

1. Introducción a JF

JF es un FrameWork escrito en PHP y desarrollado con la arquitectura MVC, pensado en un desarrollo práctico y fácil para adaptarse a la aplicación.

El nacimiento de este software comenzó a partir de un programa de escritorio hecho en c# 2010, a mediados del 2012, luego la idea fue madurando y ahora en su versión PHP vía web.

Entre las características de Janos FrameWork (JF) tenemos:

- Implementa Clases para autenticación de usuarios.
- Pensado para soporte de múltiples bases de datos, usando PDO dentro del marco de desarrollo.
- El mapeo de las clases con respecto a las tablas de la base de datos es dinámica, si pensabas declarar los aburridos sets y gets a las clases de tu base de datos, déjame decirte que te olvidaras de eso, aquí en JF, esto se crea en forma automática, además que los métodos básicos para insertar, actualizar, eliminar, seleccionar registros de una tabla se hacen de una forma muy práctica.
- Soporte de THEMES.
- El uso de HELPERS también está incluido, tanto para formularios como para inputs, select, etc.

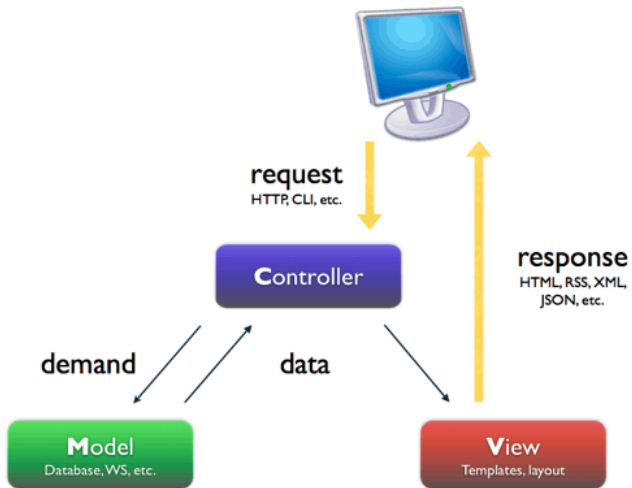
Acerca de MVC:

Según Wikipedia:

Es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. El patrón MVC se ve frecuentemente en aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página; el modelo es el Sistema de Gestión de Base de Datos y la Lógica de negocio; y el controlador es el responsable de recibir los eventos de entrada desde la vista.

Entre las capas que contiene MVC son:

- Modelo: Todo lo relacionado con la extracción de datos, ya sea a un archivo de texto, una base de datos etc.
- Vista: La parte donde se interacciona con el usuario, se especifican cosas como posición de datos, y como se desplegaran.
- Controlador: Pone orden entre los dos anteriores decide cuando se hace una llamada de datos, y cuando se despliega algo.



Este paradigma no tiene que ver con el lenguaje de programación, cada programador puede aplicarlo casi en cualquier lenguaje, aunque ya hay FrameWork que hacen esto.

Entre los FrameWork en esta arquitectura tenemos:

En PHP:

Framework
Akelos
Cake PHP
CodeIgniter
Zend Framework
Yii
Kohana
Symfony

Y por último **Janos Framework**(JF) que se suma a esta larga lista.

En JAVA:

Framework
Struts
JFS
Tapestry
Spring MVC
Grails

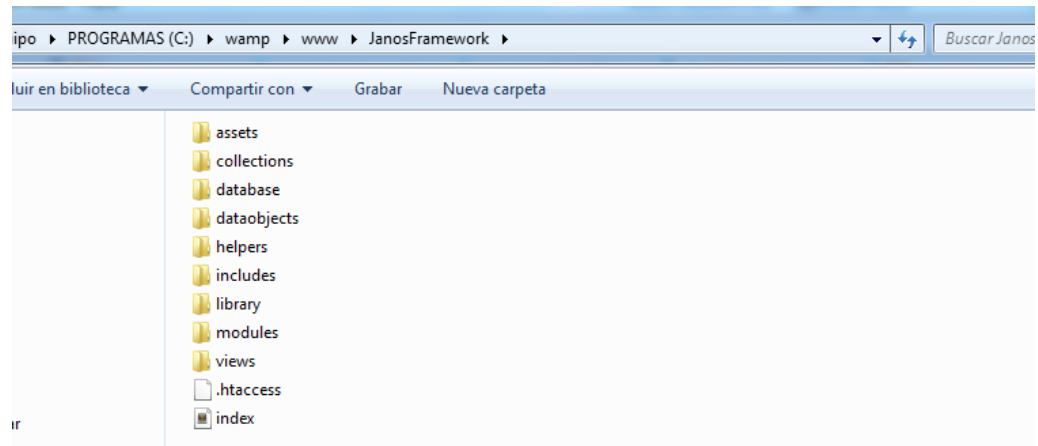
En .NET

Framework
Asp.NET MVC2
Asp.NET MVC3
Asp.NET MVC4

2. Mi Primera Aplicación con JF

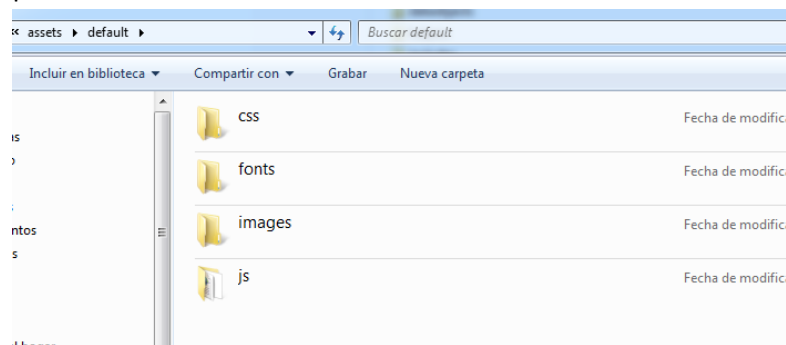
a. Conociendo el entorno de Trabajo

La estructura de JF está compuesta por las siguientes partes:



Assets:

Aquí colocamos los archivos o recursos como imágenes, JavaScript, css agrupados por temas.



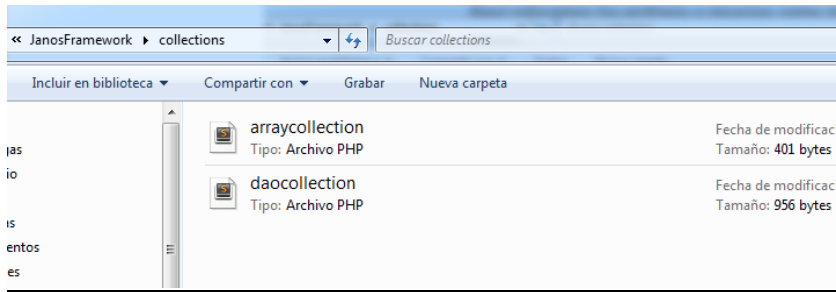
El tema por defecto es default.

DataObjects:

Colocamos aquí la lista de objetos, generalmente cada archivo representa a la tabla de la base de datos.

Collections:

Contiene las clases que representan a la lista de objetos mencionados en lo anterior. Esto representa las reglas de negocio.



Database:

Contiene la lista de clases que representan a los gestores de base de datos.

Helpers:

Contiene métodos para creación de controles como textbox, combobox, radiobutton, menús, navigation, etc en forma rápida y sencilla.

Includes:

Contiene los archivos necesarios para configuración y funcionamiento de la aplicación.

Library:

Contiene la lista de funciones para manipulación de archivos, exportaciones de un formato a otro, etc.

Modules:

Contiene las clases que representan a los controladores de la aplicación.

Views:

Contiene las páginas que serán presentadas al usuario final.

- b. Creando un Proyecto de JF
 - i. Invocando al controlador por defecto.

Jf tiene un controlador por defecto llamado index, este lo encontramos dentro de la carpeta modules.

Luego, vamos a ejecutar la siguiente url en nuestro navegador para comenzar: (50 es el puerto con el que esta trabajando la aplicacion)

<http://localhost:50/jfprueba/>

⬅ Pagina Principal-> Bienvenido

Esta es una pagina de Inicio.

Gracias por usar Janos Framework..

ii. Agregando el primer controlador

Luego de haber dado una breve leída acerca del concepto de “Controllers”, vamos a ver cómo crear uno en JF.

Los controllers se almacenan en la carpeta “**modules**” y tienen las siguientes condiciones:

- El nombre del archivo que contiene el controlador es el mismo nombre de la clase.
- El nombre de la clase del controlador no debe coincidir con el nombre de las clases en la carpeta “**dataobjects**”.

Vamos a crear un controlador llamado products y lo guardamos en la carpeta “modules”.

Los controladores tienen un método principal por defecto llamado **defaultaction**.

Vamos a proceder a crear uno llamado “**products**”.

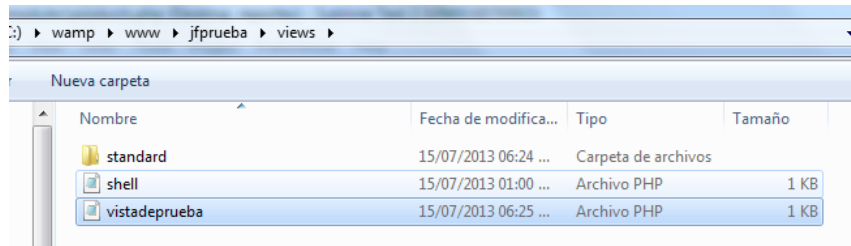
```
products.php x
1  <?php
2
3  class products
4  {
5      public function defaultaction()
6      {
7          // Metodo Aqui...
8      }
9  }
10 }
```

c. Renderizando una pagina

i. Creando y generando una vista

Para poder invocar a una vista se hace uso de la clase **view**.

Vamos a crear una vista, en la cual crearemos un archivo llamado vistadeprueba.php y lo guardaremos en la sgte ruta:

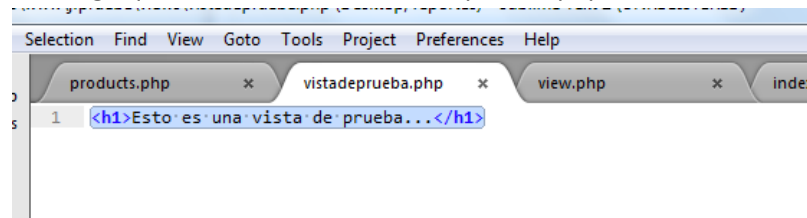


Y para poder llamar a la vista desde el controlador haremos lo siguiente:

```
<?php
```

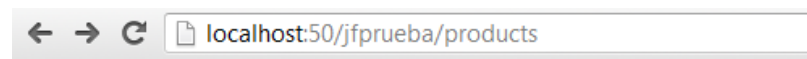
```
class products
{
    public function defaultaction()
    {
        echo view::show("vistadeprueba");
    }
}
```

El código que contiene la vista vistadeprueba.php es:



Ejecutamos la url en nuestro navegador:

<http://localhost:50/jfprueba/products>



Esto es una vista de prueba...

ii. Enviando datos a una vista

Para enviar datos a una vista podemos proceder de 2 maneras.

a) Almacenando en un array.

```
<?php
```

```
class products
{
    public function defaultaction()
    {
```

```

$params = array();
$params["titulo"] = "Mi Titulo 1";
$params["mensaje"] = "Este es mi Mensaje...";
echo view::show("vistadeprueba",$params);
    }
}

```

b) Enviando la data directamente:

```

<?php

class products
{
    public function defaultaction()
    {
        echo view::show("vistadeprueba",array(
            'titulo' => "Mi Titulo 1",
            'mensaje' => "Este es mi Mensaje..."
        ));
    }
}

```

En ambos métodos, el modo para acceder a los valores de la vista enviada se hace a través del elemento **\$view["valor"]**.

```

<h1><?php echo $view["titulo"];?></h1>

```

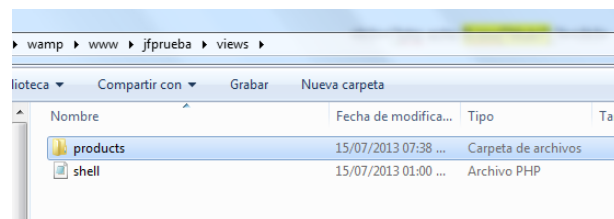
```

<p>
<?php
    echo $view["mensaje"];
?>
</p>

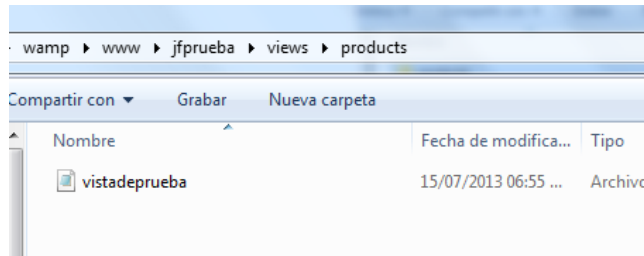
```

iii. Creando vistas en Carpetas.

Puede darse el caso en el que deseemos guardar una vista dentro de un directorio. Por ejemplo si deseamos crear una carpeta llamada products dentro del directorio **"views"**:



Por ultimo cortamos y pegamos el archivo vistadeprueba.php dentro de la carpeta products.



Ahora para llamar dicha vista desde el controlador seria así:

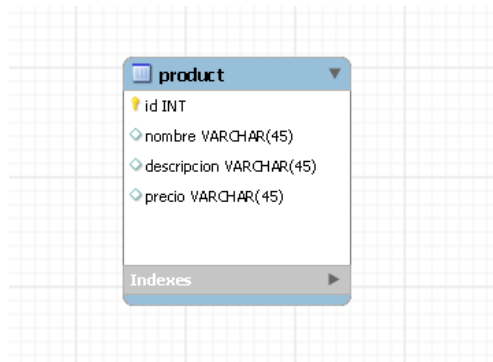
```
<?php
```

```
class products
{
    public function defaultaction()
    {
        echo view::show("products/vistadeprueba");
    }
}
```

d. Una aplicación de Inicio

i. Diseñando el modelo

Este es el punto de partida para comenzar a crear un proyecto usando Janos Framework, pues vamos a crear un Carrito de Compras.



3. SportStore: Una aplicación real

a. Como comenzar

i. Httaccess

El archivo .htaccess es el archivo que es lo primero que tiene que configurarse al poner en marcha una aplicación.

Cambiamos la variable **PATH**

Options +FollowSymLinks

RewriteEngine On

The first 2 conditions may or may not be relevant for your needs

If the request is not for a valid file

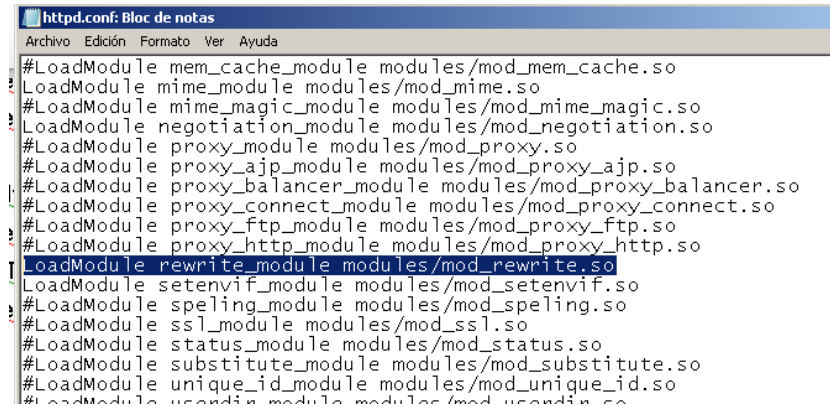
RewriteCond %{REQUEST_FILENAME} !-d


```
RewriteCond %{REQUEST_URI} !^/{PATH}/.*$
RewriteRule ^(.*)$ /{PATH} /$1 [QSA,L]
```

```
# If the request is not for a valid directory
RewriteCond %{REQUEST_FILENAME} !-f
# This rule converts your flat link to a query
RewriteRule ^(.*)$ /{PATH} /index.php?url=$1 [L,NC,NE]
```

ii. Httpd.conf

Considerar que la siguiente línea en el archivo httpd.conf este habilitada:



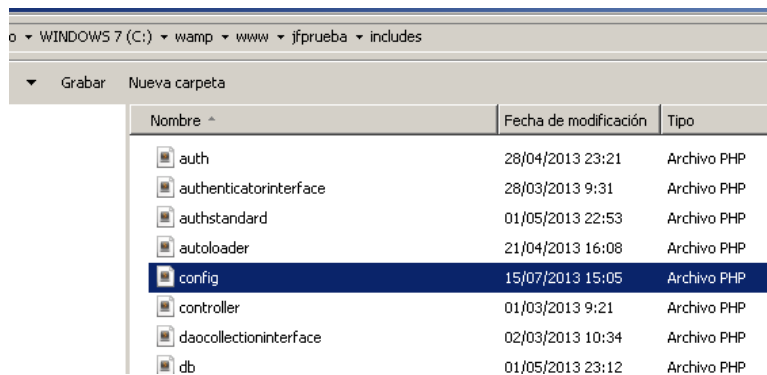
```
#LoadModule mem_cache_module modules/mod_mem_cache.so
LoadModule mime_module modules/mod_mime.so
#LoadModule mime_magic_module modules/mod_mime_magic.so
LoadModule negotiation_module modules/mod_negotiation.so
#LoadModule proxy_module modules/mod_proxy.so
#LoadModule proxy_ajp_module modules/mod_proxy_ajp.so
#LoadModule proxy_balancer_module modules/mod_proxy_balancer.so
#LoadModule proxy_connect_module modules/mod_proxy_connect.so
#LoadModule proxy_ftp_module modules/mod_proxy_ftp.so
#LoadModule proxy_http_module modules/mod_proxy_http.so
LoadModule rewrite_module modules/mod_rewrite.so
LoadModule setenvif_module modules/mod_setenvif.so
#LoadModule spelling_module modules/mod_spelling.so
#LoadModule ssl_module modules/mod_ssl.so
#LoadModule status_module modules/mod_status.so
#LoadModule substitute_module modules/mod_substitute.so
#LoadModule unique_id_module modules/mod_unique_id.so
#LoadModule userdir_module modules/mod_userdir.so
```

iii. Config.php

El siguiente paso de la aplicación es configurar el path de la aplicación tal como la base de datos, temas, lenguaje, etc.

Para iniciar solamente modificaremos el path de la aplicación.

Abrimos el archivo config.php que se encuentra en la ruta:



Nombre	Fecha de modificación	Tipo
auth	28/04/2013 23:21	Archivo PHP
authenticatorinterface	28/03/2013 9:31	Archivo PHP
authstandard	01/05/2013 22:53	Archivo PHP
autoloader	21/04/2013 16:08	Archivo PHP
config	15/07/2013 15:05	Archivo PHP
controller	01/03/2013 9:21	Archivo PHP
daocollectioninterface	02/03/2013 10:34	Archivo PHP
db	01/05/2013 23:12	Archivo PHP

Procedemos a cambiar estos valores:

```
<?php
```

```
/*Config Application*/
```

```
$config = array();

// General
$config['path'] = "my path";
$config['language'] = 'en';
$config['theme'] = 'my theme';
$config['urlbase'] = 'url localhost';
```

Ejemplo:

Si nuestra aplicacion tuviera como path la sgte url:

<http://mydomain.com/mypath/>

Nuestro config.php seria asi:

```
<?php
```

```
/*Config Application*/
```

```
$config = array();

// General
$config['path'] = "mypath";
$config['language'] = 'en';
$config['theme'] = 'my theme';
$config['urlbase'] = 'http://mydomain.com/';
```

b. Comenzando con el modelo de Dominio

i. Conociendo los DataObjects

Vamos a crear la clase que representara a la entidad producto, que esta representada por la tabla del mismo nombre, cuyos atributos se pueden ver arriba tales como: id, nombre, descripción y precio.

Para eso crearemos un archivo llamado product.php y lo guardaremos en la carpeta objects.

Luego agregamos el siguiente código al archivo.

```
<?php
```

```
class product extends dao
{
    protected $table = __CLASS__;
    protected $key = "id";
}
```

Todos los objetos heredaran de la clase DAO, cuya característica principal es que implementan los métodos para actualizar, eliminar y registrar en la base de datos. Además

de eso podemos acceder y modificar las propiedades del objeto sin necesidad de declararlos en la clase. De esto se encarga la clase DAO al momento de asignar las variables de la clase. Para terminar el atributo \$key, hace referencia al nombre de la clave primaria de la tabla.

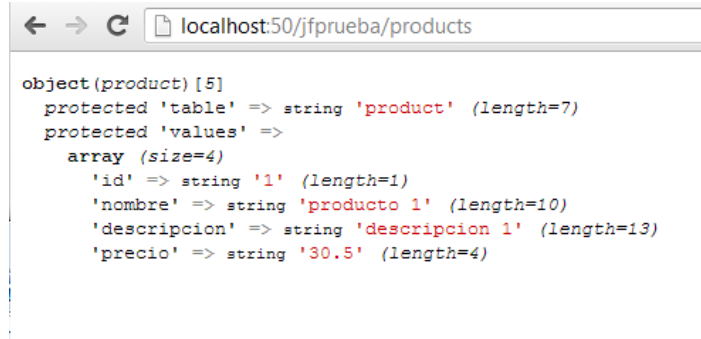
Asi por ejemplo:

Seteando a los valores de una entidad.

```
<?php
class products
{
    public function defaultaction()
    {
        $product = new product();
        $product->id = "1";
        $product->nombre = "producto 1";
        $product->descripcion = "descripcion 1";
        $product->precio = "30.5";

        var_dump($product);
    }
}
```

Al ejecutar aparece asi:



```
object(product) [5]
  protected 'table' => string 'product' (length=7)
  protected 'values' =>
    array (size=4)
      'id' => string '1' (length=1)
      'nombre' => string 'producto 1' (length=10)
      'descripcion' => string 'descripcion 1' (length=13)
      'precio' => string '30.5' (length=4)
```

Obteniendo los valores de una entidad.

```
<?php
class products
{
    public function defaultaction()
    {
        /*Asignando valores*/
        $product = new product();
        $product->id = "1";
        $product->nombre = "producto 1";
```

```

$product->descripcion = "descripcion 1";
$product->precio = "30.5";

var_dump($product);

/*Obteniendo Valores*/
echo "Id : ".$product->id." ,Nombre :".$product->nombre.
      ",descripcion :". $product->descripcion.
      ",precio :". $product->precio;

    }

}

```

ii. Creando falsos repositorios(collections)

Los Collection son clases que almacenan un conjunto de objetos definidos en los dataobjects.

Estos heredan de la clase daocollection e implementan la interfaz daocollectioninterface, cuyo único contrato define el método para obtener la lista de objetos del tipo que se quiere.

Para crear un objeto **collection** de tipo product crearemos el archivo productcollection.php y lo guardamos en la carpeta **collections**.

Escribimos el sgte código:

```

<?php
class productcollection extends daocollection implements daocollectioninterface
{
    public function __construct($current=NULL)
    {
        $this->current = $current;
    }
    public function getlist()
    {
        /*Metodo para implementar*/
    }
}

```

Poblando data al productcollection.

Vamos a implementar el método getlist():

```

<?php
class productcollection extends daocollection implements daocollectioninterface

```

```

{
    public function __construct($current=NULL)
    {
        $this->current = $current;
    }
    public function getlist()
    {
        /*Product 1*/
        $product1 = new product();
        $product1->id = "1";
        $product1->nombre = "producto 1";
        $product1->descripcion = "descripcion 1";
        $product1->precio = "30.5";

        /*Product 2*/
        $product2 = new product();
        $product2->id = "2";
        $product2->nombre = "producto 2";
        $product2->descripcion = "descripcion 2";
        $product2->precio = "3.5";

        /*Product 2*/
        $product3 = new product();
        $product3->id = "3";
        $product3->nombre = "producto 3";
        $product3->descripcion = "descripcion 3";
        $product3->precio = "20.5"
        $this->populate(array((array)$product1,(array)$product2,(array)$product3),
'product');
    }
}

```

Ahora vamos al controlador y editamos el sgte código:

```

<?php
class products
{
    public function defaultaction()
    {
        $productcollection = new productcollection();
        $productcollection->getlist();
        var_dump($productcollection);
    }
}

```

Usando var_dump() esto seria el código que se genera.

```

← → ↻ localhost:50/jfprueba/products

object(productcollection) [5]
  protected 'position' => int 0
  protected 'storage' =>
    array (size=3)
      0 =>
        object(product) [9]
          protected 'table' => string 'product' (length=7)
          protected 'values' =>
            array (size=2)
              ...
      1 =>
        object(product) [10]
          protected 'table' => string 'product' (length=7)
          protected 'values' =>
            array (size=2)
              ...
      2 =>
        object(product) [11]
          protected 'table' => string 'product' (length=7)
          protected 'values' =>
            array (size=2)
              ...
  public 'current' => null

```

Recorriendo un Collection

Vamos a listar los objetos creados anteriormente usando productcollection:

<?php

```

class products
{
    public function defaultaction()
    {
        $productcollection = new productcollection();
        $productcollection->getlist();

        foreach ($productcollection as $product) {

            //Accedemos a los objetos.
            echo "ID : ". $product->id.", Product : ".$product->nombre."<br/>";
        }
    }
}

```

Resultado:



ID : 1, Product : producto 1
ID : 2, Product : producto 2
ID : 3, Product : producto 3

Entre las propiedades y métodos de las colecciones tenemos:

- a) **Populate(\$array, \$dataobject)** => almacena el conjunto de objetos transformados en array en un \$dataobject, donde \$dataobject es un string que representa el nombre de la entidad donde se va a almacenar la tabla.
- b) **Current()** => Obtiene el objeto actual.
- c) **Key()** => Devuelve la posición del objeto
- d) **Next()** => Avanza una posición a la colección.
- e) **Rewind()** => Reinicializa la posición del objeto a CERO.
- f) **Valid()** => Verifica si existe el objeto actual.

Recorriendo los objetos Collection usando current()

Podemos hacer la iteración anterior usando el método current() y mediante key() ver la posición en la que se encuentra nuestro recorrido.

```
<?php
```

```
class products
```

```
{
```

```
    public function defaultaction()
```

```
    {
```

```
        $productcollection = new productcollection();
```

```
        $productcollection->getlist();
```

```
        foreach ($productcollection as $product)
```

```
        {
```

```
            var_dump($productcollection->key());
```

```
            var_dump($productcollection->current());
```

```
        }
```

```
    }
```

```
}
```

Al ejecutar:

```

int 0

object(product)[8]
  protected 'table' => string 'product' (length=7)
  public 'values' =>
    array
      'id' => string '1' (length=1)
      'nombre' => string 'producto 1' (length=10)
      'descripcion' => string 'descripcion 1' (length=13)
      'precio' => string '30.5' (length=4)

int 1

object(product)[9]
  protected 'table' => string 'product' (length=7)
  public 'values' =>
    array
      'id' => string '2' (length=1)
      'nombre' => string 'producto 2' (length=10)
      'descripcion' => string 'descripcion 2' (length=13)
      'precio' => string '3.5' (length=3)

int 2

object(product)[10]
  protected 'table' => string 'product' (length=7)
  public 'values' =>
    array
      'id' => string '3' (length=1)
      'nombre' => string 'producto 3' (length=10)
      'descripcion' => string 'descripcion 3' (length=13)
      'precio' => string '20.5' (length=4)

```

Recorriendo la colección usando **next()**

```
<?php
```

```
class products
```

```
{
```

```
    public function defaultaction()
```

```
    {
```

```
        $productcollection = new productcollection();
```

```
        $productcollection->getlist();
```

```
        // Posicion 0
```

```
        var_dump($productcollection->current());
```

```
        $productcollection->next();
```

```
        // Posicion 1
```



```

        var_dump($productcollection->current());

        // Posicion 2

        $productcollection->next();

        var_dump($productcollection->current());

    }

}

```

- c. Mostrando una lista de Productos(CheckBoxList, Table, RadioButtonList, SelectList)

Para seguir con nuestro recorrido mostraremos una lista de objetos y le daremos el formato de una tabla. Bien pudiéramos hacerlo a través de un foreach recorriendo los objetos uno a uno. En JF tenemos el componente **table** que te evitara estos pasos.

Veamos cómo funciona:

El objeto table contiene se enlaza directamente con los objetos collections. Para este caso colocaremos como datasource productcollection.

Agregamos el siguiente código:

```

$objtable = new table();
$objtable->cabecera = array('ID','NOMBRE','DESCRIPCION','PRECIO');
$objtable->datasource = $productcollection;
$objtable->datatextfield = array('id','nombre', 'descripcion','precio');
echo view::show("products/listar",array("tabla"=>$objtable->build()));

```

En la cabecera colocamos las cabeceras de la tabla. Estas representan a cada columna.

En datasource va el objeto collection.

La propiedad datatextfield contiene un array de las columnas que se desea que salgan. Los nombres son las propiedades de la entidad product.

Y por último el método build() construye la tabla y luego de eso la tabla estará lista para ser mostrada.

Ahora crearemos la vista llamada listar.php y lo guardaremos en views/products/.

Listar.php

```

<style type="text/css">
table{

```

```

        width: 100%;
        border: 1px solid #999;
    }
    table tr td{
        border: 1px solid #999;
    }
</style>
<?php
echo $view["tabla"];
?>

```

Aquí le dimos un poco de estilo a la tabla.

El código completo donde mostrara el controlador products es:

```
<?php
```

```

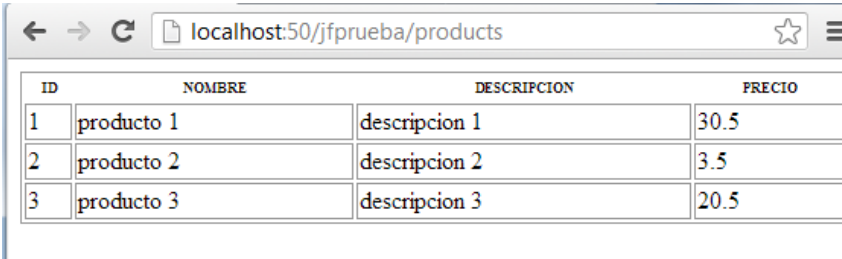
class products
{
    public function defaultaction()
    {

        $productcollection = new productcollection();
        $productcollection->getlist();

        $objtable = new table();
        $objtable->cabecera = array('ID','NOMBRE','DESCRIPCION','PRECIO');
        $objtable->datasource = $productcollection;
        $objtable->datatextfield = array('id','nombre','descripcion','precio');
        echo view::show("products/listar",array("tabla"=>$objtable->build()));
    }
}

```

Resultado:



| ID | NOMBRE | DESCRIPCION | PRECIO |
|----|------------|---------------|--------|
| 1 | producto 1 | descripcion 1 | 30.5 |
| 2 | producto 2 | descripcion 2 | 3.5 |
| 3 | producto 3 | descripcion 3 | 20.5 |

d. Conectando a la Base de Datos

Anteriormente se trabajó con datos escritos en código duro. Ahora el proceso consistirá en usar un gestor de base de datos, en este caso MYSQL, uno ya conocido y usaremos la entidad product.

i. Definiendo el esquema de Base de Datos.

Crearemos una tabla en mysql llamado product con los campos id, nombre, descripción y precio en una base de datos llamada jfprueba y luego insertamos unos registros.

Codigo SQL.

```
CREATE TABLE IF NOT EXISTS `product` (  
  `id` int(11) NOT NULL AUTO_INCREMENT,  
  `nombre` varchar(40) NOT NULL,  
  `descripcion` varchar(40) NOT NULL,  
  `precio` int(11) NOT NULL,  
  PRIMARY KEY (`id`)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=5 ;
```

```
--  
-- Volcado de datos para la tabla `product`  
--
```

```
INSERT INTO `product` (`id`, `nombre`, `descripcion`, `precio`) VALUES  
(1, 'Aceite Friol', 'Aceite Friol', 31),  
(2, 'Jabon Bolivar', 'Jabon Bolivar', 10),  
(3, 'Agua San Carlos', 'Agua San Carlos', 4),  
(4, 'Fideos Don Camilo', 'Fideos Don Camilo', 5);
```

Preparando el archivo config.php

Vamos a configurar nuestro archivo para poder trabajar ya con la base de datos mysql.

Datos:

Host: localhost

User: root

Pass: " "

Database: jfprueba

Driver: mysql

Nuestro config.php debe tener estos datos:

```
$config['driver'] = "mysql";  
$config['host'] = "localhost";  
$config['database'] = "jfprueba";  
$config['user'] = "root";  
$config['pass'] = "";
```

ii. Integración con PDO(Múltiples Origenes de BD)

El manejo de las bases de datos usados en JF está basado en la librería PDO que viene por defecto en algún software como wampserver, etc. Por lo que su uso da cierta familiaridad para quienes han utilizado esta librería.

El objetivo de la misma es tener una misma forma de realizar consultas y transacciones sin tener que preocuparse de la base de datos en la cual se maneje.

Todos los gestores de base de datos en JF se basan en la clase abstracta “db” cuyos métodos tienen que implementarse y son los sgtes:

```
<?php
abstract class db{
    public static function factory($type)
    {
        return call_user_func(array($type,'getInstance'));
    }
    abstract public function execute($query);
    abstract public function getArray($query);
    abstract public function insertGetID($query);
    abstract public function getNumRows($query);
}
```

Métodos:

Factory: Obtiene la instancia del tipo de gestor al cual se va a llamar.

Execute: Ejecuta una consulta sql y devuelve un numero de filas afectadas. Vale su uso para inserts, updates o deletes.

GetArray (): Transforma y devuelve en un array el resultado de una consulta select.

InsertGetID (): Devuelve el ID de un insert realizado.

GetNumRows (): Devuelve el número de filas de una consulta select

- iii. Seleccionando el DataSource(Mysql, Sql Server, Oracle, PostGress)
En esta sección vamos a comenzar haciendo uso de nuestra primera consulta a mysql usando JF.

Creando un repositorio real

Vamos a modificar la implementación del método **getlist ()** de nuestro objeto productcollection por el siguiente código:

```

<?php
class productcollection extends daocollection implements daocollectioninterface
{
    public function __construct($current=NULL)
    {
        $this->current = $current;
    }

    public function getlist()
    {
        $connection = db::factory('mysql');
        $connection->connect();
        $sql = "select * from product;";
        $results = $connection->getArray($sql);
        $this->populate($results, 'product');
    }
}

```

Y si visualizamos en el navegador la url obtendremos el mismo resultado.

e. Creando un Mantenedor

Preparando el escenario

El primer punto de partida para realizar este ejemplo es modificar el archivo listar.php que se encuentra dentro de views/listar.php

Codigo listar.php

```

<style type="text/css">
table{
    width: 100%;
    border: 1px solid #999;
}
table tr td{
    border: 1px solid #999;
}
</style>

<a href = "<?php echo uri::getPath();?>products/register">[Nuevo Producto]</a>
<?php
echo $view["tabla"];
?>
Cuya vista final sera esto:

```



[\[Nuevo Producto\]](#)

| ID | NOMBRE | DESCRIPCION | PRECIO | Accion |
|----|-------------------|-------------------|--------|---|
| 1 | Aceite Friol | Aceite Friol | 31 | Edit / Delete / |
| 2 | Jabon Bolivar | Jabon Bolivar | 10 | Edit / Delete / |
| 3 | Agua San Carlos | Agua San Carlos | 4 | Edit / Delete / |
| 4 | Fideos Don Camilo | Fideos Don Camilo | 5 | Edit / Delete / |

Ahora modificaremos la clase products que se encuentra en modules/products.php:

```
<?php
```

```
class products
```

```
{
```

```
    public function defaultaction()
    {
```

```
        $productcollection = new productcollection();
        $productcollection->getlist();
```

```
        $objtable = new table();
        $objtable->cabecera = array('ID','NOMBRE','DESCRIPCION','PRECIO');
        $objtable->datasource = $productcollection;
        $objtable->datatextfield = array('id','nombre','descripcion','precio');
```

```
        $objtable->edit = true;
```

```
        $objtable->delete = true;
```

```
        $objtable->addAction = true;
```

```
        $objtable->editurl = "products/edit/id";
```

```
        $objtable->deleteurl = "products/delete/id";
```

```
        echo view::show("products/listar",array("tabla"=>$objtable->build()));
    }
```

```
}
```

Las propiedades edit y delete por defecto están en false, al colocarlas en true, aparecen las opciones para “Editar” y “Eliminar las los registros”.

Estas anteriores no se muestran si la propiedad addAction no esta en true.

Las propiedades editurl y deleteurl son los templates de cual sera la url que se tomara al darle clic en el link para eliminar y actualizar respectivamente.

Almacenando un Producto:

Para registrar un nuevo producto vamos a crear un nuevo método en el controlador products llamado register

```
public function register(){
    echo view::show('products/registrar');
}
```

Cuya vista será la siguiente.

```
<form action="<?php echo uri::getPath();?>products/procesarregistro" method="POST" id =
"register">
```

```
    <table>

        <tr>

            <td> IdProducto </td>
            <td> <input type="text" placeholder="ID" readonly="readonly" value="0"
id="id"/></td>
        </tr>
        <tr>

            <td> Nombre </td>
            <td> <input name="nombre" id="nombre" type="text" placeholder="Nombre"
/></td>
        </tr>
        <tr>

            <td> Descripcion </td>
            <td> <input name="descripcion" id="descripcion" type="text"
placeholder="descripcion" /></td>
        </tr>
        <tr>

            <td> Precio </td>
            <td> <input name="precio" id="precio" type="text" placeholder="precio" /></td>
        </tr>
        <tr>

            <td colspan="2"><input type="submit" class="" value="Registrar"
id="btnRegistrar"></div></td>
        </tr>
    </table>
</form>
```

Cuya vista es la siguiente:



Nuevo Producto

IdProducto	<input type="text" value="0"/>
Nombre	<input type="text" value="Nombre"/>
Descripcion	<input type="text" value="descripcion"/>
Precio	<input type="text" value="precio"/>
<input type="button" value="Registrar"/>	

Cuando se dé clic en el botón Registrar se dirigirá a la url : products/procesarregistro, cuyo atributo se encuentra en el formulario, además del envío mediante POST.

Vamos a implementar el método para registrar el producto.

```
public function procesarregistro(){  
  
    $product = new product();  
  
    $product->nombre = $_POST["nombre"];  
    $product->descripcion = $_POST["descripcion"];  
    $product->precio = $_POST["precio"];  
  
    $issuccess = $product->save();  
  
    uri::sendto('products/');  
}
```

Lo único que hacemos aquí es tomar los valores enviados por POST y crear un objeto de tipo product y asignamos sus variables. Tomar en cuenta que los valores de los campos nombre, descripción y precio de la tabla product en la base de datos guardan una estrecha relación con lo asignado aquí. Esto es parte del ORM del framework ya que no necesitamos crear ni set ni get así que simplemente accedemos directamente.

Volviendo a lo nuestro, llenamos el formulario de registro de producto y le damos

Nuevo Producto

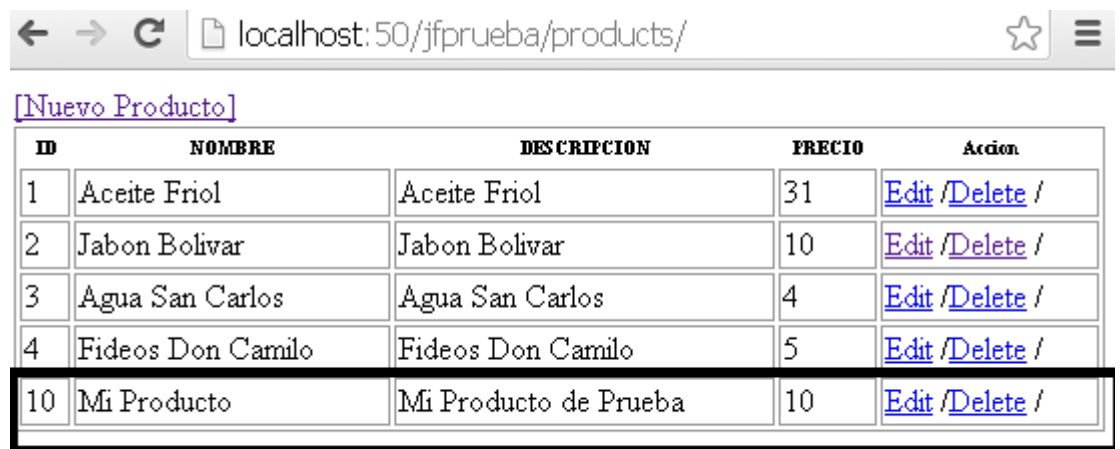
IdProducto

Nombre

Descripcion

Precio

Clic en Registrar. Esto nos reenviara a la pantalla de la lista de productos.



ID	NOMBRE	DESCRIPCION	PRECIO	Acción
1	Aceite Friol	Aceite Friol	31	Edit / Delete /
2	Jabon Bolivar	Jabon Bolivar	10	Edit / Delete /
3	Agua San Carlos	Agua San Carlos	4	Edit / Delete /
4	Fideos Don Camilo	Fideos Don Camilo	5	Edit / Delete /
10	Mi Producto	Mi Producto de Prueba	10	Edit / Delete /

Efectivamente, en pocos pasos logramos registrar un producto.

Validando un poco:

Como que un poco de validación de datos valdría bien verdad?

Bueno, si deseamos que los valores de nombre y precio sean siempre requeridos.

Vamos a echarle a modificar un poco nuestro formulario.

Veamos...

Modificamos nuestro método procesarregistro():

```
public function procesarregistro(){
```

```
    $product = new product();
```

```
    $nombre = $_POST['nombre'];
```

```
    $precio = $_POST['precio'];
```

```

        if(empty($nombre)){
            lib::seterror('Nombre debe ser ingresado');
            uri::sendto('products/register');
        }

        if(empty($precio)){
            lib::seterror("Precio debe ser ingresado");
            uri::sendto('products/ register');
        }
        $product->descripcion = $_POST["descripcion"];

        $issuccess = $product->save();
        uri::sendto('products/');
    }

```

Algo sencillo de explicar en el código. Si no ingresamos ningún valor en las variables nombre y precio automáticamente nos redirigirá a la pantalla registro de producto sin insertar ningún registro y no sabríamos que paso.

Esto debe de ser un poco engorroso ya que no mostramos ningún mensaje de alerta al usuario indicado que está mal. Para eso en la vista de registro agregamos esto al inicio de la página de registro (views/products/ register".php) indicando los mensajes tanto de errores como de éxito.

```

<?php
    echo view::show('standard/default/errors');

?>

<?php
    echo view::show("standard/default/success");

?>

```

Ahora volvemos a ejecutar el código y a propósito enviamos nuestro formulario sin ingresar ningún tipo de dato.

Nuevo Producto

× **Error!**

Nombre debe ser ingresado

IdProducto

Nombre

Descripcion

Precio

Registrar

Se mostró un mensaje indicando que el nombre debe ser ingresado. Si recargamos nuevamente la página, este mensaje desaparece. Pues si, si tiempo de vida es una sola vez mostrada a la siguiente recarga desaparece.


Ahora bien si todo ingreso ha sido correcto l que es el guardado o error al registro de datos del producto entonces agregamos este codiguito al final del método procesarregistro():

```
if($issuccess >= 1){  
    lib::setsuccess('El producto se ha registrado con Exito');  
}  
else{  
    lib::seterror('Se ha producido un error al registrar un producto');  
}  
uri::sendto('products/');
```

Y nuevamente este código en views/products/listar.php al inicio de la pagina.

```
<?php  
    echo view::show('standard/default/errors');  
    echo view::show('standard/default/success');  
?>
```

Ingresamos nuevos valores en nuestro formularioy si todo sale bien veremos esto:

 **Success!**

El producto se ha registrado con Exito

[\[Nuevo Producto\]](#)

ID	NOMBRE	
1	Aceite Friol	Aceite Friol
2	Jabon Bolivar	Jabon Bolivar
3	Agua San Carlos	Agua San Carlos
4	Fideos Don Camilo	Fideos Don Camilo
10	Mi Producto	Mi Producto de Prueba
11		

Actualizando un Producto

Para editar un producto simplemente vamos a crear un método edit en el modulo products.

Este método debe tener la forma de nuestra url enviada en la tabla:

\$objtable->editurl = "products/edit/id";

```
public function edit($param){  
    var_dump($param);  
}
```

El valor de \$param es un array cuyas posiciones son:

[0]=> Nombre del módulo o controlador (products)

[1]=> Nombre del método (edit)

[2] a mas=> Parámetros enviados por GET.

Vamos a comprobar esto mediante seleccionando un elemento de la tabla:



```
array  
0 => string 'products' (length=8)  
1 => string 'edit' (length=4)  
2 => string '1' (length=1)
```

Necesitamos crear un método en la clase productcollection para buscarxid:

```
public function obtenerlistaxid($id)  
{  
    $connection = db::factory('mysql');  
    $connection->connect();  
    $sql = "select * from product where id = '{$id}'";  
    $results = $connection->getArray($sql);  
    $this->populate($results, 'product');  
}
```

Acto seguido modificamos nuestro método edit del modulo products:

```

public function edit($param){
    $id = $param[2];
    $collection = new productcollection();
    $collection->obtenerlistaxid($id);

    var_dump($collection->current());

}

```

Como de antemano sabes que siempre devolverá un solo elemento accedemos directamente con el método `current()` del collection.

```

object(product)[8]
  protected 'table' => string 'product' (length=7)
  protected 'key' => string 'id' (length=2)
  public 'values' =>
    array
      'id' => int 1
      'nombre' => string 'Aceite Friol' (length=12)
      'descripcion' => string 'Aceite Friol' (length=12)
      'precio' => int 31

```

Ahora esto lo enviaremos a una vista con el producto seleccionado:

```

public function edit($param){
    $id = $param[2];

    $collection = new productcollection();
    $collection->obtenerlistaxid($id);

    echo view::show("products/editar", array('producto'=>$collection->current()));
}

```

Ahora nos queda recrear la vista editar:

```

<h1>Nuevo Producto</h1>
<?php
    echo view::show('standard/default/errors');
?>
<?php
    echo view::show('standard/default/success');
?>
<form action="<?php echo uri::getPath();?>products/procesaractualizar " method="POST" id = "register">

    <table>

        <tr>

            <td> IdProducto </td>
            <td> <input type="text" name="id" placeholder="ID" readonly="readonly"
                value="<?php echo $view['producto']->id;?>" id="id"/></td>

        </tr>
        <tr>

            <td> Nombre </td>

```

```

        <td> <input name="nombre" id="nombre" type="text" placeholder="Nombre"
            value="<?php echo $view['producto']->nombre;?>" /></td>
    </tr>
    <tr>
        <td> Descripcion </td>
        <td> <input name="descripcion" id="descripcion" type="text"
            placeholder="descripcion"
            value="<?php echo $view['producto']->descripcion;?>" /></td>
    </tr>
    <tr>
        <td> Precio </td>
        <td> <input name="precio" id="precio" type="text" placeholder="precio"
            value="<?php echo $view['producto']->precio;?>" /></td>
    </tr>
    <tr>
        <td colspan="2"><input type="submit" class="" value="Registrar"
            id="btnRegistrar"></div></td>
    </tr>
</table>
</form>

```



Editar Producto

IdProducto	<input type="text" value="1"/>
Nombre	<input type="text" value="Aceite Friol"/>
Descripcion	<input type="text" value="Aceite Friol"/>
Precio	<input type="text" value="31"/>
<input type="button" value="Actualizar"/>	

Creamos el método procesaractualizar() en el modulo productos.

```

public function procesaractualizar(){
    $product = new product();
    $product->id = $_POST['id'];
    $nombre = $_POST['nombre'];
    $precio = $_POST['precio'];

    if(empty($nombre)){
        lib::seterror('Nombre debe ser ingresado');
        uri::sendto('products/edit');
    }

    if(empty($precio)){
        lib::seterror('Precio debe ser ingresado');
        uri::sendto('products/edit');
    }

    $product->nombre = $nombre;

```

```

$product->precio = $precio;
$product->descripcion = $_POST["descripcion"];

$issuccess = $product->save();

if($issuccess >= 1){
    lib::setsuccess('El producto se ha actualizado con Exito');
}
else{
    lib::seterror('Se ha producido un error al actualizar un producto');
}

uri::sendto('products');
}

```

Notese que aquí acabamos de enviar el id del producto como propiedad del objeto product. Esto servirá para hacer la actualización respectiva ya que si no se enviar lo tomara como un nuevo registro.

f. Estilos

i. Creando masterpages.

Algo que podría ahorrarte la vida es el hecho de reutilizar código y evitar los famosos “copy and paste” y tanto include y require.

En JF puedes crearte y tomarte la libertad de crear un tema o seleccionar uno ya hecho en JF.

ii. Seleccionando temas

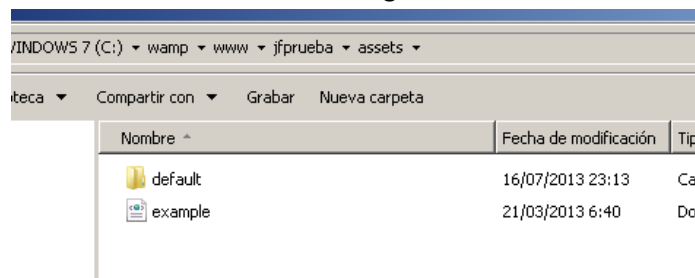
Para esto tenemos que hacer lo siguiente:

Abrimos el archivo config.php y editamos lo siguiente:

El tema que usaremos es el por defecto “default”.(nombre de la carpeta)

```
$config['theme'] = 'default';
```

Este hace referencia a la ruta siguiente:



Aquí guardamos todos nuestros archivos css, js e imágenes, para llevar un orden.

iii. Usando vistas conocidas(header, footer, sidebar, errors, success)

Ahora el siguiente paso es crear el archivo masterpage. Esta será nuestra plantilla en la cual trabajara casi toda la aplicación.

Este masterpage lo encontraremos en :
views/standard/default/masterpage. Y tendrá una estructura que por lo general será así:

```
<?php echo view::show('standard/default/header');?>
<div class="page secondary">
    <div class="page-header">
        <div class="page-header-content">
            <h1><?php echo $titulo;?><small><?php echo $subtitulo;?></small></h1>
            <a href="/" class="back-button big page-back"></a>
        </div>
    </div>

    <div class="page-region">
        <div class="page-region-content">
            <div class="grid">
                <div class="row">
                    <div class="span10">

                        <?php echo $pagemaincontent;?>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>
<?php echo view::show('standard/default/footer');?>
```

Obviamente esto depende de nuestro tema a seleccionar.
Nuestro masterpage contendrá nada mas que 3 variables, titulo, subtitulo y contenido.
Asi si deseamos que una pagina herede de este masterpage colocamos estas líneas de código en la pagina de donde deseamos que use este tema.

En el caso de la vista para listar los productos usamos :

```
<style type="text/css">
table{
    width: 100%;
    border: 1px solid #999;
}
table tr td{
    border: 1px solid #999;
}
</style>
<?php
    echo view::show('standard/default/errors');
?>
<?php
    echo view::show('standard/default/success');
?>
<a href = "<?php echo uri::getPath();?>products/register">[Nuevo Producto]</a>
```



```
<?php
echo $view["tabla"];
?>
<?php
    master::load(ob_get_contents(),"Lista de Productos","Descripcion de Productos");
?>
```

Y si volvemos a visualizar se vera de esta forma:

localhost: 50/jfprueba/products

← Lista de Productos Descripcion de Productos

[Nuevo Producto]

ID	NOMBRE	DESCRIPCION	PRECIO	Accion
1	Aceite Friol	Aceite Friol	31	Edit / Delete /
2	Jabon Bolivar	Jabon Bolivar	10	Edit / Delete /
3	Agua San Carlos	Agua San Carlos	2	Edit / Delete /
4	Fideos Don Camilo	Fideos Don Camilo	5	Edit / Delete /
10	Mi Producto	Mi Producto de Prueba	10	Edit / Delete /
14		Producto de Prueba	0	Edit / Delete /
16		Nuevo Producto	0	Edit / Delete /

localhost: 50/jfprueba/products/edit/1

← Edicion de Productos Descripcion de Productos

IdProducto	<input type="text" value="1"/>
Nombre	<input type="text" value="Aceite Friol"/>
Descripcion	<input type="text" value="Aceite Friol"/>
Precio	<input type="text" value="31"/>
<input type="button" value="Actualizar"/>	

Puedes personalizar este tema enviando mas variables al masterpage pues para ello tendríamos que matricular estas variables en introducirlas en el archivo masterpage.php ubicado en views/standard/default/master.php y también en includes/master.php

Por ejemplo:

```
<?php echo view::show('standard/default/header');?>
<div class="page secondary">
    <div class="page-header">
        <div class="page-header-content">
            <div class="page-header-content">
                <h1><?php echo $titulo;?><small><?php echo $subtitulo;?></small></h1>
                <a href="/" class="back-button big page-back"></a>
            </div>
        </div>
    </div>

    <div class="page-region">
        <div class="page-region-content">
            <div class="grid">
```

```

<div class="row">
  <div class="span10">

    <?php echo $pagemaincontent;?>
  </div>
</div>
</div>
</div>
</div>
</div>
</div>
<?php echo view::show('standard/default/footer');?>

```

Las variables subrayadas en Amarillo previamente guardan una estrecha relación con el método master.php(includes/master.php)

```

<?php
class master{

    public static function load($content,$titulo,$subtitulo)
    {
        global $config;

        ob_end_clean();
        $titulo = $titulo;
        $subtitulo = $subtitulo;
        $pagemaincontent = $content;
        $path = $_SERVER['DOCUMENT_ROOT']
        .'/'.$config["path"].'/views/standard/'.$config["theme"].'/master.php';
        if (is_readable($path)) include $path;
    }

}
?>

```

Así, si deseamos agregar mas variables, habría que agregar mas parámetros en el método de la clase master.

Recordemos que el uso de los temas siempre se configuran en el archivo config.php

4. Autenticación de Usuarios

i. El objeto Session

Para poder usar una sesión en JF solamente basta con hacer lo siguiente:

```

//Asignamos una variable de session
session::setitem('Nombre',$username);

```

```

//Obtenemos la variable.
session::getitem('Nombre');

```

Cuyas definiciones las encontramos en la clase session ubicados en includes/session.php

ii. Autenticacion y acceso.

Algo muy común es ver el uso de sesiones en aplicaciones web para el manejo de logeo de usuarios y restricciones al sistema.

La clase auth de JF contiene una serie de funcionalidades en cuanto a seguridad se refiere y entre ellas podemos hacer lo siguiente:

Comprobar si un usuario esta logeado en el sistema contamos con un método llamado:

```
if(!auth::isloggedin()){  
    // si no esta logeado  
}  
else{  
  
    //Si lo esta.  
}
```

La implementación del método isloggedin() va a depender del algoritmo de seguridad se quiera realizar. Su implementación la encontramos en la clase /includes/autostandard, cuyos parámetros son el usuario y la contraseña como elementos básicos.

```
<?php  
class authstandard implements authenticatorinterface  
{  
    public function authenticate(user $user, $password){  
        // Algoritmo de autenticacion  
        Return true;  
    }  
}
```