Manual de MVC: (3) Controladores

En la primera entrega del manual, estuvimos viendo como crear el controlador principal de la aplicación. Aprendimos cómo éste, se comunicaba de forma directa con el controlador del modelo correspondiente, actuando de "ruteador" de los recursos. En esta entrega final nos concentraremos en las responsabilidades de los controladores y en los casos en las que éstas intervienen.

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:
Cursos de programación a Distancia: www.cursosdeprogramacionadistancia.com
Web personal: www.eugeniabahit.com

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

En la primera entrega del Manual de MVC publicada en Hackers & Developers Magazine Nro. 1, hablamos de `FrontController` y aprendimos que éste actúa de *"ruteador"* de cada recurso, instanciando directamente al controlador del modelo.

Como recordarán, cada controlador debería estar preparado -desde su método constructor- para realizar una llamada de retorno al recurso correspondiente. Dicha acción, se encontraba en relación directa a la instancia generada por `FrontController`.

La idea de esta última entrega en concentrarnos en los recursos que los controladores de los modelos suelen manejar, sus responsabilidades y los casos en las que las mismas intervienen.

Características de los controladores

Todo controlador llevará el nombre del modelo, seguido de la palabra `Controller`:

```
# En Python
```

```
class VidrioController(object):
    pass


# En PHP
class VidrioController { }
```

Todo controlador debe preparar su método constructor para recibir al menos 2 parámetros en la instancia que realice el `FrontController`:

```
# En PHP
class VidrioController {

    function __construct($recurso='', $argumento) {
    }

}
```

En el caso de Python, además, será muy útil que el `FrontController` le entregue un tercer parámetro: el diccionario `environ` para que el controlador pueda recuperar los datos enviados desde los formularios (vía `POST`) a través de la clave `wsgi.input` (si existe). En este caso, el controlador, deberá disponer de dicho valor en una propiedad:

> Por favor, notar que esto es solo necesario si se está trabajando en el supuesto de WSGI sobre Apache.

```
# En Python
class VidrioController(object):

    def __init__(self, recurso='', argumento=0, environ={}):
        self.pd = env['wsgi.input'].read() if 'wsgi.input' in environ else ''
```

Los métodos constructores deberán realizar una llamada de retorno a los recursos correspondientes, pasándoles el argumento como parámetro:

```
# En Python
class VidrioController(object):

    def __init__(self, recurso='', argumento=0, environ={}):
        self.pd = env['wsgi.input'].read() if 'wsgi.input' in environ else ''
        getattr(self, recurso)(int(argumento))


# En PHP
class VidrioController {

    function __construct($recurso='', $argumento) {
        call_user_func(array($this, $recurso), $argumento);
```

```
    }

}
```

Los métodos de los controladores, representarán los recursos disponibles:

```
# En Python
class VidrioController(object):

    def __init__(self, recurso='', argumento=0, environ={}):
        self.pd = env['wsgi.input'].read() if 'wsgi.input' in environ else ''
        getattr(self, recurso)(int(argumento))

    def agregar(self, *arg):
        pass

    def guardar(self, *arg):
        pass

    def ver(self, id=0):
        pass


# En PHP
class VidrioController {

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

    function agregar() { }

    function guardar() { }

    function ver($id=0) { }

}
```

Básicamente existen 5 tipos de recursos estándar:

```
agregar()       Muestra el formulario para crear un nuevo objeto
editar(id)      Muestra el formulario para modificar un objeto existente
guardar()       Guarda los cambios realizados a través de editar() y agregar()
eliminar(id)    Elimina un objeto existente
listar()        (opcional) Muestra la colección completa de objetos basados en el
                mismo modelo
ver(id)         (opcional) Permite ver los datos de un objeto
```

Los recursos `editar()`, `eliminar()` y `ver()` suelen recibir la ID del objeto como parámetro (argumento recibido desde `FrontController`).

Responsabilidades de los controladores

1. Llamar al modelo (crear instancia). Generalmente en los recursos `editar`, `guardar`, `eliminar` y `listar` (en este último caso, llamará al colector).
2. Sanear y asegurar los datos recibidos desde el usuario ANTES de modificar el modelo: el saneamiento y aseguración de datos recibidos por el usuario, está en manos de los controladores y NO de los modelos.
3. Modificar sus propiedades cuando sea necesario. Generalmente en los recursos donde realiza llamadas al modelo.
4. Hacer las llamadas (al) a los métodos correspondientes del modelo. De igual forma que los anteriores, en los casos en los que se llama al modelo y se lo modifica.
5. Entregar los datos obtenidos a la vista:
    - En el recurso `agregar`, suele llamar directamente a la vista sin pasar datos;
    - En los recursos `editar`, `ver` y `listar`, entrega los datos del objeto a la vista;
    - En los recursos `eliminar` y `guardar`, suele pasar de largo a la vista y en cambio, redirige al usuario hacia otro recurso.

¿Cómo se ven los métodos de un controlador?

Un ejemplo con código comentado:

```
# En Python
class VidrioController(object):

    def __init__(self, recurso='', argumento=0, environ={}):
        self.pd = env['wsgi.input'].read() if 'wsgi.input' in environ else ''
        getattr(self, recurso)(int(argumento))

    def agregar(self, *arg):
        # Llama directamente a la vista
        vista = VidrioView()
        self.output = vista.mostrar_form_alta()

    def editar(self, id=0):
        # Instancia al modelo
        modelo = Vidrio()
        # Modifica las propiedades necesarias
        modelo.vidrio_id = int(id)
        # Llama al método correspondiente (necesita recuperar el objeto)
        modelo.get()
        # Le entrega la información a la vista
        vista = VidrioView()
        self.output = vista.mostrar_form_edicion(modelo)

    def guardar(self, *arg):
```

```
        # Obtiene los datos enviados del el formulario
        # Esto generalmente debe hacerse desde un Helper
        post = self.pd.split('&')
        _POST = {}
        for par in post:
            field, value = par.split('=')
            _POST[field] = value

        # Instancia al modelo
        modelo = Vidrio()
        # Modifica las propiedades necesarias
        modelo.vidrio_id = int(_POST['vidrio_id'] if 'vidrio_id' in _POST else 0
        modelo.color = _POST['color']
        # Llama al método correspondiente (necesita guardar el objeto)
        modelo.save()
        # Pasa de largo a la vista y en cambio, recurre a otro recurso
        self.listar()

    def eliminar(self, id=0):
        # Instancia al modelo
        modelo = Vidrio()
        # Modifica las propiedades necesarias
        modelo.vidrio_id = int(id)
        # Llama al método correspondiente (necesita destruir al objeto)
        modelo.destroy()
        # Pasa de largo a la vista y en cambio, recurre a otro recurso
        self.listar()

    def listar(self, *arg):
        # Recurre al colector para traer toda la colección de objetos Vidrio
        coleccion = VidrioCollector().get()
        # Le entrega la información a la vista
        vista = VidrioView()
        self.output = vista.mostrar_listado(coleccion)
```

```
# En PHP
class VidrioController {

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

    function agregar() {
        # Llama directamente a la vista
        $vista = new VidrioView();
        $vista->mostrar_form_alta();
    }

    function editar($id=0) {
        # Instancia al modelo
        $modelo = new Vidrio();
        # Modifica las propiedades necesarias
        $modelo->vidrio_id = (int)$id;
        # Llama al método correspondiente (necesita recuperar el objeto)
        $modelo->get();
        # Le entrega la información a la vista
        $vista = new VidrioView();
        $vista->mostrar_form_edicion($modelo);
    }
```

```
    function guardar() {
        # Instancia al modelo
        $modelo = Vidrio()
        # Modifica las propiedades necesarias
        $id = isset($_POST['vidrio_id']) ? (int)$_POST['vidrio_id'] : 0;
        $modelo->vidrio_id = $id;
        $modelo->color = $_POST['color'];
        # Llama al método correspondiente (necesita guardar el objeto)
        $modelo->save();
        # Pasa de largo a la vista y en cambio, recurre a otro recurso
        $this->listar();
    }

    function eliminar($id=0) {
        # Instancia al modelo
        $modelo = new Vidrio();
        # Modifica las propiedades necesarias
        $modelo->vidrio_id = (int)$id;
        # Llama al método correspondiente (necesita destruir al objeto)
        $modelo->destroy();
        # Pasa de largo a la vista y en cambio, recurre a otro recurso
        $this->listar();
    }

    function listar() {
        # Recurre al colector para traer toda la colección de objetos Vidrio
        $coleccion = VidrioCollector::get();
        # Le entrega la información a la vista
        $vista = new VidrioView();
        $vista->mostrar_listado($coleccion);
    }

}
```