



# Cómo crear e interactuar con una API

PHP



Qualentum Lab

# Cómo crear e interactuar con una API

Las APIs juegan un papel fundamental en los desarrollos de backend y de aplicaciones web hoy en día. Son imprescindibles para el uso y tratamiento de información. A lo largo de este fastbook se detalla cómo crear e interactuar con una API en diferentes frameworks en PHP, no sin antes repasar los fundamentos de la arquitectura REST.

¡Empezamos!

*Autor: Jesús Donoso*

-  [Introducción](#)
-  [Tipos de API](#)
-  [API REST](#)
-  [Creación de REST API en diferentes frameworks de PHP](#)
-  [Conclusión](#)
-  [Enlaces de interés](#)

# Introducción



---

Una API (interfaz de programación de aplicaciones) es un conjunto de reglas y protocolos que facilitan la comunicación entre aplicaciones informáticas.

Se trata de un mecanismo que permite que una aplicación se comunique con otra de manera estandarizada.

En un sentido más sencillo, una API establece cómo **las aplicaciones pueden interactuar con un software o servicio específico**. Proporciona un conjunto de funciones y directrices que posibilitan el acceso y el intercambio de datos entre aplicaciones de manera eficiente y uniforme.

---

**Estas interfaces son fundamentales en el ámbito del desarrollo de software, ya que posibilitan la integración de distintos sistemas, servicios y plataformas, fomentando la interoperabilidad entre aplicaciones.**

---

Pueden adoptar diversas formas como APIs web, bibliotecas de funciones, o incluso interfaces de hardware. A continuación, vamos a ver varias razones clave por las que las APIs juegan un papel tan relevante en el desarrollo de software:

- Permiten la interoperabilidad entre diferentes sistemas y aplicaciones, ya que proporcionan un estándar para la comunicación y el intercambio de datos. Esto facilita la integración de diversas tecnologías y plataformas.
- Favorecen la reutilización de código al ofrecer funciones y servicios a través de una interfaz estandarizada. Esto posibilita la utilización de componentes de software en múltiples aplicaciones sin necesidad de reescribir todo el código.
- Contribuyen al desarrollo ágil al permitir que equipos de desarrollo trabajen de manera simultánea en módulos específicos de una aplicación. Esto acelera el proceso de desarrollo y mejora la eficiencia.
- Contribuyen a la escalabilidad al posibilitar la construcción modular de aplicaciones. Los desarrolladores pueden agregar o cambiar componentes sin afectar el conjunto completo de la aplicación.
- Fomentan la innovación rápida al permitir que los desarrolladores aprovechen las funcionalidades de servicios externos sin comprender completamente su implementación interna. Esto acelera el desarrollo y facilita la creación de aplicaciones más avanzadas.
- Simplifican la integración de servicios de terceros, como servicios de pago, redes sociales, etc. Las aplicaciones pueden conectarse eficientemente a recursos externos mediante sus APIs.

- Abstraen la complejidad interna de una aplicación al proporcionar una interfaz clara y documentada. Esto permite a los desarrolladores utilizar funcionalidades complejas sin entender todos los detalles internos.
- Ofrecen capas de seguridad y control al permitir que los desarrolladores establezcan restricciones en el acceso a los datos y funciones de una aplicación, asegurando un uso conforme.

Podríamos decir entonces que **una API actúa como un mediador** que posibilita a las aplicaciones intercambiar información y realizar operaciones de manera coherente y estandarizada. Por lo tanto, facilita la integración y colaboración entre sistemas y servicios diversos.

# Tipos de API



Existen varios tipos de API, cada uno diseñado para **propósitos específicos y adaptados a diferentes contextos de desarrollo**, por ejemplo:

- Las **API web** utilizan protocolos estándar de la web, como HTTP o HTTPS, para favorecer la comunicación entre sistemas. Pueden proporcionar acceso a recursos y servicios a través de endpoints bien definidos. API REST y SOAP son APIs web.
- Las **API REST** (transferencia de estado representacional) utilizan los métodos HTTP para realizar operaciones CRUD sobre recursos, centrándose en la simplicidad, escalabilidad y eficiencia.
- Las **API SOAP** (protocolo de acceso a objeto simple) son protocolos de comunicación que permiten la creación de servicios web. Utilizan XML para la representación de datos y pueden emplear varios protocolos de transporte, incluyendo HTTP y SMTP. Su enfoque principal está en la seguridad y la transacción.
- GraphQL** es un lenguaje de consulta y una especificación para API que permite a los clientes especificar la estructura exacta de los datos que necesitan. A diferencia de REST, donde la respuesta está predefinida por el servidor, GraphQL ofrece flexibilidad en las consultas.

- JSON-RPC** y **XML-RPC** son protocolos de comunicación que permiten la ejecución de funciones remotas mediante la transmisión de datos en formato JSON o XML, respectivamente. Facilitan la creación de servicios que pueden ser llamados de manera remota.
- gRPC** es un framework de código abierto desarrollado por Google que utiliza el protocolo HTTP/2 para la comunicación entre servicios. Se basa en el intercambio de datos en formato protobuf (Protocol Buffers), ofreciendo eficiencia en tamaño y rendimiento.
- OData**, un protocolo estándar basado en REST, define convenciones para la creación y el consumo de servicios web. Facilita la creación de APIs que son accesibles y comprensibles.

Cada tipo de API tiene sus propias ventajas y consideraciones, y la elección dependerá de los requisitos específicos de tu proyecto, la arquitectura preferida y las necesidades de comunicación entre sistemas.

# API REST



En el ámbito del desarrollo web, la arquitectura RESTful ha surgido como un enfoque esencial para diseñar servicios web que facilitan la comunicación eficiente y escalable entre sistemas. El término 'RESTful' hace referencia a la implementación de los principios de la arquitectura REST (transferencia de estado representacional) en el desarrollo de servicios web. REST establece un conjunto de principios y restricciones que orientan la creación de APIs web coherentes y eficientes.

## Principios fundamentales de la arquitectura REST

1

**Recursos identificables:** en el contexto de la arquitectura RESTful, cada entidad significativa se considera un recurso, y cada recurso se identifica mediante un único URI.

2

**Representación del estado:** los recursos poseen un estado que puede ser representado en diversos formatos, como JSON o XML. Estas representaciones se transfieren entre el cliente y el servidor.

3

**Operaciones CRUD con métodos HTTP:** las operaciones relacionadas con los recursos siguen los métodos estándar de HTTP, como GET para obtener información y POST para crear nuevos recursos.

4

**Sin estado (stateless):** REST sigue el principio de ser sin estado, lo que implica que cada solicitud del cliente al servidor contiene toda la información necesaria para comprender y procesar la solicitud.

5

**HATEOAS (*Hypermedia As The Engine Of Application State*):** HATEOAS fomenta la inclusión de enlaces hipertextuales en las representaciones, permitiendo que el cliente navegue dinámicamente entre estados de la aplicación.

6

**Uniformidad de la interfaz:** la interfaz para interactuar con los recursos es uniforme, utilizando URLs y métodos HTTP estándar para acceder y manipular recursos.

7

**Sistema cliente-servidor:** la arquitectura REST separa la interfaz de usuario (cliente) de la lógica del servidor, facilitando la evolución independiente de ambos componentes.

8

**Cacheable:** las respuestas del servidor pueden almacenarse en caché, mejorando la eficiencia al reducir la necesidad de repetir solicitudes similares.

Estos principios fundamentales no solo proporcionan coherencia en el diseño de APIs, sino que también permiten la construcción de servicios web que son escalables, flexibles y fáciles de entender. Al adoptar la arquitectura RESTful, se fomenta la interoperabilidad entre sistemas y se simplifica el desarrollo de aplicaciones distribuidas.

## Métodos HTTP

---

**Los métodos HTTP, como GET, POST, PUT, PATCH y DELETE, son comandos fundamentales utilizados en la comunicación entre clientes y servidores en la web.**

---

Cada uno cumple un propósito específico y está asociado con operaciones CRUD (crear, leer, actualizar y eliminar) en una API RESTful. A continuación, vamos a compartirte una descripción de estos métodos y su relación con las operaciones CRUD.

- **GET (obtener)**

---

El método GET se emplea para recuperar información de un recurso identificado por un URI.

Es una solicitud de solo lectura y no modifica el estado del recurso en el servidor. Correspondiente a la operación de 'Leer', se utiliza para obtener datos de un recurso.

```
GET /productos/123
```

- **POST (enviar)**

---

El método POST se utiliza para enviar datos al servidor con el propósito de crear un nuevo recurso.

Puede llevar datos en el cuerpo de la solicitud. Corresponde a la operación de 'Crear', utilizado para agregar un nuevo recurso.

```
POST /productos
```

- **PUT (actualizar)**

---

El método PUT se utiliza para actualizar un recurso o crear uno nuevo si no existe.

Requiere que se envíen todos los datos del recurso, ya que sobrescribe completamente el recurso existente. Correspondiente a la operación de 'Actualizar', se utiliza para modificar un recurso existente.

```
PUT /productos/123
```

- **PATCH (parche)**

---

Similar a PUT, pero se utiliza para actualizar parcialmente un recurso, es decir, solo se envían los datos que se desean modificar.

Corresponde a la operación de 'Actualizar', utilizada para modificar parcialmente un recurso existente.

```
PATCH /productos/123
```

- **DELETE (eliminar)**

---

El método DELETE se aplica para eliminar un recurso identificado por un URI.

Corresponde a la operación de 'Eliminar' o borrar un recurso existente.

```
DELETE /productos/123
```

Como vemos, estos métodos HTTP proporcionan una interfaz consistente y predecible para interactuar con recursos en una API RESTful, cosa que ayuda al desarrollo y la comprensión de sistemas distribuidos.

# Creación de REST API en diferentes frameworks de PHP



Qualentum Lab

En el desarrollo de sistemas web modernos la creación de APIs RESTful se ha vuelto esencial para la comunicación entre aplicaciones. Por ello, en este apartado se detalla cómo crear una API REST en Symfony, Laravel y Codeigniter, frameworks de desarrollo en PHP muy utilizados hoy día.

1

## Symfony

La creación de una REST API en Symfony implica seguir una serie de pasos que aprovechan las funcionalidades de este framework. Aquí te los compartimos a modo de guía:

Instala Symfony mediante Composer:

```
composer create-project symfony/skeleton nombre_del_proyecto
```

Para la creación de controladores y otros elementos aplica el siguiente comando:

```
composer require maker
```

A la hora de crear una entidad que represente los datos que hay que gestionar utiliza MakerBundle:

```
php bin/console make:entity
```

La configuración de la conexión de la base de datos se realiza en *config/packages/doctrine.yaml* y se ejecutan las migraciones:

```
php bin/console doctrine:database:create  
php bin/console doctrine:migrations:migrate
```

Para crear un controlador específico para la API usa MakerBundle:

```
php bin/console make:controller
```

Puedes definir acciones para realizar operaciones CRUD, como la recuperación de datos, así:

```
// src/Controller/Api/ProductController.php
namespace App\Controller\Api;

use App\Entity\Product;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\JsonResponse;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\Routing\Annotation\Route;

/**
 * @Route("/api/products")
 */
class ProductController extends AbstractController
{
    /**
     * @Route("/", name="api_product_list", methods={"GET"})
     */
    public function list(): JsonResponse
    {
        $products = $this->getDoctrine()->getRepository(Product::class)->findAll();

        $data = [];
        foreach ($products as $product) {
            $data[] = [
                'id' => $product->getId(),
                'name' => $product->getName(),
                'price' => $product->getPrice(),
                // Otros campos según tu entidad Product
            ];
        }
        return $this->json($data);
    }
}
```

Este ejemplo básico proporciona acciones CRUD a través de una API REST en Symfony, pero ten en cuenta que no es más que es un punto de partida y que estas acciones se deben adaptar a los requisitos específicos de la aplicación. Además, es importante considerar aspectos de seguridad, autenticación y validación según las necesidades de cada caso.

Las rutas para las acciones se establecen en `config/routes/api.yaml`:

```
api_controller:  
    path: /api  
    controller: App\Controller\ApiController::index  
    methods: [GET]
```

Y para convertir objetos a formato JSON emplea el componente Serializer:

```
// En el controlador  
return $this->json($data);
```

2

Laravel

---

Laravel, un poderoso framework PHP, facilita este proceso al proporcionar herramientas y convenciones que simplifican la construcción de APIs robustas y escalables.

La siguiente guía que te compartimos también detalla los pasos necesarios para la creación de una API completa utilizando Laravel.

Se inicia el proceso creando un nuevo proyecto Laravel mediante el uso de Composer:

```
composer create-project --prefer-dist laravel/laravel nombre_del_proyecto  
cd nombre_del_proyecto
```

Para la configuración de la conexión a la base de datos se utiliza en el archivo .env, especificando detalles como el host, puerto, nombre de la base de datos, usuario y contraseña. Posteriormente, hay que ejecutar las migraciones para crear las tablas correspondientes:

```
php artisan migrate
```

Luego, se genera un modelo para la entidad deseada, por ejemplo, Product:

```
php artisan make:model Product -a
```

---

**Este comando automáticamente crea el modelo, la migración, el controlador y las vistas relacionadas.**

---

A continuación, se modifica el controlador generado en `app/Http/Controllers/ProductController.php` para implementar las operaciones CRUD básicas. Estas operaciones permiten listar, mostrar, crear, actualizar y eliminar productos:

```
namespace App\Http\Controllers;
use App\Models\Product;
use Illuminate\Http\Request;

class ProductController extends Controller
{
    public function index()
    {
        $products = Product::all();
        return response()->json($products);
    }

    public function show($id)
    {
        $product = Product::find($id);
        return response()->json($product);
    }

    public function store(Request $request)
    {
        $product = Product::create($request->all());
        return response()->json($product, 201);
    }

    public function update(Request $request, $id)
    {
        $product = Product::find($id);
        $product->update($request->all());
        return response()->json($product, 200);
    }

    public function destroy($id)
    {
        $product = Product::find($id);
        $product->delete();
        return response()->json(null, 204);
    }
}
```

Las rutas de la API se definen en el archivo `routes/api.php`. Se establece una ruta de recursos para el controlador `ProductController`:

```
// routes/api.php  
use App\Http\Controllers\ProductController;  
Route::resource('products', ProductController::class);
```

Para realizar pruebas mediante herramientas como Postman o cURL, hay que acceder a las rutas de la API, por ejemplo, `http://localhost:8000/api/products`. Además, es aconsejable utilizar herramientas de documentación como Laravel API Documentation Generator o Swagger para documentar la API de manera efectiva.

Un apunte más: como en el caso de Symfony, te hemos presentado un ejemplo básico, pero **se puede ampliar según los requisitos específicos del proyecto**, incorporando características adicionales como validaciones, autenticación y autorización a medida que se avanza en el desarrollo.



Y no olvides que Laravel proporciona una documentación completa que puede ser consultada durante el proceso de desarrollo. Puedes acceder a ella desde haciendo clic en [Laravel Documentation](#).

3

## Codeigniter

Por último, en este apartado vamos a estudiar los pasos básicos para la creación de una API REST pero utilizando ahora CodeIgniter. Como sabes, este es otro framework de desarrollo web en PHP que ofrece una estructura ágil y sencilla a la hora de construir aplicaciones robustas.

Para iniciar el proyecto se utiliza Composer, nuestra herramienta de gestión de dependencias para PHP. Y se crea un nuevo proyecto Codeigniter mediante el siguiente comando:

```
composer create-project codeigniter4/appstarter nombre_del_proyecto  
cd nombre_del_proyecto
```

---

**Este comando descarga e instala Codeigniter junto con sus dependencias en el directorio especificado.**

---

La conexión a la base de datos se configura en el archivo *app/config/database.php*, donde se establecen detalles como el host, el nombre de la base de datos, el usuario y la contraseña.

```
$active_group = 'default';  
$query_builder = true;  
  
$db['default'] = [  
    'DSN'      => '',  
    'hostname' => 'localhost',  
    'username' => 'tu_usuario',  
    'password' => 'tu_contraseña',  
    'database' => 'tu_base_de_datos',  
    // ...  
];
```

Luego, se definen las rutas de la API en el archivo `app/config/Routes.php`, estableciendo las rutas para las operaciones CRUD deseadas.

```
$routes->get('products', 'Api\Products::index');
$routes->get('products/(:num)', 'Api\Products::show/$1');
$routes->post('products', 'Api\Products::create');
$routes->put('products/(:num)', 'Api\Products::update/$1');
$routes->delete('products/(:num)', 'Api\Products::delete/$1');
```

El controlador llamado 'Products' se crea en el directorio `app/Controllers/Api` para manejar las operaciones CRUD de la API. Este controlador extiende la clase `BaseController`. namespace App\Controllers\Api;

```
use App\Controllers\BaseController;

class Products extends BaseController
{
    public function index()
    {
        // Lógica para obtener todos los productos
    }

    public function show($id)
    {
        // Lógica para obtener un producto por ID
    }

    public function create()
    {
        // Lógica para crear un nuevo producto
    }

    public function update($id)
    {
        // Lógica para actualizar un producto por ID
    }

    public function delete($id)
    {
        // Lógica para eliminar un producto por ID
    }
}
```

A continuación, se genera un modelo llamado 'ProductModel' en el directorio `app/Models` para manejar las consultas a la base de datos, pero para entenderlo mejor, observa el siguiente ejemplo de cómo podría ser el modelo:

```
namespace App\Models;

use CodeIgniter\Model;

class ProductModel extends Model
{
    protected $table = 'products';
    protected $primaryKey = 'id';
    protected $allowedFields = ['name', 'price'];

    // Métodos para operaciones CRUD
}
```

# Conclusión



Desde el inicio de este fastbook, hemos querido incidir en la importancia de las APIs en el mundo actual del desarrollo. De ahí que API REST se haya convertido en nuestro cabeza de cartel.

Hemos explorado cómo trabajar con API REST en PHP, centrándonos en **tres frameworks muy utilizados**: Symfony, Laravel y CodeIgniter. Te hemos proporcionado ejemplos específicos para cada framework, destacando sus componentes particulares. Pero esto solo ha sido una exposición teórica, una mera aproximación: ahora te toca a ti descubrir todo su potencial de manera práctica para mejorar la eficiencia de tus desarrollos.

# Enlaces de interés



- 
- Documentación oficial de Symfony:  
[\*https://symfony.com/doc/current/index.html\*](https://symfony.com/doc/current/index.html).
  - Documentación oficial de Laravel: [\*https://laravel.com/docs/10.x\*](https://laravel.com/docs/10.x).
  - Documentación oficial de Codeigniter:  
[\*https://codeigniter.es/documentacion.php\*](https://codeigniter.es/documentacion.php).

**¡Enhорabuena! Fastbook superado**



[Qualentum.com](http://Qualentum.com)