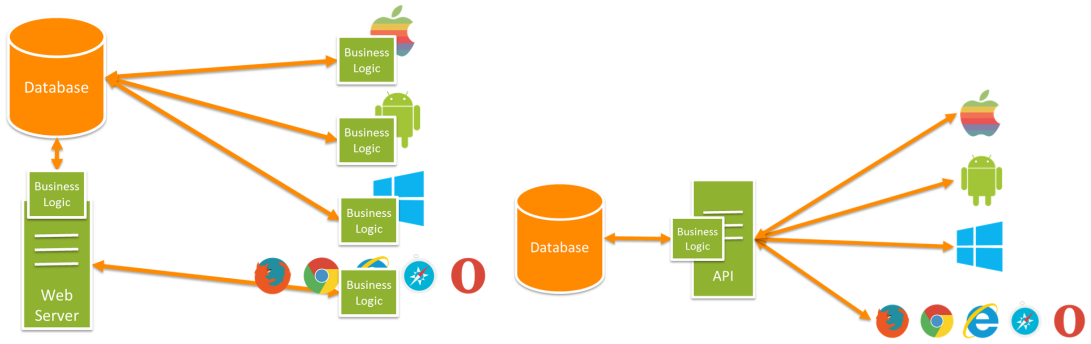
Servicios REST

Sitio: Aula virtual do IES Pazo da Mercé
Curso: Desenvolvemento Web Contorno Servidor (24-25)
Libro: Servicios REST
Impreso por: Lara Comesaña Varela
Data: mércores, 15 de xaneiro de 2025, 10:54 AM

Táboa de contidos

1. ¿Qué es un servicio web?

¿Qué es un servicio web?



Arquitectura de una aplicación sin Web API vs con Web API

En ocasiones las aplicaciones que desarrollamos necesitarán **compartir información con otras aplicaciones**.

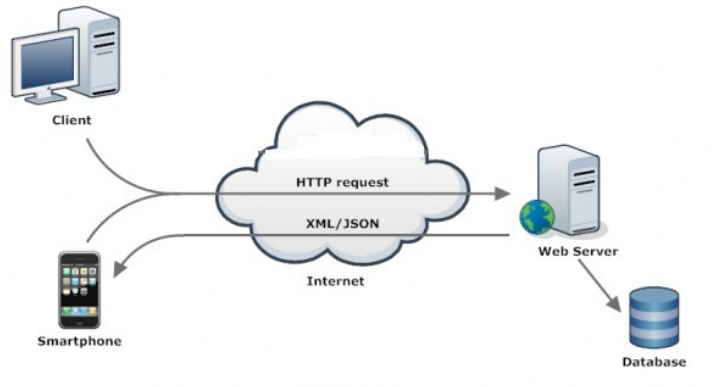
Por ejemplo, podría ser que tengamos un cliente con mucha facturación que nos solicite poder acceder a los productos que le vendemos para poder realizar una integración en su ERP o tal vez decidimos hacer una aplicación móvil para vender los productos de nuestra tienda. En estos dos casos, dos **aplicaciones externas necesitan acceder a parte de los datos que mostramos en nuestra web**.

Una opción para compartir esta información podría ser dar acceso a la base de datos a las otras aplicaciones pero, por norma general, esto **no es**

recomendable. Cualquier usuario/app que tenga acceso a la base de datos, tendría capacidad para hacer/deshacer lo que quisiera. Podríamos limitar las operaciones que tiene el usuario de base de datos pero, ¿nos interesa que puedan verse desde fuera datos internos? Por ejemplo, nosotros mostramos el precio de nuestros productos pero los clientes no saben el margen de beneficio real que tenemos. Los usuarios y perfiles de base de datos están pensandos para restricciones a nivel de eschema y table pero no llegan a niveles tan detallados.

Por otro lado, los desarrolladores de la otra aplicación **deberían conocer perfectamente la estructura de la BBDD** y las relaciones entre tablas para obtener todos los datos que necesiten del producto. Si tenemos ya una capa de negocio que realiza estas operaciones, ¿es óptimo **re-codificar** esa lógica? Y si queremos hacer un **cambio en la estructura de la base de datos**, ¿tenemos que avisar a la otra empresa para que cambie su src?

Un servicio web es una aplicación que se encuentra en el lado servidor y permite que otra aplicación cliente conecte con ella a través de Internet para el intercambio de información utilizando el protocolo HTTP. Una de las principales características de los servicios web es que no es necesario que ambas aplicaciones (servidor y cliente) estén escritas en el mismo, lo que hace que la interoperabilidad sea máxima. Por ejemplo, podríamos crear un servicio web en PHP y utilizarlo conectándonos desde una aplicación móvil con Android, desde otra aplicación programada en Java o incluso desde otro servicio web escrito con .NET.



Rest WebService

Como los servicios web utilizan los protocolos **HTTP y HTTPS** y estos puertos suelen estar abiertos por defecto en las organizaciones que alojan páginas web, no es necesario abrir puertos en el firewall o hacer configuraciones complicadas para compartir datos entre aplicaciones. Como nosotros programamos el servicio web, no es necesario dar accesos a servicios y siempre podemos **controlar los datos que compartimos** con la aplicación remota, qué datos le **permitimos modificar** e incluso que política de **logging** llevamos a cabo.

2. Ventajas y desventajas de un servicio REST

Ventajas y desventajas de un servicio REST

Durante 5 minutos, recopila las ventajas y desventajas que tiene el uso de servicios web como mecanismo de intercambio de información entre sistemas informáticos vs compartir la fuente de datos en crudo.

3. REST Web Service

REST Web Service

Los Servicios Web REST son Servicios Web que cumplen una serie de requisitos según un patrón de arquitectura definida hacia el año 2000 y que se ha extendido siendo el patrón predominante a la hora de implementar este tipo de aplicaciones.

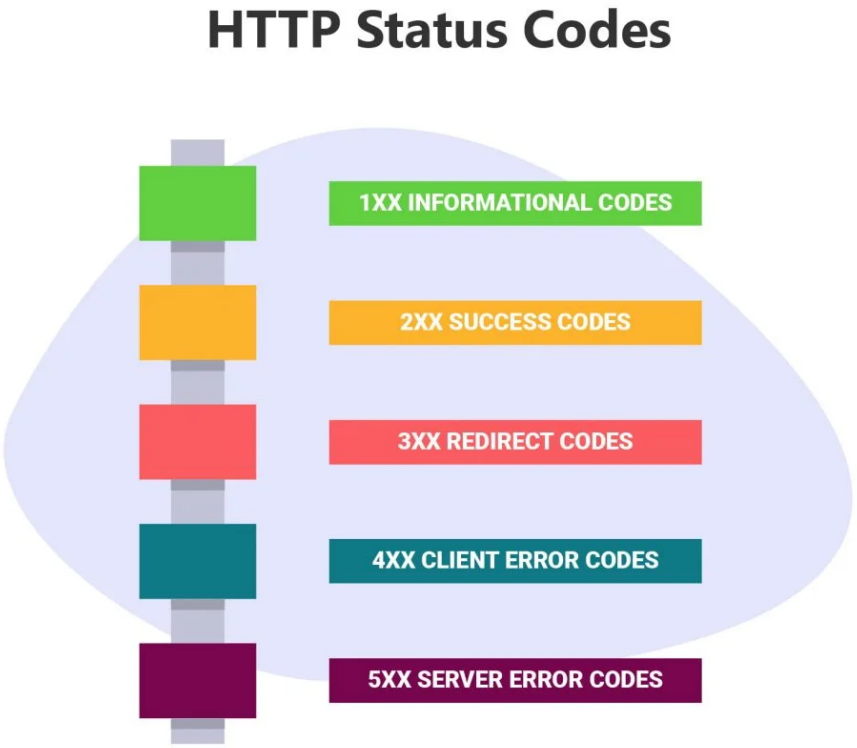
Básicamente consiste en seguir una serie de reglas que definen dicha arquitectura. Entre ellas están el uso del procotolo HTTP por ser el más extendido a lo largo de Internet en la actualidad. Además, cada recurso del servicio web tiene que ser identificado por una dirección web (una URL (Uniform Resource Locator)) siguiendo una estructura determinada. Además, la respuesta tendrá que tener una estructura determinada en forma de texto que normalmente vendrá en alguno de los formatos abiertos más conocidos como XML o JSON.

Una característica importante de los servicios REST es que es un protocolo sin estado (stateless)

Resource	Verb	Expected Outcome	Response Code
/Products	GET	A list of all products in the system	200/OK
/Products?Colour=red	GET	A list of all products in the system where the colour is red	200/OK
/Products	POST	Creation of a new product	201/Created
/Products/81	GET	Product with ID of 81	200/OK
/Products/881(a product ID which does not exist)	GET	Some error message	404/Not Found
/Products/81	PUT	An update to the product with an ID of 81	204/No Content
/Products/81	DELETE	Deletion of the product with an ID of 81	204/No Content
/Customers	GET	A list of all customers	200/OK

4. Códigos de respuesta

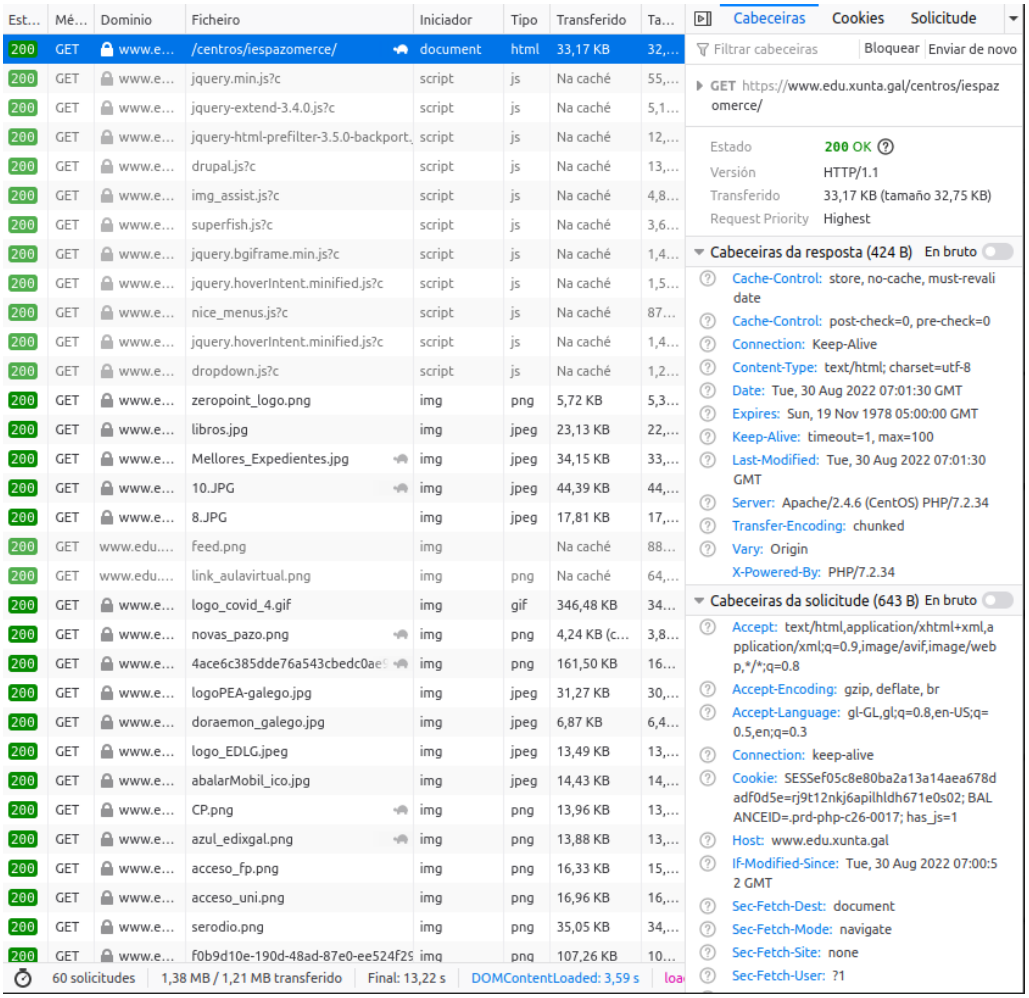
Códigos de respuesta



HTTP es un protocolo de comunicación basado en el uso de mensajes de texto para gestionar peticiones a un servidor y para las **respuestas** desde éste. Se trata de un protocolo que es síncrono, es decir, la respuesta a la petición ocurre inmediatamente. Tanto las peticiones (request) o las respuestas (response) siguen un vocabulario y una sintaxis definida. Por ejemplo, las peticiones generalmente están basadas en métodos verbos, que se denominan métodos, como GET o POST.

Una vez que hemos enviado una petición a un servidor este nos devolverá una **respuesta** que vendrá establecida por una **codificación de posibles respuestas estandarizadas**. Usando las herramientas para desarrolladores que nos ofrece Chrome o Firefox podemos comprobar el estado de las respuestas a las diferentes peticiones que ocurren al cargar una página web.

Si abrimos las herramientas de desarrollador (F12) pestaña Network podemos ver que, tras cargar una página con esta pestaña abierta, se carga una lista con todas las peticiones que se hacen para cargar la página web solicitada. En la primera columna podemos ver los códigos de respuesta de dichas peticiones.



Podemos hacer una clasificación general de los códigos de respuesta guiandonos por el dígito posicionado en la centena:

2xx – Indican que la petición se ha satisfecho correctamente

Serán habituales el código **200** (todo fue bien y se ha encontrado el recurso y procesado la respuesta), el 201 (todo OK se ha creado el nuevo recurso) o 204 (todo bien pero no se ha devuelto ningún contenido).

3xx- indican que el recurso solicitado está en otra URI diferente

Son comunes la **301** (el recurso se ha cambiado de ubicación definitivamente y nos indica la nueva ubicación), el **302** (el recurso que solicitas se ha movido temporalmente a este otro recurso).

4xx – indican códigos de error en el lado del cliente

En este caso el uso de **400** nos indica que hay un error en el formato de la petición o que es demasiado largo, **401** para decirnos que no tenemos autorización para ese recurso (aunque podríamos validarnos) o **403**, para indicarnos que el acceso al recurso no está permitido y el servidor ha rechazado nuestra petición. También es relevante el error 405 que nos indica que el método no se admite en la API o el muy conocido **404** para indicarnos que no se encuentra el recurso.

5xx- indican códigos de error en el lado del servidor

En este último grupo están el error 500, cuando ha habido algún error interno del servidor, 502, cuando el servidor hace de Gateway o proxy y ha recibido una respuesta de error a donde quiera que intentase conectarse o el 503 cuando el servicio en el servidor no está accesible, normalmente debido a una sobrecarga o algún problema similar.

Podemos consultar los códigos de respuesta en la página de Mozilla.

Para establecer un código de respuesta en nuestra página PHP podemos utilizar las funciones header() o http_response_code().

5. Métodos de una petición

Métodos de una petición

HTTP Verb	CRUD	Sample	Description	Result

GET	Read	/books	list all books. Pagination, sorting and filtering is used for big lists.	**200 (OK)**
		/books/{id}	get specific book by id	**200 (OK)** **404 (Not Found)** - id not found or invalid
POST	Create	/books	create new book	**201 (Created)** - return a Location header with a link to the newly-created resource /books/{id}. **409 (Conflict)** - indicated that resource already exists
		/books/id	Return **405 (Method Not Allowed)**, avoid using POST on single resource	
PUT	Update/ Replace	/books	Return **405 (Method Not Allowed)**, unless you want to **replace every resource** in the entire collection of resource - use with caution.	
		/books/{id}	Update a book	**200 (OK)** or **204 (No Content)** - indicate successful completion of the request **201 (Created)** - if new resource is created and creating new resource is allowed **404 (Not Found)** - if id not found or invalid and creating new resource is not allowed.
PATCH	Partial Update	/books	Return **405 (Method Not Allowed)**, unless you want to modify the collection itself..	
		/books/{id}	Partial update a book	**200 (OK)** or **204 (No Content)** - indicate successful completion of the request **404 (Not Found)** - if id not found or invalid
DELETE	Delete	/books	Return **405 (Method Not Allowed)**, unless you want to **delete the whole collection** - use with caution	
		/books/{id}	Delete a book	**200 (OK)** or **204 (No Content)** or **202 (Accepted)** **404 (Not Found)** - if id not found or invalid

Los métodos que podemos usar vienen definidos por lo que permite la API en función de lo que han definido los desarrolladores de la misma. Habitualmente usaremos alguno de estos métodos:

- **GET:** nos sirve para **solicitar datos de lectura**, al API Rest. Los datos no serán modificados de nignua manera. Posibles respuestas:
  - Si se encuentra el recurso solicitado y se tiene permiso para obtener los datos se obtiene un código **200** y en el body enviaremos los datos en el formato estipulado (JSON, XML, Raw...).
  - Si no se encuentra el recurso **404**.
  - En caso de que la petición no esté bien formada: **400**.

```
{
 "id": 5,
 "oclc": null,
 "isbn10": null,
 "isbn13": "9780006716792",
 "title": "Prince Caspian"
}
```

- **POST:** Sólo se debe utilizar para la **creación de nuevos recursos** en el servidor . La invocación de dos llamadas POST iguales al recurso conllevaría la creación de dos elementos exactamente iguales con ids diferentes. Códigos de respuesta:
  - Si el recurso se ha creado correctamente: **201** y los datos del nuevo recurso en el body (respuesta similar a si hubiésemos solicitado el recurso por GET). Si el recurso que hemos creado no es accesible mediante una URI, podemos establecer una respuesta **200** o **201** sin contenido en el cuerpo.
  - Si el recurso ya existe (por ejemplo DNI duplicado): **409**

```
{
 "id": 5,
 "oclc": null,
 "isbn10": null,
 "isbn13": "9780006716792",
 "title": "Prince Caspian"
}
```

- **PUT:** la usamos para **actualizar datos en el servidor (UPDATE)**, en la petición deberíamos enviar la representación de los datos que queremos modificar en el recurso alojado en el servidor. El uso de PUT debería restringirse a la actualización completa de un recurso individual mientras que POST se usa para conjuntos de recursos.
  - En caso de actualización correcta **200** o **204.**
  - Si el recurso a modificar no existe **404.**

- **PATCH:** Se utiliza para hacer una **actualización parcial de un recurso**, al contrario que PUT que actualiza el recurso entero. No todas las APIs permiten el uso de este método.
  -
  - En caso de actualización correcta **200** o **204.**
  - Si el recurso a modificar no existe **404.**

- **DELETE:** te puedes imaginar que este método sirve para **eliminar recursos** del servidor.
  - En caso de borrado correcto: **200** o **204**.
  - Si el recurso a borrar no existe: **404**.

Se muestran algunos códigos posibles de respuesta pero no son los únicos. Cualquier petición con formato inválido o en la que falten datos debería mostrar un código **400 Bad Request**.

6. Detectar el método de una petición en PHP

Detectar el método de una petición en PHP

Podemos detectar el método por el que recibimos una petición accediendo a la superglobal $_SERVER['REQUEST_METHOD'].

Por ejemplo:

```php
if ($_SERVER['REQUEST_METHOD'] == 'GET')
{

}
// Crear un nuevo post
 else if ($_SERVER['REQUEST_METHOD'] == 'POST')
{

}
//Borrar
else if ($_SERVER['REQUEST_METHOD'] == 'DELETE')
{

}
```

Todas las peticiones que no sean POST o GET enviarán sus parámetros dentro del cuerpo de la petición y no podremos acceder a ellas de la forma habitual. Para poder obtener estas variables podemos utilizar la siguiente función que podemos incluir en BaseController:

```php
function getRequestVars() : array{
        parse_str(file_get_contents("php://input"),$postVars);
        return $postVars;
    }
```

Crea un fichero test.php con este código y realiza peticiones de diversos tipo con Postman para comprobar el funcionamiento.

```
if ($_SERVER['REQUEST_METHOD'] == 'POST')
{
    http_response_code(200);
    echo json_encode($_POST);
}
// Si no es POST
 else
{
    http_response_code(200);

    echo json_encode($_GET);
}
```

Tarea

Crea un controlador que detecte el método request y muestre con un echo:

- Si es GET y recibe un parámetro id: Mostrar elemento
- Si es GET sin más parámetros: Mostrar listado
- Si es POST y recibe parámetro nombre: Insertar + $nombre
- Si es POST y no recibe parámetro nombre: Insertar bad request (header: 400)
- Si es PUT y recibe id + nombre: Modificar + $id + por + $nombre. SI falta algún parámetro 400
- Si es DELETE y recibe id: Borrar + $id si no envía parámetro 400

7. Creación de un servicio web con PHP

Creación de un servicio web con PHP

Redirigir todas las peticiones a index.php

Lo primero que debemos hacer es crear dentro de la carpeta public un fichero .htaccess que obligue a redirigir todas las peticiones al fichero index.php

```
RewriteEngine On
RewriteBase /
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)$ index.php/$1 [L]
```

Para poder probar esta configuración, creamos en el directorio /etc/apache2/sites-available el fichero 007-api1.conf con el siguiente contenido:

```
<VirtualHost *:80>

    DocumentRoot /ruta/hasta/public
    ServerName api1.localhost

    # Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
    # error, crit, alert, emerg.
    # It is also possible to configure the loglevel for particular
    # modules, e.g.
    #LogLevel info ssl:warn

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

    # For most configuration files from conf-available/, which are
    # enabled or disabled at a global level, it is possible to
    # include a line for only one particular virtual host. For example the
    # following line enables the CGI configuration for this host only
    # after it has been globally disabled with "a2disconf".
    #Include conf-available/serve-cgi-bin.conf
    <Directory /ruta/hasta/public/>
          Options +Indexes
```

```
        AllowOverride All
    </Directory>
</VirtualHost>

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet
```

Escribimos ahora los siguientes comandos:

```
sudo a2ensite 007-api1.conf

sudo systemctl restart apache2
```

En la carpeta public, creamos un fichero php con el siguiente contenido

```
<?php

echo $_SERVER['PATH_INFO']
```

Abrimos en el navegador http://api1.localhost/hola/test y debería mostrar por pantalla **/hola/test**

Modificar funcionamiento de FrontController

Vamos a editar el funcionamiento de FrontController para que, en vez de coger el controlador por el parámetro $_GET['controller'] lo coja por la primera string que haya entre dos barras. Por ejemplo en petición http://localhost:8082/**categoria**/2 querríamos coger la string **categoria** y cargar el controlador **CategoriaController**

La variable superglobal $_SERVER en la posicion PATH_INFO, tiene todo lo que está a partir de la barra que marca el final del host y el comienzo del documento. Ejemplos:

URL	$_SERVER['PATH_INFO']	Array a obtener
http://api1.localhost/	*null*	empty

http://api1.localhost/hola/dev	/hola/dev	`array (size=2)`
		`0 => string 'hola' (length=4)`
		`1 => string 'dev' (length=3)`

http://api1.localhost/hola/dev?var=variable	/hola/dev	`array (size=2)`
		`0 => string 'hola' (length=4)`
		`1 => string 'dev' (length=3)`

http://api1.localhost/hola/dev/	/hola/dev/	`array (size=2)`
		`0 => string 'hola' (length=4)`
		`1 => string 'dev' (length=3)`

Sabiendo esto, ¿cómo harías para a partir de $_SERVER['PATH_INFO'] obtener un array en el que el elemento en la posición cero sea el nombre del controlador? En los ejemplos anteriores sería obtener los arrays que se proponen en la tercera columna.

Una vez obtenido el nombre del controlador, insertamos nuestro código el FrontController y hacemos una estructura if / else anidada para cargar el controlador en base al nombre de controlador obtenido en el paso anterior. En este ejemplo variable $controllerUri contiene el valor $array[0]:

```php
if(strtolower($controllerUri) == 'categoria'){
    $controller = new \Com\Daw2\Controllers\CategoriaController();
}
else if(strtolower($controllerUri) == 'usuarios'){
    $controller = \Com\Daw2\Controllers\UsuarioController();
}
else if(strtolower($controllerUri) == 'otrocontrolador'){
    $controller = \Com\Daw2\Controllers\OtroController();
}
else{
    //Si no entra por ninguna regla cargamos el controlador por defecto
    $controller = \Com\Daw2\Controllers\ControladorDefecto();
}
```

Como en los servicios REST lo que discrimina que queremos hacer no es el action recibido si no el tipo de petición GET, PUT, POST... lo que vamos a hacer siempre es cargar el método index() de los controladores y ahí discriminaremos en base a la petición recibida. Por lo tanto podemos modificar el resto del FrontController para que lo único que haga es llamar a la función index() del controlador elegido.

```php
if(strtolower($controllerUri) == 'categoria'){
    $controller = new \Com\Daw2\Controllers\CategoriaController();
}
else if(strtolower($controllerUri) == 'usuarios'){
    $controller = \Com\Daw2\Controllers\UsuarioController();
}
else if(strtolower($controllerUri) == 'otrocontrolador'){
    $controller = \Com\Daw2\Controllers\OtroController();
}
else{
    //Si no entra por ninguna regla cargamos el controlador por defecto
    $controller = \Com\Daw2\Controllers\ControladorDefecto();
}

$controller->index();
```

Ya no es necesario controlar si existe la clase o el método porque nosotros manualmente estamos haciendo que sólo se carguen clases y métodos válidos.

Codificación del controlador

En este ejemplo vamos a codificar el controlador de categorias. Ofrecerá los siguientes métodos:

URL	METHOD	Parámetros	Resultado
/categoria	GET	*Ninguno*	***200***. *Listado con todas las categorías.* ***500***. *Error indeterminado en el lado del servidor.*
/categoría/$id	GET	*Ninguno*	***200***. *Todos los datos de la categoría con id = $id* ***404***. *Si no existe la categoría con id = $id*
/categoria	POST	id_padre: int. Idenficador del padre categoria: string. Nombre de la categoría. No vacía.	***201***. *En caso de alta satisfactoria. Necesita recibir un id_padre válido o nulo y categoría es string no vacía.* **409**. Si el alta no se ejecuta porque ya existe en BBDD **422**. Si id_padre es un entero pero no existe en la base de datos
/categoria/$id	DELETE	*Ninguno*	**200**. Si se borra correctamente **404**. Si no existe la categoría a borrar.
/categoria/$id	PUT	id_padre: int. Idenficador del padre categoria: string. Nombre de la categoría. No vacía.	**200**. Si se modifica correctamente. **404**. Si no existe la categoría a modificar.

Si se hace una petición en la que falten parámetros o en la que estos sean incorrectos, se devuelve un código 400.

Creación del controlador

Como podemos observar, en el controlador vamos a tener que coger en algunos casos la posición 1 del array con el path de la petición para obtener el id de la categoría, para no duplicar código vamos a crear en BaseController un atributo privado $pathArray y dos métodos:

- El primero guarda en pathArray el array con los diferentes niveles de la ruta.
- El segundo es un getter para obtener este valor

Fichero modificado: