Desarrollo Avanzado con Liferay 7

Victor Herrero Cazurro

Contenidos

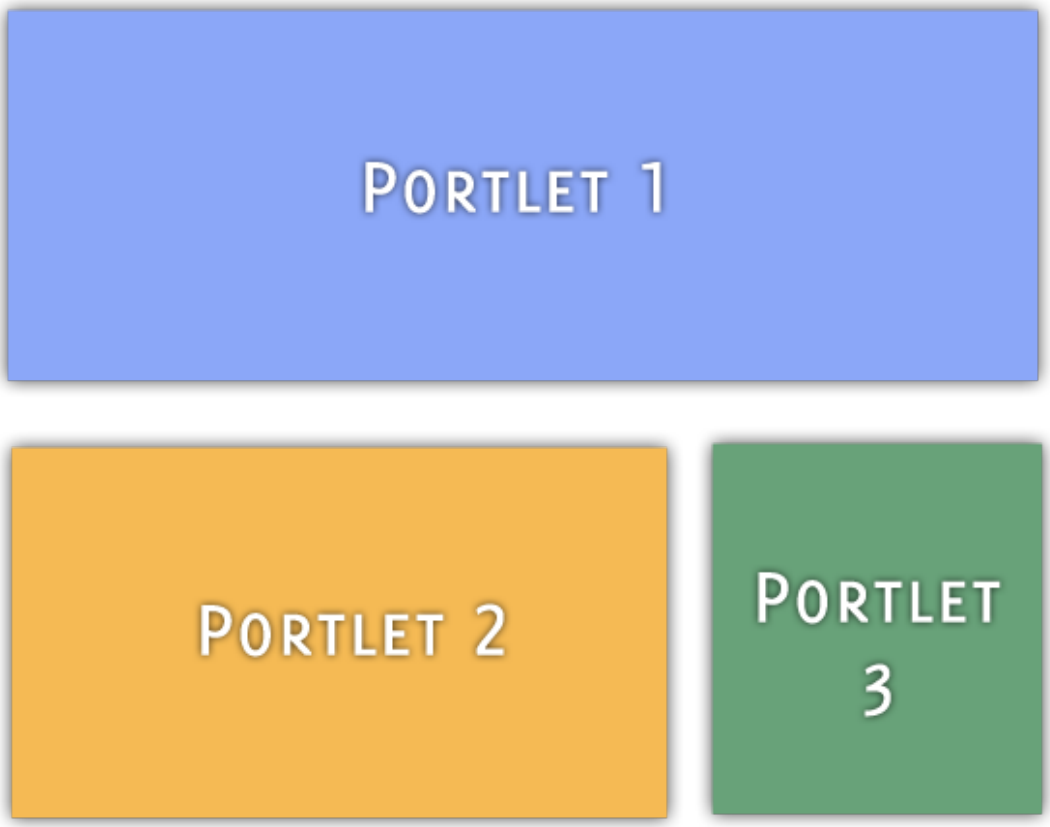
1. Portlets

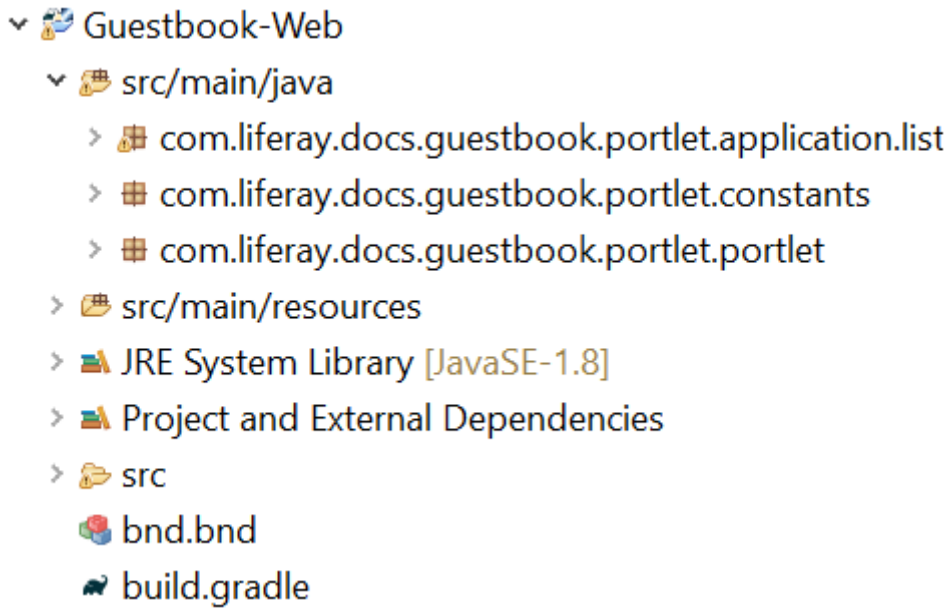
1.1. Introduccion

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.

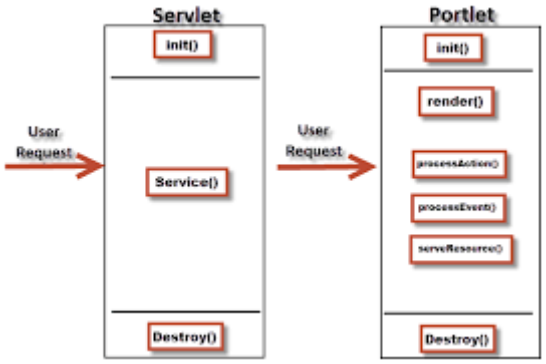


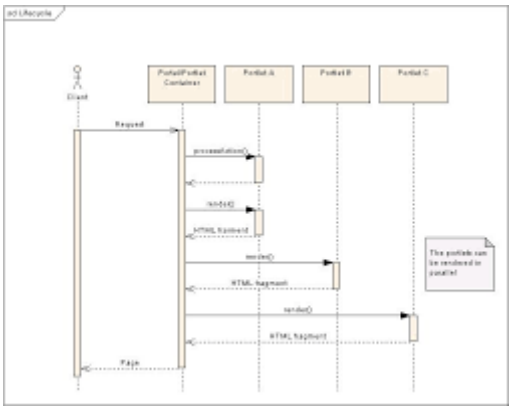


1.2. Ciclo de vida

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**

2.4. Libreria de JSPs AlloyUI

Para emplearla, hay que añadir el taglib a la JSP

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

2.4.1. Formularios

```
<aui:form action="<%= addEntryURL %>" name="<portlet:namespace />fm">
</aui:form>
```

2.4.2. Modelo

Permite definir un Bean a emplear para rellenar los campos del formulario. Se puede emplear para editar con un formulario un dato existente.

Ejemplo de uso, donde al renderizar el formulario, el input **name** *tendra como valor el campo* **name** *del bean* **entry**

```
<aui:form action="<%= addEntryURL %>" name="<portlet:namespace />fm">
    <aui:model-context bean="<%= entry %>" model="<%= Entry.class %>"
/>
    <aui:input name="name"></aui:input>
</aui:form>
```

2.4.3. FieldSet

Agrupacion de campos de formulario

```
<aui:fieldset>
</aui:fieldset>
```

2.4.4. Input

Define un campo del formulario, puede ser de varios tipos: text, textarea, hidden, …¬ para lo cual se configura el atributo **type**

```
<aui:input name="name">
</aui:input>
```

2.4.5. Button Row

Permite definir una fila de botones

```
<aui:button-row>
</aui:button-row>
```

2.4.6. Button

Permite definir un boton de formulario

```
<aui:button-row>
    <aui:button id="useEmailButton" value="Use My Email
Address"></aui:button>
</aui:button-row>
```

2.4.7. Validator

Permite establecer una validacion sobre un campo de entrada, existen varios validadores: email, required, …¬

```
<aui:input name="email" >
    <aui:validator name="email" />
    <aui:validator name="required" />
</aui:input>
```

2.4.8. Layout

Permite definir una distribucion de la pantalla, basada en columnas, indicando dentro de cada columna los elementos que compondrán dicha columna

```
<aui:layout>
    <aui:column>
        <div id="message1-div"></div>
    </aui:column>
</aui:layout>
```

- **columnWidth**: Indica el ancho de la columna
- **first**: Indica que es la primera columna
- **last**: Indica que es la última columna

2.4.9. Nav-Bar

```
<aui:nav-bar>
    <aui:nav>
        <aui:nav-item href="/..." label="view-all" selected='<%="view-
all".equals(toolbarItem) %>' />
        <aui:nav-item dropdown="true" iconCssClass="icon-plus"
                      label="add"
selected='<%="add".equals(toolbarItem) %>'>
            <aui:nav-item href="/..." label="user" />
            <aui:nav-item href="/..." label="organization" />
            <aui:nav-item href="/..." label="group" />
        </aui:nav-item>
        <aui:nav-item href="/..." iconCssClass="icon-download"
                      label="export"
selected='<%="export".equals(toolbarItem) %>' />
    </aui:nav>
</aui:nav-bar>
```

2.4.10. Nav

Permite la definicion de un menu de navegacion.

Se puede representar de varias formas: pestañas, …¬

Representacion en forma de pestañas, empleando la clase CSS **nav-tabs**

```
<aui:nav cssClass="nav-tabs">
</aui:nav>
```

2.4.11. Nav-Item

Cada una de los elementos que componen el menu de navegacion. Estan compuestos por

- **cssClass** ¬ Estilo de pintado, para la representacion en pestañas, se emplea **active**
- **href** ¬ URL que se ha de cargar al seleccionar la opción, es habitual que se cargue siempre la misma ventana, cambiando los datos cargados a traves de parametros asociados al enlace y en caso de representacion con pestañas, el estilo de la pestaña seleccionada.
- **label** ¬ Texto que se asocia a la opcion del menu

Representacion en forma de pestañas, empleando la clase CSS **nav-tabs**

```
<aui:nav cssClass="nav-tabs">

    <portlet:renderURL var="viewPageURL">
        <portlet:param name="mvcPath" value="/html/guestbook/view.jsp"
/>
        <portlet:param name="guestbookId"
            value="<%=String.valueOf(curGuestbook.getGuestbookId())%>"
/>
    </portlet:renderURL>

    <aui:nav-item cssClass="<%=cssClass%>"
href="<%=viewPageURL%>"aui:nav>
```

2.4.12. Script

Permite emplear apis de la libreria JS de AlloyUI.

Se han de definir los modulos a emplear en la propiedad **use**, pudiendo tomar como valores: node, event, aui-char-counter, …¬

Dentro del script existirá una variable **A** que permite el acceso a los apis de los modulos cargados.

```
<aui:script use="node, event">
    var useNameButton = A.one('#useNameButton');
</aui:script>
```

Los modulos disponibles en Liferay se pueden encontrar en **<Liferay portal boundle directory>\tomcat-7.0.62\webapps\ROOT\html\js\aui**

2.5. Libreria Liferay-UI

Para emplearla, hay que añadir el taglib a la JSP

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

2.5.1. Menu de Navegacion Pop-Up

La etiqueta **<liferay-ui:icon-menu>**, permite definir un listado de etiquetas hijas **<liferay-ui:icon>** que definen el menú.

```
<liferay-ui:icon-menu>
    <liferay-ui:icon iconCssClass="icon-user" message="Use"
url="www.liferay.com" />
    <liferay-ui:icon iconCssClass="icon-film" message="Film"
url="www.liferay.com" />
    <liferay-ui:icon iconCssClass="icon-edit" message="Edit"
url="www.liferay.com" />
    <liferay-ui:icon iconCssClass="icon-trash" message="Trash"
url="www.liferay.com" />
</liferay-ui:icon-menu>
```

La etiqueta **<liferay-ui:icon-menu>** dispone entre otros de los siguientes atributos configurables.

- **direction**: Posibles direccion en la que se despliega el menu. Puede valer left, right, up, o down. El valor por defecto es left.
- **maxDisplayItems**: Numero de iconos a mostrar en el area visible, antes de usar croll, el valor por defecto es 15.
- **message**: Texto a mostra en el boton que muestra el menú. El valor por defecto es actions.

- **showArrow**: Establece cuando se muestra una flecha dentro del boton de accion, por defecto es true.
- **useIconCaret**: Establece si cambia la direccion del icono al abrir el menu, por defecto es false.

2.5.2. Panel

Permite definir paneles colapsables que agrupan componentes visuales.

```
<liferay-ui:panel defaultState="closed" extended="<%= false %>"
id="populatePanel" persistState="<%= true %>" title="populate">
</liferay-ui:panel>
```

- **defaultState**: Indica como aparece el panel por defecto
- **persistState**: Indica si se mantendrá el estado último del panel ante recargas de la pagina

2.5.3. UI-Header

Permite definir como se pinta la cabecera del Portlet, lo mas habitual es para poner un boton de volver atras.

```
<portlet:renderURL var="viewURL">
    <portlet:param name="mvcPath" value="/guestbookwebportlet/view.jsp"
/>
</portlet:renderURL>

<liferay-ui:header backURL="<%= viewURL.toString() %>" title="search"
/>
```

2.5.4. Search-container

Permite representar una coleccion de datos como una tabla, donde se disponen de distintas tipologias de columnas predefinidas

Principalmente hay que definir las siguientes etiquetas:

- **<liferay-ui:search-container-results>** ¬ Indica los registros a representar, referencia a un objeto coleccion de java.

- **<liferay-ui:search-container-row>** ¬ Incluye la definicion de columnas, referencia al tipo de dato a representar y a la variable a emplear en la iteracion por todos los elementos.
- **<liferay-ui:search-iterator />** ¬ Indica que se ha de establecer la iteracion sobre la coleccion de objetos java.

```
<jsp:useBean id="entries" class="java.util.ArrayList" scope="request"/>

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

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

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

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

NOTE	La propiedad **escapedModel** del la etiqueta **<liferay-ui:search-container-row>** garantiza que los texto mostrados son escaados

Cuando se emplean los servicios de **Service Builder**, la definicion del **Search-Container** será similar a

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

results="<%=EntryLocalServiceUtil.getEntries(scopeGroupId.longValue(),
                    guestbookId, searchContainer.getStart(),
                    searchContainer.getEnd())%>" />
    ...
</liferay-ui:search-container>
```

Search-container-column-text

Tipo de columna que representa un texto

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

Search-container-column-jsp

Tipo de columna que permite incluir un JSP como columna, es util para modularizar la vista y reutilizar partes de la JSP.

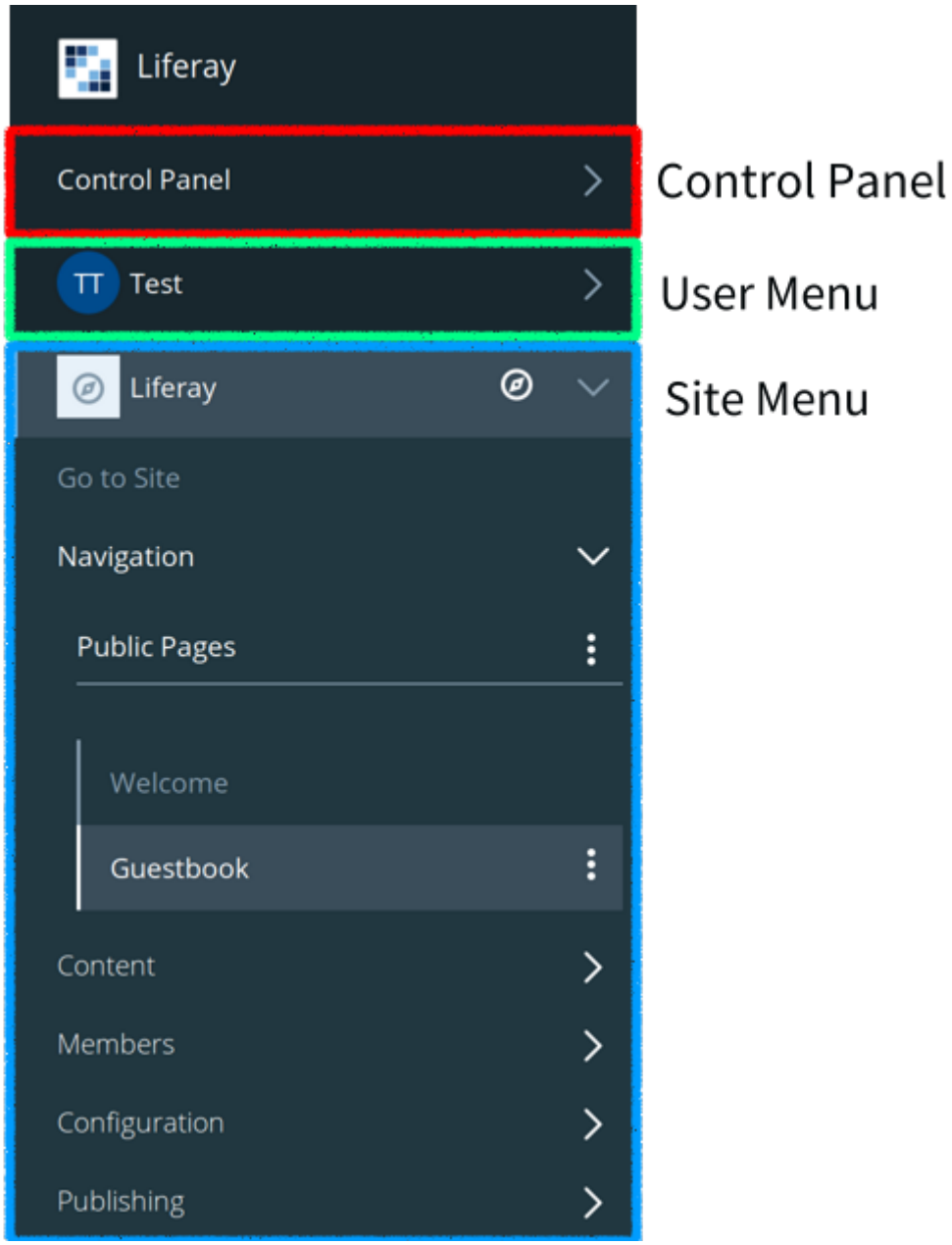
```
<liferay-ui:search-container-column-jsp
path="/guestbookwebportlet/entry_actions.jsp" align="right" />
```

3. Portlet Administrativo

La definicion de un Portlet Admnistrativo ha variado con respecto a anteriores versiones de Liferay, ahora se considera una nueva opcion del **Panel de Administracion/Product Menu**.

El **Panel de Administracion** se divide en tres secciones

- **Panel de Control/Control Panel**.
- **Menu de Usuario/User Menu**.
- **Menu de Sitios/Sites Menu**.



Cada sección, es una **Categoria del Panel** y las categorias estan formadas por **panel apps**.

Se proporciona una plantilla para su creación **panel-app**.

```
> blade create -t panel-app -p com.liferay.docs.guestbook.portlet -c
GuestbookAdmin Guestbook-Web
```

Se obtienen varios elementos con la plantilla

- Una clase basada en **MVCPortlet**
- Nueva dependencia en **build.gradle**

- Una clase basada en **BasePanelApp**

3.1. MVCPortlet

La que implementa la funcionalidad del Portlet Administrativo.

```
@Component(
    immediate = true,
    property = {
        "com.liferay.portlet.add-default-resource=true",
        "com.liferay.portlet.display-category=category.hidden",
        "com.liferay.portlet.layout-cacheable=true",
        "com.liferay.portlet.private-request-attributes=false",
        "com.liferay.portlet.private-session-attributes=false",
        "com.liferay.portlet.render-weight=50",
        "com.liferay.portlet.use-default-template=true",
        "javax.portlet.display-name=Guestbook-Web Portlet",
        "javax.portlet.expiration-cache=0",
        "javax.portlet.init-param.template-path=/",
        "javax.portlet.init-param.view-template=/view.jsp",
        "javax.portlet.name=" + GuestbookAdminPortletPortletKeys
.GuestbookAdminPortlet,
        "javax.portlet.resource-bundle=content.Language",
        "javax.portlet.security-role-ref=power-user,user",
        "javax.portlet.supports.mime-type=text/html"
    },
    service = Portlet.class
)
public class GuestbookAdminPortlet extends MVCPortlet {
}
```

A reseñar la propiedad **com.liferay.portlet.display-category** con valor **category.hidden** que hace invisible el portlet para la creacion de paginas del portal y la homogenizacion del valor de la propiedad **javax.portlet.name** con la referencia a una constante de la clase **GuestbookAdminPortletPortletKeys**, que se usará posteriormente.

Si se desea implementar alguna funcionalidad en el Portlet, las tipicas serian añadir, editar y borrar, se deberán añadir **Acciones** al Portlet que delegen en la capa de **Service Builder**, para ello se ha de obtener la referencia al servicio con OSGI.

```
private GuestbookLocalService _guestbookLocalService;
```

```
@Reference(unbind = "-")
protected void setGuestbookService(GuestbookLocalService
guestbookLocalService) {
    _guestbookLocalService = guestbookLocalService;
}

public void addGuestbook(ActionRequest request, ActionResponse
response)
    throws PortalException {

    ServiceContext serviceContext = ServiceContextFactory.getInstance(
        Guestbook.class.getName(), request);

    String name = ParamUtil.getString(request, "name");

    try {
        _guestbookLocalService.addGuestbook(
            serviceContext.getUserId(), name, serviceContext);
    }
    catch (PortalException pe) {

        Logger.getLogger(GuestbookAdminPortlet.class.getName()).log(
            Level.SEVERE, null, pe);

        response.setRenderParameter(
            "mvcPath", "/guestbookadminportlet/edit_guestbook.jsp");
    }
}

public void updateGuestbook(ActionRequest request, ActionResponse
response)
    throws PortalException {

    ServiceContext serviceContext = ServiceContextFactory.getInstance(
        Guestbook.class.getName(), request);

    String name = ParamUtil.getString(request, "name");
    long guestbookId = ParamUtil.getLong(request, "guestbookId");

    try {
        _guestbookLocalService.updateGuestbook(
            serviceContext.getUserId(), guestbookId, name,
serviceContext);
```

```
    } catch (PortalException pe) {

        Logger.getLogger(GuestbookAdminPortlet.class.getName()).log(
            Level.SEVERE, null, pe);

        response.setRenderParameter(
            "mvcPath", "/guestbookadminportlet/edit_guestbook.jsp");
    }
}

public void deleteGuestbook(ActionRequest request, ActionResponse
response)
    throws PortalException {

    ServiceContext serviceContext = ServiceContextFactory.getInstance(
        Guestbook.class.getName(), request);

    long guestbookId = ParamUtil.getLong(request, "guestbookId");

    try {
        _guestbookLocalService.deleteGuestbook(guestbookId,
serviceContext);
    }
    catch (PortalException pe) {

        Logger.getLogger(GuestbookAdminPortlet.class.getName()).log(
            Level.SEVERE, null, pe);
    }
}
```

En las vistas tambien se añadirá la referencia a la capa de servicios, pero en este caso de forma tradicional empleando la clase **GuestbookLocalServiceUtil**, para el caso de la vista **view.jsp** que será la empleada por el modo de renderizacion por defecto se tendrá

```
<%@include file="../init.jsp"%>

<liferay-ui:search-container
    total="<%=
GuestbookLocalServiceUtil.getGuestbooksCount(scopeGroupId) %>">
    <liferay-ui:search-container-results
        results="<%=
GuestbookLocalServiceUtil.getGuestbooks(scopeGroupId,
            searchContainer.getStart(), searchContainer.getEnd()) %>"
/>

    <liferay-ui:search-container-row
        className="com.liferay.docs.guestbook.model.Guestbook"
modelVar="guestbook">

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

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

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

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

<aui:button-row cssClass="guestbook-admin-buttons">
    <portlet:renderURL var="addGuestbookURL">
        <portlet:param name="mvcPath"
            value="/guestbookadminportlet/edit_guestbook.jsp" />
        <portlet:param name="redirect" value="<%= "currentURL" %>" />
    </portlet:renderURL>

    <aui:button onClick="<%= addGuestbookURL.toString() %>"
        value="Add Guestbook" />
</aui:button-row>
```

En este caso, se esta haciendo uso de la etiqueta **<liferay-ui:search-container-column-jsp>** que permite incluir una JSP como columna del **search container**, la cual tendra el siguiente contenido

```
<%@include file="../init.jsp"%>

<%
    String mvcPath = ParamUtil.getString(request, "mvcPath");

    ResultRow row = (ResultRow) request
                    .getAttribute("SEARCH_CONTAINER_RESULT_ROW");

    Guestbook guestbook = (Guestbook) row.getObject();
%>

<liferay-ui:icon-menu>

    <portlet:renderURL var="editURL">
        <portlet:param name="guestbookId"
            value="<%=String.valueOf(guestbook.getGuestbookId()) %>" />
        <portlet:param name="mvcPath"
            value="/guestbookadminportlet/edit_guestbook.jsp" />
    </portlet:renderURL>

    <liferay-ui:icon image="edit" message="Edit"
            url="<%=editURL.toString() %>" />

    <portlet:actionURL name="deleteGuestbook" var="deleteURL">
            <portlet:param name="guestbookId"
                value="<%= String.valueOf(guestbook.getGuestbookId())
%>" />
    </portlet:actionURL>

    <liferay-ui:icon-delete url="<%=deleteURL.toString() %>" />

</liferay-ui:icon-menu>
```

Donde se incluye un menu con botones tipo icono que conectan con las acciones del Portlet.

Finalizando las Vistas, se define una ultima vista para la edicion e los datos de **Guestbook**, denominada **edit_guestbook.jsp**, donde:

- Se inicializa un objeto **Guestbook** para rellenar los campos del formulario en caso de que le llegue a la vista.
- Se indica con la etiqueta **<aui:model-context>** que será ese Bean e empleado para rellenar los campos del formulario.

- Se incluye el namespace en el nombre del formulario con **<portlet:namespace />**
- Se envia el Id del Bean editado como campo oculto.

```
<%@include file = "../init.jsp" %>

<%
	long guestbookId = ParamUtil.getLong(request, "guestbookId");

	Guestbook guestbook = null;

	if (guestbookId > 0) {
		guestbook =
GuestbookLocalServiceUtil.getGuestbook(guestbookId);
	}
%>

<portlet:renderURL var="viewURL">
	<portlet:param name="mvcPath"
value="/guestbookadminportlet/view.jsp" />
</portlet:renderURL>

<portlet:actionURL name='<%= guestbook == null ? "addGuestbook" :
"updateGuestbook" %>' var="editGuestbookURL" />

<aui:form action="<%= editGuestbookURL %>" name="<portlet:namespace
/>fm">

	<aui:model-context bean="<%= guestbook %>" model="<%=
Guestbook.class %>" />

	<aui:input type="hidden" name="guestbookId"
	    value='<%= guestbook == null ? "" :
guestbook.getGuestbookId() %>' />

	<aui:fieldset>
	     <aui:input name="name" />
	</aui:fieldset>

	<aui:button-row>
	     <aui:button type="submit" />
	     <aui:button onClick="<%= viewURL %>" type="cancel"  />
	</aui:button-row>
</aui:form>
```

3.2. Dependencias

Necesarias para compilar el resto de clases necesarias

```
compileOnly group: "com.liferay", name:
"com.liferay.application.list.api", version: "2.0.0"
```

3.3. BasePanelApp

Se encarga de crear el nuevo **panel-app** dentro del **Panel Administrativo**, haciendo referencia al Portlet.

```
@Component(
    immediate = true,
    property = {
        "panel.app.order:Integer=100",
        "panel.category.key=" + GuestbookAdminPortletPanelCategoryKeys
.CONTROL_PANEL_CATEGORY
    },
    service = PanelApp.class
)
public class GuestbookAdminPortletPanelApp extends BasePanelApp {
    @Override
    public String getPortletId() {
        return GuestbookAdminPortletPortletKeys.GuestbookAdminPortlet;
    }
    @Override
    @Reference(
        target = "(javax.portlet.name=" +
GuestbookAdminPortletPortletKeys.GuestbookAdminPortlet + ")",
        unbind = "-"
    )
    public void setPortlet(Portlet portlet) {
        super.setPortlet(portlet);
    }
}
```

En este caso se puede reseñar, la propiedad **panel.category.key** con valor **GuestbookAdminPortletPanelCategoryKeys.CONTROL_PANEL_CATEGORY**, que crea una nueva categoria para el Portlet, de no quererse crear una nueva cateogoria y desear ponerla en la categoria **Contenido del Sitio**, se deberá cambiar

el valor anterior por **PanelCategoryKeys.SITE_ADMINISTRATION_CONTENT**.

La propiedad **panel.app.order**, indica en que orden se muestra.

4. Service Builder

4.1. Introduccion

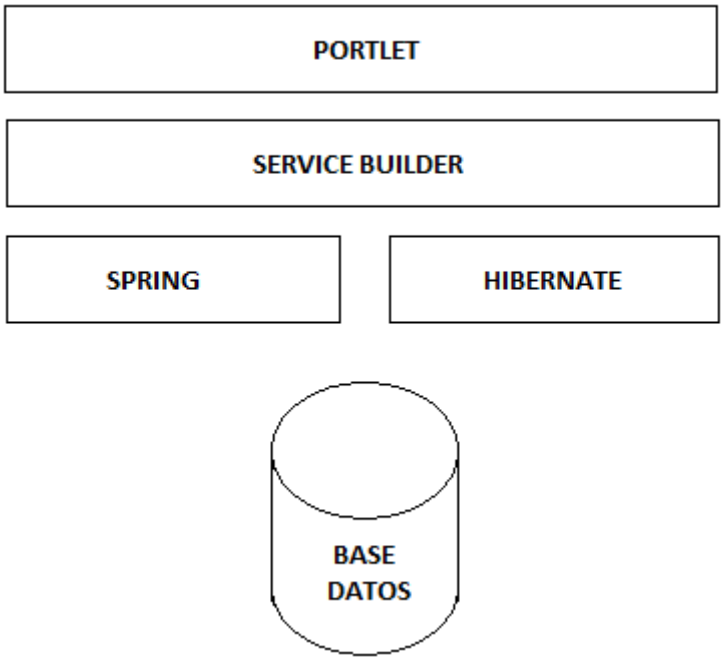
Permite definir una capa de Persistencia y de Servicio de forma automatica, considerando las funcionalidades mas basicas.

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 tienen 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. 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.

5.1. 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**

5.2. 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**

6. Customizacion

7. 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**

7.1. 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.

7.2. 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**

7.3. 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="color"
    label="true"/>
```

7.4. 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.

7.4.1. 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**

7.4.2. 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'
            }
        ]
    }
);
```

7.5. Incluir una entidad nueva en el ExpandoPortlet

Para que se muestre en el Portlet **Custom Fields** como una opcion mas, habrá que definir un nuevo componente OSGi de tipo **CustomAttributesDisplay**, partiendo de la clase **BaseCustomAttributesDisplay**, que implmente el método **getClassName** rotornando el tipo de la entidad que va a ser extendida con **Custom Fileds**.

```
@Component(
        immediate = true,
        property = "javax.portlet.name=" + GuestbookPortletKeys
.GUESTBOOK,
        service = CustomAttributesDisplay.class
    )
public class GuestbookCustomAttributesDisplay extends
BaseCustomAttributesDisplay {

    @Override
    public String getClassName() {
        return Guestbook.class.getName();
    }

}
```

8. Extensiones de Modulos Existentes

Con Liferay se proporcionan preinstalados una serie de **Aplicaciones** que hacen referencia a **Servicios** los cuales pueden ser extendidos/sobrescritos.

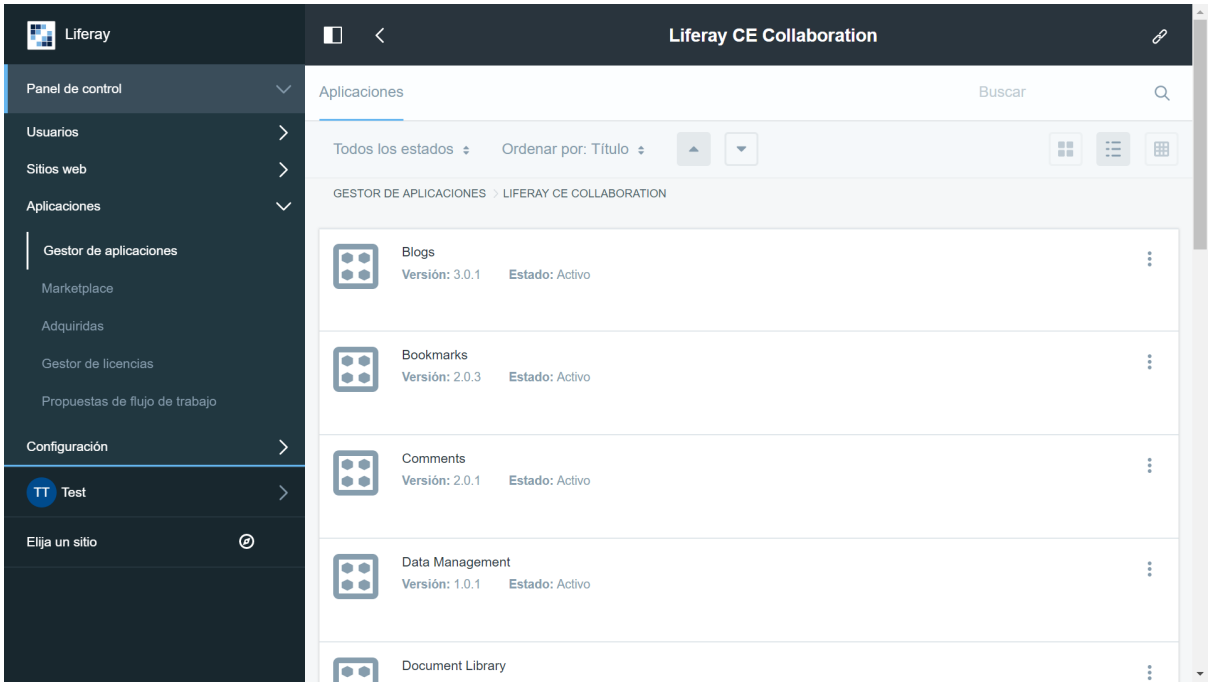
Para poder sobreescribirlos, lo primero es identificar el **Servicio** a sobreescribir, para ello, se puede emplear el **Gestor de Aplicaciones**, al cual se puede acceder desde **Panel de control ¬ Aplicaciones ¬ Gestor de Aplicaciones**, el cual nos permite

En esta funcionalidad, se muestran las distintas **Suites** preinstaladas

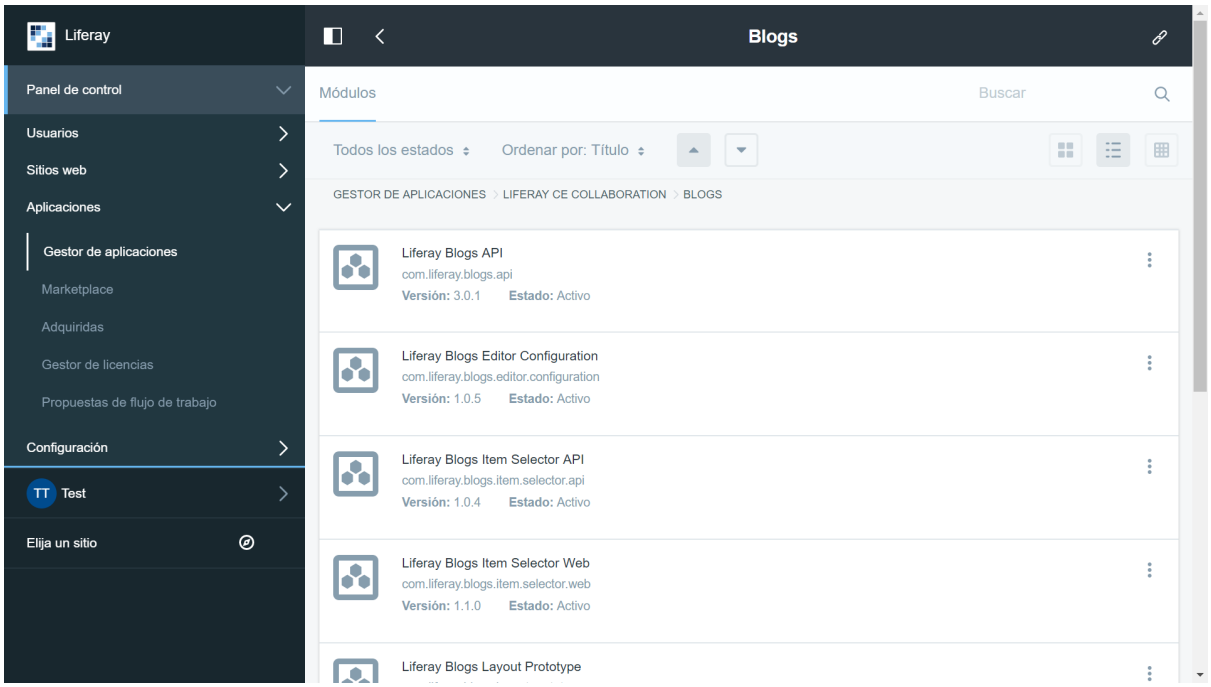
- Liferay CE Collaboration
- Liferay CE Forms and Workflow
- Liferay CE Foundation
- Liferay CE Static
- Liferay CE Sync Connector
- Liferay CE Web Experience
- Independent Modules
- Liferay CE Static
- Liferay Marketplace

NOTE	Los Modulos propios desarrollados, saldran en la seccion **Independent Modules**

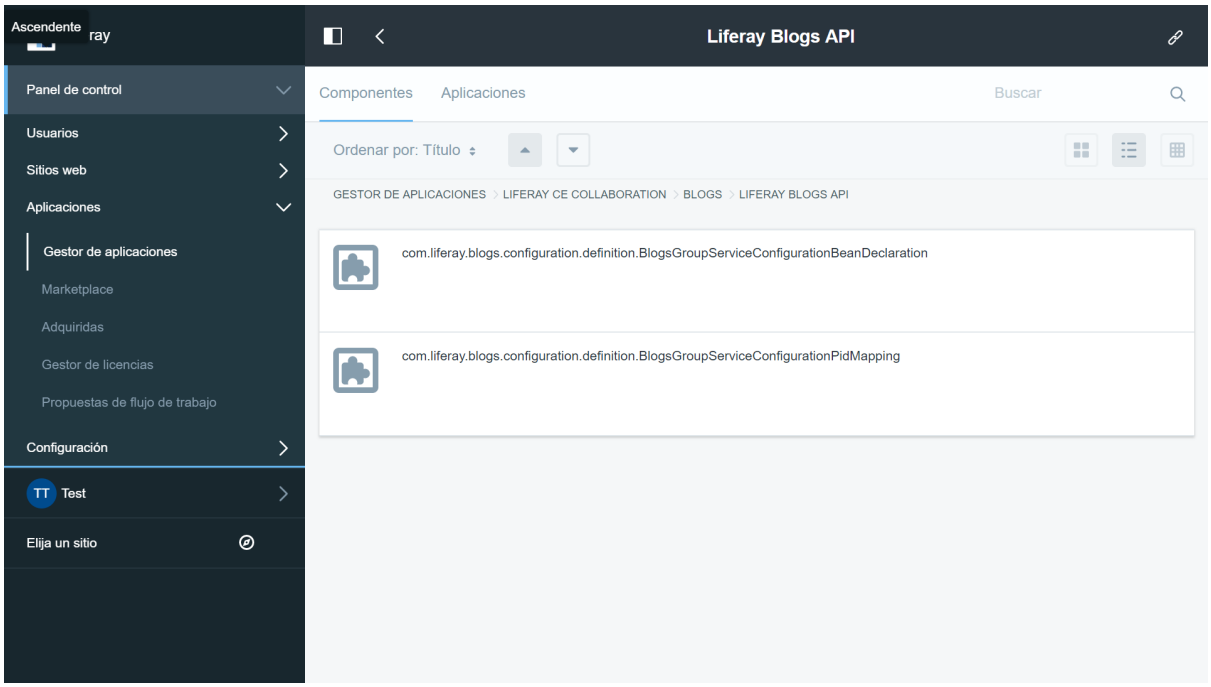
Si se accede a la **Suite**, se obtienen las **Aplicaciones** que la componen.



Y accediendo a las **Aplicaciones**, se obtienen los **Modulos** que componen cada **Aplicación**.

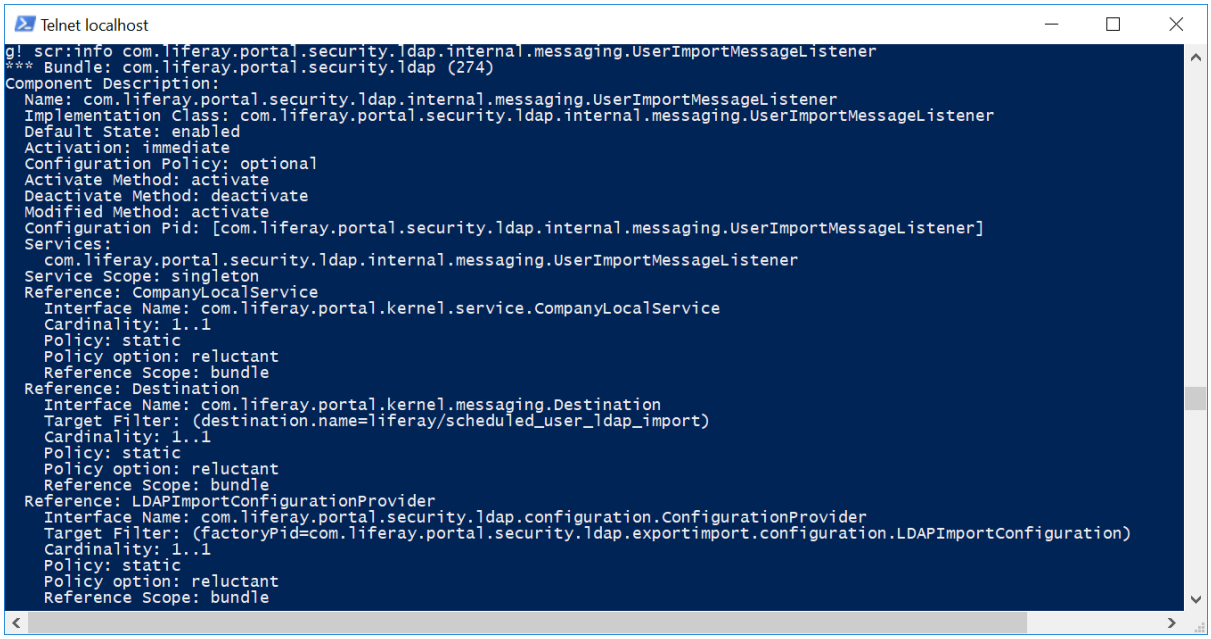


Finalmente, accediendo al **Modulo**, se obtienen las implementaciones de todos las **Funcionalidades** que lo componen



Una vez obtenida la **Funcionalidad**, se puede obtener el listado de los **Servicios** a los que hace **referencia**, entre los cuales estará el que se desea extender, con la consola **Gogo Shell**.

```
g! > scr:info "nombre de la clase que implementa la funcionalidad que
hace referencia al servicio a extender"

g! > scr:info
com.liferay.blogs.configuration.definition.BlogsGroupServiceConfigurati
onBeanDeclaration
```



De toda la informacion obtenida, las interesantes son las secciones **Reference**,

donde se tiene la **Interface** del servicio.

```
Reference: CompanyLocalService
  Interface Name: com.liferay.portal.kernel.service.CompanyLocalService
  Cardinality: 1..1
  Policy: static
  Policy option: reluctant
  Reference Scope: bundle
```

8.1. Implementacion de la extension

Una vez localizada la interface que se quiere extender, se ha de crear un nuevo modulo, que necesita tener la dependencia con el jar que define la **Interface**.

```
dependencies {
    compileOnly group: "grupo", name: "nombre", version: "version"
}
```

Y definir una nueva clase que implemente la interface, anotandola con **@Component** y haciendo referencia a la interface a extender.

```
@Component(
    immediate = true,
    service = SomeService.class
)
public class CustomServiceImpl implements SomeService {}
```

Si se quisiera tener referencia al servicio existente, se podría obtener por inyeccion con la siguiente configuracion

```
    @Reference  (
        unbind = "-",
        target =
"(component.name=override.my.service.reference.service.impl.SomeService
Impl)"
    )
    private SomeService _defaultService;
```

8.2. Configuracion de la extension

Una vez definida la extension, es necesario activarla, para ello se ha de definir un fichero de configuracion que se ha de llamar como el componente que hace uso del servicio, nombre canonico de la clase, con extension **.config** o **.cfg**

```
override.my.service.reference.portlet.OverrideMyServiceReferencePortlet
.config
```

Donde el contenido de este fichero **.config** ha de ser

```
_someService.target="(component.name\=overriding.service.reference.serv
ice.CustomServiceImpl)"
```

Y el del fichero **.cfg**

```
_someService.target=(component.name=overriding.service.reference.servic
e.CustomServiceImpl)
```

Donde _**someService** es el atributo dentro del componente que hace uso del servicio que referencia al servicio y el valor de **component.name** es la clase que sobrescribe el servicio.

Se puede encontrar una referencia a como componer estos ficheros aquí

Y se ha de copiar en la carpeta **<Liferay_Home>/osgi/configs**

9. 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**.

10. Message Bus

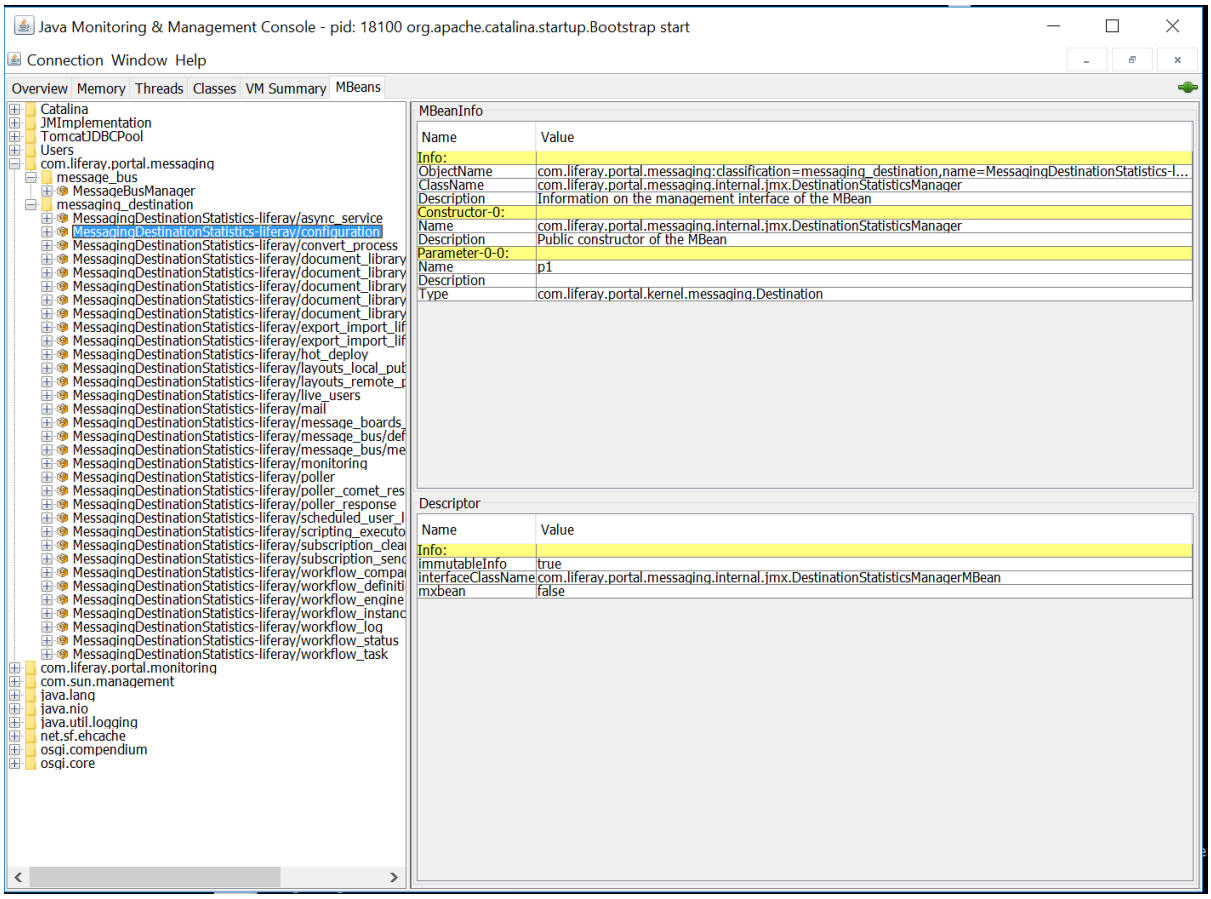
Permite el procesa de tareas de forma asincrona.

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)

10.1. 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**.

10.1.1. 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).

10.1.2. Creacion de un Destino

Añadir la dependencia

```
compileOnly group: "org.osgi", name: "org.osgi.core", version: "6.0.0"
```

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.

10.2. 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;
}
```

10.3. 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()
);
    }
}
```

10.4. 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);
    }
}
```

10.5. 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;
}
```

10.5.1. 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;
}
```

11. 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();
```

12. Javascript

12.1. Javascript Liferay

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**.

12.1.1. 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.')
}
```

12.1.2. Language

Permite obtener textos traducidos en el Portal.

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

12.1.3. 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_')
```

12.1.4. 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

12.1.5. 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.

12.1.6. 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;
    }
);
```

12.1.7. 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.

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);
    }
);
```

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.

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);}
);
```

12.2. Modulos

Para emplear un modulo, hay que declararlo sobre el objeto **AUI()**

```
<aui:script use="aui-char-counter">
AUI().use(
  function(A) {
  }
);
</aui:script>
```

Existen multiples modulos que se pueden emplear en el API de AlloyUI.

12.2.1. CharCounter

El modulo **aui-char-counter** permite contar los caracteres que restan por introducir en un input, definiendo un máximo.

Se necesita definir:

- Un campo de entrada de texto, de emplear el API de AUI, tener en cuenta que se introduce de forma automatica el **Namespace**.
- Un contenedor para incluir el numero de caracteres calculado

```
<aui:input id="message" type="textarea" name="message">
    <aui:validator name="required" errorMessage="Please enter a
message." />
</aui:input>

<div id="counterContainer"><p><span id="counter"></span> character(s)
remaining</p></div>

<aui:script use="aui-char-counter">
AUI().use(
  function(A) {
    new A.CharCounter(
      {
        counter: '#counter',
        input: '#<portlet:namespace />message',
        maxLength: 140
      }
    );
  }
);
</aui:script>
```

12.2.2. DOM

El modulo **node** permite el manejo del arbol DOM de componentes HTML que se genera en los navegadores.

Permite realizar principalmente tareas de selección, como

- one
- get
- all
- select
- after
- next

```
<c:if test="<%= themeDisplay.isSignedIn() %>">
    <aui:script use="node">

        var name = A.one('#<portlet:namespace/>name');

        name.val('<%= user.getFullName() %>');

        var email = A.one('#<portlet:namespace/>email');

        email.val('<%= user.getEmailAddress() %>');

    </aui:script>
</c:if>
```

Y tambien de validación y edicion del arbol DOM

- hasChildNodes
- append

```
<aui:button-row>
        <aui:button
            id="generateMessagesButton"
            value="Generate Sample Messages">
        </aui:button>
</aui:button-row>

<div id="messages">
    <aui:layout>
        <aui:column>
            <div id="message1-div"></div>
        </aui:column>
    </aui:layout>
</div>

<aui:script use="node, event">

    var generateMessagesButton = A.one('#generateMessagesButton');

    var message1Div = A.one('#message1-div');

    generateMessagesButton.on('click', function(event) {

            var entryMessages = [
```

```
                    'Amazing!',
                    'Be careful!',
                    'Best wishes!',
                    'Bravo!',
                    'Congratulations!',
                    'Great job!',
                    'Have fun!',
                    "How's it going?",
                    'You did it!',
                    "Wow!"
            ];

            if (message1Div.hasChildNodes()) {
                    message1Div.get('children').remove(true);
            }

            var rand1 = Math.floor(Math.random() *
entryMessages.length);

            message1Div.append(
                '<p class="message" id="message1">' +
entryMessages[rand1] +
                '</p><p id="use-message1">' +
                '<input class="btn" onclick="useMessage1();"
type="button" value="Use Message" /></p>');

            entryMessages.splice(rand1, 1);
    });
</aui:script>
```

Tareas de lectura de atributos de los nodos

- html
- val

```
option.val();
```

- attr

```
option.attr('value');
```

12.2.3. Event

El modulo **Event** permite definir escuchadores de eventos que se produzcan sobre los componentes del arbol DOM

```
<c:if test="<%= themeDisplay.isSignedIn() %>">

    <aui:script use="node, event">

        var useNameButton = A.one('#useNameButton');

        useNameButton.on('click', function(event) {
                var name = A.one('#<portlet:namespace/>name');

                name.val('<%= user.getFullName() %>');
        });

        var useEmailButton = A.one('#useEmailButton');

        useEmailButton.on('click', function(event) {
                var email = A.one('#<portlet:namespace/>email');

                email.val('<%= user.getEmailAddress() %>');
        });
    </aui:script>
</c:if>
```

12.2.4. Carousel

Permite visualizar un conjunto de imagenes, que se muestran de forma dinamica.

Para emplearlo, se han de seguir una serie de pasos:

- Incluir las imagenes en el proyecto, tipicamente en el directorio **img** del contenido web, para proyectos Maven **src/main/webapp/img**
- Definir en el HTML una estructura similar a la siguiente, con tantos bloques **div** como imagenes compongan el **Carousel**

```
<div id="myCarousel">
  <div id="image1"></div>
  <div id="image2"></div>
</div>
```

- Añadir el script que controla el **Carousel**

```
<aui:script>
    AUI().use('aui-carousel',function(Y) {
        new Y.Carousel(
            {
            contentBox: '#myCarousel',
            height: 250,
            width: 700
            }
        ).render();
    });
</aui:script>
```

- Definir estilos CSS que optimicen la visualización de las imagenes, asi como la referencia a las imagenes a mostrar.

```
div.carousel-item
{
    width: 700px;
    height: 250px;
    opacity: 100;
}

#image1
{
    background-image: url("../img/YourImageFile1.jpg");
}

#image2
{
    background-image: url("../img/YourImageFile2.jpg");
}
```

12.2.5. Modulos Javascript

Permite modularizar las funcionalidades de JS.

Lieray ofrece compatibilidad con Modulos de:

- ES2015
- AMD
- AUI (YUI)

12.3. 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']
    }
);
```

12.4. 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>
```

12.5. 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>
```

12.6. 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>
```

13. Web Service

Liferay permite la creación de Servicios Web empleando los estandares JAX-WS (Soap) y JAX-RS (rest).

13.1. JAX-RS

Se proporciona una plantilla para la creación de estos proyectos, denominada **rest**

```
> blade b create -t rest -p <nombre paquete> <nombre proyecto>
```

Con esta plantilla se obtiene un servicio web JAX-RS, que escucha peticiones:

- [GET] http://localhost:8080/o/<nombre proyecto>/greetings
- [GET] http://localhost:8080/o/<nombre proyecto>/morning
- [GET] http://localhost:8080/o/<nombre proyecto>/morning/{name}

El proyecto creado consta de las siguientes configuraciones

- Fichero **build.gradle**

```
dependencies {
    compileOnly group: "javax.ws.rs", name: "javax.ws.rs-api", version:
"2.0.1"
    compileOnly group: "org.osgi", name:
"org.osgi.service.component.annotations", version: "1.3.0"
}
```

- Fichero **bnd.bnd**

```
Bundle-Name: EjemploServicioRest
Bundle-SymbolicName: com.curso.liferay.rest
Bundle-Version: 1.0.0
Liferay-Configuration-Path: /configuration
```

Donde es importante la configuracion **Liferay-Configuration-Path**, que indica el path del classpath donde se situan dos ficheros **properties**, que configuran:

- **Publisher**: Define el Path a partir del cual se publicarán los servicios, por defecto se toma el nombre del proyecto y la forma de autenticación.

com.liferay.portal.remote.cxf.common.configuration.CXFEndpointPublisherConfiguration-cxf.properties

```
contextPath=/EjemploServicioRest
authVerifierProperties=auth.verifier.BasicAuthHeaderAuthVerifier.urls.i
ncludes=*
```

- **Extender**: Extiende la definición anterior añadiendo la referencia a las clases que implementan los servicios.

com.liferay.portal.remote.rest.extender.configuration.RestExtenderConfiguration-rest.properties

```
contextPaths=/EjemploServicioRest
jaxRsServiceFilterStrings=(component.name=com.curso.liferay.rest.applic
ation.EjemploServicioRestApplication)
```

- Fichero **com.curso.liferay.rest.application.EjemploServicioRestApplication.java**

```
@Component(
    immediate = true,
    service = Application.class
)

@ApplicationPath("/greetings")
public class JAXRSApplication extends Application {

    public Set<Object> getSingletons() {
        return Collections.<Object>singleton(this);
    }

    @GET
    @Produces("text/plain")
    public String working() {
        return "It works!";
    }

    @GET
    @Path("/morning")
    @Produces("text/plain")
    public String hello() {
        return "Good morning!";
    }

    @GET
    @Path("/morning/{name}")
    @Produces("text/plain")
    public String morning(
        @PathParam("name") String name,
        @QueryParam("drink") String drink) {

        String greeting = "Good Morning " + name;

        if (drink != null) {
            greeting += ". Would you like some " + drink + "?";
        }

        return greeting;
    }
}
```

13.2. JAX-WS

Similar al anterior caso, salvo que cambian

- Nueva dependencia en el fichero **build.gradle**

```
dependencies {
    compileOnly group: "javax.ws.rs", name: "javax.ws.rs-api", version:
"2.0.1"
    compileOnly group: "org.osgi", name:
"org.osgi.service.component.annotations", version: "1.3.0"
    compileOnly group: "org.osgi", name: "org.osgi.core", version:
"6.0.0"
}
```

- El nombre del fichero de configuracion del extender

com.liferay.portal.remote.soap.extender.configuration.SoapExtenderConfiguration-soap.properties

```
contextPaths=/soap
jaxWsServiceFilterStrings=(component.name=com.curso.liferay.soap.Calcul
ator)
```

- La implementacion del servicio

```
import javax.jws.WebMethod;
import javax.jws.WebService;
import org.osgi.service.component.annotations.Component;


@Component(
    immediate = true,
    property = "jaxws=true",
    service = Calculator.class
)


@WebService
public class Calculator {

   @WebMethod
   public int divide(int a, int b) {
       return a / b;
   }


   @WebMethod
   public int multiply(int a, int b) {
       return a * b;
   }


   @WebMethod
   public int subtract(int a, int b) {
       return a - b;
   }


   @WebMethod
   public int sum(int a, int b) {
       return a + b;
   }
}
```

14. 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'
```

14.1. 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 que 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 necesarios 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
```

14.2. 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(...)
```

14.3. 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 de 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>
```

Si se trabaja directamente sobre los resultados del indice, se trabaja siempre con un objeto **Document**

```
Hits hits = indexer.search(searchContext);
<liferay-ui:search-container delta="10" emptyResultsMessage="no-
entries-were-found"
    total="<%= hits.getLength() %>">

        <liferay-ui:search-container-results results="<%= hits.toList()
%>" />

        <liferay-ui:search-container-row
            className="com.liferay.portal.kernel.search.Document"
            escapedModel="<%= false %>"
            keyProperty="UID"
            modelVar="doc"
            stringKey="<%= true %>">

                <liferay-ui:search-container-column-text name="Titulo"
                value="<%=doc.get(Field.TITLE)%>" />

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

15. Tests

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

15.1. Pruebas Unitarias

Una prueba unitaria es una forma de probar el correcto funcionamiento de un módulo de código. Esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado.

En las pruebas unitarias se prueban clases concretas, no conjuntos de clases, es decir una prueba unitaria prueba la funcionalidad de un método de una clase suponiendo que los objetos que emplea realizan sus tareas de forma correcta.

15.2. JUnit

Framework para pruebas.

Paquetes junit.* (JUnit 3) y org.junit.* (JUnit 4) Embedido en Eclipse (JUnit 3 y 4), eclipse proporciona la creación de Junit Test Case (caso de prueba) y Junit Test Suite (conjunto de casos de pueba).

JUnit 4 admite timeout, excepciones esperadas, tests ignorables, test parametrizados, …¬

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**

15.2.1. Test Suites

Conjunto de pruebas a ejecutar de forma conjunta.

```
@RunWith(Suite.class)
@SuiteClasses({C1Test.class, C2Test.class})
public class TestSuite{

}
```

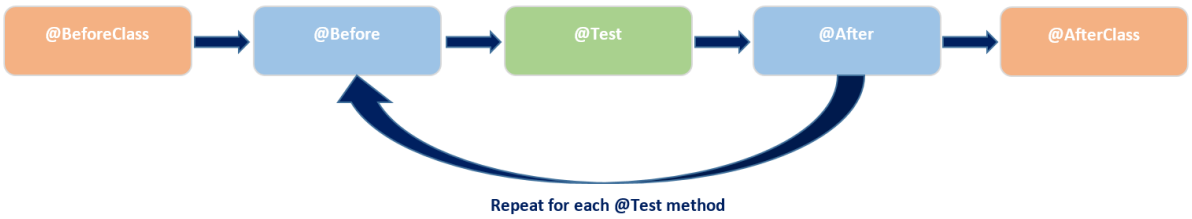
15.2.2. Ciclo de vida de los Test

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.



15.2.3. Probando el lanzamiento de excepciones

La anotacion @Test, permite realizar pruebas, donde el resultado esperado sea una **Excepcion**.

```
@Test(expected=InvalidIngresoException.class)
public void comprobamosQueLanzamosException() throws
InvalidIngresoException{
}
```

15.2.4. Probando restricciones temporales

La anotacion @Test, permite realizar pruebas, donde el tiempo transcurrido en la ejecución esta limitado por el requisito.

```
@Test(timeout=12000)
public void testDeRendimiento() {
}
```

15.2.5. Test parametrizados

El API incorpora un **Runner**, que permite la ejecución repetida de Test, cada vez con unos datos de prueba, para lo cual hay que

- Anotar la clase con **@RunWith**

```
@RunWith(Parameterized.class)
```

- Crear un método publico estatico que retorne un **Array de Arrays**, anotado con **@Parameters**

```
@Parameters
public static Collection<Object[]> data() {}
```

NOTE

Puede ser interesante emplear la clase de utilidad **Arrays**

```
Arrays.asList(new Object[][] {{},{},{}});
```

- Crear Atributos de clase de la misma tipologia que los elementos de los Arrays.
- Constructor que establezca los atributos.

15.2.6. Asertos

Los asertos, son métodos estaticos del API, que permiten realizar validaciones, para poder comprobar que los datos obtenidos estan dentro del rango esperado.

Algunos de los asertos que se proporcionan son:

- assertEquals
- assertFalse
- assertTrue
- assertNotNull
- assertNull
- assertNotSame
- assertSame
- fail

NOTE

Implementar una clase que obtenega los impuestos según los ingresos siguiendo las siguientes reglas:

- Ingreso ¬ 8000 no paga impuestos.
- 8000 < Ingreso ¬ 15000 paga 8% de impuestos.
- 15000 < Ingreso ¬ 20000 paga 10% de impuestos.
- 20000 < Ingreso ¬ 25000 paga 15% de impuestos.
- 25000 < Ingreso paga 19.5% de impuestos.

Creamos una clase con un método calcularImpuestosPorIngresos, que recibiendo una cantidad double como parámetro, que son los ingresos de una persona, retornará los impuestos que ha de pagar dicha persona (double).

15.3. Mockito

Para poder crear un buen conjunto de pruebas unitarias, es necesario centrarse exclusivamente en la unidad (normalmente será un metodo de una clase concreto) a testear, para ello se pueden simular, con **Mocks** el resto de clases involucradas, de esta manera se crean test unitarios potentes que permiten detectar los errores allí donde se producen y no en dependencias del supuesto código probado.

Mockito es una herramienta que permite generar **Mocks** dinámicos. Estos pueden ser de clases concretas o de interfaces. Esta parte de la generación de las pruebas, no se centra en la validación de los resultados, sino en los que han de retornar aquellos componentes de los que depende la clase probada.

La creación de pruebas con Mockito se divide en tres fases

- **Stubbing**: Definición del comportamiento de los Mock ante unos datos concretos.
- **Invocación**: Utilización de los Mock, al interaccionar la clase que se esta probando con ellos.
- **Validación**: Validación del uso de los Mock.

Se pueden definir los **Mock** con

- La anotacion @Mock aplicada sobre un atributo de clase.

```
@Mock
private IUserDAO mockUserDao;
```

De emplearse las anotaciones, se ha de ejecutar la siguiente sentencia para que se procesen dichas anotaciones y se generen los objetos **Mock**

```
MockitoAnnotations.initMocks(testClass);
```

O bien emplear un **Runner** especifico de Mockito en la clase de Test que emplee Mockito, el **MockitoJUnitRunner**

```
@RunWith(MockitoJUnitRunner.class)
```

- O con el método estatico **mock**.

```
private IDataSesionUserDAO mockDataSesionUserDao = mock
(IDataSesionUserDAO.class);
```

15.3.1. Stubing

Se persigue definir comportamientos del **Mock**, para ello se emplean los métodos estaticos de la clase **org.mockito.Mockito**, que son

- atLeast
- atMost
- atLeastOnce
- doNothing
- doReturn
- doThrow
- when
- inOrder
- never
- only

- verify
- mock

Y de la clase **org.mockito.Matchers**, que son

- any
- anyString
- anyObject
- contains
- endsWith
- startsWith
- eq
- isA
- isNull
- isNotNull

Algunos ejemplos de definicion de comportamientos del Mock

```
when(mockUserDao.getUser(validUser.getId())).thenReturn(validUser);

when(mockUserDao.getUser(invalidUser.getId())).thenReturn(null);

when(mockDataSesionUserDao.deleteDataSesion((User) eq(null), anyString
())).thenThrow(new OperationNotSupportedException());

when(mockDataSesionUserDao.updateDataSesion(eq(validUser), eq(validId),
anyObject())).thenReturn(true);

when(mockDataSesionUserDao.updateDataSesion(eq(validUser), eq(
invalidId), anyObject())).thenThrow(new OperationNotSupportedException
());

when(mockDataSesionUserDao.updateDataSesion((User) eq(null), anyString
(), anyObject())).thenThrow(new OperationNotSupportedException());
```

Por defecto todos los métodos que devuelven valores de un mock devuelven null, una colección vacía o el tipo de dato primitivo apropiado, salvo que se defina un comportamiento distinto.

15.3.2. Verificación

Se puede verificar el orden en el que se han ejecutado los métodos del **Mock**, pudiendo llegar a diferenciar el orden de invocación de un mismo método por los parametros enviados.

En este ejemplo se esta verificando que el orden de ejecucion de los métodos **getUser** *del mock* **mockUserDao**, *se ejecuta antes que el método* **deleteDataSesion** *del mock* **mockDataSesionUserDao**

```
ordered = inOrder(mockUserDao, mockDataSesionUserDao);
ordered.verify(mockUserDao).getUser(validUser.getId());
ordered.verify(mockDataSesionUserDao).deleteDataSesion(validUser,
validId);
```

Tambien se puede verificar el numero de veces que se ha invocado una funcionalidad

En este ejemplo se verifica que el método **someMethod** *del* **mock** *no se ejecuta nunca, que el método* **someMethod(int)** *se ejecuta 1 sola vez y que el método* **someMethod(string)** *se ejecuta 2 veces.*

```
verify(mock, never()).someMethod();
verify(mock, only()).someMethod(2);
verify(mock, times(2)).someMethod("some arg");
```

15.4. Selenium

Paquete de herramientas para automatizar pruebas de aplicaciones Web en distintas plataformas.

La documentación la podremos obtener aquí

Las herramientas que componen el paquete son

- Selenium IDE.
- Selenium Remote Control (RC) o selenium 1.
- Selenium WebDriver o selenium 2.

15.4.1. Selenium IDE

Se trata de un plugin de Firefox, que nos permitirá grabar y reproducir una macro con una prueba funcional, la cual podremos repetir las veces que deseemos. Se puede descargar aquí

Las acciones que se realizan en la navegación mientras se graba la macro, se traducen en comandos.

La macro por defecto se guarda en HTML, aunque también se puede obtener como java, c#, Python, …¬

También se podrán insertar validaciones y no solo acciones sobre la pagina, aunque las validaciones son mas faciles de escribir en el código generado (java, c#, …¬)

Una vez grabada la macro, el HTML que se genera tiene una tabla con 3 columnas:

- Comando de Selenium.
- Primer parámetro requerido
- Segundo parámetro opcional

Los comandos de selenium se dividen en tres tipos:

- Acciones– Acciones sobre el navegador.
- Almacenamiento– Almacenamiento en variables de valores intermedios.
- Aserciones– Verificaciones del estado esperado del navegador.

Los comandos de navegación mas habituales son:

- **open**: abre una página empleando la URL.
- **click/clickAndWait**: simula la acción de click, y opcionalmente espera a que una nueva pagina se cargue.
- **waitForPageToLoad**: para la ejecución hasta que la pagina esperada es cargada. Es llamada por defecto automáticamente cuando se invoca clickAndWait.
- **waitForElementPresent**: para la ejecución hasta que el UIElement esperado, esta definido por un tag HTML presente en la pagina.
- **chooseCancelOnNextConfirmation**: Predispone a seleccionar en la próxima ventana de confirmación el botón de Cancel.

Los comandos de almacenamiento mas habituales son:

- **store**: Almacena en la variable el valor.
- **storeElementPresent**: Almacenara True o False, dependiendo de si encuentra el UI Element.
- **storeText**: Almacena el texto encontrado. Es usado para localizar un texto en un lugar de la pagina especifico.

Los comandos de verificación mas habituales son:

- **verifyTitle/assertTitle**: verifica que el titulo de la pagina es el esperado.
- **verifyTextPresent**: verifica que el texto esperado esta en alguna parte de la pagina.
- **verifyElementPresent**: verifica que un UI element esperado, esta definido como tag HTML en la presente pagina.
- **verifyText**: verifica si el texto esperado y su tag HTML estan presentes en la pagina.
- **assertAlert**: verifica si sale un alert con el texto esperado.
- **assertConfirmation**: verifica si sale una ventana de confirmacion con el texto esperado.

15.4.2. Selenium WebDriver

Es el motor de pruebas automatizadas de Selenium, se encarga de arrancar un navegador que responde a las ordenes del Test, provocando la ejecución de la macro grabada.

No todas las versiones de Firefox son compatibles con **Selenium WebDriver**, se puede encontrar mas información aquí o aquí

Para descargar versiones antiguas de Firefox, se puede hacer desde aquí

NOTE	Se puede probar con la version de selenium 2.52.0 y Firefox 45.

Para la dependencia del proyecto con selenium webdriver, añadir

```
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>2.19.0</version>
</dependency>
```

El codigo obtenido de la macro grabada con Selenium IDE, tendra las siguientes sentencias

- La creación del objeto que representa la interaccion con el navegador

FirefoxDriver driver = new FirefoxDriver();

- La peticion

driver.get(baseUrl + "/05-Servidor/");

15.5. 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 -Xmx2048m -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>
```

15.6. 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;
}
```

15.7. 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;

}
```

16. 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.

16.1. 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**.

17. Temas

17.1. Temas

Los Temas permiten configurar el modo en el que el Portal se muestra. Son una combinacion de

- CSS
- JavaScript
- HTML
- Plantillas FreeMarker.
- XML

Se ofrecen varias herramientas para trabajar con los temas

- Theme Builder Gradle Plugin
- The Liferay Theme Generator
- IDE
- Blade CLI's Theme Template

Si solo se quiere cambiar parcialmente un tema, se puede optar por contruir un **themelet**, que es una pieza reusable de tema.

Liferay proporciona dos temas bases

- unstyled
- styled: hereda del anterior añadiendo algun estilo extra.

17.1.1. Liferay Theme Generator

Herramienta que se basa en otras herramientas como

- Node.js
- Npm (Gestor de librerias para Node)
- Yeoman (Generador de estructuras de proyectos)
- gulp (Herramienta de contruccion tipo Maven o Gradle)

NOTE	Actualmente esta en desarrollo, por lo que no se garantiza su funcionamiento

Instalacion

Instalar **node.js**

Definir el fichero **/Users/[username]/.npmrc** con contenido

```
prefix=/Users/[username]/.npm-packages
```

Definir la varaible de entorno **NPM_PACKAGES** con valor

```
NPM_PACKAGES=/Users/[username]/.npm-packages
```

Definir la variable de entorno **NODE_PATH** con valor

```
NODE_PATH=/Users/[username]\.npm-packages\node_modules
```

Añadir a la variable **PATH** del sistema la variable de entorno **NPM_PACKAGES** donde se instalaran los modulos de Node que se instalen de forma global.

```
PATH=%PATH%;%NPM_PACKAGES%
```

Instalar las modulos de Node **Yeoman** y **Gulp** de forma global (-g) para el usuario, así las tendremos disponibles en todos los proyectos de Temas con los que se trabajen.

```
npm install -g yo gulp
```

Por ultimo instalar la herramienta **Liferay Theme Generator**

```
npm install -g generator-liferay-theme
```

Esta herramienta, instala en cada tema generado, una herramienta llamada **Liferay Theme** cuya documentacion se puede encontrar aquí.

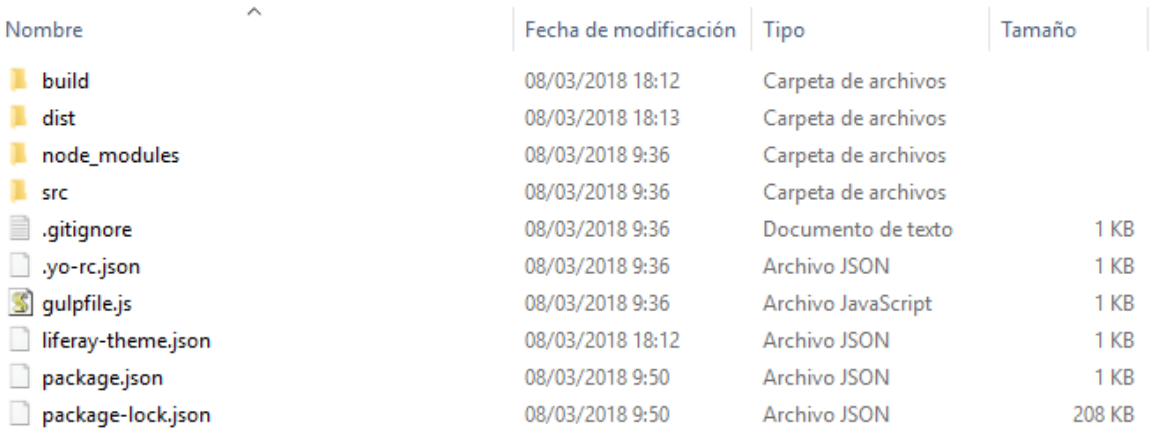
17.1.2. Creacion de un Tema

La creacion de una tema se hace con la herramienta **yeoman**, en este caso con el

comando

```
> yo liferay-theme
```

Una vez lanzado el comando y rellenados los datos que pide, se lanzará de forma automatica el comando npm install, que instalará sobre el proyecto las librerias de npm necesarias, dejando una estructura de proyecto como la siguiente



Nombre	Fecha de modificación	Tipo	Tamaño
build	08/03/2018 18:12	Carpeta de archivos	
dist	08/03/2018 18:13	Carpeta de archivos	
node_modules	08/03/2018 9:36	Carpeta de archivos	
src	08/03/2018 9:36	Carpeta de archivos	
.gitignore	08/03/2018 9:36	Documento de texto	1 KB
.yo-rc.json	08/03/2018 9:36	Archivo JSON	1 KB
gulpfile.js	08/03/2018 9:36	Archivo JavaScript	1 KB
liferay-theme.json	08/03/2018 18:12	Archivo JSON	1 KB
package.json	08/03/2018 9:50	Archivo JSON	1 KB
package-lock.json	08/03/2018 9:50	Archivo JSON	208 KB

Dentro de los ficheros generados destacan

- **package.json**: Fichero donde se describen las librerias de npm necesarias para el proyecto, con este fichero basta ejecutar en el proyecto **npm install** para que se descarguen todas las librerias necesarias para el proyecto, así como el tema padre, lenguaje de plantillas, etc.
- **gulpfile.js**: Fichero donde se define la tarea de Gulp **liferayThemeTasks**
- **liferay-theme.json**: Contiene la configuracion de despliegue en Liferay, paths de la instalacion.

La construccion del tema se realiza con un comando de **Gulp**

```
> gulp build
```

Con la construccion se generan en la carpeta **build**, todos los ficheros que componen el tema, los personalizados y los obtenidos por herencia.

Nombre	Fecha de modificación	Tipo
css	08/03/2018 18:13	Carpeta de archivos
images	08/03/2018 18:12	Carpeta de archivos
js	08/03/2018 18:12	Carpeta de archivos
templates	08/03/2018 18:12	Carpeta de archivos
WEB-INF	08/03/2018 18:12	Carpeta de archivos

El espliegue del tema se realiza de nuevo con un comando de **Gulp**

```
> gulp deploy
```

Se porporciona una forma de despliegue rapido, pensada para editar el tema en caliente

```
> gulp watch
```

Con el comando **extend**, se puede camiar de quien hereda el Tema, aunque actualmente solo se puede elegir entre **_styled** y **_unstyled**.

```
> gulp extend
```

Si se quiere extender del tema **classic**, la forma de hacerlo es copiar los ficheros del original, que se pueden obtener de aquí, en el nuevo tema y modificarlos. Se ha de mantener la herencia de **_styled**, pero la forma mas facil, es con el siguiente comando

```
> gulp kickstart
```

17.1.3. Creacion de un Tema con el IDE

El procedimineto es similar a versiones anteriores, ya que se basa en una estructura de **War** y en las herramientas de contruccion **Maven** o **Gradle**.

El proceso pasa por la creación de un nuevo **Liferay Module Project** con la plantilla **Theme**.

Una vez generado la estructura del proyecto será

Solo se incluye el _**custom.scss**, el resto de ficheros se consiguen al lanzar el comando de **Gradle/Maven buildTheme**

Despues de lanzar el comando en la carpeta **/build/buildTheme** del proyecto se tienen todos los ficheros que componen el tema padre

Para modificar cualquiera de ellos, basta con copiarlo a la carpeta **src/main/webapp**, siguiendo la misma estructura que tienen en la carpeta **/build/buildTheme**.

Si se desean configurar los valores por defecto del plugin de gradle, se han de indicar de la siguiente manera

```
buildTheme{
    parentName "_unstyled"
}
```

17.1.4. Plantillas

La distribucion del Tema, viene marcada por las plantillas, de las cuales se proporcionan las siguientes por defecto

- init_custom.ftl ¬ Fichero para la definición de variables a emplear en el resto de plantillas
- init.ftl ¬ Fichero con las variables que define Liferay como, $theme_display, $portletDisplay, $portletURLFactory y $portalUtil, que son empleadas en el resto de plantillas.
- navigation.ftl ¬ Menú principal de navegación.
- portal_normal.ftl ¬ Estructura general HTML de todas las paginas del portal.
- portlet.ftl ¬ Estructura HTML de cada uno de los portlets. Básicamente es un section con su encabezado, el menú de opciones de cada portlet y el contenido del mismo.

Analizando el contenido de **portal_normal.ftl**, se encuentran las siguientes sentencias

Esta sentencia permite incluir el fichero **init.ftl**

```
<#include init />
```

Esta sentencia permite incluir el fichero **<Liferay Workspace>\bundles\tomcat-8.0.32\webapps\ROOT\html\common\themes\top_head.jsp**

```
<@liferay_util["include"] page=top_head_include />
```

Esta sentencia permite incluir el fichero **<Liferay Workspace>\bundles\tomcat-8.0.32\webapps\ROOT\html\common\themes\body_top.jsp**

```
<@liferay_util["include"] page=body_top_include />
```

Esta sentencia permite incluir el fichero **<Liferay Workspace>\bundles\tomcat-8.0.32\webapps\ROOT\html\common\themes\body_bottom.jsp**

```
<@liferay_util["include"] page=body_bottom_include />
```

Esta sentencia permite incluir el fichero **<Liferay Workspace>\bundles\tomcat-8.0.32\webapps\ROOT\html\common\themes\bottom_head.jsp**

```
<@liferay_util["include"] page=bottom_include />
```

Esta sentencia permite incluir los Portlets, se aprecia como se hace referencia al fichero **portlet.ftl**

```
<section id="content">
    <h1 class="hide-accessible">${the_title}</h1>

    <nav id="breadcrumbs">
        <@liferay.breadcrumbs />
    </nav>

    <#if selectable>
        <@liferay_util["include"] page=content_include />
    <#else>
        ${portletDisplay.recycle()}

        ${portletDisplay.setTitle(the_title)}

        <@liferay_theme["wrap-portlet"] page="portlet.ftl">
            <@liferay_util["include"] page=content_include />
        </@>
    </#if>
</section>
```

17.1.5. Estilos

Para modificar los estilos, se dispone de dos ficheros css.

- main.css ¬ Importación de todos los estilos, se podría añadir uno nuevo
- _custom.scss (custom.css) ¬ Fichero vacío, que se importa en ultimo lugar, destinado a incluir los estilos personalizados.

17.1.6. Javascript

En los Temas se define un único fichero javascript **main.js**, que permite definir codigo a ejecutar en tres momentos de la carga de las paginas:

- AUI().ready ¬ Se ejecuta cuando se termina de carga la pagina
- Liferay.Portlet.ready ¬ Se ejecuta cada vez que se cargue un portlet de la página actual.
- Liferay.on('allPortletsReady') ¬ Se ejecutará después de que todos los portlets de la página estén cargados.

Si se desea añadir un nuevo js al Tema, este se ha de añadir al fichero **<Liferay**

Workspace>\bundles\tomcat-8.0.32\webapps\ROOT\html\common\themes\top_js-ext.jspf, teniendo en cuenta que es una JSP.

Por defecto liferay minimiza los ficheros js y css para que sea mas rápido su carga, esto se puede cambiar con propiedades del **portal.properties**

- javascript.fast.load=false
- theme.css.fast.load=false
- minifier.enabled=false

17.1.7. Actualizar un Tema de la 6.2

Se dispone del comando **upgrade**, que permite transformar un Tema de la version 6.2 en uno de la 7.0

```
> gulp upgrade
```

Y el subcomando **revert** para deshacerlo

```
> gulp upgrade:revert
```

17.2. Themelet

Son pequeñas piezas de codigo reusables y extensibles, compuestas por:

- CSS
- javascript
- freemarker templates.

La diferencia entre un Tema y un Themelet, es que mientras que los temas requieren múltiples componentes, ya que definen completamente como un Sitio se va a mostrar, un Themelet solo requiere los archivos que desea extender.

Los Themelets son modulares, por lo que pueden ser aplicados a distintos temas.

La creacion de una themelet se hace con la herramienta **yeoman** con el comando

```
> yo liferay-theme:themelet
```

Una vez creado el Themelet, este se ha de instalar de forma global como paquete de Node.js, para ello basta con crear un link simbolico a la carpeta del proyecto en la carpeta **<directorio del usuario>\.npm-packages\node_modules\<nombre del themelet>**.

Se puede realizar con **npm**, situandose dentro de la carpeta donde se ha creado el themelet y lanzando el siguiente comando

```
npm link
```

Este comando puede dar problemas si el themelet esta en otra unidad distinta a **C:**, por lo que se puede lanzar el comando de windows

```
mklink /D C:\Users\Victor\.npm-packages\node_modules\simple-themelet
D:\GitLab\Liferay\workspaces\WorkspaceBladeTemas\themes\simple-themelet
```

Una vez definido un Themelet, para poder emplearlo en un tema, se ha de lanzar el siguiente comando en la raiz del tema que empleará el Themelet

```
gulp extend
```

Los Themelet, se pueden obtener de dos fuentes, del repositorio local del usuario (global para todos los proyectos) o de los publicados en la web.

Al finalizar la referencia al Themelet, el fichero **package.json** deberia tener el siguiente contenido

fichero package.json con referencia a un themelet descargado de la web de forma local al proyecto y almacenado en la carpeta **node-modules** *del proyecto*

```
{
    "name": "tema-resource-importer-yeoman-theme",
    "version": "1.0.0",
    "main": "package.json",
    "keywords": [
        "liferay-theme"
    ],
    "liferayTheme": {
        "baseTheme": "styled",
        "screenshot": "",
        "rubySass": false,
        "templateLanguage": "ftl",
        "version": "7.0",
        "themeletDependencies": {
            "lfr-product-menu-animation-themelet": {
                "liferayTheme": {
                    "themelet": true,
                    "version": "7.0"
                },
                "name": "lfr-product-menu-animation-themelet",
                "version": "0.0.5"
            }
        }
    },
    "devDependencies": {
        "gulp": "^3.8.10",
        "liferay-theme-tasks": "^1.4.2",
        "liferay-theme-deps-7.0": "~1.2.1"
    },
    "publishConfig": {
        "tag": "7_0_x"
    },
    "dependencies": {
        "lfr-product-menu-animation-themelet": "0.0.5"
    }
}
```

Se ha de tener especial cuidado con el contenido del fichero **_custom.scss** de la carpeta **src/css**, que suele ser el fichero que mas habitualmente se modifica, ya que deberá tener al menos este contenido, ya que es el contenido que permite inyectar los contenidos de los **Themelets**

```
/* Use these inject tags to dynamically create imports for
themelet styles. You can place them where ever you like in this file.
*/

/* inject:imports */

/* endinject */

/* This file allows you to override default styles in one central
location for easier upgrade and maintenance. */
```

Por ejemplo para el ejemplo anterior, cuando se lanza el comando **deploy**, se acaba generando en dicho fichero la siguiente referencia al **.scss** del themelet.

```
/* Use these inject tags to dynamically create imports for
themelet styles. You can place them where ever you like in this file.
*/

/* inject:imports */
@import "../themelets/lfr-product-menu-animation-
themelet/css/_custom.scss";
/* endinject */

/* This file allows you to override default styles in one central
location for easier upgrade and maintenance. */
```

17.3. Importar Recursos junto con el Tema

Cuando se desarrollan Temas para su venta, es necesario poder mostrar como el tema queda aplicado a un Portal, para ello es necesario que el Portal tenga algo de contenido, con **Resource Importer** se consigue que estos contenidos acompañen a los Temas.

Hay que tener en cuenta que es algo deseable unicamente para desarrollo o para mostrar un ejemplo, no para producción, donde habrá que deshabilitar esta caracteristica, dado que los recursos de ejemplo no tendrán cabida en el sitio de produccion.

Desde la version 7 **Resources importer** es un modulo de OSGi, que se despliega junto con el tema por defecto, esto es así porque en el fichero **/src/WEB-INF/liferay-plugin-package.properties** viene establecida la siguiente propiedad a **true**.

```
resources-importer-developer-mode-enabled=true
```

Esta propiedad se encarga de recrear el **Sitio** o la **Plantilla de Sitio** sobre la que se aplica el Tema, volviendo a generar los recursos para el **Sitio** o los **Sitios** a los que afecte la **Plantilla de Sitio**, lo cual es peligroso.

Para importar directamente los recursos en un sitio, se puede indicar en el fichero **/src/WEB-INF/liferay-plugin-package.properties** con

```
resources-importer-target-class-
name=com.liferay.portal.kernel.model.Group

resources-importer-target-value=[site-name]
```

Esta configuracion no es muy recomendable por la preligrosidad que implica.

Los recursos a importar se situan todos en la carpeta **[theme-name]/src/WEB-INF/src/resources-importer**, debiendo mantener la siguiente estructura en su interior.

- sitemap.json - define las páginas, plantillas de diseño, el contenido web, los assets y portlets a importar.
- assets.json - (opcional) especifica detalles sobre los assets
- document_library/ - Carpeta que contiene lo relativo a los assets de tipo documento
    - ¬ documents/ - contiene documentos y archivos multimedia
- journal/ - Carpeta que contiene lo relativo a los assets de tipo **Contenido Web**
    - ¬ articles/- contiene los HTML y subcarpetas que agrupan estos contenidos por plantillas. Cada nombre de carpeta debe coincidir con el nombre de archivo de la plantilla correspondiente. Por ejemplo, en la carpeta **Template 1** se incluyen artículos basados en un archivo de plantilla **Template 1.ftl**.
    - ¬ structures/- contiene ficheros que definen estructuras en formato JSON y carpetas que agrupan las subestructuras. Cada nombre de carpeta debe coincidir con el nombre de archivo de la estructura padre correspondiente. Por ejemplo, la carpeta **Structure 1** contendrá hijos del archivo de estructura **Structure 1.json**.

- templates/- agrupa plantillas (VM o FTL) en carpetas, a emplear con las estructuras de datos definidas en **structures**. Cada nombre de carpeta debe coincidir con el nombre de archivo de la estructura correspondiente. Por ejemplo la carpeta **Structure 1** contendrá plantillas para el archivo de estructura **Structure 1.json**.

La anterior estructura es la estructura mas compleja y granular, alternativamente se puede sustituir todo por un fichero **.lar** obtenido como exportacion de recursos desde un Portal existente, que es mas comodo pero menos modular.

Para obtener un fichero **.lar** se puede hacer desde el menú **

El problema que pueden plantear los ficheros **.lar** es que son para versiones especificas del Portal.

17.3.1. sitemap.json

A continuacion se muestra un fichero de ejemplo **sitemap.json**

```
{
    "layoutTemplateId": "2_columns_ii",
    "privatePages": [
        {
            "friendlyURL": "/private-page",
            "name": "Private Page",
            "title": "Private Page"
        }
    ],
    "publicPages": [
        {
            "columns": [
                [
                    {
                        "portletId":
"com_liferay_login_web_portlet_LoginPortlet"
                    },
                    {
                        "portletId":
"com_liferay_site_navigation_menu_web_portlet_SiteNavigationMenuPortlet
"
                    },
                    {
                        "portletId":
```

```
"com_liferay_journal_content_web_portlet_JournalContentPortlet",
                        "portletPreferences": {
                            "articleId": "Without Border.html",
                            "groupId": "${groupId}",
                            "portletSetupPortletDecoratorId":
"borderless"
                        }
                    },
                    {
                        "portletId":
"com_liferay_journal_content_web_portlet_JournalContentPortlet",
                        "portletPreferences": {
                            "articleId": "Custom Title.html",
                            "groupId": "${groupId}",
                            "portletSetupPortletDecoratorId": "
decorate",
                            "portletSetupTitle_en_US": "Web Content
Display with Custom Title",
                            "portletSetupUseCustomTitle": "true"
                        }
                    }
                ],
                [
                    {
                        "portletId":
"com_liferay_hello_world_web_portlet_HelloWorldPortlet"
                    },
                    {
                        "portletId":
"com_liferay_site_navigation_menu_web_portlet_SiteNavigationMenuPortlet
_INSTANCE_${groupId}",
                        "portletPreferences": {
                            "displayStyle": "[custom]",
                            "headerType": "root-layout",
                            "includedLayouts": "all",
                            "nestedChildren": "1",
                            "rootLayoutLevel": "3",
                            "rootLayoutType": "relative"
                        }
                    },
                    "Web Content with Image.html",
                    {
                        "portletId":
"com_liferay_nested_portlets_web_portlet_NestedPortletsPortlet",
                        "portletPreferences": {
```

```
                            "columns": [
                                [
                                    {
                                        "portletId":
"com_liferay_journal_content_web_portlet_JournalContentPortlet",
                                        "portletPreferences": {
                                            "articleId": "Child Web
Content 1.xml",
                                            "groupId": "${groupId}",

"portletSetupPortletDecoratorId": "decorate",
                                            "portletSetupTitle_en_US":
"Web Content Display with Child Structure 1",
                                            "
portletSetupUseCustomTitle": "true"
                                        }
                                    }
                                ],
                                [
                                    {
                                        "portletId":
"com_liferay_journal_content_web_portlet_JournalContentPortlet",
                                        "portletPreferences": {
                                            "articleId": "Child Web
Content 2.xml",
                                            "groupId": "${groupId}",

"portletSetupPortletDecoratorId": "decorate",
                                            "portletSetupTitle_en_US":
"Web Content Display with Child Structure 2",
                                            "
portletSetupUseCustomTitle": "true"
                                        }
                                    }
                                ]
                            ],
                            "layoutTemplateId": "2_columns_i"
                        }
                    }
                ]
            ],
            "friendlyURL": "/home",
            "nameMap": {
                "en_US": "Welcome",
                "fr_FR": "Bienvenue"
```

```
            },
            "title": "Welcome"
        },
        {
            "columns": [
                [
                    {
                        "portletId":
"com_liferay_login_web_portlet_LoginPortlet"
                    }
                ],
                [
                    {
                        "portletId":
"com_liferay_hello_world_web_portlet_HelloWorldPortlet"
                    }
                ]
            ],
            "friendlyURL": "/layout-prototypes-parent-page", "layouts":
[
                {
                    "friendlyURL": "/layout-prototypes-page-1",
                    "layoutPrototypeLinkEnabled": "true",
                    "layoutPrototypeUuid": "371647ba-3649-4039-bfe6-
ae32cf404737",
                    "name": "Layout Prototypes Page 1",
                    "title": "Layout Prototypes Page 1"
                },
                {
                    "friendlyURL": "/layout-prototypes-page-2",
                    "layoutPrototypeUuid": "c98067d0-fc10-9556-7364-
238d39693bc4",
                    "name": "Layout Prototypes Page 2",
                    "title": "Layout Prototypes Page 2"
                }
            ],
            "name": "Layout Prototypes",
            "title": "Layout Prototypes"
        },
        {
            "columns": [
                [
                    {
                        "portletId":
"com_liferay_login_web_portlet_LoginPortlet"
```

```
                    }
                ],
                [
                    {
                        "portletId":
"com_liferay_hello_world_web_portlet_HelloWorldPortlet"
                    }
                ]
            ],
            "friendlyURL": "/parent-page",
            "layouts": [
                {
                    "friendlyURL": "/child-page-1",
                    "name": "Child Page 1",
                    "title": "Child Page 1"
                },
                {
                    "friendlyURL": "/child-page-2",
                    "name": "Child Page 2",
                    "title": "Child Page 2"
                }
            ],
            "name": "Parent Page",
            "title": "Parent Page"
        },
        {
            "friendlyURL": "/url-page",
            "name": "URL Page",
            "title": "URL Page",
            "type": "url"
        },
        {
            "friendlyURL": "/link-page",
            "name": "Link to another Page",
            "title": "Link to another Page",
            "type": "link_to_layout"
            "typeSettings": "linkToLayoutId=1"
        },
        {
            "friendlyURL": "/hidden-page",
            "name": "Hidden Page",
            "title": "Hidden Page",
            "hidden": "true"
        }
    ]
```

```
}
```

En este fichero se puede apreciar lo siguiente

- Lo primero y mas importante para entender la configuracion, es que a las **paginas** se las llama **layout**
- En la primera linea se establece la plantilla para las paginas, en este caso **2_columns_ii**, se pueden encontrar todas las plantillas del portal en la carpeta **<carpeta_del_bundle_de_liferay>\tomcat-8.0.32\webapps\ROOT\layouttpl**.
- Posteriormente se definen las paginas privadas y las publicas, si se aplica el tema a una plantilla de sitio, solo se aplicaran unas u otras, no las dos, pero si el tema se aplica a un sitio, se aplicarán ambas configuraciones.
- Al definir las paginas, se puede indicar:
  - ¬ Nombre
  - ¬ Título
  - ¬ URL amigable
  - ¬ Si esta oculta.
- Se puede definir que Portlets van a aparecer en cada pagina indicando su **ID de portlet**, que se pueden encontrar en el **Administrador de aplicaciones** del **Panel de control**. Se dispone de un listado de IDs aquí

Los campos que pueden aparecer son los siguientes

- colorSchemeId: Especifica el ID del esquema de color diferente al predeterminado a emplear.
- columns: Especifica un array de elementos que se incluiran en las distintas columnas del layout especificado.
- friendlyURL: establece la URL amigable de la pagina.
- hidden: establece si la pagina está oculta.
- layoutCss: Establece un CSS personalizado para la pagina, que secargará después del tema.
- layoutPrototypeLinkEnabled: establece si la pagina hereda los cambios realizados en la plantilla de página (si es que la pagina esta basada en una plantilla de pagina).

- layoutPrototypeName: especifica la plantilla de página (por nombre) que se usará para la pagina. Si esta propiedad es definida, no es necesario indicar el UUID de la plantilla de página.
- layoutPrototypeUuid: especifica la plantilla de página (por UUID) para usar en la pagina. Si layoutPrototypeName está definido no es obligatorio.
- layoutTemplateId: establece la plantilla de pagina.
- layouts: especifica páginas secundarias.
- name: Nombre de la pagina.
- nameMap: objeto en forma de pares de **clave/valor**, que define los distintos nombres de la pagina para los distintos codigos idiomaticos.
- portletPreferences: especifica las preferencias del portlet. Estas cambiaran para cada portlet.
- portletSetupPortletDecoratorId: especifica el decorador de portlet (borde) que se usará para el portlet, las opciones para el Tema Classic son: borderless, barebone o decorate.
- portlets: especifica los portlets que se mostrarán en cada columna. Se pueden anidar portlets empleando de forma recursiva **columns**.
- privatePages: especifica paginas privados.
- publicPages: especifica los paginas públicas.
- themeId: especifica un tema diferente (por ID) a aplicar en la pagina.
- title: título de la pagina.
- type: establece el tipo de pagina, el valor predeterminado es portlet (página vacía). Los valores posibles son: copy (copia de una página de este sitio), embedded, full_page_application, link_to_layout, node (conjunto de páginas), panel, portlet, y url(enlace a la URL).
- typeSettings: especifica la configuración (utilizando pares clave / valor) para el type aplicado a la pagina, si fuera necesario, por ejemplo el type **link_to_layout**, necesita que se le especifique a que pagina se linka **linkToLayoutId=1**

17.3.2. assets.json

En este fichero se especifican detalles sobre los assets importados, entre ellos

- Las etiquetas se pueden aplicar a cualquier assets.

- Resúmenes y thumbnail de artículos de contenido web.

El siguiente fichero **assets.json** especifica dos etiquetas para la imagen **company_logo.png**, una etiqueta para el contenido web **Custom Title.xml** y un resumen y thumbnail para la estructura **Child Web Content 1.json**.

```
{
    "assets": [
        {
            "name": "company_logo.png",
            "tags": [
                "logo",
                "company"
            ]
        },
        {
            "name": "Custom Title.xml",
            "tags": [
                "web content"
            ]
        },
        {
            "abstractSummary": "This is an abstract summary.",
            "name": "Child Web Content 1.xml",
            "smallImage": "company_logo.png"
        }
    ]
}
```

17.3.3. Contenido Web

A continuacion, se muestra la estructura de un fichero de contenido web, para ser empleado, el fichero XML deberá estar en **resources-importer/journal/articles/**

```
<?xml version="1.0"?>

<root available-locales="en_US" default-locale="en_US">
    <dynamic-element name="content" type="text_area" index-type=
"keyword" index="0">
        <dynamic-content language-id="en_US">
           <![CDATA[
             <center>
             <p><img alt="" src="[$FILE=space-program-history.jpg$]"
/></p>
             </center>

             <p>In the mid-20th century, after two of the
             most violent wars in history, mankind turned
             its gaze upwards to the stars. Instead of
             continuing to strive against one another,
             man choose instead to strive against the
             limits that we had bound ourselves to. And
             so the Great Space Race began.</p>

             <p>At first the race was to reach space--get
             outside the earth's atmosphere, and when
             that had been reached, we shot for the moon.
             After sending men to the moon, robots to
             Mars, and probes beyond the reaches of our
             solar system, it seemed that there was
             nowhere left to go.</p>

             <p>The Space Program aims to change that.
             Beyond national boundaries, beyond what
             anyone can imagine that we can do. The sky
             is not the limit.</p>
           ]]>
        </dynamic-content>
    </dynamic-element>
</root>
```

17.3.4. Estructuras

A continuacion, se muestra la estructura de un fichero de estructura de contenido web, para ser empleado, el fichero JSON deberá estar en **resources-importer/journal/structures/**

```
{
    "availableLanguageIds": [
        "en_US"
    ],
    "defaultLanguageId": "en_US",
    "fields": [
        {
            "label": {
                "en_US": "Content"
            },
            "predefinedValue": {
                "en_US": ""
            },
            "style": {
                "en_US": ""
            },
            "tip": {
                "en_US": ""
            },
            "dataType": "html",
            "fieldNamespace": "ddm",
            "indexType": "text",
            "localizable": true,
            "name": "content",
            "readOnly": false,
            "repeatable": false,
            "required": false,
            "showLabel": true,
            "type": "ddm-text-html"
        }
    ]
}
```

Se puede obtener el JSON de una estructura definida en el Portal, si al editar, se accede a la opcion **origen/source**.

17.3.5. Plantillas

A continuacion, se muestra la estructura de un fichero de plantilla de contenido web, para ser empleado, el fichero FTL, VM o XSLT deberá estar en **resources-importer/journal/templates/**

```
${content.getData()}
```

Se puede obtener el FTL, VM o XLST de una plantilla definida en el Portal, si al editar la plantilla se copia el script.

17.4. Modo de Desarrollador

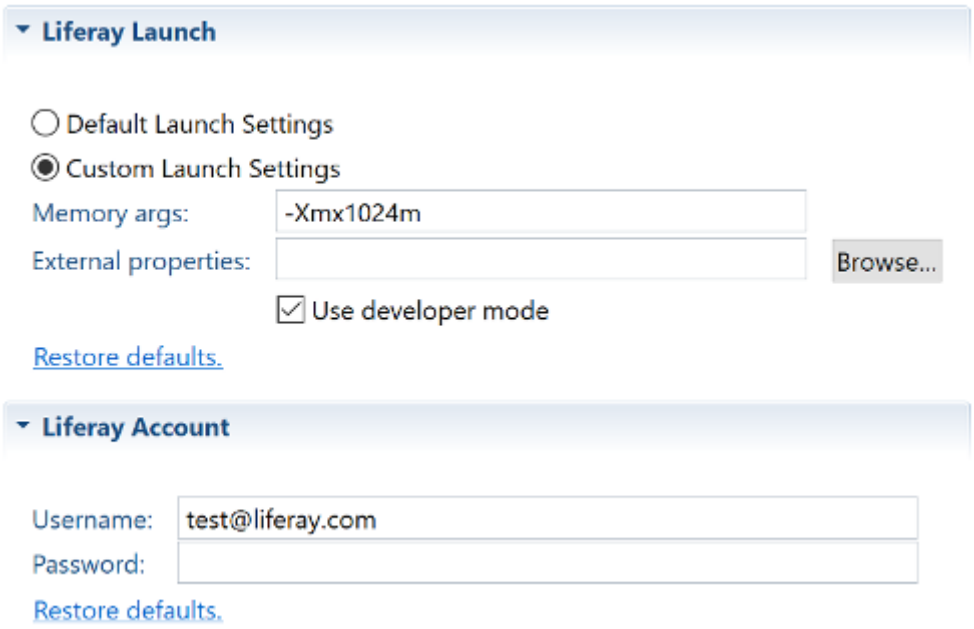
El modo desarrollador, permite visualizar los cambios en los ficheros del Tema sin redesplegar el tema, ya que elimina todas las caches que se generan en el portal.

Para activar el modo de desarrollador se ha de incluir el fichero **<directorio de tomcat>\webapps\ROOT\WEB-INF\classes\portal-developer.properties** como parte de la configuracion del portal que tiene el siguiente contenido

```
#Indica que los CSS se cargan individualemnte en lugar de agrupados
theme.css.fast.load=false
theme.css.fast.load.check.request.parameter=true
theme.images.fast.load=false
theme.images.fast.load.check.request.parameter=true

#Indica que el JS se carga rapidamente
javascript.fast.load=true
javascript.log.enabled=false

#Indica si las plantillas se almacenan en cache
layout.template.cache.enabled=false

browser.launcher.url=

combo.check.timestamp=true

#No se minimizan los CSS y JS
minifier.enabled=false

openoffice.cache.enabled=false

com.liferay.portal.servlet.filters.cache.CacheFilter=false
com.liferay.portal.servlet.filters.etag.ETagFilter=false
com.liferay.portal.servlet.filters.header.HeaderFilter=false
com.liferay.portal.servlet.filters.themepreview.ThemePreviewFilter=true
```

Hay tres formas de incluir este fichero en la configuracion del portal

- Desde el IDE en la configuracion del servidor



- Editando el fichero **portal-ext.properties** del portal.
- Añadiendo en el fichero **gradle.porperties** la siguiente linea

```
liferay.workspace.environment=dev
```

17.5. Macros

Las Macros permiten definir fragmentos de plantilla asociados a una variable, con el objetivo de reutilizar dicho fragmento.

Definicion del Macro

```
<#macro control_menu>
    <#if themeDisplay.isImpersonated() || (is_setup_complete &&
is_signed_in)>
            <@liferay_product_navigation["control-menu"] />
    </#if>
</#macro>
```

Uso del Macro

```
<@liferay.control_menu />
```

Se pueden definir macros que aceptan parametros

Definicion del Macro con parametros

```
<#macro language key>
    ${languageUtil.get(locale, key)}
</#macro>
```

Uso del Macro

```
<@liferay.language key="powered-by" />


En el portal se definen una serie de Macros, que están disponibles para
la definicion de las plantillas

|===

| Macro | Parametros | Descripcion

| breadcrumbs | default_preferences | Añade la miga de pan con
preferences opcionales

| control_menu | N/A | Añade el portlet menu de Control

| css | filename | Añade una hoja de estilos con el nombre indicado

| date | format | Imprime una fecha en el formato indicado

| js | filename | Añade un javascript con el nombre indicado

| language | key | Imprime el mensaje con la key indicada en el locale
actual

| language_format | arguments key | Imprime el mensaje parametrizado
con los arguments con la key indicada en el locale actual

| languages | default_preferences | Añade el portlet de Idiomas

| navigation_menu | default_preferences instance_ID | Añade el portlet
menu de Navegacion con preferencias opcionales y el ID de la instancia.
${freeMarkerPortletPreferences} es el valor mas comun para las
preferencias.

| search | default_preferences | Añade el portlet de busqueda con
```

```
preferences opcionales

| user_personal_bar | N/A | Añade el portlet de User Personal Bar

|===

//---------------------------------------------
//---------------------------------------------
=== Imagen de Miniatura
//---------------------------------------------
//---------------------------------------------

//https://dev.liferay.com/es/develop/tutorials/-/knowledge_base/7-
0/creating-a-theme-thumbnail

//---------------------------------------------
//---------------------------------------------

Es la imagen que se emplea desde el portal para identificar el Tema,
debe representar como quedan las paginas del portal cuando se aplica el
tema.

Debe tener unas dimensiones de 150 pixels de ancho y 120 pixels de
alto.

Y debe llamarse *thumbnail.png* y situarse en el directorio
*src/images*

//---------------------------------------------
//---------------------------------------------
=== Esquemas de color
//---------------------------------------------
//---------------------------------------------

//https://dev.liferay.com/es/develop/tutorials/-/knowledge_base/7-
0/specifying-color-schemes
//http://sergioglez.webcindario.com/cargarArticulo.php?id=56
//https://dev.liferay.com/es/develop/tutorials/-/knowledge_base/6-
2/creating-a-theme-thumbnail

//---------------------------------------------
//---------------------------------------------

Un esquema de color es una variante de un Tema, en el que se mantienen
el Tema original únicamente variando la gama de colores empleados.
```

```
Para definirlo basta con crear una nueva entrada en el fichero */WEB-
INF/liferay-look-and-feel.xml*.

[source, xml]
```

<theme id="Nuevo-theme" name="Nuevo">

```
<color-scheme id="01" name="default">
    <default-cs>true</default-cs>
    <css-class>default</css-class>
    <color-scheme-images-path>${images-
path}/color_schemes/default</color-scheme-images-path>
</color-scheme>
```

```
<color-scheme id="02" name="blue">
    <css-class>blue</css-class>
    <color-scheme-images-path>${images-path}/color_schemes/blue</color-
scheme-images-path>
</color-scheme>
```

</theme>

```
Los parametros a configurar son

* *default-cs*: Para indicar si es el por defecto (true/false).

* *css-class*: Nombre de la clase CSS que se añadirá al nodo principal
para cambiar estilos y adecuarlos al esquema de color.

[source, css]
```

blue #wrapper {

```
  background:blue;
}
```

```
* *color-scheme-images-path*: Para esta propiedad, es interesante el
empleo de la variable ${images-path}. Suele emplearse valores como
*${images-path}/color_schemes/default*

Por ultimo se ha de definir un nuevo fichero CSS, donde se definan los
nuevos estilos y cargarlo desde *main.css*

[source, css]
```

@import url(color_schemes/default.css);

```
Dentro de las imagenes a personalizar, las mas habituales, son
*screenshot.png*, *favicon.ico* y *thumbnail.png*

* Para *screenshot.pmg*, se recomienda un ancho de 1024px.
* Para *thumbnail.png*, se recomienda un ancho de 150px y un alto de
120px.
* Para *favicon.ico*, se recomienda un ancho de 16px y un alto de 16px.

//-----------------------------------------------
//-----------------------------------------------
=== Parametrizacion de las Plantillas
//-----------------------------------------------
//-----------------------------------------------

//https://dev.liferay.com/es/develop/tutorials/-/knowledge_base/7-
0/making-themes-configurable-with-settings

//-----------------------------------------------
//-----------------------------------------------

Es posible parametrizar los temas, añadiendo los parámetros al fichero
*/WEB-INF/liferay-look-and-feel.xml* con la etiqueta

[source, xml]
```

<settings> <setting key="mi-setting" value="mi-valor-1" /> <settings>

```
La etiqueta *setting* tiene los siguientes parámetros

* *Key*: Clave para acceder a la propiedad

* *Configurable*: Booleano que permite editar la propiedad.

* *Type*: Tipo de control para representar el parametro

** Select
** Checkbox

* *Options*: Valores posibles para el tipo select.

* *Value*: Valor por defecto para el parametro.

Siendo estos parámetros accesibles desde las plantillas

[source, freemarker]
```

${theme_settings["mi-setting"]}

```
Es habitual encontrar referencias a estos *settings* en el fichero
*init_custom.ftl* asociando su valor a variables que posteriormente se
reutilizan en el resto de plantillas.

[source, freemarker]
```

<#assign my_variable_name = getterUtil.getBoolean(theme_settings["theme-setting-key"])>

```
//-----------------------------------------------
//-----------------------------------------------
=== Layout Templates
//-----------------------------------------------
//-----------------------------------------------

//https://dev.liferay.com/es/develop/tutorials/-/knowledge_base/7-
0/layout-templates-intro

//-----------------------------------------------
//-----------------------------------------------

Permiten definir la estructura de las paginas del portal.

Esta basado en Bootstrap y su division del espacio basado en 12
columnas.

Se puede generar un *Layout Template* con la herramienta *Yeoman*

[source, console]
```

yo liferay-theme:layout

```
El comando se lanza sobre el directorio raiz de un tema generado por la
herramienta de *Yeoman*.

El comando ira preguntando por la disposicion del layout fila a fila,
lo primero que pregunta es el numero de columnas de la primera fila, y
el tanto por ciento que ocupa la primera columna de esa primera fila,
una vez definidas todas las columnas, pregunta si se desea añadir
alguna fila mas volviendose a repetir el proceso.

Finalmente se generan los ficheros de la plantilla y una imagen que
representa la plantilla, que se coloca en la carpeta
*src\layouttpl\custom\<identificador del layout>* y un fichero
*liferay-layout-templates.xml* en la carpeta *src/WEB-INF*, este
fichero declara la plantilla, pero dicha configuración deberá estar en
el fichero *liferay-look-and-feel.xml*.

[source, xml]
```

<theme id="porygon" name="Porygon"> . . . <layout-templates> <custom> <layout-template id="2-2-2-layout" name="2-2-2-layout"> <template-

path>\layouttpl\custom\2-2-2-layout-layouttpl/2_2_2_layout.tpl</template-path> <thumbnail-path>\layouttpl\custom\2-2-2-layout-layouttpl/2_2_2_layout.png</thumbnail-path> </layout-template> </custom> </layout-templates . . . </theme>

```
//include::temas\12-Theme_and_Portlets.adoc[]

//include::temas\13-Lexicon_CSS.adoc[]

//-----------------------------------------------
//-----------------------------------------------
=== Portlet Decorator
//-----------------------------------------------
//-----------------------------------------------

//https://dev.liferay.com/es/develop/tutorials/-/knowledge_base/7-0/portlet-decorators

//-----------------------------------------------
//-----------------------------------------------

Permiten definir estilos de pintado de los bordes de los Portlets.

Se han de declarar en el fichero *liferay-look-and-feel.xml*.

[source, xml]
```

<?xml version="1.0"?> <!DOCTYPE look-and-feel PUBLIC "-//Liferay//DTD Look and Feel 7.0.0//EN" "http://www.liferay.com/dtd/liferay-look-and-feel_7_0_0.dtd">

<look-and-feel> <compatibility> <version>7.0.0+</version> </compatibility> . . . <theme id="classic" name="Classic"> . . . <portlet-decorator id="barebone" name="Barebone"> <portlet-decorator-css-class>portlet-barebone</portlet-decorator-css-class> </portlet-decorator> <portlet-decorator id="borderless" name="Borderless"> <portlet-decorator-css-class>portlet-borderless</portlet-decorator-css-class> </portlet-decorator> <portlet-decorator id="decorate" name="Decorate"> <default-portlet-decorator>true</default-portlet-decorator> <portlet-decorator-css-class>portlet-decorate</portlet-decorator-css-class> </portlet-decorator> </theme> </look-and-feel>

```
Donde se definen los siguientes valores:

* *id*: Identificador unico
* *name*: nombre a mostrar en el seleccionable de *Decorator*
* *portlet-decorator-css-class*: Clase de CSS que se aplicará al
contenedor de los Portlet
* *default-portlet-decorator*: El Decorator por defecto.

Una vez declarados los Decorator, se han de definir los estilos
asociados a los *css-class*, para ello se define normalmente el fichero
*_portlet_decorator.scss*

[source, scss]
```

portlet-decorate .portlet-content {

```
    background: #FFF;
    border: 1px solid #DEEEEE;
}
```

portlet-barebone .portlet-content {

```
    padding: 0;
}
```

```
Que se habrá de incluir en los estilos del Tema, para ello se añade al
fichero *_custom.scss*

[source, scss]
```

@import "portlet_decorator"

```
Tambien se puede cambiar el comportamiento frente a los *Decorator* en
la plantilla que se encarga de pintar los Portlet, esta es
*portlet.ftl*.

En esta plantilla por defecto se recoje el campo *portletDecoratorId*
del objeto *portlet_display* y se muestra el titulo del Portlet si es
distinto de barebone

[source, ftl]
```

<#if portlet_display.getPortletDecoratorId() != "barebone"> <h2 class="portlet-title-text">${portlet_title}</h2> </#if>

```
```