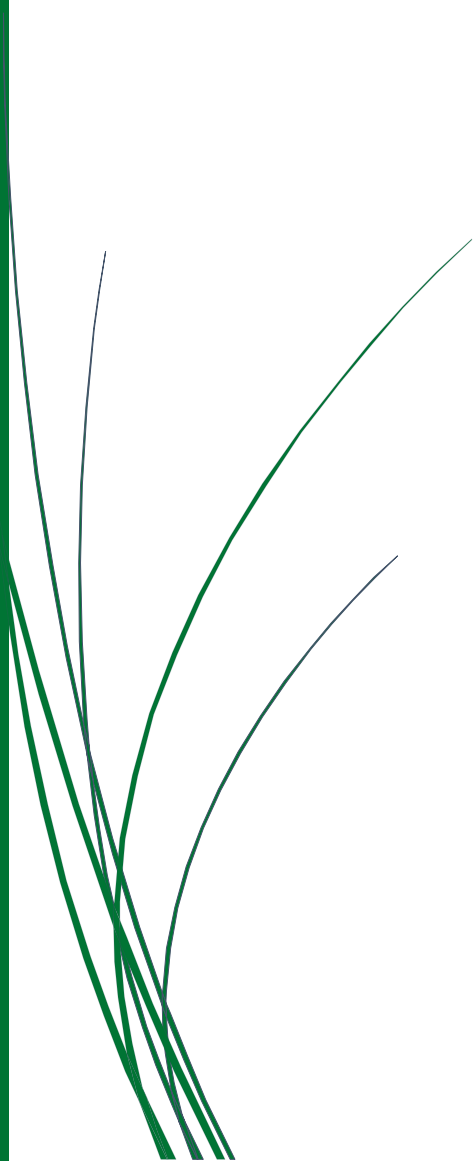
https://github.com/Apretaste

http://facebook.com/apretaste

8 de marzo del 2016

Cómo crear un servicio para Apretaste!



por Salvi Pascual

COMO CREAR UN SERVICIO

Guía paso a paso para programadores

CONTENIDO

COMO CREAR UN SERVICIO

Guía paso a paso para programadores

PREÁMBULO

Este documento es una guía paso a paso de cómo crear un servicio para la plataforma Apretaste (http://apretaste.com), la cual es una red de servicios interconectados que comparten información con usuarios que acceden a los mismos. Es bien parecido a la Web, solo que toda la información se transfiere por medio del email. Hasta ahora Apretaste ofrece diversos servicios, entre ellos Wikipedia, Mapas, Chistes, Traducción, Noticias y más. Apretaste es usado en Cuba, pero puede ser accedido en cualquier parte del mundo.

Cuando un usuario desea acceder a un servicio de Apretaste, digamos a Wikipedia, el usuario envía un email a apretaste@gmail.com poniendo en el asunto del email la palabra *wikipedia* seguida de la frase a buscar. Por ejemplo: **WIKIPEDIA jose marti**. Apretaste automáticamente busca un artículo sobre "jose marti" en Wikipedia y lo retorna mediante un email.

A continuación te explicamos paso a paso como crear un servicio. Asumimos que eres programador y sabes programar en PHP (http://php.net), trabajar con archivos XML y utilizar el motor de plantillas Smarty (http://smarty.net). No obstante incluimos en los anexos una pequeña guía de cómo utilizar Smarty.

Aunque no es imprescindible, sería bueno que hayas usado alguna vez Apretaste y tengas experiencia como usuario.

COMO CREAR UN SERVICIO

Guía paso a paso para programadores

PASO 1: PREPARA TU AMBIENTE DE TRABAJO

Apretaste está programado en PHP 5.4+, y debe ser utilizado con el servidor web Apache 2. Además utiliza una base de datos en MySQL 5. Entonces, lo primero que debes hacer es tener instalado y configurado tu servidor web con Apache, PHP y MySQL. En la web puedes encontrar muchas formas de hacerlo, aunque es muy probable que ya lo tengas :). En el caso de los que usen Microsoft Windows como sistema operativo, recomendamos la distribución XAMPP (https://www.apachefriends.org/es/download.html).

Apretaste además está programado sobre el *framework* Phalcon. Debes descargar Phalcon PHP Framework desde su sitio web oficial http://phalconphp.com. Descarga la versión que corresponda con las características de tu PC. Debes configurar tu PHP para que trabaje con Phalcon. Este *framework* es una extensión para PHP, así que bastará con colocar el binario descargado (.DLL o .SO) en la carpeta de extensiones de php (php/ext), editar el archivo de configuración de PHP (php.ini) y agregar una línea similar a **extension=php_phalcon.dll** o **extension=php_phalcon.so**. No te será necesario por ahora aprender a usar ese *framework* para poder hacer un servicio en Apretaste.

Contamos con un "sandbox", se trata de una versión de Apretaste para que los programadores puedan desarrollar servicios. La última versión siempre se encuentra en GitHub (https://github.com/Apretaste/sandbox) por lo que puedes clonarlo si ya tienes Git instalado en tu PC (**git clone https://github.com/Apretaste/sandbox.git**) o descargar el último lanzamiento (*release*). Cuando lo tengas colócalo en tu servidor Apache y configúralo para que pueda ser accedido desde tu navegador web preferido.

Posiblemente debas configurar el módulo Rewrite de Apache para permitir URL limpias, pues el Sandbox de Apretaste lo necesita. Recomendamos que configures un virtual host (vhost) en Apache para acceder a tu sandbox mediante una URL dedicada, por ejemplo http://apretaste.local, o utilizarlo como sitio por defecto. Evita configurar el sandbox como un "alias" o una subcarpeta.

Cuando tengas todo eso configurado accede desde tu navegador a http://apretaste.local/run. En el formulario que te aparece escribe "PRIMO 5" sin las comillas en el campo **Subject** y haz clic en **Test service.**

PASO 2: DEFINE EL NOMBRE DEL SERVICIO QUE VAS A CREAR

Define un nombre, es decir, **la primera palabra en el asunto del correo** que utilizarán los usuarios para utilizar tu servicio. Ese nombre debe contener **solo letras y números, sin espacio, empezando con una letra** y preferiblemente en minúsculas. El nombre debe ser representativo al servicio que quieres brindar y no debe ser ninguno de los servicios que ya existe en la plataforma, es decir, **debe ser único**.

Por ejemplo, utilizaremos "primo", como un servicio que permitirá saber si un número es primo. Un número es primo si es solo divisible por él y por el número 1, como por ejemplo, el 3.

Este paso 1 es importante, pues ese nombre debe coincidir con en el archivo de configuración y con el nombre de la clase PHP que crearás en los siguientes pasos.

PASO 3: CREA LA ESTRUCTURA DE DIRECTORIOS

Elije un lugar en tu PC para crear el servicio, creando **una carpeta con el mismo nombre definido en el paso 1**. Aunque este nombre de carpeta no es obligatorio te sugerimos que lo hagas como buena práctica. Luego crea 2 archivos llamados ***service.php*** y ***config.xml*** dentro de la carpeta creada anteriormente. Además crea una carpeta llamada ***templates*** y dentro un archivo llamado ***basic.tpl.*** Al concluir este paso debes tener la estructura de archivos siguiente:

```
- primo
      - templates
            - basic.tpl
      - config.xml
      - service.php
```

PASO 4. EDITA EL ARCHIVO DE CONFIGURACIÓN CONFIG.XML

Abre el archivo config.xml con tu editor de código XML preferido (Notepad será suficiente) y pega el siguiente código:

```
<service>
      <serviceName>primo</serviceName>
      <serviceDescription>Permite saber si un número es primo</serviceDescription>
      <creatorEmail>tucorreo@electronico.com</creatorEmail>
      <serviceCategory>academico</serviceCategory>
      <serviceUsage>
            <![CDATA[
                  Escriba un nuevo email a {APRETASTE_EMAIL} y en el asunto
                  ponga la palabra PRIMO seguida de un número entero.

                  Por ejemplo:
                  Para: {APRETASTE_EMAIL}
                  Asunto: PRIMO 5

                  Deberá recibir un email con respuesta la respuesta de
                  si 5 es primo un número primo o no.
            ]]>
      </serviceUsage>
</service>
```

Luego cambia las propiedades en el archivo config.xml que se adecúen para tu servicio, es decir:

a. **serviceName**, es el nombre del servicio y es requerido, el que definiste en el paso 1.
b. **serviceDescription**, no es requerido, pero es una descripción corta de tu servicio y aparecerá en la lista de servicios cuando los usuarios la soliciten.
c. **creatorEmail**, debe ser tu dirección de correo electrónico, es decir, una dirección de correo válida. Se mostrará a los usuarios que usen el servicio y se utilizará para contactarte en caso necesario.
d. **serviceCategory**, es la categoría del servicio y es requerida. Las posibles categorías son: *negocios, ocio, academico, social, comunicaciones, informativo, adulto, otros.*
e. **serviceUsage**, es un texto que ayuda a cómo utilizar tu servicio y que los usuarios podrán solicitar. No es requerido pero sugerimos que lo escribas. Debes hacerlo dentro de las etiquetas **<![CDATA[** y **]]>**. Si lees la descripción en el ejemplo, aprenderás como los usuarios usan Apretaste y su funcionamiento básico.

Por último, Apretaste sustituirá el texto **{APRETASTE_EMAIL}** dentro del archivo config.xml por alguna de las direcciones válidas a las cuales los usuarios escriben para utilizar la plataforma.

COMO CREAR UN SERVICIO

Guía paso a paso para programadores

PASO 4: PROGRAMA TU SERVICIO

Abre el archivo ***service.php*** y pega el siguiente código:

```php
<?php
class primo extends Service
{

    public function _main (Request $request)
    {
        $es_primo = true;
        $numero = intval($request->query);
        for ($i = 2; $i <= intval($numero / 2); $i ++)
            if ($numero % $i == 0) {
                $es_primo = false;
                break;
            }
        $respuesta = "El numero $numero " . ($es_primo ? "es" : "no es") . " primo";
        $response = new Response();
        $response->setResponseSubject($respuesta);
        $response->createFromTemplate("basic.tpl", array(
                "respuesta" => $respuesta
        ));
        return $response;
    }
}
```

Observa el código y nota que la clase PHP creada se llama igual que el nombre de tu servicio y hereda de la clase ***Service***. Esta clase debe tener por defecto el método público **_main** el cual será llamado cuando utilicen tu servicio. El parámetro que recibe este método es un objeto de tipo **Request** (Ver Anexo 2) que contiene la información del email recibido por Apretaste. Cuando se termina de procesar la petición, se retorna un objeto de tipo **Response** que contiene la información de respuesta al usuario.

PASO 5: DEFINE TU BASE DE DATOS (OPCIONAL)

Tu servicio puede opcionalmente utilizar una base de datos. Para ello debes crear un archivo SQL que generen tus tablas, vistas y funciones necesarias y guardarlo en la carpeta de tu servicio, por ejemplo, en **primo/primo.sql.** Al hacer esto el equipo de Apretaste entiende que necesitas correr ese SQL en la base de datos de la plataforma por lo que será revisado previamente. La regla fundamental para que tu código SQL esté correcto es que todos los objetos de base de datos que crees deben tener como prefijo **dos guiones bajos seguido del nombre del servicio y luego de un guion bajo.** En el ejemplo que estamos trabajando sería caso **__primo_**.

Por ejemplo, supón que quieras guardar estadísticas del uso de tu servicio. En el archivo primo/primo.sql estaría el siguiente código:

```
CREATE TABLE __primo_estadistica(
	id int(11) PRIMARY KEY AUTO_INCREMENT,
	fecha timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
	usuario varchar(255) NOT NULL,
	solicitud varchar(255) NOT NULL
);
```

Para ejecutar una sentencia SQL durante la ejecución de tu servicio debes:

1. Conectarte a la db: **$db = new Connection();**
2. Ejecutar la sentencia SQL y obtener el resultado en una variable cuando sea una instrucción SELECT. El resultado es un arreglo de objetos:

```
$db->deepQuery("INSERT INTO __primo_estadistica (usuario, solicitud)
	VALUES ('{$request->email}','{$request->query}')");
$r = $db->deepQuery("SELECT * FROM __primo_estadistica
	WHERE usuario ='{$request->email}';");
if ($r !== false)
   foreach($r as $item)
	$solicitud = $item->solicitud;
```

PASO 6: EDITA TU PLANTILLA

En el paso anterior, la plantilla utilizada para responderle al usuario fue *basic.tpl* ubicada en la carpeta *templates*. Abre ese archivo para editarlo y pega el siguiente código:

```
<h1>Numeros primos</h1>
<p>{$respuesta}</p>
```

Las plantillas deben ser creadas para el motor Smarty. Al utilizar el método createFromTemplate() de la clase Response, el segundo parámetro es un arreglo asociativo, donde el índice **"respuesta"** es una variable para Smarty (Ver Anexo 1), el cual sustituirá con su valor en la plantilla donde esté el código **{$respuesta}**. Puedes definir en ese arreglo tantas variables como desees.

Además, puedes crear otras plantillas si tu servicio necesita dar distintos tipos de respuesta. En el ejemplo que tratamos en esta guía, otra posible respuesta sería cuando el usuario no envió el número en el asunto del correo después de la palabra PRIMO, y se le debe responder diciéndole de su error.

Existen otros tags especializados (propios de Apretaste) que puedes utilizar en tus plantillas (Ver Anexo 3). Tal es el caso de lis **enlaces**, **botones** e **imágenes**.

Supón que quieras incrustar un enlace a otro servicio de Apretaste. El siguiente código genera un enlace (link) que dirá "Ver mi perfil", el cual prepara un correo con el asunto PERFIL, listo para enviarlo hacia Apretaste:

```
{link href="PERFIL" caption="Ver mi perfil"}
```

Si quisieras que luciera como un botón en vez de un enlace, podrías poner:

```
{button href="PERFIL" caption="Ver mi perfil" color="blue"}
```

PASO 7: AGREGA UN SUBSERVICIO (OPCIONAL)

Este paso es opcional, pero es posible que lo necesites y aprenderás aspectos nuevos. La clase de tu servicio (en el ejemplo la clase **primo**) puede contener tantos métodos y propiedades como desees. Define los métodos privados cuando no deban ser utilizados desde fuera de la clase. Por ejemplo, cambiemos un poco el código de la clase extrayendo la parte donde se decide si un número es primo o no. Observa cómo queda:

```php
<?php

class primo extends Service
{

    public function _main (Request $request)
    {
        $numero = intval($request->query);
        $es_primo = $this->esPrimo($numero);

        $respuesta = "El numero $numero " . ($es_primo ? "es" : "no es") . " primo";

        $response = new Response();
        $response->setResponseSubject($respuesta);
        $response->createFromTemplate("basic.tpl", array(
                "respuesta" => $respuesta
        ));
        return $response;
    }

    private function esPrimo ($numero)
    {
        for ($i = 2; $i <= intval($numero / 2); $i ++)
            if ($numero % $i == 0) {
                return false;
            }
        return true;
    }
}
```

Aunque no lo hicimos en esta guía, es buena práctica comentar el código de cada método. Lo que hicimos fue organizar un poco el código, pues el nuevo método nos será útil para hacer el subservicio. Para definir un subservicio se crean **métodos públicos que comiencen con un guión bajo** (_) y serán llamados si coinciden en nombre con la **segunda palabra del asunto** del correo.

Ahora, añade el siguiente método a tu servicio:

```
public function _lista (Request $request)
    {
        $max = intval($request->query);

        if ($max > 10000) {
            $response = new Response();
            $response->setResponseSubject("Numero demasiado grande para procesar");
            $response->createFromText("El numero $max es demasiado grande para
procesar la lista de primos.");
            return $response;
        }

        $primos = array();

        for ($i = 1; $i <= $max; $i ++) {
            if ($this->esPrimo($i)) $primos[] = $i;
        }

        $response = new Response();
        $response->setResponseSubject("Lista de numeros primos hasta el $max");
        $response->createFromTemplate("lista.tpl", array(
                "primos" => $primos,
                "max" => $max
        ));
        return $response;
    }
```

Observa varios puntos importantes:

1. El subservicio se llama LISTA, de manera que cuando el usuario escriba en el asunto **PRIMO LISTA 100** y envíe el correo a Apretaste, se llamará al servicio **primo** porque la *primera palabra* es "PRIMO" (no importan las mayúscula) y no se llamará al método **_main** sino al método **_lista** porque la *segunda palabra* es "LISTA".
2. La propiedad **$request->query** tendrá el valor "100", mientras que **$request->subject** tendrá el asunto completo.
3. Se comprueba que $max no sea mayor que 10000 y se retorna una respuesta dedicada a esa situación. Esta vez no utilizamos una plantilla, sino que la respuesta **se crea desde un texto** plano mediante el método **Response::createFromText().**
4. Se *reutiliza* el método privado **$this->esPrimo()** saber si un número es primo.
5. Se devuelve una respuesta usando una plantilla **lista.tpl** que se le pasa la lista de primos que es un arreglo y el número máximo analizado.

Ahora crea el archivo **lista.tpl** en la carpeta **templates** y añade el siguiente código Smarty:

```
<h1>Lista de n&uacute;meros primos</h1>
<p>Esta es la lista de n&uacute;meros primos hasta el {$max}</p>
{foreach item = numero from = $primos}
{$numero}  
{/foreach}
```

Esta plantilla devolverá la lista de primos obtenida por el servicio separados por espacios.

PASO 8: EMPAQUETA TU SERVICIO

Una vez terminado tu servicio debes empaquetarlo. Esto se hace comprimiendo el contenido de la carpeta del servicio en formato ZIP. **No comprimas la carpeta del servicio**, sólo su contenido; los archivos ***service.php*** y ***config.xml*** y la carpeta ***templates*** deben estar en la raíz de este comprimido. El nombre del archivo comprimido debe ser igual al nombre del servicio. Para el ejemplo trabajado sería **primo.zip.** Al terminar tu comprimido deberá tener esta estructura, similar a la carpeta donde programaste:

```
primo.zip
- templates
      - basic.tpl
      - lista.tpl
- config.xml
- service.php
- primo.sql
```

PASO 9: ENVIAR TU SERVICIO PARA SER REVISADO Y PUBLICADO

Cuando tengas tu servicio empaquetado, envíalo al email salvi.pascual@gmail.com para ser revisado y agregado a Apretaste. Te dejaremos saber por email cuando sea publicado.

COMO CREAR UN SERVICIO

Guía paso a paso para programadores

PREGUNTAS FRECUENTES

A continuación una lista de preguntas frecuentes a la hora de desarrollar un servicio para Apretase.

¿CÓMO CREAR UN ARCHIVO TEMPORAL?

1. Obtén la ruta al directorio temporal de Apretaste:

```
$di = \Phalcon\DI\FactoryDefault::getDefault();
$wwwroot = $di->get('path')['root'];
$tempdir = "$wwwroot/temp";
```

2. Define un nombre para el archivo, puedes usar uno aleatorio:

```
$filename = $this->utils->generateRandomHash()."tmp";
```

3. Escribe tu archivo en ese directorio:

```
file_put_contents("$tempdir/$filename", "Este es mi archivo temporal");
```

¿CÓMO PUEDO ENVIAR UNA IMAGEN A UN USUARIO?

1. Obtén el contenido de la imagen en una variable, por ejemplo, desde una URL

```
$imgContent = file_get_contents("http://misitio.com/imagen.jpg");
```

2. Crea un archivo temporal con ese contenido en la ruta $p.

```
file_put_contents("$tempdir/$p", $imgContent);
```

3. Optimiza la imagen para mandarla por e-mail:

```
$this->utils->optimizeImage($p);
```

4. Crea la respuesta desde una plantilla como sigue:

```
$response->createFromTemplate('basic.tpl', array('image' => $p), array($p));
```

5. Edita tu plantilla e incrusta la imagen con el tag especializado IMG, observa que 'images' es una variable de plantilla definida en el paso anterior:

```
{img width="100%" src="{$image}" alt="Mi imagen"}
```

¿CÓMO PUEDO ADJUNTAR UN ARCHIVO EN EL EMAIL DE RESPUESTA?

1. El archivo debe estar en la carpeta temporal o en la carpeta del servicio, obtén su ruta en $p

2. Crea la respuesta desde una plantilla como sigue:

```
$response->createFromTemplate('basic.tpl', array(), array(), array($p));
```

Ver Anexo 2

ANEXOS

ANEXO 1: GUÍA BÁSICA DE SMARTY PARA CREAR PLANTILLAS

Smarty es un motor de plantillas para PHP. Más específicamente, esta herramienta facilita la manera de separar la aplicación lógica y el contenido en la presentación. En un servicio de Apretaste, la lógica está en service.php y el contenido en las plantillas en la carpeta templates.

VARIABLES

Smarty tiene varios tipos diferentes de variables. El tipo de variable depende de cual símbolo este prefijado (incluido dentro). Para mostrar una variable, simplemente coloque ésta entre delimitadores siendo ésta la única cosa entre ellos. Ejemplos:

```
{$Name}

{$Contacts[row].Phone}

<body bgcolor="{#bgcolor#}">
```

Puedes también referenciar matrices asociativas en variables que son definidas desde PHP especificando la clave después del simbolo '.'(punto).

```
$response->createFromTemplate('contacts.tpl', array(
    'Contacts' => array(
          'fax' => '555-222-9876',
          'email' => 'zaphod@slartibartfast.com',
          'phone' => array('home' => '555-444-3333',
                           'cell' => '555-111-1234')));
```

Este es el contenido de contacts.tpl:

```
{$Contacts.fax}
{$Contacts.email}
{$Contacts.phone.home}
{$Contacts.phone.cell}
```

La salida será:

```
555-222-9876
zaphod@slartibartfast.com
555-444-3333
555-111-1234
```

MODIFICADORES DE VARIABLES

Los modificadores de variables pueden ser aplicados a variables, funciones habituales o cadenas. Para aplicar un modificador, especifique el valor seguido por |(pipe) y el nombre del modificador. Un modificador necesita parámetros adicionales que afectan en su funcionamiento. Estos parámetros siguen al nombre del modificador y son separados por ":" (dos puntos).

Ejemplo de modificador:

```
{$title|upper}
{$topic|truncate:40:"..."}
{"now"|date_format:"%Y/%m/%d"}
{mailto|upper address="me@domain.dom"}
```

Ahora mostramos algunos modificadores útiles.

DEFAULT

Este es usado para definir un valor por defecto para una variable. Si esta variable estuviera vacía o no estuviera definida, el valor por defecto es mostrado. El valor por defecto es usado como argumento.

```
{$articleTitle|default:"no title"}
{$myTitle|default:"no title"}
```

STRING_FORMAT

Esta es una manera de formatear cadenas, como números decimales y otros. Use la sintaxis de sprintf para formatearlo.

```
{$number}
{$number|string_format:"%.2f"}
{$number|string_format:"%d"}
```

```
23.5787446
23.58
24
```

NL2BR

Todos los saltos de línea serán convertidos a etiquetas
 como datos de la variable. Esto equivale a la función nl2br() de PHP.

```
{$articleTitle|nl2br}
```

FUNCIONES INTEGRADAS

Smarty cuenta con varias funciones integradas. Acá mostramos las más utilizadas:

FOREACH,FOREACHELSE

Atributo	Tipo	Requerido	Default	Descripción
from	array	Si	*n/a*	El nombre de la matriz a la que usted estará pegando los elementos
item	string	Si	*n/a*	El nombre de la variable que es el elemento actual
key	string	No	*n/a*	El nombre de la variable que es la llave actual
name	string	No	*n/a*	El nombre del ciclo foreach para acessar a las propiedades del foreach

Los ciclos (loop) *foreach* son una alternativa para loop *section*. *foreach* es usado para pegar cada elemento de una matriz asociativa simple. La sintaxis para *foreach* es mucho más simple que *section*, pero tiene una desventaja de que solo puede ser usada en una única matriz. La etiqueta *foreach* debe tener su par */foreach*. Los parámetros requeridos son *from* e *item*. El nombre del ciclo (loop) foreach puede ser cualquier cosa que usted quiera, hecho de letras, números y subrayados. Los ciclos (loop) *foreach* pueden ser anidados, y el nombre de los ciclos (loop) anidados debe ser diferente uno de otro. La variable *from* (normalmente una matriz de valores) determina el número de veces del ciclo (loop) *foreach*. *foreachelse* y ejecutando cuando no hubieren más valores en la variable *from*.

```
{* este ejemplo muestra todos los valores de la matriz $custid *}
{foreach from = $custid item=curr_id}
        id: {$curr_id}<br>
{/foreach}

SALIDA:

id: 1000<br>
id: 1001<br>
id: 1002<br>
```

```
{* La llave contiene la llave para cada valor del ciclo(loop):

contacts => array(
      array("phone" => "1", "fax" => "2", "cell" => "3"),
      array("phone" => "555-4444", "fax" => "555-3333", "cell" => "760-1234")));
*}

{foreach name=outer item=contact from=$contacts}
  {foreach key=key item=item from=$contact}
    {$key}: {$item}<br>
  {/foreach}
{/foreach}

SALIDA:
```

```
phone: 1<br>
fax: 2<br>
cell: 3<br>
phone: 555-4444<br>
fax: 555-3333<br>
cell: 760-1234<br>
```

El ciclo (Loop) foreach también tiene sus propias variables para manipular las propiedades del foreach. Estas son indicadas así: {$smarty.foreach.foreachname.varname} con foreachname siendo el nombre especificado del atributo *name* del foreach.

ITERATION

iteration es usado para mostrar la interación actual del ciclo(loop).

iteration siempre comienza en 1 incrementado uno a uno en cada interación.

FIRST

first Toma el valor true si la interación actual del foreach es la primera.

LAST

last Toma el valor de true si la interación actual del foreach es la ultima.

SHOW

show Es usado como parámetro para el foreach. *show* Es un valor booleano, true o false. Si es false, el foreach no será mostrado. Si tuviera un foreachelse presente, este será alternativamente mostrado.

TOTAL

total Es usado para mostrar el número de interaciones del foreach. Este puede ser usado dentro o después de el.

IF,ELSEIF,ELSE

Los comandos if del Smarty tiene mucho de la flexibilidad del comando if de php, con algunas adiciones para la herramienta de template. Todo *{if}* debe tener su *{/if}*. *{else}* y *{elseif}* también son permitidos. Toda las condicionales de PHP son reconcidas, tal como ||, *or*, &&, *and*, etc.

Ejemplos del uso de IF

```
{if $name eq "Fred"}
        Welcome Sir.
{elseif $name eq "Wilma"}
        Welcome Ma'am.
{else}
        Welcome, whatever you are.
{/if}

{* Un ejemplo con "or" logico *}
{if $name eq "Fred" or $name eq "Wilma"}
        ...
{/if}

{* El mismo que arriba *}
{if $name == "Fred" || $name == "Wilma"}
        ...
{/if}

{* La siguiente sintaxis no funcionara, el calificador de condición deben estar separados
entre ellos por espacios *}
{if $name=="Fred" || $name=="Wilma"}
        ...
{/if}


{* los parentesis son permitidos *}
{if ( $amount < 0 or $amount > 1000 ) and $volume >= #minVolAmt#}
        ...
{/if}

{* Usted también puede colocar funciones de PHP *}
{if count($var) gt 0}
        ...
{/if}

{if $var is even}
        ...
{/if}
{if $var is odd}
        ...
{/if}
{if $var is not odd}
        ...
{/if}

{if $var is div by 4}
        ...
{/if}

{* Checa si la variable var es igual, agrupandola por dos. i.e.,0=even, 1=even, 2=odd, 3=odd,
4=even, 5=even, etc. *}
{if $var is even by 2}
        ...
{/if}
```

ANEXO 2: LAS CLASES PHP DE APRETASTE QUE DEBES CONOCER

REQUEST

El único uso del objeto Request es enviar a la función _main() y a los subservicios detalles de la petición del usuario. Estos pueden ser necesarios para el trabajo interno que el servicio realiza. En el código mostrado arriba, se utiliza $request->query, lo cual devuelve la parte del asunto escrita por el usuario después del nombre del servicio y los subservicios. Por ejemplo: en el asunto: "LETRA before I forget", el *query* sería el texto "before I forget".

A continuación la definición de la clase Request:

```
class Request {
	public $email; // email del usuario
	public $name; // nombre del usuario, si existe
	public $subject; // asunto completo, EJ: LETRA before I forget
	public $body; // cuerpo del email, aveces es necesario
	public $attachments; // arreglo de archivos adjuntos al email
	public $service; // nombre del servicio, P.E.: LETRA
	public $subservice; // nombre del subservicio si existe, EJ: AYUDA
	public $query; // texto detrás del servicio y subservicio EJ: before I forget
}
```

RESPONSE

Se encarga de enviar la respuesta al núcleo de Apretaste para ser enviadas de nuevo al usuario. No es necesario llamar literalmente a la función que envía emails, de hecho no se aconseja su uso. Cada servicio debe retornar un objeto de tipo Response o un arreglo de objetos Response. Cada objeto Response que se retorne será convertido en un email para el usuario.

A continuación la definición de la clase Response, seguida por la explicación de sus funciones.

```
class Response {
	public function setResponseSubject($subject);
	public function setResponseEmail($email);
	public function createFromText($text);
	public function createFromTemplate($template, $content, $images, $attachments)
}
```

setResponseSubject($subject): Personaliza el asunto del email de respuesta que recibirá el usuario. Aunque es aconsejable escribir asuntos personalizados, no es obligatorio. Si no se llama a esta función el asunto del email de respuesta será creado por defecto. El parámetro $subject debe ser un texto.

setResponseEmail($email): Por defecto el email de retorno de una llamada a Apretaste va dirigido al usuario que invocó el servicio. Use esta función para cambiar la dirección de retorno. Aunque en la mayoría de los casos no es necesario su uso, esta función permite enviar alertas al programador, o en servicios donde el usuario interactúe con otros usuarios, enviar confirmaciones a los mismos. Un servicio de Apretaste puede retornar un arreglo de objetos Response, y cada objeto puede estar destinado a un usuario diferente. El parámetro $email debe ser un email válido.

createFromText($text): Genera una respuesta al usuario a partir de un texto. Esta función autogenera una plantilla con el texto pasado mediante el parámetro $text, la cual será utilizada en el email de respuesta al usuario. Esta función permite facilitar la creación de mensajes de error y advertencias. Observe su uso en la función _ayuda() del ejemplo anterior. Aunque puede insertarse código HTML mediante el parámetro $text, es recomendado crear una plantilla para cada respuesta que contenga más que un texto.

createFromTemplate($template, $content, $images, $attachments): Genera una respuesta al usuario a partir de una plantilla creada anteriormente y un arreglo de valores. Use esta función para especificar que plantilla usar al responder al usuario, al igual que el texto que será insertado en la plantilla dinámicamente. También puede especificar imágenes a usar en la plantilla o archivos adjuntos. A continuación se detallan sus parámetros:

- $template: Nombre de la plantilla a usar. Debe coincidir con el nombre de un archivo dentro de la carpeta "templates". Este parámetro es requerido.
- $content: Arreglo asociativo de nombre de variables y su contenido, el cual será insertado en la plantilla en el lugar donde se llame a las variables en la plantilla. Este parámetro es requerido.
- $images: Arreglo de las imágenes que serán utilizadas en la plantilla. Las imágenes en el email no trabajan igual que en la web, si no se especifica el camino (*path)* completo para cada imagen como un elemento de este arreglo, ninguna imagen llamada desde la plantilla será mostrada al usuario. Las imágenes del arreglo deben estar incluidas en la carpeta del servicio o en la carpeta temporal de Apretaste, no se pueden mostrar imágenes directamente desde la web. Este parámetro no es requerido.
- $attachments: Arreglo de archivos que serán enviados como adjuntos del email de respuesta. Debe siempre escribirse el camino completo a cada archivo, y estos deben estar incluidos en la carpeta del servicio o en la carpeta temporal de Apretaste. Este parámetro no es requerido.

SERVICE

Todo servicio creado extiende de la clase Service, la cual provee la información básica del servicio así como recursos para facilitar el trabajo del programador. La información básica del servicio se guardará en la clase Service, cuya información será guardada una vez se haga el despliegue del servicio en el servidor. El programador puede hacer referencia a estos parámetros para referenciar al propio servicio como tal. Hay dos atributos importantes:

- ➔ **$pathToService:** Contiene el camino al servicio para facilitar la inclusión de imágenes y librerías externas que se encuentren en la carpeta del servicio.
- ➔ **$utils:** Contiene un objeto Utils, el cual define funciones útiles para crear un servicio.

A continuación la definición de la clase Service.

```
class Service {
	public $serviceName; // nombre del servicio en config.xml, EJ: PRIMO
	public $serviceDescription; // descripción del servicio en config.xml
	public $creatorEmail; // email del creador del servicio en config.xml
	public $serviceCategory; // categoría del servicio en config.xml
	public $serviceUsage; // texto explicando cómo usar el servicio en config.xml
	public $insertion Date; // fecha en que el servicio fue creado
	public $pathToService; // camino/path al servicio, útil para incluir imágenes y recursos
	public $utils; // objeto de la clase Utils, mencionada más adelante
}
```

UTILS

El objeto Utils puede ser accedido desde cualquier clase Service mediante $this->utils, y contiene funciones que facilitan el trabajo del programador. A continuación la definición de la clase Utils.

```
class Utils {
	public function serviceExist($serviceName); // revisa si un servicio existe
	public function personExist($personEmail); // revisa si un usuario existe
	public function getPerson($email); // devuelve la información de un usuario en forma de objeto
	public function getCurrentRaffle(); // devuelve la rifa en curso en forma de objeto
	public function generateRandomHash(); // crea un hash aleatorio
	public function optimizeImage($imagePath); // optimiza/redimensiona una imagen
	public function fullNameToNamePieces($name); // pica un nombre y devuelve un arreglo
                                                 // con cada una de sus partes (nombre, apellido…
}
```

ANEXO 3: TAGS ESPECIALIZADOS QUE PUEDES USAR EN TUS PLANTILLAS

Observa el siguiente código fuente de una plantilla en **templates/basic.tpl**:

```
<center>
        {img src="logo.jpg" alt="Lyrics Logo" width="120" height="75"}
        <h1>{$cancion}</h1>
        {space10}
        {$letra}
</center>
```

Es una plantilla pequeña y el código es relativamente sencillo de entender. Las llamadas a **{$cancion}** y a **{$letra}** serán reemplazados por el texto de ambas variables pasadas a la plantilla. Cada imagen pasada a la plantilla debe posicionarse con el tag ***{img src="image.jpg"},*** y el tag **{space10}** crea un espacio de 10px entre el título de la canción y la letra. El resto es código HTML.

A continuación la lista completa de tags especializados que pueden ser usados en la plantilla:

{apretaste_email}	Devuelve uno de los emails Apretaste que se usan para contactar el servicio.
{hr}	Crea una linea divisoria horizontal. El uso del tag no es aconsejado por cuestiones de compatibilidad y diseño.
{separator}	Crea una línea vertical para separar links. EJ: link1 | link2 | link3
{space5}	Crea un espacio horizontal de 5px. El uso del tag no es aconsejado por cuestiones de compatibilidad.
{space10}	Crea un espacio horizontal de 10px. El uso del tag no es aconsejado por cuestiones de compatibilidad.
{space15}	Crea un espacio horizontal de 15px. El uso del tag no es aconsejado por cuestiones de compatibilidad.
{space30}	Crea un espacio horizontal de 30px. El uso del tag no es aconsejado por cuestiones de compatibilidad.
{img src="..." width="..." height="..."}	Crea una imagen. El atributo *src* debe contener el nombre de la imagen. Los atributos *width* y *height* son opcionales.
{link href="..." caption="..."}	Crea un link a otro servicio. El atributo *href* debe tener el asunto del email, EJ: "LETRA before I forgot". El atributo *caption* contiene el texto del link.
{button href="..." caption="..."}	Crea un botón que llama otro servicio. El atributo *href* debe tener el asunto del email, EJ: "LETRA before I forgot". El atributo *caption* contiene el texto del botón.