COMP519 Practical 17
Ajax

Introduction

- This worksheet contains further exercises that are intended to familiarise you with Ajax [2, 5].

  While you work through the exercises below compare your results with those of your fellow students and ask for help and comments if required.
- It is assumed that you have completed the exercises in the previous two COMP519 practicals, that you have an account with the departmental MySQL DBMS, and that you have created a `meetings` table on it.
- You might proceed more quickly if you cut-and-paste code from this PDF file. Note that a cut-and-paste operation may introduce extra spaces into your code. It is important that those are removed and that your code exactly matches that shown in this worksheet.
- The exercises and instructions in this worksheet assume that you use the Department's Linux systems to experiment with Ajax.
- To keep things simple, we will just use a text editor, a terminal, and a web browser. You can use whatever text editor and web browser you are most familiar or comfortable with.

Exercises

1. In the lectures we have seen how we can use Ajax to create the options of a drop-down menu. In the following we want to use the same technique to create options for datalists [3].

   a. Using a text editor, create a file `ajax17A.html` in your `public_html` directory with the following content.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ajax 17A</title>
    <script>
function createDatalists() {
  console.log('Starting createDatalists...');
  var request = new XMLHttpRequest()
  request.open('POST','getOptions.php',true)
  request.onreadystatechange = function() {
    console.log(this.readyState + ": " + this.responseText)
    if (this.readyState == 4 && this.status == 200) {
       addDatalistOptions(this.responseText)
  } }
  console.log('Sending XMLHttpRequest...');
  request.send()
}
function addDatalistOptions(jsonData) {
  console.log('Starting addSelectOptions...')
```

```
  console.log('Received data '+jsonData)
  entries = JSON.parse(jsonData)
  // Code for Exercise 1e
}
function insert() {
  console.log('Starting insert...')
  clearResponse()
  var formElem = document.getElementById('form1')
  var data    = new FormData(formElem)
  logFormData(data)
  // Code for Exercise 2a
  request.onreadystatechange = function() {
    console.log(this.readyState + ": " + this.responseText + " " +
                this.status)
    if (this.readyState == 4) {
      // Code for Exercise 2b
    }
  }
  console.log('Sending XMLHttpRequest...');
  request.send(data)
}
function del() {
  console.log('Starting delete...')
  var request = new XMLHttpRequest()
  request.open('DELETE','delete.php?slot=' + data.get('slot'),true)
}
function query() {
  console.log('Starting query...')
  var request = new XMLHttpRequest()
  var url = "query.php" + "?name="+encodeURIComponent(data.get('name'))
  request.open('GET',url,true)
}
function logFormData(formData) {
  for(var pair of formData.entries())
    console.log('FormData['+pair[0] + '=' + pair[1] + ']')
}
function clearResponse() {
  r = document.getElementById('response')
  r.innerHTML = ""
}
createDatalists()
    </script>
  </head>
  <body>
    <h1>Ajax</h1>
    <form name="form1" id="form1" method="post">
      Slot: <input type="number" name="slot" min="1" max="60"><br>
      Name: <input type="text" name="name" list="names" size="60"><br>
      <datalist id="names">
      </datalist>
```

```
      Email: <input type="text" name="email" list="emails" size="60"><br>
      <datalist id="emails">
      </datalist>
    </form>
    <button onclick="insert()">Insert into DB</button>
    <button onclick="del()">Delete from DB</button>
    <button onclick="query()">Query DB</button>
    <p>Response</p>
    <div id="response"></div>
  </body>
</html>
```

b. You can see in `ajax17A.html` that the XMLHttpRequest `request` uses a PHP script named `getOptions.php`. Using a text editor, create such a file in your `public_html` directory, with the following content:

```
<?php
require_once 'php17db.php';
header('Content-Type: application/json; charset=utf-8');
try {
  $pdo    = new PDO($dsn,$db_username,$db_password,$opt);
  $tpl    = "select email,name from meetings";
  $stm    = $pdo->prepare($tpl);
  $success = $stm->execute();
  $data   = $stm->fetchAll();
  echo json_encode($data);
} catch (PDOException $e) {
  echo json_encode(array('error' => $e));
}
?>
```

c. The script `getOptions.php` in turn uses a file `php17db.php`. Using a text editor, create such a file in your `public_html` directory, with the following content:

```
<?php
$db_hostname = "studdb.csc.liv.ac.uk";
$db_database = "<user>";
$db_username = "<user>";
$db_password = "------";
$db_charset = "utf8mb4";

$dsn = "mysql:host=$db_hostname;dbname=$db_database;charset=$db_charset";
$opt = array(
    PDO::ATTR_ERRMODE          => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
    PDO::ATTR_EMULATE_PREPARES => false
);
?>
```

Replace *<user>* with your University (MWS) username and fill in your password for the MySQL DBMS.

d. We are ready to conduct a first test. In a terminal, execute the PHP script `getOptions.php` using the command

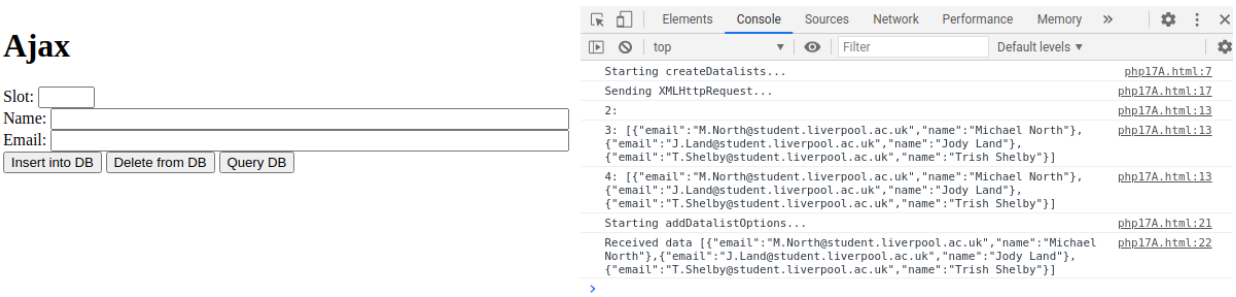

Figure 1: Web page `ajax17A.html`

```
php $HOME/public_html/getOptions.php
```

The output should be something similar to the following:

```
[{"email":"M.North@student.liverpool.ac.uk","name":"Michael North"},
 {"email":"J.Land@student.liverpool.ac.uk","name":"Jody Land"},
 {"email":"T.Shelby@student.liverpool.ac.uk","name":"Trish Shelby"}]
```

e. Next, open a web browser and access the url

`https://student.csc.liv.ac.uk/~<user>/ajax17A.html`

where *<user>* should be replaced by your University (MWS) username. In the web browser open the console. Figure 1 shows you what you should see. In particular, in the console you see the diagnostic output produced by the JavaScript code. It tells you that the functions `createDatalists` and `addDatalistOptions` have been executed. You should also again see the JSON data. If the output you see deviates from that, try to find out why and fix the mistakes in your code.

f. So far, the function `addDatalistOptions` does nothing more than produce diagnostic output and convert JSON data into an array of objects with name and email attributes. Add code to `addDatalistOptions()` that (i) extends the HTML datalist element with id '`emails`' with an HTML option element for each email in the email/name objects in the array `entries` and (ii) extends the HTML datalist element with id '`names`' with an option for each name in the email/name objects in the array `entries`.

g. Reload the page in your browser and check that your code is working. If it does, then once you click on the input field for names you should see a list of names from the `meetings` database table. Analogously, if you click on the input field for email addresses you should see a list of email addresses from the `meetings` database table.

2. Our web page also includes three buttons that so far don't do much. We first get the 'Insert into DB' button working.

a. In analogy to the code in the function `createDatalists` extend the code of the function `insert` at the point 'Code for Exercise 2a' so that the variable `request` stores an `XMLHttpRequest` object that is initialised to send a `POST` request to a PHP script `insert.php` and does not wait for a response.

b. Extend the code of the function `insert` at the point 'Code for Exercise 2b' so that in the `div` element with id '`response`' we will see both the HTTP response body and the HTTP response code for the request. Reload `ajax17A.html` in the web browser.

c. Using a text editor, create a file `insert.php` in your `public_html` directory with the following content.

```php
<?php
require_once 'php17db.php';
// Exercise 2d
// Modify the condition below
if (TRUE) {
  try {
    // Code for Exercise 2e
    if ($success) {
      $status  = 201;
      $response = $_REQUEST;
    } else {
      $status  = 400;
      $response = ["error" => "INSERT query failed"];
    }
  } catch (PDOException $e) {
    $response = ["error" => $e->getMessage()];
    $status = 400;
  }
} else {
  $response = $_REQUEST;
  $status  = 400;
}
header("Content-Type: application/json",TRUE,$status);
echo json_encode($response);
?>
```

Check that you have no syntax errors by executing `insert.php` on the command line.

d. Modify the condition of the conditional statement in `insert.php` so that its then-branch is only executed if `$_REQUEST['slot']`, `$_REQUEST['name']`, and `$_REQUEST['email']` are set and contain non-empty strings. Hint: You've done the same in Practical 16 Exercise 3c.

e. Extend the code of `insert.php` at the point 'Code for Exercise 2e' so that a new entry in the `meetings` table is created with the values `$_REQUEST['slot']`, `$_REQUEST['name']`, and `$_REQUEST['email']`. The code should set the variable `$success` to TRUE if this operation has been successful and to FALSE otherwise. Hint: Use a prepared statement and, again, you may have already done this for Practical 16 Exercise 3c.

f. Test `insert.php`. You can do so by adding code to the start of the script that assigns values to `$_REQUEST['slot']`, `$_REQUEST['name']`, and `$_REQUEST['email']`; then execute the script on the command line. Once you are sure that the script is working, remove the assignments again.

g. Now enter the following into the form on `ajax17A.html`, then click on the 'Insert into DB' button.

Slot	Name	Email
11	Ben Booker	bb@protonmail.com

If everything works correctly, then you should see something like

```
{"slot":"11","name":"Ben Booker","email":"bb@protonmail.com"} 201
```

in the 'response' div element below the buttons. In the console of the web browser you should also see some diagnostic output with the data that was sent to `insert.php` and the data that was sent back in response.

If this does not happen, try to find the mistakes in your code and fix them.

h. Without changing any of the inputs, just click on the 'Insert into DB' button again. As the `slot` attribute is the primary key of the `meetings` table, the database operation should fail and `insert.php` should send back JSON data with an error message plus an HTTP response code 400. You should see something like the following in the 'response' div element.

```
{"error":"SQLSTATE[23000]: Integrity constraint violation: 1062
Duplicate entry '11' for key 'PRIMARY'"} 400
```

In the console you should also see a response with 400 BAD REQUEST highlighted.

If this does not happen, try to find the mistakes in your code and fix them.

3. Next, let's turn our attention to the 'Delete from DB' button. The intended use is that the user enters a slot number into the form then clicks on that button and our application removes any entry from the `meetings` table with matching slot number (there should be at most one such entry).

   In contrast to the `insert` function that used a `POST` HTTP request, we want to use a `DELETE` HTTP request instead, mainly to demonstrate that this is possible. However, `DELETE` HTTP requests do not have a request body, so the slot number needs to be passed via the query part of the URL we use. The code for that is already present in the `del` function.

   a. In analogy to the code in the function `insert`, complete the code for the function `del`. Do not change any of the code already present, just add code appropriately. Reload `ajax17A.html` in the web browser.

   b. In analogy to `insert.php`, create a PHP script `delete.php` in your `public_html` directory. The script accepts a slot number via `$_REQUEST['slot']` and attempts to delete entries with that slot number from the `meetings` table.

      - If at least one entry has been deleted, then the script should respond with an HTTP response code 204 and should send the contents of `$_REQUEST` back (that's for debugging purposes, to see whether the right slot number was deleted).
      - If no entry has been deleted, then the script should respond with an HTTP response code 400 and should send back an error message that deletion of `$_REQUEST['slot']` failed.

      Hint: Use a prepared statement and, again, you may have already implemented a similar operation for Practical 16 Exercise 3d.

   c. Test `delete.php`. You can do so by adding code to the start of the script that assigns a value to `$_REQUEST['slot']`. Once you are convinced that the script is working remove that assignment again.

   d. Now enter a slot number into the form on `ajax17A.html`. First try one that should occur in the `meetings` table, say, 11, then click on the 'Delete from DB' button. In the 'response' div element you should see a 204 HTTP response code. If so, leave the slot number unchanged and click on the 'Delete from DB' button again. As the entry with slot number 11 has already been deleted, there should now be an error message and a 400 HTTP response code. (Note that once we cover web services, we'll learn that the correct response code should again be 204 as the response code is meant to indicate that the slot number is not present in the `meetings` table.)

If the application does not show the desired behaviour, try to find the mistakes in your code and fix them.

4. Finally, we deal with the 'Query DB' button that should trigger a function that allows us to query the database using regular expressions by entering such an expression into the 'name' input field.

   Here we want to use a GET HTTP request, again, mainly to demonstrate that this is possible. Just like a DELETE HTTP request, a GET HTTP request has no request body, so the regular expression needs to be passed via the query part of the URL we use. This time we have the added complication that since regular expressions can contain a wide variety of characters, we need to apply URL encoding to it before it can be added to the URL. The code for this is already present in the query function.

   a. In analogy to the code in the function del, complete the code for the function query. Do not change any of the code already present, just add code appropriately. Reload ajax17A.html in the web browser.

   b. In analogy to del.php, create a PHP script query.php in your public_html directory. The script accepts a regular expression via $_REQUEST['name'] and attempts to select entries from the meetings table that match that regular expression.

      The list of matching entries should be returned in JSON. Hints:

      - You may have constructed a similar query for Practical 16 Exercise 4a. If not, you might have to refer to [1].
      - Use fetchAll() to transfer the query result into an array (of arrays). Then use json_encode to encode that array in JSON. See getOptions.php how it was done there.

   c. Test query.php. You can do so by adding code to the start of the script that assigns a value to $_REQUEST['name'], say, '.*'. This regular expression should match every entry in the meetings table. Once you are convinced that the script is working remove that assignment again.

   d. Now enter a regular expression into the name field of the form on ajax17A.html. You can again start with .*. In the 'response' div element you should see something like the following

```
[{"slot":1,"name":"Michael North",
  "email":"M.North@student.liverpool.ac.uk"},
 {"slot":5,"name":"Jody Land","email":"J.Land@student.liverpool.ac.uk"},
 {"slot":7,"name":"Trish Shelby",
  "email":"T.Shelby@student.liverpool.ac.uk"}] 200
```

      If the application does not show the desired behaviour, try to find the mistakes in your code and fix them.

References

[1] Oracle Corporation. *MySQL 8.0 Reference Manual: Regular Expressions*. dev.mysql. 2020. URL: https://dev.mysql.com/doc/refman/8.0/en/regexp.html (accessed 27 November 2021).

[2] Mozilla and individual contributors. *Ajax*. MDN Web Docs. 21 October 2021. URL: https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX (accessed 27 November 2021).

[3] Mozilla and individual contributors. *<datalist>: The HTML Data List element*. MDN Web Docs. 03 October 2021. URL: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/datalist (accessed 27 November 2021).

[4] Mozilla and individual contributors. *FormData*. MDN Web Docs. 17 September 2021. URL: https://developer.mozilla.org/en-US/docs/Web/API/FormData (accessed 28 November 2021).

[5] Mozilla and individual contributors. *XMLHttpRequest*. MDN Web Docs. 14 September 2021. URL: https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest (accessed 27 November 2021).