Liferay Enterprise eCommerce

Adding eCommerce functionality to Liferay
Reading Time: 10 minutes
# Broadleaf + Liferay

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>eCommerce + Portal Options</td>
<td>4</td>
</tr>
<tr>
<td>Integration Details</td>
<td>5</td>
</tr>
<tr>
<td>REST APIs</td>
<td>5</td>
</tr>
<tr>
<td>Integrated</td>
<td>7</td>
</tr>
<tr>
<td>IFrame</td>
<td>8</td>
</tr>
<tr>
<td>Separate</td>
<td>9</td>
</tr>
<tr>
<td>Conclusion</td>
<td>10</td>
</tr>
<tr>
<td>APPENDIX A</td>
<td>11</td>
</tr>
<tr>
<td>Authentication</td>
<td>11</td>
</tr>
<tr>
<td>Portlet SEO URLs</td>
<td>16</td>
</tr>
<tr>
<td>Search</td>
<td>17</td>
</tr>
</tbody>
</table>
Broadleaf + Liferay

An enterprise commerce framework based on best of breed open source technologies

For businesses seeking to implement eCommerce functionality within their current Liferay Portal Solution, Broadleaf Commerce is a Java-based eCommerce framework trusted by Fortune 500 retailers. Broadleaf and Liferay share many of the same technologies, allowing businesses to:

1) **Simplify integration** between systems that use the same architecture,
2) **Leverage investments** of existing development and infrastructure, and
3) **Increase revenues** by providing additional eCommerce capabilities.

Liferay customers know the tremendous benefits of building a solution on the Liferay platform, and Broadleaf shares many of the same attributes, including:

<table>
<thead>
<tr>
<th>Cost Effective</th>
<th>Enterprise Editions</th>
<th>Fortune 500 Clients</th>
</tr>
</thead>
<tbody>
<tr>
<td>Highly Extensible</td>
<td>Active Communities</td>
<td>Strong Partner Network</td>
</tr>
<tr>
<td>Spring Framework</td>
<td>Proven Performance</td>
<td>Open Source Software</td>
</tr>
</tbody>
</table>

Simply stated, Broadleaf Commerce and Liferay are glove fits with regards to technology, process and people. The methods of integration, however, are varied, and highly dependent upon specific business and technology needs.

This paper examines four approaches for integrating Liferay with an eCommerce platform – using REST APIs, through an Integrated infrastructure, using IFrame technology, and through a Separate infrastructure.
Introducing eCommerce functionality into an existing portal implementation can be done in a variety of ways. The right answer varies by company and depends largely on the eCommerce and portal needs of the company.

Both Broadleaf Commerce and Liferay are frameworks for building highly customized applications. As such, both can be altered to fit integration methods based on needs. The chart below gives a brief overview of each integration method while identifying the major benefits and detriments to the integration options.

**NOTE:** Minus indicates such feature as stated is more difficult.
Integration Details

The following are examples of how companies can utilize the eCommerce functionality of Broadleaf and the site-building features of Liferay. While each option is presented here as *mutually exclusive* options, an amalgamation between different approaches can also be used. For instance, if companies decide to use the separated approach you can still use the Broadleaf REST APIs to hook into core Broadleaf functionality *from pre-developed portlets*.

**REST APIs**

Integrating with Broadleaf via REST endpoints usually involves deploying Broadleaf on a separate application server from Liferay. Portlets can then be developed in any controller-level (Spring Portlet MVC, JSF, Struts) and view-level (Thymeleaf, JSP) frameworks the users are most comfortable with since under the covers the REST endpoints being utilized are agnostic. These endpoints can also be immediately used in any custom Liferay overrides such as model listeners or service builder.
Pros of the REST API Approach

- Portlet framework agnostic (Spring MVC, JSF, Struts, etc)
- No additional work to allow other systems to use the Broadleaf features as they can all use the same services
- Scaling can be differentiated between Broadleaf and the Liferay portal if Broadleaf and eCommerce is a relatively small portion of your entire portal needs (or vice-versa)
- Easy to bring in additional external systems or Liferay plugins to communicate with Broadleaf in the future
- Separate upgrade paths between Broadleaf and Liferay

Cons of the REST API Approach

- Additional work developing a client-side framework to interact with the Broadleaf endpoints along with the Liferay portlets themselves. This can be mostly mitigated by automatic processes that can build Java classes based on the Broadleaf endpoint definitions (Jersey WADLs)
- Potential for additional application server hardware

Other Considerations

- All user requests to view any portlets developed in this fashion will require an additional HTTP request to the Broadleaf REST endpoints. If both the Broadleaf endpoints and Liferay are deployed to the same Tomcat instance or elsewhere on the same physical server then each user HTTP request will actually require 2 request threads (at a minimum): one to request the portlet page from Liferay and another to get a response back from any Broadleaf endpoint. This has the potential to impact scalability.
- If customers wish to utilize the Broadleaf endpoints publicly, additional security concerns might have to take place (such as API keys or OAuth). These concerns are outside of the scope of this paper.
**Integrated**

This integration involves bundling the Broadleaf dependencies directly inside of a web application archive (with potentially multiple portlets) and then deploying this web application archive on top of Liferay. This solution is really only recommended if customers have very few (if any) custom Liferay extensions that need to additionally utilize eCommerce functionality.

**Pros of an Integrated Approach**

- Ability to inject Broadleaf services and beans directly inside of your portlet controllers
- Single codebase to work from

**Cons of an Integrated Approach**

- Large portlet web application archive (~80MB) since all Broadleaf code and dependencies must be included
- Inability to inject Broadleaf Spring beans and services for use in other portlets located in different web application archives
- Inability to inject Broadleaf Spring beans and services for use in any core Liferay extensions (such as Liferay’s Service Builder or Model Listener)
- Scalability between Liferay and Broadleaf is very tightly coupled
- Lock-step upgrades between Liferay and Broadleaf that can additional challenges
- Spring Portlet MVC and Thymeleaf required
**IFrame**

The IFrame integration involves developing and deploying Broadleaf as a normal servlet application deployed on a different instance and available through a different URL. The IFrame portlet can simply point to a URL handled by the Broadleaf instance and display the results within a Liferay page.

**Pros of an IFrame Approach**

- Separate upgrade paths for Broadleaf and Liferay
- Minimal (if any) portlet development required

**Cons of an IFrame Approach**

- Stylesheet duplication between Broadleaf and Liferay
- Inability to provide a seamless UX experience
Separate

The separate approach is similar to the IFrame integration except that there is no extra IFrame portlet within Liferay to display any Broadleaf pages alongside of other portlets. This is by far the simplest but also most limited solution.

Pros of a Separate Approach

• No configuration required between Broadleaf and Liferay
• Independent scaling
• Separate upgrade paths between Liferay and Broadleaf

Cons of a Separate Approach

• Shared authentication requires an SSO type of solution
• Requires building two completely different sites
• Duplicated style development if consistent UX is a concern
• Consolidated search is difficult between the disparate systems
Conclusion

Broadleaf Commerce integrates with Liferay through multiple methods. Whether through REST APIs, Direct, IFrame or Separate integrations, Liferay portlets can be used to process eCommerce transactions using the Broadleaf Commerce framework. With a shared technology stack and development tools, businesses already running Liferay can leverage their infrastructure investment and developer base to provide Broadleaf’s best of breed eCommerce toolset across a variety of catalog, cart and checkout needs.

Integrating Broadleaf with a Liferay portal allows businesses to continue leveraging Liferay for integrated content, and Broadleaf for eCommerce transactions. Through technical methods outlined in this paper Broadleaf and Liferay prove business value through:

1) **Simplifying integration** between systems that use the same architecture,
2) **Leveraging investments** of existing development and infrastructure, and
3) **Increasing revenues** by providing additional eCommerce capabilities.

For more information on the web visit: [http://www.broadleafcommerce.com/liferay](http://www.broadleafcommerce.com/liferay)
Or email: [info@broadleafcommerce.com](mailto:info@broadleafcommerce.com)
APPENDIX A

Common challenges between all integrations

**Authentication**

Specifically, this section presents solutions to associating a Broadleaf Customer to the Liferay User so that orders can stay in sync between users.

**REST Authentication Considerations**

Liferay’s user attributes (reference: [http://www.liferay.com/community/wiki/-/wiki/Main/How%20to%20use%20Custom%20Attributes%20to%20enhance%20Users](http://www.liferay.com/community/wiki/-/wiki/Main/How%20to%20use%20Custom%20Attributes%20to%20enhance%20Users)) provide a convenient mechanism to store the Broadleaf customer id. From the Liferay portlet this looks something like:

```java
User liferayUser = UserServiceUtil.getUserById(Long.parseLong(request.getRemoteUser()));
if (liferayUser.getExpandoBridge().getAttribute("broadleafCustomerId") == null) {
    OrderWrapper wrapper = restTemplate.postForObject("http://broadleaf-instance/cart", null, OrderWrapper.class);
    liferayUser.getExpandoBridge().setAttribute("broadleafCustomerId", wrapper.getCustomer().getId());
}
```

After this association has been made, subsequent calls to the Broadleaf REST endpoints that require a Broadleaf customer ID can now utilize the ‘broadleafCustomerId’ Liferay custom user attribute. For example, to obtain the current Liferay user’s cart sometime later:

```java
Map<String, Object> vars = new HashMap<String, Object>();
vars.put("customerId",
liferayUser.getExpandoBridge().getAttribute("broadleafCustomerId");
restTemplate.getForObject("http://broadleaf-instance/cart", OrderWrapper.class, vars);
```
This relationship can also be realized from the Broadleaf endpoints by overriding Broadleaf's RestAPICustomerStateFilter. By default, this filter checks for a ‘customerId’ request parameter. Assuming that the Broadleaf Customer has been extended to store the Liferay user id like so:

```java
@Table("LIFEREY_CUSTOMER")
@Entity
public class LiferayCustomerImpl extends CustomerImpl {
    @Column(name = "LIFEREY_USER_ID")
    protected Long liferayUserId;

    public Long getLiferayUserId() {
        return liferayUserId;
    }

    public void setLiferayUserId(Long liferayUserId) {
        this.liferayUserId = liferayUserId;
    }
}
```


Then a service can be created to actually look up a Broadleaf customer based on the Liferay user id:

```java
public LiferayCustomerService {
    public Customer findCustomerByLiferayUser(Long liferayUserId) {
        //invoke dao to execute database query
    }
}
```
And finally, the filter could be extended to look for a ‘liferayUserId’ request parameter:

LiferayAPICustomerStateFilter extends RestAPICustomerStateFilter {
    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException {
        String userId = servletRequest.getParameter("liferayUserId");
        if (userId != null) {
            Long liferayUser = Long.parseLong(userId);
            LiferayCustomer liferayCustomer = liferayCustomerService.findCustomerByLiferayUser(liferayUser);
            if (liferayCustomer == null) {
                liferayCustomer = customerService.createCustomer();
                liferayCustomer.setLiferayUserId(liferayUser);
            }
            CustomerState.setCustomer(liferayCustomer);
            filterChain.doFilter(request, servletResponse);
        } else {
            super.doFilter(servletRequest, servletResponse, filterChain);
        }
    }
}

**Integrated Authentication Considerations**

Security via the integrated approach involves using Spring Security interceptors. These interceptors are responsible for duplicating the Broadleaf servlet filters (that cannot be used in portlets) and correctly associating a Broadleaf customer with the Liferay User on each request. If the integrated solution appeals to you, contact Broadleaf at info@broadleafcommerce.com for more information on these Spring Security interceptors.
**IFrame Authentication Considerations**

Liferay’s out of the box IFrame portlet only supports 2 types of authentication: HTTP basic and form-based. However, in both cases the username and password from Liferay is only sent on the first request to load the initial IFrame. If a user clicks through on a link within the IFrame itself, the Liferay instance does not send the authentication information again.

In order to save the current user's information from Liferay, one solution is to have Broadleaf store the Liferay user in session to be retrieved at a later date. Assume that you have configured a Liferay IFrame portlet with the following values:

Authentication Type: **FORM**  
Form Method: **GET**  
User Name: liferayEmail=@email_address@

Similar to the REST authentication solution you will need a custom override to the Broadleaf Customer:

```java
@Table("LIFERAY_CUSTOMER")
@Entity
public class LiferayCustomerImpl extends CustomerImpl {
    @Column(name = "LIFERAY_EMAIL")
    protected String liferayEmail;

    public String getLiferayEmail() {
        return liferayUserId;
    }

    public void setLiferayEmail(String liferayEmail) {
        this.liferayEmail = liferayEmail;
    }
}
```
and an override to the CustomerStateFilter:

LiferayAPICustomerStateFilter extends CustomerStateFilter {
    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException {
        String liferayEmail = servletRequest.getParameter("liferayEmail");
        if (liferayEmail != null) {
            LiferayCustomer liferayCustomer = liferayCustomerService.findCustomerByLiferayUser(liferayUser);
            if (liferayCustomer == null) {
                liferayCustomer = customerService.createCustomer();
                liferayCustomer.setLiferayEmail(liferayEmail);
            }
            CustomerState.setCustomer(liferayCustomer);
            filterChain.doFilter(request, servletResponse);
        } else {
            super.doFilter(servletRequest, servletResponse, filterChain);
        }
    }
}

Separate Authentication Considerations

When Liferay and Broadleaf are in completely different containers Broadleaf recommends integrating with an SSO provider that Liferay integrates with such as SAML, CAS or OpenID.
Portlet SEO URLs

When developing portlets (via either the REST or integrated option) Liferay provides a FriendlyURLMapper which can be referenced through liferay-portlet.xml and corresponding route configuration (reference: http://www.liferay.com/community/wiki/-/wiki/Main/FriendlyURLMapper). For instance, routes can be configured to handle catalog URLs that have been configured through the Broadleaf admin. Consider the following liferay-portlet.xml and routes.xml configuration for a catalog portlet (designed to handle both category and product URLs):

liferay-portlet.xml

```xml
<portlet>
  <portlet-name>broadleaf-catalog</portlet-name>
  <icon>/icon.png</icon>
  <friendly-url-mapper-class>com.liferay.portal.kernel.portlet.DefaultFriendlyURLMapper</friendly-url-mapper-class>
  <friendly-url-mapping>catalog</friendly-url-mapping>
  <friendly-url-routes>catalog-routes.xml</friendly-url-routes>
  <instanceable>false</instanceable>
</portlet>
```

routes.xml

```xml
<routes>
  <route>
    <pattern>{catalogUrl}</pattern>
  </route>
</routes>
```

Now all catalog URLs will be of the form:
http://yourcompany.com/-/catalog/category-url
http://yourcompany.com/-/catalog/product-url
...
Note: The above approach is only available via the integrated and REST approaches. Not included here is also the MVC implementation of receiving the ‘catalogUrl’ request parameter within the catalog portlet and responding appropriately.

Additional references: http://www.liferay.com/community/wiki/-/wiki/Main/FriendlyURLMapper

Search

Integrated search is available through hooking into Liferay’s Faceted Search capabilities as you would any other custom entity. Details surrounding specific search Indexer implementations are outside the scope of this document. Liferay provides an excellent guide for implementing faceted search for custom objects at http://www.liferay.com/web/raymond.auge/blog/-/blogs/faceted-search-in-liferay-6-1.

Note: since integrated search uses a Liferay extension this is only available via the REST integration method.