CMIS 4500 Week 4 Assignment

In this assignment we will practice how to work with sessions to store array values through multiple post backs. In the week 3 assignment, our arrays did not save changes made it beyond a button click. By using sessions, any changes made to the array can persist by storing the array in a session.

This exercise needs to be completed in steps. You are required to work on every php file in the folder called *YourName*_Week4. Please replace "*YourName*" with your actual name while saving the folder. You will also be required to answer the reflection questions one by one in notes.txt. Please use main.css as your style sheet.

Part 1

Step 1: Storing arrays in a session and using the session variable to retrieve array values.

Please go over the code provided to you in the file *index_step4.php.* This file is very similar to the index_step4.php from week 3 assignment, except with the notable difference that it contains session variable called $_SESSION['employee_salary']. Changes such as addition, deletion , or update take place via this session variable. Data is displayed using this session variable. In fact, this session variable can be used in lieu of the array reference variable $employee_salary from week 3 assignments.

While naming the session variable $_SESSION['employee_salary'], 'employee_salary' is a key that would be used to identify the contents of what would have been in the $employee_salary array. A session can carry multiple units of data (arrays or otherwise). Each data unit stored in a session is identified by the key of that data unit. In this example, the identifier / key is 'employee_salary'.

The array that we want to store in a $_SESSION is already a complex, associative array that has its own key. For example, userId is already a key. So, when we call $_SESSION['employee_salary'], we deal with multiple keys. One key, 'employee_salary' identifies the associative array ($employee_salary in week 3) itself. Then the userId key identifies each element of the associative array.

This file contains the php code to do the following. Listed below are the alterations made to the file index_step4.php provided to you in week 3 assignment. Please understand the code thoroughly.

- Check if a session exists (lines 19- 25) . If not, create one. Without this little check, pre-existing sessions can trigger a warning.
- Check if the session variable $_SESSION['employee_salary'] exists. If not, create one and initialize it to an empty array (lines 27-29)

- Load the session variable $_SESSION[‘employee_salary’] with a few elements (lines 32-34) by specifying the userId(key) and hourlySalary(value).
- In the if-statement block that adds and entry, add data directly to the array stored in the session (line 52). To do this use the session variable $_SESSION[‘employee_salary’] and the key in $userID and value in $hourlySalary.
- Similarly, update changes directly to the array stored in the session variable (line 64).
- Also, delete any row in the array stored in the session, as in line 72. Unset will delete the row.
- Display contents of the array in the session, as shown in line 99.
- Add validation for Add and Update and Delete Entry by checking if a key (userId) already exists / doesn’t exist in the session array. If the key exists, give an error message and donot add the new entry.

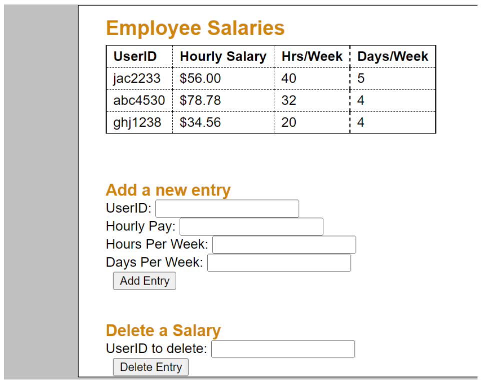
Please write this code by yourself and practice how to work with session arrays. There is no need of any validation code at this point. You do not need to add anything more to the code supplied in index_step4.php.

Step 2: Store a 2D array in a session and use the session variable in your code to add and delete array contents.

Please open the file *index_step5.php.* Write the code to do the following:

To do in index_step5.php:

- Check if an session exists. If not, create one. Without this little check, pre-existing sessions can trigger a warning.
- Check if the session variable $_SESSION[‘employee_salary2D’] exists. If not, create one and initialize it to an empty array.
- Load the session variable $_SESSION[‘employee_salary2D’] with a few elements (lines 10-12 in the comments) by specifying the values of the super and sub keys. Use the same array values that you did in week 3 assignment.
- Write the html code that displays the contents of the $_SESSION[‘employee_salary2D’].
- Write the html code to create two forms : one to Add an Entry and another to Delete an entry. This part is similar to what you did in week 3 .



Employee Salaries

UserID	Hourly Salary	Hrs/Week	Days/Week
jac2233	$56.00	40	5
abc4530	$78.78	32	4
ghj1238	$34.56	20	4

Add a new entry

UserID:

Hourly Pay:

Hours Per Week:

Days Per Week:

Add Entry

Delete a Salary

UserID to delete:

Delete Entry

- Retrieve the button value and set the $action variable (as you did in week 3).
- Based on the value of $action, filter the data from the text fields.
- Validate the userId for each form. If userID exists, give error messages for “Add entry” and do not add an entry. If the userID doesnot exist, do not delete entry and give appropriate error messages=.
- If validation passes:
    - Add data to the new row , by adding directly to the $_SESSION[‘employee_Salary2D’].
    - Delete the required row from $_SESSION[‘employee_Salary2D’].

Part II

In this part of the assignment, you will apply whatever you have learnt thus far to create an application that allows users to login. Traditional applications have a data-driven login function that queries a database to check username and passwords. Data persistence is, therefore, important to implement login functions. However, we have not covered databases in this course yet. Therefore, we will emulate data persistence using sessions. More, specifically, we will emulate a user table containing usernames and passwords, using a 2D array. And we will store this 2D user array in a session to emulate data persistence for our application. You will be writing your solutions into the files : login.php , display_login.php and user_display.php.

Here are the things your application needs to accomplish (We call them user stories in app development):

As an admin, I want to login to the application, so that I can see who all the users are.

As an employee, I want to login to the application, so that I can see who my clients are.

As a client, I want to login to the application, so that I can see who the site admin is.

While there are several ways to implement the above user stores, we will pick one way so that you get some practice working with session across multiple pages. You could essentially do all of this one page too, but bear with me – we are going to have multiple pages.

__Lets consider the following login process:__

__Your To dos:__

- A login page containing a login form allows the user to enter a username and password, and a submit button allows the user to submit data to the server.
- The login form, as described above, will send the user to a page called **display_login.php**.
- The user's inputs are filtered and validated as follows:
    - The user's input – username and password values are searched in a session array called $_SESSION['user']. [*There is a cheat on how to create this array in almost invisible ink at the end of this document*] The session array stores is assumed to store all the usernames and passwords (*not a good practice, but since we don't have a database, we will grudgingly tolerate storing these sensitive data in a session array that is accessible to all the pages in the application – a vulnerability indeed !*). In addition to the username and passwords, the session array also stores the userType , which could take the values : "admin", "employee", or "client". *You may use the username as a primary key in the 2D array (although you could use any unique primary key and let username be another attribute in the array, along with password and userType. Both options have their own sets of advantages and disadvantages that depends on the specific application's problem's context.*
    - If the username and password values do not match for any of the rows (primary key) in the session array , then user is displayed the login form again. (Remember, how you dealt with a failed validation in previous assignment) .You may give an error statement to the user and have them re-enter the username and password and resubmit these values.
    - If the username and password match: Display the user a welcome information in the display_login page.
    - If the username and password match: also, store the username (alone) in a separate session variable and $_SESSION['user_name']. Also store the userType value for that user In a session variable called $_SESSION['user_type'] We will need to retrieve these values in other pages of the application.

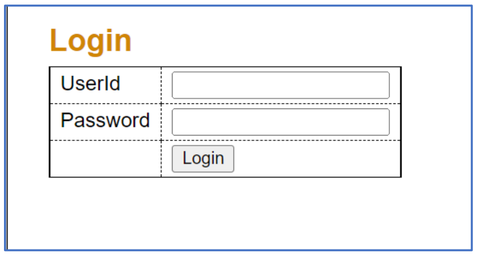
- The display_login.php page displays a button called "Get More Info". The user gets to see this 'magic button' only if they pass the login validation. The button click should take the user into a page called user_display.php. This page will discriminate the users based on their userType values.
    - The display of contents in the user_display.php page depends on the userType value stored in $_SESSION['user_type'].
    - If the userType is "admin", display the usernames and userTypes of all the items in the user table stored in the $SESSION['users'].
    - If the userType is "client", provide a contact email for the admin (just make up an email).
    - If the user-Type is "employee" , show all the client's usernames.
    - You can choose to display data in any format – tables, lists etc.

Helpful Tip: If at any point you want to reset /delete any session array, you may do so by writing $_SESISON = array(); This should empty any previously created session array.

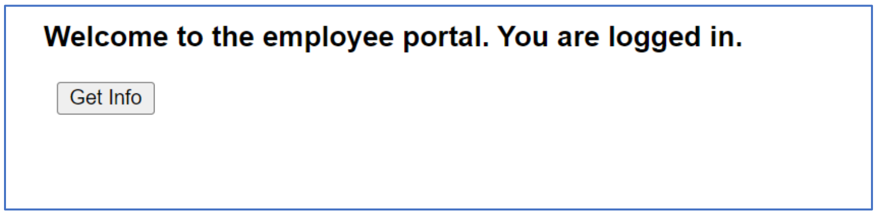
Needless to say – var_dump all the arrays and session variables as needed on all the pages. You may retain these in the final code- helps me grade too. Check those pesky <?php } ?> in your html segments , particularly in the page user_display.php .

Sample screenshots of pages:

1. **login.php form :**

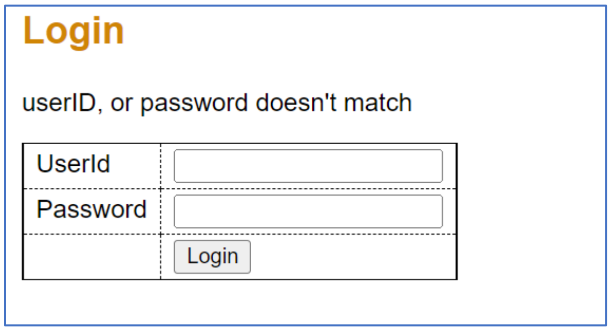


2. **display_login.php (if login validation is correct)**



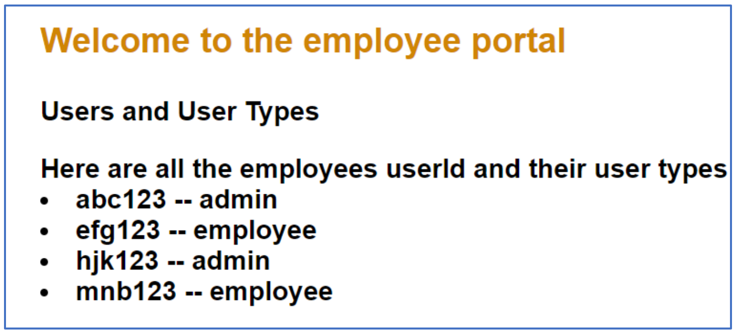
Welcome to the employee portal. You are logged in.

3. **display.login.php (if username/password do not match)**



userID, or password doesn't match

4. **user_display.php (if $_SESSION['user_type"] is an "admin")**



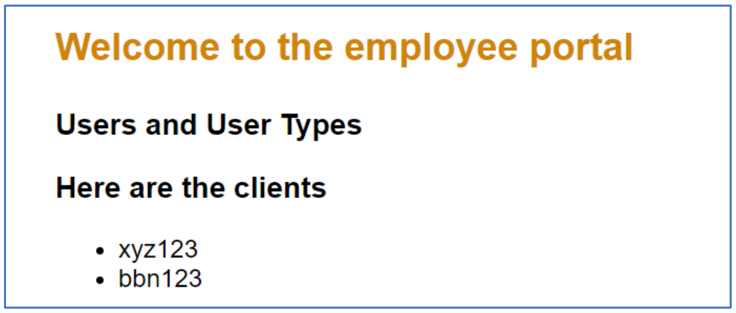
Welcome to the employee portal

Users and User Types

Here are all the employees userId and their user types

- abc123 -- admin
- efg123 -- employee
- hjk123 -- admin
- mnb123 -- employee

5. user_display.php (if $_SESSION['user_type"] is an "employee")



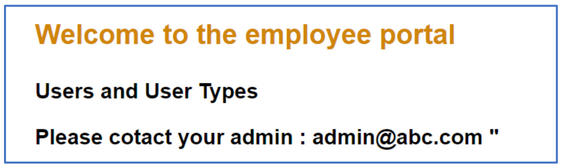
Welcome to the employee portal

Users and User Types

Here are the clients

- xyz123
- bbn123

6. user_display.php (if $_SESSION['user_type"] is a "client")



Welcome to the employee portal

Users and User Types

Please cotact your admin : admin@abc.com "

Ok, since you got to reading this far – I'll give you a hint (actually the code) on how to create your user array in a session :

```
if(empty($_SESSION['user'])){
                    $_SESSION['user'] = array();
                    $_SESSION['user']['abc123'] = array("password" => "pass1", "userType" => "admin");
                    $_SESSION['user']['efg123'] = array( "password" => "pass2", "userType" => "employee");
                    $_SESSION['user']['hjk123'] = array("password" => "pass3", "userType" => "admin");
                    $_SESSION['user']['mnb123'] = array( "password" => "pass4", "userType" =>
"employee");
                    $_SESSION['user']['xyz123'] = array( "password" => "pass5", "userType" => "client");
```

```
            $_SESSION['user']['bbn123'] = array( "password" => "pass6", "userType" => "client");
    }
```

Grading :

Part I to dos:

index_step5.php – 35 pts

Part II : login application :

login.php : 20 pts

display_login.php : 35

user_display.php : 20