Content

[1. Product Technologies 2](#_Toc24955734)

[Front-end 4](#_Toc24955735)

[Javascript library 4](#_Toc24955736)

[Lexicon: 4](#_Toc24955737)

[Template Language 4](#_Toc24955738)

[Velocity Template (.vm) 4](#_Toc24955739)

[FreeMarker Template (.ftl) 5](#_Toc24955740)

[2. Product Architecture: 6](#_Toc24955741)

[3. MVC Portlet Development 7](#_Toc24955742)

[3.1 Liferay MVC Basics 7](#_Toc24955743)

[3.2 Creating a basic module 8](#_Toc24955744)

[3.3 Creating a Module through IDE 8](#_Toc24955745)

[3.4 Deploying Modules 9](#_Toc24955746)

[3.5 Access the Portlet 10](#_Toc24955747)

[3.6 Module Structure 10](#_Toc24955748)

[3.7 Portlet Class 11](#_Toc24955749)

[4. Liferay Action URL : 13](#_Toc24955750)

[4.1 Taglib 13](#_Toc24955751)

[4.2 Portlet action URL 13](#_Toc24955752)

[4.3 Portlet URL with Parameter 14](#_Toc24955753)

[4.4 Form and Action URL snippet: 15](#_Toc24955754)

[5.Liferay Render URL 17](#_Toc24955755)

[5.1 Taglib 17](#_Toc24955756)

[2.2 5.2 Portlet render URL 17](#_Toc24955757)

[5.3 Redirection to another jsp from Action Method 18](#_Toc24955758)

[6. Liferay Resource URL example 19](#_Toc24955759)

[6.1 Taglib 19](#_Toc24955760)

[6.2 Portlet resource URL without any framework 19](#_Toc24955761)

[ServeResourceMethod: 20](#_Toc24955762)

[6.3 Portlet resource URL with any framework[Alloy] 21](#_Toc24955763)

[6.4 Portlet resource URL with any framework [jquery ajax] 23](#_Toc24955764)

[7. Liferay Search Container ( Table) 25](#_Toc24955765)

[7.1 Taglib 25](#_Toc24955766)

[7.2. liferay-ui-search-container usage 25](#_Toc24955767)

[8. Passing Attributes from action to view: 27](#_Toc24955768)

[9. MVCCommands 29](#_Toc24955769)

[10. Bootstrap forms 29](#_Toc24955770)

[11. CRUD operation with Service Builder 29](#_Toc24955771)

Liferay Development

# Product Technologies



With JavaEE and OSGi at the bottom of the stack , Here is the core technologies:

* Spring for transactions (and Dependency Injection in the core)
* Hibernate for database access (along with direct JDBC access for optimized queries)
* Elasticsearch for indexing and searching
* Ehcache for caching.

## Front-end

* Alloy-UI (yahoo UI implementation)
* Bootstrap
* SaSS ( stylesheet language)

## Javascript library

* Metal.js (developed by Liferay)
* AlloyUI (developed by Liferay)
* jQuery (included)
* Lodash (included)
* Angular 1 or 2
* React
* Your library of choice

## Lexicon:

Liferay Portal follows a design language created by our designers at Liferay called [Lexicon Experience Language](https://lexicondesign.io/), which has been implemented for use of the web as [Lexicon](https://liferay.github.io/clay/).

* Lexicon is automatically made available to application developers through a set of CSS classes and markup, although it’s even easier to use our tag library.

## Template Language

For templating, JavaEE’s JSP is there as expected as well as Velocity Template and FreeMarker.

### Velocity Template (.vm)

A velocity template file is a file which has ***vm*** as file extension (for example ***porta\_normal.vm***). It is used in theme creation in liferay ( init.vm, navigation.vm…etc)

If condition:

#set ($temp = 99)

#if( $temp > 100 )

    <b>The variable is greater than </b>

#elseif( $temp < 100 )

    <b>The variable is less than 100</b>

#else

    <b>Not matching any conditions</b>

#end

Forloop:

#set($collection = [100, "hello", 10, "proliferay"])

#foreach( $obj in $collection )

    <br/>

    $obj

#end

### FreeMarker Template (.ftl)

**FreeMarker** is a **free** Java-based **template** engine, originally focusing on dynamic web page generation with MVC software architecture

Assignment:

<#assign name ="meera"/>

<#assign categoryId =100/>

Array:

<#assign cat = [100,101,102] />

<#assign selectVocab = ["Department", "Client Scope"] />

Output:

${ name }

If condition:

<#if categoryId != 0>

<#--  Do something -->

</#if>

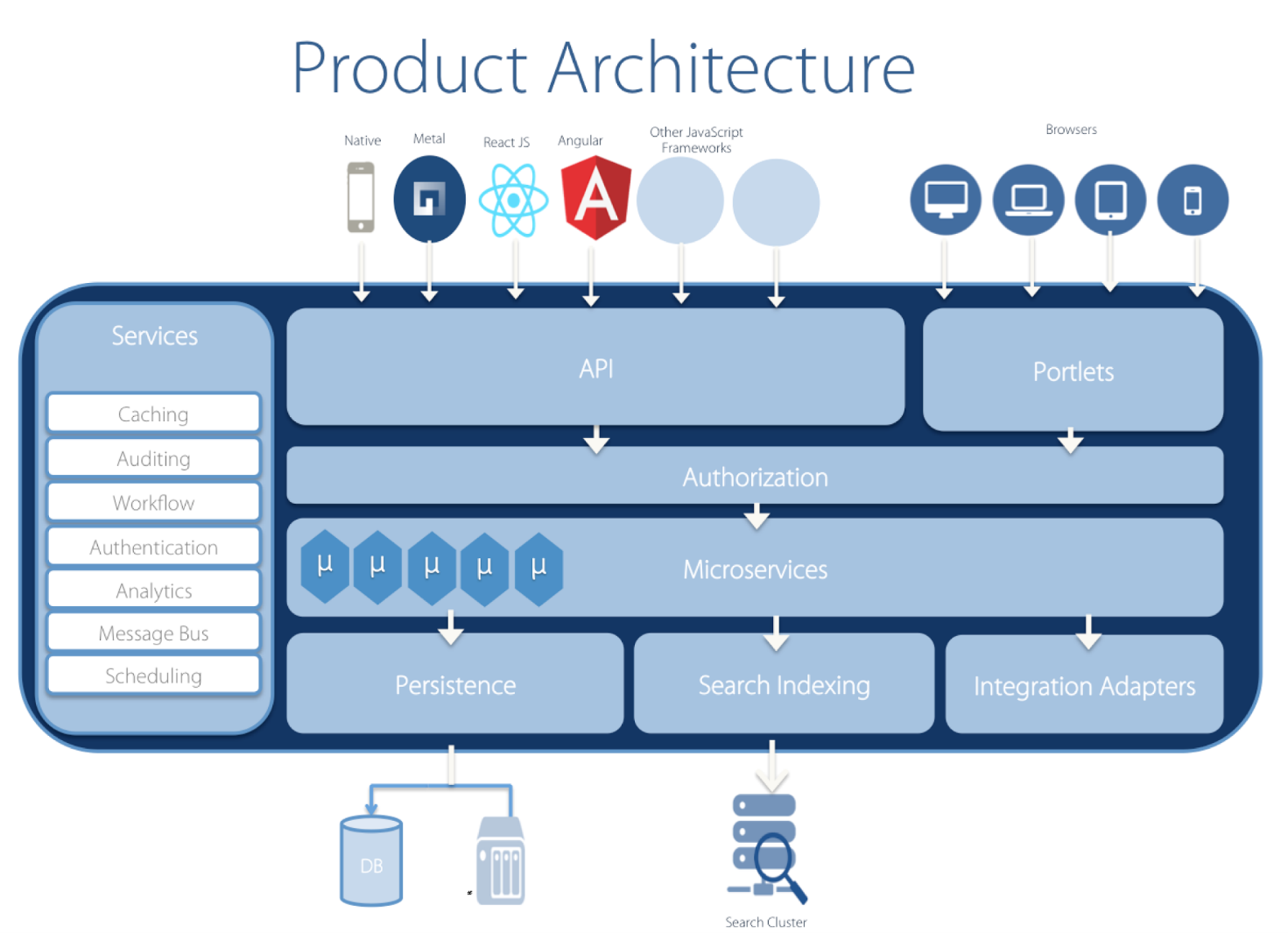
For loop:

<#assign categoryIds ={13,39,45,10}/>

<#list categoryIds as categoryId>

</#list>

# Product Architecture:



# MVC Portlet Development

## 3.1 Liferay MVC Basics

In MVC, there are three layers,

**Model:** The model layer holds the application data and logic for manipulating it.

**View:** The view layer contains logic for displaying data.

**Controller:** The middle man in the MVC pattern, the Controller contains logic for passing the data back and forth between the view and the model layers.

* It’s lightweight, as opposed to many other Java MVC frameworks.
* There are no special configuration files that need to be kept in sync with your code.
* It’s a simple extension of GenericPortlet.
* You avoid writing a bunch of boilerplate code, since Liferay’s MVC framework simply looks for some pre-defined parameters when the init() method is called.
* The controller can be broken down into MVC command classes, each of which handles the controller code for a particular portlet phase (render, action, and resource serving phases).

The Liferay MVC portlet framework is light, it hides part of the complexity of portlets, and it makes the most common operations easier.

Keep in mind that you can take two paths with your Liferay MVC portlet implementation.

If you have a small application that won’t be heavy on controller logic (maybe just a couple of action methods), you can put all your controller code in the -Portlet class.

## 3.2 Creating a basic module

Module projects are created in the modules folder by default.

Module through blade command:

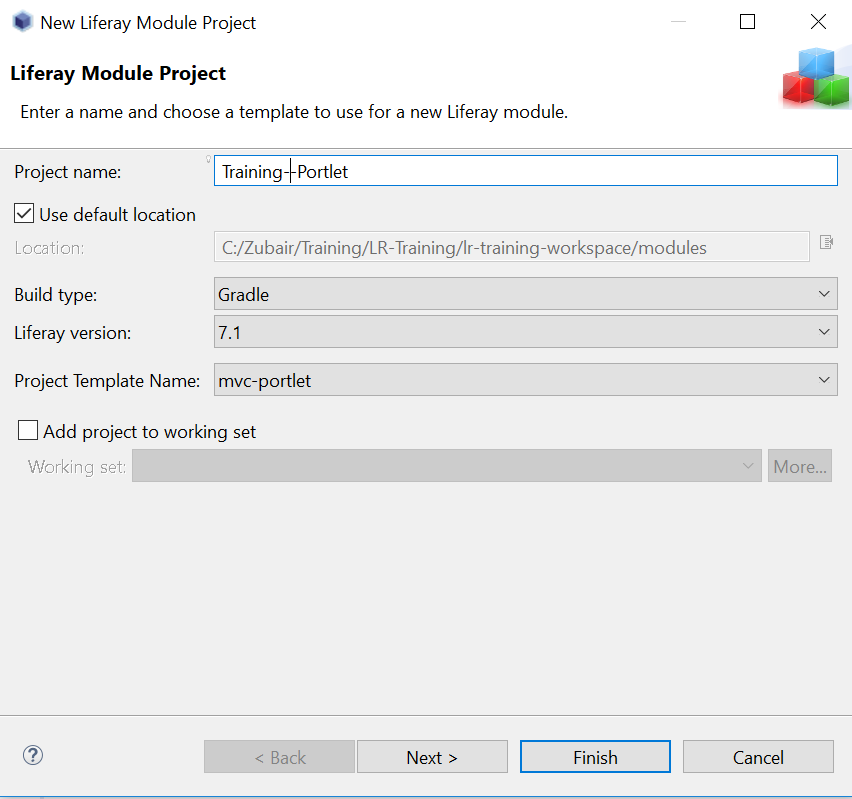
blade create -t mvc-portlet -p com.liferay.docs.mymodule -c Training-Portlet my-module

## 3.3 Creating a Module through IDE

-Click “New Liferay Module Project “ from liferay icon tool bar

-Give the “Module Name” and “Component Name”

-Finish

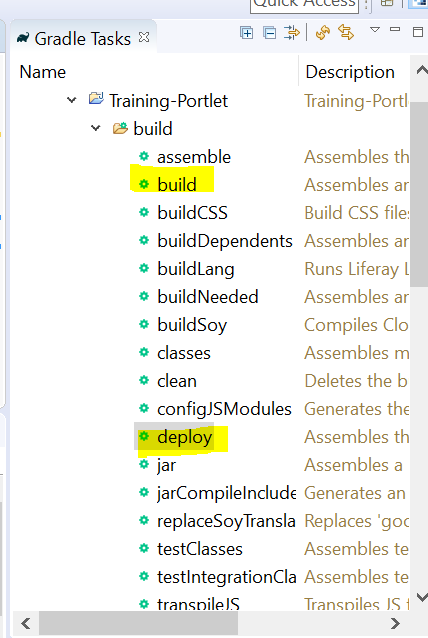


## 3.4 Deploying Modules

* 1. Right click the module and Refresh Gradle
  2. Go to Gradle Task -> open module 🡪 Build and Deploy

Or

Drag and drop the module to Liferay Server.

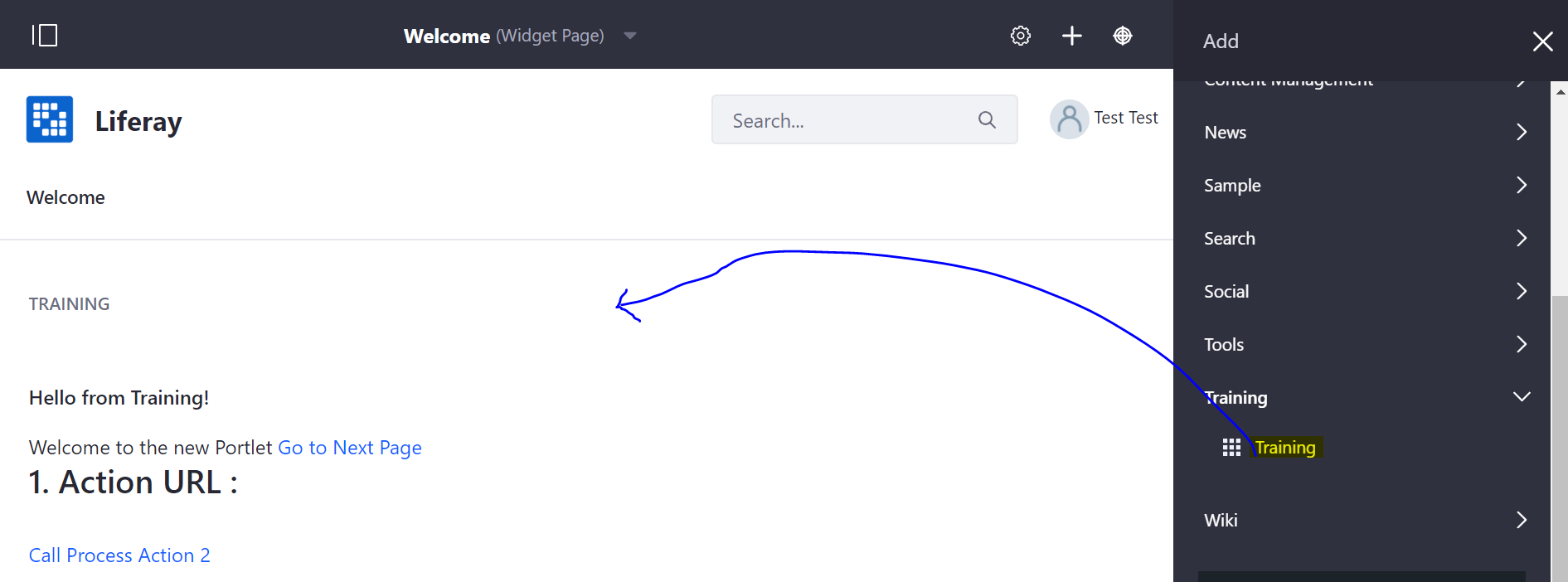


## 3.5 Access the Portlet

- Start the liferay https://localhost:8080

- Select Module / Portelt from Add **Widget** section ( right corner)

- Drag, and Drop

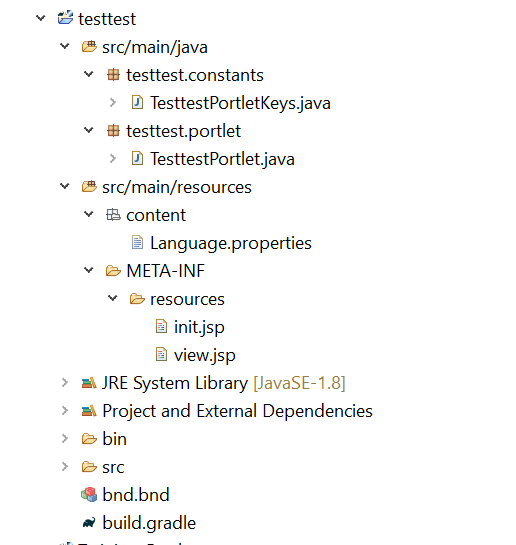


## 3.6 Module Structure

Here’s the module project anatomy:

* src/main/java/ → Java package root
* src/main/resources/content/ (optional) → Language resource bundle root
* src/main/resources/META-INF/resources/ (optional) → Root for UI templates, such as JSPs
* bnd.bnd → Specifies essential OSGi module manifest headers
* build.gradle → [Configures dependencies](https://portal.liferay.dev/docs/7-0/tutorials/-/knowledge_base/t/configuring-dependencies) and more using Gradle

The figure below shows an MVC portlet module project.



## 3.7 Portlet Class

@Component annotation, which tells the OSGi container how it should treat this module.

immediate=true, you’re saying that when this module is deployed and all of its dependencies are satisfied, it should be started immediately instead of being lazy-loaded.

Property for specifying the configurations for portlet behaviours and attributes. It wad declared in older versions , now it is moved to annotations.

Liferay’s MVCPortlet class (that extends GenericPortlet, that implements Portlet).

We have to override the portlet life cycle methods (**render** or **process** or **serve** **resource**) from portlet classes ( MVC , Liferay or Generic…etc )

@Component(

immediate = true,

property = {

"com.liferay.portlet.display-category=category.sample",

"com.liferay.portlet.instanceable=true",

"javax.portlet.display-name=Training-portlet",

"javax.portlet.init-param.template-path=/",

"javax.portlet.init-param.view-template=/view.jsp",

"javax.portlet.resource-bundle=content.Language",

"javax.portlet.security-role-ref=power-user,user"

},

service = Portlet.class

)

public class TrainingPortlet extends MVCPortlet {

@Override

public void render(RenderRequest renderRequest,RenderResponse renderResponse)

throws IOException, PortletException

{

ThemeDisplay themeDisplay = (ThemeDisplay)

renderRequest.getAttribute(WebKeys.THEME\\_DISPLAY);

User user = themeDisplay.getUser();

renderRequest.setAttribute("userName", user.getFirstName());

super.render(renderRequest, renderResponse);

}

}

# Liferay Action URL :

Liferay portlet provides three different types of URLs to support render and action phases. those are : Action URL, Render URL and resource URL.

## 4.1 Taglib

Initially, we have to import portlet tag library using following import statement in our jsp file:

<%@ taglib uri="http://java.sun.com/portlet\_2\_0" prefix="portlet" %>

## 4.2 Portlet action URL

Add the following code to jsp page. Here, we are using portlet tag library to create action URL. This URL can be used to call action inside the scope of this jsp page.

<portlet:actionURL var="submitURL1" name="callProcessActionMethod">

</portlet:actionURL>

Or

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

<**liferay-portlet**:actionURL var="submitURL1" name="callProcessActionMethod">

</lifray-portlet:actionURL>

**Action** URL tag also provides few attributes which are:

***name***: This attribute indicates the name of the **action** **method** which is invoked by the action URL.

***var****:*This attribute is a name of the **URL**, which can be used to call action URL.

## 4.3 Portlet URL with Parameter

We can also add parameters to action URL using tag inside action URL tag as shown above.

<portlet:actionURL var="submitURL1" name="callProcessActionMethod">

<portlet:param name="name" value="zubair"/>

</portlet:actionURL>

Created action URL has new parameter “**name**” which contains value “**Zubair**”

Now add the following code inside the action class, which handles the action URL request.

public class TrainingPortlet extends MVCPortlet {

@ProcessAction(name="processActionMethod")

public void callProcessActionMethod(ActionRequest actionRequest, ActionResponse actionResponse){

String paramValue = ParamUtil.getString(actionRequest, "name");

System.out.println("#######callProcessActionMethod##############");

}

}

## 4.4 Form and Action URL snippet:

View.jsp

<portlet:actionURL var="personalInfoURL" name="submitPersonalInfo">

</portlet:actionURL>

<form name="completeFrm" method="post">

First Name : <input type="text" name="<portlet:namespace />firstName"/>

Last Name : <input type="text" name="<portlet:namespace />lastName"/>

Iqama No : <input type="text" name="<portlet:namespace />iqama"/>

<input type="button" value="Submit Form" onclick="submitCompleteFrm()">

</form>

<script>

function submitCompleteFrm() {

document.completeFrm.action='<%= personalInfoURL %>';

document.completeFrm.submit();

}

</script>

TrainingPortlet.java

@ProcessAction(name = "submitPersonalInfo")

**public** **void** submitPersonalInfo(ActionRequest actionRequest, ActionResponse actionResponse) {

String firstName = ParamUtil.*getString*(actionRequest, "firstName");

String lastName = ParamUtil.*getString*(actionRequest, "lastName");

String iqama = ParamUtil.*getString*(actionRequest, "iqama");

System.***out***.println("#######firstName##############" + firstName);

System.***out***.println("#######lastName##############" + lastName);

System.***out***.println("#######iqama##############" + iqama);

}

# 5.Liferay Render URL

To invoke render phase of portlet, we can use **render** **URL**. Render phase is always executed after the execution of **action** **phase**.

## 5.1 Taglib

Initially, we have to import portlet tag library using following import statement in our jsp file:

<%@ taglib uri="http://java.sun.com/portlet\_2\_0" prefix="**portlet**" %>

## 2.2 5.2 Portlet render URL

Add the following code to jsp page. Here, we are using portlet tag library to create **render** **URL**.

Here, we are setting a parameter for render URL named as “renderParam” and also we are rendering jsp page ***success.jsp*** on clicking the anchor tag.

the **var** attribute is the reference of the render URL, which can be used inside the scope of this URL.

<portlet:renderURL var="renderPortletURL" >

<portlet:param name="mvcPath" value="/success.jsp"/>

<portlet:param name="name" value="zubair"/>

</portlet:renderURL>

<a href="<%=renderPortletURL %>" >Render URL</a>

Create a new jsp page called ***success.jsp,***which will be rendered on clicking the tag on **view**.**jsp** page.

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

<%@page import="com.liferay.portal.kernel.util.ParamUtil"%>

<%@ taglib uri="http://java.sun.com/portlet\_2\_0" prefix="portlet" %>

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

Welcome Mr. <%=name %>

**Tips**: You can call the jsp page from the render url as **jspPage** or **mvcPath** parameters.

<portlet:param name="**jspPage**" value="/success.jsp"/>

or

<portlet:param name="**mvcPath**" value="/success.jsp"/>

## 5.3 Redirection to another jsp from Action Method

actionResponse.getRenderParameters().setValue("mvcPath", "/success.jsp");

# Liferay Resource URL example

Resource URL is used to retrieve images, XML, JSON or any other type of resource. Resource URL invokes the serveResource() method of portlet. This is helpful for operations like dynamically getting the content from server(using AJAX), downloading a file, etc.

After serve resource call is handled, it will simply returns response back to client side and no other methods are executed. Hence we have to use AJAX  method while making a serve resource call to fetch data from server side.

## 6.1 Taglib

Initially, we have to import liferay portlet tag library using following import statement in our jsp file:

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

## 6.2 Portlet resource URL without any framework

We can call the ajax using normal java script or alloy ui or jquery…etc. Here we can call the ajax with javascript.

 In ***xmlhttp.open("GET",url,true)*** the first parameter is ***http*** method, second is the ***url*** that you need to invoke as ajax and third parameter is ***true*** means its an Ajax call. If its ***false*** then its not an ajax call.

<portlet:resourceURL var="resourceURL"/>

<script type="text/javascript">

function callServeResource(url) {

url = url + "&<portlet:namespace/>**fname**=Gnaniyar&<portlet:namespace/>**lname**=Zubair";

var xmlhttp=new XMLHttpRequest();

xmlhttp.open("GET",url,true);

xmlhttp.send();

}

</script>

<input type="button" value="Ajax Call" onclick=" callServeResource ('<%=resourceURL.toString()%>')">

We are sending two parameters in the ajax call. One is ***fname*** and another is ***lname***. Read those parameter in **serve resource** method as usual.

## ServeResourceMethod:

@Override

public void serveResource(ResourceRequest resourceRequest, ResourceResponse resourceResponse)

throws IOException, PortletException {

String firstName = ParamUtil.getString(resourceRequest, "fname");

String lastName = ParamUtil.getString(resourceRequest, "lname");

System.out.println("#######fName##############" + firstName);

System.out.println("#######lName##############" + lastName);

// TODO Auto-generated method stub

super.serveResource(resourceRequest, resourceResponse);

}

## 6.3 Portlet resource URL with Alloy UI

<%@ taglib uri="http://java.sun.com/portlet\_2\_0" prefix="portlet" %>

<%@ taglib uri="http://liferay.com/tld/portlet"

prefix="liferay-portlet" %>

<liferay-portlet:resourceURL var="resourceURL" >

<portlet:param name="name" value="zubair"/>

</liferay-portlet:resourceURL>

<input type="button" value="Ajax Call" onclick="callServeResource1('<%=resourceURL.toString()%>')">

<br/>

<h4>Serve resource response will be printed here:

<br/>

<span id="resHolder"></span></h4>

<script>

function callServeResource(url)

{

AUI().use('aui-io-request', function(A){

A.io.request(url, {

dataType: 'json',

on: {

success: function() {

var data = this.get('responseData');

document.getElementById('resHolder').innerHTML=data.myValue;

},

failure: function(){alert('Error');}

}

})

});

</script>

**ServeResourceMethod**:

@Override

public void serveResource(ResourceRequest resourceRequest, ResourceResponse resourceResponse)

throws PortletException, IOException {

System.out.println("###inside servr resource ( alloy )########" );

String param = ParamUtil.getString(resourceRequest, "name");

Writer wtr = resourceResponse.getWriter();

JSONObject jsonObj = JSONFactoryUtil.createJSONObject();

jsonObj.put("value", param);

wtr.write(jsonObj.toString());

wtr.flush();

}

## 6.4 Portlet resource URL with [jquery ajax]

Include jquery: ( if not available)

<script src=*"https://code.jquery.com/jquery-3.4.1.min.js"*></script>

Tips: include the jquery in theme or **init**.jsp.

**View.jsp**

<h1>Resource URL jQuery AJAX :</h1> <br/>

<liferay-portlet:resourceURL var=*"resourceURL"* >

</liferay-portlet:resourceURL>

<input type=*"button"* value=*"Ajax Call"* onclick=*"callServeResourceJQuery('*<%=resourceURL.toString()%>*')"*>

<h4>Serve resource response will be printed here:

<span id=*"resHolder1"*></span></h4>

<script>

function callServeResourceJQuery(url)

{

jQuery.ajax({ type : "POST",

url : url,

data : ({

<portlet:namespace/>fname: "Gnaniyar",

<portlet:namespace/>lname : "Zubair"

}),

dataType: 'json',

success:function(data)

{

document.getElementById('resHolder1').innerHTML=data.fname+""+data.lname;

},

async : false,

});

}

</script>

**ServeResource** method in portlet controller:

@Override **public** **void** serveResource(ResourceRequest resourceRequest,

ResourceResponse resourceResponse) **throws** PortletException, IOException {

System.***out***.println("####inside servr resource ( jquery )#########" );

String param1 =ParamUtil.*getString*(resourceRequest,"fname");

String param2 =ParamUtil.*getString*(resourceRequest,"lname");

// Creating a JSON object.

JSONObject jsonObj = JSONFactoryUtil.*createJSONObject*();

jsonObj.put("fname", param1);

jsonObj.put("lname", param2);

System.***out***.println("#######jsonObj.toString()##############" +jsonObj.toString() );

// Writing the result in resourceResponse writer.

Writer wtr = resourceResponse.getWriter();

wtr.write(jsonObj.toString());

wtr.flush();

}

# Liferay Search Container ( Table)

Liferay Search Container is Liferay UI component to display data in Grid format and its containing pagination. This is simple JSTL tag we can use in JSP pages.

## 7.1 Taglib

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

## 7.2. liferay-ui-search-container usage

To create the table format, liferay provide ui taglib.

**< Liferay-Ui:search-Container>**

This is the main tag used to declare the search container.  
**total** : total number of items available in the table.  
**delta**: items per page  
**deltaConfigurable**: Flag to allow user to change the item per page from UI  
**emptyResultsMessage**: Message To Display when there are no items

**< Liferay-Ui:search-Container-Results>**

Using this tag we can iterate through the list of items.  
**result**: A variable to hold result

**< Liferay-Ui:search-Container-Row>**

<%@page import="java.util.List"%>

<%@page import="com.liferay.portal.kernel.model.User"%>

<%@page import="com.liferay.portal.kernel.util.ListUtil"%>

<%@page import="com.liferay.portal.kernel.service.UserLocalServiceUtil"%>

<% List<User> users = UserLocalServiceUtil.getUsers(-1,-1);%>

<liferay-ui:search-**container** total="<%=users.size()%>" var="searchContainer" delta="1" deltaConfigurable="true"

emptyResultsMessage="Oops. There Are No Users To Display, Please add Users">

<liferay-ui:search-**container-results** results="<%=ListUtil.subList(users, searchContainer.getStart(),searchContainer.getEnd())%>" />

<liferay-ui:search-**container-row** className="com.liferay.portal.kernel.model.User" modelVar="user" keyProperty="userId" >

<liferay-ui:search-container-column-text name="User Id" value="${user.userId}"/>

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

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

<liferay-ui:search-container-column-text name="Email" value="${user.emailAddress}"/>

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

<liferay-ui:search-iterator />

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

# Passing Attributes from action to view:

Action:

public void callProcessActionMethod(ActionRequest actionRequest, ActionResponse actionResponse){

String firstName = ParamUtil.getString(actionRequest, "firstName");

String lastName = ParamUtil.getString(actionRequest, "lastName");

String iqama = ParamUtil.getString(actionRequest, "iqama");

System.out.println("#######firstName########" + firstName);

System.out.println("#######lastName########" + lastName);

System.out.println("#######iqama###########" + iqama);

// converting data to list

List<String> list = new ArrayList<String>();

list.add(firstName);

list.add(lastName);

list.add(iqama);

//setting attributes in the request

actionRequest.setAttribute("list",list);

//redirection actionResponse.getRenderParameters().setValue("mvcPath",

"/success.jsp");

}

View

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

<%@page import="java.util.List"%>

<% List<String> list = (List<String>)request.getAttribute("list"); %>

Data Added Successfully.

<a href=*"*<portlet:renderURL/>*"*><<< Go Back</a>

<table class=*"table table-striped"*>

<thead>

<tr>

<th scope=*"col"*>First Name</th>

<th scope=*"col"*>Last Name</th>

<th scope=*"col"*>Iqama</th>

</tr>

</thead>

<tr>

<c:forEach items="${list}" var=*"list"*>

<td><c:out value="${list}" /></td>

</c:forEach>

</tr>

</table>

# MVCCommands

In a larger application, Liferay provides MVC command classes to break up your controller functionality:

MVCActionCommand: Use -ActionCommand classes to hold each of your portlet actions, which are invoked by action URLs.

MVCRenderCommand: Use -RenderCommand classes to hold a render method that dispatches to the appropriate JSP, by responding to render URLs.

MVCResourceCommand: Use -ResourceCommand classes to execute resource serving in your MVC portlet, by responding to resource URLs.



# Liferay APIs

## SessionMessages

Ref:

<https://portal.liferay.dev/docs/7-0/tutorials/-/knowledge_base/t/adding-failure-and-success-messages>

SessionMessages.add(actionRequest, "entryAdded");

<liferay-ui:success key="entryAdded" message="entry-added" />

**Language\_ar.properties:** entry-added=fdsdfsdf

## Languages changes

<liferay-ui:message key="first-name" />

## Portlet Session

### set

PortletSession session = renderRequest.getPortletSession();

session.setAttribute("product", "sanad123",PortletSession.PORTLET\_SCOPE);

### get

PortletSession session = renderRequest.getPortletSession();

String sessionvalue = )String) session.setAttribute("product", "sanad123",PortletSession.PORTLET\_SCOPE);

### get session in jsp:

<%@page import="javax.portlet.PortletSession"%>

<% PortletSession renderSession = renderRequest.getPortletSession();

String sessionValue = (String) renderSession.getAttribute("product",PortletSession.PORTLET\_SCOPE); %>

Session Values: <%=sessionValue %>

# CRUD operation with Service Builder