Manual de MVC: (1) FrontController

ARQUITECTURA

Desde esta segunda edición de Hackers & Developers Magazine, damos inicio al Manual de MVC en Python y PHP. Cada mes, una nueva entrega del manual, abarcando en detalle, el desarrollo de aplicaciones Web modulares con el patrón arquitectónico modelo-vista-controlador. En esta primera entrega, haremos una introducción a MVC y hablaremos sobre el FrontController: el patrón de diseño, que nos permitirá utilizar una arquitectura MVC en sistemas modulares.

Escrito por: **Eugenia Bahit** (Arquitecta GLAMP & Agile Coach)

Eugenia es **Arquitecta de Software**, **docente** instructora de tecnologías **GLAMP** (GNU/Linux, Apache, MySQL, Python y PHP) y **Agile coach** (UTN) especializada en Scrum y eXtreme Programming. Miembro de la **Free Software Foundation** e integrante del equipo de **Debian Hackers**.

Webs:
www.cursosdeprogramacionadistancia.com
www.eugeniabahit.com

Redes sociales:
Twitter / Identi.ca: **@eugeniabahit**

FrontController es el **patrón de diseño** requerido para trabajar con **MVC**, en **arquitecturas modulares**, ya que el mismo, permite manejar desde un único archivo, las peticiones del usuario, analizarlas e invocar al controlador responsable de proveer la información solicitada al usuario. Pero antes de ir de lleno al `FrontController`, haremos una breve introducción sobre el Patrón Arquitectónico (mal llamado "patrón de de diseño"), MVC.

Entendiendo el Patrón Arquitectónico modelo-vista-controlador

El patrón MVC debe ser entendido desde dos ópticas diferentes: la del desarrollador y la

del usuario. Desde el punto de vista del desarrollador, MVC nos permite mantener individualizadas, las "responsabilidades" dentro de un sistema, permitiéndonos diferenciar y aislar el diseño del sistema (objetos que componen los "modelos"), de la interfaz gráfica del usuario (GUI) y su correspondiente lógica de negocios (vistas), utilizando como "conector intermediario" un objeto controlador.

El orden modelo-vista-controlador, es el que el programador debe seguir en el proceso de desarrollo del sistema. Sin embargo, desde el punto de vista del usuario, todo comienza en el controlador. Veamos cómo:

- A nivel funcional, en MVC todo se inicia con una solicitud del usuario (petición);
- Dicha solicitud (algo que el usuario quiere hacer con respecto al sistema), es representada mediante la estructura de la URI;
- En una arquitectura modular, la estructura de la URI, guardará la forma: (dominio)/modulo/modelo/recurso[/argumentos] siendo modulo, modelo y recurso datos obligatorios y argumentos, opcionales. De esta forma, si un usuario quisiera agregar una nueva nota de crédito y el modelo (objeto) "comprobante", perteneciese al módulo de contabilidad, dicha petición sería realiza a través de la siguiente URI: http://mymvcapp.net/contabilidad/comprobante/agregar
- Cuando la solicitud del usuario es enviada, ésta es recibida por el controlador del modelo, quien se comunica con éste solicitándole la información necesaria;
- Una vez que el modelo retorna la información al controlador, éste le entrega dicha información a la vista del modelo, quien será la encargada de procesar la información recibida, colocarla en la GUI y mostrársela al usuario.

Veamos un ejemplo:

```
# MODELO: /myapp/modulo/models/vidrio.php
class Vidrio {

    function __construct() {
        $this->vidrio_id = 0;
        $this->color = '';
    }

    function save() {
        # Guarda un nuevo objeto u objeto existente
    }

    function get() {
        # Recupera un objeto
    }

    function destroy() {
        # Destruye un objeto
    }

}
```

```
# MODELO: /myapp/modulo/models/vidrio.py
class Vidrio(object):

    def __init__(self):
        self.vidrio_id = 0
        self.color = ''

    def save(self):
        """Guarda un nuevo objeto u objeto existente"""
        pass

    def get(self):
        """Recupera un objeto"""
        pass

    def destroy(self):
        """Destruye un objeto"""
        pass


# GUI (para PHP): /myapp/static/html/ver_vidrio.html
<h1>Vidrio {vidrio_id}</h1>
<p>Vidrio de color {color}.</p>


# VISTA: /myapp/modulo/views/vidrio.php
class VidrioView {

    function ver($objeto=NULL) {
        settype($objeto, 'array');
        $comodines = array_keys($objeto);
        foreach($comodines as &$comodin) {
            $comodin = "\{$comodin\}";
        }
        $valores = array_values($objeto);
        $template = file_get_contents("/myapp/static/html/ver_vidrio.html");
        print str_replace($comodines, $valores, $template);
    }

}



# GUI (para Python): /myapp/static/html/ver_vidrio.html
<h1>Vidrio $vidrio_id</h1>
<p>Vidrio de color $color.</p>

# VISTA: /myapp/modulo/views/vidrio.py
class VidrioView(object):

    def ver(self, objeto):
        with open("/myapp/static/html/ver_vidrio.html", "r") as archivo:
            template = archivo.read()
        return Template(template).safe_substitute(vars(objeto))
```

```
# CONTROLADOR: /myapp/modulo/controllers/vidrio.php
class VidrioController {

    function ver($id=0) {
        $vidrio = new Vidrio();
        $vidrio->vidrio_id = $id;
        $vidrio->get();

        $view = new VidrioView();
        $view->ver($vidrio);
    }

}
```

```
# CONTROLADOR: /myapp/modulo/controllers/vidrio.py
class VidrioController(object):

    def ver(self, id=0):
        vidrio = Vidrio()
        vidrio.vidrio_id = id
        vidrio.get()

        view = VidrioView()
        self.output = view.ver(vidrio)
```

Si el usuario quisiera ver el vidrio con id 15, su solicitud sería enviada a través de: http://mymvcapp.net/modulo/vidrio/ver/15 y la misma, sería tramitada por VidrioController, pero **¿cómo llegará la solicitud a VidrioController?** Para responder a esta pregunta, **tendremos que hablar de FrontController**.

Recibiendo las solicitudes del usuario con FrontController desde Python

En Python (bajo el supuesto que trabajamos con WSGI bajo Apache[7]) nuestra `application` será quien haga las veces de FrontController. En primer lugar, deberá analizar la URI a fin de obtener el módulo, modelo, recurso y los argumentos opcionales que le permitirán conocer a qué controlador debe invocar:

```
# Divido la URI utilizando como separador la barra diagonal
peticiones = environ['REQUEST_URI'].split('/')

# Elimino el primer elemento puesto que estará vacío
peticiones.pop(0)

# Cuento las peticiones, para saber si hay o no argumentos
```

7 Si no sabes como codear una Web en Python "crudo" (sin Frameworks) corriendo bajo Apache, te recomiendo leer un artículo que publiqué en Debian Hackers **"Una Web en Python sobre Apache en 3 pasos"** ingresando en http://www.debianhackers.net/una-web-en-python-sobre-apache-sin-frameworks-y-en-solo-3-pasos

```
cantidad = len(peticiones)

# Obtengo el módulo, modelo, recurso (y argumentos, si existen)
if cantidad == 3:
    modulo, modelo, recurso = peticiones
elif cantidad == 4:
    modulo, modelo, recurso, arg = peticiones
```

A continuación, deberá -utilizando los datos obtenidos-, importar el módulo del controlador, instanciarlo y entregarle la información necesaria para que éste actúe:

```
# Obtengo el nombre del controlador
controller_name = '%sController' % modelo

# Para poder importar el controlador, debo agregar el path de la aplicación
from sys import path
path.append(environ['SCRIPT_FILENAME'].replace('frontcontroller.py', ''))

# Importo el módulo del controlador
exec 'from %s.controllers.%s import %s' % (modulo, modelo, controller_name)

# Instancio al controlador y le envío el recurso y argumentos
controller = locals()[controller_name](recurso, arg)

# Capturo la salida
output = controller.output
```

Finalmente, se llama a start_response() y se retorna la salida, normalmente:

```
def application(environ, start_response):
    peticiones = environ['REQUEST_URI'].split('/')
    peticiones.pop(0)
    cantidad = len(peticiones)

    if cantidad == 3:
        modulo, modelo, recurso = peticiones
    elif cantidad == 4:
        modulo, modelo, recurso, arg = peticiones

    controller_name = '%sController' % modelo.capitalize()

    from sys import path
    path.append(environ['SCRIPT_FILENAME'].replace('frontcontroller.py', ''))

    exec 'from %s.controllers.%s import %s' % (modulo, modelo,
        controller_name)

    controller = locals()[controller_name](recurso, arg)
    output = controller.output

    start_response('200 OK', [('Content-Type', 'text/html; charset=utf-8')])
    return output
```

El FrontController, nos obligará a que nuestros controladores, estén preparados para

recibir dos parámetros: el recurso (para hacer la llamada dinámica al método actuante) y el argumento (que pueda ser requerido por algunos métodos):

```
class VidrioController(object):

    def __init__(self, recurso='', arg=0):
        getattr(self, recurso)(int(arg))

    def ver(self, id=0):
        vidrio = Vidrio()
        vidrio.vidrio_id = id
        vidrio.get()

        view = VidrioView()
        self.output = view.ver(vidrio)
```

Recibiendo las solicitudes del usuario con FrontController desde PHP

En PHP, el trabajo que debemos dejar preparado para poder actuar, requiere un mayor esfuerzo. En primer lugar, debemos preparar a nuestro servidor, para trabajar con las denominadas "Friendly URL" (URL amigables). Esto requiere:

Habilitar el módulo `rewrite` de Apache (debes convertirte en súper usuario):

```
a2enmod rewrite
```

Modificar la directiva AllowOverride de nuestro VirtualHost:

```
AllowOverride All
```

(Si utilizas el `VirtualHost` por defecto, localizarás el archivo en `/etc/apache2/sites-available/default`. Ten la precaución de modificarlo como súper usuario. La directiva `AllowOverride` se encuentra dentro del tag `Directory`) ¡No olvides **reiniciar Apache** para que los cambios surjan efecto! `service apache2 restart`

Y finalmente, crear un archivo `.htaccess` en el directorio raíz de la aplicación (estará en el mismo directorio que frontcontroller.php)

```
# Encender el motor de reescritura de las URL
RewriteEngine On
# Crear una regla que redirija todas las peticiones (excepto las realizadas
# al directorio static) hacia el frontcontroller.php
RewriteRule !(^static) frontcontroller.php
```

Una vez "preparado el terreno", estaremos en condiciones de crear nuestro FrontController.

Al igual que en Python, el primer paso será analizar la URI a fin de obtener el módulo, el modelo, el recurso y los argumentos que opcionalmente, podría estar enviando el

usuario:

```
# Divido la URI utilizando como separador la barra diagonal
$peticiones = explode('/', $_SERVER['REQUEST_URI']);

# Cuento las peticiones, para saber si hay o no argumentos
$cantidad = count($peticiones);

# Obtengo el módulo, modelo, recurso (y argumentos, si existen)
if($cantidad == 3) {
    list($modulo, $modelo, $recurso) = $peticiones;
} elseif($cantidad == 4) {
    list($modulo, $modelo, $recurso, $arg) = $peticiones;
}
```

Luego, con los datos obtenidos, está en condiciones de importar el archivo del controlador , instanciarlo y entregarle la información necesaria para que éste actúe:

```
# Obtengo el nombre del controlador
$controller_name = ucwords($modelo) . "Controller";

# Para poder importar el controlador, debo agregar el path de la aplicación
ini_set('include_path', str_replace('frontcontroller.php', '',
    $_SERVER['SCRIPT_FILENAME']));

# Importo el módulo del controlador
require_once("$modulo/controllers/$modelo.php");

# Instancio al controlador y le envío el recurso y argumentos
$controller = new $controller_name($recurso, $arg);
```

Finalmente, el FrontController deberá auto-ejecutarse:

```
class FrontController {

    public static function handler() {
        $peticiones = explode('/', $_SERVER['REQUEST_URI']);
        $cantidad = count($peticiones);
        if($cantidad == 3) {
            list($modulo, $modelo, $recurso) = $peticiones;
        } elseif($cantidad == 4) {
            list($modulo, $modelo, $recurso, $arg) = $peticiones;
        }
        $controller_name = ucwords($modelo) . "Controller";
        ini_set('include_path', str_replace(
            'frontcontroller.php', '', $_SERVER['SCRIPT_FILENAME']));
        require_once("$modulo/controllers/$modelo.php");
        $controller = new $controller_name($recurso, $arg);
    }

}

FrontController::handler();
```

Por favor, notar que la directiva ini_set, debería estar en un archivo de configuración (un settings.php) y no, dentro del FrontController.

Nuestro FrontController, ahora nos estará obligando a preparar los controladores de nuestros módulos, para recibir dos parámetros: el recurso (para hacer la llamada dinámica al método actuante) y el argumento (que pueda ser requerido por algunos métodos):

```
class VidrioController {

    function __construct($recurso='', $arg=0) {
        call_user_func(array($this, $recurso), $arg);
    }

    function ver($id=0) {
        $vidrio = new Vidrio();
        $vidrio->vidrio_id = $id;
        $vidrio->get();

        $view = new VidrioView();
        $view->ver($vidrio);
    }

}
```

En la siguiente entrega de MVC en Python y PHP, nos ocuparemos especialmente de las vistas.