Tutoriales programación

Tutoriales de programación en PHP y Android

domingo, 25 de agosto de 2013

Ejemplo PHP de servicio RESTful - Parte 1

Vamos a crear un sencillo servicio RESTful orientado a objetos, con el que podremos realizar las siguientes peticiones, utilizando para ello URL's amigables:
- Pedir una lista de usuarios mediante una petición GET. URL de la petición: http://localhost/login_restful/usuarios
- Crear un nuevo usuario mediante una petición POST en la que aportaremos los datos del usuario. El sistema asignará el identificador del recurso. URL de la petición: http://localhost/login_restful/crearUsuario
- Solicitar autenticación de un usuario mediante una petición POST y los datos necesarios para la autenticación. URL de la petición: http://localhost/login_restful/login
- Actualizar un usuario mediante una petición PUT a un determinado identificador de recurso. URL de la petición: http://localhost/login_restful/actualizarNombre/1
- Eliminar un usuario mediante una petición DELETE. URL de la petición: http://localhost/login_restful/borrarUsuario/1

Por cada petición vamos a recibir un objeto JSON ya sea informando de un error o dando información de que la petición se ha resuelto correctamente.

Nota: como vamos a crear un ejemplo sencillo no se ha tenido en cuenta que las operaciones solo las puede hacer un usuario autorizado del servicio. Por lo tanto, lo correcto sería que junto a cada petición se enviasen las credenciales del usuario que quiere realizar una operación en el servicio. Para comprobar si el usuario puede realizar dichas operaciones. Recuerda que un servicio RESTful no guarda el estado. Por lo tanto no es como una aplicación en la que una vez nos autenticamos se guardan las credenciales en una sesión. En cada petición se tienen que enviar las citadas credenciales del usuario.
Otra solución podría ser un sistema de envío de credenciales a través de las cabeceras.

Clase Rest

Vamos a empezar con la clase Rest. Esta clase se ocupa de dos tareas principalmente:
- Devolver las cabeceras con el código de estado y el resultado de la petición al cliente.
- Filtrar los datos enviados en la petición.

Los métodos para enviar las cabeceras y el resultado al cliente no tiene ninguna dificultad.
A) El método *mostrarRespuesta* recibe los parámetros $data, que contiene la respuesta JSON a enviar al cliente, y $estado que especifica el código de estado HTTP que acompañará a la respuesta. Este método se encarga de asignar el código de estado con el que se configurarán las cabeceras. Configurará las cabeceras que se van a enviar junto con la respuesta, mediante la llamada al método *setCabecera*. Y mostrará dicha respuesta.
B) El método *setCabecera* crea dos cabeceras que acompañarán a la respuesta de la petición. Para ello utilizará el código de estado asignado en el método *mostrarRespuesta* y la descripción del código obtenida mediante el método *getCodEstado*. Estas cabeceras no serán enviadas hasta que no se envíe la respuesta en *mostrarRespuesta* con la instrucción *echo $data.*
C) El método getCodEstado contiene un array asociativo donde las claves son los posibles códigos de estado y los valores son las descripciones asociadas a esos códigos. Por lo tanto a partir del código de estado que se enviará junto a las cabeceras y la respuesta, devolverá su descripción.

Los método encargados de limpiar los datos se encargan de sanear los datos que acompañan a las peticiones GET, POST, PUT y DELETE.
A) El método *tratarEntrada* se encarga de sanear el array datos de entrada llamando al método *limpiarEntrada* y asigna dicho array de datos al atributo **$datosPetición**. Para ello primero comprueba cual es el método de petición ($_SERVER['REQUEST_METHOD']) y pasa los datos del array superglobal de la petición a *limpiarEntrada*. Esto quiere decir que si el método de estrada es GET se tratarán los datos del array $_GET. Y si es POST se tratarán los datos que puedan estar contenidos en $_POST.
B) El método *limpiarEntrada* se encarga de sanear los datos que se le pasen como parámetro. Es un método recursivo para tratar cada uno de los valores de un array.

```
<?php
class Rest {
```

```
  public $tipo = "application/json";
  public $datosPeticion = array();
  private $_codEstado = 200;
  public function __construct() {
    $this->tratarEntrada();
  }
  public function mostrarRespuesta($data, $estado) {
    $this->_codEstado = ($estado) ? $estado : 200;//si no se envía $estado por de
fecto será 200
    $this->setCabecera();
    echo $data;
    exit;
  }
  private function setCabecera() {
    header("HTTP/1.1 " . $this->_codEstado . " " . $this->getCodEstado());
    header("Content-Type:" . $this->tipo . ';charset=utf-8');
  }
  private function limpiarEntrada($data) {
    $entrada = array();
    if (is_array($data)) {
      foreach ($data as $key => $value) {
        $entrada[$key] = $this->limpiarEntrada($value);
      }
    } else {
      if (get_magic_quotes_gpc()) {
        //Quitamos las barras de un string con comillas escapadas
        //Aunque actualmente se desaconseja su uso, muchos servidores tienen acti
vada la extensión magic_quotes_gpc.
        //Cuando esta extensión está activada, PHP añade automáticamente caracter
es de escape (\) delante de las comillas que se escriban en un campo de formulario
.
        $data = trim(stripslashes($data));
      }
      //eliminamos etiquetas html y php
      $data = strip_tags($data);
      //Conviertimos todos los caracteres aplicables a entidades HTML
      $data = htmlentities($data);
      $entrada = trim($data);
    }
    return $entrada;
  }
  private function tratarEntrada() {
    $metodo = $_SERVER['REQUEST_METHOD'];
    switch ($metodo) {
      case "GET":
        $this->datosPeticion = $this->limpiarEntrada($_GET);
        break;
      case "POST":
        $this->datosPeticion = $this->limpiarEntrada($_POST);
        break;
      case "DELETE"://"falling though". Se ejecutará el case siguiente
      case "PUT":
        //php no tiene un método propiamente dicho para leer una petición PUT o D
ELETE por lo que se usa un "truco":
        //leer el stream de entrada file_get_contents("php://input") que transfie
re un fichero a una cadena.
        //Con ello obtenemos una cadena de pares clave valor de variables (variab
le1=dato1&variable2=data2...)
        //que evidentemente tendremos que transformarla a un array asociativo.
        //Con parse_str meteremos la cadena en un array donde cada par de element
os es un componente del array.
        parse_str(file_get_contents("php://input"), $this->datosPeticion);
        $this->datosPeticion = $this->limpiarEntrada($this->datosPeticion);
        break;
      default:
        $this->response('', 404);
        break;
    }
  }
  private function getCodEstado() {
    $estado = array(
      200 => 'OK',
      201 => 'Created',
      202 => 'Accepted',
      204 => 'No Content',
```


Etiquetas

```
      301 => 'Moved Permanently',
      302 => 'Found',
      303 => 'See Other',
      304 => 'Not Modified',
      400 => 'Bad Request',
      401 => 'Unauthorized',
      403 => 'Forbidden',
      404 => 'Not Found',
      405 => 'Method Not Allowed',
      500 => 'Internal Server Error');
    $respuesta = ($estado[$this->_codEstado]) ? $estado[$this->_codEstado] : $est
ado[500];
    return $respuesta;
  }
}
?>
```

.htaccess

Como comentamos al principio del tutorial, queremos que el servicio pueda ser utilizado mediante URL's amigables. Para ello deberemos de especificar una regla de reescritura de URL's en un fichero .htaccess. Que lo definiremos en la raíz del proyecto.

```
RewriteEngine on
RewriteBase /login_restful/
RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-l
RewriteRule ^(.+)$ Api.php?url=$1 [QSA,NC,L]
```

Con esta regla de reescritura estamos especificando:
- El directorio base es /login_restful/
- Se han añadido tres condiciones para restringir la reescritura sólo a rutas que no existan previamente. Es decir, que no valdría realizar reescritura, por ejemplo, para www.dominio.com/img/img.png (suponemos que esta ruta y recurso existe). La primera condición previene los directorios que ya existan con la bandera !-d. La segunda condición hace que se ignoren ficheros que ya existan con la bandera !-f. Y la tercera condición hace que se ignoren los enlaces simbólicos que ya existan con !-l.
- Luego con la regla de reescritura transformaremos una URL amigable a una URL con la que el servidor pueda trabajar. Por lo tanto http://localhost/login_rest/borrarUsuario/1 será transformado internamente a http://localhost/login_restful/Api.php?url=borrarUsuario/1. Ya que /borrarUsuario/1 será tomado como referencia $1.
Para más información mirar el conjunto de tutoriales de reescritura.

Clase Api

Anteriormente definimos la clase Rest y vamos a utilizarla como clase base a heredar. De esta forma tendremos disponibles tanto sus métodos como atributos. La clase que vamos a definir a continuación (Api) va a ser la encargada de:
- Conectar con la base de datos
- Determinar que método debe de utilizar para resolver la petición, a partir de la url de dicha petición.
- Utilizar los métodos heredados de la clase padre para sanear los datos de entrada de la petición y devolver el resultado al cliente.

A) El método *conectarDB* como su propio indica, se encarga de conectar con la base de datos que contendrán los recursos del servicio. Para ello utilizaremos PDO como librería de abstracción para tratar con la base de datos MySQL.
B) El método *devolverError* utiliza un identificador recibido como parámetro para devolver un array asociativo con dos elementos: elemento 'estado' con valor error y elementos 'msg' con la descripción de error.
C) El método *procesarLLamada* es uno de los métodos más importantes ya que es el encargado de procesar la URL de la petición y a partir de ella se llamará al método que resolverá la citada petición. Como hemos comentado en el apartado anterior, si se hace una petición a la URL http://localhost/login_rest/borrarUsuario/1, esta URL será transformado internamente a http://localhost/login_restful/Api.php?url=borrarUsuario/1. Por lo tanto, si obtenemos el valor de la variable global 'url' ($_REQUEST['url']) obtendremos borrarUsuario/1. Y esto nos puede sugerir que necesitamos invocar al método borrarUsuario con el 1 como argumento para resolver la petición. Evidentemente no todas las URL estarán asociadas a llamadas a métodos con argumentos. Por ejemplo: http://localhost/login_rest/usuarios.

Los siguientes métodos de la clase son los encargados de realizar cada una de la operaciones asociadas con las peticiones.
A) El método *usuarios* comprueba si se ha accedido a él mediante una petición GET y devuelve un objeto JSON con los datos de los usuarios del servicio.
B) El método *login* comprueba si se ha accedido a él mediante una petición POST, comprueba si lo datos que acompañan a dicha petición ($datosPeticion) son adecuados y devuelve un objeto JSON indicando si el usuario existe.
C) El método *actualizarNombre($id)* tras comprobar que se ha accedido a él con una petición PUT, utiliza los datos pasados junto a dicha petición y el parámetro $id, extraído de la URL en

- petición (2)
- PHP (43)
- plantilla (3)
- POO (19)
- POST (2)
- proveedor de contenido (1)
- recursos (1)
- redirección (2)
- reescritura URL (8)
- RESTful (5)
- routing (1)
- saneamiento (3)
- seguridad (10)
- serializar (1)
- servicio (1)
- servicio web (1)
- sesión (2)
- SQL (7)
- SQL injection (1)
- static (4)
- Symfony 2 (7)
- tema (1)
- trait (1)
- transacción (2)
- Twig (1)
- validación (1)
- view (3)
- visibilidad (5)
- vista (4)
- XML (1)
- XSS (1)

procesarLLamada, para actualizar el nombre de un usuario existente (con dicho identificador). Posteriormente devuelve un objeto JSON confirmando que se ha actualizado los datos de un usuario.
D) El método *borrarUsuario($id)* tras comprobar que se ha accedido a él con una petición DELETE, utiliza el parámetro $id, extraído de la URL en *procesarLLamada,* para borrar el usuario con dicho identificador. Posteriormente devuelve un objeto JSON confirmando que se ha borrado el usuario.
E) Finalmente el método *crearUsuario* tras comprobar que se ha accedido a él mediante una petición POST, utiliza los datos pasados junto a dicha petición para crear a un nuevo usuario. Posteriormente se devuelve un objeto confirmando la creación del usuario.

Finalmente se creará una instancia de la clase Api y se llamará al método *procesarLLamada.* Esto es necesario ya que cada vez que se realice una petición a una URL del servicio, gracias a la regla de reescritura se llama al script Api.php. Y de esta manera se creará el objeto Api y se invocará al método que procesa la URL y llama al encargado de resolver la petición.

```
<?php
require_once("Rest.php");
class Api extends Rest {
  const servidor = "localhost";
  const usuario_db = "usuario";
  const pwd_db = "";
  const nombre_db = "autorizacion";
  private $_conn = NULL;
  private $_metodo;
  private $_argumentos;
  public function __construct() {
    parent::__construct();
    $this->conectarDB();
  }
  private function conectarDB() {
    $dsn = 'mysql:dbname=' . self::nombre_db . ';host=' . self::servidor;
    try {
      $this->_conn = new PDO($dsn, self::usuario_db, self::pwd_db);
    } catch (PDOException $e) {
      echo 'Falló la conexión: ' . $e->getMessage();
    }
  }
  private function devolverError($id) {
    $errores = array(
      array('estado' => "error", "msg" => "petición no encontrada"),
      array('estado' => "error", "msg" => "petición no aceptada"),
      array('estado' => "error", "msg" => "petición sin contenido"),
      array('estado' => "error", "msg" => "email o password incorrectos"),
      array('estado' => "error", "msg" => "error borrando usuario"),
      array('estado' => "error", "msg" => "error actualizando nombre de usuario")
,
      array('estado' => "error", "msg" => "error buscando usuario por email"),
      array('estado' => "error", "msg" => "error creando usuario"),
      array('estado' => "error", "msg" => "usuario ya existe")
    );
    return $errores[$id];
  }
  public function procesarLLamada() {
    if (isset($_REQUEST['url'])) {
      //si por ejemplo pasamos explode('/','////controller///method////args///')
el resultado es un array con elem vacios;
      //Array ( [0] => [1] => [2] => [3] => [4] => controller [5] => [6] => [7] =
> method [8] => [9] => [10] => [11] => args [12] => [13] => [14] => )
      $url = explode('/', trim($_REQUEST['url']));
      //con array_filter() filtramos elementos de un array pasando función callba
ck, que es opcional.
      //si no le pasamos función callback, los elementos false o vacios del array
 serán borrados
      //por lo tanto la entre la anterior función (explode) y esta eliminamos los
 '/' sobrantes de la URL
      $url = array_filter($url);
      $this->_metodo = strtolower(array_shift($url));
      $this->_argumentos = $url;
      $func = $this->_metodo;
      if ((int) method_exists($this, $func) > 0) {
        if (count($this->_argumentos) > 0) {
          call_user_func_array(array($this, $this->_metodo), $this->_argumentos);

        } else {//si no lo llamamos sin argumentos, al metodo del controlador
          call_user_func(array($this, $this->_metodo));
        }
      }
```

```
      else
        $this->mostrarRespuesta($this->convertirJson($this->devolverError(0)), 40
4);
    }
    $this->mostrarRespuesta($this->convertirJson($this->devolverError(0)), 404);

  }
  private function convertirJson($data) {
    return json_encode($data);
  }

  private function usuarios() {
    if ($_SERVER['REQUEST_METHOD'] != "GET") {
      $this->mostrarRespuesta($this->convertirJson($this->devolverError(1)), 405)
;
    }
    $query = $this->_conn->query("SELECT id, nombre, email FROM usuario");
    $filas = $query->fetchAll(PDO::FETCH_ASSOC);
    $num = count($filas);
    if ($num > 0) {
      $respuesta['estado'] = 'correcto';
      $respuesta['usuarios'] = $filas;
      $this->mostrarRespuesta($this->convertirJson($respuesta), 200);
    }
    $this->mostrarRespuesta($this->devolverError(2), 204);
  }

  private function login() {
    if ($_SERVER['REQUEST_METHOD'] != "POST") {
      $this->mostrarRespuesta($this->convertirJson($this->devolverError(1)), 405)
;
    }
    if (isset($this->datosPeticion['email'], $this->datosPeticion['pwd'])) {
   //el constructor del padre ya se encarga de sanear los datos de entrada
      $email = $this->datosPeticion['email'];
      $pwd = $this->datosPeticion['pwd'];
      if (!empty($email) and !empty($pwd)) {
        if (filter_var($email, FILTER_VALIDATE_EMAIL)) {
          //consulta preparada ya hace mysqli_real_escape()
          $query = $this->_conn->prepare("SELECT id, nombre, email, fRegistro FRO
M usuario WHERE
          email=:email AND password=:pwd ");
          $query->bindValue(":email", $email);
          $query->bindValue(":pwd", sha1($pwd));
          $query->execute();
          if ($fila = $query->fetch(PDO::FETCH_ASSOC)) {
            $respuesta['estado'] = 'correcto';
            $respuesta['msg'] = 'datos pertenecen a usuario registrado';
            $respuesta['usuario']['id'] = $fila['id'];
            $respuesta['usuario']['nombre'] = $fila['nombre'];
            $respuesta['usuario']['email'] = $fila['email'];
            $this->mostrarRespuesta($this->convertirJson($respuesta), 200);
          }
        }
      }
    }
    $this->mostrarRespuesta($this->convertirJson($this->devolverError(3)), 400);

  }

  private function actualizarNombre($idUsuario) {
    if ($_SERVER['REQUEST_METHOD'] != "PUT") {
      $this->mostrarRespuesta($this->convertirJson($this->devolverError(1)), 405)
;
    }
    //echo $idUsuario . "<br/>";
    if (isset($this->datosPeticion['nombre'])) {
      $nombre = $this->datosPeticion['nombre'];
      $id = (int) $idUsuario;
      if (!empty($nombre) and $id > 0) {
        $query = $this->_conn->prepare("update usuario set nombre=:nombre WHERE i
d =:id");
        $query->bindValue(":nombre", $nombre);
        $query->bindValue(":id", $id);
        $query->execute();
```

```
        $filasActualizadas = $query->rowCount();
        if ($filasActualizadas == 1) {
          $resp = array('estado' => "correcto", "msg" => "nombre de usuario actua
lizado correctamente.");
          $this->mostrarRespuesta($this->convertirJson($resp), 200);
        } else {
          $this->mostrarRespuesta($this->convertirJson($this->devolverError(5)),
400);
        }
      }
    }
    $this->mostrarRespuesta($this->convertirJson($this->devolverError(5)), 400);

  }

  private function borrarUsuario($idUsuario) {
    if ($_SERVER['REQUEST_METHOD'] != "DELETE") {
      $this->mostrarRespuesta($this->convertirJson($this->devolverError(1)), 405)
;
    }
    $id = (int) $idUsuario;
    if ($id >= 0) {
      $query = $this->_conn->prepare("delete from usuario WHERE id =:id");
      $query->bindValue(":id", $id);
      $query->execute();
      //rowcount para insert, delete. update
      $filasBorradas = $query->rowCount();
      if ($filasBorradas == 1) {
        $resp = array('estado' => "correcto", "msg" => "usuario borrado correctam
ente.");
        $this->mostrarRespuesta($this->convertirJson($resp), 200);
      } else {
        $this->mostrarRespuesta($this->convertirJson($this->devolverError(4)), 40
0);
      }
    }
    $this->mostrarRespuesta($this->convertirJson($this->devolverError(4)), 400);

  }
  private function existeUsuario($email) {
    if (filter_var($email, FILTER_VALIDATE_EMAIL)) {
      $query = $this->_conn->prepare("SELECT email from usuario WHERE email = :em
ail");
      $query->bindValue(":email", $email);
      $query->execute();
      if ($query->fetch(PDO::FETCH_ASSOC)) {
        return true;
      }
    }
    else
      return false;
  }

  private function crearUsuario() {
    if ($_SERVER['REQUEST_METHOD'] != "POST") {
      $this->mostrarRespuesta($this->convertirJson($this->devolverError(1)), 405)
;
    }
    if (isset($this->datosPeticion['nombre'], $this->datosPeticion['email'], $thi
s->datosPeticion['pwd'])) {
      $nombre = $this->datosPeticion['nombre'];
      $pwd = $this->datosPeticion['pwd'];
      $email = $this->datosPeticion['email'];
      if (!$this->existeUsuario($email)) {
        $query = $this->_conn->prepare("INSERT into usuario (nombre,email,passwor
d,fRegistro) VALUES (:nombre, :email, :pwd, NOW())");
        $query->bindValue(":nombre", $nombre);
        $query->bindValue(":email", $email);
        $query->bindValue(":pwd", sha1($pwd));
        $query->execute();
        if ($query->rowCount() == 1) {
          $id = $this->_conn->lastInsertId();
          $respuesta['estado'] = 'correcto';
          $respuesta['msg'] = 'usuario creado correctamente';
          $respuesta['usuario']['id'] = $id;
```

```
          $respuesta['usuario']['nombre'] = $nombre;
          $respuesta['usuario']['email'] = $email;
          $this->mostrarRespuesta($this->convertirJson($respuesta), 200);
        }
        else
          $this->mostrarRespuesta($this->convertirJson($this->devolverError(7)),
400);
      }
      else
        $this->mostrarRespuesta($this->convertirJson($this->devolverError(8)), 40
0);
    } else {
      $this->mostrarRespuesta($this->convertirJson($this->devolverError(7)), 400)
;
    }
  }
}
$api = new Api();
$api->procesarLLamada();
```

Hemos creado el servicio para que acepte URL's amigables pero evidentemente podemos utilizar URL's de este tipo http://localhost/login_restful/Api.php?url=actualizarNombre/7 para realizar las peticiones. Pero no http://localhost/login_restful/Api.php?url=actualizarNombre&id=7 ya que el servicio no ha sido diseñado de esta forma.

Entradas relacionadas

Ejemplo PHP de servicio RESTful - Parte 2
PDO - Parte 1
PDO - Parte 2
Reglas de reescritura - Parte 1: Introducción
Reglas de reescritura - Parte 2: Directivas
Reglas de reescritura - Parte 3: Ejemplos completos
Seguridad PHP - Validación y filtrado 1
Seguridad PHP - Validación y filtrado 2
PHP orientado a objetos - Introducción
PHP orientado a objetos - Herencia
Introducción servicios RESTful

Publicado por Iván Posilio Gellida en domingo, agosto 25, 2013
Etiquetas: cabecera, herencia, PDO, PHP, POO, reescritura URL, RESTful, saneamiento

7 comentarios:

Javier ™ 9 de junio de 2014, 22:38

Hola muchísimas gracias, me ha ayudado mucho a comprender cómo funciona REST.

Sin al listar los datos, en caso de que en la tabla haya algún nombre con tilde se obtiene null, y no entiendo por qué... quizás me lo puedas aclarar?

Gracias !

Responder

David Acurero 11 de mayo de 2015, 19:49

Excelente post, veo que tiene algo de tiempo y pocos comentarios, espero lean este, tengo problemas con el POST y el PUT, no he podido resolverlo y ya no se que puede ser, la variable $datosPeticion no se esta cargando con los datos se le envian, el array siempre aparece vacio y por ende no entra en la condicion para hacer login.

Que podra ser?, help please

Responder

Respuestas

jorge almonacid 11 de enero de 2016, 16:31

Me encuentro en la misma situación, lograste solucionarlo? cree un cliente javascript que interactúe con el servidor pero no logro solucionar ese error

Germán Quintos López 18 de mayo de 2016, 17:57

Hola amigo, que error te arroja?
Probé la función de Login y funciona todo perfecto (con unos cambios en la función Login() de Api.php, pero unicamente en la consulta)
Rest.php no lo toqué

Responder

Hernán Aguilar 26 de septiembre de 2016, 7:25

Sigo sin enteder Rest y Soap..para que sirve exactamente? si supuestamente queremos pedir info de un sistema o otro..porque no realizar peticiones a la base da datos? o es una manera de obtener informacion de otro sistema sin tener que entrar a la base de datos?.

Responder

Respuestas

Guillermo Hernández 7 de noviembre de 2016, 19:08

Es para consumir servicios.
Imagina que estás desarrollando una tienda en línea en la que se podrá pagar con tarjeta; por lo tanto necesitas acceso a la base de datos del banco para poder hacer los cargos. Pero el banco obviamente no te va a dar ese acceso, en vez de eso lo que hace el banco es "exponer un servicio" que haga los cargos a la tarjeta y te regrese si el cargo fue exitoso o no, y tú desde tu sitio lo consumas.
Generalmente para exponer servicios se usa SOAP o REST.

Guillermo Hernández 7 de noviembre de 2016, 19:10

Es para consumir servicios.
Imagina que estás desarrollando una tienda en línea en la que se podrá pagar con tarjeta; por lo tanto necesitas acceso a la base de datos del banco para poder hacer los cargos. Pero el banco obviamente no te va a dar ese acceso, en vez de eso lo que hace el banco es "exponer un servicio" que haga los cargos a la tarjeta y te regrese si el cargo fue exitoso o no, y tú desde tu sitio lo consumas.
Generalmente para exponer servicios se usa SOAP o REST.

Responder

Introduce tu comentario...

Comentar como: Cuenta de Goc ▾

Publicar Vista previa

Enlaces a esta entrada

Crear un enlace

Entrada más reciente Página principal Entrada antigua

Suscribirse a: Enviar comentarios (Atom)