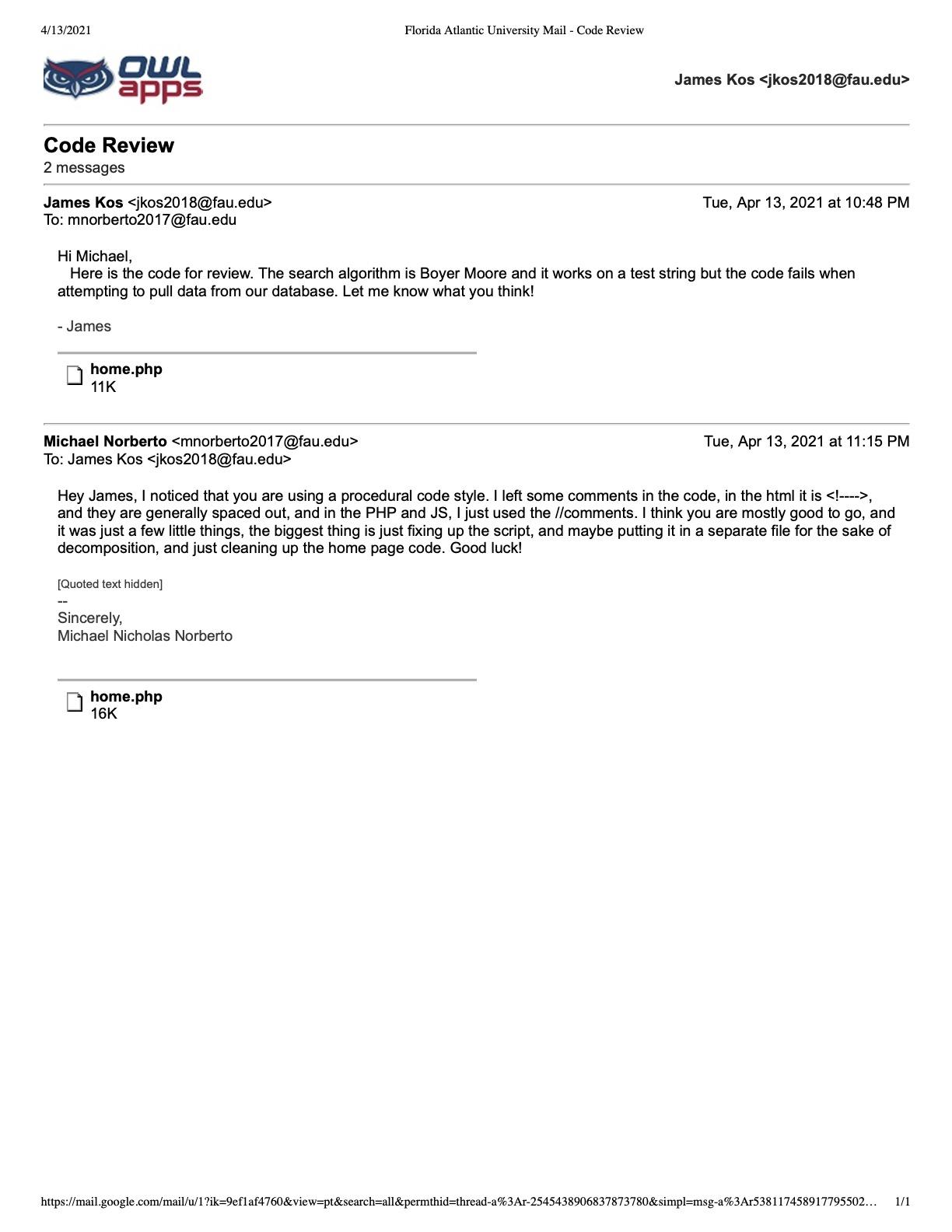
//window.alert(output);

index++;

}

}

</script>

****

**Self Check on Best Practices for Security (1/2 Pages)**

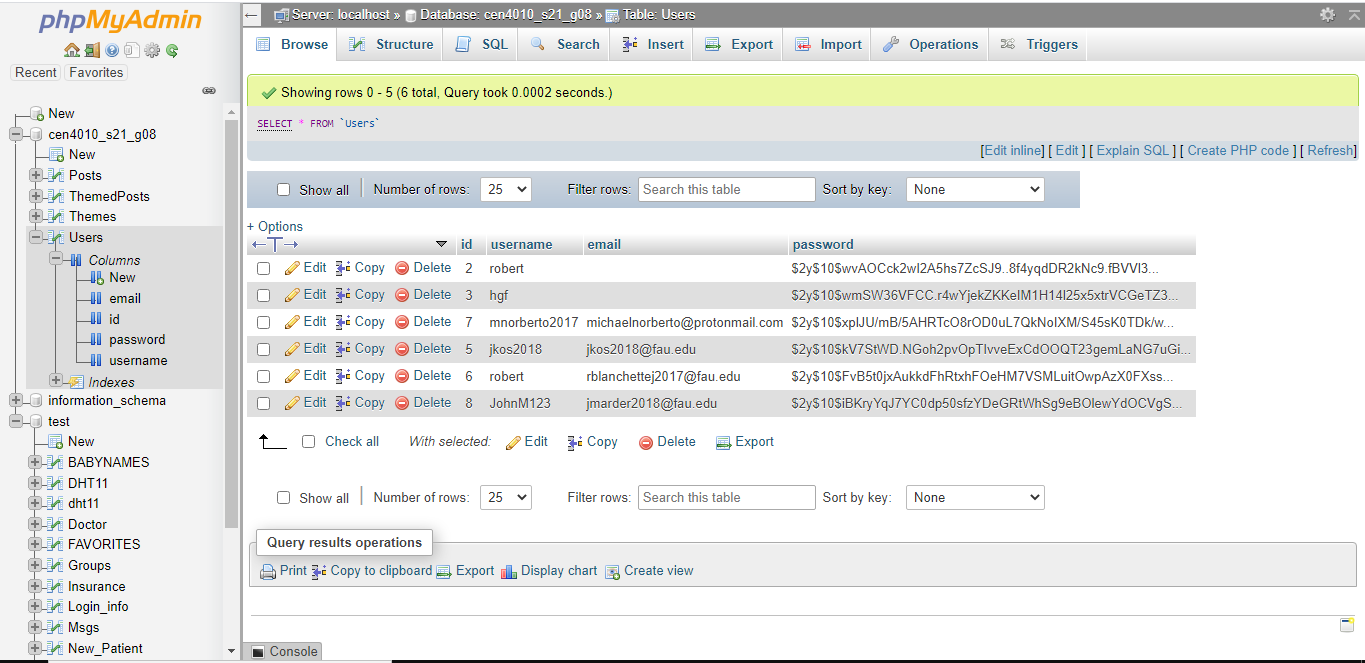
**1. List major assets you are protecting:**

PostHut does not have much information about the user, however, we are protecting:

* Email addresses
* Passwords
* Usernames
* Posts

We make it a priority to protect a user’s password by using a hash function encryption key. As you can see below, for each user, their password is encrypted using a unique hash function, with minimal collision, meaning that their passwords are safe. Usernames and emails are visible for administrators to see, but the user’s password is only known by them. Posts are controlled by the user, but other users have the right to report the post if it does not fit within our community guidelines. If such an action happens, our team will decide whether or not for the post to be taken down and to give a necessary punishment.

**2. Confirm that you encrypt password in the DB**



**3. Confirm Input data validation (list what is being validated and what code you used) –we request that you validate search bar input;**

**-Search Bar Validation code: This code checks that the only text entered in the search bar is either a-z or a hyphen. This is to ensure that there are no code injections, or other possible attacks. The code is listed below:**

jQuery(function(){

$("#btn-search").click(function(){

$(".error").hide();

var hasError = false;

var searchReg = /^[a-zA-Z0-9-]+$/;

var searchVal = $("#search-text").val();

if(searchVal == '') {

$("#search-text").after('<span class="error">Please enter a search term.</span>');

hasError = true;

} else if(!searchReg.test(searchVal)) {

$("#search-text").after('<span class="error">Enter valid text.</span>');

hasError = true;

}

if(hasError == true) {return false;}

});

});

**-Registration Form Validation: The validation for the login form checks that the username that is entered for registration is more than 3 letters, that the email follows a normal email format, and that the password is longer than 9 letters. The code is listed below:**

if($\_SERVER["REQUEST\_METHOD"] == "POST"){

if(empty(trim($\_POST["username"]))){

$username\_err = "Please enter a username";

} elseif(strlen(trim($\_POST["username"])) < 3){

$username\_err = "Username should be more than 3 characters";

}

else{

$username = trim($\_POST["username"]);

}

// Validate username

if(empty(trim($\_POST["email"]))){

$email\_err = "Please enter your Email Address.";

} else{

// Prepare a select statement

$sql = "SELECT id FROM Users WHERE email = ?";

if($stmt = mysqli\_prepare($link, $sql)){

// Bind variables to the prepared statement as parameters

mysqli\_stmt\_bind\_param($stmt, "s", $param\_email);

// Set parameters

$param\_email = trim($\_POST["email"]);

// Attempt to execute the prepared statement

if(mysqli\_stmt\_execute($stmt)){

/\* store result \*/

mysqli\_stmt\_store\_result($stmt);

if(mysqli\_stmt\_num\_rows($stmt) == 1){

$email\_err = "This email is already in use";

}

else{

$email = trim($\_POST["email"]);

}

} else{

echo "Oops! Something went wrong. Please try again later.";

}

// Close statement

mysqli\_stmt\_close($stmt);

}

}

// Validate password

if(empty(trim($\_POST["password"]))){

$password\_err = "Please enter a password.";

} elseif(strlen(trim($\_POST["password"])) < 9){

$password\_err = "Password must have at least 9 characters";

}

else{

$password = trim($\_POST["password"]);

}

// Check input errors before inserting in database

if(empty($email\_err) && empty($username\_err) && empty($password\_err)){

// Prepare an insert statement

$sql = "INSERT INTO Users (email, username, password) VALUES (?, ?, ?)";

if($stmt = mysqli\_prepare($link, $sql)){

// Bind variables to the prepared statement as parameters

mysqli\_stmt\_bind\_param($stmt, "sss", $param\_email, $param\_username, $param\_password);

// Set parameters

$param\_email = $email;

$param\_username = $username;

$param\_password = password\_hash($password, PASSWORD\_DEFAULT); // Creates a password hash

// Attempt to execute the prepared statement

if(mysqli\_stmt\_execute($stmt)){

// Redirect to login page

header("location: index.php");

} else{

echo "Something went wrong. Please try again later.";

}

// Close statement

mysqli\_stmt\_close($stmt);

}

}

// Close connection

mysqli\_close($link);

}

**-Login Form Validation: For the login form, the form checks that the username is in use, and checks that the account matches the username, it also has prevention against SQL injections. The code is listed below:**

// Define variables and initialize with empty values

$username = $password = "";

$username\_err = $password\_err = "";

// Processing form data when form is submitted

if($\_SERVER["REQUEST\_METHOD"] == "POST"){

// Check if username is empty

if(empty(trim($\_POST["username"]))){

$username\_err = "Please enter your username.";

} else{

$username = trim($\_POST["username"]);

}

// Check if password is empty

if(empty(trim($\_POST["password"]))){

$password\_err = "Please enter your password.";

} else{

$password = trim($\_POST["password"]);

}

// Validate credentials

if(empty($username\_err) && empty($password\_err)){

// Prepare a select statement

$sql = "SELECT id, username, password FROM Users WHERE username = ?";

if($stmt = mysqli\_prepare($link, $sql)){

// Bind variables to the prepared statement as parameters

mysqli\_stmt\_bind\_param($stmt, "s", $param\_username);

// Set parameters

$param\_username = $username;

// Attempt to execute the prepared statement

if(mysqli\_stmt\_execute($stmt)){

// Store result

mysqli\_stmt\_store\_result($stmt);

// Check if username exists, if yes then verify password

if(mysqli\_stmt\_num\_rows($stmt) == 1){

// Bind result variables

mysqli\_stmt\_bind\_result($stmt, $id, $username, $hashed\_password);

if(mysqli\_stmt\_fetch($stmt)){

if(password\_verify($password, $hashed\_password)){

// Password is correct, so start a new session

session\_start();

// Store data in session variables

$\_SESSION["loggedin"] = true;

$\_SESSION["id"] = $id;

$\_SESSION["username"] = $username;

// Redirect user to welcome page

header("location: https://lamp.cse.fau.edu/~cen4010\_s21\_g08/home.php");

} else{

// Display an error message if password is not valid

$password\_err = "The password you entered was not valid.";

}

}

} else{

// Display an error message if username doesn't exist

$username\_err = "No account found with that username.";

}

} else{

echo "Oops! Something went wrong. Please try again later.";

}

// Close statement

mysqli\_stmt\_close($stmt);

}

}

// Close connection

mysqli\_close($link);

}

?>

**6. Self-Check Adherence to Original Non-Functional Specifications**

Copy all original non-functional specs as in high level application document published at the very beginning of the class and then for each say DONE if it is done (which is expected and required); ON TRACK if it is in the process of being done and you are sure it will be completed on time; or ISSUE meaning you have some problems and then explain it.

Note: you must adhere to all original non-functional specs as published in the original high-level specification document. Failure to do so may cause reduced grade

1. **DONE** Response Time: Users with a decent internet connection should not take longer than 2 seconds to load a page or 5 seconds to submit a new post. The computational complexity of sorting algorithms for posts should not exceed O(nlogn).
2. **DONE** Usability: Since our application is honing on the prospects of being simple to manage and use, the usability of PostHut will seem as simple as hitting simple inputs.
3. **DONE** Accessibility: The PostHut application will be available to access on all mobile devices as well as desktops and laptops. Mobile device accessibility is available for but not limited to Apple, Samsung, Google, and LG. Because this application is done through interacting with others, a stable internet connection is going to be needed to get as much out of the app as possible. No internet connection is unfortunately something we can’t work around.
4. **DONE** Expected Load: fewer than 100 users.
5. **DONE** Security Requirements: Passwords should be at least eight characters long and encrypted and stored somewhere inaccessible to users. All inputs should be checked to avoid SQL injection.
6. **DONE** Storage: Passwords are encrypted and stored in a secret location. Posts, user information, and everything else should be stored in an SQL server.
7. **DONE** Availability: PostHut should be available through browser access everywhere where an internet connection is available. It may also be accessed on mobile devices with internet connections through the app. Updates to the interface should not be noticeable as the browser could update while switching pages and the mobile app should be updated automatically by the device’s store app while not in use. Other updates should take place at midnight EST and not take longer than 5 seconds. They should also not cause any post currently being typed to be reset or unable to post; if an update requires such to happen, it must be announced at least three days before with a scheduled downtime period unless it is a critical security update; in that case, a notification with an apology must be displayed to users the next time they log in if it is within three days of the incident.
8. **In Progress (can’t fully test)** Fault Tolerance: No more than one post per 1,000 should fail to send due to errors on our end.

**Youtube Demo Link**

<https://www.youtube.com/watch?v=mkBbnZMpgwA&t=7s>