Lab 08: Form Processing – PHP Arrays and Superglobals

Task 1: Given the following PHP code to enter 6 test scores ("addScores.php")

```
<!DOCTYPE html>
<html>
<body>
<h1>Enter Scores:</h1>
<form method="POST" action="grader.php" >
   Score 1 <input type="text" name="score1"/><br/>
   Score 2 <input type="text" name="score2"/><br/>
   Score 3 <input type="text" name="score3"/><br/>
   Score 4 <input type="text" name="score4"/><br/>
   Score 5 <input type="text" name="score5"/><br/>
   Score 6 <input type="text" name="score6"/><br/><br/>
   <input type="submit">
</form>
</body>
</html>
```

(a) Modify the PHP program that you have written in Lab06 – Task 2 "grader.php" to process the form request and calculate average grade.

- Define variables to store the input data, all the scores sent from the form with the POST method in the "addScores.php" page.
- Define a variable called "scores" to takes a single parameter: an array of test scores (all numbers).
- Compute and return the average score in the array, **rounded to one decimal place**.
- Write a new function named "checkGrade" that use program control statement to print the grade of the average score. The grading scheme is as follows:
  - 80 – 100: A
  - 60 – 79.9: B
  - 40 – 59.9: C
  - 20 – 39.9: D
  - 1 – 19.9: E
  - 0 – 0.99: F
- Examples:
  Sample scores: 90, 98, 89, 100, 100, 86
  Sample output:
  Average score for [90 98 89 100 100 86] is 93.8
  Average grade is A

  Sample scores: 40, 65, 77, 82, 80, 54
  Sample output:
  Average score for [40 65 77 82 80 54] is 66.3
  Average grade is B

(b) Save "addScores.php" as **"addScoresSelf.php"**:

i. Modify the *form action* to use PHP superglobal variable `$_SERVER['PHP_SELF']` to point to this file itself for processing form data:

```
<form method="post"
action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
```

ii. Copy the grader.php code into checkgradeSelf.php file to process the form request on this file itself.

NOTE: `$_SERVER["PHP_SELF"]` exploits can be avoided by using the `htmlspecialchars()` function.

(c) Save "addScores.php" as "**addScoresGet.php**" and modify the form method from HTTP POST method to HTTP GET method. Next modify the "grader.php"code to accept both HTTP POST and GET methods using the if…else statement and superglobal variable `$_SERVER['REQUEST_METHOD']`.

(d) Test all the web pages (addScores.php, addScoresSelf.php, addScoresGet.php and grader.php) in a browser to see the differences between HTTP POST and GET methods.

References:

https://www.w3schools.com/php/php_arrays.asp

https://www.w3schools.com/php/php_superglobals.asp

https://www.w3schools.com/php/php_forms.asp

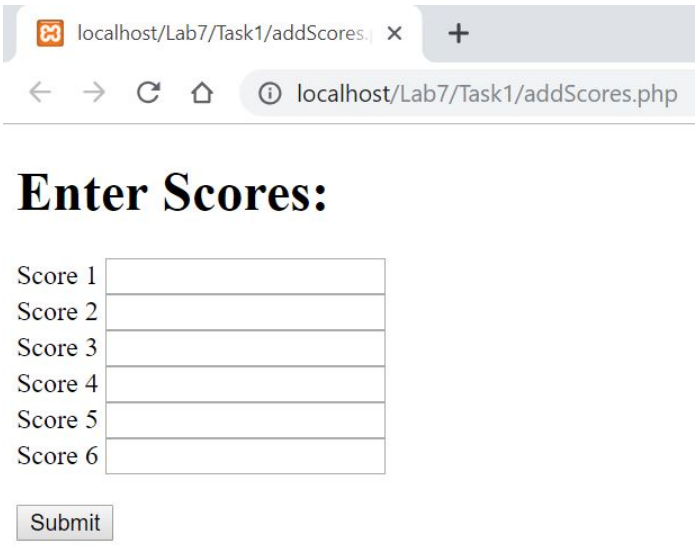

Figure 1: Screenshot for addScores.php

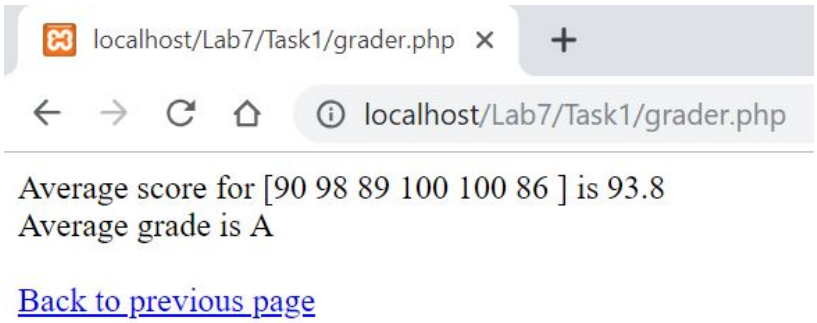

Figure 2: Screenshot for grader.php using POST method

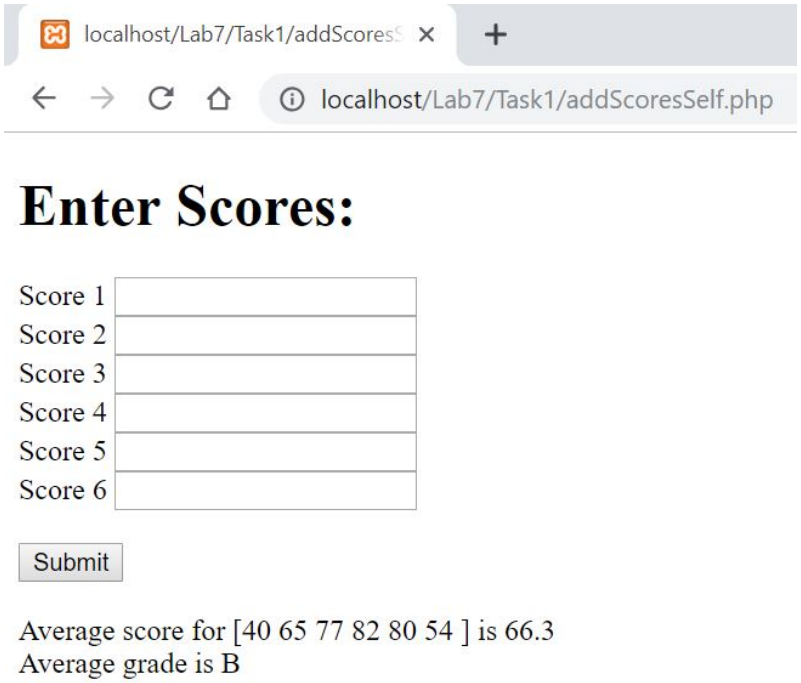

Figure 3: Screenshot for addScoresSelf.php

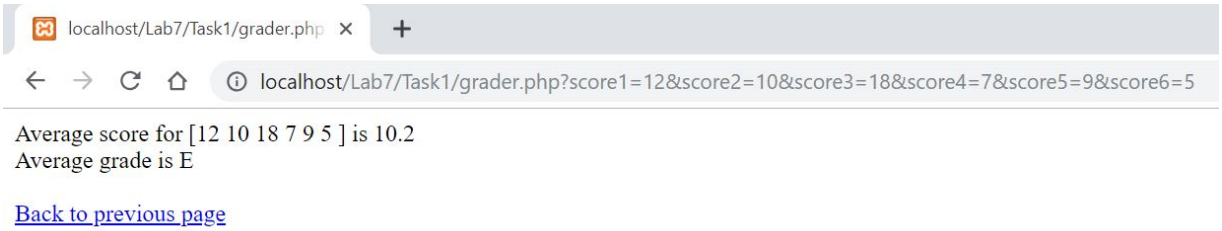

Figure 4: Screenshot for grader.php using GET method

Task 2: Using PHP in JavaJam Coffee House website:

Create new folder and files:

- Create a new folder on your C drive's `\xampp\htdocs` called "**javajam6**". Copy **all** the files from your previous lab folder, e.g. Lab 6's folder (javajam5) into the "**javajam6"** folder.
- Create a new PHP file name "**signup.php**" and store it in "**javajam6**" folder.
- Create a new PHP file name "**login.php**" and store it in "**javajam6**" folder.
- Create a new PHP file named "**processSignup.php**" and store it in "**javajam6"** folder.
- Create a new PHP file named "**profile.php**" and store it in "**javajam6"** folder.
- Create a new text file named "**javamember.txt**" and store it in "**javajam6"** folder.

NOTE: You can reuse the UI design that you have used for existing **JavaJam Coffee House** web pages (e.g. index.html, jobs.html) but modify the navigation menu to include new navigation links:

- Sign Up" links to signup.php
- "Login" links to login.php
- "Profile" links to profile.php
- "Logout" links to logout.php

Instructions:

1. Open the signup.php file and include the following HTML elements:
    a. Add a <h1> with the text: "Sign Up as JavaJam Member"
    b. Add a paragraph: "Please fill in this form to create an account. Required information is marked with an asterisk (*)."
2. Create a signup form using the HTML <form> element to collect user inputs:
    a. Add the form attributes: method="post", action="processSignup.php "
    b. Add the following **labels** and **input**s:
        i. ***Name:** using *text* input type
        ii. ***E-mail:** using *email* input type
        iii. ***Password:** using *password* input type
        iv. ***Repeat Password:** using *password* input type
    c. Add a submit button with the value "Sign Up"
    d. Add 'required' attribute to validate a required input is not empty

3. Modify the form to include a **name** for each input field. For example:

```
<div class="form-group">
<label for="mName">*Name: </label> <br>
<input type="text" class="form-control col-sm-4" id="mName"
name="mName" required="required"> <br>
</div>
<div class="form-group">
<label for="myEmail">*E-mail: </label> <br>
<input type="email" class="form-control col-sm-4" id="mEmail"
name="mEmail" required="required"> <br>
```

```
<div class="form-group">
<label for="password">*Password:</label>
<input class="form-control col-sm-4" id="password" name="password"
type="password" required>
</div>
<div class="form-group">
<label for="rpassword">*Repeat Password:</label>
<input class="form-control col-sm-4" id="rpassword" name="rpassword"
type="password" required>
            </div>
```

4. Open the login.php file and include the following HTML elements:
    a. Add <h1> with the text: “Login”
    b. Add a paragraph: “Please sign in”
    c. Create a login form using the HTML <form> element to collect user input:
        i. Add the form attributes: method="post", action=”processLogin.php"
        ii. Add the following labels and inputs:
        iii. E-mail: using *email* input type
        iv. Password: using *password* input type
        v. Add a submit button with the value “Sign in”

When the user fills out the form in *“signup.html”* and clicks the ***submit*** button, the form data is sent for processing to a PHP file named *“processSignup.php”*. The form data is sent with the **HTTP POST** method.

5. In the *“processSignup.php”* file, write PHP codes to:
    a. Define variables to store the input data, sent from the previous page, Sign Up form (signup.html) with the POST method. For example:

```
$uname = $_POST["uname"];
$email = $_POST["email"];
```

    b. Use the PHP File functions (e.g. fopen, fwrite, fclose, file_put_contents) to add (append) all the input data into the *“javamember.txt”* file.
    c. Add the following line at the end of your PHP codes to redirect to **PROFILE page**, *“profile.php”* with the *“login_success”* status:

```
header("Location: profile.php?action=login_success");
```

6. In the same *“profile.php”* file, write PHP and HTML codes to:
    a. Read the last line of the “javamember.txt” file and read the new user’s name and email from the last line of data.
    b. Print a message to welcome the new user to **JavaJam Coffee House** member’s page.
    c. Display all the user data (except password) passed from *“signup.html”* and stored in the *“javamember.txt”* in an HTML table.

HINTS:

- Reference to read the last line of file in PHP: https://www.tutorialspoint.com/read-last-line-from-file-in-php
- After reading the last line, refer to the sample codes of lecture notes, **"15-readFiles.php"** to learn how to read the substring of the last line from a file when the data is comma-delimited ','. (Note: Since there is only one line, the last line, so you don't need to use `foreach` loop.)

Figures 5, 6, 7 and 8 show the screenshots of the "signup.php", "login.php", "profile.php", and "javamember.txt".

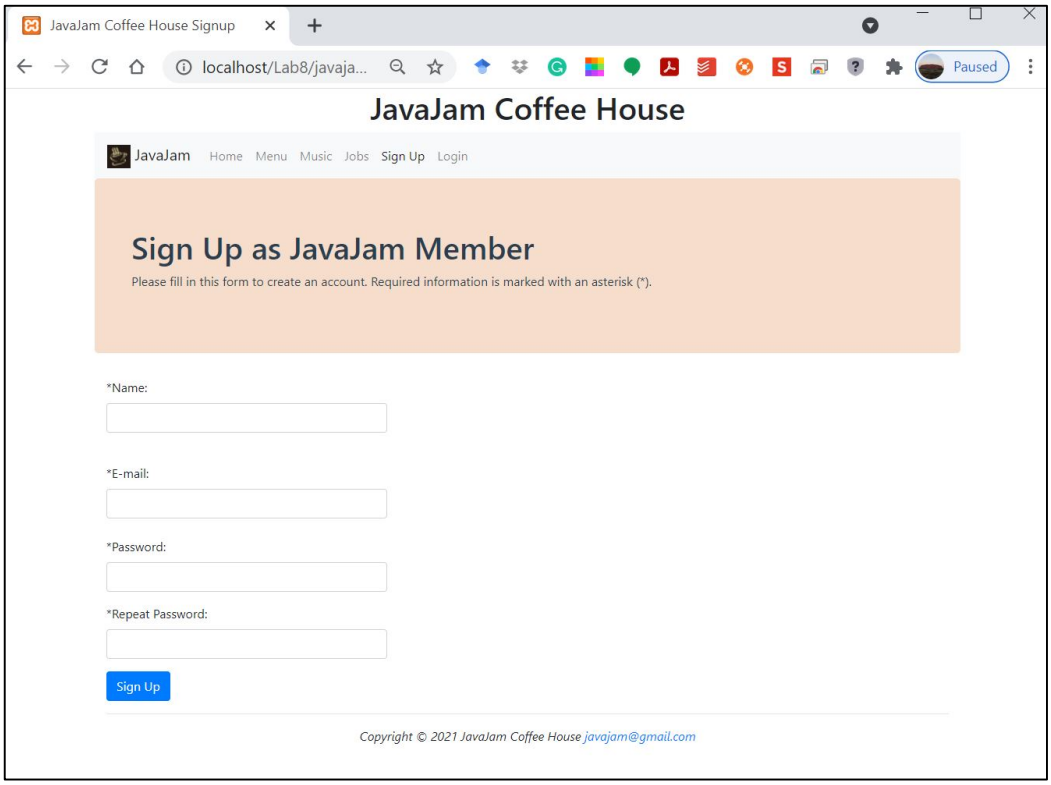

Figure 5: JavaJam Coffee House's Sign Up page

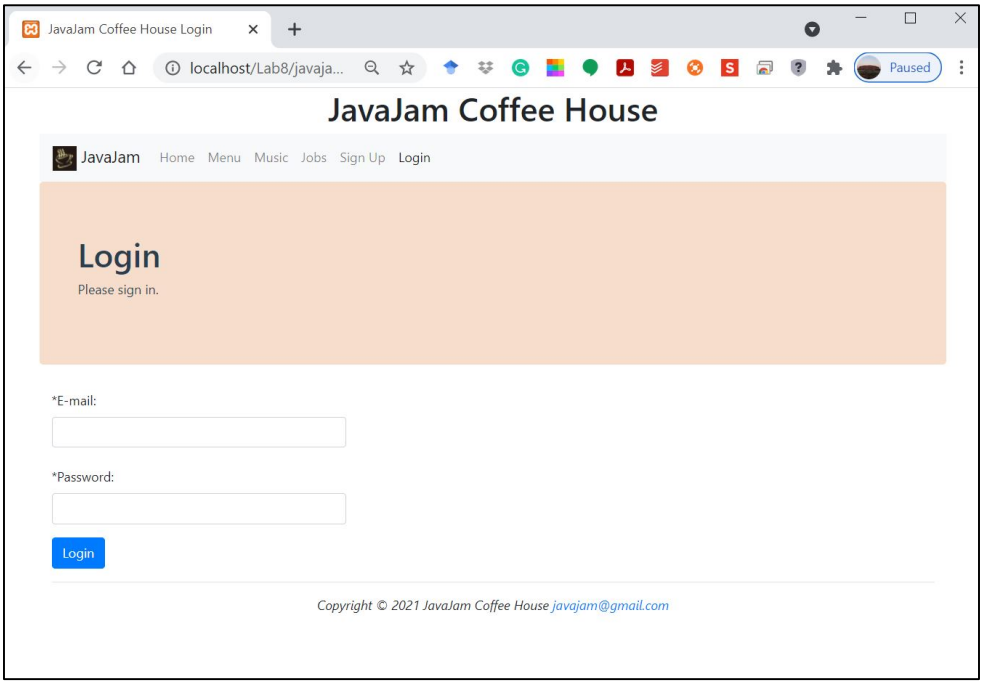

Figure 6: JavaJam Coffee House's Login page

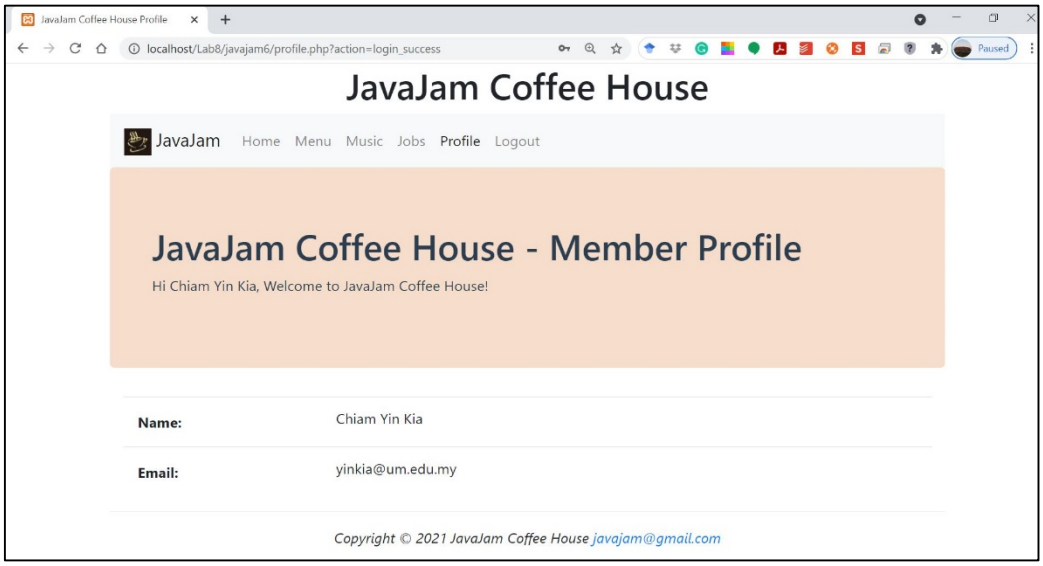

Figure 7: JavaJam Coffee House's Profile page

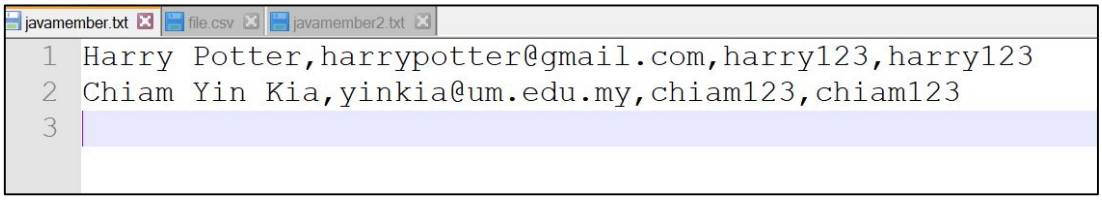

Figure 8: Sample screenshot for "javamember.txt, input data saved as a string