raffrearaUS

2º Ingeniería Informática – Ingeniería del Software

Desarrollo y consumo de una API REST con PHP

Índice

Desarrollo

Consumo

Desarrollo

Introducción

El propósito de la primera parte de este trabajo es desarrollar una API REST usando el lenguaje de scripts PHP. Además, para proporcionar datos a la API, incorporaremos una base de datos. Para ello, podemos usar cualquier base de datos que queramos, pero para poner en práctica lo aprendido en IISSI, usaremos Oracle en este caso. Debemos recordar que el estilo arquitectónico REST (Representational State Transfer) se basa fundamentalmente en el uso de recursos, que tienen una o varias representaciones, y que admiten diferentes operaciones básicas. Estas operaciones son crear, leer, actualizar y eliminar, y son comúnmente conocidas con el nombre de operaciones CRUD ("create", "read", "update" y "delete" en inglés). Las APIs que se basan en REST usan los verbos del protocolo HTTP para realizar estas operaciones: GET para leer, POST para crear, PUT para actualizar, y DELETE para eliminar.

Para nuestra API, como ya hemos indicado, usaremos el lenguaje PHP y una base de datos de Oracle. Ofreceremos dos recursos: "games", que representará juegos con un ID, un nombre y una fecha de lanzamiento; y "gamelists", que representa una lista de juegos con una ID y una propiedad "games" que es un array con juegos.

Las operaciones que se van a poder realizar son las siguientes:

GET games.php – Obtiene una lista con todos los juegos.

GET games.php/{id} – Obtiene el juego con la ID dada.

POST games.php – Crea un juego con la información del cuerpo de la petición.

PUT games.php – Actualiza un juego con la información del cuerpo de la petición.

DELETE games.php/{id} – Elimina el juego con la ID dada.

GET gamelists.php – Obtiene una lista con todas las listas de juegos.

GET gamelists.php/{id} – Obtiene la lista con la ID dada.

POST gamelists.php – Crea una lista vacía con la información del cuerpo de la petición.

DELETE gamelists.php/{id} – Elimina la lista con la ID dada.

POST gamelists.php/{lid}/{gid} – Añade un juego a una lista.

DELETE gamelists.php/{lid}/{gid} – Elimina un juego de una lista.

A continuación se proporciona el código SQL para la creación de las dos tablas correspondientes a los recursos en la base de datos:

```
CREATE TABLE GAMES (
GID CHAR(6) NOT NULL PRIMARY KEY,
GNAME VARCHAR(50) NOT NULL,
RELEASEDATE DATE NOT NULL
);

CREATE TABLE GAMELISTS (
LID NUMBER NOT NULL PRIMARY KEY
);
```

Como sabemos, una columna de una tabla no puede contener una colección (listas, arrays…), por lo que, según lo aprendido, necesitamos crear una tabla auxiliar que nos ayude a añadir juegos a una lista de juegos:

```
CREATE TABLE CONTAINS (
CID NUMBER NOT NULL PRIMARY KEY,
LID NUMBER NOT NULL,
GID CHAR(6) NOT NULL,
FOREIGN KEY (LID) REFERENCES GAMELISTS,
FOREIGN KEY (GID) REFERENCES GAMES
);
```

Para generar los ID de esta última tabla necesitamos crear una secuencia:

```
CREATE SEQUENCE SEQ_CONTAINS;
```

Con estas tablas ya tenemos lo necesario para comenzar con nuestra API. Lo primero, si se quiere, será introducir algunos datos en la base de datos. Posteriormente, para poder comunicar Oracle y PHP, crearemos un fichero “gestionBD.php”, prácticamente idéntico a los que se hicieron en clase, cambiando los datos de acceso. En este fichero indicamos también que los errores que ocurran lancen excepciones (del tipo PDOException). Además, por si ocurre un error en la propia conexión a la base de datos,

habrá que capturar la excepción y realizar un tratamiento. En el “catch” de todos los bloques “try/catch” de la API haremos simplemente un “echo” del mensaje de error y detendremos la ejecución del script con la función “die()”.

Crearemos también los archivos “games.php” y “gamelists.php”, que serán los archivos a los que accederemos para usar la API. Además, serán necesarios dos últimos archivos para la parte del desarrollo, que son “gestionJuegos.php” y “gestionListas.php”, donde tendremos todas las funciones necesarias para realizar todas las operaciones que soportará nuestra API.

Veremos ahora la estructura base de “games.php” y “gamelists.php”, y en los siguientes apartados veremos la implementación necesaria para cada una de las operaciones.

Lo primero que necesitamos hacer es incluir los ficheros de gestión correspondientes, usando la función “require_once”: para ambos hará falta el archivo que contiene las funciones de la base de datos, y luego el archivo de funciones correspondiente para juegos o listas (para las listas también necesitamos las funciones para juegos).

A continuación, necesitamos saber qué método se está usando en una petición a la API. Esta información la tenemos en $_SERVER[“REQUEST_METHOD”], y simplemente tenemos que guardarla en una variable para usarla posteriormente.

Dado que vamos a permitir el uso de plantillas para las URIs, tendremos que extraer la información que esta pueda contener. Para ello podemos usar $_SERVER[“PATH_INFO”], que nos devuelve lo que hay después del “.php” y antes de los parámetros de un GET (encabezados por “?”). Esta variable es una cadena, por lo que tendremos que manipularla para poder convertirla en un array con los parámetros para usarlos más fácilmente. En primer lugar usaremos la función “trim” con “/” para eliminar la primera barra, y luego, con la función “explode” con “/”, dividiremos la cadena por cada barra que haya y obtendremos el array que queremos. Lo guardamos en una variable, pero, en caso de que $_SERVER[“PATH_INFO”] no exista (“!isset”) tendremos que guardar null. Para el caso de que tengamos parámetros adicionales en el GET los extraemos fácilmente de $_GET (o $_REQUEST). En nuestro caso, vamos a

permitir que en el GET a “games.php” podamos realizar una petición con orden o con paginación, por lo que necesitaremos los parámetros “sort”, “limit” y “offset”. De nuevo, si no existen, debemos guardar un null.

Lo último que tenemos que extraer de la petición es el cuerpo. El cuerpo se obtiene con “file_get_contents(“php://input”)”, pero al estar en formato JSON no lo podemos usar como tal. PHP proporciona funciones para la conversión entre arrays y JSON, y en este caso necesitamos la función “json_decode”. Le pasamos como parámetro el cuerpo extraído como se indicó antes, y debemos pasar como segundo parámetro un “true” para que el array devuelto sea un array asociativo. Luego guardamos este array en una variable.

A continuación, para evitar ensuciar el código abriendo y cerrando la conexión con la base de datos en cada operación, lo haremos solo una vez. Primero, guardamos la conexión devuelta por “crearConexionBD” de “gestionBD.php” en una variable. A continuación, para elegir lo que haremos en función del método HTTP de la petición, tendremos varios “if” que tendrán como parámetro el método (que extrajimos al principio). El contenido detallado de esta parte es lo que veremos en los siguientes apartados. Finalmente, cerramos la conexión con “cerrarConexiónBD”.

Con esto, ya tenemos guardados en variables el método HTTP, todos los parámetros de la URI (tanto los de la plantilla como los adicionales incluidos en el GET), el contenido del cuerpo de la petición, y una conexión a la base de datos. A continuación veremos el tratamiento correspondiente a cada uno de los métodos.

GET

Este es el método más simple, ya que sólo se requiere hacer una consulta a la base de datos y devolver el resultado. Sin embargo, la adición de parámetros adicionales hace que la creación de la consulta se complique un poco.

Primero, tanto para “games” como para “gamelists”, comprobamos si la URI tenía alguna ID. Si la tenía, entonces tenemos que comprobar si en la base de datos existe un juego o una lista con dicha ID. Recordemos que todas las funciones que trabajan con la base de datos las tenemos en

“gestionJuegos.php” y en “gestionListas.php”. Esto lo hacemos con una consulta que cuente (“COUNT(*)”) el número de filas que coinciden con la ID dada. Si existe, hacemos un “SELECT” para obtener los datos del recurso que tiene esa ID y lo devolvemos. Como la función devuelve un PDOStatement, debemos extraer el resultado de la consulta y convertirlo a JSON. Para ello, tenemos que usar la función “fetch”, usando además el parámetro “PDO::FETCH_ASSOC” para indexar el array devuelto por los nombres de las columnas, y luego colocar este array dentro de la función “json_encode” para transformarlo a JSON. Para indicar en la respuesta que lo que estamos enviando es un fichero en formato JSON, es necesario modificar un campo de la cabecera de HTTP. La función “header” de PHP, que usamos en clase para redirecciones, sirve para modificar (o añadir) campos de la cabecera. En este caso, tenemos que cambiar el tipo del contenido, y para ello pasamos como parámetro la cadena “Content-type: application/json”. Posteriormente hacemos un “echo” de la consulta.

En el caso de las listas, antes de convertir el resultado a JSON, tenemos que añadir todos los juegos que contiene. Hacemos una consulta a la tabla CONTAINS pasando como parámetro la ID dada, y con el PDOStatement devuelto usamos la función “fetchAll” (también con el parámetro “PDO::FETCH_ASSOC”). Este resultado se lo añadimos a la lista que sacamos anteriormente (recordemos que la lista es un array con sus propiedades), y ya podemos convertirla a JSON.

En caso de que no exista un recurso con la ID dada, tenemos que indicarlo de alguna forma. Las APIs REST usan los códigos de estado de HTTP para este fin. Por lo tanto, usando la misma función “header” de antes, indicamos “HTTP/1.1 404 Not Found”. Existe también la función “http_response_code” que toma como parámetro un código de error, pero “header” permite indicar fácilmente la cabecera exacta que se enviará.

Si no había ninguna ID en la URI, entonces estamos ante una petición de todos los juegos o todas las listas de la base de datos. En el caso de las listas, tenemos que consultar todas, y, posteriormente, hacer un bucle que vaya comprobando los juegos de cada lista (usando su ID) y los vaya añadiendo tal y como hicimos antes. Convertimos el resultado a JSON y añadimos el campo de cabecera correspondiente.

Si la petición era a “games”, entonces tenemos que tener en cuenta si se indicaron parámetros adicionales. Como ya vimos, los parámetros que permitiremos son “sort”, que ordena la consulta; “limit”, que indica el número máximo de resultados que se devolverán; y “offset” que es el número de resultados que tenemos que saltarnos en la consulta. Estos dos últimos parámetros sirven para paginación. La consulta base que tenemos es “SELECT * FROM GAMES”, y tendremos que añadir lo que sea necesario según los parámetros que se hayan indicado. Para el caso del “sort”, simplemente tenemos que comprobar por qué columna hay que ordenar, por ejemplo con un “switch”, y añadir el “ORDER BY” correspondiente. Si el “offset” existe pero el “limit” no, hay que hacer una consulta paginada (como se vio en clase, anidando tres SELECT) pero sin usar la restricción de ROWNUM que limita el número de resultados, que sería el “<”. Si es al revés, entonces no usamos la restricción “>” (y además podemos anidar solo dos SELECT). Si ambos parámetros existen, entonces se usan ambas restricciones. La consulta resultante es la que usaremos. De nuevo, transformamos a JSON y añadimos el campo de cabecera necesario.

POST

Para el método POST vamos a usar el cuerpo de la petición, que ya teníamos extraído. Lo primero es comprobar si ya existe un juego o una lista con la ID del recurso proporcionado en la petición. Si es así, entonces tenemos que indicar que la petición es incorrecta, con el campo de cabecera “HTTP/1.1 400 Bad Request”. Si no, hay que añadir el juego o lista a la base de datos. Para añadir un juego, solamente tenemos que hacer un “INSERT” donde los valores de los distintos campos son los que se han enviado en el cuerpo de la petición. Para añadir una lista, solo hay que insertar la ID, ya que no vamos a permitir que se envíen listas con juegos ya incluidos. Una vez hecha la inserción del recurso, estableceremos el campo “HTTP/1.1 201 Created”, así como “Content-type: application/json”, y devolvemos el recurso creado (que como ya usamos “json_decode” al extraer la petición, tenemos que usar “json_encode” ahora). Si es una lista, antes hay que añadirle el campo “games” vacío.

Recordemos que en las listas teníamos también un POST para añadir un juego a una lista. En este caso, lo que nos interesa son las ID que se han

pasado en la URI. Como vimos, los valores de la URI plantilla los transformamos en un array. En este caso, el primer valor del array es la ID de la lista, y el segundo la ID del juego. Tras comprobar que la URI tenía parámetros y que había exactamente dos, tenemos que ver si existen tanto la lista como el juego. En caso contrario, indicamos en la cabecera un error 404. Si ambos existen, el siguiente paso es comprobar si el juego ya estaba en la lista. Si es así, enviamos el error 400. Si no, entonces añadimos a la tabla CONTAINS una simple fila con la ID generada por la secuencia y las ID de la lista y el juego. Posteriormente, para devolver la lista modificada, seguimos prácticamente los mismos pasos que en el GET de una lista: la buscamos por ID, añadimos los juegos que contiene, establecemos el código de estado 201 (aquí la única diferencia), indicamos el tipo de contenido, y la devolvemos convertida en JSON.

PUT

En nuestra API, solo el recurso “games” soporta la operación PUT. Dado que los juegos tienen pocas propiedades, la implementación de este método es muy simple. Necesitaremos de nuevo el cuerpo de la petición. Tenemos que comprobar primero que el juego exista, usando para ello la ID que encontramos en la petición (esta propiedad no se puede actualizar). Si no existe, enviamos un error 404. En caso de que sí exista, hacemos un “UPDATE” a la base de datos, donde actualizaremos los valores del juego correspondiente a la ID dada con los datos que se han pasado en la petición. Esta operación no necesita devolver nada, por lo que tenemos que establecer en la cabecera de la respuesta “HTTP/1.1 204 No Content”. De esta forma, indicamos que el cuerpo de la respuesta está vacío.

DELETE

Para esta operación necesitamos una ID dada en la URI, ya sea de un juego o de una lista. Por lo tanto, el primer paso en la implementación es comprobar si la URI tiene parámetros y si el array correspondiente tiene exactamente un elemento (veremos luego el caso en el que hay que eliminar un juego de una lista). Si la URI no tenía ningún parámetro,

tenemos que enviar el error 400 para indicar que la petición fue incorrecta. Si tenía una ID, lo siguiente es comprobar si existe un juego o una lista con esa ID. Si no existe, devolvemos un error 404. En caso contrario, en principio, tendríamos que eliminar la lista o el juego con la ID proporcionada. Sin embargo, tenemos que recordar que existe una tabla auxiliar, CONTAINS, para saber qué juegos contiene cada lista. Por lo tanto, antes de borrar de la base de datos el recurso especificado, es necesario eliminar de la tabla CONTAINS todas las filas que hagan referencia a dicho recurso. Alternativamente, podemos añadir a las dos claves ajenas de dicha tabla "ON DELETE CASCADE". De esta forma, la eliminación de un recurso que tenga alguna entrada en la tabla CONTAINS borrará también todas estas entradas automáticamente. Una vez eliminadas las filas necesarias, como no tenemos que devolver nada, de nuevo enviamos el código de estado 204.

Si lo que queremos es eliminar un juego de una lista, la URI tendrá dos parámetros. Igual que en el caso del POST, el primero corresponde a la ID de la lista objetivo, y el segundo a la ID del juego. Para empezar tenemos que comprobar que en la tabla CONTAINS exista una fila que contenga ambas ID, o sea, que la lista indicada contenga el juego especificado. Si no lo contiene, devolvemos el error 404. En caso de que lo contenga eliminamos la fila correspondiente, y enviamos el código de estado 204.

Consumo

Introducción

En esta sección vamos a consumir la API que hemos desarrollado. Vamos a utilizar, de nuevo, el lenguaje de scripts PHP. En este caso necesitaremos usar la biblioteca cURL ("client URL"), que dispone de multitud de funciones para comunicarnos con servidores, y algunas de ellas son especialmente útiles para consumir APIs. Veremos cómo realizar llamadas a cada una de las operaciones que hemos implementado anteriormente. Para ello, vamos a usar una página HTML simple con un formulario por cada una de las llamadas posibles. Para simplificar la gestión de estos formularios, tendremos una página PHP diferente para cada uno de ellos, y los llamaremos según la operación que realicen ("postjuego", "deletelista"…). En cada una de estas páginas usaremos las funciones cURL

necesarias para el consumo de la API. Antes de ver en detalle cómo efectuar cada llamada, vamos a ver las funciones de la librería cURL que son relevantes para nosotros:

- curl_init($url): inicia una nueva sesión cURL, con la URL indicada, y devuelve un manipulador para su posterior uso.

- curl_exec($ch): ejecuta la sesión que se ha pasado como parámetro y devuelve una respuesta.

- curl_close($ch): cierra la sesión y libera recursos.

- curl_setopt($ch, $opción, $valor): establece una opción con el valor especificado en la sesión dada. Las opciones tienen la estructura "CURLOPT_X", donde "X" es la opción en sí. Las más interesantes en nuestro caso son "RETURNTRANSFER" (si está a "true", la ejecución de la sesión devuelve el contenido de la respuesta, en caso contrario solo devuelve "true" o "false" en caso de éxito o error respectivamente), "POST" (para indicar que la petición usará dicho método), "POSTFIELDS" (para introducir datos en el cuerpo de la petición POST), "HTTPHEADER" (para establecer los campos de la cabecera de la petición con un array), y "CUSTOMREQUEST" (para realizar una petición diferente, como por ejemplo PUT o DELETE). También puede omitirse la URL al iniciar la sesión e indicarla usando este método con la opción "URL". Las opciones deben establecerse antes de ejecutar la sesión.

- curl_getinfo($ch, $info): obtiene cierta información sobre una sesión. En nuestro caso esta función la usaremos para conocer el código de estado devuelto por la API, usando como segundo parámetro "CURLINFO_HTTP_CODE".

Una vez vistas las funciones necesarias, podemos proceder a la ejecución de las diferentes llamadas.

GET

Las peticiones que utilizan este método son las que menos funciones requieren, ya que no es necesario enviar nada en el cuerpo ni especificar el método. Simplemente tenemos que iniciar una sesión cURL con la dirección del recurso que vayamos a usar. Si queremos buscar un juego o

una lista por ID, dado que hemos creado un formulario para ello, podemos acceder a lo que hayamos escrito con $_REQUEST (o $_POST y $_GET, según como hayamos hecho el formulario) y añadirlo a la URI. En el caso de la búsqueda de juegos con parámetros adicionales (que eran “sort”, “limit” y “offset”) tendremos que añadirlos también. Para mantener la estructura correcta de la petición en este caso, podemos extraer los parámetros que tengamos y luego, en función de cuántos haya, añadimos el “?” y los “&” según hagan falta. Posteriormente debemos establecer la opción “RETURNTRANSFER” a “true” para obtener la respuesta de la API. Después ejecutamos la sesión y guardamos la respuesta, obtenemos el código de estado como se indicó anteriormente, y cerramos la sesión. Por último, para visualizar la respuesta, hacemos un “echo” del código de error y otro de la respuesta en sí. Es conveniente poner en medio un salto de línea para ver mejor el resultado.

POST

Para crear un nuevo juego o lista, necesitaremos añadir un cuerpo a la petición, pero en el caso de que el POST sea para añadir un juego a una lista el proceso es más sencillo. Si queremos crear un nuevo recurso, primero iniciamos una sesión. Luego creamos una variable que será el cuerpo de la petición. Esta variable será un array donde las claves serán los atributos del recurso, y los valores los datos que hemos pasado en el formulario. Debemos establecer la opción “POST” a “true”, y lo mismo ocurre con “RETURNTRANSFER” (recordemos que al hacer un POST se devuelve el recurso creado). Es necesario también poner la opción “HTTPHEADER”, con un valor que sea un array con una cadena que será “Content-type: application/json” para indicar que el cuerpo tendrá formato JSON. Para indicar cuál es el cuerpo de la petición, usaremos la opción “POSTFIELDS” pasando como parámetro la variable que creamos anteriormente, convirtiéndola previamente a JSON con la función “json_encode”. Después ya podemos ejecutar la sesión y guardar la respuesta, obtener el código de estado, cerrar la sesión, y mostrar el código y la respuesta.

Para añadir un juego a una lista, como en el formulario indicamos ambas ID, solamente tenemos que añadirlas a la URI, y no establecer las opciones “HTTPHEADER” ni “POSTFIELDS” ya que no estamos enviando nada en el

cuerpo de la petición. El resto de los pasos necesarios son los mismos que para el POST explicado anteriormente.

PUT

Para actualizar un juego, también necesitamos que la petición tenga cuerpo. El procedimiento es prácticamente idéntico al que hemos seguido para hacer el POST. Tenemos que iniciar la sesión cURL y crear un array con los parámetros que recogemos del formulario. Como opciones, tenemos que establecer “HTTPHEADER” y “POSTFIELDS” de la misma forma que para el POST, pero en esta ocasión necesitamos usar “CUSTOMREQUEST” para especificar que la operación a realizar es un PUT. Para comprobar que, según implementamos en la API, esta operación no devuelve nada, pondremos “RETURNTRANSFER” a “true” para que al mostrar la respuesta veamos que no aparece nada, aunque desde un punto de vista práctico no es necesario. Ejecutamos la sesión, guardamos el código de estado, cerramos la sesión y mostramos el resultado.

DELETE

Dado que para esta operación no es necesario añadir nada al cuerpo de la petición, los pasos a seguir se asemejarán más a un GET. Como vamos a usar una URI plantilla, tenemos que recoger las ID necesarias del formulario y añadirlas al iniciar la sesión. Luego tenemos que usar “CUSTOMREQUEST” para indicar que el método a utilizar es DELETE, y, al igual que con el PUT, usaremos “RETURNTRANSFER” aunque no es necesario. Luego ejecutamos, obtenemos el código de estado, cerramos la sesión y hacemos “echo” del código y la respuesta.