

API-CONNECT



Realizado por: Mario Martín Godoy

Curso: 2º Desarrollo Aplicaciones Web

Módulo: Desarrollo de entorno servidor

Tecnologías: HTML, CSS, PHP

ÍNDICE DE CONTENIDOS

1. Explicación del proyecto.....	página	2
2. Justificación tecnología utilizada.....	página	3
3. Documentación técnica.....	página	5
4. Prueba funcionamiento.....	página	8
5. Explicación código.....	página	12
6. Tecnologías utilizadas.....	página	18

1. EXPLICACIÓN DEL PROYECTO

API-Connect: Servicio web API-Rest para consumición de datos.

Bienvenido a API-Connect. Es una aplicación realizada en PHP que levanta una API REST a partir de un fichero .xml. Se basa en crear un servicio web realizado con PHP para, posteriormente, consumirlo con una aplicación cliente. El funcionamiento lineal de la aplicación es: Leer un fichero XML, transformarlo a una array de objetos PHP, pasarlo a JSON y levantar la API con los datos obtenidos.

Lectura de ficheros:

La aplicación PHP lee un fichero XML pasado por ruta en un archivo config, posteriormente lo transforma a un array de objetos de php. Cada objeto libro viene instanciado por su clase con los atributos indicados y los métodos correspondiente de inserción en la API Rest.

Creación de JSON:

Para cada uno de los objetos PHP, se crea utiliza una función nativa de PHP para pasar todo el array de respuesta obtenido, a un array de objetos tipo JSON legible por el servicio web y por el consumidor.

Innovación:

¿Qué nos hace únicos?: API-Connect va más allá de ser un simple servicio web para consumir datos. Buscamos constantemente innovar con características únicas que mejoren la experiencia de lectura de datos, manteniendo a nuestros clientes contentos y garantizando calidad en su negocio.

Con API-Connect, estamos construyendo una plataforma de servicio virtual donde los consumidores de la API sean capaces de convertir cualquier conjunto de datos a un formato JSON, para posteriormente realizar consultas fácilmente. ¡Sumérgete en la esencia de API-Connect!

2. JUSTIFICACIÓN DE LA TECNOLOGÍA UTILIZADA

En el desarrollo de servicios web, la elección de la tecnología adecuada es crucial para garantizar la eficiencia, la escalabilidad y la mantenibilidad del sistema. La arquitectura de API REST (Representational State Transfer) se ha convertido en una de las opciones más populares y ampliamente utilizadas para la implementación de servicios web. A continuación, se presentan las características principales de API REST y una comparación con otras tecnologías de servicios web.

Características principales de API REST:

1. **Arquitectura basada en estándares:** API REST utiliza los estándares fundamentales del protocolo HTTP, como GET, POST, PUT y DELETE, para realizar operaciones CRUD (Create, Read, Update, Delete) en los recursos del servidor. Esto proporciona una estructura coherente y predecible para el desarrollo de APIs.
2. **Stateless (sin estado):** API REST no mantiene ningún estado de sesión en el servidor entre las solicitudes del cliente. Cada solicitud del cliente contiene toda la información necesaria para que el servidor comprenda y procese la solicitud. Esta característica simplifica la escalabilidad y la distribución de la carga.
3. **Interfaz uniforme:** API REST sigue el principio de una interfaz uniforme, lo que significa que utiliza una sintaxis común y consistente para interactuar con los recursos del servidor a través de URI (Identificador de Recurso Uniforme), métodos HTTP y representaciones de recursos (generalmente JSON o XML).
4. **Capacidad de caché:** API REST aprovecha las capacidades de caché del protocolo HTTP para mejorar el rendimiento y la eficiencia. Los clientes pueden almacenar en caché las respuestas de las solicitudes y reutilizarlas si las condiciones lo permiten, lo que reduce la carga en el servidor y mejora la velocidad de respuesta.
5. **Independencia de plataforma:** API REST permite la comunicación entre sistemas heterogéneos, ya que no impone restricciones sobre las tecnologías o plataformas utilizadas por el cliente y el servidor. Esto facilita la integración con una amplia variedad de aplicaciones y servicios.

Comparación con otras tecnologías de servicios web:

1. **SOAP (Simple Object Access Protocol):** A diferencia de SOAP, que utiliza un enfoque más pesado basado en XML y requiere una estructura compleja de mensajes, API REST se basa en HTTP y utiliza formatos de datos más ligeros como JSON. Esto hace que API REST sea más fácil de implementar y consumir, especialmente en entornos web y móviles.
2. **GraphQL:** Si bien GraphQL ofrece flexibilidad en las consultas de datos al permitir que los clientes soliciten solo los campos específicos que necesitan, API REST sigue siendo preferible en casos donde se requiere una comunicación más simple y predecible. Además, REST se integra fácilmente con herramientas y frameworks existentes de HTTP, mientras que GraphQL puede requerir una curva de aprendizaje más pronunciada.
3. **RPC (Remote Procedure Call):** Aunque RPC permite invocar funciones remotas de manera similar a las llamadas locales, carece de la capacidad de caché y la uniformidad de interfaz proporcionada por API REST. Además, API REST se alinea mejor con los principios de diseño web, como la navegación a través de URI y la manipulación de recursos, lo que lo hace más adecuado para aplicaciones web modernas y escalables.

En resumen, la elección de API REST como tecnología para desarrollar servicios web proporciona una serie de ventajas, incluida su simplicidad, escalabilidad, independencia de plataforma y capacidad de integración. Estas características lo convierten en una opción atractiva para una amplia gama de aplicaciones y casos de uso en comparación con otras tecnologías de servicios web disponibles.

3. DOCUMENTACIÓN TÉCNICA

3.1 Interfaz de uso del servicio web

Para acceder al servicio web realizado, deberemos utilizar una URL de base que nos llevará al punto de control del servicio. En este caso, nuestra URL base será la siguiente:

BASE URL: <http://localhost/DWES/API-Connect/books>

Al no estar desplegada en un servidor con hosting, lo desplegamos en nuestro propio equipo con XAMPP. Este acceso sería un acceso no parametrizado, sin embargo, el cliente podrá filtrar la información parametrizando la URL.

3.2 Peticiones esperadas:

Las peticiones esperadas al servicio web en este caso será únicamente la petición GET para recibir los datos de la API. En el propio código de la aplicación se podrían añadir funcionalidades para gestionar las peticiones esperadas, de esa manera, el cliente podría realizar peticiones POST y PUT.

3.3 Parámetros disponibles:

Los parámetros disponibles para filtrar la información recibida del servicio son:

Filtrar por ID:

Ejemplo de petición: <http://localhost/DWES/API-Connect/books?id=bk112>

Filtrar por Autor:

Ejemplo de petición: <http://localhost/DWES/API-Connect/books?autor=Galos>, Mike

Filtrar por Género:

Ejemplo de petición: <http://localhost/DWES/API-Connect/books?genero=Computer>

Paginación:

Respecto a la paginación, no encontraba el sentido a realizar una paginación para este tipo de peticiones, ya que, en nuestro caso, la estructura de datos es demasiado pequeña. Por tanto, si realizamos una paginación como su concepto indica que se debe realizar, al paginar por 5, seguiría mostrando resultados.

Mi planteamiento ha sido que muestre un limit a los resultados. Si introduces un 5, muestra 5 libros. Esta me ha parecido la solución más lógica en este proyecto.

Ejemplo de petición: <http://localhost/DWES/API-Connect/books?pagina=2>

Estos parámetros son los únicos soportados por el servicio web. Al introducir un parámetro que no sea el esperado por la aplicación, la aplicación devuelve un mensaje de error.

3.4 Respuesta JSON:

El servicio web, devuelve un conjunto de datos transformados de XML a formato JSON para que sea legible por un cliente que los consuma. En este caso, hemos elegido JSON porque es el más utilizado y tiene una tratabilidad de los datos muy fácil de utilizar.

Un ejemplo de la estructura de datos que devuelve el servicio web al realizar una petición GET sería el siguiente:

```
{
  "result": "ok",
  "libros": [
    {
      "id": "bk101",
      "autor": "Gambardella, Matthew",
      "titulo": "XML Developer's Guide",
      "genero": "Computer",
      "precio": 44.95,
      "lanzamiento": "2000-10-01",
```

```

    "descripcion": "An in-depth look at creating applications\nwith XML."
  },
  {
    "id": "bk102",
    "autor": "Ralls, Kim",
    "titulo": "Midnight Rain",
    "genero": "Fantasy",
    "precio": 5.95,
    "lanzamiento": "2000-12-16",
    "descripcion": "A former architect battles corporate zombies,\nan evil sorceress, and her own childhood to become queen\nof the world."
  },
  ....
]
}

```

3.5 Manejo de errores

En cuanto al manejo de errores, encontramos:

- Si el fichero no ha sido cargado: Si no ha encontrado el archivo o ha ocurrido algún error, el servicio web devolverá un error con el siguiente mensaje: "No se ha podido cargar el archivo".
- Si los parámetros no están permitidos, el servicio web devolverá el siguiente mensaje de error: "Error en la solicitud".
- Si el método de solicitud no está permitido, devolverá un result "error".
- Si no encuentra ningún resultado, devolverá un array vacío.

4. PRUEBA FUNCIONAMIENTO

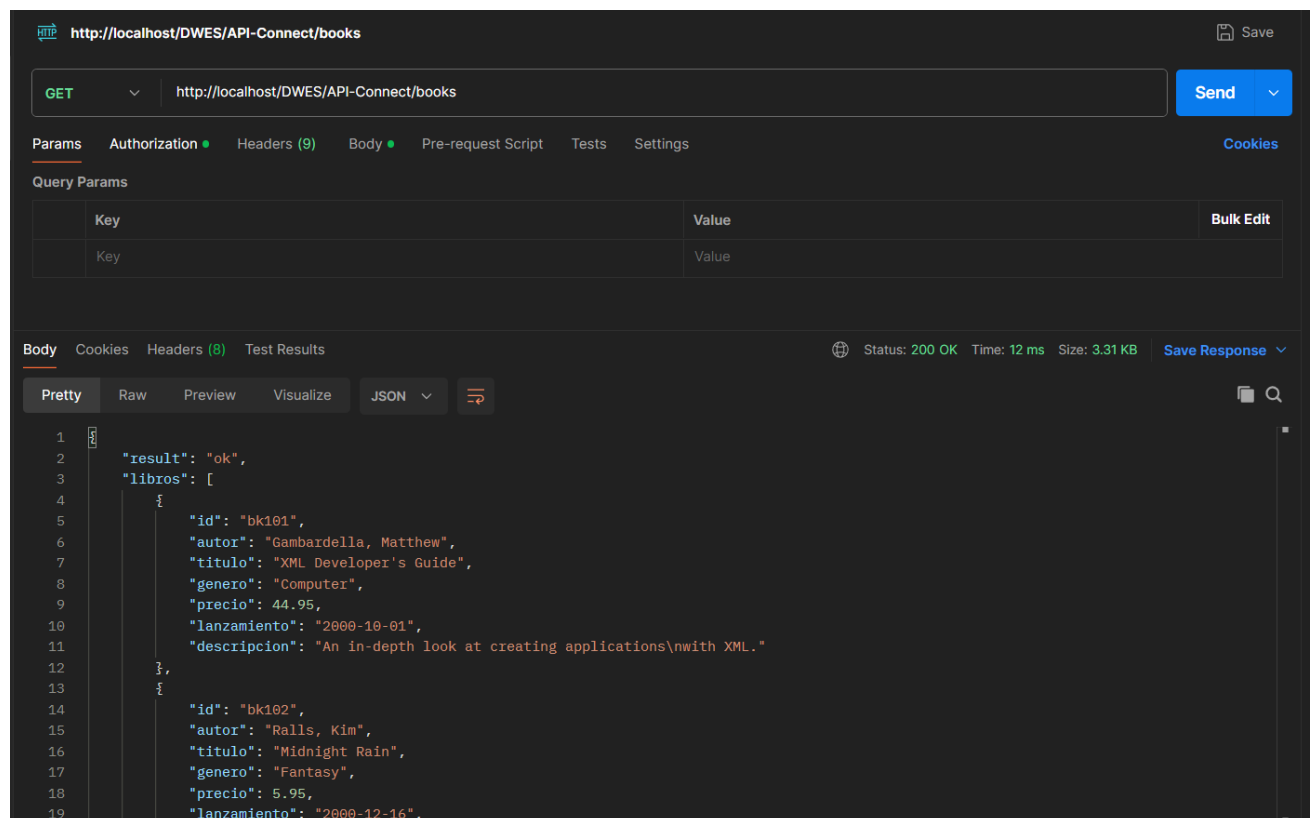
Para la comprobación del correcto funcionamiento y configuración del servicio web realizado en la práctica, he utilizado una aplicación llamada POSTMAN, que permite la prueba de solicitudes a cualquier API introduciendo la URL de la misma.

En esta aplicación, las solicitudes se pueden realizar con parámetros. Además, nos permite elegir qué tipo de petición realizamos, en qué formato queremos la respuesta y qué partes del cuerpo de la respuesta queremos modificar (en caso de que sea un PATCH o PUT).

Primera prueba: Acceso al punto de ruptura de la aplicación.

En este caso, el punto de ruptura es: <http://localhost/DWES/API-Connect/books>.

Esta petición nos debería devolver un documento JSON con todos los objetos libro obtenidos del fichero, ya que no hemos realizado ningún filtro en la petición.



La prueba ha salido correctamente, ya que nos ha devuelto todos los datos que se encuentran en el fichero.

Segunda prueba: Peticiones parametrizadas con los parámetros permitidos.

Por ID:

Postman interface showing a GET request to `http://localhost/DWES/API-Connect/books?id=bk102`. The response is a JSON object with a single book entry.

Key	Value
<input checked="" type="checkbox"/> id	bk102

```
1 {
2   "result": "ok",
3   "libros": {
4     "1": {
5       "id": "bk102",
6       "autor": "Ralls, Kim",
7       "titulo": "Midnight Rain",
8       "genero": "Fantasy",
9       "precio": 5.95,
10      "lanzamiento": "2000-12-16",
11      "descripcion": "A former architect battles corporate zombies, \nan evil sorceress, and her own childhood to become queen\nof the world."
12    }
13  }
}
```

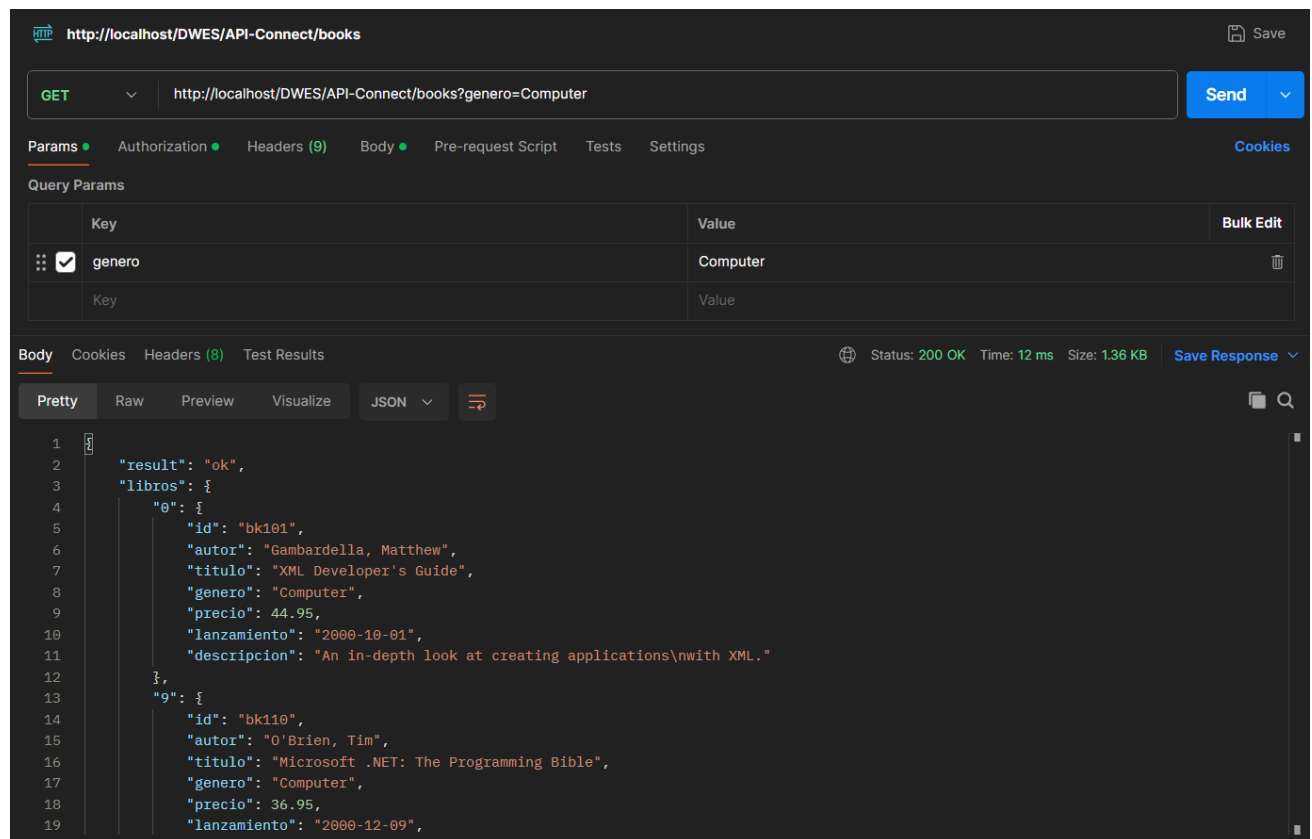
Por Autor:

Postman interface showing a GET request to `http://localhost/DWES/API-Connect/books?autor=Gambardella, Matthew`. The response is a JSON array containing one book entry.

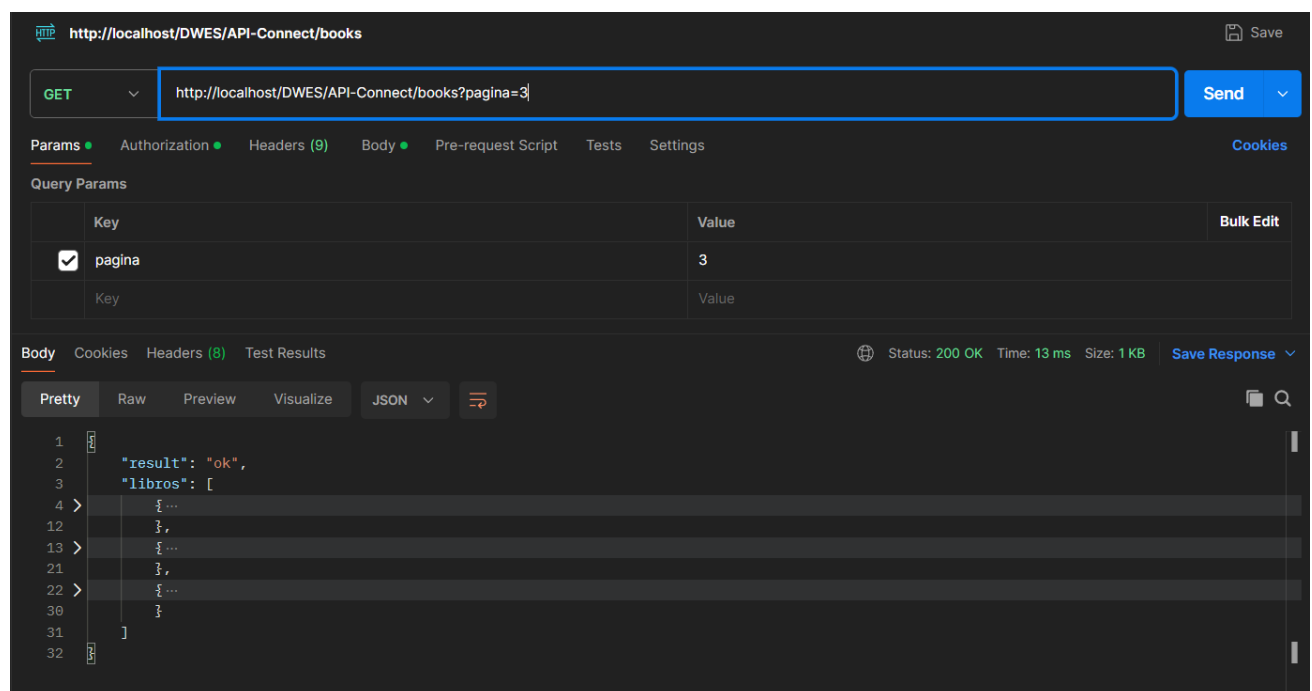
Key	Value
<input checked="" type="checkbox"/> autor	Gambardella, Matthew

```
1 {
2   "result": "ok",
3   "libros": [
4     {
5       "id": "bk101",
6       "autor": "Gambardella, Matthew",
7       "titulo": "XML Developer's Guide",
8       "genero": "Computer",
9       "precio": 44.95,
10      "lanzamiento": "2000-10-01",
11      "descripcion": "An in-depth look at creating applications\nwith XML."
12    }
13  ]
}
```

Por Género:

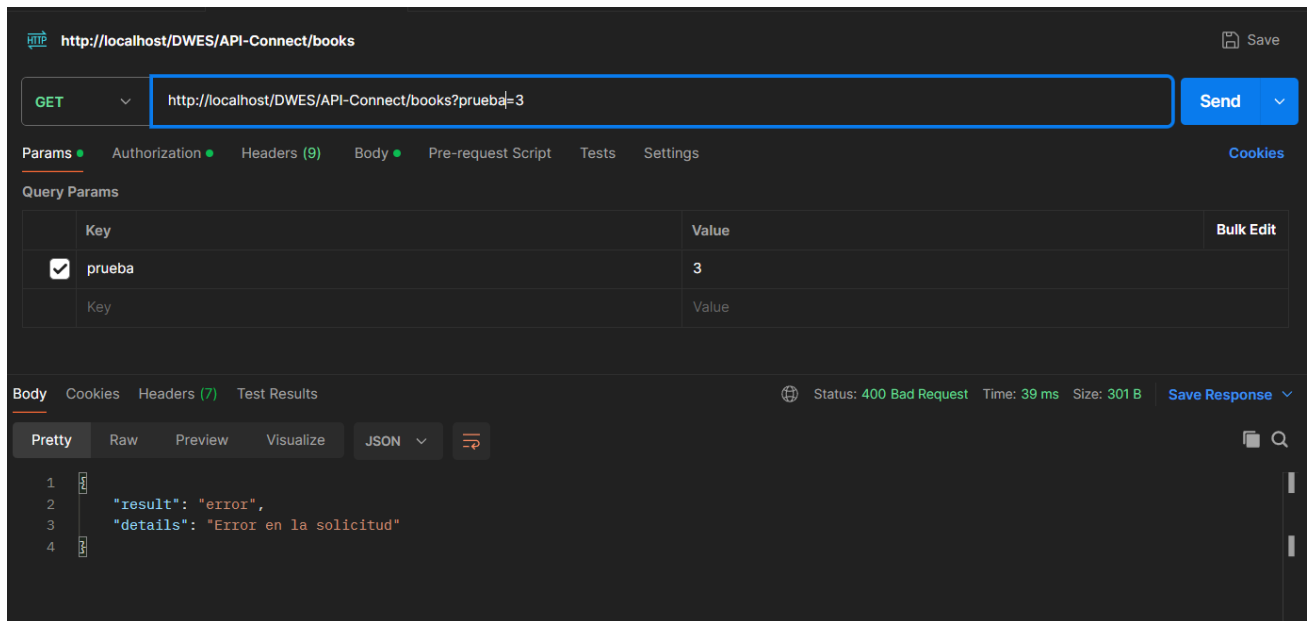


Por página:



Todas las pruebas realizadas con los parámetros permitidos han funcionado correctamente.

Tercera prueba: Peticiones parametrizadas con parámetros no permitidos



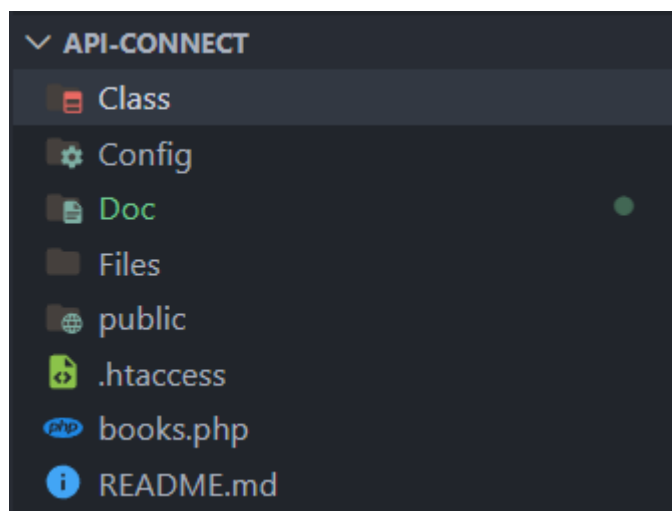
La prueba ha funcionado porque ha devuelto un result error y un mensaje de error descriptivo.

En resumen, las pruebas han sido todas satisfactorias. Podemos comprobar que el servicio web ha sido desarrollado y ejecutado correctamente y cumple todas sus funciones. Además, hemos realizado pruebas para comprobar el control y manejo de errores, y en todas ellas, el resultado ha sido el esperado.

5. EXPLICACIÓN DEL CÓDIGO

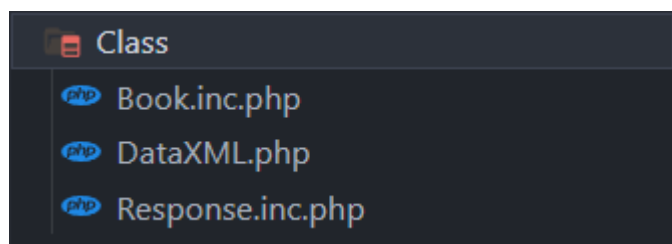
5.1 Estructura general del proyecto

Para empezar, el proyecto está dividido en varios directorios que serán explicados a continuación. Esta es la estructura general de la carpeta raíz del proyecto.



5.2 Directorio Class

En este directorio encontramos las clases necesarias para el correcto funcionamiento del servicio web. Encontramos la clase book con las funciones de comprobación de parámetros y de consulta de fichero. La clase DataXML que incluye las funciones que leen el fichero XML y lo recorren para crear un array de objetos PHP. Por último, encontramos la clase Response que transforma un conjunto de datos a tipo JSON para que sea legible por el servicio.



5.2.1 Clase Book.Inc.php

Esta clase es la encargada de realizar las comprobaciones de los parámetros de las peticiones GET. En caso de que el parámetro sea válido, realiza un response con el array obtenido, en caso contrario, devuelve un mensaje de error descriptivo.

```
6 class Book extends DataXML
7 {
8
9     //indicamos los parámetros válidos para las peticiones get mediante un array
10    private $allowedConditions_get = array(
11        'id',
12        'autor',
13        'genero',
14        'pagina'
15    );
16
17    /**
18     * Método get: recibe los parámetros de la petición get,
19     * los recorre para comprobar si son válidos,
20     * si no lo son los elimina y devuelve una respuesta json de error,
21     * si lo son realiza la consulta a DB y devuelve un json con la respuesta correcta
22     *
23     * @param array $params Los parámetros get usados en BD
24     * @return [array | void] Los usuarios de la BD
25     */
```

```
26    public function get($params){
27        //Recorremos los parámetros get
28        foreach ($params as $key => $param) {
29            //si los parámetros no están permitidos...
30            if(!in_array($key, $this->allowedConditions_get)){
31                //eliminamos los parámetros
32                unset($params[$key]);
33                //creamos el array de error
34                $response = array(
35                    'result' => 'error',
36                    'details' => 'Error en la solicitud'
37                );
38                //devolvemos la petición de error convertida a json
39                Response::result(400, $response);
40                exit;
41            }
42        }
43        //Llamamos
44        $books = parent::getBooks($params);
45
46        return $books;
47    }
48 }
```

5.2.2 Clase DataXML.php

Esta clase es la encargada de la gestión de la información recibida mediante un fichero XML. La primera función recorre el fichero y crea un array de objetos php con los datos obtenidos de dicho objeto. La segunda función devuelve solo los libros filtrados por el parámetro pasado en la petición

```

3     class DataXML {
4
5
6         public static function readXML () {
7             $books = [];
8             $xml = simplexml_load_file(XMLFILE);
9             if($xml){
10                 try {
11                     foreach ($xml as $book){
12                         $books[] = [
13                             'id' => (string) $book['id'],
14                             'autor' => (string) $book->author,
15                             'titulo' => (string) $book->title,
16                             'genero' => (string) $book->genre,
17                             'precio' => (float) $book->price,
18                             'lanzamiento' => (string) $book->publish_date,
19                             'descripcion' => (string) $book->description,
20                         ];
21                     }
22                 } catch (Exception $e) {
23                     die("No se ha cargado el archivo. Excepción: " . $e );
24                 }
25             }
26         } else{
27             echo "No se ha podido cargar el archivo";
28         }
29         return $books;
30     }
31 }

```

```

33     public static function getBooks($pFilter = null) {
34         $books = self::readXML();
35
36         if ($pFilter === null) {
37             return $books;
38         }
39
40         if(isset($pFilter["pagina"])){
41             return array_slice($books, 0, (int)$pFilter['pagina']);
42         }
43
44         return array_filter($books, function($book) use ($pFilter) {
45             foreach ($pFilter as $key => $value) {
46                 if (!isset($book[$key]) || $book[$key] !== $value) {
47                     return false;
48                 }
49             }
50             return true;
51         });
52     }
53 }

```

5.2.3 Clase Response.inc.php

```

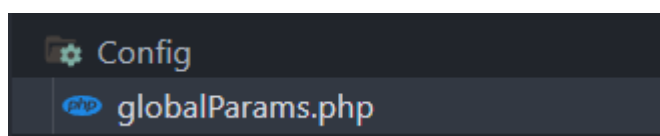
5     class Response
6     {
7         /**
8          * Método result: transforma el array de la respuesta en un json que se
9          * devuelve al cliente como resultado de su petición
10          *
11          * @param int $code El código de la respuesta
12          * @param array $response El array de datos que vamos a convertir a json
13          * @return void
14          */
15         public static function result($code, $response){
16
17             header('Content-type: application/json');
18             http_response_code($code);
19
20             echo json_encode($response);
21         }
22     }

```

Esta clase contiene una función result que transforma el código recibido a formato JSON.

5.3 Directorio Config

En este directorio únicamente encontramos un archivo PHP con las variables globales necesarias para el funcionamiento de nuestro servicio web.



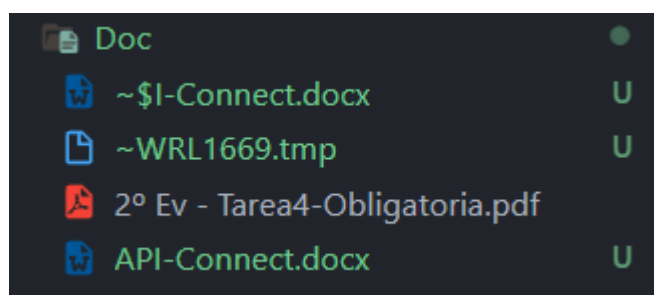
5.3.1 Archivo globalParams.php

```
1 <?php
2 //PARÁMETROS DE CONFIGURACIÓN GLOBALES
3 const XMLFILE = "../Files/books.xml";
4 ?>
```

En este archivo se deberá modificar la ruta desde la que se obtiene el fichero xml que utilizará el servicio web para la lectura de datos.

5.4 Directorio DOC

En este directorio se encuentran archivos de documentación de la práctica.



5.5 Directorio Files

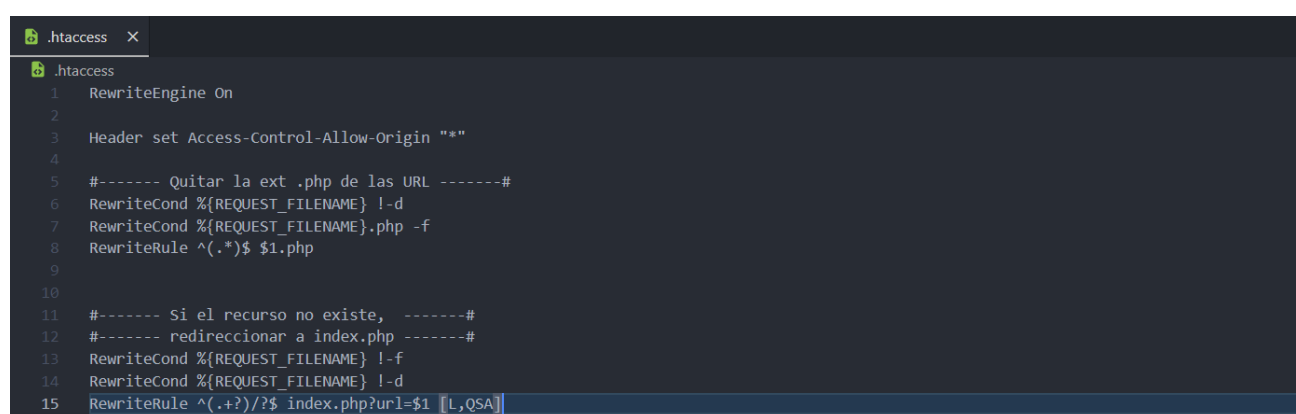
En este directorio se encuentra el archivo xml desde el que se obtendrán todos los datos.

5.6 Directorio Public

En este directorio se encuentra el logo de la aplicación. Únicamente es utilizado para el archivo README.md, por tanto, no es relevante en el servicio web.

5.7 Archivo .htaccess

En este archivo encontramos las configuraciones necesarias para que no sea necesario poner las extensiones .php y para que si el recurso solicitado no existe, se rediriga la solicitud al endpoint.

A screenshot of a code editor showing the content of a .htaccess file. The editor has a dark theme. The file name ".htaccess" is visible in the top left corner. The code is as follows:

```
1 RewriteEngine On
2
3 Header set Access-Control-Allow-Origin "*"
4
5 #----- Quitar la ext .php de las URL -----#
6 RewriteCond %{REQUEST_FILENAME} !-d
7 RewriteCond %{REQUEST_FILENAME}.php -f
8 RewriteRule ^(.*)$ $1.php
9
10
11 #----- Si el recurso no existe, -----#
12 #----- redireccionar a index.php -----#
13 RewriteCond %{REQUEST_FILENAME} !-f
14 RewriteCond %{REQUEST_FILENAME} !-d
15 RewriteRule ^(.+)?/?$ index.php?url=$1 [L,QSA]
```

5.8 Archivo books.php

Este es el endpoint del servicio web. Cuando llega la petición, este es el archivo que gestiona cómo tratar dicha petición. En este caso, solo encontramos la opción de una petición GET, pero en el switch se podría modificar el código para añadir otro tipo de peticiones.

En este caso, si la petición es GET y los parámetros están permitidos, devuelve la consulta formateada al solicitante. Si es otro tipo de petición, devuelve un error.

```

7
8 //Creamos el objeto de la clase User para manejar el endpoint
9 $book = new Book();
10
11 //Comprobamos de qué tipo es la petición al endpoint
12 switch ($_SERVER['REQUEST_METHOD']) {
13     //Método get
14
15     case 'GET':
16         //Recogemos los parámetros de la petición get
17         $params = $_GET;
18
19         //Llamamos al método get de la clase User, le pasamos los
20         //parámetros get y comprobamos:
21         //1º) si recibimos parámetros
22         //2º) si los parámetros están permitidos
23         $books = $book->get($params);
24
25         //Creamos la respuesta en caso de realizar una petición correcta
26         $response = array(
27             'result' => 'ok',
28             'libros' => $books
29         );
30
31         Response::result(200, $response); //devolvemos la respuesta a la petición correcta
32
33         break;
34
35     default:
36         //creamos el array de error
37         $response = array(
38             'result' => 'error'
39         );
40         //devolvemos la respuesta
41         Response::result(404, $response);
42
43         break;
44 }

```

5.9 Archivo README.md

Este es un archivo MarkDown, que es un procesador de texto como html. En este archivo realizo una pequeña documentación descriptiva del proyecto, con instrucciones para desplegar el servicio.

6. TECNOLOGÍAS UTILIZADAS

En el desarrollo de esta aplicación web, he utilizado las siguientes herramientas:

Un entorno de desarrollo de código: Visual Studio Code con diversos plugins para facilitar y agilizar el trabajo en los lenguajes utilizados.

4 lenguajes de programación: HTML para la estructura base de la web, CSS para estilar los diseños. Y, por último, PHP, para aplicar la lógica al sistema de control del servidor.

Una aplicación para probar el funcionamiento del servicio web, en este caso he elegido POSTMAN. Esta aplicación permite realizar solicitudes al servicio con los parámetros que se desee y con la url requerida. Es una forma fácil y cómoda de realizar pruebas a una API.

Una aplicación para el control de versiones: GIT y GITHUB.

Git lo he utilizado para subir todo el proyecto en su respectiva versión mediante comandos por consola. Y en GitHub se encuentra el repositorio en la nube al que se suben los archivos.