COMP519 Practical 19
REST (2)

Introduction

- This worksheet contains exercises relating to the implementation of a RESTful web service using PHP.

  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 COMP519 Practicals 15, 17, and 18. You should have an account with the departmental MySQL DBMS and you have created a `meetings` table on it. You have created PHP scripts `php17db.php`, `insert.php`, `delete.php` and `query.php` that interact with the MySQL DBMS. You have created a sub-directory `$HOME/public_html/rw` and a file `.htaccess` in it. You are familiar with the use of Postman or </> RESTED.

- 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 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. We want to develop a simple RESTful web service that deals with meetings.

   (i) All existing meetings can be retrieved via a GET request to
   `https://student.csc.liv.ac.uk/~`*`<user>`*`/rw/meetings`

   (ii) All existing meetings involving someone whose name matches a regular expression *`regex`* can be retrieved via a GET request to
   `https://student.csc.liv.ac.uk/~`*`<user>`*`/rw/meetings?name=`*`regex`*

   (iii) Information on a specific meeting with slot *`number`* can be retrieved via a GET request to `https://student.csc.liv.ac.uk/~`*`<user>`*`/rw/meetings/`*`number`*

   (iv) A new meeting can be created by a POST request on
   `https://student.csc.liv.ac.uk/~`*`<user>`*`/rw/meetings`
   with JSON data `{"slot":`*`number`*`,"name":"`*`Name`*`","email":"`*`Email`*`"}`.

   (v) An existing meeting can be deleted by a DELETE request on
   `https://student.csc.liv.ac.uk/~`*`<user>`*`/rw/meetings/`*`number`*

2. In the last practical you should have created a directory `$HOME/public_html/rw` with a file `.htaccess`. The rewrite rule in the file should currently redirect some HTTP requests for sub-directories of `rw` to a PHP script `file3.php`.

   Change the rewrite rule and rewrite condition so that instead of `file3.php` it redirects requests to `REST.php` and also change the regular expression so that all requests are redirected.

3. Let us create the basics of our RESTful web service.

   a. Using a text editor, create a file `Database.php` in the directory `$HOME/public_html/rw` with the following content

```
<?php
class Database {
  private $host   =
  private $user   =
  private $passwd =
  private $database =
  public $conn;

  public function __construct() {
    $opt = array(PDO::ATTR_ERRMODE        => PDO::ERRMODE_EXCEPTION,
                 PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
                 PDO::ATTR_EMULATE_PREPARES => false);
    $this->conn = null;
    try {
      $this->conn = new PDO('mysql:host=' . $this->host . ';dbname=' .
                            $this->database . ';charset=utf8mb4',
                            $this->user,$this->passwd,$opt);
    } catch (PDOException $e) {
      throw new Exception($e->getMessage(),500);
    }
  }
}
```

   b. Complete the code by assigning the correct values to the private properties of `Database` so that constructing a `Database` object creates a connection to your database on the departmental MySQL DBMS.

   c. Test your code by adding a line

```
$db = new Database();
```

   at the end of `Database.php`, then execute the file on the command line. If working correctly you won't see an exception. Remove the additional line of code again.

   d. Using a text editor, create a file `Model.php` in the directory `$HOME/public_html/rw` with the following content

```
<?php
require_once 'Database.php';

class Meeting {
  // Private properties do not appear in the JSON encoding
  // of an object.
  private $conn;
  private static $table = 'meetings';
  private $parts = ['slot','name','email'];

  public $slot, $name, $email, $_links;

  public function __construct($db) {
    $this->conn = $db->conn;
  }

  public function set($source) {
    if (is_object($source))
      $source = (array)$source;
    foreach ($source as $key=>$value)
      if (in_array($key,$this->parts))
        $this->$key = $value;
      else
        throw new Exception("$key not an attribute of meetings",400);
  }

  public function validate() {
    foreach ($this->parts as $key)
      if (is_null($this->$key))
        return FALSE;
    return TRUE;
  }

  public function __toString() {
    return json_encode($this,
                       JSON_UNESCAPED_UNICODE|JSON_UNESCAPED_SLASHES);
  }

  public function setLinks($slot) { }
  public function store() { }
  public function read() { }
  public function delete() { }
  public static function readAll($db,$regexp) { }
}
?>
```

e. Using a text editor, create a file `REST.php` in the directory `$HOME/public_html/rw` with the following content

```
<?php
require_once('Database.php');
require_once('Model.php');
$db = new Database();

set_exception_handler(function ($e) {
  $code = $e-> getCode() ?: 400;
  header ("Content-Type: application/json", NULL, $code);
  echo json_encode(["error" => $e-> getMessage()]);
  exit ;
});

$method   = $_SERVER['REQUEST_METHOD'];
$resource = explode('/', $_REQUEST['resource']);
$regexp   = isset($_REQUEST['name']) ? $_REQUEST['name'] : NULL;
```

```
$data     = json_decode(file_get_contents('php://input'),TRUE);

switch($method) {
  case 'GET':
    [$data,$status] = readData($db,$resource,$regexp);
    break;
  case 'POST':
    [$data,$status] = createData($db,$resource,$data);
    break;
  case 'DELETE':
    [$data,$status] = deleteData($db,$resource);
    break;
  default:
    throw new Exception('Method Not Supported', 405);
}
header("Content-Type: application/json",TRUE,$status);
echo json_encode($data,JSON_UNESCAPED_UNICODE|JSON_UNESCAPED_SLASHES);

function createData($db,$resource,$data) {
  return [$data,201]
}
?>
```

f. Now the basics of our RESTful web service are in place and we can perform a first test. In Postman or </> `RESTED` set up a POST request to

`https://student.csc.liv.ac.uk/~`*`<user>`*`/rw/meetings`

where you have to replace *`<user>`* by your University (MWS) username, and with JSON data

```
{"slot":19,"name":"Lois Lane","email":"l.lane@dp.com"}
```

Then send the request. The response should have HTTP response code 201 and the response body contains the same JSON data as was sent out. If this does not happen, try to find the problem and fix it.

4. Let us first focus on operations (i) to (iii) of our RESTful web service.

   a. In analogy to the `setLinks()` function for `Address` objects shown in the lecture notes, complete the code for the `setLinks()` function in the `Meeting` class, so that it adds HATEOAS links. The HTTP requests that can be performed for a resource `meetings/`*`slot`* are GET and DELETE.

   b. In analogy to the `read()` function for `Address` objects shown in the lecture notes, complete the code for the `read` function in the `Meeting` class. You should assume that the `slot` property already has a value. Also, use the `setLinks()` method to add HATEOAS links.

      You should cater for the possibility that for the value of the `slot` property there is no entry in the database (`fetch()` will return `FALSE`). You should throw an exception with code 404 (`NOT FOUND`) in this case.

   c. Add the following code at the end of `REST.php`:

```
function readData($db,$resource,$regexp) {
  if ((count($resource) == 1) &&
     ($resource[0] = 'meetings')) {
    return Meeting::readAll($db,$regexp);
  } elseif ((count($resource) == 2) &&
          ($resource[0] = 'meetings')) {
    if (preg_match('/^\d+$/',$resource[1])) {
      // Code for Exercise 4d
      return [$m,200];
    } else {
      throw new Exception("Invalid slot ".$resource[1],400);
    }
  } else {
      throw new Exception("Bad request ".implode('/',$resource),400);
} }
```

d. At the point 'Code for Exercise 4d' add code that create a new `Meeting` object with a value for the `slot` attribute provided by `$resource[1]`, reads the remaining attributes from the database, and sets the HATEOAS links.

e. Test your solution for Exercise 4d. In Postman or </> `RESTED` send a GET request to

   `https://student.csc.liv.ac.uk/~`*`<user>`*`/rw/meetings/`*`slot`*

   where you have to replace *`<user>`* by your University (MWS) username and *`slot`* by a slot number that you know occurs in your `meetings` table. The web service should respond with the corresponding entry retrieved from the database in JSON and a response code 200.

   Next send a GET request to

   `https://student.csc.liv.ac.uk/~`*`<user>`*`/rw/meetings/`*`slot`*

   where you have to replace *`<user>`* by your University (MWS) username and *`slot`* by a slot number that you know does *not* occur in your `meetings` table. The web service should send back a response code 404 and nothing else.

f. Add the following code to the function `readAll()`:

```
    $allMeetings = [];
    if (is_null($regexp)) {
      // Retrieve all entries
    } else {
      // Retrieve all entries where the name matches $regexp
    }
    foreach ($stmt as $row) {
      // Construct a new Meeting object $m based on the
      // information in the array $row
      array_push($allMeetings,$m);
    }
    return [$allMeetings,200];
```

   Then try to complete the code.

g. Test your solution for Exercise 4f. In Postman or </> `RESTED` send a GET request to

   `https://student.csc.liv.ac.uk/~`*`<user>`*`/rw/meetings`

   where you have to replace *`<user>`* by your University (MWS) username. The web service should respond with a list of all entries in the database in JSON and a response code 200.

5. Let us next deal with operation (iv) (adding a new meeting)

a. In analogy to the `store()` function for `Address` objects shown in the lecture notes, complete the code for the `store` function in the `Meeting` class.

b. Extend the `createData` function in `REST.php` with a conditional statement that checks whether the array `$resource` contains exactly one element and that element is equal to `'meetings'`.

   In the then-branch create a new `Meeting` object using the data in `$data`, store it in the database, then return it together with a 201 response code. Have a look at the `createAddress` and `createStudent` function in the lecture notes to see how to do that. In the else-branch throw an exception for a bad request.

c. Test your solution. In Postman or </> `RESTED` set up a POST request to

   `https://student.csc.liv.ac.uk/~<user>/rw/meetings`

   where you have to replace *<user>* by your University (MWS) username, and with JSON data

```
{"slot":19,"name":"Lois Lane","email":"l.lane@dp.com"}
```

   Then send the request. The response should have HTTP response code 201 and the response body contains the same JSON data as was sent out.

d. See what happens if you provide incomplete information. In Postman or </> `RESTED` set up a POST request to

   `https://student.csc.liv.ac.uk/~<user>/rw/meetings`

   where you have to replace *<user>* by your University (MWS) username, and with JSON data

```
{"slot":21,"name":"Clark Kent"}
```

   You should receive an error response. If not, then your code is probably incomplete. Try to find out what is missing.

6. It remains to deal with (v) (removing a meeting). The way this has to be implemented is quite similar to the operation of retrieving a specific meeting, only that we now want to delete that meeting.

   a. Following this analogy, complete the code for the `delete` function in the `Meeting` class. You should assume that the `slot` property already has a value.

      You should cater for the possibility that for the value of the `slot` property there is no entry in the database. You need to detect that nothing has been delete from the database and throw an exception with code 404 (`NOT FOUND`) in this case.

   b. Add the following code at the end of `REST.php`:

```
function deleteData($db,$resource) {
  if ((count($resource) == 2) &&
      ($resource[0] = 'students')) {
    if (preg_match('/^\d+$/',$resource[1])) {
      // Code for Exercise 6c
    } else {
      throw new Exception("Invalid slot ".$resource[1],404);
    }
  } else {
    throw new Exception("Bad request ".implode('/',$resource),400);
  }
}
```

c. Complete the code of `deleteData` at the point 'Code for Exercise 6c'. At the end of the then-branch you need to return an array with data and response code. It's up to you whether you want to return the deleted enty and code 200, or you return an empty array and code 204.

d. Test your solution for Exercise 6c. In Postman or </> `RESTED` send a DELETE request to

   `https://student.csc.liv.ac.uk/~<user>/rw/meetings/slot`

   where you have to replace *<user>* by your University (MWS) username and *slot* by a slot number that you know occurs in your `meetings` table, e.g., 19. The web service should send a response code 200 or 204 back, according to your solution to Exercise 6c.

   Next send a DELETE request to

   `https://student.csc.liv.ac.uk/~<user>/rw/meetings/slot`

   where you have to replace *<user>* by your University (MWS) username and *slot* by a slot number that you know does *not* occur in your `meetings` table. The web service should send back a response code 404.