Intro to REST Web Services

REST = Representation State Transfer

Example: an Order Service

- Create an order
- Retrieve the order to check status
- Update/replace the order
- Delete the order

But instead of accessing its own DB, it sends requests over the Internet...

This Order Service Maps into HTTP verbs as follows:

<table>
<thead>
<tr>
<th>Verb</th>
<th>URI</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>/orders</td>
<td>Create new order</td>
</tr>
<tr>
<td>GET</td>
<td>/orders/1234</td>
<td>Get info on order 1234</td>
</tr>
<tr>
<td>PUT</td>
<td>/orders/1234</td>
<td>Update all of order 1234</td>
</tr>
<tr>
<td>DELETE</td>
<td>/orders/1234</td>
<td>Delete order 1234</td>
</tr>
</tbody>
</table>

- The new order URI is returned from the POST in the Location header.
- Here the HTTP status code is 201, Created, unlike redirection, which uses status code 302.
- To override the PHP-header default status code of 302, use a third argument:
  
  ```php
  header('Location: ' . $url . $orderID, true, 201);
  ```

From the client viewpoint:

**POST an Order and find out its new URL (one request-response cycle):**

1. POST JSON (or XML) describing the order to http://server.com/rest/orders/, the collection URL.
2. Get back JSON for order with id filled in, say order 22, and Location header with http://server.com/rest/orders/22, or alternatively, just get back the new URL in Location.

- This means this order’s new resource has URL http://server.com/rest/orders/22, so a GET to that URL will retrieve the JSON representation.
- Note: Although we see JSON on the wire, the data in the server is usually in ordinary database tables.

From client viewpoint: Find out the order status (one request/response cycle):

1. Do GET to http://server.com/rest/orders/22
2. Get back JSON for order with current status filled in

- Note that the server-side software can change the status over time, so a later GET may return a changed order.
- The rules of REST say the server should not change the order because of the GET. GET is “read-only”. If you want to change a resource, use POST or PUT.

JSON and XML: similar capabilities

```json
{
  "ID": "1",
  "Name": "M Vaqqas",
  "Email": "m.vaqqas@gmail.com",
  "Country": "India"
}
```

```xml
<Person>
  <ID>1</ID>
  <Name>M Vaqqas</Name>
  <Email>m.vaqqas@gmail.com</Email>
  <Country>India</Country>
</Person>
```

From
http://www.drdobbs.com/web-development/restful-web-services-a-tutorial/240169069
The idea of REST is to use HTTP directly.

With REST, we use multiple HTTP verbs:
• GET for reading data (no changes allowed in server!)
• POST for creating new data items
• PUT for updating old data items
• DELETE for deleting old data items
• HTTP headers are also used. One so far, Location, but more to come.

The idea of REST is to use HTTP directly.

There’s no message “envelope” as seen in other web service methodologies, like SOAP

• We can say REST is a software architectural style for distributed systems.
• It’s OK to say “REST protocol” as long as you understand it’s really just the HTTP protocol.
• It was created by Roy Fielding, and described in his widely-read PhD thesis.
• He got a degree in 2000 after doing a lot of important work on the HTTP and URL specs.

“Everything is a resource”

— the RESTful way.

Each resource has its own URL

Example: http://server.com/rest/orders/22 for order 22

REST is a “Stateless” Protocol

Each request contains all the information needed by the receiver to understand and process it. (This is also true for SOAP.)

That’s just like HTTP, after all.

Note the Wikipedia article on REST


Displaying and Understanding Deep Arrays

Recall example from Chapter 11

```
```

Bill
Jen
Ann
Roy
Pete
Murach
Murach
Sue
Hans
Krwalife

1003  5335  2083  7296  7738
Creating this array

```php
$array0 = array('first'=>'Mike', 'last'=>'Murach','id'=>6453);
// Or alternatively, use [
$array0 = ['first'=>'Mike', 'last'=>'Murach','id'=>6453];
$array1 = array('first'=>'Joel', 'last'=>'Murach','id'=>5635);
$a = array($array0,$array1);
// or alternatively:
$a =[$array0,$array1];
```

Printing out the array

```php
print_r($a);
```

```
Array ( 
[0] => Array ( 
[first] => Mike 
[last] => Murach 
[id] => 6453 ) 
[1] => Array ( 
[first] => Joel 
[last] => Murach 
[id] => 5635 ) 
)
```

Much better output!

```php
print_r($a);
```

```
Array ( 
[0] => Array ( 
[first] => Mike 
[last] => Murach 
[id] => 6453 ) 
[1] => Array ( 
[first] => Joel 
[last] => Murach 
[id] => 5635 ) 
)
```

From newer YouTube app: $data

```php
$array = 
[0] => Array ( 
[kind] => youtube#searchListResponse 
[etag] => "tbWC5XrSXxe1WOAx6MK9z4hHSU8/cSqvDUAWEuyI_04UBZ1VjqSkQnE" 
[nextPageToken] => CAUQAA 
[pageInfo] => Array ( 
[totalResults] => 1000000 
[resultsPerPage] => 5 ) 
[items] => Array 
[0] => Array ( 
[kind] => youtube#searchResult 
[etag] => "tbWC5XrSXxe1WOAx6MK9z4hHSU8/0AePmGSSrYn8E4js8CGXESKeY8Y" 
[id] => Array ( 
[kind] => youtube#video 
[videoId] => ek_W75G_JJw ) 
[snippet] => Array ( 
[publishedAt] => 2015-05-03T13:00:01.000Z 
[channelId] => UCboMX_UNgaPBsUOIgasn3-Q 
[title] => LOST IN SPACE - Astronaut Simulator Gameplay ) 
) 
)
```

From user notes of PHP doc on print_r:

“I add this function to the global scope on just about every project I do, it makes reading the output of print_r() in a browser infinitely easier.”

```php
<?php
function print_r2($val){
  echo '<pre>'; print_r($val); echo '</pre>;
}
?>
```

Project 2 supply orders

Product 11 is flour, 12 is cheese.
Order 40 units of flour and 60 units of cheese

```php
$item0 = array('productID'=>11, 'quantity'=>40); 
$item1 = array('productID'=>12, 'quantity'=>60); 
$order = array('customerID'=> 1, 'items' => array($item0, $item1));
```
Supply order ($order) in PHP

Array
{
    [customerId] => 1
    [items] => Array
        {
            [0] => Array
                {
                    [productID] => 11
                    [quantity] => 40
                }
            [1] => Array
                {
                    [productID] => 12
                    [quantity] => 60
                }
        }
}

Supply order in JSON

• JSON is the encoding we’ll use in our web services to transport data across the network
• PHP makes it easy to convert data to JSON:

```php
echo json_encode($order);
```

// See output as it will go out:

```json
{"customerId":1,"orderID":3,"delivered":true,"items":[{"productID":11,"quantity":40},{"productID":12,"quantity":60}]
```

// We want better-looking output again for debugging…

Pretty printing JSON

`// Use a flag for better formatting, for human consumption`

```php
function print_JSON($val){
    echo '<pre>
    echo json_encode($val,JSON_PRETTY_PRINT);
    echo '</pre>';
}
```

Pretty-printed JSON

Order received back from server: Just the basic facts, no “ItemID”

```json
{"customerId":1,"orderID":3,"delivered":true,"items":[{"productID":11,"quantity":40},{"productID":12,"quantity":60}]
```

JSON Essentials

The following is from
• http://www.w3schools.com/json/json_syntax.asp
• http://www.tutorialspoint.com/json/json_schema.htm

JSON Syntax Rules

JSON syntax is derived from JavaScript object notation syntax:
• Values are numbers, strings, objects, arrays, true, false, or null
• Strings are in double-quotes and encoded in UTF-8
• Curly braces hold objects, with name/value pairs for properties
• Square brackets hold arrays of values
• Syntax diagrams
JSON Values

JSON values can be:

- A number (integer or floating point)
- A string (in double quotes, in UTF-8)
- A Boolean (true or false)
- An array (in square brackets)
- An object (in curly braces)
- null

Example: GET /order/1

```
{
    "customerID": 1,
    "orderID": 3,
    "delivered": true,
    "items": [
        {
            "productID": 11,
            "quantity": 40
        },
        {
            "productID": 12,
            "quantity": 60
        }
    ]
}
```

Here, see:

- Numbers, Boolean true
- Many strings
- 1 array
- 3 objects, 2 nested
- 6 name/value pairs with integer values
- 1 name/value pair with Boolean value
- 1 name/value pair with array value

• Now have orderID from DB

JSON Objects

- JSON objects are written inside curly braces.
- JSON objects can contain zero, one or multiple name/value pairs, for example
  
  ```
  {"firstName": "John", "lastName": "Doe"}
  ```

- This is set containment. The following is considered the same object
  ```
  {"lastName": "Doe", "firstName": "John"}
  ```

- This is a big difference from XML, but generally helpful in applications
- The names must be strings and should be different from each other.

JSON Arrays

- JSON arrays are written inside square brackets, and are ordered.
- A JSON array can contain zero or more objects, or other values:
- Example array of objects
  ```
  [{"firstName": "John", "lastName": "Doe"},
   {"firstName": "Anna", "lastName": "Smith"},
   {"firstName": "Peter", "lastName": "Jones"}
  ]
  ```

More JSON array examples

- Array of numbers
- [1, 2, 3]
- Array of numbers and another array
- [1, 2, [2, 3]]
- Array of strings
- ["apple", "banana"]
- Array of strings and numbers (not a great idea)
- ["apple", 3, 20.5]

FYI: JSON Schema

- For a long time (’99–10), JSON was considered inferior to XML because it had no schemas
- A schema is a way to specify format
- JSON Schema is an Internet Draft, currently version 5, Oct. 2016.
- Version 0 is dated in 2010
- Schemas allow a server to specify needed formats of received data, and also the sent data.
- For more info, see
  http://spacetelescope.github.io/understanding-json-schema/
JSON schema for a product

```json
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "Product",
  "description": "A product from Acme's catalog",
  "type": "object",
  "properties": {
    "id": {
      "description": "The unique identifier for a product",
      "type": "integer"
    },
    "name": {
      "description": "Name of the product",
      "type": "string"
    },
    "price": {
      "type": "number",
      "minimum": 0,
      "exclusiveMinimum": true
    }
  },
  "required": ["id", "name", "price"]
}
```

Valid object:
```json
{
  "id": 2,
  "name": "CD",
  "price": 12.50
}
```

PHP JSON Functions

- **json_encode**
  - Returns the JSON representation of a PHP value
  - **PARAMETERS:**
    - value: The value being encoded. This function only works with UTF-8 encoded data (this includes ASCII).
    - options: This optional value is a bitmask consisting of JSON_HEX_QUOT, JSON_HEX_TAG, JSON_HEX_AMP, JSON_HEX_APOS, JSON_NUMERIC_CHECK, JSON_PRETTY_PRINT, JSON_UNESCAPED_SLASHES, JSON_FORCE_OBJECT
  - We shouldn’t need any of these options except JSON_PRETTY_PRINT

- **json_decode**
  - Decodes a JSON string to PHP
  - **SYNTAX:**
    ```php
    json_decode($json [, $assoc = false [, $depth = 512 [, $options = 0 ]]])
    ```
  - **PARAMETERS:**
    - $json_string: It is encoded string which must be UTF-8 encoded data
    - $assoc: It is a boolean type parameter, when set to TRUE, returned objects will be converted into associative arrays (default is Standard Object). We need to use this, but not the following two arguments:
      - $depth: It is an integer type parameter which specifies recursion depth
      - $options: It is an integer type bitmask of JSON decode, JSON_BIGINT_AS_STRING is supported.

Decoding JSON in PHP (json_decode)

PHP `json_decode()` function is used for decoding JSON in PHP. This function returns the value decoded from json to appropriate PHP type.

**Use of json_decode in proj2_server**

```php
<?php
$bodyJson = '{
  "productID": "1", 
  "categoryID": "1", 
  "productCode": "strat", 
  "productName": "Fender Stratocaster", 
  "description": "The Fender Stratocaster is a classic solid-body electric guitar. The 21-fret fingerboard features a maple fingerboard and rosewood fretboard. The Stratocaster's signature sound is defined by its three single-coil pickups, the Rosewood fingerboard, and the traditional Stratocaster body shape. Tinted neck",
  "listPrice": "699.00", 
  "discountPercent": "30.00", 
  "dateAdded": "2013-10-30 09:32:40", 
  "categoryName": "Guitars"
}';

$body = json_decode($bodyJson); 

var_dump($body);
```

The stdClass is a built-in class used for typecasting to object, etc.
Proj2 REST web services

Supplied web services

• POST /proj2_server/rest/day sets the current day number for the server.
• GET /proj2_server/rest/day returns the current day
• GET /proj2_server/rest/products/2 returns info on product 2
• POST /proj2_server/rest/products adds a new product

You will add

• POST /proj2_server/orders creates a new supply order
• GET /proj2_server/orders/123 returns info on order 123, including status (i.e., delivered or not)
• GET /proj2_server/orders returns info on all orders

Here "proj2_server" is short for /cs637/username/proj2/proj2_server

Resources

• Two kinds of resources:
  • day, a singleton, no collection involved
    – POST to .../day to change value
    – GET to .../day to read value
  • products, orders: normal collection resources
    – POST JSON to .../products -> new product, say products/12
    – POST JSON to .../orders -> new order, say orders/12
    – New URI returned in Location header
    – GET to .../products/12 gets JSON for product 12
    – GET to .../orders/12 gets JSON for order 12
    – GET to .../orders gets JSON for all orders

Client and Server

• Server: proj2_server, derived from ch24_guitar_shop
  – Has web service code for the provided services
  – Has a new category for pizza supplies
  – You add web services for supply orders

• Client: pizza2, fixed-up pizza1 solution
  – You add inventory tracking, ordering supplies by using the web services
  – Has "restclient" code to get you started, exercises provided services

Logging, position-independence

• Both projects have the improved main.php
  – Include path set to project root using __DIR__
  – Error level set to debugging level
  – Error logging set up to output to local files

• Both projects can be moved around on the web server and still work
  – This involves using $app_path for internal URLs, as seen in ch20_guitar_shop

• Delivery is expected in /cs637/username/proj2/pizza2
  /proj2_server

Web service code

• PHP code in rest/index.php of proj2_server
  – This code does not include util/main.php
  – Some of main.php code is here, though
  – Web server code needs to avoid sending error text in response: will mess up other end’s interpretation
    – i.e., don’t “echo” debugging info: use error_log() in web service code outputs to php_server_errors.log (instead of php_errors.log)
  – Also rest/.htaccess is important—will discuss
    – As “dot file”, not listed by ls command in Linux/Mac
    – Need to use ls -a to see it
    – You don’t need to change this file, just be sure it’s there!
web service client code

- pizza2 has /restclient directory with client code for its web service requests, just for testing
- This code should help you write needed code for the “real” client, the rest of pizza2
- restclient has two PHP files:
  - index.php at top level, including util/main.php
  - scan.php: another app, FYI, not related to proj2
- index.php uses Guzzle to send off GETs and POSTs:
  - Guzzle is installed in ../vendor

```php
// Instantiate Guzzle HTTP client
 httpClient = new \GuzzleHttp\Client();
$url = 'http://'.$base_url.'/day/';
error_log('......restclient: POST day = 3 to '. $url); try {
    $response = $httpClient->request('POST', $url, [ 'json' => 3]);
    $status = $response->getStatusCode();
} catch (\GuzzleHttp\Exception $e) {
    $status = 'POST failed, error = '. $e;
    error_log($status);
}
```
Command-line curl example 2

With `-v` for verbose:

topcat$ curl -v -d 9 -H Content-Type:text/plain
   http://localhost/cs637/eoneil/proj2_server/rest/day/
   * Hostname was NOT found in DNS cache
   * Trying 127.0.0.1...
   * Connected to localhost (127.0.0.1) port 80 (#0)
   > POST /cs637/eoneil/proj2_server/rest/day/ HTTP/1.1
   > User-Agent: curl/7.35.0
   > Host: localhost
   > Accept: */*
   > Content-Type:text/plain
   > Content-Length: 1
   
   * upload completely sent off: 1 out of 1 bytes
   * good news!
   * more good news
   < HTTP/1.1 200 OK
   < Date: Tue, 04 May 2015 14:33:16 GMT
   < Server: Apache/2.4.7 (Ubuntu) is not blacklisted
   < X-Powered-By: PHP/5.5.9-1ubuntu4.9
   < Content-Length: 0
   < Content-Type: text/html
   <
   
   Shell scripts

We can automate command line work with shell scripts (even on Windows):
topcat$ more test1.sh
curl -i -d 9 -H Content-Type:text/plain
   http://localhost/cs637/eoneil/proj2/proj2_server/rest/day/
   
Fills in eoneil for $1 in script:
topcat$ chmod +x test1.sh
topcat$ test1.sh eoneil

For Windows: test1.cmd: use %1% instead of $1.
See shell and .cmd files in proj2_tests directory.

Command-line curl example 2

With `–i` for status info: less clutter, get the basic facts:
curl -i -d 9 -H Content-Type:text/plain
   http://localhost/cs637/eoneil/proj2/proj2_server/rest/day/

Explanation of arguments:

- `–d 9` data: use method POST, with POST data "9". Defaults to Content-Type for encoded parameters, like form submission fields x=10&y=20 or whatever
- `–H Content-Type:text/plain` override the default Content-Type to this type, plain text

So this command does a POST to the URL with just "9" as POST data, and reports verbosely on the action

topcat$ curl -i -d 9 -H Content-Type:text/plain
   http://localhost/cs637/eoneil/proj2/proj2_server/rest/day/

Explanation of arguments:

- `-i` return status info
- `-d 9` data: use method POST, with POST data "9". Defaults to Content-Type for encoded parameters, like form submission fields x=10&y=20 or whatever
- `-H Content-Type:text/plain` override the default Content-Type to this type, plain text

So this command does a POST to the URL with just "9" as POST data, and reports verbosely on the action