Changing the Theme in Liferay

Posted on April 12, 2012

At this point you have all the portlets you want on your page and you've created and logged in with your own user. For security, delete the test user that you logged in with in the previous section. To delete the test user, click on the Users tab in Enterprise Admin portlet and search for the test user. Click on the Delete icon

You can now change the look and feel of your portal. Liferay Portal comes pre-bundled with different themes that you can apply to your page.
Changing the Theme
• Click on the Page Settings link at the top of your page.
• Select the page you would like to theme on the left hand side. Note that by default all children pages will inherit the theme from the parent.
• Now select the Look and Feel tab for that page.
• You will see a number of bundled themes that are available with Liferay Portal. Choose your theme and color scheme. You can experiment with it as much as you like until you find a theme that pleases you. Don't spend too much time here now… time to move on…
NOTE: Although Liferay comes with many bundled themes, there are also a vast number of themes contributed to Liferay Portal from the community. You can view these on our website or develop your own custom theme for your company or application. Please read the Building Themes chapter in the Liferay Portal Developers Guide.

Posted in Liferay Articles, Liferay Examples, Tutorial | Leave a comment

Simple JSP Portlet

Posted on April 12, 2012

Although a JSPPortlet does little more than display content, there is still some work that needs to be done. Let's start by creating a new directory called myjspportlet within {Liferay}\ext\ext-web\docroot\html\portlet\ext\. Next,
open portlet-ext.xml within {Liferay}\ext\ext-web\docroot\WEB-INF\.
Note: You may need to associate .xml files to Eclipse if your .xml files are being opened in a separate editor. You can do this by selecting Window from the menu bar and then Preferences. Expand the Workbench navigation, and click on File Associations. From there you can add *.xml as a new File type and associate it to open in Eclipse.
Notice how the portlets are uniquely identified by their portlet-name. As such, you will want to create a new portlet that is an increment of the portlet name, such as EXT_2. Since we are creating a JSPPortlet, you will want the portlet-class to reference the full class name: com.liferay.portlet.JSPPortlet. For this tutorial, add the following to your portlet-ext.xml (you may find it easier to copy and paste EXT_1 and just make the necessary changes):

```
<portlet>
<portlet-name>EXT_2</portlet-name>
<display-name>My JSPPortlet</display-name>
```

<portlet-class>com.liferay.portlet.JSPPortlet</portlet-class>
<init-param>
<name>view-jsp</name>
<value>/portlet/ext/myjspportlet/view.jsp</value>
</init-param>
<expiration-cache>300</expiration-cache>
<supports>
<mime-type>text/html</mime-type>
</supports>
<portlet-info>
<title>My JSP Portlet</title>
</portlet-info>
<security-role-ref>
<role-name>Power User</role-name>
</security-role-ref>
<security-role-ref>
<role-name>User</role-name>
</security-role-ref>
</portlet>

Here is a basic summary of what each of the elements represents:

portlet-name The portlet-name element contains the canonical name of the portlet. Each portlet name is unique within the portlet application.

display-name The display-name type contains a short name that is intended to be displayed by tools. It is used by displayname elements. The display name need not be unique.

portlet-class The portlet-class element contains the fully qualified class name of the portlet.

init-param The init-param element contains a name/value pair as an initialization param of the portlet.

expiration-cache Expiration-cache defines expiration-based caching for this portlet. The parameter indicates the time in seconds after which the portlet output expires. -1 indicates that the output never expires.

supports The supports element contains the supported mime-type.
Supports also indicates the portlet modes a portlet supports for a specific content type. All portlets must support the view mode.

portlet-info Portlet-info defines portlet information.

security-role-ref The security-role-ref element contains the declaration of a security role reference in the code of the web application. Specifically in Liferay, the role-name references which role's can access the portlet. (A Power User can personalize the portal, whereas a User cannot.)

Now that you have configured your portlet-ext.xml, the next step is to create the jsp pages. Within your /myjspportlet directory, add a file called init.jsp. Within this file, add the following two lines of code:

```
<%@ include file="/html/common/init.jsp" %>
<portlet:defineObjects />
```

These two lines import all the common class files and also set common variables used by each portlet. If you

need to
import portlet specific classes or initialize portlet specific variables, be sure to add them to their directory specific
init.jsp, as opposed to the common/init.jsp.
These two lines import all the common class files and also set common variables used by each portlet. If you need to
import portlet specific classes or initialize portlet specific variables, be sure to add them to their directory specific
init.jsp, as opposed to the common/init.jsp.
Now, add a view.jsp. This jsp file will hold the content of your JSPPortlet. Write “Hello [your name here]# within
the jsp. So the question is then, how does the portal know how to load these particular files? If you look back at the
portlet element that was added within portlet-ext.xml, you will notice that you initialized a view-jsp parameter as
having the value /ext/myjspportlet/view.jsp. By specifying this init-param, you are giving the portlet a default jsp to load.
Finally, in order to be able to add your portlet to the portal, you need to define the name within Languageext.properties by adding the following line:
javax.portlet.title.EXT_2=My JSP Portlet
Since you have setup the Extension Environment, you need to deploy the changes you have made to your applicable
server by running deploy within the build.xml of {Liferay}\ext\. In Eclipse, double click the deploy [default] target
within your Ant view. Start Tomcat again as soon as the deployment finishes. Browse to the Home tab of the portal,
and in the Add Portlet to Wide Column dropdown add “My JSP Portlet# to your portal.

Posted in Liferay Examples, Tutorial | Leave a comment

Creating Liferay Themes

Posted on April 12, 2012

Add themes to your project

1) Open command prompt or terminal and go to your project directory. Next we are going to create a theme using the Liferay theme template. Run:

```
mvn archetype:generate \
-DarchetypeArtifactId=liferay-theme-archetype \
-DarchetypeGroupId=com.liferay.maven.archetypes \
-DarchetypeVersion=6.1.0 \
-DartifactId=sample-theme \
-DgroupId=com.liferay.sample \
-Dversion=1.0-SNAPSHOT
```

For 6.1 EE ga1 use -DarchetypeVersion=6.1.10.
Now you have your theme project in sample-theme directory with following structure.

sample-theme

sample-theme/pom.xml
sample-theme/src
sample-theme/src/main
sample-theme/src/main/resources
sample-theme/src/main/webapp
sample-theme/src/main/webapp/WEB-INF
sample-theme/src/main/webapp/WEB-INF/liferay-plugin-package.properties
sample-theme/src/main/webapp/WEB-INF/web.xml
2) Open the theme pom.xml file. From the properties section
remove *liferay.version* and *liferay.auto.deploy.dir* properties. These properties should be defined in the pom.xml in your project root just as we did with the portlet project.

You should also note that there's two additional properties *liferay.theme.parent* and *liferay.theme.type*. These set the parent theme and the theme template language just like in ant based plugins sdk. The property *liferay.theme.parent* however allows you to define basically any war artifact as the parent. The syntax is *groupId:artifactId:version* or you can use the core themes:*_unstyled*, *_styled*, *classic* and *control_panel*.

3) Now you can add your customizations in src/main/webapp. Just follow the same structure as you would do in _diffs. So your custom.css would go to src/main/webapp/css/custom.css.

4) Once you've done your customizations and want to create the war file just run

mvn package
It will create the war file just like with any maven war type project. Another thing it will do is download and copy your parent theme and then overlay your changes on top of it. It will also create a thumbnail from src/main/webapp/images/screenshot.png just like ant based plugins sdk does. These are accomplished by adding the theme-merge and build-thumbnail goals into the generate-sources phase.

5) Now deploy the theme into your Liferay bundle by running:

mvn liferay:deploy
Posted in Liferay Articles, Liferay Examples | Leave a comment

Liferay JSON Service Example

Posted on April 12, 2012

The past couple of years I've seen many examples of using Liferay's built-in JSON services in various ways. The architecture and syntax of this feature has undergone several refinements in the past few versions, so the documentation/examples you can find via a website search are usually slightly wrong and misleading. Recently I saw this thread come alive and decided to sit down and make a non-trival read/write example of using Liferay's JSON services work.

Bear in mind that this applies to Liferay 6.0.x (I am using 6.0.6 CE in the examples). In Liferay 6.1 there are new interfaces coming like (such asRESTful interfaces), ability to do automatic serialization and improved method argument passing, and there are also existing "heavy lifting" web service interfaces like SOAP endpoints that one can use. So this is not the only way to do things. But it is great for prototyping and getting things to work quickly without dragging a bunch of dependencies and debugging hard-to-understand wire protocols. I hope this example is still relevant!

The only dependency I am using here is Apache Commons HTTPClient. I also decided to write it in Java (as opposed to Ray's earlier example on 5.2 in PHP).

Couple of things to be aware of:

- By default, access is through Liferay's tunnel-web web app. So the proper full URL in a default Liferay 6.0.6 install is `http://localhost:8080/tunnel-web/secure/json` .
- Since it is a "secure" (authenticated) interface we need[de] a username and password. This is done using HTTP Basic Authentication, w[priate] for a production environment, since the password is unencrypted (it is in[wing] HTTP Basic Authentication). The default username/password is "test/
- There's no error checking whatsoeve[r] a real world scenario.

First Example

So, here's the first example. A simple "[same] thing as Ray's example, only using Liferay 6 and written in simple Java. It [simply access the "Country"] service and returns a list of country entities known to Liferay.

```
import org.apache.http.HttpHost;
import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.auth.AuthScope;
import org.apache.http.auth.UsernamePasswordCredentials;
import org.apache.http.client.AuthCache;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.client.protocol.ClientContext;
import org.apache.http.impl.auth.BasicScheme;
import org.apache.http.impl.client.BasicAuthCache;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicNameValuePair;
import org.apache.http.protocol.BasicHttpContext;

import java.util.ArrayList;
import java.util.Calendar;
import java.util.List;

public class TestLiferayJSON {

    public static void main(String[] args) throws Exception {

        HttpHost targetHost = new HttpHost("localhost", 8080, "http");
        DefaultHttpClient httpclient = new DefaultHttpClient();
        httpclient.getCredentialsProvider().setCredentials(
                new AuthScope(targetHost.getHostName(), targetHost.getPort()),
                new UsernamePasswordCredentials("test", "test"));

        // Create AuthCache instance
        AuthCache authCache = new BasicAuthCache();
        // Generate BASIC scheme object and add it to the local
        // auth cache
        BasicScheme basicAuth = new BasicScheme();
        authCache.put(targetHost, basicAuth);

        // Add AuthCache to the execution context
        BasicHttpContext ctx = new BasicHttpContext();
        ctx.setAttribute(ClientContext.AUTH_CACHE, authCache);
```

```
        HttpPost post = new HttpPost("/tunnel-web/secure/json");

        List<NameValuePair> params = new ArrayList<NameValuePair>();
        params.add(new BasicNameValuePair("serviceClassName", "com.liferay.portal.service.CountryServi
        params.add(new BasicNameValuePair("serviceMethodName", "getCountries"));
        UrlEncodedFormEntity entity = new UrlEncodedFormEntity(params, "UTF-8");
        post.setEntity(entity);

        HttpResponse resp = httpclient.execute(targetHost, post, ctx);
        resp.getEntity().writeTo(System.out);
        System.out.println();
        httpclient.getConnectionManager().shutdown();
    }
}
```

If you compile and run this (you'll have to download the Apache Commons HTTPClient libraries and put them on the classpath and/or add them as dependencies in your IDE), then you should see something like this on your output screen (this is the returned content from the HTTP POST):

```
[{"countryId":20,"idd":"093","name":"Afghanistan","active":true,"a2":"AF","number":"4","a3":"AFG"},{"c

.....many more countries listed

{"countryId":227,"idd":"263","name":"Zimbabwe","active":true,"a2":"ZW","number":"716","a3":"ZWE"}]
```

Notice where I specify the username/password using Apache HTTPClient APIs. This should be easily translatable to your favorite client (or if you are using curl or some other RESTful client test console such as rest-client and you want to specify the authentication header manually, use an Authorization header with a value of `Basic dGVzdDp0ZXN0Cg==`)

Also note the serviceClassName parameter. This name (`com.lifery.portal.service.CountryServiceUtil`) specifies the service name (and maps to an actual class). Not all services have remote service endpoints (for example, there's no`com.liferay.portlet.social.service.SocialActivityServiceUtil` for creating new activity stream items. I wish there were).

The parameters are encoded into the body of the HTTP POST request using the Apache Commons HTTPClient's`UrlEncodedFormEntity` utility. This is the same as Ray's `$a->addPostData` examples in PHP.

Second Example

Ok, now that the easy one works (it does work for you, right?), let's move on to something trickier using the Web Content system. Notice that this system used to be called "Journal" so all the APIs refer to the Journal Service since the actual APIs were not changed in the interest of compatibility. This second example calls the JournalArticle service to retrieve a sample article using the default install's groupId of 10156 and the articleId of 10455. These numbers are automatically generated during initial startup the first time and may be different for you. If they are you'll need to change them. You can find them through the Control Panel by going to Communities -> Actions -> Edit to find the groupId, and pick any articleId from Web Content.

This example calls a specific method by identifying it using its formal parameters and passes the values for the parameters.

```
    public static void journal() throws Exception {
        HttpHost targetHost = new HttpHost("localhost", 8080, "http");
        DefaultHttpClient httpclient = new DefaultHttpClient();
        httpclient.getCredentialsProvider().setCredentials(
                new AuthScope(targetHost.getHostName(), targetHost.getPort()),
                new UsernamePasswordCredentials("test", "test"));

        // Create AuthCache instance
        AuthCache authCache = new BasicAuthCache();
        // Generate BASIC scheme object and add it to the local
        // auth cache
        BasicScheme basicAuth = new BasicScheme();
        authCache.put(targetHost, basicAuth);

        // Add AuthCache to the execution context
        BasicHttpContext ctx = new BasicHttpContext();
        ctx.setAttribute(ClientContext.AUTH_CACHE, authCache);

        HttpPost post = new HttpPost("/tunnel-web/secure/json");

        // create Liferay API parameters
        List<NameValuePair> params = new ArrayList<NameValuePair>();
        params.add(new BasicNameValuePair("serviceClassName", "com.liferay.portlet.journal.service.Jou
        params.add(new BasicNameValuePair("serviceMethodName", "getArticle"));
        params.add(new BasicNameValuePair("serviceParameters", "[groupId,articleId]"));
        params.add(new BasicNameValuePair("groupId", "10156"));
        params.add(new BasicNameValuePair("articleId", "10455"));
        UrlEncodedFormEntity entity = new UrlEncodedFormEntity(params, "UTF-8");
        post.setEntity(entity);

        // make actual HTTP request and print results to System.out
        HttpResponse resp = httpclient.execute(targetHost, post, ctx);
        resp.getEntity().writeTo(System.out);
        httpclient.getConnectionManager().shutdown();

    }
```

Notice here that the “setup” is exactly the same as before, only the parameters are different. Also note that the list of parameter names (`serviceParameters`) starts and ends with brackets. It’s an array of Strings! So don’t forget the brackets.

The getArticle method returns a JSON-encoded article from Liferay’s web content system, so if this example works for you, you should get this on your output stream:

```
{"urlTitle":"welcome","indexable":true,"statusDate":"1287600093000","type":"general","smallImageId":10
```

Third Example

Ok, now that you a trivial and almost-trivial example, let’s do something more interesting. Let’s add (and remove) a Journal Article (this is what the initial thread asked about anyway).

Here are two methods: `addArticle` and `removeArticle`. They use a hard-coded 60000 for `articleId` to make removal easy. There are a ton of parameters for addArticle (29 to be exact), and since there are

multiple `addArticle` methods in the `JournalArticleServiceUtil` class we have to specify a `serviceParameterTypes` list to tell Liferay which service API we wish to invoke. So the parameter list gets nasty and I didn't do a very good job of coding it to look nice. You can do that though.

```
public static void addArticle() throws Exception {
	HttpHost targetHost = new HttpHost("localhost", 8080, "http");
	DefaultHttpClient httpclient = new DefaultHttpClient();
	httpclient.getCredentialsProvider().setCredentials(
			new AuthScope(targetHost.getHostName(), targetHost.getPort()),
			new UsernamePasswordCredentials("test", "test"));

	// Create AuthCache instance
	AuthCache authCache = new BasicAuthCache();
	// Generate BASIC scheme object and add it to the local
	// auth cache
	BasicScheme basicAuth = new BasicScheme();
	authCache.put(targetHost, basicAuth);

	// Add AuthCache to the execution context
	BasicHttpContext ctx = new BasicHttpContext();
	ctx.setAttribute(ClientContext.AUTH_CACHE, authCache);

	HttpPost post = new HttpPost("/tunnel-web/secure/json");
	Calendar yesterday = Calendar.getInstance();
	yesterday.add(Calendar.DAY_OF_YEAR, -1);
	Calendar nextWeek = Calendar.getInstance();
	nextWeek.add(Calendar.WEEK_OF_YEAR, 1);
	List<NameValuePair> params = new ArrayList<NameValuePair>();
	params.add(new BasicNameValuePair("serviceClassName", "com.liferay.portlet.journal.service.Jou
	params.add(new BasicNameValuePair("serviceMethodName", "addArticle"));
	params.add(new BasicNameValuePair("serviceParameters", "[groupId,articleId,autoArticleId,title
	params.add(new BasicNameValuePair("serviceParameterTypes", "[long,java.lang.String,boolean,jav
	params.add(new BasicNameValuePair("groupId", "10156"));
	params.add(new BasicNameValuePair("articleId", "60000"));
	params.add(new BasicNameValuePair("autoArticleId", "false"));
	params.add(new BasicNameValuePair("title", "Test JSON Article"));
	params.add(new BasicNameValuePair("description", "Test JSON Description"));
	params.add(new BasicNameValuePair("content", "<?xml version='1.0' encoding='UTF-8'?><root avai
			"\ttest content</p>]]></static-content></root>"));
	params.add(new BasicNameValuePair("type", "general"));
	params.add(new BasicNameValuePair("structureId", ""));
	params.add(new BasicNameValuePair("templateId", ""));
	params.add(new BasicNameValuePair("displayDateMonth", "" + (1 + yesterday.get(Calendar.MONTH))
	params.add(new BasicNameValuePair("displayDateDay", "" + yesterday.get(Calendar.DAY_OF_MONTH))
	params.add(new BasicNameValuePair("displayDateYear", "" + yesterday.get(Calendar.YEAR)));
	params.add(new BasicNameValuePair("displayDateHour", "" + yesterday.get(Calendar.HOUR_OF_DAY))
	params.add(new BasicNameValuePair("displayDateMinute", "" + yesterday.get(Calendar.MINUTE)));
	params.add(new BasicNameValuePair("expirationDateMonth", "" + (1 + nextWeek.get(Calendar.MONTH
	params.add(new BasicNameValuePair("expirationDateDay", "" + nextWeek.get(Calendar.DAY_OF_MONTH
	params.add(new BasicNameValuePair("expirationDateYear", "" + nextWeek.get(Calendar.YEAR)));
	params.add(new BasicNameValuePair("expirationDateHour", "" + nextWeek.get(Calendar.HOUR_OF_DAY
	params.add(new BasicNameValuePair("expirationDateMinute", "" + nextWeek.get(Calendar.MINUTE)))
	params.add(new BasicNameValuePair("neverExpire", "false"));
	params.add(new BasicNameValuePair("reviewDateMonth", "" + (1 + nextWeek.get(Calendar.MONTH))))
	params.add(new BasicNameValuePair("reviewDateDay", "" + nextWeek.get(Calendar.DAY_OF_MONTH)));
	params.add(new BasicNameValuePair("reviewDateYear", "" + nextWeek.get(Calendar.YEAR)));
	params.add(new BasicNameValuePair("reviewDateHour", "" + nextWeek.get(Calendar.HOUR_OF_DAY)));
	params.add(new BasicNameValuePair("reviewDateMinute", "" + nextWeek.get(Calendar.MINUTE)));
	params.add(new BasicNameValuePair("neverReview", "false"));
	params.add(new BasicNameValuePair("indexable", "true"));
	params.add(new BasicNameValuePair("articleURL", "articleURL"));
```

```
        params.add(new BasicNameValuePair("serviceContext", "{}"));
        UrlEncodedFormEntity entity = new UrlEncodedFormEntity(params, "UTF-8");
        post.setEntity(entity);
        HttpResponse resp = httpclient.execute(targetHost, post, ctx);
        System.out.println(resp.getStatusLine());
        resp.getEntity().writeTo(System.out);
        httpclient.getConnectionManager().shutdown();

    }

    public static void removeArticle() throws Exception {
        HttpHost targetHost = new HttpHost("localhost", 8080, "http");
        DefaultHttpClient httpclient = new DefaultHttpClient();
        httpclient.getCredentialsProvider().setCredentials(
                new AuthScope(targetHost.getHostName(), targetHost.getPort()),
                new UsernamePasswordCredentials("test", "test"));

        // Create AuthCache instance
        AuthCache authCache = new BasicAuthCache();
        // Generate BASIC scheme object and add it to the local
        // auth cache
        BasicScheme basicAuth = new BasicScheme();
        authCache.put(targetHost, basicAuth);

        // Add AuthCache to the execution context
        BasicHttpContext ctx = new BasicHttpContext();
        ctx.setAttribute(ClientContext.AUTH_CACHE, authCache);

        HttpPost post = new HttpPost("/tunnel-web/secure/json");
        Calendar now = Calendar.getInstance();
        Calendar nextWeek = Calendar.getInstance();
        nextWeek.add(Calendar.WEEK_OF_YEAR, 1);
        List<NameValuePair> params = new ArrayList<NameValuePair>();
        params.add(new BasicNameValuePair("serviceClassName", "com.liferay.portlet.journal.service.Jou
        params.add(new BasicNameValuePair("serviceMethodName", "deleteArticle"));
        params.add(new BasicNameValuePair("serviceParameterTypes", "[long,java.lang.String,java.lang.S
        params.add(new BasicNameValuePair("serviceParameters", "[groupId,articleId,articleURL,serviceC
        params.add(new BasicNameValuePair("groupId", "10156"));
        params.add(new BasicNameValuePair("articleId", "60000"));
        params.add(new BasicNameValuePair("articleURL", "articleURL"));
        params.add(new BasicNameValuePair("serviceContext", "{}"));

        UrlEncodedFormEntity entity = new UrlEncodedFormEntity(params, "UTF-8");
        post.setEntity(entity);
        HttpResponse resp = httpclient.execute(targetHost, post, ctx);
        System.out.println(resp.getStatusLine());
        resp.getEntity().writeTo(System.out);
        httpclient.getConnectionManager().shutdown();

    }
```

More Notes:

- I specified “” (blank) strings for the structureId and templateId. This means that the article has no structure and no template, so it will just be interpreted as raw HTML (as though you had created a new Web Content Article and didn’t specify a structure or template).
- The content parameter is XML with the content inside of a CDATA block. If there were an associated structure for this content, the content would look different. That’s for an advanced reader to explore.
- The timestamps use Java’s Calendar method to set the display date and other dates.
- The articleId is hard-coded to 60000. If you wish to auto-generate specify `autoArticleId` to be `true`.
- The `serviceContext` parameter is special and tricky and not well documented. The “content” of the

parameter is a JSON-serialized instance of the `com.liferay.portal.service.ServiceContext` class. For other services, it might require a more complex serialized instance of the ServiceContext class. For example, instead of `{}` (which, which decoded, results in a simple new instance of the ServiceContext class with `null`/empty/blank/0 for all of its properties), one might have something like `{scopeGroupId:themeDisplay.getScopeGroupId()}` (I took this from this example). Anyway, for this example, an empty `{}` works.

If `addArticle` works, you should get in return a serialized version of the new article:

```
{"urlTitle":"test-json-article","indexable":true,"statusDate":"","type":"general","smallImageId":14540
```

Since `deleteArticle` returns `void`, you won't get anything (But the article should be gone, which you can verify in the GUI, or via a database browser).

Posted in Liferay Examples | Leave a comment

Liferay Cheat Sheet

Posted on April 12, 2012

Click The Below Link Download

http://refcardz.dzone.com/refcardz/essential-liferay-leading-open#refcard-download-social-buttons-display

Posted in Download, Liferay Examples | Leave a comment

Liferay Sample Project

Posted on April 12, 2012

Clcik the Below Link

http://forum.springsource.org/showthread.php?114595-Spring-Portlet-on-Liferay-Sample-Project

Posted in Liferay Examples | Leave a comment

Liferay is a collaboration tool

Posted on April 12, 2012

Liferay Portal is ideal for setting up collaboration environments among workgroups. Whether you call these environments communities or virtual team rooms, Liferay can be used to help your team get their work done. It does this by providing applications that are geared specifically toward document sharing and communicating with one another.
One of the portlets you can add to a page in Liferay is Document Library. This application provides a facility for sharing documents with your entire team. It keeps a complete version history of all your documents and is integrated with Liferay's permissions system. This integration allows you to grant access to shared documents or prevent some of your users from accessing sensitive documents. And if your users need an easier way to access the documents than the web interface provides, Document Library supports WebDAV, allowing documents to be uploaded and downloaded through their operating system's native interface.

Documents are one thing, but what about communication? Liferay's portlets allow for communication in

context, so your users can keep all the relevant information in the right place. Document Library allows your users to create discussion threads next to the documents they need to talk about. Wiki does the same thing. And applications
are provided for both chat and email, so that currently logged-on users can communicate in real time no matter what their physical distance is from one another.

Posted in Liferay Articles | Leave a comment