Florida Atlantic University

CEN 4010 Principles of Software Engineering, Fall 2021

Milestone 4 Beta Launch and Final Project Reviews

Strikers

Group: 20

Project Members

Nicholas Giacobbe (Team Lead)

ngiacobbe2020@fau.edu

Truong Nguyen

truongnguyen2019@fau.edu

Shay Reardon

sreardon2018@fau.edu

Alex Rodriguez

alexrodrigue2018@fau.edu

Luiza Trebesqui

ltrebesqui2018@fau.edu

Revision History

Version #	Date of Revision	Changes
1	09/28/2021	Initial Version
2	10/25/2021	Updated for Milestone 3
3	11/15/2021	Updated for Milestone 4

Product Summary

Website URL: https://lamp.cse.fau.edu/~cen4010_fa21_g20/strikerBeta/

Striker:

Striker is a platform used to view news articles. The beta launch focuses on posting and viewing articles. The user will be able to choose if they want to be an author or reader. In the launch version, we will complete the payment infrastructure.

Create Articles:

By moving to the profile portion of the system, a user can create an article. Clicking the start button will move the user to a webpage that will allow for the insertion of a title and a body. Once the post button has been clicked, the article will be posted and viewable on the main page.

Edit and Delete Articles:

By moving to the profile portion of the system, a user can edit or delete articles created. By clicking the edit button located next to the article, the user will be able to change the information previously inputted. Likewise, by clicking the delete button, the user will be able to delete the article posted.

Viewable Articles:

On the Main Page, a list of articles that have been created can be found. Once the article has been clicked on, the user will be able to view the article's contents.

Search:

By inputting words in the search bar at the top of the main page and clicking search, articles sharing this keyword will be displayed.

Login/Signup:

By clicking the login button located at the top right of the page, the user will be redirected to the login page. If the user already has an account, they will be able to login using the set-up username and password. If the user has not set up an account, they may press the signup button located at the bottom of the page.

Our goals for the next release are to implement payments and integrate google firebase to make the sign-in easy. We also intend to differentiate between writers and readers to give writing privileges to only writers. One significant change we made is how purchasing articles work. Due to the fees stripe charges, the only way to economically do the project is to buy at least two articles at a time, so we are implementing a token system. The paywall system still has not been implemented.

Usability Test Plan

Test Objectives

The main test objectives for the usability test for Striker is to test if the post function works correctly and if our interface is easy to navigate and use.

The post function within Striker is the backbone of the system. The purpose of our product revolves around if this system is functional and easy to use for the user. If the system does not display the correct information, then the system itself has no purpose. If the system is unable to display the information to all users, then the system does not function properly. The test to check if the post function works properly would cover these bases, ensuring that the system can display information accurately to all users.

If the system's interface is easy to navigate and use, this site's purpose is to inform users. The site cannot be too cluttered or complex; instead, menus and options should be easy to find not to frustrate users. Testing Striker's interface means ensuring that it is easy on the eyes and an overall pleasant experience for the user.

Test Plan

Scope:

The scope of the test plan is to test the functionality of the posts function using two significant browsers. The browsers that will be used are Google Chrome, Mozilla Firefox, and Apple Safari.

Purpose:

The purpose of this test plan is to check the functionality of the post's function. These tests will include the basic posting function, checking whether the post is persistent on each user's main landing page, and checking spam protection.

Equipment:

In order to complete these tests, personal computers will be used. Each member will test their portions on their machines which a single device will later test. Each test will be done on Google Chrome, Mozilla Firefox, and Apple Safari.

Scenarios:

The scenarios used to check the functionality would be filling in different information into the articles to check for bugs and check for a user-friendly site environment.

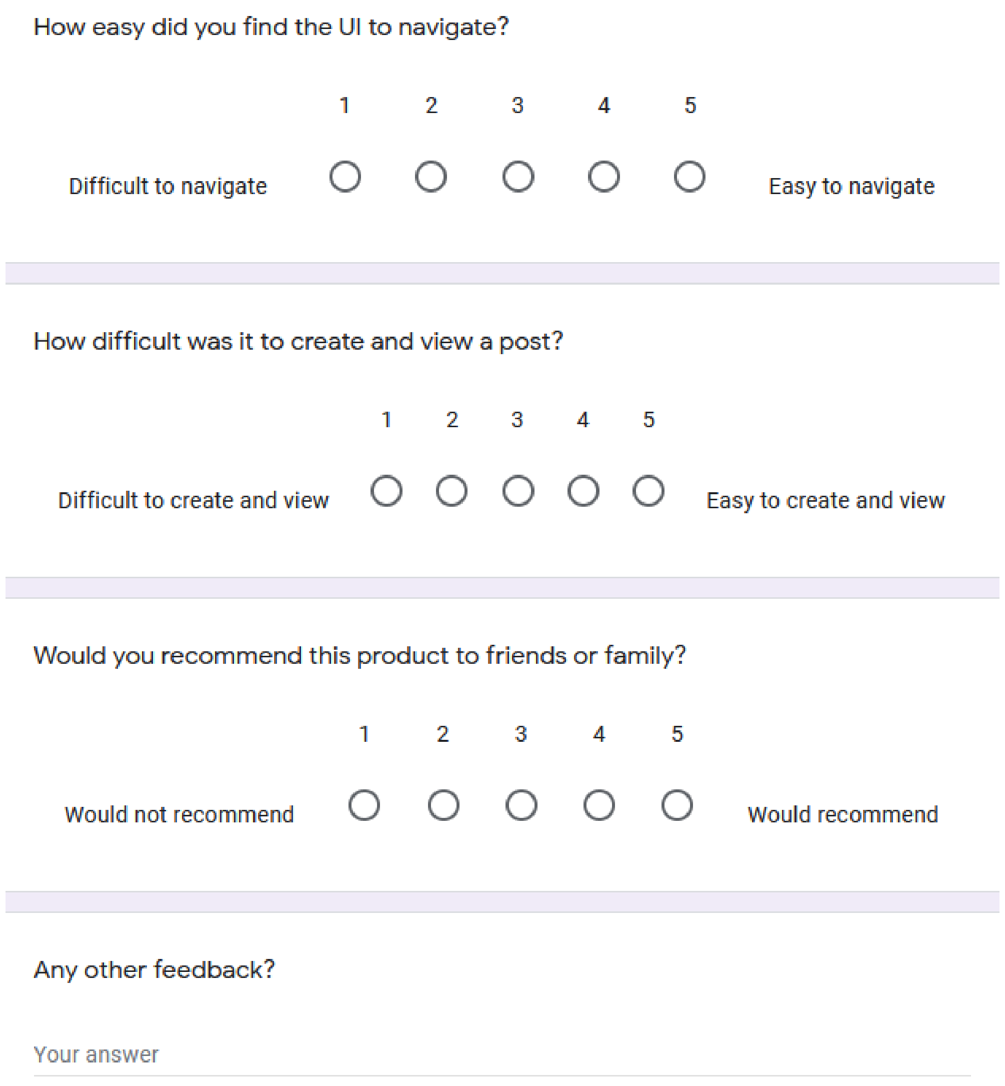
Quantitative Metrics:

Some quantitative metrics that would be used would be the percentage of participants satisfied with the UI and the percentage satisfied with the post function. This percentage would recommend the site to friends and family and the number of bugs that are reported.

Questionnaire Link:

https://forms.gle/rwJQdkEpC3iRMf2V8

Questionnaire Preview:



How easy did you find the UI to navigate?

	1	2	3	4	5	
Difficult to navigate	○	○	○	○	○	Easy to navigate

How difficult was it to create and view a post?

	1	2	3	4	5	
Difficult to create and view	○	○	○	○	○	Easy to create and view

Would you recommend this product to friends or family?

	1	2	3	4	5	
Would not recommend	○	○	○	○	○	Would recommend

Any other feedback?

Your answer

QA Test Plan

The objectives of this test are as follows

1) Ensure the integrity of the post function.
2) Ensure the post function, functions as needed.

    a) Post is viewable by users on the main landing page.
    b) Post can be deleted and edited.
    c) Post is persistent on the main landing page for every user.
3) Post function works properly on multiple browsers.
    a) Tested on Google Chrome, Mozilla Firefox, Apple Safari.

Hardware And Software Setup

Server Side Hardware: Unknown, FAU LAMP Server

Server Side Software: CentOS, PHP 7.0.25, phpMyAdmin 4.9.5, and MySQL 5.1.73

Local Hardware: Computers capable of running Google Chrome, Mozilla Firefox, and Apple Safari.

Local Software: Google Chrome Chrome, Mozilla Firefox, Apple Safari, XAMPP, and VS Code

Feature To Be Tested

The feature that will be tested is the posting of articles on the website. The purpose of choosing this feature is due to how integral it is to the website’s purpose of allowing users to view articles for a set fee. This feature will be tested on Google Chrome, Mozilla Firefox, and Apple Safari to ensure the range of testing is broad.

Actual Test Cases

Case 1: The first actual test case will be that posting an article ensures that it is viewable by the user creating it. The result expected from this would be that the post is viewable by the user who created it.

Case 2: The second actual test case will be ensuring that created posts are viewable by other users. The expected result is that if the article is created by one user, another will be able to find that same article on the main landing page of the site.

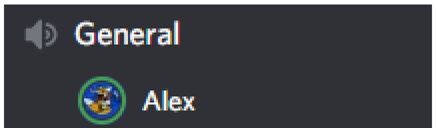
Case 3: The second actual test case is meant to test spam protection within the post system. The expected result is that the article will not be posted if both the title and the body are left blank.

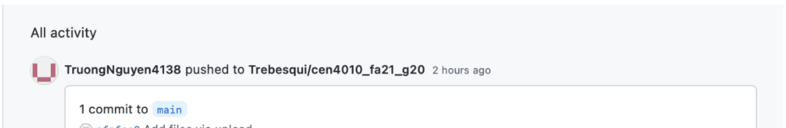
Test #	Test Title	Test Description	Test Input	Correct Output	Pass/Fail
1	Filled Post	In order for the site's purpose to be executed, the post must be located on the main page with the contents filled. To test this, an account will be created and a post will be created under said account. This post will have the title of "test 2" with a body of "test test test test." If the post is viewable on the main page and the contents can be found by clicking the linked article, then the test is successful.	Article with title "test 2" and body "test test test test" created.	Post is viewable on the main page and the contents are correct.	Pass
2	Post Persistence	In order to ensure all content is viewable from the main page to every user, post persistence across multiple users is required. To test this, a user is created under one email with another user created under a different email. The first created user will create a post called "test 2" with a body of "test test test	Article with title "test 2" and body "test test test test" created.	Post viewable from second account.	Pass

		test.” This account will then be logged out of and the second account will be used to check if the post persists on the main page.			
3	Blank Post	In order to ensure the spam creation of posts does not occur, a blank post will not be allowed to be created. In order to prevent any textual errors on the main page, the post will not be allowed to be left blank. In order to test this, an article will be attempted to be created with both the title and the body left blank. As mentioned, this test will ensure the safety and the performance of the system implemented.	Title and body left blank and post created.	Post not created.	Pass

Code Review

- The Striker project employs the use of Functional Programming.
- As it was covered in the Usability and QA Test plans, the Code Review will also be directed towards the “post” feature of Striker.
- Nothing notable was wrong with the code at all and the only things that can be improved upon are addition of things that aren’t there and the reallocation of others to new files.

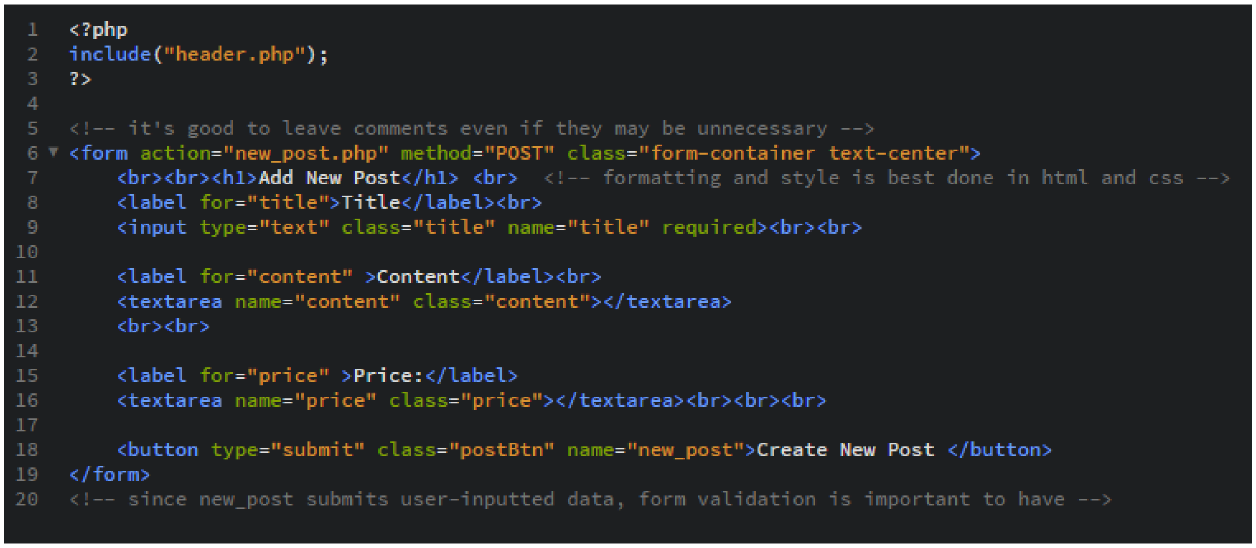



All activity

TruongNguyen4138 pushed to Trebesqui/cen4010_fa21_g20 2 hours ago

1 commit to main

- The above two images represent the group's communication throughout the Code Review process.

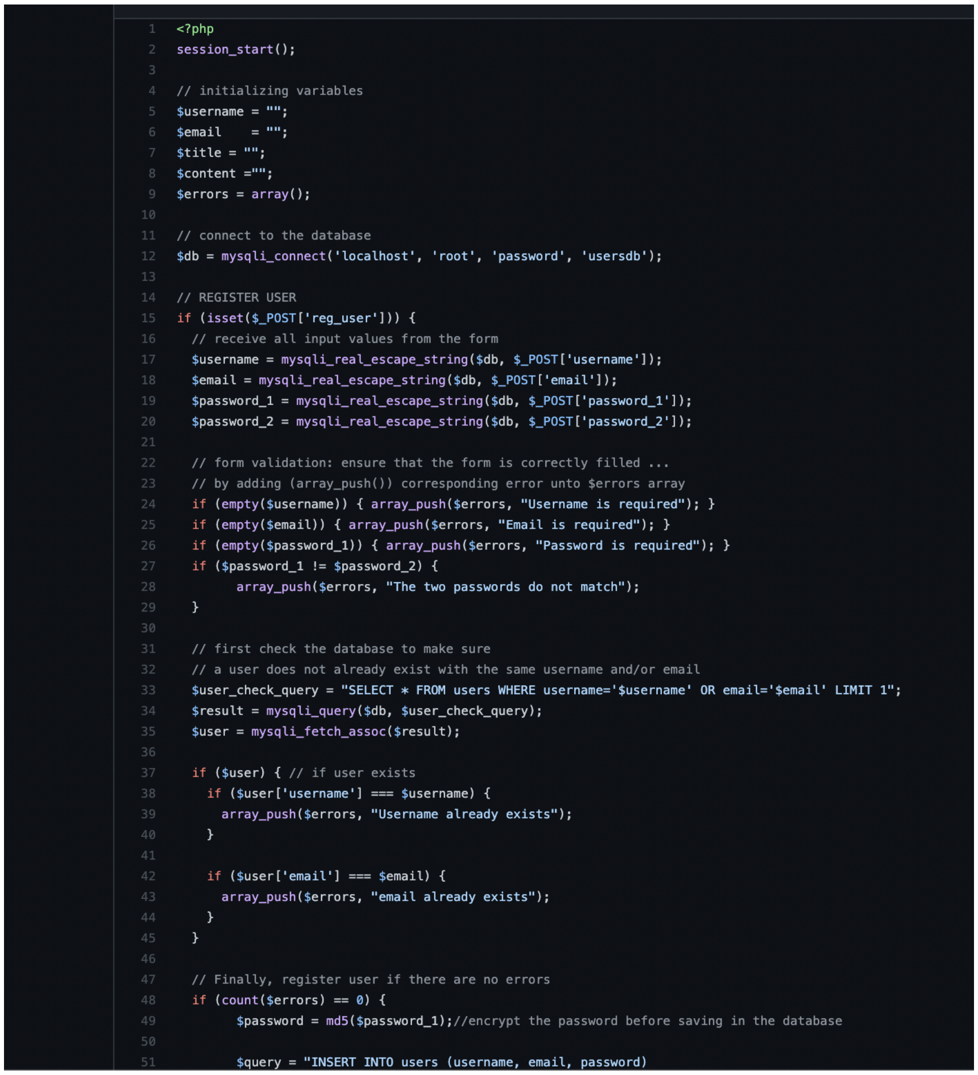


```
<?php
include("header.php");
?>

<!-- it's good to leave comments even if they may be unnecessary -->
<form action="new_post.php" method="POST" class="form-container text-center">
    <br><br><h1>Add New Post</h1> <br>  <!-- formatting and style is best done in html and css -->
    <label for="title">Title</label><br>
    <input type="text" class="title" name="title" required><br><br>

    <label for="content" >Content</label><br>
    <textarea name="content" class="content"></textarea>
    <br><br>

    <label for="price" >Price:</label>
    <textarea name="price" class="price"></textarea><br><br><br>

    <button type="submit" class="postBtn" name="new_post">Create New Post </button>
</form>
<!-- since new_post submits user-inputted data, form validation is important to have -->
```

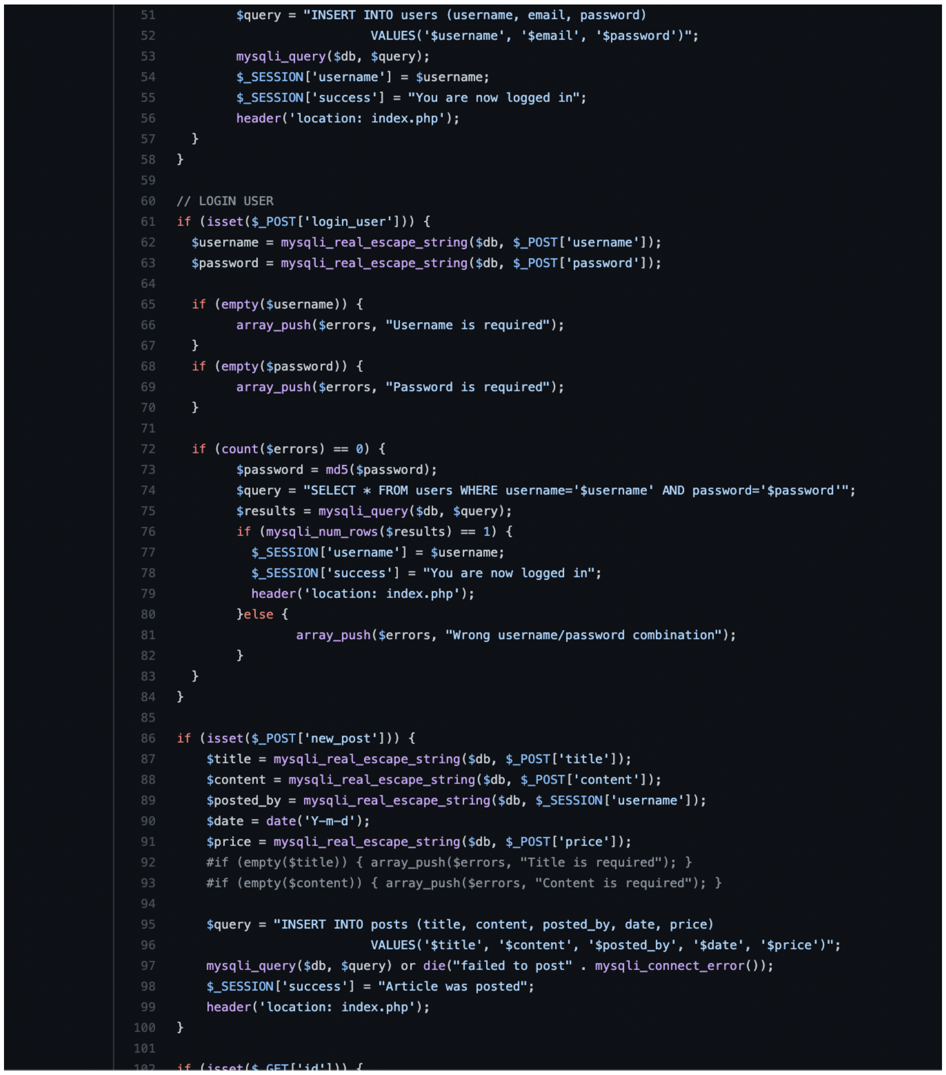
new_post.php

Selfcheck- Security Best Practices

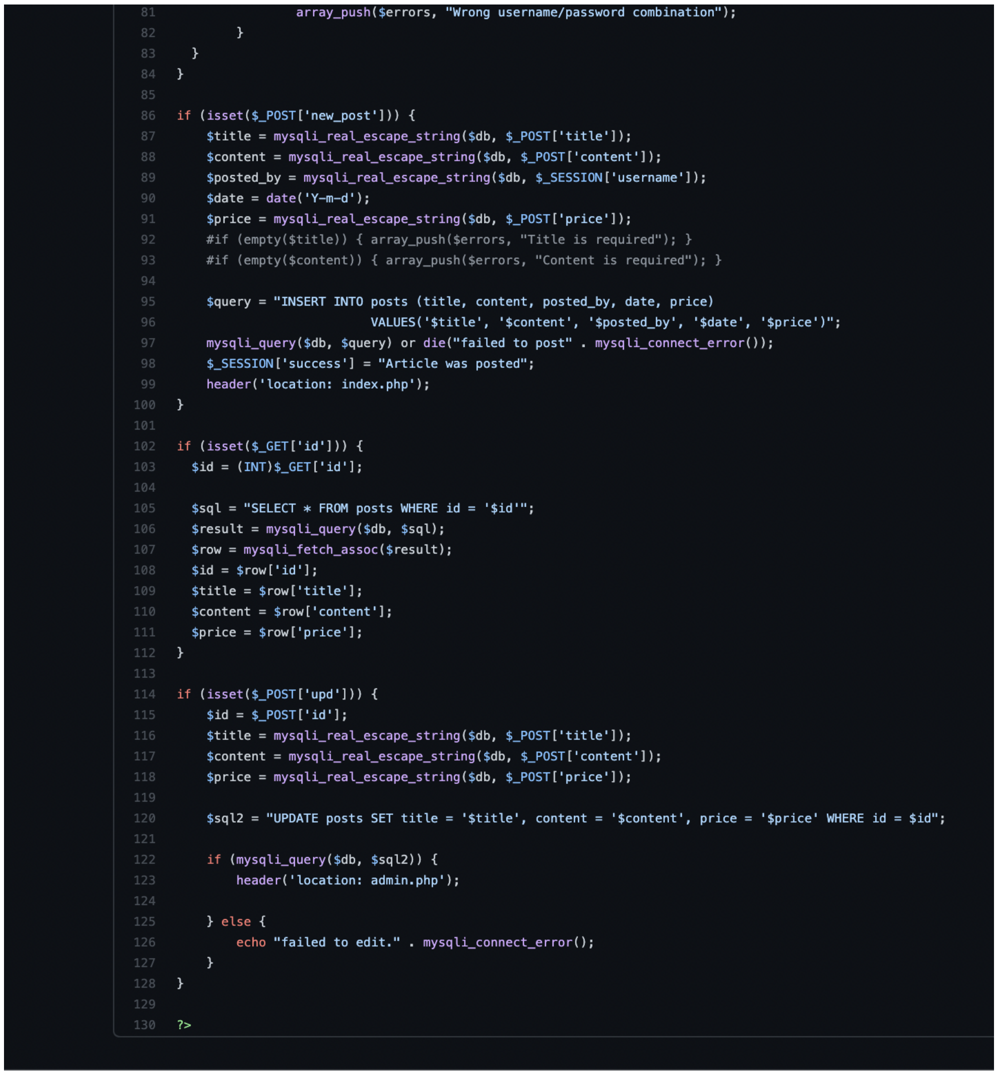
1. The major assets we are protecting are passwords. No other sensitive data is stored on the website. All payment information is handled by stripe through their robust security system.
2. Confirm that you encrypt the password in the database- the password is encrypted in the database.
3. We have used validation on the login bar to ensure that duplicates or invalid inputs are not entered. For example the below code will check if an entry field is empty or a username already exists. The comments provided in the code give a more detailed explanation on what we are actually checking.



```php
<?php
session_start();

// initializing variables
$username = "";
$email    = "";
$title = "";
$content ="";
$errors = array();

// connect to the database
$db = mysqli_connect('localhost', 'root', 'password', 'usersdb');

// REGISTER USER
if (isset($_POST['reg_user'])) {
  // receive all input values from the form
  $username = mysqli_real_escape_string($db, $_POST['username']);
  $email = mysqli_real_escape_string($db, $_POST['email']);
  $password_1 = mysqli_real_escape_string($db, $_POST['password_1']);
  $password_2 = mysqli_real_escape_string($db, $_POST['password_2']);

  // form validation: ensure that the form is correctly filled ...
  // by adding (array_push()) corresponding error unto $errors array
  if (empty($username)) { array_push($errors, "Username is required"); }
  if (empty($email)) { array_push($errors, "Email is required"); }
  if (empty($password_1)) { array_push($errors, "Password is required"); }
  if ($password_1 != $password_2) {
        array_push($errors, "The two passwords do not match");
  }

  // first check the database to make sure
  // a user does not already exist with the same username and/or email
  $user_check_query = "SELECT * FROM users WHERE username='$username' OR email='$email' LIMIT 1";
  $result = mysqli_query($db, $user_check_query);
  $user = mysqli_fetch_assoc($result);

  if ($user) { // if user exists
    if ($user['username'] === $username) {
      array_push($errors, "Username already exists");
    }

    if ($user['email'] === $email) {
      array_push($errors, "email already exists");
    }
  }

  // Finally, register user if there are no errors
  if (count($errors) == 0) {
        $password = md5($password_1);//encrypt the password before saving in the database

        $query = "INSERT INTO users (username, email, password)
```



```php
        $query = "INSERT INTO users (username, email, password)
                          VALUES('$username', '$email', '$password')";
        mysqli_query($db, $query);
        $_SESSION['username'] = $username;
        $_SESSION['success'] = "You are now logged in";
        header('location: index.php');
  }
}

// LOGIN USER
if (isset($_POST['login_user'])) {
  $username = mysqli_real_escape_string($db, $_POST['username']);
  $password = mysqli_real_escape_string($db, $_POST['password']);

  if (empty($username)) {
        array_push($errors, "Username is required");
  }
  if (empty($password)) {
        array_push($errors, "Password is required");
  }

  if (count($errors) == 0) {
        $password = md5($password);
        $query = "SELECT * FROM users WHERE username='$username' AND password='$password'";
        $results = mysqli_query($db, $query);
        if (mysqli_num_rows($results) == 1) {
          $_SESSION['username'] = $username;
          $_SESSION['success'] = "You are now logged in";
          header('location: index.php');
        }else {
                array_push($errors, "Wrong username/password combination");
        }
  }
}

if (isset($_POST['new_post'])) {
    $title = mysqli_real_escape_string($db, $_POST['title']);
    $content = mysqli_real_escape_string($db, $_POST['content']);
    $posted_by = mysqli_real_escape_string($db, $_SESSION['username']);
    $date = date('Y-m-d');
    $price = mysqli_real_escape_string($db, $_POST['price']);
    #if (empty($title)) { array_push($errors, "Title is required"); }
    #if (empty($content)) { array_push($errors, "Content is required"); }

    $query = "INSERT INTO posts (title, content, posted_by, date, price)
                          VALUES('$title', '$content', '$posted_by', '$date', '$price')";
    mysqli_query($db, $query) or die("failed to post" . mysqli_connect_error());
    $_SESSION['success'] = "Article was posted";
    header('location: index.php');
}

if (isset($_GET['id'])) {
```



```php
                array_push($errors, "Wrong username/password combination");
        }
  }
}

if (isset($_POST['new_post'])) {
    $title = mysqli_real_escape_string($db, $_POST['title']);
    $content = mysqli_real_escape_string($db, $_POST['content']);
    $posted_by = mysqli_real_escape_string($db, $_SESSION['username']);
    $date = date('Y-m-d');
    $price = mysqli_real_escape_string($db, $_POST['price']);
    #if (empty($title)) { array_push($errors, "Title is required"); }
    #if (empty($content)) { array_push($errors, "Content is required"); }

    $query = "INSERT INTO posts (title, content, posted_by, date, price)
                          VALUES('$title', '$content', '$posted_by', '$date', '$price')";
    mysqli_query($db, $query) or die("failed to post" . mysqli_connect_error());
    $_SESSION['success'] = "Article was posted";
    header('location: index.php');
}

if (isset($_GET['id'])) {
  $id = (INT)$_GET['id'];

  $sql = "SELECT * FROM posts WHERE id = '$id'";
  $result = mysqli_query($db, $sql);
  $row = mysqli_fetch_assoc($result);
  $id = $row['id'];
  $title = $row['title'];
  $content = $row['content'];
  $price = $row['price'];
}

if (isset($_POST['upd'])) {
    $id = $_POST['id'];
    $title = mysqli_real_escape_string($db, $_POST['title']);
    $content = mysqli_real_escape_string($db, $_POST['content']);
    $price = mysqli_real_escape_string($db, $_POST['price']);

    $sql2 = "UPDATE posts SET title = '$title', content = '$content', price = '$price' WHERE id = $id";

    if (mysqli_query($db, $sql2)) {
        header('location: admin.php');

    } else {
        echo "failed to edit." . mysqli_connect_error();
    }
}

?>
```

Selfcheck- Non-Functional Specifications

Performance:

1. Load Speed: The ability to move between web pages within the system should take around 600 MS to 1.3 seconds to complete. Reaching the main feed should take around that time frame as well as loading into articles. Done

2. Responsive Time: The system will remain responsive no matter the resolution or type of device in place. Whether it be a resolution of 1920x1080 or 1792x828, the site will remain responsive. Done

Security Requirements:

1. Account Based System:

   The site, Striker, would be using an account system in order to keep track of comments and other rating metrics. To ensure the protection of user personal data:

   a. A security question and phone number will be needed to create an account. In progress(should be finished with integration with firebase)

   b. The security question will be required in order to reset the password if needed. In progress(should be finished with integration with firebase)

   c. Passwords will be required to be 8 characters long and include both uppercase and lowercase characters. In progress(should be finished with integration with firebase)

   d. Password hashing will be put in place to protect the user's account from malicious intent. In progress(should be finished with integration with firebase)

2. Payment Protection:

   With payment processing being a part of our system, protection would need to be put in place to ensure the safety of our users. To ensure this, the site will not store any information regarding a user's preferred payment process. Done

3. User Privacy:

   In order to ensure the privacy of the user, all user data will only be accessible to the user and the development team. The user would also have the option to omit their full name for a different display name in order to ensure anonymity. Done

4. Spam Protection:

   In order to ensure the safety of the user from bot spam accounts:

   a) A captcha will be asked on the creation of an account using the reCAPTCHA API. In progress(should be finished with integration with firebase)

   b) To protect against spam in comments and articles, both of these systems will require an account to access and use. Done

Storage:

1. Data Storage: All of the data used within this system will be stored within the MySQL database in the LAMP Server. Done

2. File Storage: Files will primarily be held within the LAMP server. All files used will be backed up to GitHub and Google Drive to ensure its safety. Done

Availability:

1. Uptime: The website, Striker, will be accessible 24 hours a day, 7 days a week. Done

2. Downtime: This system does not expect much downtime to occur unless LAMP is unavailable. If downtime is necessary, an announcement will be shown on the page prior and a redirection to a separate page stating the site is down will occur during downtime. Done

Ease of Use:

1. Training: Minimal training time would be required in order to use our system. The only required training would be how to create and upload articles to our site. Past this, the site would be user-friendly and usable by new users. Done

Interoperability Requirements:

1. Browser Compatibility: This site, Striker, would display and run properly on web browsers Google Chrome, Mozilla Firefox, and Apple Safari. Done

2. Device Compatibility: This system should display and run properly on multiple devices, such as mobile devices, desktop computers, and laptops. Design would stay consistent throughout desktop computers and laptops, with it slightly deviating on mobile. Done

3. Fault Tolerance: Exception Handling: If an error were to occur at any moment, an error code will be displayed to the user. Using this error, the user will be able to troubleshoot the issue. Done

Supportability Requirements:

1. Code Standards: This system will be programmed using HTML, CSS, and PHP in order to accomplish its goals. Each group member will be using the data names mentioned previously in code and documentation. Done

Expected Load:

1. User Load: The site will be able to handle a maximum of 35 people before performance begins to degrade. Even if performance begins to degrade, no crashes are expected to happen due to user load. Done

Demo URL:
https://www.youtube.com/watch?v=2pCeKmsm7kU