COMP3322 MODERN TECHNOLOGIES ON WORLD WIDE WEB

Workshop – PHP, JavaScript and XHR

Overview

In this workshop, we will develop a simple web-based course attendance system, with which we can maintain an attendance list. As shown in Fig. 1, each attendance entry includes the attendance status (PRESENT or ABSENT), student name, student major, course code, and date.

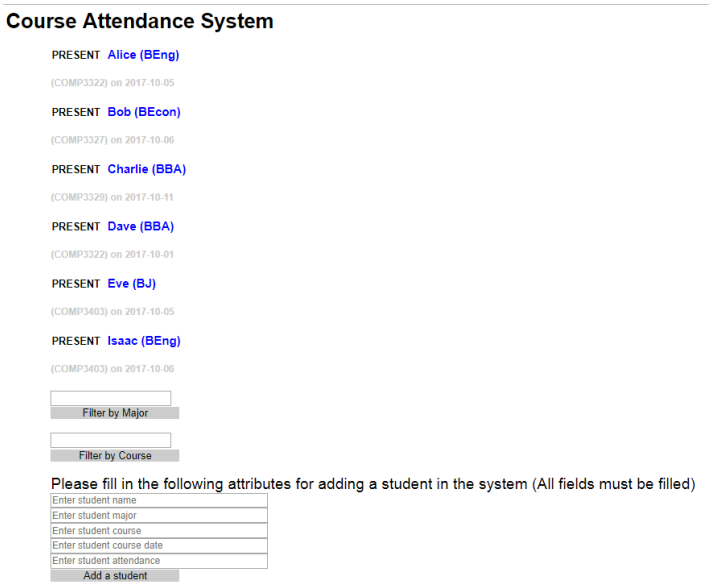


Fig. 1

We will practice loading the list from the database and toggling between the PRESENT status and the ABSENT status, as well as adding a new student in the attendance system. We will also implement a few buttons to decide the (selected) entries to be displayed. For example, when "BBA" is typed into the textbox above "Filter by Major" button and the button is pressed, the filtered result will be shown up as in Fig. 2.

Fig. 2

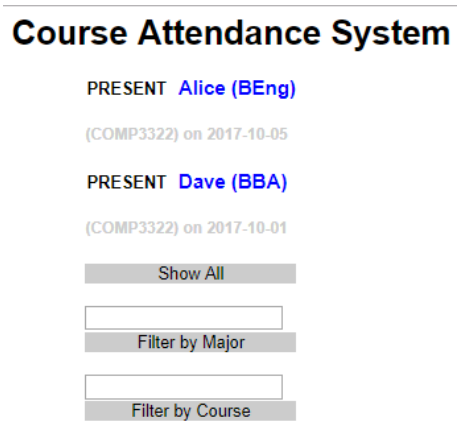


Fig. 3

When "COMP3322" is typed into the textbox above "Filter by Course" button and the button is pressed, the filtered result will be shown up as in Fig. 3. When a student's information is

typed in the textboxes below “Please fill in the following attributes for adding a new student in the system (All field must be filled)” and then press the button “Add a new student”, the newly added student will be displayed, as shows up in Fig. 4 & 5.

Note that a “Show All” button is displayed in Fig. 2 and Fig. 3, and when it is clicked, the page view goes back to Fig. 1.

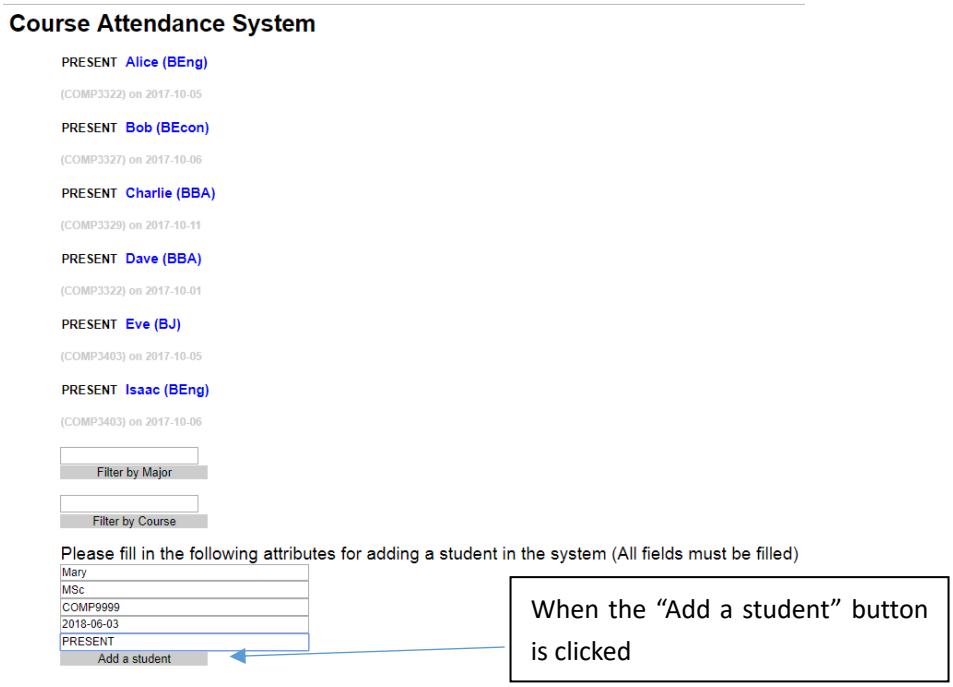


Fig. 4

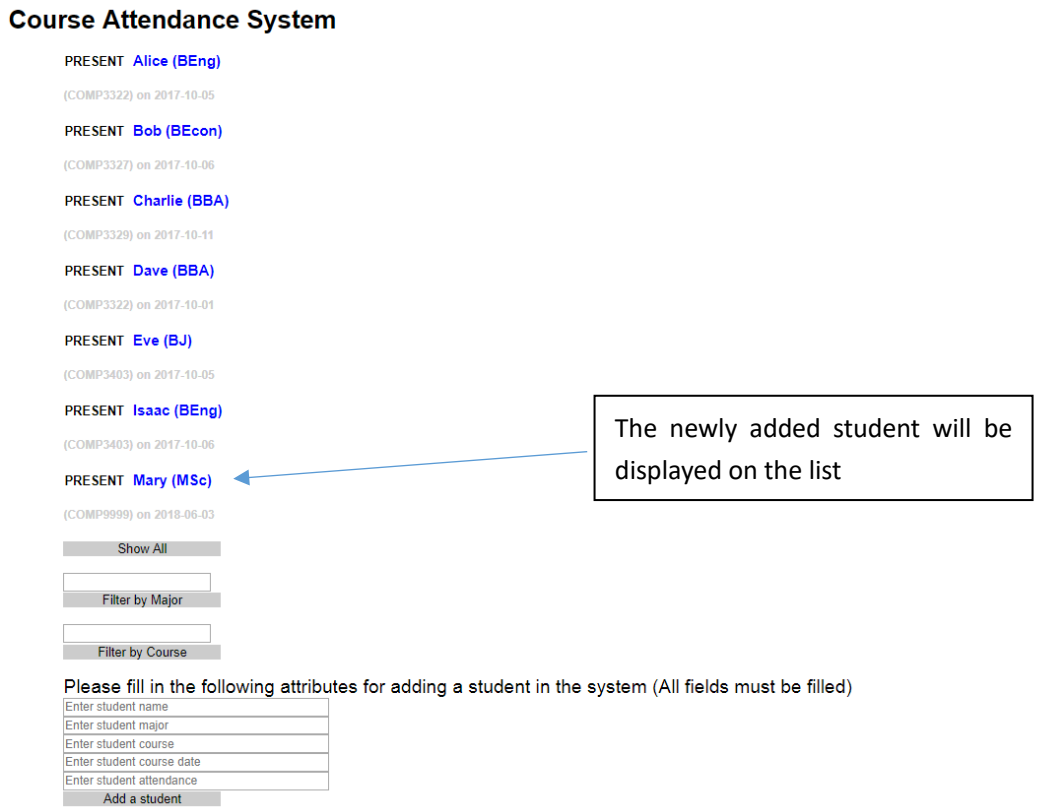


Fig. 5

These functionalities will be implemented through server-side implementation (PHP) and the client-side implementation (HTML, JavaScript, AJAX). We adopt MySQL as the back-end database server to store the attendance information.

Task 1: Prepare the database

We expect you use the course LAMP container set to work on the workshop.

Step1: Log on to phpMyAdmin page (http://localhost:9000/) using the account "dummy" and password "c3322b"

Step2: Select the database ***db3322***.

Step3: Click the SQL button and type the following SQL statements to create a new table and insert new records into the table.

```
create table attendancelist (id int not null primary key auto_increment, studentname varchar(256), major varchar(256), course varchar(256), coursedate DATE, attendOrNot varchar(7) not null default 'PRESENT');

insert into attendancelist values(1,'Alice', 'BEng', 'COMP3322', '2018-10-15', 'PRESENT');
insert into attendancelist values(2,'Bob', 'BEcon', 'COMP3327', '2018-10-16', 'PRESENT');
insert into attendancelist values(3,'Charlie', 'BBA', 'COMP3329', '2018-10-11', 'PRESENT');
insert into attendancelist values(4,'Dave', 'BBA', 'COMP3322', '2018-10-12',  'PRESENT');
insert into attendancelist values(5,'Eve', 'BJ', 'COMP3403', '2018-10-15', 'PRESENT');
insert into attendancelist values(6,'Issac', 'BEng', 'COMP3403', '2018-10-16', 'PRESENT');
```

Task 2: Load entries from the database

Create a folder named "WK4" in the public_html folder of the c3322-WWW container. Download the materials from Moodle- "WK4-PHP.zip" to the WK4 folder. Unzip it and you will find all four files we need in this workshop.

Open **index.html** with a text editor. You will see it contains the following HTML content:

```
<html>
  <head>
  <meta charset="utf-8"/>
    <title>Attendance System</title>
    <link rel="stylesheet" type="text/css" href="style.css">
  </head>
  <body>
    <h1>Course Attendance System</h1>
    <div id="List">
```

```
            <div id="entries">
            </div>
            <div id="button_all" class="buttons">
                <p> Show All</p>
            </div>
            <input id="major" type="text">
            <div class="buttons">
                <p>Filter by Major</p>
            </div>

            <input id="course" type="text">
            <div class="buttons">
                <p>Filter by Course</p>
            </div>

            <font size="5" color="black"> Please fill in the following attributes for adding a new student in the system (All field must be filled) </font> <br/>

            <input id="newname" type="text" placeholder="Enter student name" size="40"> <br>
            <input id="newmajor" type="text" placeholder="Enter student major" size="40">  <br>
            <input id="newcourse" type="text" placeholder="Enter student course" size="40"> <br>
            <input id="newcoursedate" type="text" placeholder="Enter student course date" size="40"> <br>
            <input id="newattendance" type="text" placeholder="Enter student attendance" size="40"> <br>
            <div class="buttons">
                <p> Add a student </p>
            </div>
        </div>
        <script>
          //to be implemented
          //The remaining code in the file is omitted in this document.
        </script>
    </body>
</html>
```

Task 2.1: Implement client-side code for loading all entries

When the page is loaded **or** the "Show All" button (on the page view as in Fig. 1) is clicked, the showAll() function is invoked to load and display all the attendance entries from the attendanceList table in the database, using AJAX.

Step 1: In **index.html,** identify the events that initiate the AJAX communication and implement corresponding event handlers as follows:

1. When the <body> is loaded. Event handler: onload. Add following code inside <script></script>.

```
window.onload = function() {
        showAll();
}
```

2. When Show All is clicked. Event handler: onclick. Add following code inside <script></script>.

```
var btn_all = document.getElementById("button_all");
btn_all.addEventListener('click', showAll);
```

Step 2: Define the event handler (JavaScript function): **showAll()** for both events. In the event handler, create an **XMLHttpRequest** object.

```
<script>
    function showAll(){
        var xmlhttp = new XMLHttpRequest();

        //More code will be implemented in the following steps
    }
</script>
```

Step 3: Define the response actions when the server's response is received.

```
    xmlhttp.onreadystatechange = function(){
        if (xmlhttp.readyState == 4    && xmlhttp.status ==200){
            var mesgs = document.getElementById("entries");
            mesgs.innerHTML = xmlhttp.responseText;
        }
    }
```

Step 4: Define the request that is sent through POST:

```
xmlhttp.open("POST", "queryEntries.php",true);
```

Step 5: Set the request header and send the request.

```
xmlhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");

xmlhttp.send("show=all");
```

Step 6: Hide the "Show All" button from the full list display view (as in Fig. 1), as follows:

```
document.getElementById("button_all").style.display = "none";
```

Task 2.2: Implement client-side code for loading selected entries based

on “Major” or “Course” and adding new student into the attendance system

When the “Filter by Major” button is clicked, the systems loads and displays all the attendance entries whose student major matches the entered text in the textbox above the “Filter by Major” button, using AJAX.

Step 1: In **index.html,** add an event handler on the button “Filter by Major”, so that when clicked, the JavaScript function filterM() will be called.

```
document.getElementById("majBtn").addEventListener('click', filterM);
```

Step 2: Define the event handler (JavaScript function): **filterM()** for the event. In the event handler, create an **XMLHttpRequest** object and define the response actions when the server’s response is received, following steps 2 and 3 in Task 2.1.

```
function filterM() {
    var xmlhttp = new XMLHttpRequest();
    xmlhttp.onreadystatechange = function () {
        if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
            var mesgs = document.getElementById("entries");
            mesgs.innerHTML = xmlhttp.responseText;
        }
    }
    //More code will be implemented in the following steps.
}
```

Define the request that is sent by POST as follows:

```
xmlhttp.open("POST", "queryEntries.php", true);
```

With reference to step 5 in Task 2.1, send the above POST request which contains the value entered by the user in the textbox above “Filter by Major” button using the format of “show=major&value=…”.

```
xmlhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
xmlhttp.send("show=major&bymajor="+document.getElementById("major").value);
```

Step 3: Display the “Show All” button in the page view (as in Fig. 2) through modifying the display properties of the “Show All” button, with reference to step 6 in Task 2.1. In addition,

clear the filter value (e.g. “BBA” in Fig. 2) entered by the user in the textbox above “Filter by Major” button after receiving the AJAX response.

```
xmlhttp.onreadystatechange = function () {
    if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
        var mesgs = document.getElementById("entries");
        mesgs.innerHTML = xmlhttp.responseText;
        document.getElementById("major").value = "";
    }
}
btn_all.style.display = "block";
```

Step 4: Similar to the above steps, implement the client-side code to achieve the functionality that when “Filter by Course” button is clicked, load and display all the attendance entries whose course code matches with the entered text in the textbox above the “Filter by Course” button, using AJAX by implementing codes in filterC().

```
document.getElementById("cBtn").addEventListener('click', filterC);
```

Please note that the “Show All” button should also be displayed when a course attendance list filtered by course is displayed. In addition, clear the filter value (e.g. “COMP3322” in Fig. 3) entered by the user in the textbox above “Filter by Course” button after receiving the AJAX response.

Note: The format of passing data when “Filter by Course” button is clicked is “show=course&value=…”.

```
xmlhttp.send("show=course&bycourse="+document.getElementById("course").value);
```

Step 5: Implement a JavaScript function **addstudent()** which will be called when “Add a student” button is clicked. Notice that this function will only be called when

- All the attributes are filled (include student name, student major, student course, student course date and student attendance)
- The entered student course date format matches with the date format of YYYY-MM-DD, which Y represents year, M represents month and D represents day.
- The entered student attendance is either “PRESENT” or “ABSENT” (case-sensitive).

Otherwise an alert box displaying the message “Check your input date format (YYYY-MM-DD), Attendance record (PRESENT/ABSENT) and whether all fields are filled.” will be shown.

```
function addstudent(){
    let sname = document.getElementById("newname");
```

```
    if (sname.validity.valueMissing) {
        alert("Missing student name!!");
        sname.focus();
          return;
    }
    // other attributes are omitted here, you need to finish it by yourself.
}
```

If user's input satisfies the above conditions, you need to retrieve the values entered by the user in the input boxes and pass all the values through AJAX to **queryEntries.php** using "POST" method.

Remember to clear all the values (include student name, student major, student course, student course date and student attendance) entered by the user in the textboxes above "Add a student" button after receiving the AJAX response.

Note: The format of passing data when "Add a student" button is clicked is "show=add&value=…".

Task 3: Implement server-side code for loading entries from the database and adding new student into the attendance system

In queryEntries.php**,** load the required entries from the database upon client's request.

Step1: Connect to the database "db3322" in the MySQL server running inside c3322-db container. Within the Docker network, our MySQL server is given the name "mydb".

```
<?php

//Connect to docker database
define("DB_HOST", "mydb");
define("USERNAME", "dummy");
define("PASSWORD", "c3322b");
define("DB_NAME", "db3322");

$conn=mysqli_connect(DB_HOST, USERNAME, PASSWORD, DB_NAME) or die('Error! '. mysqli_connect_error($conn));


?>
```

Step 2: Implement necessary server-side codes (includes SQL query) according to the following 4 different client requests:

- When the "Show all" button is clicked ($_POST['show'] == 'all'),
- When the "Filter by Major" button is clicked ($_POST['show'] == 'major'),
- When the "Filter by course" button is clicked ($_POST['show'] == 'course'),
- When the "Add a student" button is clicked ($_POST['show'] =='add').

For example, the following SQL query is applicable in "Show all" button which selects all entries from the attendancelist table.

```
//Construct your SQL query here
$query = ' select * from attendancelist';
```

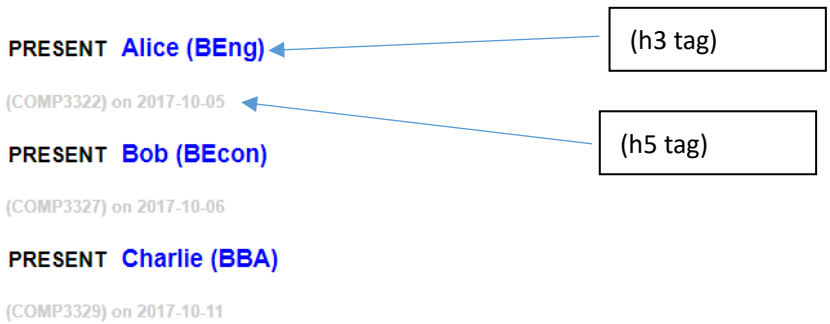
The following code is for executing the SQL query.

```
//Execute SQL query
$result = mysqli_query($conn, $query) or die('Error! '. mysql_error($conn));
```

The following code is for fetching the results and generating HTML code that displays the entries.

```
while($row = mysqli_fetch_array($result)) {
   print "<div id=".$row['id'].">";
   print "<span>".$row['attendOrNot']."</span>";
   print " ..."//Add code to display the entries
   print "</div>";
}
```

It is expected the attendance of the student (PRESENT/ABSENT) is printed within the , followed by printing the student name and student major (with heading level 3 <h3>), as well as the course code and date (with heading level 5 <h5>).



Upon finishing this task, it is expected the system can achieve the following functions:

1. Display all the students' records in the attendance system when the page is loaded (as Fig. 1 shows).
2. Display part of the students' records that are either filtered by students' major or course

when the “Filter by Major” or “Filter by Course” button is clicked (as Fig. 2 and Fig. 3 show).

3. Display all the students’ record in the attendance system when the “Show All” button is clicked (the “Show All” button is only displayed when filtered list is displayed).
4. Allows users to add a new student and all the students’ records (including the newly added one) would be displayed when the “Add a student” button is clicked (as Fig. 4 and Fig. 5 show).

Task 4: Toggle PRESENT/ABSENT for changing attendance record

Tapping the word PRESENT or ABSENT in each entry on the page will change the status from one to the other, and initiate AJAX communication to the server for changing the attribute “attendOrNot” of this entry in the database table, without reloading the entire web page.

Task 4.1

Add the “onclick” event handler to the **** element of each display entry (do this by adding the code to **queryEntries.php**). Implement the JavaScript function **changeState(elem)** as the event handler in index.html. With reference to the steps in Task 2.1, implement the AJAX code for asynchronous communication with the server (send the request through “**GET**”).

```
function changeState(elem) {
    var oldValue = elem.innerHTML;
    var newvalue;
    var itemID = elem.parentNode.getAttribute('id');
    if (oldValue == 'PRESENT') {
        newvalue = 'ABSENT';
    } else {
        newvalue = 'PRESENT';
    }
    var xmlhttp = new XMLHttpRequest();
    // Add AJAX code here.
```

Hint: the request that sent through GET:

```
xmlhttp.open("GET", "updateState.php?id="+itemID+"&value="+newvalue,true);
```

Task 4.2

Under **updateState.php,** implement the server-side logic in order to change the “attendOrNot” attribute of the corresponding entry in the database, as well as print out the updated attendance status. The following code is given for your reference.

```
define("DB_HOST", "mydb");
define("USERNAME", "dummy");
```

```
define("PASSWORD", "c3322b");
define("DB_NAME", "db3322");
$conn=mysqli_connect(DB_HOST, USERNAME, PASSWORD, DB_NAME) or die('Error! '. mysqli_connect_error($conn));

// Implement the code here.

mysqli_close($conn);
```

Upon finishing this task, by clicking on the word PRESENT or ABSENT in each display entry on the page, you can change the attendance status of the student by updating the database as well as showing the new attendance status without re-loading the page.

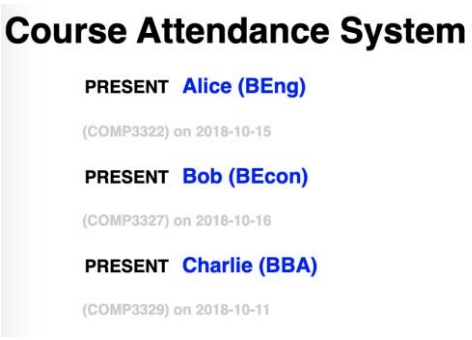


Fig. 6 Before click "PRESENT" before Alice

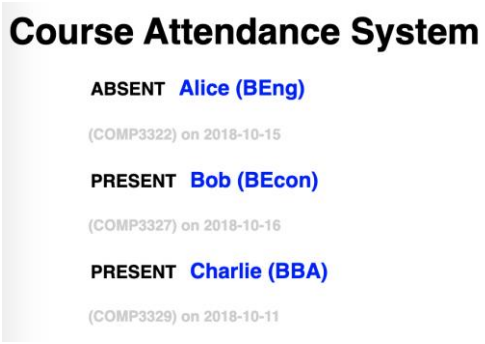


Fig. 7 After click "PRESENT" before Alice

Test your page

Browse and test your Web page at:
http://localhost:9080/WK4/