

# **Programación III**

# **API REST - SLIM**

## **Clase 8**

**Maximiliano Neiner  
Villegas Octavio  
Rampi Mario**

# Temas a Tratar

- ❖ Introducción a REST
- ❖ Composer
- ❖ Slim Framework

# Temas a Tratar

- ❖ Introducción a REST
- ❖ Composer
- ❖ Slim Framework

# REST (1/3)



## Representational State Transfer

- Es un tipo de arquitectura de desarrollo web que se apoya totalmente en el estándar HTTP.
- REST nos permite crear servicios y aplicaciones que pueden ser usadas por cualquier dispositivo o cliente que entienda HTTP, por lo que es increíblemente más simple y convencional que otras alternativas que se han usado en los últimos diez años como SOAP y XML-RPC.
- Por lo tanto REST es el tipo de arquitectura más natural y estándar para crear APIs para servicios orientados a Internet.

# REST (2/3)

- ➊ REST es un conjunto de convenciones para aplicaciones web y servicios web, que se centra principalmente en la manipulación de recursos a través de especificaciones HTTP.
- ➋ Podemos decir que REST es una interfaz web estándar y simple que nos permite interactuar con servicios web de una manera muy cómoda.
- ➌ Gracias a REST la web ha disfrutado de escalabilidad como resultado de una serie de diseños fundamentales clave:

# REST (3/3)

## Un protocolo cliente/servidor sin estado:

- cada mensaje HTTP contiene toda la información necesaria para comprender la petición.
- Como resultado, ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes.

## Un conjunto de operaciones bien definidas

- que se aplican a todos los recursos de información: HTTP en sí define un conjunto pequeño de operaciones, las más importantes son POST, GET, PUT y DELETE.

# Temas a Tratar

- ❖ Introducción a REST
- ❖ Composer
- ❖ Slim Framework

# Composer

Composer es una herramienta para la gestión de dependencias en PHP.

Permite declarar las bibliotecas de las que depende tu proyecto y las administrará (instalará/actualizará) para vos.



<https://getcomposer.org/download/>

# Temas a Tratar

- ❖ Introducción a REST
- ❖ Composer
- ❖ Slim Framework

# Temas a Tratar

- ❖ Introducción a REST
- ❖ Composer
- ❖ Slim Framework
  - Generalidades
  - Configurar index.php & .htaccess
  - Ruteo
  - POO

# Slim Framework



**Slim es un framework micro de PHP que ayuda a escribir rápidamente aplicaciones Web y APIs sencillas pero poderosas.**

**En esencia, Slim es un despachador que recibe una solicitud HTTP, invoca una rutina de devolución de llamada apropiada y devuelve una respuesta HTTP.**

<https://www.slimframework.com/docs/start/installation.html>

# Temas a Tratar

- ❖ Introducción a REST
- ❖ Composer
- ❖ Slim Framework
  - Generalidades
  - Configurar index.php & .htaccess
  - Ruteo
  - POO

# Configurar index.php (1/3)

## index.php

- En este archivo se pondrán los *verbos HTTP* que se recibirán y las rutas por las cuales vamos a poder acceder a ellos.

```
1 <?php
2 use \Psr\Http\Message\ServerRequestInterface as Request;
3 use \Psr\Http\Message\ResponseInterface as Response;
4
5 require_once '../composer/vendor/autoload.php';
6
7 $config['displayErrorDetails'] = true;
8 $config['addContentLengthHeader'] = false;
9
10 $app = new \Slim\App(["settings" => $config]);
```

En la línea 7 se habilita para poder obtener información sobre los errores, que podemos mostrarlos en la salida de la consola .

La octava línea permite al servidor Web establecer el encabezado Content-Length, lo que hace que Slim se comporte de manera más predecible.

# Configurar index.php (2/3)

## Definir el conjunto de operaciones

- Son: POST, GET, PUT y DELETE.

```
$app = new \Slim\App(["settings" => $config]);

$app->get('[/]', function (Request $request, Response $response) {
    $response->getBody()->write("GET => Bienvenido!!! ,a SlimFramework");
    return $response;
});
$app->post('[/]', function (Request $request, Response $response) {
    $response->getBody()->write("POST => Bienvenido!!! ,a SlimFramework");
    return $response;
});
$app->put('[/]', function (Request $request, Response $response) {
    $response->getBody()->write("PUT => Bienvenido!!! ,a SlimFramework");
    return $response;
});
$app->delete('[/]', function (Request $request, Response $response) {
    $response->getBody()->write(" DELETE => Bienvenido!!! ,a SlimFramework");
    return $response;
});

$app->run();
```

# Realizar...

- ❖ A partir de apirest\_starter.rar, generar los verbos faltantes (POST, DELETE y PUT)

# Configurar index.php (3/3)

## index.php

- Todas la peticiones de HTTP son atendidas por este archivo y no tendría que ponerse explícitamente en la ruta de acceso.

*Esto esta mal :*

`http://localhost:8080/miApiRest/index.php/saludo`

*Esto esta bien:*

`http://localhost:8080/miApiRest/saludo`

Para lograr esto se debe crear o modificar el archivo “.HTACCESS”

# .htaccess

- Los ficheros .htaccess o "*ficheros de configuración distribuida*", facilitan la forma de realizar cambios en la configuración de contexto de un directorio.
- Un fichero, que contiene una o más directivas, se coloca en un documento específico de un directorio, y estas directivas aplican a ese directorio y todos sus subdirectorios.

Aquí está la ayuda en la documentación:

<https://www.slimframework.com/docs/tutorial/first-app.html>

**Don't forget** to restart your server process now you've changed the configuration!

I also have a `.htaccess` file in my `src/public` directory; this relies on Apache's rewrite module being enabled and simply makes all web requests go to `index.php` so that Slim can then handle all the routing for us. Here's my `.htaccess` file:

```
RewriteEngine on
RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule . index.php [L]
```

# Temas a Tratar

- ❖ Introducción a REST
- ❖ Composer
- ❖ Slim Framework
  - Generalidades
  - Configurar index.php & .htaccess
  - Ruteo
  - POO

# Slim Framework

## Rutas

- Sin valores opcionales.

```
58     $app->get('/ruteo/', function (Request $request, Response $response) {
59         $response->getBody()->write("Ruteo, sin params");
60         return $response;
61     });
62
63
64     $app->get('/ruteo/{param}', function (Request $request, Response $response) {
65         $response->getBody()->write("Ruteo, con params");
66         return $response;
67     });
68 }
```

## Rutas con valoresopcionales

- Los corchetes “[ ]” indican que es opcional en la ruta.

```
70     $app->get('/ruteoOpcional[/]', function (Request $request, Response $response) {
71         $response->getBody()->write("Ruteo, sin params y / opcional");
72         return $response;
73     });
74
75
76     $app->get('/ruteoOpcional/{param}', function (Request $request, Response $response) {
77         $response->getBody()->write("Ruteo, con params opcional");
78         return $response;
79     });
80 }
```

# Slim Framework

## Parámetros por argumentos

```
/* parametros como argumentos*/
$app->get('/parametros/{id}', function ($request, $response, $args) {
    $id=$args['id'];
    $response->getBody()->write("GET, tomo el parametro como argumento id: $id ");
    return $response;
});
```

/apirestV1/parametros/33

```
$app->post('/parametros/{id}', function ($request, $response, $args) {
    $id=$args['id'];
    $response->getBody()->write("POST, tomo el parametro como argumento id: $id ");
    return $response;
});
```



## Varios valores

- Todos los valores entre llaves y separados con /

```
$app->get('/parametros/{id}/{nombre}', function ($request, $response, $args) {
    $id=$args['id'];
    $nombre=$args['nombre'];
    $response->getBody()->write("tomo los parametros como argumentos id: $id , nombre: $nombre ");
    return $response;
});
```

# Slim Framework

## ANY

- Recibe todos los verbos HTTP que respetan la ruta

```
/* atender todos los verbos de HTTP*/
$app->any('/cualquiera/{id}', function ($request, $response, $args) {

    var_dump($request->getMethod());
    $id=$args['id'];
    $response->getBody()->write("cualquier verbo de ajax parametro: $id ");
    return $response;
});
```

## MAP

- Recibe solo los verbos que están detallados

```
/* atender algunos los verbos de HTTP*/
$app->map(['GET', 'POST'], '/mapeado', function ($request, $response, $args) {

    var_dump($request->getMethod());
    $response->getBody()->write("Solo POST y GET");
});
```

# Slim Framework

## GROUP

- Recibe todos los verbos HTTP que respetan la ruta y los redirige a las que tiene definida

```
$app->group('/saludo', function () {  
  
    $this->get('/', function ($request, $response, $args) {  
        $response->getBody()->write("HOLA, Bienvenido a la apirest... ingresá tu nombre");  
    });  
  
    $this->get('/{nombre}', function ($request, $response, $args) {  
        $nombre=$args['nombre'];  
        $response->getBody()->write("HOLA, Bienvenido <h1>$nombre</h1> a la apirest");  
    });  
  
    $this->post('/', function ($request, $response, $args) {  
        $response->getBody()->write("HOLA, Bienvenido a la apirest por post");  
    });  
});
```

# Slim Framework

## GROUP & MAP

- Combinación de las dos formas

```
$app->group('/usuario/{id:[0-9]+}', function () {

    $this->map(['POST', 'DELETE'], '', function ($request, $response, $args) {
        $response->getBody()->write("Accedo al usuario por ID (con POST o DELETE) ");
    });

    $this->get('/nombre', function ($request, $response, $args) {
        $response->getBody()->write("Accedo al usuario por ID y retorno el nombre (con GET) ");
    });
});
```

# Temas a Tratar

 **Introducción a REST**

 **Composer**

 **Slim Framework**

- Generalidades
- Configurar index.php & .htaccess
- Ruteo
- POO

# Slim Framework

## ✖ Retronar objetos

```
$app->get('/datos/', function (Request $request, Response $response) {
    $datos = array('nombre' => 'rogelio', 'apellido' => 'agua', 'edad' => 40);
    $newResponse = $response->withJson($datos, 200);
    return $newResponse;
});
```

## ✖ Recibir objetos

```
$app->post('/datos/', function (Request $request, Response $response) {
    $ArrayDeParametros = $request->getParsedBody();
    // var_dump($ArrayDeParametros);
    $objeto= new stdclass();
    $objeto->nombre=$ArrayDeParametros['nombre'];
    $objeto->apellido=$ArrayDeParametros['apellido'];
    $objeto->edad=$ArrayDeParametros['edad'];
    $newResponse = $response->withJson($objeto, 200);
    return $newResponse;
});
```

# Slim Framework

## Recibir objetos y archivos

```
/*SUBIDA DE ARCHIVO*/
$this->post('/', function (Request $request, Response $response) {

    $ArrayDeParametros = $request->getParsedBody();
    //var_dump($ArrayDeParametros);
    $titulo= $ArrayDeParametros['titulo'];
    $cantante= $ArrayDeParametros['cantante'];
    $año= $ArrayDeParametros['anio'];
    $micd = new cd();
    $micd->titulo=$titulo;
    $micd->cantante=$cantante;
    $micd->año=$año;
    $micd->InsertarElCdParametros();

    $archivos = $request->getUploadedFiles();
    $destino="../fotos/";
    //var_dump($archivos);
    //var_dump($archivos['foto']);
    $nombreAnterior=$archivos['foto']->getClientOriginalName();
    $extension= explode(".", $nombreAnterior) ;
    //var_dump($nombreAnterior);
    $extension=array_reverse($extension);
    $archivos['foto']->moveTo($destino.$titulo.".". $extension[0]);
    $response->getBody()->write("cd");
    return $response;
});
```

**Para tener en cuenta...**

# Slim Framework

- ❖ Hacer métodos de instancia que puedan recibir las peticiones que SlimFramework les va a redirigir.
  - Se debe respetar la firma de las funciones .
  - Después de la ruta en la API se pasa el método con la sintaxis de la imagen.

```
class cdApi extends cd implements IApiUsable
{
    public function TraerUno($request, $response, $args) {
        $id=$args['id'];
        $elCd=cd::TraerUnCd($id);
        $newResponse = $response->withJson($elCd, 200);
        return $newResponse;
    }
}
```

```
$app->group('/cd', function () {
    $this->get('/', \cdApi::class . ':traerTodos');

    $this->get('/{id}', \cdApi::class . ':traerUno');

    $this->post('/', \cdApi::class . ':CargarUno');

    $this->delete('/', \cdApi::class . ':BorrarUno');

    $this->put('/', \cdApi::class . ':ModificarUno');
});
```

# Slim Framework

## 🔗 Tareas bien definidas

- Definir las interfaces necesarias.

```
interface IApiUsable{
    public function TraerUno($request, $response, $args);
    public function TraerTodos($request, $response, $args);
    public function CargarUno($request, $response, $args);
    public function BorrarUno($request, $response, $args);
    public function ModificarUno($request, $response, $args);
}
```

- Llamar a los métodos de la clase encargada de responder a los verbos HTTP

```
$app->group('/cd', function () {
    $this->get('/', \cdApi::class . ':traerTodos');

    $this->get('/{id}', \cdApi::class . ':traerUno');

    $this->post('/', \cdApi::class . ':CargarUno');

    $this->delete('/', \cdApi::class . ':BorrarUno');

    $this->put('/', \cdApi::class . ':ModificarUno');

});
```

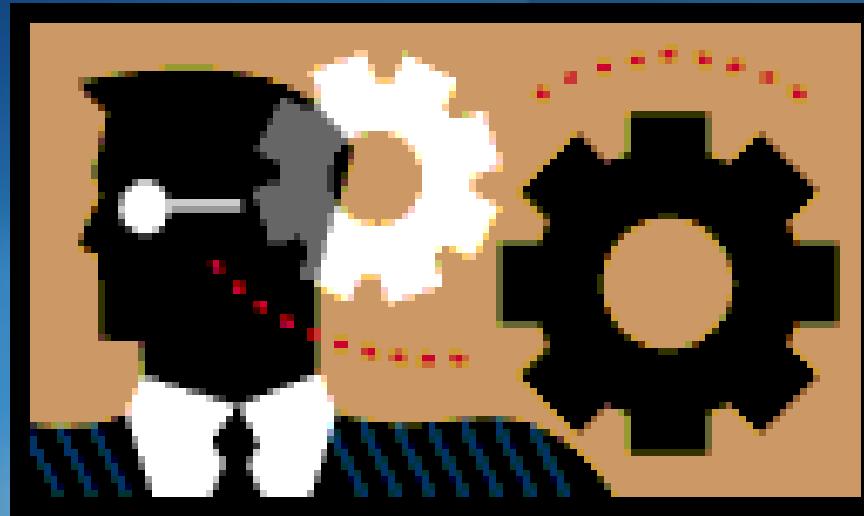
# Slim Framework

- Los métodos encargados de atender los verbos HTTP, no deberían tener contacto con la fuente de datos y deben hacerse aquí todas las validaciones necesarias.

```
class cdApi extends cd implements IApiUsable
{
    public function TraerUno($request, $response, $args) {
        $id=$args['id'];
        $elCd=cd::TraerUnCd($id);
        $newResponse = $response->withJson($elCd, 200);
        return $newResponse;
    }
}
```

- Los métodos que interactúan con la fuente de datos no deberían tener validaciones, solo las acciones sobre la fuente de datos.

```
class cd
{
    public static function TraerUnCd($id)
    {
        $objetoAccesoDato = AccesoDatos::dameUnObjetoAcceso();
        $consulta = $objetoAccesoDato->RetornarConsulta("select id, titel as titulo,
            interpret as cantante,jahr as año from cds where id = $id");
        $consulta->execute();
        $cdBuscado= $consulta->fetchObject('cd');
        return $cdBuscado;
    }
}
```



# Ejercitación

# Ejercicios

- ❖ Crear las tablas para usuarios
- ❖ Crear la api para el ABM de usuarios
- ❖ Crear la api para el login de usuarios