

# Servicios REST

Antonio Espín Herranz

# Contenidos

- Desarrollo de APIs REST en PHP
  - Introducción
  - Fundamentos de RESTful APIs.
  - Creación de un servicio REST en PHP con Slim Framework o programación sin framework.
  - Consumo de APIs REST desde PHP.
  - Autenticación y seguridad en APIs REST.
- Composer
  - Instalador de paquetes para PHP.
  - Se utiliza para instalar el framework de **Slim**
  - Descargar y colocar en el PATH
  - <https://getcomposer.org/download/>

# Contenidos II

- Slim

# **Servicios REST**

# Introducción

- El estilo REST *es una forma ligera de crear Servicios Web.*
- Se basan en las URLs.
- Proporcionan acceso a URLs para obtener información o realizar alguna operación.
- Son interesante para utilizar con **peticiones** de tipo **AJAX** y para acceder con **dispositivos con pocos recursos.**

# Características

- Sistema cliente / servidor.
- No hay estado → sin sesión.
- Soporta un sistema de caché
- ***Cada recurso tendrá una única dirección de red.***
- Sistema por capas.
- Variedad de formatos:
  - XML, HTML, text plain, JSON, etc.

# Recursos

- Un recurso REST es cualquier cosa que sea direccionable a través de la Web.
- Algunos ejemplos de recursos REST son:
  - Una noticia de un periódico
  - La temperatura de Alicante a las 4:00pm

# Algunos formatos soportados

Formato	Tipo MIME
Texto plano	text/plain
HTML	text/html
XML	application/xml
JSON	application/json



# URI

- Una URI, o **Uniform Resource Identifier**, en un servicio web RESTful es un hiper-enlace a un recurso, y es la única forma de intercambiar representaciones entre clientes y servidores.
- Un servicio web RESTful expone un conjunto de recursos que identifican los objetivos de la interacción con sus clientes.

# Formato de las peticiones

- Las peticiones REST tienen un formato con este:
- `http://localhost:8080/app/trabajadores/101`
- **trabajadores: representa un recurso.**
- **101:** El identificador del Trabajador, es el equivalente a `.../trabajadores?id=101`
- La URL de REST está orientada a recursos y localiza un recurso.

# Verbos REST

- Los verbos nos permiten llevar a cabo acciones con los recursos.
- Se asocian con las operaciones **CRUD**.
  - **GET**: Obtener información sobre un recurso. El recurso queda identificado por su URL. **Operación read**.
  - **POST**: Publica información sobre un recurso. **Operación create**.
  - **PUT**: Incluye información sobre recursos en el Servidor. **Operación update**.
  - **DELETE**: Elimina un recurso en el Servidor. **Operación delete**.

# REST vs SOAP

	REST	SOAP
Características	<p>Las operaciones se definen en los mensajes.</p> <p>Una dirección única para cada instancia del proceso.</p> <p>Cada objeto soporta las operaciones estándares definidas.</p> <p>Componentes débilmente acoplados.</p>	<p>Las operaciones son definidas como puertos WSDL.</p> <p>Dirección única para todas las operaciones.</p> <p>Múltiple instancias del proceso comparten la misma operación.</p> <p>Componentes fuertemente acoplados.</p>
Ventajas declaradas	<p>Bajo consumo de recursos.</p> <p>Las instancias del proceso son creadas explícitamente.</p> <p>El cliente no necesita información de enrutamiento a partir de la URI inicial.</p> <p>Los clientes pueden tener una interfaz “listener” (escuchadora) genérica para las notificaciones.</p> <p>Generalmente fácil de construir y adoptar.</p>	<p>Fácil (generalmente) de utilizar.</p> <p>La depuración es posible.</p> <p>Las operaciones complejas pueden ser escondidas detrás de una fachada.</p> <p>Envolver APIs existentes es sencillo</p> <p>Incrementa la privacidad.</p> <p>Herramientas de desarrollo.</p>
Posibles desventajas	<p>Gran número de objetos.</p> <p>Manejar el espacio de nombres (URIs) puede ser engorroso.</p> <p>La descripción sintáctica/semántica muy informal (orientada al usuario).</p> <p>Pocas herramientas de desarrollo.</p>	<p>Los clientes necesitan saber las operaciones y su semántica antes del uso.</p> <p>Los clientes necesitan puertos dedicados para diferentes tipos de notificaciones.</p> <p>Las instancias del proceso son creadas implícitamente.</p>

# ¿Dónde es útil REST?

- El servicio Web no tiene estado.
- Tanto el productor como el consumidor del servicio conocen el contexto y contenido que va a ser comunicado
- El ancho de banda es importante y necesita ser limitado.
  - REST es particularmente útil en dispositivos con escasos recursos como PDAs o teléfonos móviles
- Los desarrolladores pueden utilizar tecnologías como **AJAX**

# ¿Dónde es útil SOAP?

- Se establece un contrato formal para la descripción de la interfaz que el servicio ofrece → WSDL.
- La arquitectura necesita manejar procesamiento asíncrono e invocación.

# Composer

# Instalar composer

- Composer es un gestor de paquetes
- Descargar instalador para Windows
- Seleccionar la ruta de instalación
- Indicar la ruta donde se encuentra el ejecutable de PHP
- Si tenemos algún problema en la instalación, donde hace referencia a OpenSSL, descargar el certificado de:
  - <https://curl.se/docs/caextract.html>



# Instalar composer

- 1. Descargar el certificado actualizado: **cacert-2025-07-15.pem**
- 2. Configurar PHP para usar ese archivo
  - Abre tu archivo php.ini (el archivo ini que estamos referenciando, lo podemos ver con el comando: `php --ini`)
- Añade o modifica estas líneas, en el php.ini
- **[openssl]**
  - **openssl.cafile = "C:\php8\extras\ssl\cacert.pem"**

# Instalar composer

- **[curl]**
  - **curl.cainfo = "C:\php8\extras\ssl\cacert.pem"**
  - Asegúrate de que la ruta sea correcta y que el archivo exista.
- 3. Verifica que la extensión OpenSSL esté habilitada
  - En el mismo **php.ini**, asegúrate de que esta línea esté activa (sin ; al inicio):
  - **extension=openssl**
- 4. Activar la extensión zip en el php.ini
  - **extension=zip**
- 5. Reinicia tu terminal y vuelve a intentar instalar **composer**

# Errores: solución 1

- Si falla el instalador, probar a descargar el archivo php directamente, y se guarda en la carpeta raíz de php8.
- <https://getcomposer.org/installer>
- El archivo descargado se guarda como: **composer-setup.php**
- Este se ejecuta en modo comando con:
  - php composer-setup.php
  - **Puede que también falle porque va a intentar verificar el certificado y todavía no esta PHP preparado.**

# Errores: soluciones 2

- <https://getcomposer.org/composer-stable.phar>
- Descargar y guardar en una carpeta: C:\composer
- Ejecutar el comando:
  - **php composer.phar --versión**
  - Tenemos que ver la versión: 2.x.x
  - Crear un archivo **composer.bat** en la misma carpeta con este contenido:
    - **@php "%~dp0composer.phar" %\***
- ***Esto permite que al escribir composer en CMD, se ejecute el .phar con PHP***
- Añadir al **PATH**, la carpeta: **C:\composer**
- **Probar en una consola:**
- **composer --version**

C:\php8

C:\apache\Apache24\bin

C:\composer

# Errores soluciones 2

- Al ejecutar desde una consola nueva: **composer --version**

```
C:\>composer --version
Composer version 2.8.10 2025-07-10 19:08:33
PHP version 8.2.29 (C:\php8\php.exe)
Run the "diagnose" command to get more detailed diagnostics output.
```

Con esto ya tenemos instalado **composer** y no hace falta utilizar el instalador

# Instalar slim con composer

- Es el framework que vamos a utilizar para desarrollar servicios REST con PHP.
- Nos situamos en la carpeta donde vamos a desarrollar el proyecto de PHP.
- Podemos comprobar los certificados con:
  - `php -r "print_r(openssl_get_cert_locations());"`
- Disponemos de la opción:
  - **composer diagnose**
- Si composer continúa fallando, podemos limpiar la cache de composer:
  - **composer clear-cache**
  - **composer config --global --unset disable-tls**

# Instalar Slim con composer

- Hay veces que composer continúa fallando
- Desactivar la comprobación de certificados:
  - **composer config --global disable-tls true**
  - **composer require slim/slim:"^4.0"**
- O También: **composer require slim/slim → la última estable**
- Para una concreta: **composer require slim/slim:4.0.0**
- Después volver a activar:
  - **composer config --global disable-tls false**

# Situación en el proyecto

```
mi-proyecto/  
├── frontend/           ← Vue.js (SPA)  
│   ├── src/  
│   ├── public/  
│   └── package.json  
├── backend/           ← Backend PHP con Slim  
│   ├── slim/          ← Código Slim Framework  
│   │   ├── public/    ← Punto de entrada (index.php)  
│   │   ├── src/       ← Controladores, rutas, middlewares  
│   │   ├── vendor/    ← Dependencias Composer  
│   │   └── composer.json  
│   ├── beans/         ← Clases de entidad (POJOs estilo Java)  
│   └── daos/          ← Acceso a datos (consultas SQL, etc.)
```



# Otras dependencias de slim

- `composer require slim/psr7`      # Para manejar peticiones/respuestas
- `composer require nyholm/psr7`      # Alternativa ligera a slim/psr7
- `composer require vlucas/phpdotenv` # Para usar archivos .env
- `composer require selective/basepath`

# nyholm/psr7

- **Implementa PSR-7:** Define interfaces para objetos como Request, Response, Stream, URI, etc.
- **Compatible con PSR-17:** Incluye fábricas para crear esos objetos de forma estándar.
- **Optimizada para rendimiento:** Tiene menos líneas de código que otras alternativas como Guzzle o Laminas, y ofrece mejor rendimiento en benchmarks.
- **Ideal para Slim Framework:** Slim necesita una implementación PSR-7 para funcionar correctamente. nyholm/psr7 es una opción moderna y eficiente.

# autoload

- Con composer se puede configurar la carga automática de clases.
- En el fichero **composer.json** se añade esto debajo de las dependencias:

```
"autoload": {  
    "psr-4": {  
        "App\\": "src/"  
    }  
}
```

- Con esto estamos diciendo que todas las clases que empiezan con el namespace **App\\** se encuentran en la carpeta **src/**
  - App\\Controllers\\HomeController → estará en src/Controllers/HomeController
  - Ojo con las mayúsculas, aunque estemos en Windows
- Lanzar el comando:
  - **composer dump-autoload**
- ***Cada vez que añadimos una nueva ruta del algún servicio.***

# .htaccess

- RewriteEngine On
- RewriteCond %{REQUEST\_FILENAME} !-f
- RewriteCond %{REQUEST\_FILENAME} !-d
- RewriteRule ^ index.php [QSA,L]

# Slim

FrameWork de PHP para desarrollo de Servicios REST

Método HTTP	Ruta	Acción
GET	/categorias	Obtener todas las categorías
GET	/categorias/{id}	Obtener una categoría por ID
POST	/categorias	Crear una nueva categoría
PUT	/categorias/{id}	Actualizar una categoría
DELETE	/categorias/{id}	Eliminar una categoría