Web Services & APIs

Day 1

Mohamed Shehata

Day 1: API Consumption & Integration

Goal: Master the art of connecting PHP applications to external data sources.

- **Introduction to Web Services:** Understanding the Request/Response cycle and the role of an HTTP Client.
- **HTTP Protocol Fundamentals:** Deep dive into Methods (GET, POST), Headers, and Status Codes (200, 404, 500).
- **Data Interchange with JSON:** Structure, syntax, and parsing JSON data within PHP using `json_decode`.
- **Dependency Management:** Setting up the environment and managing libraries using **Composer**.
- **Professional Tooling:** Introduction to **Guzzle HTTP** and why it is the industry standard for PHP developers.

What is an API?

- API = Application Programming Interface
- Way for systems to communicate
- Like a waiter in a restaurant

A general term for a set of rules and specifications that define how two software components should interact.

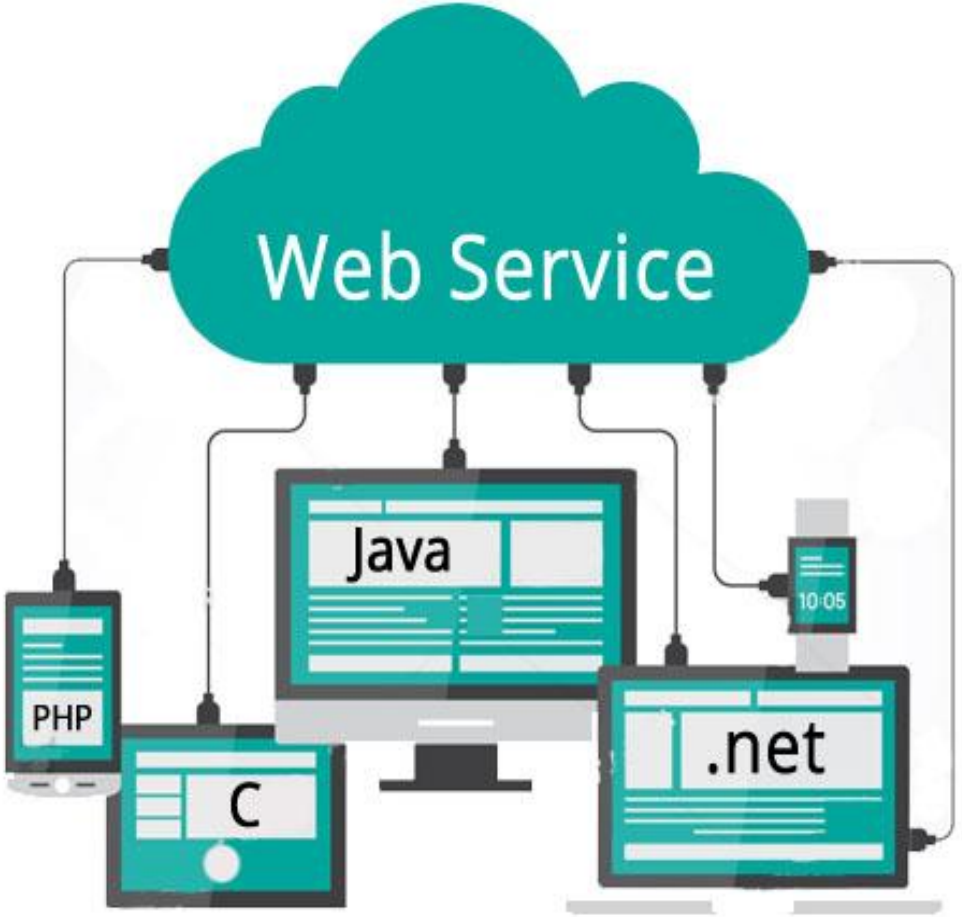
Real Life Example

- Facebook login
- Payment gateways
- Weather apps

What is a Web Service?

Web service

A standardized software, application, or cloud technology that enables software on different platforms (e.g., PHP, Java, .NET) to communicate and exchange data throughout the internet.



API vs Web Service

API: general concept
Web Service: API over network

Types of Web services

Types of Web services

- REST
- SOAP
- GraphQL
- gRPC

REST (Most Common)

- Uses HTTP
- Uses JSON
- Simple and widely used

JSON

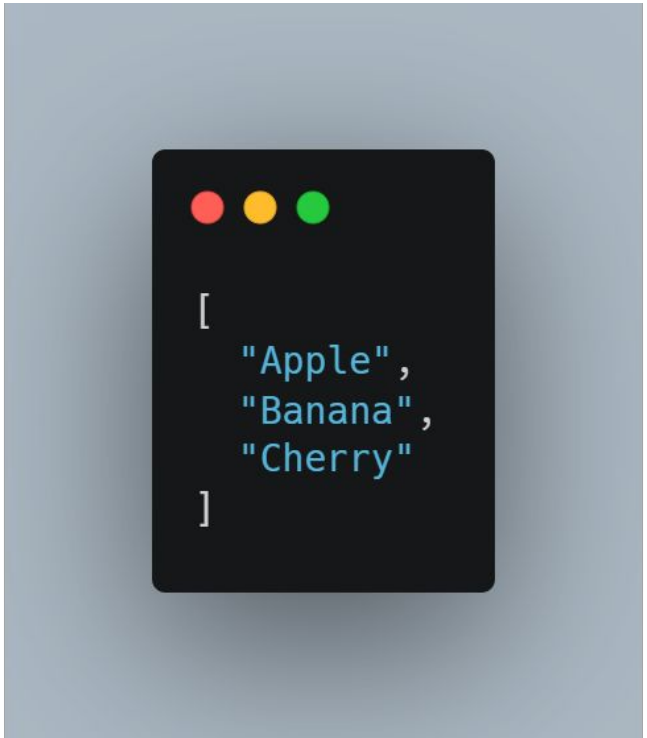
- JSON stands for **J**ava**S**cript **O**bject **N**otation. JSON is a standard lightweight data interchange format which is quick and easy to parse and generate.
- JSON, like XML, is a text-based format that's easy to write and easy to understand for both humans and computers
- JSON is based on two basic structures:
  - Object: This is defined as a collection of key/value pairs (i.e. key:value). Each object begins with a left curly bracket **{** and ends with a right curly bracket **}**. Multiple key/value pairs are separated by a comma **,**.
  - Array: This is defined as an ordered list of values. An array begins with a left bracket **[** and ends with a right bracket **]**. Values are separated by a comma **,**.

JSON STRUCTURE

JSON Object

```
{
  "book": {
    "title": "The Great Gatsby",
    "author": "F. Scott Fitzgerald",
    "published_year": 1925,
    "available": true
  }
}
```

JSON Array



```
[
  "Apple",
  "Banana",
  "Cherry"
]
```

JSON & PHP

PHP JSON Encode

JSON Object

```php
<?php

$user = ['name' => 'Ali', 'email' => 'ali@example.com', 'gender' => 'male'];

echo json_encode($user);
?>

The output of the above example will look like:

{"name":"Ali","email":"ali@example.com","gender":"male"}
```

PHP JSON Encode

JSON Array

Example

```php
<?php
// An indexed array
$colors = array("Red", "Green", "Blue", "Orange", "Yellow");

echo json_encode($colors);
?>
```

The output of the above example will look like this:

```
["Red","Green","Blue","Orange","Yellow"]
```

PHP JSON Encode

JSON Object

Example

```php
<?php
// An indexed array
$colors = array("Red", "Green", "Blue", "Orange");

echo json_encode($colors, JSON_FORCE_OBJECT);
?>
```

The output of the above example will look like this:

```
{"0":"Red","1":"Green","2":"Blue","3":"Orange"}
```

PHP JSON DECODE: OBJECT

Object

Example Run this code »

```php
<?php
// Store JSON data in a PHP variable
$json = '{"Peter":65,"Harry":80,"John":78,"Clark":90}';

var_dump(json_decode($json));
?>
```

The output of the above example will look something like this:

```
object(stdClass)#1 (4) { ["Peter"]=> int(65) ["Harry"]=> int(80) ["John"]=> int(78)
["Clark"]=> int(90) }
```

PHP JSON DECode: Associative array

Array

```php
<?php
// Store JSON data in a PHP variable
$josn = '{"Peter":65,"Harry":80,"John":78}';

var_dump(json_decode($json, true));
?>

The output of the above example will look like:

array(3) { ["Peter"]=> int(65) ["Harry"]=> int(80) ["John"]=> int(78) }
```

HTTP

How Do Web Services Utilize HTTP?

- HTTP is the underlying protocol used by web services
- This is the same protocol you just used to request this web page (a resource!) in your browser
- HTTP provides mechanisms to handle the requests and responses to RESTful services
- This includes transferring data and/or files
- The Client/Service/Server Relationship
- A client is the system connecting to and making a request to a service hosted on a server
- The server processes the request, returns a response to the client, and closes the connection

HTTP Message

- Clients and services talk to each other messages
- HTTP messages follow a request and response cycle; each request by the client requires a response from the server
- Requests to a service and responses from a service are both structured messages
- The actual message is just a series of lines of plain text

Http Request

<verb>: An HTTP method that defines the action of the request. Examples: GET, POST, PUT, DELETE,.. .

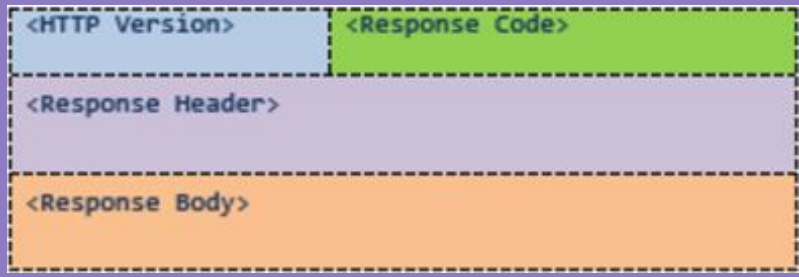
<URI>: (Uniform Resource Identifier)
A URI is a resource on a server that can be accessed by a service. The most common form of URI is a URL (Uniform Resource Locator)

<HTTP Version>: The version of HTTP

<Request Header>: Contains request metadata in key-value pairs format

<Request Body>: The actual content (data) of the message

HTTP Response



<HTTP Version>: The HTTP version used

<Response Code>: The server always returns a code which contains the status of the request. This response code is generally a 3-digit http status code

<Response Header>: Contains metadata about the response message

<Response Body>: Contains the representation if the request was successful

HTTP METHODS

- GET
  - Used to retrieve data and should not contain a request content.
  - Can be bookmarked
- POST
  - Causes changes in the server (most cases in create, so it shouldn’t be repeated, it has a request body
- PUT
  - Replaces all current representations of the target resource with the request payload
- Delete
  - Deletes the specified resource
- Are there other http verbs / methods ??

HTTP Status Codes

- 2xx Successful response
  - **200** (OK) The request succeeded
  - **201** (Created) Succeeded and a new resource was created as a result
  - **202** (Accepted) The request has been received but not yet acted upon
  - **204** (No Content) There is no content to send for this request

- 3xx Redirection
  - **301** (Moved Permanently) The URL of the requested resource has been changed permanently. The new URL is given in the response.
  - **302** (Found) The URI of requested resource changed temporarily.
  - **304** (Not Modified) This is used for caching purposes.

HTTP Status Codes

- 4xx Client error
  - **400** (Bad Request) The server cannot or will not process the request due to a client error (e.g., malformed request syntax)
  - **401** (unauthorized) Means "unauthenticated". That is, the client must authenticate to get the requested response.
  - **403** (Forbidden) The client does not have access rights to the content
  - **404** (Not Found) The server cannot find the requested resource
  - **405** (Method Not Allowed) The request method is known by the server but is not supported by the target resource
  - **422** (Unprocessable Content) The request was well-formed but was unable to be followed due to some errors (e.g., validation errors)
  - **429** (Too Many Requests) The user has sent too many requests in a given amount of time (rate limiting).

- 5xx Server error
  - **500** (Internal Server Error) The server has encountered a situation it does not know how to handle.

Using Remote Apis

Using Remote APIs

- The “Why”
  - Instead of building payment gateway we can use Fawry API or Stripe
  - Instead of building Maps System we can use Google Maps API
  - Our Application now acts as a **client** requesting data from other servers.

Using Remote APIs: Tools

Using Remote Apis from PHP

Remote Apis Calls from PHP

- There are many ways you can get information from an API using PHP.

1. **file_get_contents()**: Simple for basic tasks.
2. **curl** php extension: One of the most common ways. While this library is perfectly functional, it is designed for low-level control over HTTP communication.
3. **Guzzle**: PHP HTTP client that makes it easy to send HTTP requests and trivial to integrate with web services.It acts as an abstraction layer that can switch between different handlers (e.g., curl).
4. **HTTP Client** for Laravel: A Laravel's wrapper around Guzzle and is focused on its most common use cases and a wonderful developer experience.

Remote Apis Calls Workflow

1. Authentication: Get API key or token.
2. Endpoint: The required url like: /api/v1/weather
3. Method: GET to retrieve data or POST sending data or …
4. Parsing: Handle response data (JSON) into PHP data types

Composer

It's a dependency manager tool used everywhere in the PHP World. All large and well-known website components are using composer.
Composer will help you do the following :

1. Install 3rd party php tools and frameworks
2. Update version of the tools you use
3. Auto load your own code (so in the future you will only use composer’s autoloader!!! Cool ;)

Composer

Composer has two separate elements :

1. **Composer** itself which is a command line tool for grabbing and installing what you want to install.
2. **Packagist** - the main composer repository. This is where the packages you may want to use are stored.

- Everything Composer installs is saved in a directory named **vendor** which shouldn’t has any of your code.
- All dependencies are stored in **composer.json** file
- The currently installed versions of the dependencies are stored in **composer.lock** file

Demo Time

Remote Apis Calls using cURL

```php
<?php

// 1. Initialize cURL session
$ch = curl_init("https://jsonplaceholder.typicode.com/users/1");

// 2. Set options
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

// 3. Execute and get response
$response = curl_exec($ch);

// 4. Close session
curl_close($ch);

echo $response;
```

Remote Apis Calls using Guzzle

Add Guzzle by running the following command in your terminal: *composer require guzzlehttp/guzzle.*

```
<?php
require 'vendor/autoload.php';

use GuzzleHttp\Client;

$client = new Client(['base_uri' => 'https://dummyjson.com/', 'timeout'  => 2.0]);

$response = $client->request('GET', 'posts');// Send the GET request

if ($response->getStatusCode() == 200) {
    $data = json_decode($response->getBody(), true); // get response as PHP array
}

// Assume result is a list of Posts
foreach ($data["posts"] as $post) {
    echo "<h3>" . $post['title'] . "</h3>";
    echo "<p>" . $post['body'] . "</p>";
    echo "<hr>";
}
```

Remote Apis Calls using laravel

```
use Illuminate\Support\Facades\Http;

$response = Http::get('https://dummyjson.com/posts'); // Send the GET request

if ($response->successful()) {
    $data = $response->json(); // get response as PHP array
}

// Assume result is a list of Posts
foreach ($data["posts"] as $post) {
            …
            …
}
```