JAMES

Iniciación a la programación de módulos en JAMES

Introducción

¿Por dónde vamos?

- Introducción a Linux
- El entorno de desarrollo
- El lenguaje HTML
- Programación en PHP
- Instalación y administración de JAMES
- Introducción a la programación de módulos en JAMES
- Construcción de un módulo de ejemplo en JAMES

Introducción

Al terminar, sabremos:

- Un poco de la arquitectura de JAMES
- Cuál es la estructura de un módulo de JAMES
- Configurar el entorno de trabajo para desarrollar módulos
- Cómo empezar a programar un módulo
- Obtener información acerca del usuario y grupo activos en el sistema

¿Por qué programar en JAMES?

Tres razones fundamentales

Introducción

1. No reinventar la rueda

- Gestión de usuarios, grupos, acceso a base de datos son cosas muy comunes en una aplicación web.
- Usar un *framework* como JAMES permite olvidarse de todo eso y centrarse en la funcionalidad que queremos aportar.
- Desgraciadamente, a los informáticos nos gusta demasiado reinventar la rueda.

2. Usar módulos ya existentes

- Ejemplo:
  - Tenemos que hacer un portal para la Intranet de un grupo de investigación. Tenemos dos opciones:
    - Partir de cero, programando todos los componentes del sistema.
    - Aprovechar los módulos de JAMES que ya hay desarrollados: foro, repositorio, documentación...

3. Aprobar la asignatura ;-)

- El trabajo que hay que entregar para aprobar la asignatura es un módulo de JAMES.
- Tres créditos son una buena razón para aprender.

Introducción

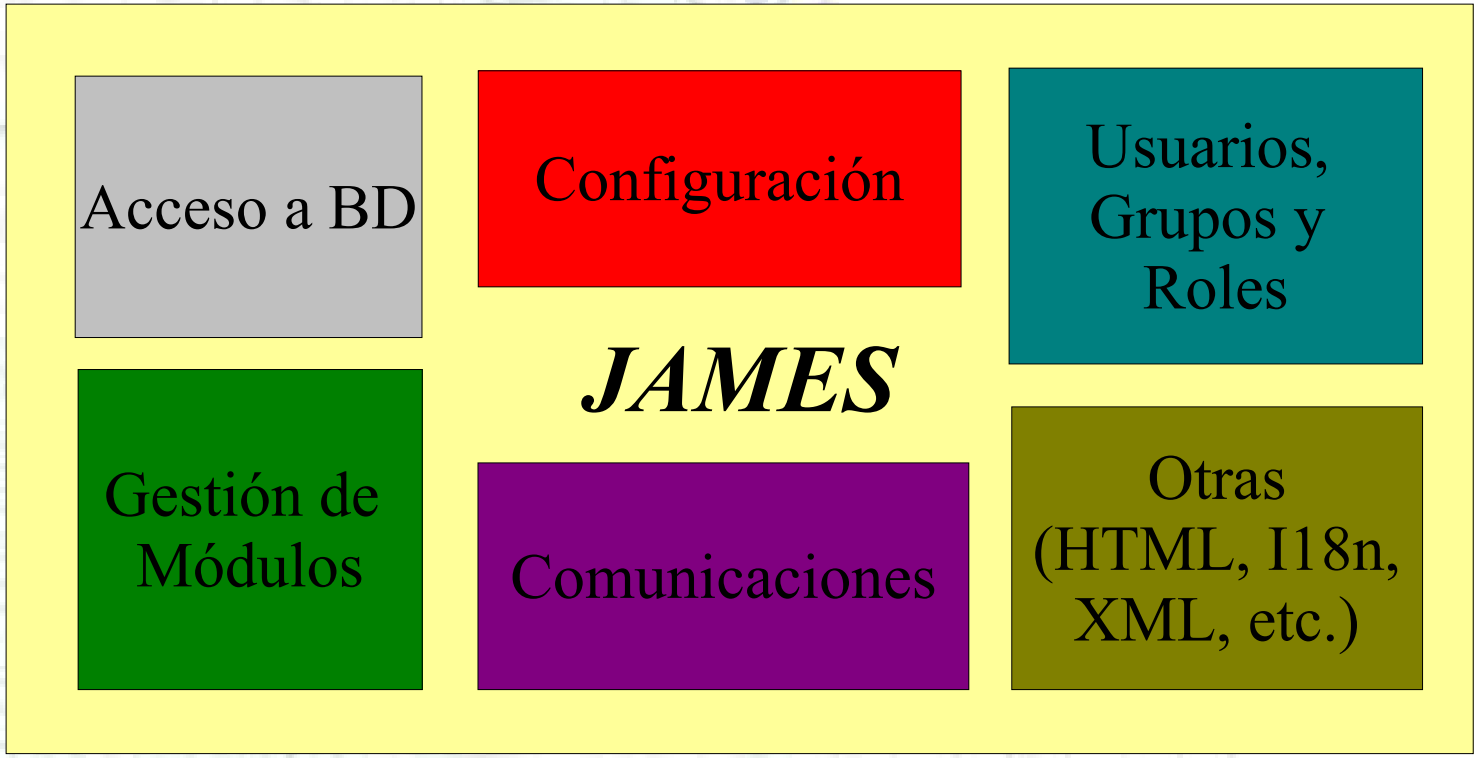
¿Qué aporta JAMES?

- Autentificación de usuarios
- Gestión de usuarios, permisos y grupos
- Acceso a base de datos
- Opciones de configuración para los módulos
- Comunicaciones con otros módulos
- Ayuda para la internacionalización (i18n)
- Clases de ayuda para generar HTML

James por dentro

A nivel conceptual...

- Varios grandes bloques



James por dentro

A nivel de directorios...

- james
  - core [El núcleo del sistema]
  - config [Ficheros de configuración]
  - modules [Módulos instalados]
  - grf [Imágenes, etc.]
  - style [Hojas de estilo]
  - logs [*Logs* del sistema]

Creando módulos

Vamos a seguir los siguientes pasos:

1. Instalación de un JAMES nuevo
2. Configuración del JAMES para poder desarrollar cómodamente
3. Instalación de un módulo vacío que servirá como punto de partida
4. Configuración del entorno de desarrollo
5. Empezar a escribir código :-)

Creando módulos

Estructura básica de un módulo:

- **init.php**: Página que se carga al lanzar el módulo.
- **config.xml**: Fichero de configuración con información sobre el módulo: nombre, descripción, permisos, etc.
- **MiModulo.class.php**: Clase utilizada para implementar la comunicación entre los módulos.

Creando módulos

“¡Hola mundo!”

init.php:

```
<html>
<head>
<title>Mi Modulo</title>
</head>
<body>
   ¡Hola mundo!
</body>
</html>
```

Creando módulos

Problemas de nuestro “Hola mundo”

- Accedamos directamente...
  - http://james.us.es/james/modules/MiModulo/init.php
- Eso es un problema porque...
  - ¿Para qué añadir y quitar módulos de grupos si luego todo el mundo podría verlos?
  - ¿De qué forma sabe el módulo cuál es el usuario activo en el sistema?

Creando módulos

Solución:

- Es necesario hacer una llamada a JAMES para inicializar el sistema.
  - Es algo parecido a lo que hacíamos con *session_start()* para inicializar la sesión.
- Esta llamada hará dos cosas:
  - Comprobar que realmente se puede cargar el módulo
  - Establecer el *contexto* de ejecución del módulo
    - Es decir, quién es el usuario activo, el grupo actual...

Creando módulos

Solución (código):

- La llamada consiste en crear un objeto de la clase: *HtmlInterface*

```
<?php
   include_once
("core/html/HtmlInterface.class.php");
   $html = new HtmlInterface("MiModulo");
?>
<html>
<head>
<title>Mi Modulo</title>
</head>
<body>
   ¡Hola mundo!
</body>
</html>
```

Creando módulos

¿Cómo sé cuál es el usuario activo?

- Usando una variable *de sistema*:
  - $GLOBALS[CURRENT_USER]
  - Esta variable devuelve un objeto de la clase *User:*
    - *Obtener el nombre*
      - *$GLOBALS[CURRENT_USER]->GetLogin();*
    - *Obtener el identificador*
      - *$GLOBALS[CURRENT_USER]->GetId();*
    - *Otras muchas funciones más...*

Creando módulos

Usuario activo (ejemplo):

```
<?php
   include_once("core/html/HtmlInterface.class.php");
   $html = new HtmlInterface("MiModulo");
?>
<html>
<head><title>Mi Modulo</title></head>
<body>
   Usuario activo:
   <?php
      echo "El login es: ".$GLOBALS[CURRENT_USER]->GetLogin();
      echo "<br>";
      echo "El id es: " . $GLOBALS[CURRENT_USER]->GetId();
      echo "<br>";
   ?>
</body>
</html>
```

Introducción

Al terminar hoy, sabremos:

- Obtener información sobre el grupo activo
- Qué poner en el fichero config.xml
- Añadir permisos al módulo y trabajar con ellos
- Crear claves de configuración y consultarlas
- Ver información sobre otros usuarios/grupos del sistema.
- Manejar la clase JError
- Cómo internacionalizar nuestros módulos

Creando módulos

¿Y sobre el grupo activo?

- Usando otra variable *de sistema*:
  - $GLOBALS[CURRENT_GROUP]
  - Que devuelve un objeto de la clase *Group*: (original,¿no?)
    - *Obtener el nombre*
      - *$GLOBALS[CURRENT_GROUP]->GetName();*
    - *Obtener el identificador*
      - *$GLOBALS[CURRENT_GROUP]->GetId();*
    - *Otras muchas funciones más...*

Creando módulos

Grupo activo (ejemplo):

```
<?php
   include_once("core/html/HtmlInterface.class.php");
   $html = new HtmlInterface("MiModulo");
?>
<html>
...
<body>
<?php
   echo "Grupo activo:<br>"
   echo "El nombre es: " . $GLOBALS[CURRENT_GROUP]->GetName();
   echo "<br>";
   echo "El id es: " . $GLOBALS[CURRENT_GROUP]->GetId();
   echo "<br>";
?>
</body>
</html>
```

Creando módulos

El *config.xml*

- Es un fichero XML que da información a James sobre el módulo: nombre, permisos, configuración, tablas...

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<module name="MiModulo">
   <screenname lang="en">My Module</screenname>
   <screenname lang="es">Mi Modulo</screenname>
</module>
```

Creando módulos

El *config.xml* (Explicación)

- La etiqueta <module>:
  - Es la etiqueta principal del XML
  - El nombre del módulo se indica en ella:
    <module name = “MiModulo”>
    - El nombre del módulo coincidirá con el del zip de instaalción y con el del directorio en que se instalará
- La etiqueta <screenname>:
  - Indica el nombre “amigable” del módulo.
  - Se puede especificar en distintos idiomas.

Creando módulos

Los permisos

- ¿Para qué sirven?
  - Nos van a permitir que usuarios distintos puedan hacer cosas distintas con el módulo. Ejemplos:
    - El administrador del sistema puede instalar módulos. Un usuario normal, no puede hacerlo.
    - Un usuario encargado de un foro podrá eliminar mensajes del mismo.
- Un rol es el conjunto de permisos que tiene un usuario.

Creando módulos

¿Cómo se si un usuario tiene un permiso?

- Usando el método *HasPermission* de la clase User
  - $GLOBALS[CURRENT_USER]->HasPermission (“permiso”);
  - **Importante:** En el nombre del permiso **importan** las mayúsculas y minúsculas.

```
...
<?php
   if ($GLOBALS[CURRENT_USER]->HasPermission(“miPermiso”))
      echo “El usuario tiene el permiso 'miPermiso'”;
   else
      echo “No tienes el permiso 'miPermiso'”;
?>
...
```

Creando módulos

Permisos. Un problema.

- Acabamos de ver el código para consultar un permiso, pero... ¿cómo le decimos al sistema que nuestro módulo tiene el permiso “miPermiso”?
  - Mediante el fichero de configuración del módulo: el *config.xml*
    - *Se hace a través de la etiqueta <permissions>*

Creando módulos

Permisos. Una solución.

- Añadimos la información necesaria a nuestro *config.xml*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<module name="MiModulo">
   <screenname lang="en">My Module</screenname>
   <screenname lang="es">Mi Modulo</screenname>
   <permissions>
      <permission name="miPermiso">
           <screenname lang="en">My permission</screenname>
           <screenname lang="es">Mi permiso</screenname>
           <description lang="es">
               Permiso de prueba para MiModulo
           </description>
      </permission>
   </permissions>
</module>
```

Creando módulos

Permisos. Otro problema.

- Bien, ya tenemos nuestro permiso en el *config.xml*, sin embargo, seguimos teniendo un problema:
  - *La información sobre los permisos se guarda en la base de datos y se crea cuando se instala el módulo.*
  - *Como hemos modificado el fichero de configuración después de instalar el módulo, la información del nuevo permiso no ha quedado registrada.*
  - *¿Qué hacemos?*

Creando módulos

Permisos. Otra solución

- Tenemos dos (o tres) soluciones:
  - Eliminar el módulo y volverlo a instalar:
    - Ventaja: Es fácil de hacer
    - Inconveniente: Se pierde la información de grupos en que estaba el módulo, permisos que tenían los usuarios, etc.
  - Modificar manualmente la base de datos:
    - Ventaja: No se pierde información sobre nada
    - Inconveniente: Hay que “toquetear” en la base de datos
  - Usar el módulo de ayuda al desarrollo de módulos
    - Ventaja: Fácil, rápido, no se pierde información de nada
    - Inconveniente: Aún está en desarrollo :-(

Creando módulos

La configuración

- ¿Para qué sirve?
  - Nos va a permitir que el sistema almacene para el módulo determinadas claves de configuración.
    - Estas claves de configuración pueden ser enteros, cadenas, listas de opciones, etc.
- El módulo sólo se tendrá que:
  - Indicar qué claves de configuración tiene.
  - Consultarlas en el código cuando sea necesario.

Creando módulos

Indicar las claves de configuración

- Añadimos la información necesaria a nuestro *config.xml*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<module name="MiModulo">
...
   <config>
      <configkey name="size" type="integer" level="system">
         <screenname lang="es">Tamaño</screenname>
         <description lang=”es”>
            Tamaño máximo asignado por usuario
         </description>
      </configkey>
   </config>
...
</module>
```

Creando módulos

El *config.xml* (Explicación)

- La etiqueta <config>:
  - Engloba a todas las claves de configuración del módulo.
- La etiqueta <configkey>:
  - Especifica una clave de configuración.
  - Atributos:
    - *name*: Nombre de la clave.
    - *type*: Puede ser: *integer*, *string*, *boolean* u *option*.
    - *level*: Nivel al que se configura: *system*, *group*, *user*.
- Las etiquetas <screenname> y <description>
  - Igual que en los permisos

Creando módulos

El *config.xml* (Explicación, 2ª parte)

- La etiqueta <options>:
  - Engloba a todas las opciones posibles para esa clave de configuración.
  - Sólo tiene sentido si el *type* de la clave es *option*
- La etiqueta <option>:
  - Especifica una opción concreta.
  - Atributo:
    - *name*: Nombre de la opción.

Creando módulos

Un ejemplo con *options*

```
<module name="MiModulo">
...
   <config>
      <configkey name="format" type="option" level="system">
         <screenname lang="es">Formato</screenname>
         <description lang=”es”>Formato de envío</description>
         <options>
            <option name=”text”>
                <screenname lang=”es”>Texto</screenname>
             </option>
            <option name=”html”>
                <screenname lang=”es”>HTML</screenname>
             </option>
      </configkey>
   </config>
...
</module>
```

Creando módulos

Consultar la configuración

- Usando el método *GetConfig* de la clase Module
  - $GLOBALS[CURRENT_MODULE]->GetConfig("size", defecto);
  - **Importante:** En el nombre de la clave de configuración **importan** las mayúsculas y minúsculas.
  - "defecto" es el valor por defecto que tendrá esa clave de configuración (por si no ha sido establecida).

```
...
<?php
   echo "El tamaño asignado es: ";
   echo $GLOBALS[CURRENT_MODULE]->GetConfig("size", 5000);
?>
...
```

Creando módulos

Otras cosillas sobre la configuración

- Se puede acceder a la configuración del núcleo con:
  - JSystem::GetConfig("nombreclave");
- La configuración se puede almacenar en varios sitios:
  - La configuración del sistema se almacena en ficheros de texto.
  - La configuración de cada usuario y los grupos, en la base de datos.

Creando módulos

¿Y otros grupos y usuarios?

- A través de $GLOBALS[JUGRSYSTEM]:
  - JUGRSYSTEM = James Users Groups Roles SYSTEM
  - *Ejemplos:*

```
// Obtener un usuario
$user = $GLOBALS[JUGRSYSTEM]->GetUser("admin");
$user->GetId();

// Obtener un grupo
$group = $GLOBALS[JUGRSYSTEM]->GetGroup(1);
$group->GetName();
```

Creando módulos

Otras funciones en JUGRSystem:

```
// Obtener todos los usuarios del sistema
$users = $GLOBALS[JUGRSYSTEM]->GetAllUsers();


// Obtener todos los grupos del sistema:
$groups = $GLOBALS[JUGRSYSTEM]->GetAllGroups();
```

Creando módulos

La clase JError

- En ocasiones, el núcleo puede devolver errores.
  - Por ejemplo, si pedimos información sobre un usuario que no existe en el sistema.
- Esos errores los devuelve como un objeto de la clase JError que contiene información sobre el error que ha ocurrido.
- ¿Cómo manejamos esos errores?

Creando módulos

Manejando errores

- Podemos saber si algo que nos devuelven ha sido un error o no usando:
  - JSystem::IsError($valorDevuelto);
- Podemos consultar información sobre el error usando los métodos de la clase JError: *GetCode, GetMessage, GetType, GetExtraInfo* y *ToString*.
- Podemos mostrar el error en pantalla con:
  - $html->PrintError($error);
    - Donde $html es un objeto de la clase *HtmlInterface*

Creando módulos

Ejemplo de manejo de errores:

```
<?php
   include_once("core/html/HtmlInterface.class.php");
   $html = new HtmlInterface("MiModulo");
?>
<html>
<head><title>Ejemplo de error</title></head>
<body>
<?php
   $usuario = $GLOBALS[JUGRSYSTEM]->GetUser(0);
   if (JSystem::IsError($usuario))
      $html->PrintError($usuario);
   else
      echo "El usuario es:".$usuario->GetLogin();
?>
</body>
</html>
```

Creando módulos

Tu también puedes devolver errores...

```
<?php
...
define(“CODIGO_ERROR”, 0);
...
function f($entrada)
{
   ...
   if ($entrada == 0)
      return JSystem::RaiseError(CODIGO_ERROR,
                                 “Entrada errónea”);
   ...
}
...
?>
```