PHP Web Services

Intro to REST Web Services

REST = Representation State Transfer

Example: an Order Service

A simple CRUD service:
- Create an order
- Retrieve the order to check status
- Update/replace the order
- Delete the order

But instead of accessing its own DB, the client sends requests over the Internet to the server:

REST client---------------REST server

This Order Service Maps into HTTP verbs as follows:

Verb	URI	Use
`POST`	**`/orders/`**	**`Create new order`**
`GET`	**`/orders/1234`**	**`Get info on order 1234`**
`PUT`	**`/orders/1234`**	**`Update all of order 1234`**
`DELETE`	**`/orders/1234`**	**`Delete order 1234`**

- The URI is added onto the server’s URL, so the actual URL in use with POST is http://example.com/exservice/orders, for example, where http://example.com/exservice is the “service endpoint” in use.
- The POST body has a JSON representation of the order, and similarly with PUT.
- Similarly, the GET returns that JSON representation.

REST client---------------REST server (or XML instead of JSON)

JSON

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 network, the data in the server is usually in ordinary database tables.

JSON and XML: similar capabilities

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

<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

From client viewpoint: Find out the order status for order 22 (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. Or some client did a PUT to change it.
- 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.

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

Also, generally a resource has just one URL, to avoid “aliasing” problems, but this is not a strict requirement.

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

Good tutorial:
http://www.drdobbs.com/web-development/restful-web-services-a-tutorial/240169069

How do we do this from PHP?

- We see that REST involves sending HTTP requests from the client to the server, and JSON data
- Client-side: so far, we've sent GETs and POSTs from the browser with the help of HTML forms and links
  - Now we need to generate them from the PHP program as needed for REST requests, and send and accept JSON data
- Server-side: so far we've handled GETs and POSTs, but just POSTS with form data
  - Now we need to accept GETs, send back JSON, and accept POSTs with JSON data, send back JSON

Most basic approach for the client side: use the libCurl Library, usually supplied with PHP, to send HTTP requests from our PHP code

From PHP docs:
PHP libcurl allows you to connect and communicate to many different types of servers with many different types of protocols.

- libcurl currently supports the http, https, ftp, gopher, telnet, dict, file, and ldap protocols

- Good to know—we’ll just use http and https, the secure version of http

- Current XAMPP has libcurl installed for you.
  - ✓ To check, write a file test.php with “<?php phpInfo();”, browse to it and see listing of installed libraries (search for curl on the page)

Common cURL functions

- `curl_init($url)`
- `curl_setopt($curl, OPTION, $value)`
- `curl_exec($curl)`
- `curl_close($curl)`

How to use the cURL functions

```
// Initialize the cURL session
$curl = curl_init('http://www.example.com');

// Set the cURL options so the session returns data
curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);

// Transfer the data and store it
$page = curl_exec($curl);

// Close the session
curl_close($curl);

// Process the data
$page = nl2br(htmlspecialchars($page));
echo $page;
```

How to use cURL to query YouTube (obsolete)

- From M&H, second edition

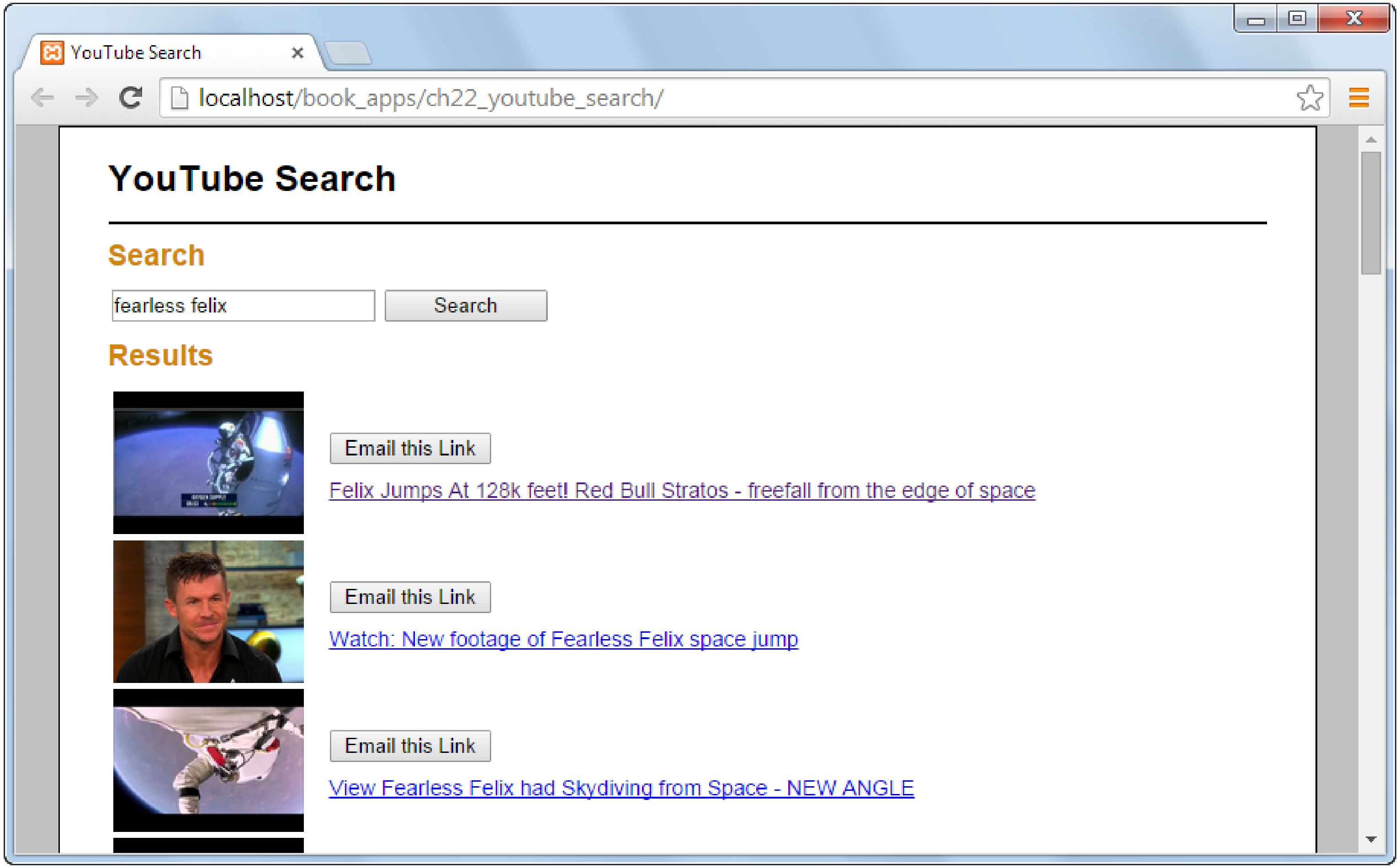
```
// Set up the URL for the query
$query = 'space shuttle';
$query = urlencode($query);
$base_url = 'http://gdata.youtube.com/feeds/api/videos';
$params = 'alt=json&q=' . $query;
$url = $base_url . '?' . $params;

// Use cURL to get data in JSON format
$curl = curl_init($url);
curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
$json_data = curl_exec($curl);
curl_close($curl);

// Get an array of videos from the JSON data
$data = json_decode($json_data, true);
$videos = $data['feed']['entry'];
```

Newer v3 YouTube API: uses HTTPS:, needs additional curl setup

Search view as it once worked



YouTube Search

Search

fearless felix

Results

Felix Jumps At 128k feet! Red Bull Stratos - freefall from the edge of space

Watch: New footage of Fearless Felix space jump

View Fearless Felix had Skydiving from Space - NEW ANGLE

Disadvantages of Curl Library

- Hard to work with headers (and we need them for Web Services)
- Hard to see the text of the actual web requests
- Old-fashioned error handling
- Not easy to work with uploads and downloads
- So, we turn to Guzzle, a PHP component

PHP Components

- A Component is a bundle of OO code meant to do one particular job, with documentation and compatibility with autoloading.
  - We'll use the well-known Guzzle component for HTTP requests, for client-side web service code
  - We'll use the Slim component for server-side web service.
  - Another component called CSV helps with reading and writing CSV (comma-separated-value) files
- De facto component registry: https://packagist.org
- Tool for using packages: composer
- See Lockhart, "Modern PHP" for good intro on components.

HTTPGuzzle

- Well-known PHP component
- OO, has autoload (like all components do)
  - So no complicated includes/requires needed
- Do GET, POST, other HTTP commands
- Set/get headers, do authorization, etc.
- Provides Exceptions on its errors.
- To be used for client-side web services
- Is implemented using the curl library

Curl vs. Guzzle

```
// Initialize the cURL session
$curl = curl_init('http://www.example.com');

// Set the cURL options so the session returns
data
curl_setopt($curl,
CURLOPT_RETURNTRANSFER, true);

// Transfer the data and store it
$page = curl_exec($curl);

// Close the session
curl_close($curl);

// Process the data
$page = nl2br(htmlspecialchars($page));
echo $page;
```

```
// set up autoloading so no includes needed
require '../vendor/autoload.php';


// Instantiate Guzzle HTTP client
$httpClient = new \GuzzleHttp\Client();


// do the GET
$response = $httpClient->get(
                    'http://www.example.com');


$page = $response->getBody();


// Process the data
$page = nl2br(htmlspecialchars($page));
echo $page;
```

Server side: use Slim

- Slim is a PHP component that helps with accepting and answering REST requests
- OO, has autoload (as all components do)
  - So no complicated includes/requires needed
- Can accept incoming GET, POST, PUT, DELETE, etc.
- Helps with parsing the incoming URL, for example picking up the id.
- Has nothing to do with curl: curl only helps with generating HTTP requests, not handling incoming ones.
- In fact, we could use plain PHP to do the server side coding, but this is the more standard approach, and can help with advanced features like authorization and CORS.

Pizza2

- Pizza2, our second programming project, will use REST web services to access the database data. It involves several cooperating codebases, i.e., projects.
  - The pizza2_phpclient project (supplied) will run the student UI and issue web service calls to get the data on toppings, orders, etc.
  - Eventually pizza2_jsclient will do the same using Javascript.
  - The pizza2_server project (in PHP) will manage the database and answer the web service requests
  - We'll continue to use the pizza1 project for the admin UI
- The pizza2_phpclient project will use the PHP component Guzzle
- Similarly, the pizza2_server project will have the Slim component installed, to help with web service code.

Pizza2

Pizza2_phpclient------------------Pizza2_server
PHP HTTP, JSON PHP

Pizza2_jsclient---------------------Pizza2_server
JS HTTP, JSON PHP

- The data flows over the network in JSON, converted to/from PHP arrays for PHP programs, or to/from JS arrays for JS programs.
- PHP is commonly used this way on the server side to assist with Javascript apps.
- Both PHP and JS can input and output JSON easily.
- You will implement pizza2_server and pizza2_jsclient in Project 2.

Provided ch05_gs_client/server

- Ch05_guitar_shop has been modified to use REST web services to access its data, to serve as a complete example for PHP web services.
- We’ll call it ch05_gs for short—

ch05_gs_client------------------ch05_gs_server

PHP HTTP, JSON PHP

- Ch05_gs_client uses REST web services to access the database data, with the help of Guzzle.
  - Otherwise it’s the same as the old ch05_guitar_shop.
- Ch05_gs_server manages the database and answers the web service requests, with the help of Slim.
  - It has no user interface itself: it only answers web requests

REST API for Ch05_guitar_shop

```
GET /categories
GET /categories/{category}/products
GET /categories/{category}/products/{pid}
POST /categories/{category}/products
DELETE /categories/{category}/products/{pid}
```

Another possible setup:

```
GET /categories
GET /products
GET /products/{pid}
POST /products
DELETE products/{pid}
```

- The hierarchical API allows a query for all the products in a certain category, a common need in this app.
- To do that in the second API, you would have to request all the products and go through them to find the ones in the category of interest.

REST Ch05_guitar_ship

REST API for project:

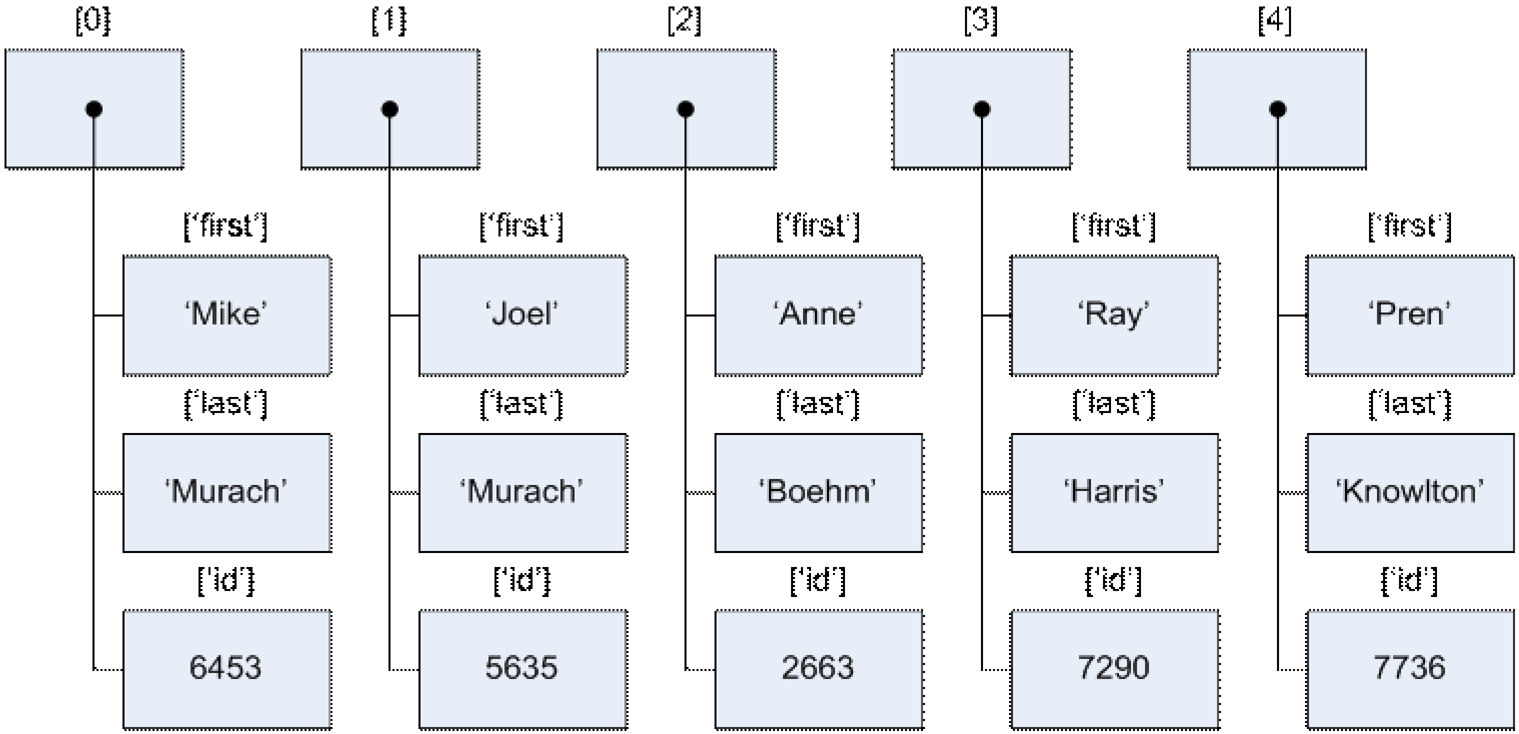
```
GET /categories
GET /categories/{category}/products
GET /categories/{category}/products/{pid}
POST /categories/{category}/products
DELETE /categories/{category}/products/{pid}

Examples:
GET /categories/guitars/products // get all guitars
POST /categories/guitars/products  // add a guitar
DELETE /categories/basses/products/2  // delete bass 2
```

We see that {category} has string values here, but alternatively it could numeric ids. In general it needs to be a unique attribute of categories.

Displaying and Understanding Deep Arrays

Recall example from Chapter 11



[0]
['first']
'Mike'
['last']
'Murach'
['id']
6453

[1]
['first']
'Joel'
['last']
'Murach'
['id']
5635

[2]
['first']
'Anne'
['last']
'Boehm'
['id']
2663

[3]
['first']
'Ray'
['last']
'Harris'
['id']
7290

[4]
['first']
'Pren'
['last']
'Knowlton'
['id']
7736

Creating this array

```
$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

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

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

Seeing it better: don’t let HTML processing mess with its natural formatting: Use <pre> to say it’s preformatted:

```
echo '<pre>';
print_r($a);
echo  '</pre>';
```

Much better output!

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

    [1] => Array
        (
            [first] => Joel
            [last] => Murach
            [id] => 5635
        )

)
```

From YouTube WS: $data array

```
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
```

`$items = $data['items'];`

$items[0]['snippet']['title'];

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

For ch05_gs: Product in PHP

```
Array
(
    [productID] => 5
    [categoryID] => 1
    [productCode] => washburn
    [productName] => Washburn D10S
    [listPrice] => 299.00
)
```

Generated by a line added to a view file of ch05_gs_client:

```
echo '<pre>' . print_r($product, true) . '</pre>';
```

Product 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:

```
echo json_encode($product);
```

See product representation (JSON) in flight across network:

```
{"productID":"4","categoryID":"1","productCode":"fg700s","productName":"Yamaha FG700S","listPrice":"489.99"}
```

Products ($products) in JSON

```
[{"productID":"7","categoryID":"2","productCode":"precision",
    "productName":"Fender Precision","listPrice":"799.99“}
{"productID":"8","categoryID":"2","productCode":"hofner",
    "productName":"Hofner Icon","listPrice":"499.99"}]
```

Generated by a line added to a view file:

```
echo echo $productsJson;
```

Products ($products) in PHP

```
Array
(
    [0] => Array
        (
            [productID] => 7
            [categoryID] => 2
            [productCode] => precision
            [productName] => Fender Precision
            [listPrice] => 799.99
        )

    [1] => Array
        (
            [productID] => 8
            [categoryID] => 2
            [productCode] => hofner
            [productName] => Hofner Icon
            [listPrice] => 499.99
        )
)
```

Generated by a line added to a view file:

```
echo '<pre>' .
print_r($products, true)
. '</pre>';
```

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

JSON Example

```
{
    "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

JSON Objects (also simple JS 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 (also JS 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 8 (officially Draft 2019-09), September, 2019.
  - Version 0 is dated in 2010
  - Version 5 is still in serious use, for example in Swagger, an important tool/description method for REST APIs using JSON.
- 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

```
{
    "$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:
  {
      "id": 2,
      "name": “CD",
      "price": 12.50,
  }
```

PHP JSON Functions

- json_encode Returns the JSON representation of a PHP value
- json_decode Decodes a JSON string to PHP
- json_last_error Returns the last error

json_encode

```
string json_encode($value [, $options = 0 ])
```

PARAMETERS:

- value: The value being encoded. This function only works with UTF-8 encoded data (this includes ASCII).
- In the event of a failure to encode, json_last_error() can be used to determine the exact nature of the error.
- 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

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.

SYNTAX:

```
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.

Json_decode Example

```
<?php
   $json = '{"a":1,"b":2,"c":3,"d":4,"e":5}';
   var_dump(json_decode($json));
   var_dump(json_decode($json, true));
?>
```

```
object(stdClass)#1 (5) {
        ["a"] => int(1)
        ["b"] => int(2)
        ["c"] => int(3)
        ["d"] => int(4)
        ["e"] => int(5)
}
array(5)
{       ["a"] => int(1)
        ["b"] => int(2)
        ["c"] => int(3)
        ["d"] => int(4)
        ["e"] => int(5)
}
```

The stdClass is a

built-in class used for

typecasting to object, etc.

- We’ll use the second form

Use of json_decode in ch05_gs_client

In rest_get_product(…) of model/web_services.php :

```
$product = json_decode($productJson, true);
```

```
Here $productJson =
   {"productID":"4","categoryID":"1","productCode":"fg700s“,
   "productName":"Yamaha FG700S","listPrice":"489.99"}
```

With the second-arg = true, we get a PHP associative array, instead of a “standard object”: So $product is the PHP associative array as follows:

```
Array
(
    [productID] => 4
    [categoryID] => 1
    [productCode] => fg700s
    [productName] => Yamaha FG700S
    [listPrice] => 489.99
)
```

Client and Server

- Server: ch05_gs_server, using Slim
- Client: ch05_gs_client, fixed-up ch05_guitar_shop, same UI
- These two projects are meant to be siblings in your cs637/username directory on pe07 or XAMPP.
  - ❑ /cs637/username/ch05_gs_client : client side
  - ❑ /cs637/username/ch05_gs_server: server side
- Kludge warning: ch05_gs_client finds its CSS from /book_apps/ch05_guitar_shop/main.css
  - – So assumes the /book_apps is available on this system.
  - – It’s not easy to make PHP projects position-independent, i.e., able to run anywhere on the web server file system
  - – So if the HTML looks crummy, fix the CSS link or /book_apps

Web service code

- PHP code in api/index.php of ch05_gs_server
  - 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()
  - error_log() in web service code outputs to same file as the client side, so label output "client" or "server", or use error_log only from the server side.
  - See slides 27-32 of Chapter 6 (6pp) for enabling and using error_log()
- Also api/.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!

Testing web services

- Web services (even old-style SOAP services) are “stateless”
- This means each service request contains all needed data for the server to do its job
- REST web services are just HTTP commands
- Therefore we can just fire HTTP commands at the server to test its services
- We can use command-line curl: we’ll cover this next time.