Desarrollo Avanzado con Liferay 7

Victor Herrero Cazurro

Contenidos

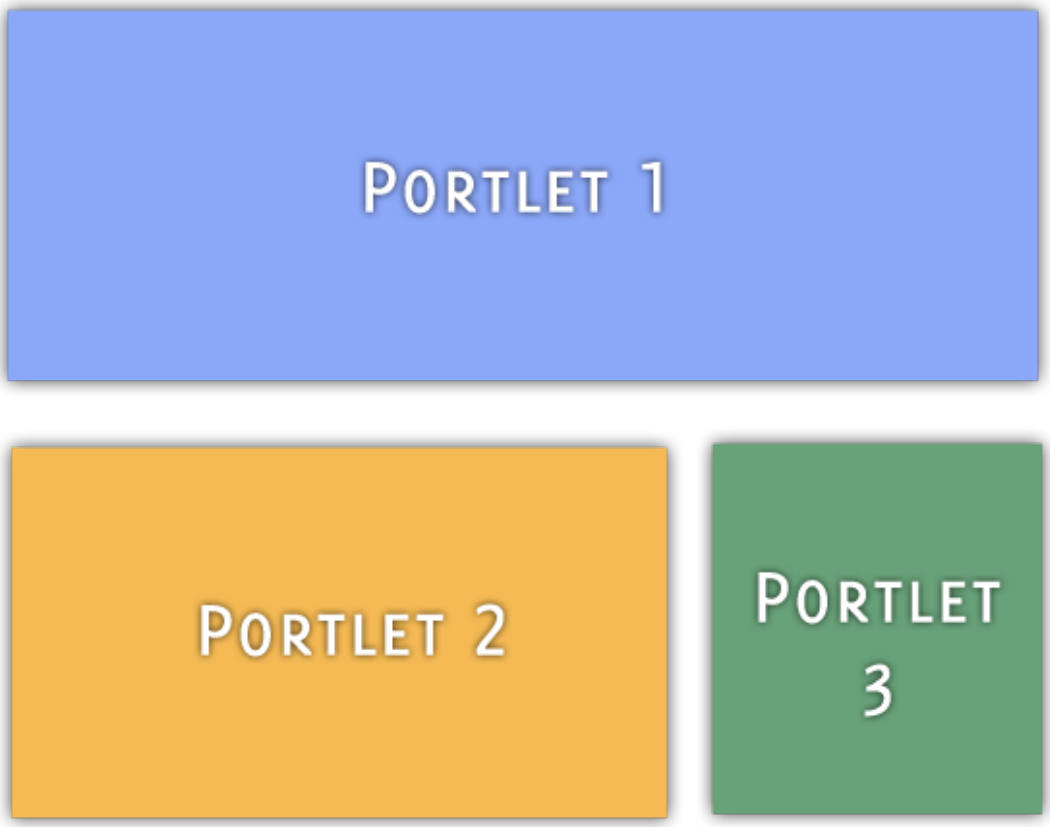
1. Portlets

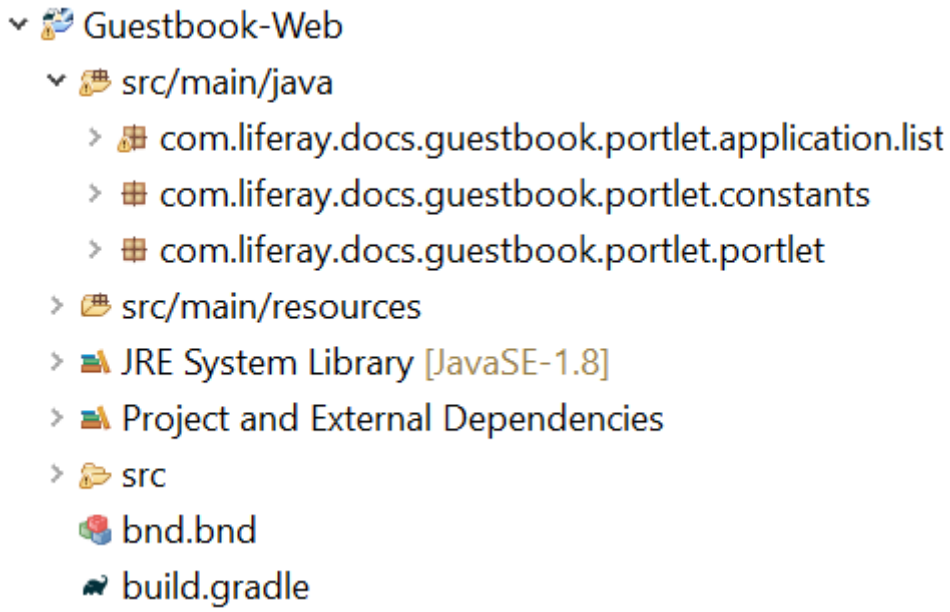
1.1. Introduccion

Portlet: Componente que define la capa de presentacion de los **Portales** (mini-aplicación). Se les asocia un **Modo de renderizacion** (VIEW, EDIT, HELP) y un **Estado** (Normal, Maximizado, Minimizado).

Portal: Aplicación web, que puede trabajar con Portlets, dispone de un **Contenedor de Portlets** y define sus páginas con Portlets.

Contenedor de Portlets: Entorno de ejecución para los Portlets, es una extensión del contenedor de Servlets que maneja el ciclo de vida y almacena las preferencias de los Portlets.

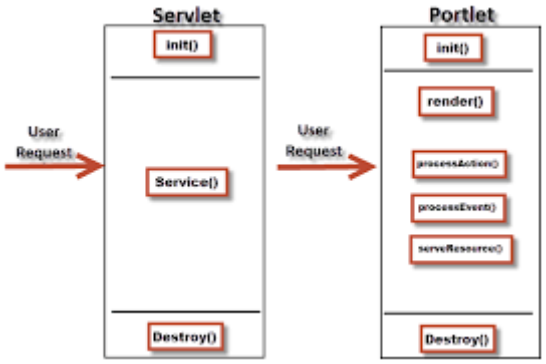


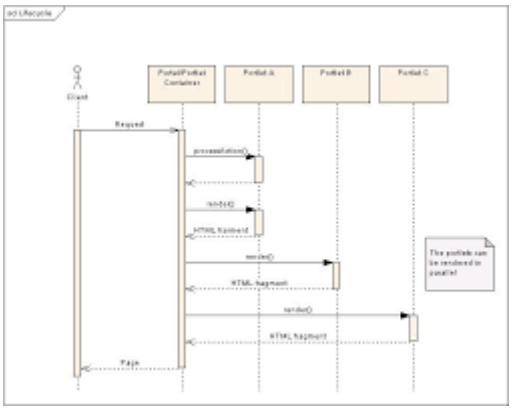


1.2. Ciclo de vida

El ciclo de vida de un Portlet, se divide en las siguientes fases

- Inicialización: Dado que los Portlets se instancian desde el contenedor, se permite inicializarlos con el método **init**
- Procesado de Acciones: Es la fase en la que el Portlet responde ante la interacción del usuario. Solo se ejecutará en el Portlet afectado.
- Procesado de Eventos: Es la fase en la que el Portlet responde ante un evento provocado por otro Portlet. Se ejecutará en todos los Portlets que hayan definido la escucha del evento lanzado.
- Renderizado: Fase en la que se realiza el pintado del Portlet. Se ejecutará en todos los Portlets de la página visualizada.
- Destrucción: Cuando el Portlet ya no es empleado.





1.3. Configuracion

Anteriormente la configuracion de un Portlet se realizaba en el fichero **Portlet.xml**, donde se podian configurar:

- El nombre de Portlet, que ha de ser único en el Portal.
- La clase que implementa la interface **javax.portlet.Portlet**, la más básica **GenericPortlet**.
- Los modos de render aceptados (VIEW, EDIT y HELP son los por defecto).
- Los estados de ventana aceptados (NORMAL, MAXIMIZED y MINIMIZED son los por defecto).

```
<portlet>
    <portlet-name>NOMBRE_UNICO</portlet-name>
    <display-name>MiPortlet</display-name>
    <portlet-class>com.ext.portlet.MiPortlet</portlet-class>
    <supports>
        <mime-type>text/html</mime-type>
        <portlet-mode>VIEW</portlet-mode>
        <portlet-mode>EDIT</portlet-mode>
        <portlet-mode>HELP</portlet-mode>
        <window-state>NORMAL</window-state>
        <window-state>MINIMIZED</window-state>
        <window-state>MAXIMIZED</window-state>
    </supports>
</portlet>
```

Pero en la version 7, al introducir OSGI, se ha trasladado esta configuracion a las propiedades del componente OSGI

```
@Component(
    immediate = true,
    property = {
        "com.liferay.portlet.display-category=category.sample",
        "com.liferay.portlet.instanceable=true",
        "javax.portlet.display-name=01_PortletMVC Portlet",
        "javax.portlet.init-param.template-path=/",
        "javax.portlet.init-param.view-template=/view.jsp",
        "javax.portlet.name=" + PortletPortletKeys.Portlet,
        "javax.portlet.resource-bundle=content.Language",
        "javax.portlet.security-role-ref=power-user,user"
    },
    service = Portlet.class
)
public class PortletPortlet extends MVCPortlet {}
```

1.4. Portlet desarrollado con el Estandar

1.4.1. Fases

El desarrollo de un Portlet estandar se basa en la herencia de la clase **GenericPortlet** y la implementacion de los siguientes métodos

- Para la fase de **Render**
  - ¬ doView()
  - ¬ doHelp()
  - ¬ doEdit()
  - ¬ O anotando un método con la misma firma (cambiando el nombre) que alguno de los anteriores con la anotación **RenderMode(name="VIEW")**

```
@RenderMode(name="CUSTOM")
public void doCustomRenderMode(RenderRequest renderRequest,
RenderResponse renderResponse) throws IOException, PortletException {}
```

- Para la fase de **Action**
  - ¬ processAction()
  - ¬ O anotando un método con la misma firma que el anterior con la anotación **@ProcessAction(name="ejecutar")**

```
@ProcessAction(name="saludarAction")
public void saludar(ActionRequest request, ActionResponse response)
throws PortletException, IOException {}
```

NOTE El acceso a las acciones se debe hacer vía POST (Method de HTTP).

- Para la fase de **Events**
  - ¬ processEvent()
  - ¬ O anotando un método con la misma firma que el anterior con **@ProcessEvent**

```
@ProcessEvent(qname = "{http://liferay.com}empinfo")
public void handleProcessempinfoEvent(javax.portlet.EventRequest
request, javax.portlet.EventResponse response) throws javax.portlet
.PortletException, java.io.IOException {}
```

- Para la fase de **Resource**
  - ¬ serveResource()

```
@Override
public void serveResource(ResourceRequest resourceRequest,
ResourceResponse resourceResponse) throws IOException, PortletException
{}
```

1.4.2. Render Mode

Permiten definir como se ha de pintar el Portlet, la vista a emplear. En el estándar se definen los siguientes modos

- VIEW
- EDIT
- HELP

Pudiéndose extender, practica habitual de las distintas implementaciones de Portales, como Liferay, donde se dispone de

- ABOUT
- CONFIG
- PRINT
- PREVIEW
- EDIT_DEFAULTS
- EDIT_GUEST

Para poder emplear estos modos, se ha de emplear el API de Portlets de Liferay.

Para activar los distintos modos, se ha de incluir en la configuracion de Portlet

```
<supports>
    <portlet-mode>VIEW</portlet-mode>
    <portlet-mode>EDIT</portlet-mode>
    <portlet-mode>CUSTOM</portlet-mode>
    <portlet-mode>ABOUT</portlet-mode>
    <portlet-mode>CONFIG</portlet-mode>
    <portlet-mode>PRINT</portlet-mode>
    <portlet-mode>PREVIEW</portlet-mode>
    <portlet-mode>EDIT_DEFAULTS</portlet-mode>
    <portlet-mode>EDIT_GUEST</portlet-mode>
</supports>
```

Se pueden definir nuevos, definiendo de forma global un nuevo **custom-portlet-mode**

```
<custom-portlet-mode>
    <description>Modo de renderizacion CUSTOM</description>
    <portlet-mode>CUSTOM</portlet-mode>
    <portal-managed>false</portal-managed>
</custom-portlet-mode>
```

Para acceder a un **Render Mode** de un Portlet concreto, se ha de invocar una URL particular, dado que los Portlet pueden estar en distintas paginas, e incluso haber mas de una instancia del mismo en la pagina, la generacion de estas URL, es dinamica, por lo que se precisa de ayuda para obtenerlas.

Desde una JSP se crearia empleando la etiqueta **portlet:renderURL**

```
<portlet:renderURL portletMode="CUSTOM" var="customRenderModeURL"/>
```

Y desde el codigo java, co el objeto **renderResponse**.

```
@Override
public void render(RenderRequest renderRequest, RenderResponse
renderResponse)
        throws IOException, PortletException {
    PortletURL customRenderModeURL = renderResponse.createRenderURL();
    customRenderModeURL.setPortletMode(new PortletMode("CUSTOM"));
}
```

1.4.3. View State

Permiten definir la forma de pintar la vista del portlet. En el estándar se definen los siguientes estados

- NORMAL
- MAXIMIZED
- MINIMIZED

Para activar los distintos estados, se ha de incluir en la configuracion de Portlet

```
<supports>
    <window-state>MAXIMIZED</window-state>
    <window-state>MINIMIZED</window-state>
    <window-state>NORMAL</window-state>
</supports>
```

Se pueden definir nuevos, definiendo de forma global un nuevo **custom-window-state**

```
<custom-window-state>
    <description>Nuevo Window STATE</description>
    <window-state>CUSTOM</window-state>
</custom-window-state>
```

Para acceder a esta caracteristica desde el Portlet, se dispone del método

request.getWindowState().

Para indicar este parametro en la definición de la URL en una JSP se haria

```
<portlet:renderURL windowState="MAXIMIZED" portletMode="VIEW"
var="viewRenderModeUrl"/>
```

Y desde el codigo java

```
@Override
public void render(RenderRequest renderRequest, RenderResponse
renderResponse)
        throws IOException, PortletException {
    PortletURL customRenderModeURL = renderResponse.createRenderURL();
    customRenderModeURL.setWindowState(WindowState.MAXIMIZED);
}
```

1.4.4. Init Param

Se pueden configurar parametros de inicio asociados al Portlet (constantes), habitualmente se emplean para definir las JSP que se emplearan como Vista.

Hasta la version 6.2, se hacia en el **portlet.xml**

```
<init-param>
    <name>init-view-template</name>
    <value>/html/view.jsp</value>
</init-param>
```

A partir de la version 7, con OSGI, se hace en la propia clase del Portlet

```
@Component(
    immediate = true,
    property = {
        "javax.portlet.init-param.view-template=/view.jsp"
    },
    service = Portlet.class
)
public class LiferayMVCPortlet extends MVCPortlet {
}
```

Para recogerlos en la implementacion.

```
String initView = getInitParameter("init-view-template");
```

1.4.5. Preferencias

Las preferencias de los Portlet, permiten definir unas variables persistentes asociadas al Portlet. Estas variables, pueden tener valores por defecto definidos en el portlet.xml

```
<portlet-preferences>
    <preference>
        <name>prefijo</name>
        <value>Hello </value>
    </preference>
</portlet-preferences>
```

Los parámetros a definir son

- name: Indica la clave de la preferencia.
- value: Indica el valor o valores por defecto (puede haber mas de uno).
- read-only: Indica que no puede ser escrito.
- preferences-validator: Indica un clase que implementa PreferencesValidator, que sirve para validar el valor asignado a la preferencia al llamar a store(), lanzando un ValidatorException si no cumple la validación.

```
public class SamplePreferencesValidator implements
PreferencesValidator{
    @Override
    public void validate(PortletPreferences portletPreferences) throws
ValidatorException {
        String miPreferencia = portletPreferences.getValue(
“MiPreferencia", "");
        List<String> valoresInvalidos = new ArrayList<String>();
        valoresInvalidos.add(miPreferencia);
        if(miPreferencia.equalsIgnoreCase("No Valido")){
            throw new ValidatorException("Se ha introducido un valor
invalido", valoresInvalidos);
        }
    }
}
```

Para recuperar las preferencias del portlet.xml, hay que ejecutar la siguiente sentencia

```
PortletPreferences preferences = request.getPreferences();
```

El acceso se tiene tanto desde RenderRequest, como de ActionRequest.

Para establecer un nuevo valor distinto al por defecto:

```
preferences.setValue("prefijo", prefijo);
preferences.store();
```

O para leer su valor

```
preferences.getValue("prefijo", null);
```

1.4.6. Internacionalizacion

Liferay define un gran numero de mensajes ya internacionalizados en el fichero **portal-impl.jar/src/content/Language.properties** que se emplean a lo largo del Portal, esa misma estructura es la que hay que seguir si se desea internazionalizar un nuevo Portlet, por lo tranto se ha de definir un fichero de properties para cada idioma, con un mismo nombre base, que ha de colocarse dentro del classpath y

declararse., tipicamente su nombre es **content/Language.properties**, pero puede ser otro.

Hasta la version 6.2 se hacia en el fichero **portlet.xml**.

```
<resource-bundle>com.my.portlets.Resource</resource-bundle>
```

Además se han de declarar los idiomas soportados por el portlet.

```
<supported-locale>es</supported-locale>
<supported-locale>en_GB</supported-locale>
<supported-locale>en_US</supported-locale>
```

A partir de la version 7, con OSGI, se hace en la propia clase del Portlet.

```
@Component(
    immediate = true,
    property = {
        "javax.portlet.resource-bundle=com.my.portlets.Resource"
    },
    service = Portlet.class
)
public class LiferayMVCPortlet extends MVCPortlet {
}
```

Se definirán los ficheros properties de cada idioma en la ruta indicada siguiendo el formato **<resource-bundle-name>_<language>_<country>.properties**.

El uso de las propiedades internacionalizadas, se realiza a partir del API de ResourceBoundle de Java, empleando en API de Liferay, en concreto la clase **LanguageUtil**

```
Locale locale = LanguageUtil.getLanguageId(request);
locale = request.getLocale();

ResourceBundle res = ResourceBundle.getBundle("Language", locale);

res.getString("message-key");
```

O haciendo uso del API de portlets.

```
ResourceBundle res = getResourceBundle(locale);
res = getPortletConfig().getResourceBundle(locale);

res.getString("message-key")
```

Si el mensaje es parametrizado

```
String mensaje = MessageFormat.format(resourceBundle.getString("saludo
"), "Juan");
```

1.5. Acceso a mensajes internacionalizados desde UI

Para emplear los mensajes internacionalizados, se pueden utilizar varias vias

- Libreria de etiquetas de Liferay **liferay-ui**

```
<%@ taglib uri="http://liferay.com/tld/ui" prefix="liferay-ui" %>

<liferay-ui:message key="titulo" />

<%Object[] argumentos = new Object[]{"Juan"}; %>

<liferay-ui:message key="saludo" arguments="<%=argumentos%>" />

<liferay-ui:error key="MiErrorEnProperties"/>
```

Para el anterior ejemplo, deberán existir en el properties los siguientes pares clave-valor

```
titulo=Esto es español
saludo= Hola {0}
```

Y se tendrá que haber añadido al objeto **SessionErrors** un error con clave **MiErrorEnProperties**

```
SessionErrors.add(request, "MiErrorEnProperties", "SE ha producido un
error en el Action");
```

- Libreria de etiquetas JSTL de formateo **fmt**

```
<fmt:message
bundle="<%=portletConfig.getResourceBundle(request.getLocale())%>"
key="saludo">
    <fmt:param value="Antonio"></fmt:param>
</fmt:message>
```

1.5.1. Eventos

Permite la comunicación entre Portlets, a base de eventos, introduciendo un nuevo componente en el ciclo de vida de los Portlets, similar al **ProcessAction**, denominado **ProcessEvent**, que se coloca en el flujo entre el **ProcessAction** y el **Render**.

Se ha de declarar en el Portlet emisor

Para versiones anteriores a 6.2

```
<portlet>
    <supported-publishing-event xmlns:x='http://liferay.com'>
        <qname>x:miEvento</qname>
    </supported-publishing-event>>
</portlet>
<event-definition xmlns:x='http://liferay.com'>
    <qname>x:miEvento</qname>
    <value-type>java.lang.String</value-type>
</event-definition>
```

Para versiones superiores a 7

```
@Component(
    immediate = true,
    property = {
        "javax.portlet.supported-publishing-
event=miEvento;http://liferay.com"
    },
    service = Portlet.class
)
public class EmisorMVCPortlet extends MVCPortlet {}
```

Y en el receptor

Para versiones anteriores a 6.2

```
 <portlet>
    <supported-processing-event xmlns:x='http://liferay.com'>
        <qname>x:miEvento</qname>
    </supported-processing-event>
</portlet>
<event-definition xmlns:x='http://liferay.com'>
    <qname>x:miEvento</qname>
    <value-type>java.lang.String</value-type>
</event-definition>
```

Para versiones superiores a 7

```
@Component(
    immediate = true,
    property = {
        "javax.portlet.supported-processing-
event=miEvento;https://liferay.com"
    },
    service = Portlet.class
)
public class ReceptorMVCPortlet extends MVCPortlet{}
```

NOTE	Para definiciones con OSGI, el formato de la declaración de los eventos es **javax.portlet.supported-processing-event=<String>;<QName>**
NOTE	El tipo de objeto a compartir, ha de ser Serializable.

Para provocar el evento

```
javax.xml.namespace.QName qName = new QName("http://liferay.com",
"miEvento", "x");
response.setEvent(qName, "Dato a enviar");
```

La escucha del evento

```
@javax.portlet.ProcessEvent(qname = "{http://liferay.com}miEvento")
public void handleProcessempinfoEvent(javax.portlet.EventRequest
request,
        javax.portlet.EventResponse response)
        throws javax.portlet.PortletException, java.io.IOException {

    javax.portlet.Event event = request.getEvent();
    String value = (String) event.getValue();
    System.out.print("Dato recibido en el evento" + value);
    response.setRenderParameter("miEvento", value);
}
```

1.5.2. Eventos Javascript

La comunicacion se puede establecer tambien en el lado del cliente a traves de eventos de javascipt

Lanzando el evento como sigue

```
function _fireIPCEvent(selectedLabel){
    Liferay.fire('miEvento',{
        param1: selectedLabel
    });
}
```

Y escuchando eventos del mismo tipo

```
Liferay.on('miEvento',function(event) {
    var parameterValue1 = event.param1;
    console.log("received value is "+ parameterValue1);
});
```

1.5.3. Libreria de Etiquetas Estandar

El estandar define una libreria de etiquetas para ayudar a generar los JSP, para emplearla se ha de declarar su uso en las cabeceras del JSP

```
<%@ taglib uri="http://java.sun.com/portlet_2_0" prefix="portlet" %>
```

Ofrece las siguientes etiquetas

- **defineObjects**: define las siguientes variables en el ambito de la JSP:
  - ¬ request
  - ¬ response
  - ¬ portletConfig
  - ¬ portletSession
  - ¬ portletSessionScope (atributos de sesión)
  - ¬ portletPreferences
  - ¬ portletPreferencesValues (map de preferencias)

```
<portlet:defineObjects />
```

- **namespace**: Incluye un literal en la vista renderizada, que permite distinguir los componentes estáticos de la vista (formularios, javascript y demás elementos de HTML), entre las distintas instancias de Portlets existentes en una página.

```
<portlet:namespace/>
```

- **actionURL**: Permite definir una url hacia un metodo de accion asociado al portlet.

```
<portlet:actionURL name="accion" portletMode="VIEW" var="actionUrl"/>
```

- **renderURL**: Permite definir una URL hacia un metodo de render asociado al portlet.

```
<portlet:renderURL portletMode="EDIT" var="editRenderModeUrl"/>
```

- **param**: Para definir un parámetro en una URL.

```
<portlet:renderURL var="editGreetingURL">
    <portlet:param name="mvcPath" value="/edit.jsp" />
</portlet:renderURL>
```

- **resourceURL**: Permite definir una URL hacia un metodo de resource asociado al portlet.

```
<portlet:resourceURL var="resourceUrl"/>
```

2. Portlets Liferay MVC

API propio de Liferay, que pretende minimiza el codigo a generar en las Clases de Portlets, extraiendo la logica de renderizado de ellos (Vista), dejando unicamente la logica de accion (Controlador).

```
@Component(
    immediate = true,
    property = {
        "com.liferay.portlet.display-category=category.sample",
        "com.liferay.portlet.instanceable=true",
        "javax.portlet.display-name=task-web Portlet",
        "javax.portlet.init-param.template-path=/",
        "javax.portlet.init-param.view-template=/view.jsp",
        "javax.portlet.name=" + TaskPortletKeys.Task,
        "javax.portlet.resource-bundle=content.Language",
        "javax.portlet.security-role-ref=power-user,user"
    },
    service = Portlet.class
)
public class TaskPortlet extends MVCPortlet {

    public void addTask(ActionRequest request, ActionResponse response)
        throws PortalException, SystemException {}
}
```

Se basa en definir con una propiedad **javax.portlet.init-param.view-template**, la JSP que se encarga de gestionar el pintado de los distintos modos de renderizado.

Y la definicion unicamente de los métodos de accion dentro de la propia clase, donde el nombre de la accion será en nombre del método y la firma será igual a la del método **processAction** del estandar.

NOTE

Si se desea cambiar la JSP que se emplea en la fase de render, se puede establecer el siguiente **renderParam**

```
actionResponse.setRenderParameter("mvcPath", "/error.jsp
");
```

A partir de la verion 7, la configuracion del Portlet, se realiza con cabeceras de OSGI en la propia clase, y no en el fichero **portlet.xml**, aquí se dispone un listado con la relación de propiedades.

2.1. Envio de Parametros por Request

Para recibir parametros que se envien en la request

```
<portlet:actionURL name="addGuestbook" var="addGuestbookURL">
    <portlet:param name="guestbookId"
            value="<%= String.valueOf(entry.getGuestbookId()) %>" />
</portlet:actionURL>
```

Se hará uso de la clase de utilidad **ParamUtil**

```
long guestbookId = ParamUtil.getLong(actionRequest, "guestbookId");
```

2.2. Envio de Atributos a JSPs

Si se desea enviar informacion entre la fase de render y response, se pasaran como atributos

En el Action

```
renderRequest.setAttribute("guestbookId", guestbookId);
```

En el Render o en la JSP

```
long guestbookId = Long.valueOf((Long) renderRequest.getAttribute(
"guestbookId"));
```

2.3. MVC Command

Permite separar en clases, los distintos bloques que componen la logica de control asociada a un Portlet.

Se permite separa en tres tipos:

- **MVCActionCommand**: Funcionalidades invocadas con **actionURL**.
- **MVCRenderCommand**: Funcionalidades invocadas con **renderURL**.
- **MVCResourceCommand**: Funcionalidades invocadas con **resourceURL**.

Será importante el **javax.portlet.name** asignado al Portlet, dado que este es el que se usa para referenciar los **MVCCommand** y el **Portlet**.

```
@Component(
    immediate = true,
    property = {
        "javax.portlet.display-name=Hello World",
        "javax.portlet.name=" + HelloWorldPortletKeys.HELLO_WORLD,
        "javax.portlet.resource-bundle=content.Language",
        "javax.portlet.security-role-ref=guest,power-user,user",
        "javax.portlet.supports.mime-type=text/html"
    },
    service = Portlet.class
)
public class HelloWorldPortlet extends MVCPortlet {
}
```

2.3.1. MVC Action Command

La idea es que una definicion de URL como la siguiente, no sea prcesada por la clase de Portlet, sino por una que solo se encargue de esta logica.

```
<portlet:actionURL name="/helloWorld" var="helloWorldURL" />
```

La definición de esta clase seria

```
@Component(
    immediate = true,
    property = {
        "javax.portlet.name=" + HelloWorldPortletKeys.HELLO_WORLD,
        "mvc.command.name=/helloWorld"
    },
    service = MVCActionCommand.class
)

public class HelloWorldMVCActionCommand extends BaseMVCActionCommand {
    // implement your action
}
```

Es importante la propiedad **javax.portlet.name**, que debe coincidir con el nombre del portlet al que se asocia este procesado de acción.

Se pueden definir mas de un **javax.portlet.name**

```
@Component(
    immediate = true,
    property = {
        "javax.portlet.name=" + BlogsPortletKeys.BLOGS,
        "javax.portlet.name=" + BlogsPortletKeys.BLOGS_ADMIN,
        "javax.portlet.name=" + BlogsPortletKeys.BLOGS_AGGREGATOR,
        "mvc.command.name=/blogs/edit_entry"
    },
    service = MVCActionCommand.class
)
public class EditEntryMVCActionCommand extends BaseMVCActionCommand {}
```

2.3.2. MVC render Command

Este API permite definir URL de Render mas especificas que empleando **Render Mode**, par alo cual habilita un nuevo parametro **mvcRenderCommandName**

```
<portlet:renderURL var="editEntryURL">
    <portlet:param name="mvcRenderCommandName"
value="/hello/edit_entry" />
    <portlet:param name="entryId" value="<%=
String.valueOf(entry.getEntryId()) %>" />
</portlet:renderURL>
```

```
@Component(
    immediate = true,
    property = {
       "javax.portlet.name=" + HelloWorldPortletKeys.HELLO_WORLD,
       "mvc.command.name=/hello/edit_entry"
    },
    service = MVCRenderCommand.class
)
public class EditEntryMVCRenderCommand implements MVCRenderCommand {
    //. . .
}
```

Funciona de forma similar al anterior caso, en este además de poder poner varios **javax.portlet.name**, se pueden definir varios **mvc.command.name**

```
@Component(
    immediate = true,
    property = {
       "javax.portlet.name=" + HelloWorldPortletKeys.HELLO_MY_WORLD,
       "javax.portlet.name=" + HelloWorldPortletKeys.HELLO_WORLD,
       "mvc.command.name=/hello/edit_super_entry",
       "mvc.command.name=/hello/edit_entry"
    },
    service = MVCRenderCommand.class
)
```

2.3.3. MVC render Command

Igual que los anteriores, pero para la fase de recurso.

```
<portlet:resourceURL id="/login/captcha" var="captchaURL" />
```

```
@Component(
    property = {
        "javax.portlet.name=" + LoginPortletKeys.FAST_LOGIN,
        "javax.portlet.name=" + LoginPortletKeys.LOGIN,
        "mvc.command.name=/login/captcha"
    },
    service = MVCResourceCommand.class
)
public class CaptchaMVCResourceCommand implements MVCResourceCommand {

    @Override
    public boolean serveResource(
        ResourceRequest resourceRequest, ResourceResponse
resourceResponse) {

        try {
            CaptchaUtil.serveImage(resourceRequest, resourceResponse);

            return false;
        }
        catch (Exception e) {
            _log.error(e, e);

            return true;
        }
    }

    private static final Log _log = LogFactoryUtil.getLog(
        CaptchaMVCResourceCommand.class);
}
```

2.3.4. Sobreescritura de MVC Command

Se pueden sobreescribir los **MVCCommand** empleados en el portal sin mas que definir un componente con la misma cabecera, pero dandole mas prioridad con la propiedad **service.ranking**, por defecto los servicios se publican con valor **0**, por lo que cualquier valor superior seria suficiente, pero para dejar margen, se suelen emplean valores como **100**

```
@Component(
    immediate = true,
    property = {
        "javax.portlet.name=" + BlogsPortletKeys.BLOGS_ADMIN,
        "mvc.command.name=/blogs/edit_entry",
        "service.ranking:Integer=100"
    },
    service = MVCActionCommand.class
)
public class CustomBlogsMVCActionCommand extends BaseMVCActionCommand
{}
```

En estos casos es muy habitual querer seguir haciendo lo que la implementación original hacia, para lo cual se necesita mantener la referencia al servicio original que se está tapando, para lo cual se necesita conocer el nombre del componente.

```
@Reference(
    target =
"(component.name=com.liferay.blogs.web.internal.portlet.action.EditEntr
yMVCActionCommand)")
protected MVCActionCommand mvcActionCommand;
```

Se necesitará conocer:

- El nombre del Portlet, para lo cual se usara la clase correspondiente de constantes con nombre **<>PortletKeys**, que se encontrará tipicamente en el fichero LPKG con sufijo API, por ejemplo **Liferay CE Foundation - Liferay CE Users Admin - API.lpkg**, que dentro tendra´un jar con tambien sufijo API **com.liferay.users.admin.api-3.0.1.jar**, donde se encuentra la clase **com.liferay.users.admin.constants.UsersAdminPortletKeys**
- La URL del Command, para lo cual se debé acceder a la implementacion que se esta sobreescribiendo, la cual se encontrará en el fichero LPKG con sufijo Impl, por ejemplo **Liferay CE Foundation - Liferay CE Users Admin - Impl.lpkg**, que tendrá dentro un fichero con sufiijo **web**, por ejemplo **com.liferay.users.admin.web-3.0.6.jar** y que dentro tendra la clase a suplantar, por ejemplo **com.liferay.users.admin.web.internal.portlet.action.EditUserMVCActionCommand**

3. Sobreescritura de JSPs

Las JSPs ahora se encuentran organizadas en sus respectivos modulos de OSGI, estos modulos se encuentran en ficheros lpkg en la carpeta **<Liferay Poral Directory>\osgi**, por ejemplo en la carpeta **<Liferay Poral Directory>\osgi\marketplace**, se pueden encontrar los siguientes ficheros

- **Liferay CE Collaboration.lpkg**
- **Liferay CE Forms and Workflow.lpkg**
- **Liferay CE Foundation.lpkg**
- **Liferay CE IP Geocoder.lpkg**
- **Liferay CE Static.lpkg**
- **Liferay CE Sync Connector.lpkg**
- **Liferay CE Web Experience.lpkg**
- **Liferay Marketplace.lpkg**

El que compone el core de Liferay es **Liferay CE Foundation.lpkg**, en el se pueden encontrar los **jar** que componen el Core de liferay, los cuales estan modularizados, siendo los **web** los que contienen las **jsp**, asi por ejemplo las jsps de la gestion de usuarios esta dentro de **com.liferay.users.admin.web-2.3.1.jar**, mas concreatamente dentro de la carpeta **META-INF/resources**

Para sobreescribir una JSP, basta con crear un proyecto de tipo **Liferay Module Fragment Project**, en este tipo de proyecto, hay que hacer rederencia a un **Bundle** (instalacion de Liferay) definida en el ambito de eclipse, por que es de esa ubicación de donde sacará los ficheros originales a sobrescribir.

El siguiente paso es seleccionar el **OSGI BUndle**, es decir, el **jar** que contiene los ficheros a sobreescribir, para los JSPs se emplearán los jar con sufijo **web**.

Una vez seleccionado el **jar**, se puede seleccionar el fichero a sobreesrbir entre aquellos que están en la carpeta **META-INF/resources** del **jar**.

4. Service Builder

4.1. Introduccion

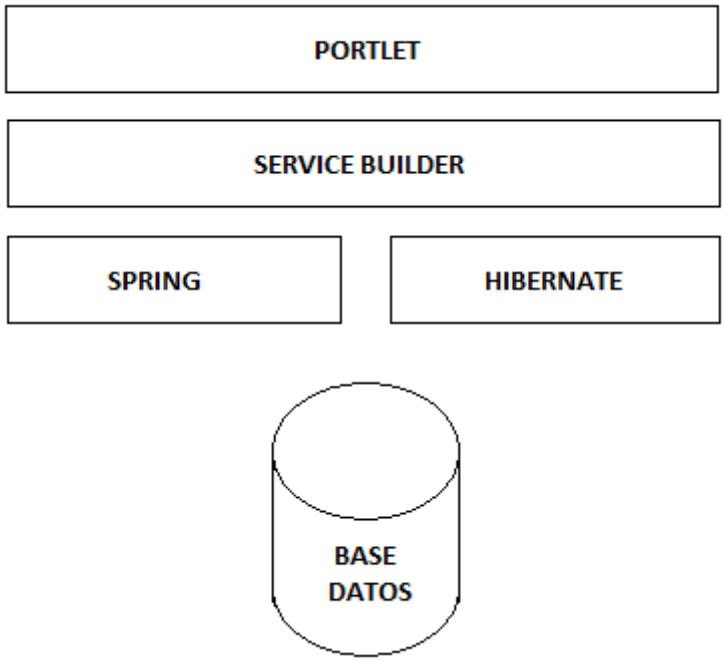
Permite definir una capa de Persistencia y de Servicio de forma automatica, considerando las funcionalidades mas basicas.

La idea de Service Builder, es homogeneizar la capa de servicios y de persistencia de las aplicaciones de Liferay, para ello proporciona un API y una herramienta (dentro del SDK de Liferay) que permite la creación de las clases de estas capas de forma automática siguiendo el patrón model-driven.

El propio Liferay esta creado empleando esta herramienta y la arquitectura que propone.

La herramienta, creará la capa de modelo y la de servicio de forma independiente, incluyendo las funcionalidades básicas de CRUD, permitiendo expandir dichas funcionalidades definiendo relaciones, Finders, Custom SQL, . . .

Esta tecnología, emplea por debajo **Hibernate** y **Spring**, aunque en principio no se tiene porque tocar las configuraciones de estos frameworks, siendo su uso transparente para el desarrollador de **service-builder**.



Hay un tipo de proyecto propio para Service Builder **service-builder**.

```
> blade create -t service-builder -p com.liferay.task.service -c Task
task
```

Se creará la siguiente estructura de proyectos

- Proyecto padre **task**
- SubProyecto **task-api**: Contendrá las interfaces de los servicios y las clases base. No será modificable, ya que se autogenera.
- SubProyecto **task-service**: Contendrá la implementacion del servicio.

4.2. Entidades

Se definen las entidades en el fichero **service.xml** del proyecto **task-service**. Que empleará el siguiente esquema

```
<!DOCTYPE service-builder PUBLIC "-//Liferay//DTD Service Builder
7.0.0//EN" "http://www.liferay.com/dtd/liferay-service-
builder_7_0_0.dtd">
```

Una vez definidas las entidades, se ha de lanzar la tarea **buildService** de Gradle sobre el proyecto principal, para que se autogeneren las clases necesarias.

```
> blade gw buildService
```

Las clases **-Impl** generadas, tanto para **Model**, **Service** y **Persistence**, son modificables, pudiendo cambiar los comportamientos definidos por la herramienta de generacion de código.

Cualquier modificación sobre estas clases, que añada un nuevo método o cambie la firma de uno existente, exigirá que se vuelva a lanzar la tarea de Maven **build-service** o para los proyectos gradle, la tarea gradle **buildService**

4.3. Service.xml

El nodo principal de este fichero será **<service-builder>**, que tendrá como atributos

- **package-path**: Paquete java donde se encontrarán las entidades definidas.
- **auto-namespace-tables**: Crear el namespace de forma automática para las tablas.

Y como nodos hijos

- **author**: Autor del modelo.
- **namespace**: Prefijo que se incluye en el nombre de la tabla con el formato.
- **exceptions**: Excepciones propias que se pueden lanzar al trabajar con el modelo.
- **service-builder-import**: Referencia a otro fichero que defina entidades, para partir en varios ficheros la definición de la capa de servicios.
- **entity**: Etiqueta principal, que permite definir las entidades que define el modelo.

El nodo **<entity>** tiene como atributos

- **name**: Nombre de la entidad.
- **uuid**: (booleano) Indica si se ha de generar este valor de forma automática.
- **local-service**: Si se permite acceder con la interface local.
- **remote-service**: Si se permite acceder con la interface remota, creando el servicio web.
- **table**: Nombre de la tabla a emplear para diferenciarla de la clase que representa la entidad.
- **cache-enabled**: Permite activar el cacheo de las entidades.
- **deprecated**: Si se considera la entidad en desuso.
- **human-name**: Nombre legible de la entidad empleado a la hora de generar la documentación.
- **json-enabled**: Si se permite el acceso a través del servicio web con JSON.
- **persistence-class**: Nombre de la clase creada hija de XXPersistenceImpl.
- **session-factory**: Bean de Spring que gestiona las sesiones de hibernate.
- **data-source**: Bean de Spring que gestiona las conexiones a la base de datos.
- **tx-manager**: Bean de Spring que gestiona las transacciones.
- **trash-enabled**: Si se habilita la posibilidad de envar a la papelera estas entidades.
- **uuid-accessor**: (booleano) indica si se genera el Getter para el uuid

Y como nodos hijos

- **column**: Definición de un campo de la entidad.

- **finder**: Nueva funcionalidad de búsqueda relacionada con la entidad, dará lugar a unos nuevos métodos en la clase de implementación de la persistencia XXPersistenceImpl.
- **order**: Establece como se ordenan los registros retornados por las consultas por defecto.
- **reference**: Indica referencias a servicios ya existentes en Liferay o definidos en otro service.xml pero que carguen en el mismo classpath, para que estén accesibles directamente en la clase de implementación de servicio.
- **tx-required**: Patrón de nombre para todos aquellos métodos que necesiten transacciones, por defecto los siguientes patrones ya están incluidos*: add*, check*, clear*, delete*, set*, and update*, el resto serán de solo lectura.

El nodo **<column>**, puede definir relaciones con otras entidades del portal, así como unos campos particulares de la arquitectura de Liferay

Portal and site scope columns

Name	Type	Primary
companyId	long	no
groupId	long	no

User column

Name	Type	Primary
userId	long	no

Audit columns

Name	Type	Primary
userId	long	no
createDate	Date	no
modifiedDate	Date	no

Los campos

- **companyId**: hace referencia a la instancia del Portal
- **groupId**: hace referencia al Sitio

La referencia a estos campos permite que los distintos Sitios e instancias de Portal tengan sus propios datos de estas tipologías.

El nodo column tiene como atributos

- **name**: Nombre del campo empleado en los Get y Set.
- **type**: Tipo Java del campo.
- **primary**: Si es clave primaria, pueden existir varias columnas formado la clave primaria.
- **entity**: Cuando el type es Collection, indica el tipo de objetos en la colección.
- **accessor**: (booleano) Si se accede por Get y Set o por propiedad.
- **convert-null**: (booleano) Convertir a null al insertar en BD.
- **db-name**: Nombre del campo en BD.
- **filter-primary**: (booleano) Permite indicar una clave primaria para los finder, sino se indica será la clave primaria por defecto.
- **id-type**: Se emplea para definir el método de generación de la clave primaria, puede ser*: class, increment, sequence o identity.
- **id-param**: Se necesita para definir la clase o la secuencia que genera la clave primaria en el tipo de generación class o sequence. En caso de emplear la secuencia, esta ha de definirse en el fichero
- **json-enabled**: (booleano) Si está habilitada la conversión a JSON
- **lazy**: (booleano) Si la carga es perezosa.
- **localized**: (booleano) Si acepta traducciones.
- **mapping-table**: Indica el nombre de la tabla intermedia en relaciones m-n.

El nodo **<order>** tiene como atributos

- **by**: Indica si la ordenación es asc o desc.

El nodo **<order>** como sub-nodos puede tener

- **order-column**: Que indica las columnas que participan en la ordenación, pudiendo haber una o más.

El nodo **<order-column>** tiene como atributos

- **name**: Nombre de la columna.
- **case-sensitive**: (boolean) Si la ordenación contempla mayúsculas y minúsculas.
- **order-by**: En lugar del atributo by del nodo order, permite indicar un tipo de

ordenación distinto para cada columna.

4.4. Relaciones

Se pueden definir dos tipos de relaciones en Service Builder, la mas basica es añadir una columna a una entidad que mantiene la relacion que refleje la FK con la otra entidad a través del PK de esta.

```
<entity name="Producto">
    <column name="productoId" type="long" primary="true" />
</entity>
<entity name="Subasta">
    <column name="subastaId" type="long" primary="true" />
    <column name="productoId" type="long"/>
</entity>
```

Con esto se consigue incluir un campo en la entidad **Subasta** que está destinado a mantener la relación, aunque no se cree la FK.

Si se desea obtener el objeto **Producto** asociado a la entidad **Subasta**, habrá que definir el método **getProducto** en la clase **SubastaImpl**, con la siguiente implementación.

```
public Producto getProducto() throws PortalException, SystemException{
    return ProductoLocalServiceUtil.getProducto(getProductoId());
}
```

La otra es definir una relación n-m con Liferay, se ha de definir dos nuevas columnas en cada una de las entidades que reflejarán la relacion, y en esas columnas definir:

- El atributo **mapping-table** en las dos entidades la relacion, haciendo referencia a la tabla que mantiene la relacion.
- El atributo **type** que ha de ser **Collection**.
- El atributo **entity** que define el tipo de la entidad con la que se asocia. Si la entidad está en otro paquete, se ha de definir el **nombre canonico**

```
<entity name="Asignatura">
    <column name="asignaturaId" type="long" primary="true" />
    <column name="alumnos" type="Collection" entity="Alumno" mapping-
table="Asignaturas_Alumnos"/>
</entity>
<entity name="Alumno">
    <column name="alumnoId" type="long" primary="true" />
    <column name="asignaturas" type="Collection" entity="Asignatura"
mapping-table="Asignaturas_Alumnos"/>
</entity>
```

4.5. Finders

Métodos que se generan en la capa de persistencia, que definen consultas sencillas, con clausulas de **where**.

Se emplea el nodo **finder**.

```
<finder name="producto" return-type="Collection">
    <finder-column name="producto" />
</finder>
```

El nodo **finder-column** define los campos por los que se realiza la búsqueda, definiendo los atributos que recibirán los métodos generados, si se aplican múltiples **finder-column** al **finder**, se aplica por defecto la cláusula **AND**.

El nodo **finder** tiene los siguientes atributos

- **db-index**: (boolean) Si es true, se genera automáticamente un índice de BD.
- **name**: Nombre del método generado en la capa de persistencia.
- **return-type**: Tipo retornado, puede ser Collection o una Entidad.
- **unique**: Indica que se retorna solo una entidad.
- **where**: Permite definir una clausula de where estatica, que no se ve afectada por los parametros de la consulta **id != 1**

El nodo **finder-column** tiene como atributos:

- **arrayable-operator**: Permite definir el valor AND o OR, y generara un nuevo método de finder, que aceptará un array por parámetro y creara una clausula

concatenando todos los valores del array con AND o OR.

- **case-sensitive**: Solo si la columna es de tipo String.
- **comparator**: Indica el tipo de comparación implementada por el método finder, puede tomar como valores: =, !=, <, ¬, >, >=, o LIKE.
- **name**: Nombre de la columna.

4.6. Custom SQL

Permiten realizar consultas SQL nativas, por lo que se pueden lanzar consultas mas complejas que con los **finder**.

Se tendrán que definir las consultas en un fichero **src/main/resources/META-INF/custom-sql/default.xml** siguiendo la siguiente estructura

```
<custom-sql>
    <sql id="[nombre canonico de la clase + metodo]">
        Consulta envuelta con <![CDATA[...]]>
        No terminar con punto y coma
    </sql>
</custom-sql>
```

Donde las consultas por convenio tendran como **id**, el nombre de la clase **Finder** mas el nombre del método que ejecutará la consulta, los cuales habrá que definir.

Ejemplo de definicion de consulta

```
<?xml version="1.0" encoding="UTF-8"?>
<custom-sql>
    <sql id=
"com.liferay.docs.guestbook.service.persistence.GuestbookFinder.buscarP
orNombre">
        <![CDATA[
                SELECT GB_Guestbook.*
                FROM GB_Guestbook
                WHERE
                    GB_Guestbook.name LIKE ?
        ]]>
    </sql>
</custom-sql>
```

NOTE	Ojo con el identificador, si se emplea el nombre de la clase, que en la implementacion del **Finder**, no llevará **Impl**.

Para el caso anterior, se debería crear la clase **GuestbookFinderImpl** y dentro el método **buscarPorNombre**, que en este caso retornará un **List<Guestbook>** y recibira un **String name**.

En un principio hay que hacer extender esta clase de **BasePersistenceImpl<>** e implementar la logica del método, para lo cual el API proporciona los siguientes servicios.

- **CustomSQLUtil** ¬ Permite obtener la consulta definida en el fichero xml.

```
String sql = CustomSQLUtil.get(GuestbookFinder.class.getName() +
".buscarPorNombre");
```

- **QueryPos** ¬ Permite sustituir las **?** por lo valores concretos.

```
SQLQuery q = session.createSQLQuery(sql);
QueryPos qPos = QueryPos.getInstance(q);
qPos.add(name);
```

- **QueryUtil** ¬ Permite realizar la ejecucion de una sentencia SQLQuery indicando paginación

```
(List<Guestbook>) QueryUtil.list(q, getDialect(), begin, end);
```

Una vez implementado el método se ha de lanzar la tarea de **build-service**, para que genere una nueva interface, en este caso **GuestbookFinder**, que habrá que implementar.

```
public class GuestbookFinderImpl extends BasePersistenceImpl<Guestbook>
implements GuestbookFinder {

    public static final String FIND_BY_NOMBRE = GuestbookFinder.class
.getName()
            + ".buscarPorNombre";

    public List<Guestbook> buscarPorNombre(String name, int begin, int
end) {

        Session session = null;
        try {
            session = openSession();

            String sql = CustomSQLUtil.get(FIND_BY_NOMBRE);

            SQLQuery q = session.createSQLQuery(sql);
            q.setCacheable(false);
            q.addEntity("GB_Guestbook", GuestbookImpl.class);

            QueryPos qPos = QueryPos.getInstance(q);
            qPos.add(name);

            return (List<Guestbook>) QueryUtil.list(q, getDialect(),
begin, end);
        } catch (Exception e) {
            try {
                throw new SystemException(e);
            } catch (SystemException se) {
                se.printStackTrace();
            }
        } finally {
            closeSession(session);
        }

        return null;
    }
}
```

Una vez generada la clase que implementa la consulta, el siguiente paso es consumirlo desde el servicio, para ello se ha de crear el método pertinente en **-ServiceImpl**, que tendra acceso a una instancia del **Finder**.

```
public List<Guestbook> buscarPorNombre(String name, int begin, int end)
{
    return guestbookFinder.buscarPorNombre(name, begin, end);
}
```

Despues de añadir el método al Servicio, recordar regenerar las clases con la tarea **buildService**.

4.7. Dynamic Query

API que permite definir consultas de forma programatica.

Se basa en la clase **DynamicQuery**.

El prcedimiento es similar al de las **Custom Queries**, en cuanto que hay que definir una clase de **Finder**, que herede de **BasePersistenceImpl<>**, implementar el método deseado con el API de **DynamicQuery** y luego lanzar la tarea de **service builder**.

```
public List<Event> findByEntryNameGuestbookName(String entryName,
String guestbookName) {

    Session session = null;
    try {
        session = openSession();

        DynamicQuery guestbookQuery = DynamicQueryFactoryUtil.forClass
(Guestbook.class)
            .add(RestrictionsFactoryUtil.eq("name", guestbookName))
            .setProjection(ProjectionFactoryUtil.property("guestbookId
"));

        Order order = OrderFactoryUtil.desc("modifiedDate");

        DynamicQuery entryQuery = DynamicQueryFactoryUtil.forClass
(Entry.class)
            .add(RestrictionsFactoryUtil.eq("name", entryName))
            .add(PropertyFactoryUtil.forName("guestbookId").in
(guestbookQuery))
            .addOrder(order);

        List<Event> entries = EventLocalServiceUtil.dynamicQuery
(entryQuery);

        return entries;
    }
    catch (Exception e) {
        try {
            throw new SystemException(e);
        }
        catch (SystemException se) {
            se.printStackTrace();
        }
    }
    finally {
        closeSession(session);
    }
}
```

El API de **DynamicQuery**, ofrece una factoria **DynamicQueryFactoryUtil** para crear las consultas, a la cual se la van pasando restricciones, orden, limite y/o proyeccion.

Una vez definida la consulta, se puede ejecutar empleando la capa de Servicios.

4.7.1. Restriciones

Para las restricciones se dispone de **RestrictionsFactoryUtil**, que permite definir restricciones de tipo

- and
- between
- gt
- lt
- eq
- like
- in
- isEmpty
- isNull

Para definir restrcciones respecto a un campo, se tiene **PropertyFactoryUtil** que permite referencias a los campos.

4.8. Model Listener

Permiten definir algun comportamiento extra asociado a las acciones de create, remove y/o update sobre las entidades de forma transparente, a traves de un modelo de eventos.

Se invocan antes de que se complete la transacción.

Algunos de los usos que se le puede dar son:

- Auditar las operaciones: Registrar en una tabla de base datos las operaciones que se van aplicando a las entidades, un historico de cambios.
- Limpiar Caches:
- Validaciones: Validar los datos del modelo antes de enviarlos a la base de datos.
- Notificaciones a los usuarios de cambios en datos de su propiedad, creados por ellos, a los que se han suscrito, etc.

Para definir un **ModelListener** se ha de crear una clase que implemente la interface

BaseModelListener<Entidad> y se ha de registrar como componente OSGI de tipo **ModelListener**.

```
@Component(
    immediate = true,
    service = ModelListener.class
)
public class CustomEntityListener extends BaseModelListener
<CustomEntity> {

    // Override one or more methods from the ModelListener interface.

}
```

4.8.1. Tipos de Eventos

- **onAfterAddAssociation**: Si nuestra entidad tiene una relacion con otra entidades a través de una tabla, este **Listener** se invoca después de agregar un registro de la relacion.
- **onAfterRemoveAssociation**: Si nuestra entidad tiene una relacion con otra entidades a través de una tabla, este **Listener** se invoca después de eliminar un registro de la relacion.
- **onAfterCreate**: Se invoca después llamar al método **create** de la capa de persistencia.
- **onAfterRemove**: Se invoca después llamar al método **remove** de la capa de persistencia.
- **onAfterUpdate**: Se invoca después llamar al método **update** de la capa de persistencia.
- **onBeforeAddAssociation**: Si nuestra entidad tiene una relacion con otra entidades a través de una tabla, este **Listener** se invoca antes de agregar un registro de la relacion.
- **onBeforeRemoveAssociation**: Si nuestra entidad tiene una relacion con otra entidades a través de una tabla, este **Listener** se invoca antes de eliminar un registro de la relacion.
- **onBeforeCreate**: Se invoca antes llamar al método **create** de la capa de persistencia.

- **onBeforeRemove**: Se invoca antes llamar al método **remove** de la capa de persistencia.
- **onBeforeUpdate**: Se invoca antes llamar al método **update** de la capa de persistencia.

4.9. Data Scopes

En Liferay se puede restringir el ambito de los datos a 3 ámbitos:

- Gobal Scope
- Site Scope
- Page Scope

4.9.1. Gobal Scope

Se puede acceder al contenido de este ambito a través de todo el portal, es decir, se puede compartir la informacion entre los sitios.

El contenido de este ambito se puede agregar desde:

- **Panel de control - Sitios - Global**
- Portlets en páginas de sitio configuradas para el ambito **Global Scope**.

4.9.2. Site Scope

Se puede acceder al contenido de este ambito desde un Sitio en particular.

Este es el alcance predeterminado para los portlets agregados en un Sitio.

El contenido en este ámbito es visible a través del conjunto de páginas públicas y privadas.

El contenido de este ambito se puede agregar desde:

- **Panel de control - Sitios - <Nombre del sitio> - Sección de contenido**
- Portlets en las páginas del sitio configuradas para el ambito **Site Scope**.

4.9.3. Page Scope

Se puede acceder al contenido de este ambito desde un conjunto de páginas

públicas o privadas, pero no ambas que estan dentro de un sitio particular

El contenido en este ambito se puede agregar desde portlets en las paginas del sitio configurados para este ambito **Page Scope**.

Asset Publisher es uno de esos portlets que puede mostrar los contenidos de todos los ámbitos en un solo lugar si está configurado correctamente.

4.9.4. Cambio de Scope

Para modificar el ambito de los datos con los que trabaja un portlet particular, se ha de acceder al menú **Configuration - Scope** del Portlet en una pagina particular, ahí se permite seleccionar el Ambito a elegir entre: **Sitio**, **Global**, **Paginas que ya tengan algun portlet con scope page** y **Pagina actual (New)**.

A partir de que se define un ambito distinto al por defecto, en la seccion de administración del contenido **Menu - Site - Content**, aparece una nueva opcion para seleccionar el ambito del cual se quieren consultar los contenidos.

4.9.5. API

Para habilitar el uso de los ambitos, en las entidades se deben tener los campos:

- **companyId (long)**: para habilitar el ambito **Gobal Scope**.
- **groupId (long): para habilitar el ambito *Site Scope**.

Y en los Portlet se debe añadir la propiedad **com.liferay.portlet.scopeable=true**

```
@Component(
    immediate = true,
    property = {
        "com.liferay.portlet.scopeable=true"
    },
    service = Portlet.class
)
public class JournalContentPortlet extends MVCPortlet {}
```

A partir de ese momento se ha de emplear **scopeGroupId** para recuperar los datos, este valor se puede obtener con

- **themeDisplay**

```
ThemeDisplay themeDisplay = (ThemeDisplay)actionRequest.getAttribute
(WebKeys.THEME_DISPLAY);

themeDisplay.getScopeGroupId();
```

```
<liferay-theme:defineObjects />
<%
themeDisplay.getScopeGroupId();
%>
```

- **serviceContext**

```
ServiceContext serviceContext = ServiceContextFactory.getInstance(User
.class.getName(), actionRequest);
serviceContext.getScopeGroupId();
```

Para recuperar datos de otras aplicaciones que esten en un ambito de sitio, se puede emplear **themeDisplay.getSiteGroupId()**

5. Custom Fields

El API de Liferay permite extender los campos de una entidad persistente ya existente, para enriquecerla con contenido distinto al generico.

Las entidades que se pueden extender son:

- **Blogs Entry**
- **Bookmarks Entry**
- **Bookmarks Folder**
- **Calendar Booking**
- **Document**
- **Documents Folder**
- **Message Boards Category**
- **Message Boards Message**
- **Organization**

- **Page**
- **Role**
- **Site**
- **User**
- **User Group**
- **Web Content Article**
- **Web Content Folder**
- **Wiki Page**

5.1. Definicion de un nuevo campo

Para configurarlos se accede al **Panel de Control ¬ Configuracion ¬ Campos Personalizados** del Portal.

Se selecciona la entidad que incluirá el campo personalizado nuevo y se pulsa **+**.

Se define la clave para referenciar al campo y el tipo.

Una vez guardado, se puede configurar las siguientes caracteristicas del campo.

- Default Value
- Hidden
- Visible with Update
- Searchability

Para asignarle un valor para una entidad dada se accede a la administración de la entidad y en la sección **Campos Personalizados**, se le puede dar valor.

5.2. Tipos de campos

- Con Valores pre-establecidos:
  - ¬ **Selección de valores enteros**
  - ¬ **Selección de valores decimales**
  - ¬ **Selección de valores de texto**
  - ¬ **Caja de texto**

    - ¬ **Cuadro de texto indexado**
    - ¬ **Texto Field-Secret**
    - ¬ **Campo de texto indexado**
- Con valores primitivos:
    - ¬ **Verdadero Falso**
    - ¬ **Fecha**
    - ¬ **Número decimal (64 bits)**
    - ¬ **Grupo de números decimales (64 bits)**
    - ¬ **Número decimal (32 bits)**
    - ¬ **Grupo de números decimales (32 bits)**
    - ¬ **Entero (32 bits)**
    - ¬ **Grupo de enteros (32 bits)**
    - ¬ **Entero (64 bits)**
    - ¬ **Grupo de enteros (64 bits)**
    - ¬ **Número decimal o entero (64 bits)**
    - ¬ **Grupo de números decimales o enteros (64 bits)**
    - ¬ **Entero (16 bits)**
    - ¬ **Grupo de enteros (16 bits)**
    - ¬ **Texto**
    - ¬ **Grupo de valores de texto**
    - ¬ **Texto localizado**

5.3. API

Si se desea definir un comportamiento de negocio basado en los nuevos campos, será necesario acceder a sus valores para ello el API de Liferay pone a nuestra disposición las siguientes funcionalidades.

- Acceso al valor de un campo

```
user.getExpandoBridge().getAttribute("attirbute-key");
```

- Establecimiento del valor de un campo

```
user.getExpandoBridge().setAttribute("attirbute-key", "value");
```

- Visualización de todos los campos

```
<%@ taglib uri="http://liferay.com/tld/ui" prefix="liferay-ui" %>

<%User user=themeDisplay.getUser();%>

<liferay-ui:custom-attribute-list
    className="<%= User.class.getName() %>"
    classPK="<%= user!= null ? user.getUserId() : 0 %>"
    editable="<%= true %>"
    label="true"/>
```

- Visualización de un campo concreto

```
<%@ taglib uri="http://liferay.com/tld/ui" prefix="liferay-ui" %>

<%User user=themeDisplay.getUser();%>

<liferay-ui:custom-attribute
    className="<%= User.class.getName() %>"
    classPK="<%= user!= null ? user.getUserId() : 0 %>"
    editable="<%= true %>"
    name="company-name"
    label="true"/>
```

5.4. Validacion

Se pueden aplicar tanto validaciones del lado cliete, incluyendo codigo javascript que valide la entrada de datos en el campo, como en el lado servidor que valide que el dato recibido cumple con unas premisas antes de ser procesado.

5.4.1. Validacion Servidor

La validacion del lado servidor se basa en la creación de un **MVCCommand** (sustituye a los Hooks de Actions de Struts) que sobreescriba el comportamiento normal del Portal, generalmente se precisará de la referencia al **MVCCommand**

original para seguir realizando dicha logica en caso de que la validación se realice correctamente.

```
@Component(
    property = {
        "javax.portlet.name=" + BlogsPortletKeys.BLOGS_ADMIN,
        "mvc.command.name=/users_admin/edit_user",
        "service.ranking:Integer=100"
    },
    service = MVCActionCommand.class)
public class CustomUserMVCActionCommand extends BaseMVCActionCommand {

    @Override
    protected void doProcessAction
        (ActionRequest actionRequest, ActionResponse actionResponse)
        throws Exception {

        //Se obtiene el objeto *ServiceContext* asociado al tipo User,
que contiene información
        //transersal del objeto, en este caso nos interesan los
*ExpandoBridgeAttributes*
        ServiceContext serviceContext = ServiceContextFactory
.getInstance(User.class.getName(), actionRequest);

        //A partir del objeto *ServiceContext*, se obtiene el valor
asignado al *Custom Field*
        //deseado, en este caso *Color*
        String color = (String)serviceContext
.getExpandoBridgeAttributes().get("Color");

        //Se aplican las validaciones pertinentes
        if (Validator.isNull(color)) {
            //Si las validaciones no se cumplen se avisa al usuario en
la misma página
            SessionErrors.add(actionRequest, "Color no puede ser Null"
);
            String mvcPath = "/edit_user.jsp";
            actionResponse.setRenderParameter("mvcPath", mvcPath);
        } else {
            //Si las validaciones se cumplen se continua realizando la
tarea original
            mvcActionCommand.processAction(actionRequest,
actionResponse);
        }
    }
```

```
    @Reference(
        target =
"(component.name=com.liferay.blogs.web.internal.portlet.action.EditEntr
yMVCActionCommand)")

    protected MVCActionCommand mvcActionCommand;

}
```

Se puede emplear la clase **com.liferay.portal.kernel.util.Validator** para aplicar las validaciones mas habituales.

En caso de error de validación, lo habitual es volver a mostrar la misma pagina de donde venimos, y enviarle a está ls errores de validacion empleando la clase **com.liferay.portal.kernel.servlet.SessionErrors**

5.4.2. Validacion Cliente

Se pueden registrar nuevas validaciones accediendo al javascript de la página

```
Liferay.Form.register(
    {
        id: '<portlet:namespace />_fm',
        fieldRules: [
            {
                body: function (val, fieldNode, ruleValue) {
                        return (val !== 'Verde');
                },
                custom: true,
                errorMessage: 'El campo no puede tener como valor
Verde',
                fieldName: '<portlet:namespace />_ExpandoAttribute--
Color--',
                validatorName: 'custom_color_validator'
            }
        ]
    }
);
```

6. Message Bus

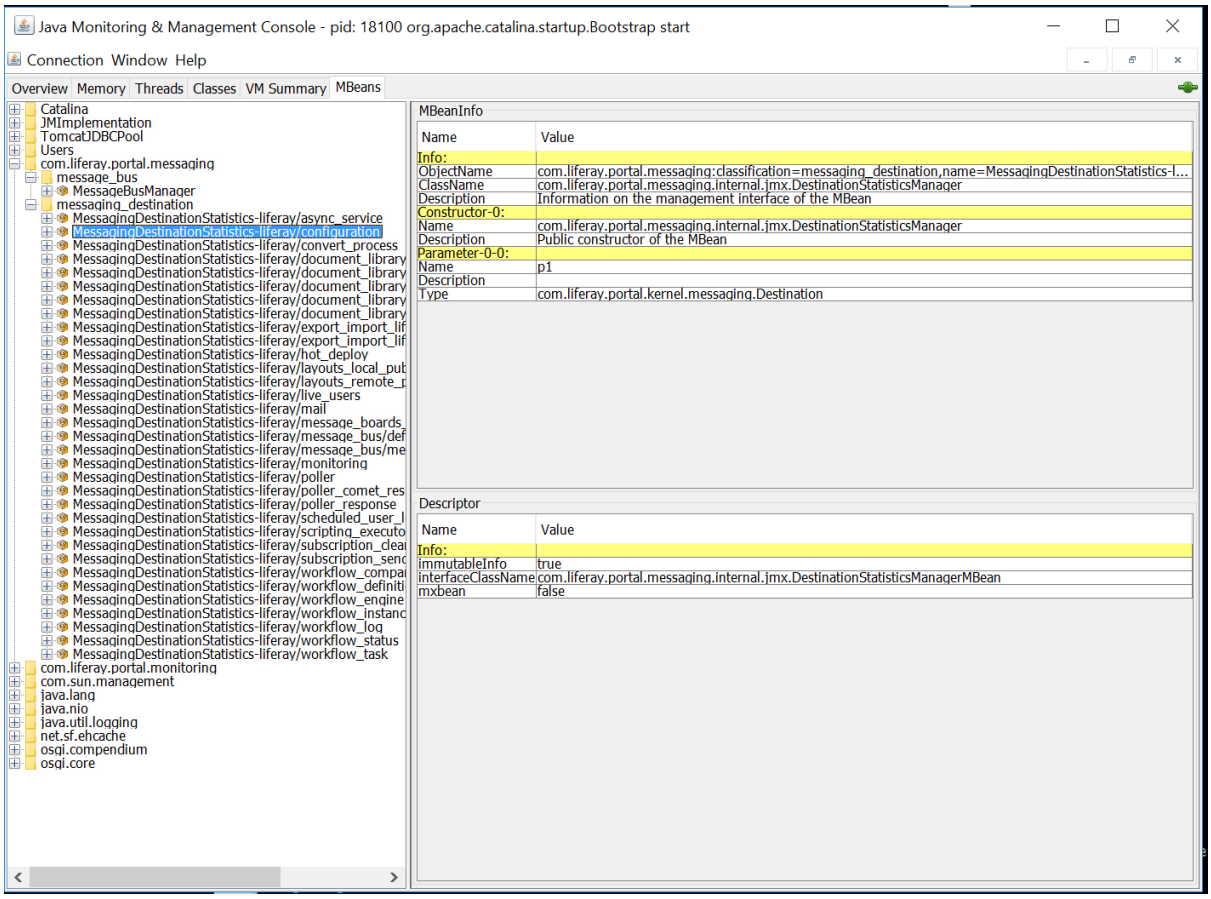
Permite el procesa de tareas de forma asincrona.

Similar a los **Topics de Java Messaging Service (JMS)**, esto es puede haber mas de un consumidor para cada mensaje, pero sin soporte transaccional, lo que lo hace más ligero.

Liferay lo emplea de forma nativa para:

- Auditoria
- Integración con el motor de búsqueda
- Subscripciones por correo electronico
- Monitorización
- Procesamiento de biblioteca de documentos
- Tareas en segundo plano
- Ejecución de solicitud en todo el clúster
- Replicación de caché agrupada

Se puede monitorizar con la herramienta **JConsole** que incluye la JDK, en la sección de MBeans



Los componentes que forman este API son:

- Destinos (Topics)
- MessageListener (Consumidores)
- MessageSender (Productores)

6.1. Destinos

Permite desacoplar productor y consumidor.

Existen diferentes tipos de **Destinos**:

- Destino paralelo: Los mensajes se encolan. Se emplea un hilo de ejecución (worker) por cada Listener que ha de procesar el mensaje recibido.
- Destino serie: Los mensajes se encolan. Se emplea un hilo de ejecución (worker) por mensaje recibido independientemente de los **Listener** que lo deban recibir.
- Destino sincróno: Los mensajes no se encolan, se envian directamente a los **Listener**. El mismo hilo que envia el mensaje al **Destino**, lo hace llegar a todos los **Listener**.

Si se desea emplear alguno de los destinos predefinidos, los nombres de los mismos están definidos en la clase **DestinationNames**.

6.1.1. DestinationConfiguration

Los **Destinos** se configuran con una instancia de la clase **DestinationConfiguration**, donde se configuran aspectos como

- **Maximum Queue Size**: Mensajes maximos de la cola.
- **Rejected Execution Handler**: Clase manejadora que extienede de * com.liferay.portal.kernel.concurrent.RejectedExecutionHandler* que gestiona cuando un mensaje es rechazado por estar la cola llena.
- **Workers Core Size**: Numero inicial de hilos para el procesamiento de los mensajes.
- **Workers Max Size**: Numero maximo de hilos para el procesamineto de los mensajes.

Se proporcionan métodos estaticos en la clase **DestinationConfiguration** para la creación de preconfiguraciones acordes a los tipos mencionados anteriormente.

- createParallelDestinationConfiguration(String destinationName).
- createSerialDestinationConfiguration(String destinationName).
- createSynchronousDestinationConfiguration(String destinationName).

6.1.2. Creacion de un Destino

Para crear un **Destino** se han de seguir los siguientes pasos

- Crear una instancia de **DestinationConfiguration**, ya sea con los métodos estaticos o con el constructor.

```
DestinationConfiguration destinationConfiguration = new
DestinationConfiguration(DestinationConfiguration.DESTINATION_TYPE_PARA
LLEL, "myDestinationName");

destinationConfiguration.setMaximumQueueSize(_MAXIMUM_QUEUE_SIZE);

RejectedExecutionHandler rejectedExecutionHandler =
    new CallerRunsPolicy() {

        @Override
        public void rejectedExecution(
            Runnable runnable, ThreadPoolExecutor threadPoolExecutor) {

            if (_log.isWarnEnabled()) {
                _log.warn(
                    "The current thread will handle the request " +
                        "because the graph walker's task queue is at "
+
                            "its maximum capacity");
            }

            super.rejectedExecution(runnable, threadPoolExecutor);
        }

    };

destinationConfiguration.setRejectedExecutionHandler(rejectedExecutionH
andler);
```

- Crear una instancia de **Destination** empleando la factoria **DestinationFactory**

```
Destination destination = _destinationFactory.createDestination
(destinationConfiguration);
```

La cual se deberá obtener como servicio OSGI

```
@Reference
private DestinationFactory _destinationFactory;
```

- Registrar el Destination como servicio OSGI, invocando el método **registerService(Destination.class, destination, properties)** de un objeto

BundleContext, donde se indican

- ¬ Tipo generico del servicio a registrar, en este caso **Destination.class**.
- ¬ Instancia del **Destino**.
- ¬ Instancia de **Dictionary** con las propiedades del **Destino**, incluido **destination.name**.

```
Dictionary<String, Object> properties = new HashMapDictionary<>();
properties.put("destination.name", destination.getName());

ServiceRegistration<Destination> serviceRegistration = _bundleContext
.registerService(Destination.class, destination, properties);
```

El objeto **BundleContext** se puede obtener como parametro del método de activación de un Componente OSGI.

```
@Component (
    immediate = true,
    service = MyMessagingConfigurator .class
)
public class MyMessagingConfigurator {

    @Activate
    protected void activate(BundleContext bundleContext) {

    //Aquí se han de poner las sentencia explicadas anteriormente

    _serviceRegistration = serviceRegistration;

    }

    @Deactivate
    protected void deactivate() {
        Destination destination = _bundleContext.getService
(_serviceRegistration.getReference());

        _serviceRegistration.unregister();

        destination.destroy();
    }

    private ServiceRegistration<Destination> _serviceRegistration;
}
```

Para mejorar el rendimiento de la instancia de Liferay, conviene poder eliminar el **Destino** creado, para ello se puede hacer uso del método de desactivación.

6.2. Message Listener

MessageListener es una interface que ofrece el API para poder recibir los mensajes enviados a un **Destino**.

Para registralo existen tres opciones:

- Como componente OSGI, donde se ha de indicar como **property** el **destination.name**

```
@Component (
    immediate = true,
    property = {"destination.name=myDestinationName"},
    service = MessageListener.class
)
public class MyMessageListener implements MessageListener {

   public void receive(Message message) {
        _log.info("Se va a procesar el mensaje: " + message);
   }
}
```

- Directamente sobre la instancia del **MessageBus**

```
@Component (
    immediate = true,
    service = MyMessageListenerRegistrator.class
)
public class MyMessageListenerRegistrator {

    @Activate
    protected void activate() {

        _messageListener = new MessageListener() {

            public void receive(Message message) {
                 _log.info("Se va a procesar el mensaje: " + message);
            }
        };

        _messageBus.registerMessageListener("myDestinationName",
_messageListener);
    }

    @Deactivate
    protected void deactivate() {
        _messageBus.unregisterMessageListener("myDestinationName",
_messageListener);
    }

    @Reference
    private MessageBus _messageBus;

    private MessageListener _messageListener;
}
```

- Directamente sobre la instancia del **Destination**

```
@Component (
    immediate = true,
    service = MyMessageListenerRegistrator.class
)
public class MyMessageListenerRegistrator {

    @Activate
    protected void activate() {

        _messageListener = new MessageListener() {

            public void receive(Message message) {
                _log.info("Se va a procesar el mensaje: " + message);
            }
        };

        _destination.register(_messageListener);
    }

    @Deactivate
    protected void deactivate() {
        _destination.unregister(_messageListener);
    }

    @Reference(target = "(destination.name=someDestination)")
    private Destination _destination;

    private MessageListener _messageListener;
}
```

6.3. Message Bus Event Listeners

MessageBusEventListener es una interface que ofrece el API para monitorizar cuando se añaden/destruyen **Destinos**.

Para emplearla, se ha de definir una clase que implemente la interface y registrarla como servicio OSGI.

```
@Component(
    immediate = true,
    service = MessageBusEventListener.class
)
public class MyMessageBusEventListener implements
MessageBusEventListener {

    void destinationAdded(Destination destination) {
        _log.info("Se ha añadido el Destino: " + destination.getName()
);
    }

    void destinationDestroyed(Destination destination) {
        _log.info("Se ha borrado el Destino: " + destination.getName()
);
    }
}
```

6.4. Destination Event Listener

DestinationEventListener es una interface que ofrece el API para monitorizar cuando se registran/desregistrar **Listener** en un **Destino**.

Para emplearla, se ha de definir una clase que implemente la interface y registrarla como servicio OSGI, indicando como **property** el **destination.name**

```
@Component(
    immediate = true,
    property = {"destination.name=myDestinationName"},
    service = DestinationEventListener.class
)
public class MyDestinationEventListener implements
DestinationEventListener {

    void messageListenerRegistered(String destinationName,
MessageListener messageListener) {
        _log.info("Se ha registrado un nuevo Listener: " +
messageListener);
    }

    void messageListenerUnregistered(String destinationName,
MessageListener messageListener) {
        _log.info("Se ha desregistrado un Listener: " +
messageListener);
    }
}
```

6.5. Envio de Mensajes al Bus

Se pueden enviar mensajes de forma

- Asincrona: Se envia el emnsaje y se siue realizando el resto de la tarea empezada, sin esperar respuesta.
- Sincrona: Se envia el mensaje y se permanece a la espera de una respuesta, esta puede ser **timeout**.

Para enviar el mensaje se pueden hacer de tres formas distintas

- Directamente sobre el **Message Bus**

```
@Component(
    immediate = true,
    service = SomeServiceImpl.class
)
public class SomeServiceImpl {

    public void sendSomeMessage() {

        Message message = new Message();
        message.put("myId", 12345);
        message.put("someAttribute", "abcdef");
        _messageBus.sendMessage("myDestinationName", message);
    }

    @Reference
    private MessageBus _messageBus;
}
```

- Empleando **MessageSender**, que permite enviar mensajes de forma asincrona.

```
@Component(
    immediate = true,
    service = SomeServiceImpl.class
)
public class SomeServiceImpl {

    public void sendSomeMessage() {

        Message message = new Message();
        message.put("myId", 12345);
        message.put("someValue", "abcdef");

        SingleDestinationMessageSender messageSender =
           _messageSenderFactory.createSingleDestinationMessageSender(
"myDestinationName");

        messageSender.send(message);
    }

    @Reference
    private SingleDestinationMessageSenderFactory
_messageSenderFactory;
}
```

- Empleando **SynchronousMessageSender**, que permite eviar mensajes de forma sincrona, parando la ejecución hasta obtener respuesta o **timeout**. Tiene dos formas de trabajar:
  - ¬ DEFAULT: Entrega el mensaje en un hilo independiente, proporcionando la opcion de continuar si se alcanza el **timeout**.
  - ¬ DIRECT: Entraga el mensaje en el mismo hilo con el que se envia, no permitiend **timeout**.

```
@Component(
    immediate = true,
    service = SomeServiceImpl.class
)
public class SomeServiceImpl {

    public void sendSomeMessage() {

        Message message = new Message();
        message.put("myId", 12345);
        message.put("someAttribute", "abcdef");

        SingleDestinationSynchronousMessageSender messageSender =
            _messageSenderFactory
.createSingleDestinationSynchronousMessageSender(
                "myDestinationName", SynchronousMessageSender.Mode
.DEFAULT);

        messageSender.send(message);

    }

    @Reference
    private SingleDestinationMessageSenderFactory
_messageSenderFactory;
}
```

6.5.1. Envio de mensajes a traves del Cluster

Para garantizar que todos los Nodos que formen el Cluster de Liferay obtenan los mensajes enviados a un destino, el destino se tiene que registrar en el Cluster, en realidad lo que se hace es añadir un nuevo **MessageListener** al **Destino**, pero esta vez no de un tipo definido por nosotros, sino de un tipo que nos proporciona el API de Liferay, en este caso **ClusterBridgeMessageListener**, este listener es el que se encarga de trasladar el mensaje a los nodos del Cluster.

Se ha de establecer una prioridad que va de **Level_1** a **Level_10**, siendo este ultimo el mas importante.

```
@Component(
    immediate = true,
    service = MyMessageListenerClusterRegistrator.class
)
public class MyMessageListenerClusterRegistrator {
    ...

    @Activate
    protected void activate() {

        _clusterBridgeMessageListener = new
ClusterBridgeMessageListener();
        _clusterBridgeMessageListener.setPriority(Priority.LEVEL_5)
        _destination.register(_clusterBridgeMessageListener);
    }

    @Deactivate
    protected void deactivate() {

        _destination.unregister(_clusterBridgeMessageListener );
    }

    @Reference(target = "(destination.name=myDestinationName)")
    private Destination _destination;

    private MessageListener _clusterBridgeMessageListener;
}
```

7. Device Recognition

API que detecta todas las capacidades que tiene el cliente que se conecta al Portal, permitiendo definir unas reglas para elegir la mejor manera de representar las paginas del portal para el cliente.

Para usar el API, lo primero es instalar una base de datos de detección de dispositivo que pueda detectar qué dispositivos móviles están accediendo al portal. Liferay Portal proporciona dicha base de datos en la aplicación **Liferay Mobile Device Detection (LMDD)** que se puede encontrar en **Liferay Marketplace**.

Una vez instalada, se ha de acceder al objeto **Device** a traves de **ThemeDisplay**

```
Device device = themeDisplay.getDevice();
```

NOTE

Para obtener el objeto **themeDisplay** basta con hacer

```
ThemeDisplay themeDisplay = (ThemeDisplay) renderRequest
.getAttribute(WebKeys.THEME_DISPLAY);
```

O en las JSPs

```
<%@ taglib uri="http://liferay.com/tld/theme"
prefix="liferay-theme" %>
<liferay-theme:defineObjects />
```

Este objeto proporiona los siguientes metodos

- **String getBrand()**: Marca del terminal
- **String getBrowser()**: Tipo de navegador empleado
- **String getBrowserVersion()**: Versión del navegador
- **Map<String,Capability> getCapabilities()** (Deprecated): Mapa con todas las caracteristicas
- **String getCapability(String name)** (Deprecated): Obtener una caracteristica por su nombre
- **String getModel()**: Modelo del terminal
- **String getOS()**: Sistema Operativo del terminal
- **String getOSVersion()**: Version del SO
- **String getPointingMethod()**: Tipo de puntero para señalar.
- **Dimensions getScreenPhysicalSize()**: Tamaño fisico de la pantalla.
- **Dimensions getScreenResolution()**: Resolución de la pantalla.
- **boolean hasQwertyKeyboard()**: Si tiene teclado QWERTY
- **boolean isTablet()**: Si es una Tablet.

```
Dimensions dimensions = device.getScreenSize();
float height = dimensions.getHeight();
float width = dimensions.getWidth();
```

8. Javascript

En la version 7 de Liferay, se ha incluido soporte para otros frameworks a parte de AlloyUI.

- EcmaScript 2015
- Metal.js (developed by Liferay)
- jQuery (included)
- Lodash (included)

A parte Liferay proporciona un API para poder obtener información del portal tambien desde el Javascript, para ello ofrece el objeto **Liferay**.

8.1. ThemeDisplay

A partir de este, se puede obtener acceso a otros objetos como **ThemeDisplay**, que entre otras ofrece la siguiente informacion

- getCompanyId: Identificador de la instancia de Liferay, puede haber varias instancias de Liferay en una misma base de datos.
- getLanguageId: Identificador del idiaoma del usuario.
- getScopeGroupId: Identificador del Sitio actual.
- getUserId: Identificador de usuario
- getUserName: Nombre de usuario.
- getPathImage: Ruta relativa del directorio de imágenes del portlet.
- getPathJavaScript: Ruta relativo al directorio que contiene los ficheros javascript del portlet.
- getPathMain: Ruta del directorio principal de la instancia del portal.
- getPathThemeImages: Ruta del directorio de imágenes del tema actual.
- getPathThemeRoot: Ruta relativa del directorio raíz del tema actual.

- isImpersonated: Indica si el usuario actual está siendo suplantado por un administrador.
- isSignedIn: Indica si el usuario esta logado.

```
if(Liferay.ThemeDisplay.isSignedIn()){
    alert('Hello ' + Liferay.ThemeDisplay.getUserName() + '. Welcome
Back.')
}
else {
    alert('Hello Guest.')
}
```

8.2. Language

Permite obtener textos traducidos en el Portal.

```
Liferay.Language.get('read-more')
```

8.3. Portlet

Permite realizar alguna tarea sobre el Portlet, como:

- **refresh**: Refrescarlo.

```
Liferay.Portlet.refresh('#p_p_id_com_liferay_blade_samples_configuratio
naction_ExampleConfigPortlet_INSTANCE_EO9erDXq86fc_')
```

- **close**: Borrarlo de la pagina.

```
Liferay.Portlet.close('#p_p_id_com_liferay_blade_samples_configurationa
ction_ExampleConfigPortlet_INSTANCE_EO9erDXq86fc_')
```

- **minimize**: Minimizarlo.

```
Liferay.Portlet.minimize('#p_p_id_com_liferay_blade_samples_configurati
onaction_ExampleConfigPortlet_INSTANCE_8jIX3oad0nBk_')
```

8.4. Browser

Otro objeto interesante es **Browser**, que ofrece informacion de las características del navegador, sin hacer uso del objeto **window.navigator** del estandar.

Los métodos disponibles en este objeo son:

- acceptsGzip: Indica si se aceptan transferencias de ficheros comprimidos.
- getMajorVersion: Retorna el principal valor de la version del navegador.
- getRevision
- getVersion: Retorna la version del navegador.
- isAir
- isChrome
- isFirefox
- isGecko
- isIe
- isIphone
- isLinux
- isMac
- isMobile
- isMozilla
- isOpera
- isRtf
- isSafari
- isSun
- isWebKit
- isWindows

8.5. Util

Ofrece un conjunto de funciones de utilidad que permiten manejar de forma mas comoda las paginas del portal. Algunas de elas son:

- **check**: Permite chekar un checkbox de un formulario.
- **endsWith/startsWith**: Permite validar si un String termina/comienza por un texto.

```
Liferay.Util.startsWith("Hola Mundo", 'H');
```

- **escapeHTML/unescapeHTML**: Aplica el scape/unescape html sobre un String.
- **randomInt**: Genera un numero aleatorio.
- **toNumber**: Convierte a numero un String.

8.6. PortletURL

Objeto que permite la creación de URLs asociadas a los portlets de tipo **actionURL**, **renderURL** y **resourceURL**.

Es necesario cargar el modulo **liferay-portlet-url** con **AUI**

```
AUI().use(
    'liferay-portlet-url',
    function(A) {
        var navigationURL;
        var portletURL = Liferay.PortletURL.createRenderURL();
        var url = themeDisplay.getLayoutURL();
        portletURL.setParameter("employerId", companyId);
        portletURL.setPortletId(A.one('#custSupportPortletId'));
        navigationURL = portletURL.toString();
        window.location = navigationURL;
    }
);
```

8.7. Service

Objeto que permite acceder al API de **Servicios Web Json** que proporciona Liferay, que se puede consultar aquí

```
Liferay.Service(
    '/user/get-user-by-email-address',
    {
        companyId: Liferay.ThemeDisplay.getCompanyId(),
        emailAddress: 'test@liferay.com'
    },
    function(obj) {
        console.log(obj);
    }
);
```

El método **Service** acepta cuatro parametros:

- **Service** (Obligatorio): Indica el servicio a invocar. Acepta dos formas
  - ¬ Simple: Se indica unicamente el **nombre del servicio**
  - ¬ Compleja: Se indica un objeto con clave el **nombre del servicio** y como valor otro objeto con los parametros a enviar al servicio.

```
Liferay.Service(
    {
        '/user/get-user-by-email-address': {
                companyId: Liferay.ThemeDisplay.getCompanyId(),
                emailAddress: 'test@liferay.com'
        }
    },
    function(obj) {
        console.log(obj);
    }
);
```

- **data** (Opcional): Indica los datos a enviar al servicio. Si el objeto pasado es el ID de un formulario o un elemento de formulario, los campos del formulario se serializarán y se usarán como datos.
- **successCallback**: Indica la función a ejecutar cuando el servidor devuelve una respuesta correcta. Recibe un objeto JSON.
- **exceptionCallback**: Indca la función a ejecutar cuando la respuesta del servidor es un error del servicio. Recibe un mensaje de excepción.

8.7.1. Invocacion Multiple

Se pueden invocar multiples servicios a la vez, pasando un array de objetos como primer parametro.

```
Liferay.Service(
    [
        {
            '/user/get-user-by-email-address': {
                    companyId: Liferay.ThemeDisplay.getCompanyId(),
                    emailAddress: 'test@liferay.com'
            }
        },
        {
            '/role/get-user-roles': {
                    userId: Liferay.ThemeDisplay.getUserId()
            }
        }
    ],
    function(obj) {
        // obj is now an array of response objects
        // obj[0] == /user/get-user-by-email-address data
        // obj[1] == /role/get-user-roles data

        console.log(obj);
    }
);
```

8.7.2. Invocacion Anidada

Se pueden realizar invocaciones anidadas

```
Liferay.Service(
    {
        "$user = /user/get-user-by-id": {
            "userId": Liferay.ThemeDisplay.getUserId(),
            "$contact = /contact/get-contact": {
                    "@contactId": "$user.contactId"
            }
        }
    },
    function(obj) {
        console.log(obj);
    }
);
```

En este caso, se define una variable **$user** asociada al objeto JSON retornado por el primer servicio, que será empleado en la invocacion del segundo servicio.

En el objeto retornado a parte del objeto **user**, que es el principal obtenido con el primer servicio, se obtiene anidado uno **contact**, por el nombre de la variable, que es el obtenido con el segundo servicio.

8.7.3. Filtrado de campos

Se puede filtrar los campos obtenidos por las peticiones a los servicios, sin mas que asociar a la variable que representa los objetos un array con los campos deseados

```
Liferay.Service(
    {
        "$user[emailAddress,firstName,contactId] = /user/get-user-by-
id": {
            "userId": Liferay.ThemeDisplay.getUserId(),
            "$contacto[emailAddress] = /contact/get-contact": {
                "@contactId": "$user.contactId"
            }
        }
    },
    function(obj) {console.log(obj);}
);
```

9. Modulos Javascript

Permite modularizar las funcionalidades de JS.

Lieray ofrece compatibilidad con Modulos de:

- ES2015
- AMD
- AUI (YUI)

9.1. Definicion de Modulos AUI

Se ha de crear un nueo modulo OSGI, en este proyecto, unicamente interesa:

- fichero **bnd.bnd**
- fichero **build.gradle** (vacio)
- carpeta **src/main/resources/META-INF/resources**

Se han de incluir en el fichero **bnd.bnd** las siguientes cabeceras:

- **Liferay-JS-Config**: indica la ubicación del fichero de configuracion.
- **Web-ContextPath**: indica el contexto web (path) para poder recuperar los recursos que forman el modulo.

Contenido del fichero **bnd.bnd**

```
Liferay-JS-Config: /META-INF/resources/js/config.js
Web-ContextPath: /my-modulo
```

Se ha de definir el fichero **config.js** donde se describe la configuración del modulo

- **groups**: Permite definir varias agrupaciones de modulos.
- **groups/<identificador del grupo>/base**: Directorio donde extraer ls recursos del grupo.
- **groups/<identificador del grupo>/combine**: Indica si la descarga de los JS se hace de forma combinada empleando las menos peticiones HTTP posibles o no, si se emplea **Liferay.AUI.getCombine()**, se está haciendo uso de la propiedad de configuracion **js_fast_load**.

- **groups/<identificador del grupo>/modules**: Definiciones de modulos.
- **groups/<identificador del grupo>/modules/<identificador del modulo>/path**: Ruta al script del modulo desde **base**.
- **groups/<identificador del grupo>/modules/<identificador del modulo>/requires**: Array de modulos necesarios para el nuevo modulo definido.
- **root**: Ruta a anteponer a los nombres de modulos para el servicio combinado.

Contenido del fichero **config.js**

```
;(function() {
    AUI().applyConfig(
        {
            groups: {
                mymodulo: {
                    base: MODULE_PATH + '/js/',
                    combine: Liferay.AUI.getCombine(),
                    modules: {
                        'liferay-my-modulo': {
                            path: 'modulo.js',
                            requires: [
                                'aui-base'
                            ]
                        }
                    },
                    root: MODULE_PATH + '/js/'
                }
            }
        }
    );
})();
```

NOTE

Se puede emplear **MODULE_PATH** para hacer referencia al path en el que se publican los recursos del modulo OSGI, que tendrá en cuenta la propiedad **Web-ContextPath** definida en **bnd.bnd**.

NOTE

Si todo va bien, en la siguiente ruta **http://localhost:8080/o/my-modulo/js/modulo.js** deberia encontrarse el fichero javacript que define el modulo.

Para definir un nuevo modulo, se emplea el API de **AlloyUI**, que emplea el objeto **AUI** y su función **add**, para añadir un nuevo modulo, definiendo

- Identificador del modulo.
- Codigo a ejecutar cuando se carga el modulo, lo normal aquí es crear algun tipo de funcionalidad que quede asociada a algún objeto global, como en este caso es al objeto **Liferay**.
- Modulos de los que depende la funcionalidad descrita, deberan ser un subconjunto de los decarados en el ficheo **config.js**.

Contenido del fichero **modulo.js**

```
AUI.add(
    'liferay-my-modulo',
    function(A) {
        var InvocarServicioUsuarios = function() {

            var resultado;

            Liferay.Service(
                {
                    "$user[emailAddress,firstName,contactId] =
/user/get-user-by-id": {
                        "userId": Liferay.ThemeDisplay.getUserId(),
                        "$contacto[emailAddress] = /contact/get-
contact": {"@contactId": "$user.contactId"}
                    }
                },
                function(obj) {console.log(obj);resultado = obj;}
            );
            return resultado;
        };

        Liferay.InvocarServicioUsuarios = InvocarServicioUsuarios;
    },
    '',
    {
        requires: ['aui-base']
    }
);
```

9.2. Uso de Modulos AUI

Para emplear un modulo, tipicamente desde una JSP de un Portlet, se necesita cargar el modulo, para ello se hace uso de la etiqueta **<aui:script>**, haciendo

referencia al identificador del modulo. teniendo dentro la seguridad de que se ha ejecutado el codigo del modulo, en este caso el que asocia al objeto **Liferay** una nueva función **InvocarServicioUsuarios**.

```
<%@ taglib uri="http://liferay.com/tld/aui" prefix="aui" %>

<aui:script use="liferay-my-modulo">
    Liferay.InvocarServicioUsuarios();
</aui:script>
```

9.3. Definicion de Modulos AMD

Otra forma de definir modulos es con **Liferay AMD** (Asynchronous Module Loader), empleando para ello el objeto **Loader** asociado al objeto **Liferay** y su función **define** para definirlo y su función **require** para usarlo.

Para definirlo habrá que indicar:

- Identificador del modulo.
- Array de Modulos necesarios para la implementacion.
- Función que describ el comortamiento asociado al modulo.

```
<script type="text/javascript">
    Liferay.Loader.define(
        "my-modulo-amd/js/modulo",
        [],
        function () {
            Liferay.MyAlert = function(){
                alert ("Desde el modulo");
            };
        }
    );
</script>
```

9.4. Uso de Modulos AMD

La fora de emplear estos modulos es similar a los de AUI, pero en este caso empleando la propiedad **require** de la etiqueta **<aui:script>**

```
<aui:script require="my-modulo-amd/js/modulo">
    Liferay.MyAlert();
</aui:script>
```

O sin esta etiqueta.

```
<script type="text/javascript">
    Liferay.Loader.require('my-modulo-amd/js/modulo',
function(myModulo) {
        Liferay.MyAlert();
    }, function(error) {
        console.error(error);
    });
</script>
```

10. Web Service

Cuando una entidad se define con service builder como remota, se generan las interfaces y clases necesarias para disponer de acceso via SOAP y JSON.

En caso de disponer de implementacion local y remota, se han de implementar las clases **_LocalServiceImpl** y **_ServiceImpl**.

La recomendacion, es invocar a la **Local** desde la **Remota** incluyendo en la remota la verificación de la seguridad.

```
@Override
public JournalArticle addArticle(...) throws PortalException {

    JournalFolderPermission.check(getPermissionChecker(), groupId,
folderId, ActionKeys.ADD_ARTICLE);

    return journalArticleLocalService.addArticle(...);
}
```

Sin mas, se disponen de los servicios JSON, para los SOAP, habrá que seguir algun paso extra.

10.1. SOAP

El listado de servicios SOAP disponibles en el Portal se puede obtener en la url

```
http://[host]:[port]/api/axis
```

Algunos ejemplos de rutas a servicios SOAP

```
http://localhost:8080/api/axis/Portal_CompanyService?wsdl

http://localhost:8080/api/axis/Portal_UserService?wsdl

http://localhost:8080/api/axis/Portal_UserGroupService?wsdl
```

Para crear uno propio, basta con definir una entidad con service builder y lanzar sobre el proyecto **-service**, la tarea **buildWSDD**.

Una vez desplegado el WSDD, el servicio SOAP se encuentra disponible en la ruta

```
http://[host]:[port]/<nombre de despliegue de la aplicacion>/api/axis
```

NOTE	**nombre de despliegue de la aplicacion** sera el nombre de la app concatenando **-service**

Como ejemplo posible

```
http://localhost:8080/foo-service/api/axis
```

Para proyectos gradle, para tener disponible la tarea **buildWSDD**, habrá que añadir al fichero **settingss.gradle** la ssiguiente configuracion

```
//Las importaciones se añaden al cominezo del fichero

import com.liferay.gradle.plugins.service.builder.ServiceBuilderPlugin
import com.liferay.gradle.plugins.wsdd.builder.WSDDBuilderPlugin

//...

//El closure al final del fichero

gradle.beforeProject {
    project ->

    project.plugins.withType(ServiceBuilderPlugin) {
        project.apply plugin: WSDDBuilderPlugin
    }
}
```

Una vez lanzada la tarea, se obtiene un jar en la carpeta **build/libs/**, ques e ha de desplegar manualmente en la carpeta **/bundles/deploy**

10.2. JSON

El listado de servicios JSON disponibles en el Portal se puede obtener en la url

```
http://[host]:[port]/api/jsonws/
```

Los servicios JSON se publican directamente al crear un servicio remoto y lanzar **BuildService** sobre las operaciones definidas en ***ServiceImpl**

11. Indexacion

Liferay permite la definicion de indices sobre los contenidos (Asset) generados, de tal forma que puedan participar de las funcionalidades de busqueda rapida, que presentan mayor rendimineto que las busquedas directas sobre la base de datos.

En la version 7 de Liferay se usa como motor para el alamcen de los indices **ElasticSearch** y como herramienta de busqueda **Lucene**.

Para poder emplear los indices desde nuestros modulos, se ha de incluir la dependencia

```
compile group: 'org.elasticsearch', name: 'elasticsearch', version:
'2.2.2'
```

11.1. Indexador

Para poder añadir nuestros registros al indice, se ha de crear una clase que realice la indexacion.

- Para ello primero se necesita añadir al proyecto de **Service Builder** las siguientes dependencias

```
compileOnly group: "com.liferay", name: "com.liferay.registry.api",
version: "1.0.0"
compileOnly group: "javax.portlet", name: "portlet-api", version: "2.0"
compileOnly group: "javax.servlet", name: "javax.servlet-api", version:
"3.0.1"
```

NOTE	Recordar aque al cambiar el **build.gradle** hay que hacer **Refresh Gradle Project** para que los cambios sean efectivos.

- Segundo, se ha de crear una clase que herede de **BaseIndexer**

Ejemplo de Clase que hereda de BaseIndexer

```
package com.liferay.docs.guestbook.search;

@Component(
    immediate = true,
    service = Indexer.class
)
public class GuestbookIndexer extends BaseIndexer<Guestbook> {

    public static final String CLASS_NAME = Guestbook.class.getName();

    private static final Log _log = LogFactoryUtil.getLog
(GuestbookIndexer.class);

    //Referencias a servicios OSGI necsarios para la implementacion
    @Reference
    protected IndexWriterHelper indexWriterHelper;
```

```
    @Reference
    private GuestbookLocalService _guestbookLocalService;

    public GuestbookIndexer() {
        //Campos por los que se indexará por defecto
        setDefaultSelectedFieldNames(
            Field.ASSET_TAG_NAMES, Field.COMPANY_ID, Field.CONTENT,
            Field.ENTRY_CLASS_NAME, Field.ENTRY_CLASS_PK, Field
.GROUP_ID,
            Field.MODIFIED_DATE, Field.SCOPE_GROUP_ID, Field.TITLE,
Field.UID);
        //Activacion del filtratado basandose en los permisos
        setPermissionAware(true);
        //Activacion del filtrado de permisos elemento a elemento
        setFilterSearch(true);
    }

    @Override
    public String getClassName() {
        return CLASS_NAME;
    }

    //Se establece que para poder buscar un registro de tipo Guestbook,
hay que tener el permiso
    //VIEW
    @Override
    public boolean hasPermission(
            PermissionChecker permissionChecker, String entryClassName,
            long entryClassPK, String actionId)
        throws Exception {

        return GuestbookPermission.contains(
            permissionChecker, entryClassPK, ActionKeys.VIEW);
    }

    //Se encarga de añadir a los criterios de busqueda el estado del
registro, para que los que tienen estado *STATUS_IN_TRASH* no
participen de los resultados.
    @Override
    public void postProcessContextBooleanFilter(
            BooleanFilter contextBooleanFilter, SearchContext
searchContext)
    throws Exception {
        addStatus(contextBooleanFilter, searchContext);
    }
```

```
    //Se incluye en los parametros de busqueda, el campo titulo
*Field.TITLE* localizado.
    @Override
    public void postProcessSearchQuery(
        BooleanQuery searchQuery, BooleanFilter fullQueryBooleanFilter,
        SearchContext searchContext)
        throws Exception {

        addSearchLocalizedTerm(searchQuery, searchContext, Field.TITLE,
false);
    }

    //Impllementa el borrado de un registro del indice
    @Override
    protected void doDelete(Guestbook guestbook) throws Exception {
        deleteDocument(guestbook.getCompanyId(), guestbook
.getGuestbookId());
    }

    //Contruye un objeto Document a partir de un Guestbook
    @Override
    protected Document doGetDocument(Guestbook guestbook)
        throws Exception {

        Document document = getBaseModelDocument(CLASS_NAME, guestbook
);

        document.addDate(Field.MODIFIED_DATE, guestbook.
getModifiedDate());

        Locale defaultLocale =
            PortalUtil.getSiteDefaultLocale(guestbook.getGroupId());
        String localizedField = LocalizationUtil.getLocalizedName(
            Field.TITLE, defaultLocale.toString());

        document.addText(localizedField, guestbook.getName());
        return document;
    }

    //Resumen del Documento
    @Override
    protected Summary doGetSummary(
        Document document, Locale locale, String snippet,
        PortletRequest portletRequest, PortletResponse portletResponse)
```

```
{

        Summary summary = createSummary(document);
        summary.setMaxContentLength(200);
        return summary;
    }

    //Reindexan los registros con los datos proporcionados
    @Override
    protected void doReindex(Guestbook guestbook)
        throws Exception {

        Document document = getDocument(guestbook);
        indexWriterHelper.updateDocument(
            getSearchEngineId(), guestbook.getCompanyId(), document,
            isCommitImmediately());
    }
    @Override
    protected void doReindex(String className, long classPK)
        throws Exception {

        Guestbook guestbook = _guestbookLocalService.getGuestbook
(classPK);
        doReindex(guestbook);
    }
    //Reindexa todos los regitros para la instancia actual del portal
Liferay (companyId)
    @Override
    protected void doReindex(String[] ids)
        throws Exception {

        long companyId = GetterUtil.getLong(ids[0]);
        reindexGuestbooks(companyId);
    }
    protected void reindexGuestbooks(long companyId)
      throws PortalException {

      final IndexableActionableDynamicQuery
indexableActionableDynamicQuery =
        _guestbookLocalService.getIndexableActionableDynamicQuery();

      indexableActionableDynamicQuery.setCompanyId(companyId);

      indexableActionableDynamicQuery.setPerformActionMethod(
```

```
        new ActionableDynamicQuery.PerformActionMethod<Guestbook>() {
          @Override
          public void performAction(Guestbook guestbook) {
            try {
              Document document = getDocument(guestbook);
              indexableActionableDynamicQuery.addDocuments(document);
            }
            catch (PortalException pe) {
              if (_log.isWarnEnabled()) {
                _log.warn(
                  "Unable to index guestbook " +
                    guestbook.getGuestbookId(),
                  pe);
              }
            }
          }
        });
      indexableActionableDynamicQuery.setSearchEngineId
(getSearchEngineId());
      indexableActionableDynamicQuery.performActions();
    }
}
```

- Por ultimo se ha de exportar los paquetes en el fichero **bnd.bnd**, para que sean accesibles.

```
Export-Package:
  com.liferay.docs.guestbook.service.permission,\
  com.liferay.docs.guestbook.search
```

11.2. Indexacion de contenidos

Una vez se ha implementado el indexador, se ha de declarar su uso en las funcionalidades del **Service Builder**, para ello se ha de añadir en los métodos de add, update y delete del **-LocalServiceImp** las siguientes anotaciones **com.liferay.portal.kernel.search.Indexable** y **com.liferay.portal.kernel.search.IndexableType**.

add

```
@Indexable(type = IndexableType.REINDEX)
public Guestbook addGuestbook(...)
```

update

```
@Indexable(type = IndexableType.REINDEX)
public Guestbook updateGuestbook(...)
```

delete

```
@Indexable(type = IndexableType.DELETE)
public Guestbook deleteGuestbook(...)
```

11.3. Busqueda de contenidos indexados

Se puede emplear un Portlet existene llamado **Busqueda Web**, donde se deberá de añadir en la configuracion avanzada, el nuevo tipo de datos añadiendolo al JSON de configuracion.

```
"values": [
    "com.liferay.portal.model.User",
    "com.liferay.portlet.bookmarks.model.BookmarksEntry",
    "com.liferay.portlet.bookmarks.model.BookmarksFolder",
    "com.liferay.portlet.blogs.model.BlogsEntry",
    "com.liferay.portlet.documentlibrary.model.DLFileEntry",
    "com.liferay.portlet.documentlibrary.model.DLFolder",
    "com.liferay.portlet.journal.model.JournalArticle",
    "com.liferay.portlet.journal.model.JournalFolder",
    "com.liferay.portlet.messageboards.model.MBMessage",
    "com.liferay.portlet.wiki.model.WikiPage",
    "com.liferay.docs.insult.model.Insult"
],
```

O emplear el API de busquedas indexadas, para ello hay que definir un **Contexto de busqueda**, donde **keywords** son las palabras por las que se desea buscar.

```
SearchContext searchContext = SearchContextFactory.getInstance(request
);
searchContext.setKeywords(keywords);
searchContext.setAttribute("paginationType", "more");
searchContext.setStart(0);
searchContext.setEnd(10);
```

NOTE

Se pueden obtener las Keywords por ejemplo ded un parametro de la request así

```
String keywords = ParamUtil.getString(request, "keywords
");
```

Una vez definido el **Contexto de busqueda**, se emplea junto con el indice.

En este caso se esta haciendo una busqueda de Entry

```
Indexer indexer = IndexerRegistryUtil.getIndexer(Entry.class);

try {
    Hits hits = indexer.search(searchContext);

    List<Entry> entries = new ArrayList<Entry>();

    for (int i = 0; i < hits.getDocs().length; i++) {
            Document doc = hits.doc(i);

            long entryId = GetterUtil
            .getLong(doc.get(Field.ENTRY_CLASS_PK));

            Entry entry = null;

            try {
                    entry = EntryLocalServiceUtil.getEntry(entryId);
            } catch (PortalException pe) {
                    _log.error(pe.getLocalizedMessage());
            } catch (SystemException se) {
                    _log.error(se.getLocalizedMessage());
            }

            entries.add(entry);
    }
}
catch (SearchException se) {
    // handle search exception
}
```

NOTE

La obtencion de la referencia al log que se emplea en el anterior codigo podria ser

```
private static Log _log = LogFactoryUtil.getLog(
"html.guestbookwebportlet.view_search_jsp");
```

Una opcion de invocacion del algoritmo de busqueda, seria a traves de un formulario de busqueda en la vista.

```
<liferay-portlet:renderURL varImpl="searchURL">
    <portlet:param name="mvcPath"
    value="/guestbookwebportlet/view_search.jsp" />
</liferay-portlet:renderURL>

<aui:form action="<%= searchURL %>" method="get" name="fm">
    <liferay-portlet:renderURLParams varImpl="searchURL" />

    <div class="search-form">
        <span class="aui-search-bar">
            <aui:input inlineField="<%= true %>" label=""
            name="keywords" size="30" title="search-entries"
type="text"/>

            <aui:button type="submit" value="search" />
        </span>
    </div>
</aui:form>
```

Es un formulario sencillo que hace uso de la etiqueta **<liferay-portlet:renderURLParams>** para incluir en el formulario como campos ocultos los parametros de la **<liferay-portlet:renderURL>**, en este caso **mvcPath**.

Y para representar los datos obtenidos, un **Search-container**

```
<liferay-ui:search-container delta="10" emptyResultsMessage="no-
entries-were-found"
    total="<%= entries.size() %>">

        <liferay-ui:search-container-results results="<%= entries %>"
/>

        <liferay-ui:search-container-row
className="com.liferay.docs.guestbook.model.Entry"
        keyProperty="entryId" modelVar="entry"
escapedModel="<%=true%>">

            <liferay-ui:search-container-column-text name="guestbook"

value="<%=guestbookMap.get(Long.toString(entry.getGuestbookId()))%>" />

            <liferay-ui:search-container-column-text property="message"
/>

            <liferay-ui:search-container-column-text property="name" />

            <liferay-ui:search-container-column-jsp
                path="/guestbookwebportlet/entry_actions.jsp"
                align="right" />

        </liferay-ui:search-container-row>

        <liferay-ui:search-iterator />
</liferay-ui:search-container>
```

12. Tests

Las pruebas para Liferay vienen marcadas por **JUnit**, **Arquilian** y **Selenium**.

12.1. JUnit

Para añadir JUnit al classpath

```
dependencies {
    testCompile group: "junit", name: "junit", version: "4.12"
}
```

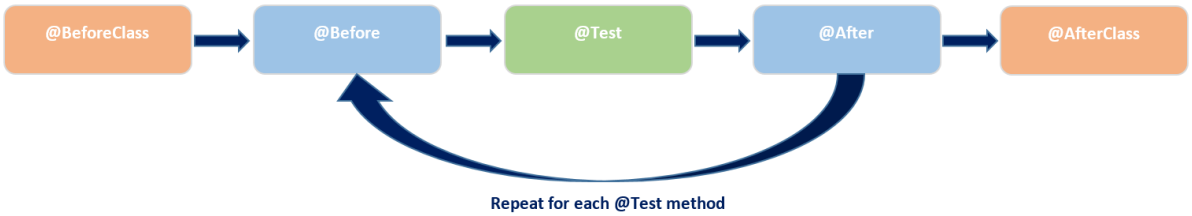
Para ejecutar las pruebas con gradle se ejecuta el comando

```
> ./gradlew test
```

Los resultados se generan en **<directorio del proyecto gradle>\build\test-results\test**

Se proporcionan anotaciones que permiten actuar en las distintas fases del ciclo de vida

- **@Before**: El método de instancia anotado con esta anotación, se ejecutara antes de cada Test de la clase, por tanto tantas veces como métodos de instancia anotados con @Test existan.
- **@After**: El método de instancia anotado con esta anotación, se ejecutara despues de cada Test de la clase, por tanto tantas veces como métodos de instancia anotados con @Test existan.
- **@BeforeClass**: El método estatico anotado con esta anotación, se ejecutara antes de cualquier otro de la clase y solo una vez.
- **@AfterClass**: El método estatico anotado con esta anotación, se ejecutara despues de todos los otros métodos de instancia de la clase y solo una vez.
- **@Test**: El método anotado con esta anotación, representa un Test.
- **@Ignore**: Permite ignorar un método de Test.



12.2. Configuracion para Test Funcionales y de Integracion

- Se ha de configurar Tomcat para soportar JMX, para ello en el fichero **<TomcatDir>/bin/setev.bat** se ha de añadir lo siguiente

Fichero setenv.bat

```
if exist "%CATALINA_HOME%/jre1.6.0_20/win" (
    if not "%JAVA_HOME%" == "" (
        set JAVA_HOME=
    )

    set "JRE_HOME=%CATALINA_HOME%/jre1.6.0_20/win"
)

set "CATALINA_OPTS=%CATALINA_OPTS% -Dfile.encoding=UTF8
-Djava.net.preferIPv4Stack=true
-Dorg.apache.catalina.loader.WebappClassLoader.ENABLE_CLEAR_REFERENCES=
false -Duser.timezone=GMT -Xmx1024m -XX:MaxPermSize=384m"

rem Lineas para dar soporte para JMX al Tomcat

set "JMX_OPTS=-Dcom.sun.management.jmxremote
-Dcom.sun.management.jmxremote.authenticate=false
-Dcom.sun.management.jmxremote.port=8099
-Dcom.sun.management.jmxremote.ssl=false"

set "CATALINA_OPTS=%CATALINA_OPTS% %JMX_OPTS%"
```

- Se han de definir las depedencias de **Arquillian**

```
testIntegrationCompile group: "com.liferay.arquillian", name:
"com.liferay.arquillian.arquillian-container-liferay", version: "1.0.6"

testIntegrationCompile group: "junit", name: "junit", version: "4.12"

testIntegrationCompile group: "log4j", name: "log4j", version: "1.2.17"

testIntegrationCompile group: "org.slf4j", name: "slf4j-log4j12",
version: "1.7.5"

testIntegrationCompile group: "org.jboss.arquillian.graphene", name:
"graphene-webdriver", version: "2.1.0.Final"

testIntegrationCompile group: "org.jboss.arquillian.junit", name:
"arquillian-junit-container", version: "1.1.11.Final"
```

- Se ha de configurar Liferay para que no muestre el wizzard de comienzo, para

ello se ha de definir el fichero **src/testIntegration/resources** con el siguiente contenido

```
#Permite lanzar el navegador directamente
browser.launcher.url=

#Evita lanzar el asistente de configuracion
setup.wizard.enabled=false
```

- Si se va a emplear **Chrome** para los test funcionales, se ha de instalar **Chrome Driver**, que se puede descargar aquí.
- Se ha de definir el fichero **src/testintegration/resources/arquillian.xml**

Fichero arquillian.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<arquillian xmlns="http://jboss.org/schema/arquillian"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://jboss.org/schema/arquillian
http://jboss.org/schema/arquillian/arquillian_1_0.xsd">

        <extension qualifier="webdriver">
            <property name="browser">chrome</property>
            <property name="chrome.binary">C:\Program Files
(x86)\Google\Chrome\Application\chrome.exe</property>
            <property name="chromeDriverBinary"
>D:\utilidades\Selenium\chromedriver\chromedriver-2.40.exe</property>
        </extension>

        <extension qualifier="graphene">
            <property name="url">http://localhost:8080</property>
        </extension>

        <engine>
            <property name="deploymentExportPath">
build/deployments</property>
        </engine>
</arquillian>
```

12.3. Test de Integracion con Arquillian

Se ha de crear una clase en **src/testIntegration/java/<paquete>** con las siguientes

caracteristicas

- Anotada con @**RunWith(Arquillian.class)**
- Un método **static** que retorne un objeto **JavaArchive** obtenido con el API de **ShrinkWrap**, cuyo contenido será un fichero jar con los test y lo que los test necesitan para ejecutarse.
- Un atributo de clase que representa el SUT a probar, injectado por OSGI a través de la anotación @**Inject**.
- Métodos anotados con @**Test** que realicen las pruebas sobre el SUT.

```
@RunWith(Arquillian.class)
public class ArquillianIntegrationTest {

    @Deployment
    public static JavaArchive create() throws Exception {
        final File tempDir = Files.createTempDir();
        String gradlew = "./gradlew";

        String osName = System.getProperty("os.name", "");
        if (osName.toLowerCase().contains("windows")) {
            gradlew = "gradle.bat";
        }

        final ProcessBuilder processBuilder = new ProcessBuilder
(gradlew, "jar", "-Pdir=" + tempDir.getAbsolutePath());

        final Process process = processBuilder.start();

        process.waitFor();

        final File jarFile = new File(tempDir.getAbsolutePath() +
"/com.st.arquillian-1.0.0.jar");

        return ShrinkWrap.createFromZipFile(JavaArchive.class, jarFile
);
    }

    @Test
    public void testAdd() throws IOException, PortalException {
        final long result = _sampleService.mul(5, 3);
        Assert.assertEquals(15, result);
    }

    @Inject
    private ArquillianExampleService _sampleService;
}
```

12.4. Test de Funcionales con Arquillian

Se ha de crear una clase en **src/testIntegration/java/<paquete>** con las siguientes caracteristicas

- Anotada con **@RunWith(Arquillian.class)** y **@RunAsClient**
- Un método **static** que retorne un objeto **JavaArchive** obtenido con el API de **ShrinkWrap**, cuyo contenido será un fichero jar con los test y lo que los test necesitan para ejecutarse.
- Un atributo de clase que representa el navegador de tipo **WebDriver** de **Selenium** anotado con **@Drone**
- Un atributo de clase que representa la URL que se va a consumir, el SUT, de tipo **URL** anotado con **@PortalURL("arquillian_example_portlet"), indicando el nombre del portlet *javax.portlet.name**.
- Métodos anotados con **@Test** que realicen las pruebas sobre el SUT.

NOTE	La anotacion **@FindBy** de **Selenium** se puede emplear para hacer referencia a componentes de la UI.

```
@RunAsClient
@RunWith(Arquillian.class)
public class ArquillianFunctionalTest {

    @Deployment
    public static JavaArchive create() throws Exception {
        final File tempDir = Files.createTempDir();

        String gradlew = "./gradlew";

        String osName = System.getProperty("os.name", "");
        if (osName.toLowerCase().contains("windows")) {
            gradlew = "./gradlew.bat";
        }

        final ProcessBuilder processBuilder = new ProcessBuilder
(gradlew, "jar", "-Pdir=" + tempDir.getAbsolutePath());

        final Process process = processBuilder.start();

        process.waitFor();

        final File jarFile = new File(tempDir.getAbsolutePath() +
"/com.st.arquillian-1.0.0.jar");

        return ShrinkWrap.createFromZipFile(JavaArchive.class, jarFile
);
```

```
    }

    @Test
    public void testAdd()throws InterruptedException, IOException,
PortalException {

        _browser.get(_portlerURL.toExternalForm());

        _firstParameter.clear();
        _firstParameter.sendKeys("6");

        _secondParameter.clear();
        _secondParameter.sendKeys("4");

        _mul.click();

        Thread.sleep(5000);
        Assert.assertEquals("24", _result.getText());
    }

    @Test
    public void testInstallPortlet() throws IOException,
PortalException {
        _browser.get(_portlerURL.toExternalForm());

        final String bodyText = _browser.getPageSource();

        Assert.assertTrue("The portlet is not well deployed",bodyText
.contains("Sample Portlet is working!"));
    }

    @FindBy(css = "button[type=submit]")
    private WebElement _mul;

    @Drone
    private WebDriver _browser;

    @FindBy(css = "input[id$='firstParameter']")
    private WebElement _firstParameter;

    @PortalURL("arquillian_example_portlet")
    private URL _portlerURL;

    @FindBy(css = "span[class='result']")
    private WebElement _result;
```

```
    @FindBy(css = "input[id$='secondParameter']")
    private WebElement _secondParameter;

}
```

13. Logs

Para añadir referencia a los logs en Liferay, se han de incluir las siguientes lineas

```
import com.liferay.portal.kernel.log.Log;
import com.liferay.portal.kernel.log.LogFactoryUtil;

public class MiClase {
    private static Log logger = LogFactoryUtil.getLog(MiClase.class);
}
```

Los niveles de Log aceptados son:

- debug: Información de eventos y aplicaciones útil para la depuración.
- trace: Proporciona más información que la depuración. Este es el nivel de mensaje más detallado.
- info: Eventos de alto nivel.
- warn: Información que podría indicar, pero no necesariamente, un problema.
- error: Errores normales. Este es el nivel de mensaje menos detallado.

13.1. Configuracion

Desde la administracion del portal, en la sección **Control Panel - Configuration - Server Administration - Log Levels**, se puede indicar el nivel de log deseado para cada paquete de modulo instalado en el Portal.

Aparecen muchos paquetes ya registrados, pudiendo dar de alta nuevos con la opcion **Add Category**.