COMP519 Web Programming

Lecture 29: REST

Handouts

Ullrich Hustadt

Department of Computer Science
School of Electrical Engineering, Electronics, and Computer Science
University of Liverpool

Contents

Useful PHP Functions

`Exception([string msg = ""[, int code = 0]])`

Creates an exception with exception message *msg* and exception code *cd*

```
throw new Exception('Method Not Supported', 405);
```

`set_exception_handler(exceptionHandler)`

- Sets the default exception handler if an exception is not caught within a try/catch block
- *exceptionHandler* should be a function that accepts an exception as an argument
- Execution will stop after the call of *exceptionHandler* is completed

```
function exceptionHandler($excpt) {
  echo "Uncaught exception: ", $excpt->getMessage(), "\n";
}
set_exception_handler('exceptionHandler');
throw new Exception('Spurious Exception');
echo "This code is not executed\n";
```

Useful PHP functions

php://*input*

A read-only stream that allows to read raw data from the request body

```
$params = json_decode(file_get_contents('php://input'),
                      true);
```

Assuming that the request body contains JSON encoded data, read the whole of php://*input* and turn it into an associative array

explode(string *delimiter*, string *str*[, int *limit*])

Returns an array of strings, with a maximum of *limit* elements, each of which is a substring of *str* formed by splitting it on boundaries formed by the string *delimiter*

```
print_r(explode('/','this/is/a/filepath'));
```

```
Array
( [0] => 'this'  [1] => 'is'  [2] => 'a'  [3] => 'filepath')
```

Useful PHP functions

`header(string hStr[, bool repl = TRUE[, int httpRspCd]])`

– Send a raw HTTP header including *hStr* and HTTP response code *httpRspCd*
– Replace a previous similar header if *repl* is `TRUE`, otherwise add

```
header('Location: http://www.example.com/');
```

Send this header back to the browser with a 302 (REDIRECT) status code to the browser, telling to browser to visit the URL indicated

```
header('Content-Type: application/json');
```

Add a header entry that indicates the request/response body contains JSON encoded data

`http_response_code([int httpRspCd])`

Returns the previous HTTP response code and sets it to *httpRspCd* if that argument is provided

```
http_response_code(201)
```

Sets the HTTP response code to 201 (CREATED)

REST.php Outline

```
<?php
require_once('Database.php');
require_once('Model.php');

$db       = new Database();
$method   = $_SERVER['REQUEST_METHOD'];
$resource = explode('/', $_REQUEST['resource']);
switch($method) {
  case 'GET':
    $data = readData($db,$resource);
    break;
  case 'PUT':
  case 'POST':
    $data = createData($db,$method,$resource);
    break;
  case 'DELETE':
    $data = deleteData($db,$resource);
    break;
  default:
    throw new Exception('Method␣Not␣Supported', 405);
}
header("Content-Type:␣application/json");
echo json_encode($data);
```

REST.php: Exception Handling

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

Database.php: Database Class (1)

```
class Database {
  private $host     = "studdb.csc.liv.ac.uk";
  private $user     = "sgfsurn";
  private $passwd   = "-------";
  private $database = "sgfsurn";
  public  $conn;

  public function __construct() {
    // we use the same options as usual
    $opt = array(  ... );
    $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);
} } }
```

Model.php: Address Class (1)

```
class Address {
  private $conn;
  private $table = 'addresses';

  public $id, $line1, $line2, $city, $postCode, $country;

  // The constructor only set the database connection and
  // possible the identifier id for an address object
  public function __construct($db,$id = NULL) {
    $this->conn = $db;
    $this->id   = $id;
  }
```

Model.php: Address Class (2)

```
// set() populates the public properties of an address
// values can be provided individually or as array
// or as another object
public function set($arg1, $line2 = NULL, $city = NULL,
                    $postCode = NULL, $country = NULL) {
  if (is_string($arg1)) {
    $this->line1    = $arg1;
    $this->line2    = $line2;
    $this->city     = $city;
    $this->postCode = $postCode;
    $this->country  = $country;
  } elseif (is_array($arg1)) {
    foreach ($arg1 as $key=>$value) {
      $this->$key = $value;
    }
  } else {
    foreach (get_object_vars($arg1) as $key=>$value) {
      $this->$key = $value;
} }
```

Model.php: Address Class (2)

```
  // create() stores an address in the database
  // in the process a unique id is generated and returned
  // ideally we woud avoid creating duplicate addresses
  public function create() {
    $query = 'INSERT INTO ' . $this->table .
            '(id,  line1, line2, city, postCode, country)
      VALUES (NULL,:line1,:line2,:city,:postCode,:country)';

    $stmt = $this->conn->prepare($query);
    $stmt->execute(array($this->line1,$this->line2,
               $this->city,$this->postCode,$this->country));
    $this->id = $this->conn->lastInsertId();
    return $this->id;
  }
} // end of class Address

// Standard sequence to create and store an address
// $adr1 = new Address($db->conn);
// $adr1->set('6 Queens Road','','Harrow','HA14 6TP','UK');
// $adr1Id = $adr1->create();
```

Model.php: Address Class (3)

```
// read() retrieves an address from the database
// $this->id must have been set when read() is called
public function read() {
  $query = 'SELECT * FROM ' . $this->table .
           ' WHERE id=:id';

  // Prepare and execute statement
  $stmt  = $this->conn->prepare($query);
  $stmt->execute(array($this->id));
  // Fetch the single row that the query returns
  $row   = $stmt->fetch();
  // Transfer database data into properties
  foreach($row as $key=>$value) {
    $this->$key = $value;
  }
}
```

Model.php: Student Class (1)

```
class Student {
  private $conn;
  private $table = 'students';

  // Student Properties
  public  $id, $sname, $fname, $prog;

  // $_links will provide URIs for addresses
  public $_links;

  // $tAddrId: Unique id of the term time address
  // $pAddrId: Unique id of the permanent address
  // These are private so that they do not occur
  // in the JSON encoding of a Student object
  private $tAddrId, $pAddrId;
```

Model.php: Student Class (2)

```php
public function __construct($db,$id = null) {
  $this->conn = $db;
  $this->id   = $id;
  /*
    Hypermedia as the engine of application state
    (HATEOAS)
    Having accessed an initial URI for a RESTful web
    service, a client should be able to use
    server-provided links to dynamically discover all
    the available services
    -> Address data will not be part of a student's
       record that our web service returns, but links
       to that data
  */
  $this->_links =
         array( (object)array('rel' => 'tAddr',
                              'href' => '/termTime' ),
                (object)array('rel' => 'pAddr',
                              'href' => '/permanent') );
}
```

Model.php: Student Class (3)

```php
  public function set($arg1     = NULL,
                      $fname    = NULL, $prog     = NULL,
                      $tAddrId = NULL, $pAddrId = NULL) {
    if (is_string($arg1)) {
      $this->sname   = $arg1;
      $this->fname   = $fname;
      $this->prog    = $prog;
      $this->tAddrId = $tAddrId;
      $this->pAddrId = $pAddrId;
    } elseif (is_array($arg1)) {
      foreach ($arg1 as $key=>$value) {
        $this->$key = $value;
      }
    } else {
      foreach (get_object_vars($arg1) as $key=>$value) {
        $this->$key = $value;
  } } }
```

Model.php: Student Class (4)

```
  public function create($id = NULL) {
    // We do not want to use autoincrement for student ids
    if (!$id) {
      $maxIdArr = $this->conn->query("SELECT max(id) from
          ↪students")->fetch(PDO::FETCH_NUM);
      $id       = min($maxIdArr[0]+1,201900001);
    }
    $this->id = $id;

    $query = 'INSERT INTO ' . $this->table .
             '(id, sname, fname, prog, tAddrId, pAddrId)
      VALUES (:id,:sname,:fname,:prog,:tAddrId,:pAddrId)';

    $stmt = $this->conn->prepare($query);
    $stmt->execute(array($this->id,     $this->sname,
                         $this->fname,  $this->prog,
                         $this->tAddrId,$this->pAddrId));
    return $this->id;
  }
} // end of Student Class
```

Model.php: createData Function (1)

```
function createData($db,$method,$resource) {
  if (($method == 'POST')  &&
      ($resource[0] = 'students') &&
      (count($resource) == 1)) {
    return createStudent($db);
  } else {
    throw new Exception('Method Not Supported', 405);
  }
}
```

Model.php: createData Function (2)

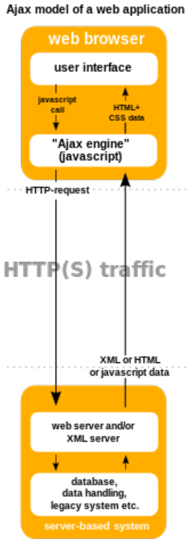
```
function createStudent($db) {
  // Retrieve and decode the student data in the HTTP
      ↪request
  $studentData = json_decode(file_get_contents('php://input'
      ↪),TRUE);
  // Record the two addresses that may be included in that
      ↪data
  createAddress($db,$studentData,'tAddr');
  createAddress($db,$studentData,'pAddr');
  // Record the rest of the student data
  $std1 = new Student($db->conn);
  $std1->set($studentData);
  $std1->create();
  // Return the Student object we have created in the
      ↪process
  return $std1;
}
```

Model.php: createData Function (3)

```
function createAddress($db,&$studentData,$addrType) {
  if ($studentData[$addrType]) {
    $addr1 = new Address($db->conn);
    $addr1->set($studentData[$addrType]);
    // Store address in the database
    // create() returns the unique id associated with the
    // address which we store as a value of the
    // $addrType . 'Id' key in the $studentData array
    $studentData[$addrType . 'Id'] = $addr1->create();
    unset($studentData[$addrType]);
    // Need to get primary key of created entry and add to
    // $studentData
  }
}
```

Summary

- Web applications almost always combine client-side scripting and server-side scripting
- Ajax (Asynchronous JavaScript and XML) is a set of techniques for sending and retrieving data from a server (asynchronously)
- On the server-side often a PHP script acts as mediator that retrieves data from a database in response to Ajax requests, possibly in the form of a web service
- Data is typically exchanged in XML or JSON (JavaScript Object Notation) format


Ajax model of a web application
web browser
user interface
javascript call
HTML+ CSS data
"Ajax engine" (javascript)
HTTP-request
HTTP(S) traffic
XML or HTML or javascript data
web server and/or XML server
database, data handling, legacy system etc.
server-based system

By DanielSHaischt, via Wikimedia Commons https://commons.wikimedia.org/wiki/File%3AAjax-vergleich.svg, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=29724785

What next?

- Development of applications typically does not start from scratch
  $\leadsto$ modules and libraries / frameworks are used
- PHP frameworks

Laravel	Zend Framework
CodeIgniter	Phalcon
Symfony	FuelPHP
CakePHP	PHPPixie

- JavaScript frameworks

jQuery	Ember.js
Angular (Google)	Node.js
React (Facebook)	Mithril
Vue.js	Polymer

- Using a framework is a skill in itself
  Popularity / use of frameworks changes quite frequently
  $\leadsto$ not clear which ones to teach / learn

Revision and Further Reading

- Read
  - Apache HTTP Server Tutorial: `.htaccess` files
    `https://httpd.apache.org/docs/2.4/howto/htaccess.html`
  - Apache Module `mod_rewrite`
    `https://httpd.apache.org/docs/2.4/mod/mod_rewrite.html`
  - RewriteRule Flags
    `http://httpd.apache.org/docs/current/rewrite/flags.html`

  of The Apache Software Foundation: Apache HTTP Server Version 2.4 Documentation. The Apache Software Foundation, 2019.
  `http://httpd.apache.org/docs/current/` [accessed 30 Nov 2019]