Cahier de TP
« Liferay 7.x - Développer un portail d'entreprise»

Pré-requis :
Poste développeur avec accès réseau Internet libre
Linux (Recommandé) ou Windows 10
Pré-installation de :

- Git
- JDK11+
- npx, npm
- IDEs : IntelliJ

Table des matières

Présentation projet fil rouge :

Progressivement, nous allons construire un mini-portail **“EventPortal”** composé :

- d’un site dédié
- de pages de navigation
- de contenus CMS structurés
- d’un mécanisme de staging opérationnel
- d’une portlet custom
- d’une UI modernisée Clay
- d’une entité Service Builder (“Event”)
- de permissions fines
- d'extensions OSGi pour personnalisation

Atelier 1: Installation Liferay 7.4 + Base de données

Objectifs

- installer un bundle Liferay 7.4 avec une base de production

1.1 Installation d'une base Postgres

Démarrer un serveur postgres et sa console d'administration

```
docker compose -f postgres-docker-compose.yml up -d
```

Accéder à localhost:81 et se logger avec admin@admin.com/admin

Enregistre un serveur avec comme paramètres de connexion :

- host : ***liferay-postgresql***
- user : ***postgres***
- password : ***postgres***

Créer :

- une base ***liferay***
- un utilisateur ***liferay*** avec tous les privilèges

```
CREATE USER liferay_user WITH PASSWORD 'liferay';

CREATE DATABASE liferay
  WITH OWNER liferay_user
  ENCODING 'UTF8'
  CONNECTION LIMIT -1;


GRANT ALL PRIVILEGES ON DATABASE liferay TO liferay_user;
```

1.2 Démarrage Bundle Tomcat

Télécharger la distribution Liferay 7.4 (bundle Tomcat) et l'extraire dans un répertoire de travail

Placer le fichier ***portal-ext.properties*** dans le répertoire d'extraction avec :

```
jdbc.default.driverClassName=org.postgresql.Driver
jdbc.default.url=jdbc:postgresql://localhost/liferay
jdbc.default.username=liferay_user
jdbc.default.password=liferay
```

Démarrer Tomcat

```
./tomcat/bin/startup.sh
```

Surveiller le démarrage :

```
tail -f ./tomcat/logs/catalina.out
```

Vérifications :

- ***Accès :*** `http://localhost:8080`, Vous devez voir la page d'accueil de Liferay 7.4 (assistant de configuration).

- Base de données : Visualiser les tables créées

1.2 Assistant de configuration initiale

1. Dans le navigateur, suivez les écrans de l'assistant :
    - choisissez la **langue** (par ex. Français),
    - définissez le **nom du portail** comme ***EventPortal***
    - créez le **compte administrateur** (adresse e-mail, mot de passe).
2. Effectuer un redémarrage et connectez-vous avec le compte administrateur créé.

Vérifications **:**
Accès à la page d'accueil du portail, avec le menu d'administration (icône "roue dentée" ou "Produits") disponible pour votre utilisateur.

Atelier 2 : Sites, Pages et Navigation

Objectifs :

- Découvrir la gestion des pages dans Liferay.
- Comprendre la différence entre *Page de contenu* et *Page widget*.
- Construire la structure de navigation d'un site d'entreprise.
- Manipuler les URLs conviviales (*Friendly URLs*) utilisées pour le référencement et l'ergonomie.
- Préparer l'architecture du site qui servira de support à tout le fil rouge.

2.1 Lister les sites

Commencer par lister les site

Menu > Panneau de contrôle > Sites
2 sites doivent apparaître :

- **EventPortal**
- **global :** Ce site est un site technique utilisé pour rassembles les assets réutilisables

2.2 Création de pages

Nous voulons créer différents types de pages :

- ***Home*** : La page d'accueil
- ***Recherche*** : La page Widget permettant d'afficher les résultats d'une recherche full-text
- ***Actualités*** : Une page Widget permettant le rendu de portlets, elle contient une sous-page :
  - ***Archives***
- ***Évènements*** : Un ensemble de page contenant les pages :
  - ***Présentation***
  - ***Liste des évènements***
  - ***Ajouter un événement***
- ***Contact*** : Une page Widget

Aller dans ***"Créateur de site" → "Pages"***

Configurer la page Home existante

- Nom : **Home**
- Friendly URL : /home

Pour la page Actualités

1. **Ajouter → Page de widget**
2. Nom : **Actualités**

Dans général :

- Disposition : 1 colonne
- URL conviviale : **`/actualites`**

La page Événements (Page de contenu)

1. **Ajouter → Page vierge**
2. Nom : **Événements**
3. Cliquer sur **Publier** sans ajouter de contenu pour le moment

URL conviviale : **`/evenements`**

La page Contact (Widget)

1. Ajouter → **Page de widget**
2. Nom : **Contact**

Disposition : 1 colonne
URL conviviale : **`/contact`**

2.3 Organiser la navigation

Depuis **Créateur de site → Pages** :

Modifier l'ordre

Glisser-déposer les pages pour obtenir :

1. Home
2. Actualités
3. Événements
4. Contact
5. Recherche (si créée maintenant)

Créer une sous-page

1. Sélectionner **Événements**
2. Ajouter → **Page de widget**
3. Nom : **Archives**
4. Publier

Définir les pages publiques / privées

Via **Configurer :**

- Contact ou Événements → Supprimer la permission Voir au rôle Guest

Atelier 3 : Fonctions CMS : contenus, structures & templates

Objectifs

- Mettre à jour une page de contenu du portail (Home ou Présentation des Événements)
- Créer un contenu structuré avec une *Structure* et un *Template*
- Publier une actualité conforme à ce modèle
- Afficher un contenu :
    - via un widget Web Content Display,
    - via un Fragment Web Content,
    - ou via une Display Page (méthode moderne recommandée)
- Utiliser les Documents & Media pour intégrer une image dans une page

3.1 Mise à jour d'un contenu statique

Essayer l'éditeur de CMS pur sur la page d'accueil par exemple :

- Ajout de texte
- Ajout d'image

3.2. Créer une structure nommé Actualité

Contenu → Contenu web → Structures

- Nom : **Actualité**

Champs :

- *titre (text)*
- *imagePrincipale (image)*
- *accroche (text)*
- *texte (rich text)*
- *datePublication (date)*

3.2 Créer un gabarit Freemarker

Activer l'onglet Modèle de document et créer un modèle associé par défaut à la structure Actualité

Nom du template : **Actualité par défaut FTL**

```
<h1>${titre.getData()}</h1>
<img src="${imagePrincipale.getData()}" />
<p>${accroche.getData()}</p>
<div>${texte.getData()}</div>
```

```
<p>Publié le ${datePublication.getData()}</p>
```

3.3 Créer un Web Content utilisant la structure

- Ajouter un article “Actualité numéro 1”
- Remplir les champs
- Publier

3.4. Afficher l’article dans une page

Méthode 1 : via Web Content Display (classique)

1. Naviguer dans la **Pages → Actualités**
2. Activer le bouton + pour ajouter du contenu
3. Sélectionner **Affichage de contenu web**
4. Configurer en sélectionnant l’actualité précédente
5. Enregistrer / Publier

Méthode 2 : via Fragment “Web Content”

1. Modifier la page Home
2. Ajouter un **fragment → Web Content**
3. Sélectionner le contenu créé
4. Publier

Méthode 3 : via Modèle de page d’affichage

Créer un modèle de page d’affichage

1. Aller dans ***Conception → Modèles de page → Modèle de page d’affichage***
2. Ajouter un modèle, le nommer : **Actualité – Page d’affichage**
3. Utiliser des fragments dynamiques pour afficher les champs de la structure Actualités
4. Ajouter un bouton **Retour aux actualités**
5. Publier

Associer le Modèle de page d’affichage à la structure

1. Retourner dans votre structure actualité
2. Cliquer sur ***Détails → Affichage → Page de visualisation***
3. Sélectionner : *Actualité – Page d’affichage*
4. Enregistrer

Définir une liste de contenus **Actualités archivées**

1. ***Créateur de sites → Liste de contenus***
2. ***Nouveau → Collection Dynamique*** **: Actualités archivées**

3. ***Contenu web → Actualités***
4. **Éventuellement les autres champs**

Sur la page *Archives* :

1. Ajouter un widget **Agrégateur de contenu (Asset Publisher)**
2. Choisir la collection que l'on vient de créer
3. Créer une nouvelle actualité et vérifier que le modèle de page d'affichage y est bien associé
4. Accéder à la page archive et Cliquer sur un élément

Atelier 4 : Tables fondamentales de Liferay

Objectifs

À la fin de ce TP, les stagiaires sauront :

- Identifier les tables essentielles de Liferay
- Comprendre la relation entre sites, pages, utilisateurs, rôles et contenus
- Exécuter des requêtes SQL pour inspecter la structure interne d'une installation Liferay
- Retrouver les identifiants internes indispensables pour diagnostiquer un problème ou croiser des données
- Comprendre la différence entre **Content Page**, **Widget Page**, **Layout**, **LayoutSet**, **JournalArticle**, **DLFileEntry**, etc.

4.1. Les sites

Les sites dans Liferay sont stockés dans la table :

```
SELECT
  g.groupId,
  g.companyId,
  g.friendlyURL,
  c.name AS company_name
FROM "group_" g
JOIN "company" c ON c.companyId = g.companyId
WHERE g.site = TRUE
ORDER BY g.groupId;
```

Utilisé le ***groupId*** dans la suite des requêtes

4.2 Utilisateurs et rôles

Utilisateurs

```
SELECT userId, screenName, emailAddress, firstName, lastName,
status
FROM "user_"
ORDER BY userId;
```

Rôles

```
SELECT roleId, name, type_
FROM "Role_"
ORDER BY roleId;
```

Interprétation `Role_.type_`

type_	Signification
1	Site Role

type_	Signification
2	Organization Role
3	Regular Role
4	Depot Role

Association User/Rôle

```
SELECT u.screenName, r.name AS roleName
FROM "Users_Roles" ur
JOIN "User_" u ON u.userId = ur.userId
JOIN "Role_" r ON r.roleId = ur.roleId
ORDER BY u.screenName;
```

4.3 Les pages

Les pages sont stockées dans :

- **`Layout`** → une ligne par *page*
- **`LayoutSet`** → Conteneurs de pages 2 par sites
- Permissions

Pages

```
SELECT layoutId, plid, parentLayoutId, name, friendlyURL, type_,
hidden, priority
FROM "layout"
WHERE groupId = groupId
  AND privateLayout = FALSE
ORDER BY parentLayoutId, priority;
```

Interprétation :

- `type_` :
  - `content` → Page de contenu
  - `portlet` → Page de widgets
  - `asset_display` → Modèle de page de visualisation
  - `utility` → Page système
  - `node` → Ensemble de page
- `friendlyURL` → /home, /actualites…
- `parentLayoutId` → hiérarchie
- `priority` → ordre

Les conteneurs de page

```
SELECT layoutId, plid, parentLayoutId, name, friendlyURL, type_,
hidden_, priority
```

```
FROM "layout"
WHERE groupId = <groupId>
  AND privateLayout = FALSE
ORDER BY parentLayoutId, priority;
```

Tu verras deux lignes :

- `privateLayout = false` → pages publiques
- `privateLayout = true` → pages privées (même si tu n'en utilises plus en 7.4)

Association Layout / LayoutSet

```
SELECT
  l.layoutId,
  l.friendlyURL,
  l.type_,
  ls.layoutSetId,
  ls.privateLayout,
  ls.themeId
FROM "layout" l
JOIN "layoutset" ls
  ON ls.groupId = l.groupId
 AND ls.privateLayout = l.privateLayout
WHERE l.groupId = <groupId>;
```

Permissions sur les pages

Les permissions sont stockées dans :

- **`ResourcePermission`** → qui peut faire quoi sur quoi
- **`ResourceAction`** → quelles actions existent
- **`Role_`** → les rôles
- **`Layout`** → les pages

Exemple : page /actualites

```
SELECT
    rp.roleId,
    r.name AS roleName,
    rp.actionIds,
    rp.primKey
FROM "ResourcePermission" rp
JOIN "Role_" r ON r.roleId = rp.roleId
WHERE rp.name = 'com.liferay.portal.kernel.model.Layout'
  AND rp.primKey LIKE (
```

```
        SELECT CAST(plid AS VARCHAR)
        FROM "Layout"
        WHERE friendlyURL='/actualites'
          AND groupId = <groupId>
  );
```

actionIds est un *bit mask* de la table resourceaction

Plus lisible :

```
SELECT
  l.plid,
  l.friendlyURL,
  l.name AS page_name,
  r.roleId,
  r.name AS roleName,
  string_agg(ra.actionId, ', ' ORDER BY ra.actionId) AS
allowedActions
FROM "resourcepermission" rp
JOIN "role_" r
  ON r.roleId = rp.roleId
JOIN "layout" l
  ON l.plid::text = rp.primKey
JOIN "resourceaction" ra
  ON ra.name = rp.name
 AND (rp.actionIds & ra.bitwiseValue) <> 0
WHERE rp.name = 'com.liferay.portal.kernel.model.Layout'
  AND rp.scope = 4              -- permissions individuelles
  AND l.groupId = 20117         -- site Event Portal
  AND l.privateLayout = FALSE   -- pages publiques
GROUP BY
  l.plid,
  l.friendlyURL,
  l.name,
  r.roleId,
  r.name
ORDER BY
  l.friendlyURL,
  r.name;
```

4.4. Contenus : Web Contents + leurs structures

Lister les contenus

```
SELECT articleId, urltitle, version, status, userName
FROM "journalarticle"
WHERE groupId = 20117;
```

Association structure contenu

```
SELECT ja.articleId, ja.urltitle, ds.name
FROM "journalarticle" ja
JOIN "ddmstructure" ds ON ds.structureid = ja.ddmstructureid
WHERE ja.groupId = 20117;
```

4.5 Documents & Media

Documents stockés dans :

- **DLFileEntry :** Un document
- **DLFileVersion :** Versionning
- **DLFolder :** Un répertoire

```
SELECT fileEntryId, title, mimeType, groupId, folderId
FROM "dlfileentry"
WHERE groupId = 20117;
```

Atelier 5 : Gestion de la configuration

Objectif

Tester les 3 techniques de configuration :

- **portal-ext.properties**
- **OSGi .config**
- **modification via System Settings**

5.1 Propriétés de portal-ext

Ajouter dans ***portal-ext.properties*** :

```
theme.css.fast.load=false
```

Redémarrer et constater le changement.

Observation attendue

- Les CSS ne sont plus concaténées/minifiées
- Le chargement front est plus verbeux (utile en debug)

5.2. Configuration OSGi

Configuration via l'UI

Panneau de contrôle > Paramètres Système > Contenu web

- Activer/désactiver "Enable comments"

Observation

- Le changement est immédiat
- Aucun redémarrage requis

Analyse

- La modification met à jour la table configuration

```
SELECT
  configurationId,
  dictionary
FROM configuration_;
```

Configuration via fichiers :

Vérifier la valeur de la case à cocher « Index de toutes les versions des articles activé »

Sortir de la page de configuration

Créer le fichier :

`com.liferay.journal.configuration.JournalServiceConfiguration.config`

Avec comme contenu :

```
indexAllArticleVersionsEnabled="false"
```

Retrouver l’effet dans l’UI

5.3 Verbosité des logs

Panneau de contrôle | Système | Administration du serveur

Visualiser les loggers

Passer ***com.liferay*** en mode DEBUG et visualiser l’effet sur ***catalina.out***

Atelier 6 : Exploitation des modules OSGI

Objectifs

- Utiliser la Gogoshell
- Lister les modules OSGi
- Identifier un module Liferay
- Arrêter et redémarrer un module
- Observer l'impact immédiat dans l'interface utilisateur

Accéder au Gogo Shell

Panneau de contrôle → Système → Gogo Shell

Taper :

```
help
lb -s
```

Identifier l'ID du bundle ***com.liferay.journal.web***

Arrêter le module :

```
stop <ID_DU_BUNDLE>
```

Observer l'impact dans le menu Contenu de l'administration de site

```
start <ID_DU_BUNDLE>
```

Atelier 7 : Staging + Export/Import

Objectif

Mettre en place du staging, publier un contenu, tester import/export.

7.1 Staging

Activer le staging

Menu du site > En cours de publication > Organiser

- Choisir **Local Staging**
- Sélectionner pages & contenus à mettre en staging

Ajouter du contenu dans le staging

- Editer la page Home et ajouter une “Bannière Cookie”
- Publier sur le staging
- Comparer le staging (pré-production) et le live (en utilisant un autre navigateur ou un onglet privé)

Publier sur le vrai site

- Publier sur le site
- Vérifier l’apparition dans le site live

7.2 Export / Import

- Exporter le site dans un `.lar`
- Importer dans un nouveau site “EventPortal_Copy”

Aller sur le vrai site, effectuer un export puis télécharger le fichier .lar

Créer une nouveau site « Copie »

Effectuer ensuite une importation sur le site « Copie »

Retrouver les pages

Atelier 8 : Développement Portlet MVC simple

Objectif

Créer une portlet “HelloEvent”.

8.1 Mise en place du workspace

Installer blade cli

https://learn.liferay.com/w/dxp/development/tooling/blade-cli

Vérifier avec

```
blade -v
```

Vous placer dans un répertoire de travail et exécuter :

```
blade init liferay-workspace
```

8.2 Import du workspace dans l’IDE

1. Ouvrez votre IDE (Eclipse/Liferay Developer Studio ou IntelliJ).
2. Importez le workspace :
    - Eclipse/Liferay Developer Studio :
        - **File > Import… > Existing Gradle Project**,
        - sélectionnez le dossier `my-liferay-workspace`.
    - IntelliJ :
        - **Open…**,
        - sélectionnez `my-liferay-workspace` et laissez IntelliJ détecter le projet Gradle.
3. Attendez la fin de l’indexation et de la synchronisation Gradle.

Contrôle :
Dans la vue de projets, vous voyez le projet `my-liferay-workspace` et son sous-répertoire `modules` (vide pour l’instant).

8.3 Création d’un module portlet

Créer le module MVC

```
cd liferay-workspace
blade create -t mvc-portlet -p org.formation.hello -c HelloPortlet hello
```

- `-t mvc-portlet` : type de module,
- `-p org.formation.hello` : package Java,

- `-c HelloPortlet` : nom de la classe portlet,
- `hello` : nom du module.

Vérifier la création du module dans votre IDE

Vérifiez dans l'IDE :

- un nouveau projet `hello` apparaît sous le dossier `modules`,
- la classe `HelloPortlet` existe dans `src/main/java/org/formation/hello`.

Ouvrez la vue JSP principale :
`hello/src/main/resources/META-INF/resources/view.jsp`.

Remplacez le contenu de `view.jsp` par quelque chose de simple, par exemple :

```
<%@ page import="com.liferay.portal.kernel.util.HtmlUtil" %>
    <%@ taglib uri="http://java.sun.com/portlet_2_0" prefix="portlet" %>
    <%@ taglib uri="http://liferay.com/tld/theme" prefix="liferay-theme" %>

    <liferay-theme:defineObjects />
    <portlet:defineObjects />

    <h1>Hello Liferay 7.4 !</h1>
    <p>
        Ce portlet a été créé dans un module OSGi à l'aide de Blade CLI.<br/>
        Utilisateur connecté : <strong><%=
    HtmlUtil.escape(themeDisplay.getUser().getFullName()) %></strong>
    </p>
```

Contrôle :

Le code compile dans l'IDE (pas d'erreur rouge dans la classe ni dans la JSP).

Déployer

Configurer la cible de déploiement en ajoutant dans le fichier gradle.properties du workspace :

```
liferay.workspace.home.dir=/chemin/vers/ton/liferay
```

Assurer vous que le serveur est démarré

Puis effectuer la commande de déploiement dans le répertoire du workspace

```
./gradlew :modules:hello-web:deploy
OU
./gradlew deploy
```

Contrôle :

Dans le répertoire `deploy` du Liferay Home, le JAR du module `hello-event` doit apparaître, puis être déplacé automatiquement une fois déployé.

Dans les traces du serveur, on doit pouvoir voir la ligne :

```
Started org.formation.helloEvent_1.0.0
```

8.4. Ajouter la portlet

1. Connectez-vous au portail avec votre compte administrateur
2. Dans la page Home de notre portail .
3. Dans le menu d'ajout d'applications, recherchez le portlet :
    - nom de la catégorie : selon la configuration du module (souvent "Exemple" ou "hello"),
    - nom du portlet : **HelloPortlet** ou libellé défini dans le bundle de ressources.
4. Ajoutez le portlet à la page.
5. Vérifiez l'affichage :
    - le message "Hello Liferay 7.4 !" apparaît,
    - le nom de l'utilisateur connecté est affiché.

Contrôle final :

Si le portlet s'affiche correctement et montre votre nom d'utilisateur connecté, le TP est réussi.

Ateliers 9 : Développement d’une Portlet MVC avec validation

Objectifs

- Implémenter un `MVCActionCommand`
- Effectuer la **validation serveur**
- Gérer des erreurs utilisateur (`SessionErrors`)
- Naviguer entre écrans via `mvcPath`

9.1 Création du module

Depuis le workspace :

```
blade create -t mvc-portlet \
  -p org.formation.event \
  -c EventPortlet \
  event
```

Modifier `view.jsp`

Créer une zone d'affichage + lien vers l’écran d’édition :

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>

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


<h1>Liste des événements</h1>

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

<a href="${editURL}">Ajouter un événement</a>
```

Créer `edit.jsp`

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<%@ taglib uri="http://java.sun.com/portlet_2_0" prefix="portlet" %>
<%@ taglib uri="http://liferay.com/tld/ui" prefix="liferay-ui" %>

<h2>Ajouter un événement</h2>
```

```
<liferay-ui:error key="title-required" message="Le titre est
obligatoire"/>

<portlet:actionURL name="saveEvent" var="saveURL" />

<form action="${saveURL}" method="post">
   <label>Titre :</label>
   <input type="text" name="title" />
   <button type="submit">Enregistrer</button>
</form>
```

9.2 Implémenter l'Action Command

Créer :

`src/main/java/org/formation/event/action/SaveEventMVCActionCommand.java`

```
@Component(
    property = {
        "javax.portlet.name=org_formation_event_EventPortlet",
        "mvc.command.name=saveEvent"
    },
    service = MVCActionCommand.class
)
public class SaveEventMVCActionCommand implements MVCActionCommand {

    @Override
    public boolean processAction(ActionRequest req, ActionResponse resp) {

        String title = ParamUtil.getString(req, "title");

        if (Validator.isNull(title)) {
            SessionErrors.add(req, "title-required");
            resp.setRenderParameter("mvcPath", "/edit.jsp");
            return false;
        }

        SessionMessages.add(req, "event-saved");
        return true;
    }
}
```

Tester dans Liferay

1. Déployer :

```
./gradlew :modules:event:deploy
```

2. Ajouter la portlet sur une page

3. Cliquer *Ajouter une entrée*
4. Tester :
    - Formulaire vide → message d’erreur
    - Formulaire rempli → succès

Atelier 10 : AUI et Navigation

Objectifs :

- Utiliser AlloyUI
- Navigation Simple

10.1 AlloyUI

Implémenter un formulaire AlloyUI

Dans `edit.jsp, remplacer le formulaire par` :

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

<portlet:actionURL name="saveEvent" var="saveURL" />

<aui:form action="${saveURL}" method="post" name="fm">
    <aui:input name="title" label="Titre" />

    <aui:button-row>
        <aui:button type="submit" value="Enregistrer" />
    </aui:button-row>
</aui:form>
```

Liste avec SearchContainer

Pour l'instant, les événements sont **mockés** (liste Java en mémoire).

Dans `view.jsp` :

```
<%@ page import="java.util.*" %>
<%@ taglib uri="http://liferay.com/tld/ui" prefix="liferay-ui" %>



<liferay-ui:search-container
    total="<%= events.size() %>"
    emptyResultsMessage="Aucun événement">

    <liferay-ui:search-container-results
        results="<%= events.subList(searchContainer.getStart(),
searchContainer.getEnd() > events.size() ? events.size() :
searchContainer.getEnd()) %>" />
```

```
    <liferay-ui:search-container-row
        className="java.lang.String"
        modelVar="event">

        <liferay-ui:search-container-column-text
            name="Nom"
            value="<%= event %>" />
    </liferay-ui:search-container-row>

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

10.2 Navigation simple

Créer une page home.jsp comme suit :

```
<%@ page import="java.util.*" %>
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8" %>

<%@ taglib uri="http://liferay.com/tld/ui" prefix="liferay-ui" %>
<%@ taglib uri="http://java.sun.com/portlet_2_0" prefix="portlet"
%>

<h1>Bienvenue sur la page d'accueil des événements</h1>
<portlet:renderURL var="listURL">
     <portlet:param name="mvcRenderCommandName"
value="/event/list" />
</portlet:renderURL>
<a href="<%= listURL %>">Afficher la liste des événements</a>
```

Faire en sorte que ce soit la page d'accueil du portlet en modifiant les paramètres de la classe portlet :

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

La page propose un lien utilisant ***mvcRenderCommandName***

Implémenter un MVCRenderCommand

Dans une classe EventListMVCRenderCommand

```
@Component(
        property = {

"javax.portlet.name=org_formation_event_EventPortlet",
                "mvc.command.name=/event/list"
        },
        service = MVCRenderCommand.class
```

```
)
    public class EventListMVCRenderCommand implements
MVCRenderCommand {
    @Override
    public String render(RenderRequest renderRequest,
RenderResponse renderResponse) throws PortletException {
        // Liste Mockée
        List<String> events = Arrays.asList(
                "Conférence Liferay",
                "Atelier Dev",
                "Meetup Architecture"
        );

        renderRequest.setAttribute("events", events);


        return "/view.jsp";
    }
}
```

Tester dans Liferay

1. Déployer :

```
./gradlew :modules:event:deploy
```

2. Vérifier le déploiement dans les traces
3. Visualiser l’affichage de la liste

Atelier 11 : Couche service via ServiceBuilder

Objectifs :

- Créer un modèle `Event` via Service Builder
- Implémenter `addEvent` et `getEvents`
- Exposer la liste via REST Builder
- Tester avec Postman / curl
- Appeler le service depuis une portlet ou une Remote App

11.1 Création du module ServiceBuilder

```
blade create -t service-builder -p org.formation.event events
```

2 modules générés :

- `events-api`
- `events-service` contient `service.xml`

Mettre à jour le fichier `service.xml` :

```
<service-builder dependency-injector="ds" package-path="org.formation.event">
 <namespace>EVENT</namespace>
 <!--<entity data-source="sampleDataSource" local-service="true" name="Foo"
remote-service="false" session-factory="sampleSessionFactory" table="foo" tx-
manager="sampleTransactionManager uuid="true"">-->
 <entity name="Event" local-service="true" remote-service="false">

 <!-- PK fields -->

 <column name="eventId" type="long" primary="true" />

 <!-- Group instance -->

 <column name="groupId" type="long" />

 <!-- Audit fields -->

 <column name="companyId" type="long" />
 <column name="userId" type="long" />
 <column name="userName" type="String" />
 <column name="createDate" type="Date" />
 <column name="modifiedDate" type="Date" />

 <!-- Other fields -->

 <column name="title" type="String" />
 <column name="dateDebut" type="Date" />
 <column name="dateFin" type="Date" />

 <!-- Order -->

 <order by="asc">
 <order-column name="title" />
```

```
            </order>

            <!-- Finder methods -->

            <finder name="DateDebut" return-type="Collection">
                  <finder-column name="groupId" />
                  <finder-column name="dateDebut" />
            </finder>

            <!-- References -->

            <reference entity="AssetEntry" package-
path="com.liferay.portlet.asset" />
            <reference entity="AssetTag" package-
path="com.liferay.portlet.asset" />
      </entity>
</service-builder>
```

Générer les classes

```
./gradlew buildService
```

Visualiser les classes générées

11.2 Implémentation de LocalServiceImpl

EventLocalServiceImpl.java :

```
public Event addEvent(long userId, long groupId, String title,
Date dateDebut, Date dateFin, ServiceContext sc) {

 long id = counterLocalService.increment(Event.class.getName());
 Event event = eventPersistence.create(id);

 event.setGroupId(groupId);
 event.setCompanyId(sc.getCompanyId());
 event.setUserId(userId);
 event.setTitle(title);
 event.setDateDebut(dateDebut);
 event.setDateFin(dateFin);

 return eventPersistence.update(event);
}


public List<Event> getEvents(long groupId, int start, int end) {
     return eventPersistence.findByGroupId(groupId,  start, end);
}

public int getEventsCount(long groupId) {
     return eventPersistence.countByGroupId(groupId);
}
```

Déployer le service

```
./gradlew deploy
```

Vérifier que le bundle est démarré

11.3 Intégration Portlet

Dans le projet du portlet, ajouter une dépendance

```
compileOnly project(":modules:events:events-api")
```

Modifier MVRenderCommand

```
@Reference
private org.formation.event.service.EventLocalService
_eventLocalService;

@Override
public String render(RenderRequest renderRequest, RenderResponse
renderResponse) throws PortletException {
    ThemeDisplay themeDisplay =
            (ThemeDisplay)
renderRequest.getAttribute(WebKeys.THEME_DISPLAY);

    List<org.formation.event.model.Event> events =
            _eventLocalService.getEvents(
                    themeDisplay.getScopeGroupId(), -1, -1);

    System.out.println("Rendering Event List with " +
events.size() + " events.");
    renderRequest.setAttribute("events", events);


    return "/view.jsp";
}
```

Modifier MVCActionCommand

```
@Reference
private org.formation.event.service.EventLocalService
_eventLocalService;

@Override
public boolean processAction(ActionRequest req, ActionResponse
resp) {

 String title = ParamUtil.getString(req, "title");
```

```
    System.out.println("Event Title: " + title);
    if (Validator.isNull(title)) {
        SessionErrors.add(req, "title-required");
        resp.setRenderParameter("mvcPath", "/edit.jsp");
        return false;
    }

    ThemeDisplay themeDisplay =
            (ThemeDisplay)
req.getAttribute(WebKeys.THEME_DISPLAY);

    ServiceContext serviceContext =
            null;
    try {
        serviceContext = ServiceContextFactory.getInstance(
                org.formation.event.model.Event.class.getName(),
                req);
    } catch (PortalException e) {
        throw new RuntimeException(e);
    }


    DateFormat dateFormat =
            DateFormatFactoryUtil.getSimpleDateFormat("yyyy-MM-
dd");

    Date dateDebut =
            ParamUtil.getDate(req, "dateDebut", dateFormat);

    Date dateFin =
            ParamUtil.getDate(req, "dateFin", dateFormat);

    _eventLocalService.addEvent(
            themeDisplay.getUserId(),
            themeDisplay.getScopeGroupId(),
            title,
            dateDebut,
            dateFin,
            serviceContext
    );

    SessionMessages.add(req, "event-saved");
    return true;
}
```

Déployer et vérifier le déploiement

Accéder au servlet, enregistrer un événement et vérifier la liste

Atelier 12 Service RESTFul via RestBuilder

Créer un module :

```
blade create -t rest-builder event-rest-builder
```

Visualiser les 2 modules générés dans l'IDE

Définir un contrat JSON

```
{
  "basePath": "/events", // Ceci définira le chemin final :
/o/events
  "applications": [
    {
      "path": "/events",
      "methods": [
        {
          "method": "GET",
          "uri": "/get-all", // L'URI de la méthode sera :
/o/events/events/get-all
          "description": "Récupère tous les événements pour un
groupe donné.",
          "parameterType": "LIST",
          "parameters": [
            {
              "name": "groupId",
              "type": "long",
              "in": "query"
            }
          ],
          "response": "EventDTO" // Renvoie une liste d'EventDTO
        }
      ]
    }
  ],
  "definitions": {
    "EventDTO": { // Définition du DTO pour le JSON
      "properties": {
        "id": {
          "type": "long"
        },
        "title": {
          "type": "string"
        },
        "dateDebut": {
```

```
          "type": "string",
          "format": "date-time"
        }
      }
    }
  }
}
```

Générer les classes

```
./gradlew buildRest
```

Visualiser les classes générées

Ajouter la dépendance vers le service

```
 compileOnly project(":modules:events:events-api")
```

Implémentation du REST

Modifier le contenu de la classe EventResourceImpl avec :

```
@Reference
private EventLocalService _eventLocalService;

@Override
public Page<Event> getEventsPage(Long groupId) throws Exception {

 // 2. Avoir le COUNT TOTAL (Obligatoire pour l'objet Page)
 int totalCount = _eventLocalService.getEventsCount(groupId);

 // 3. Avoir la LISTE des éléments paginés
 List<org.formation.event.model.Event> events =
 _eventLocalService.getEvents(groupId, 0, 20);

 // 4. Mapper la liste d'entités en liste de DTO
 List<Event> eventDTOs = events.stream()
 .map(this::toEventDTO)
 .collect(Collectors.toList());

 // 5. ENVELOPPER le tout dans l'objet Page
 return Page.of(eventDTOs);
}

private Event toEventDTO(org.formation.event.model.Event event) {
 return new Event() {{
 setId(event.getEventId());
 setTitle(event.getTitle());
 setDateDebut(event.getDateDebut());
 setDateFin(event.getDateFin());
 }};
```

```
}
```

Déployer

```
./gradlew deploy
```

Vérifier le déploiement

Tester

Récupérer le groupId dans la base de données

```
curl -v -H "Accept: application/json" --user "login:password"
"http://localhost:8080/o/events-headless/v1.0/events?
groupId=groupId"
```

Atelier 13 : Permissions & ServiceContext

Objectifs

1. Utiliser **ServiceContextFactory** dans une portlet MVC.
2. Passer un `ServiceContext` à un service de la couche métier.
3. Exploiter `ServiceContext` dans `LocalServiceImpl` (userId, groupId, dates).
4. Déclarer des **permissions de modèle** pour une entité donnée (resource-actions).
5. Vérifier les permissions dans la couche service et adapter l'IHM en conséquence.

13.1 ServiceContext dans l'ActionCommand

Dans le module `events-web` :

1. Ouvrir la classe `AddEventMVCActionCommand` (ou équivalent) qui traite la soumission du formulaire.
2. Récupérer un `ServiceContext` à partir de la requête :

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

3. Modifier l'appel au service `addEvent(...)` pour lui passer le `ServiceContext` en paramètre.

Par exemple, passer de :

```
_eventLocalService.addEvent(userId, groupId, name, date);
```

à :

```
_eventLocalService.addEvent(
    userId, groupId, name, date, serviceContext);
```

13.2 ServiceContext dans la couche service

Adapter la signature du service

Dans le module `events-service` :

1. Ouvrir l'interface `EventLocalService` générée (dans `events-api`).
2. Ajouter une signature :

```
public Event addEvent(
    long userId, long groupId, String name, Date date,
    ServiceContext serviceContext)
    throws PortalException;
```

3. Lancer `gradlew buildService` pour régénérer les implémentations.

Implémenter la logique dans `EventLocalServiceImpl`

Dans `EventLocalServiceImpl` :

1. Implémenter la méthode :

```
@Override
public Event addEvent(
        long userId, long groupId, String name, Date date,
        ServiceContext serviceContext)
    throws PortalException {

    long eventId = counterLocalService.increment(Event.class.getName());

    Event event = createEvent(eventId);

    event.setGroupId(serviceContext.getScopeGroupId());
    event.setCompanyId(serviceContext.getCompanyId());
    event.setUserId(serviceContext.getUserId());

    Date now = new Date();
    event.setCreateDate(serviceContext.getCreateDate(now));
    event.setModifiedDate(serviceContext.getModifiedDate(now));

    event.setName(name);
    event.setDate(date);

    // (Optionnel) logiques additionnelles, asset, etc.

    return updateEvent(event);
}
```

2. Redéployer le module `events-service`.

Contrôle

- Vérifier en base (ou via la console d'admin) que les événements créés portent bien :
    - le bon `groupId`
    - le bon `userId`
    - des dates cohérentes (`createDate`, `modifiedDate`).

13.3 Déclaration des permissions de modèle (resource-actions)

Créer le fichier `resource-actions/default.xml`

Dans le module `events-service` :

1. Créer le fichier `src/main/resources/resource-actions/default.xml` (s'il n'existe pas).
2. Ajouter une ressource de modèle pour `Event` :

```
<resource-action-mapping>
    <model-resource>
        <model-name>com.acme.events.model.Event</model-name>
```

```
        <permissions>
            <supports>
                <action-key>VIEW</action-key>
                <action-key>UPDATE</action-key>
                <action-key>DELETE</action-key>
            </supports>
            <guest-defaults>
                <action-key>VIEW</action-key>
            </guest-defaults>
        </permissions>
    </model-resource>
</resource-action-mapping>
```

3. Déclarer le fichier dans un `portlet.properties` ou `service.properties` (selon l'organisation) :

```
resource.actions.configs=resource-actions/default.xml
```

> Note formateur : dans un contexte moderne, on peut aussi mutualiser ce fichier au niveau du bundle service. L'idée clé est que les actions VIEW / UPDATE / DELETE existent et soient connues de Liferay.

Redéploiement et vérification

- Redéployer `events-service`.
- Dans le Control Panel, vérifier que les rôles peuvent maintenant se voir attribuer ces actions spécifiques pour le modèle `Event`.

13.4 Vérification des permissions (service + IHM)

Injection de `ModelResourcePermission<Event>`

Toujours dans `EventLocalServiceImpl` :

1. Ajouter un champ :

```
@Reference(
    target = "(model.class.name=com.acme.events.model.Event)"
)
private ModelResourcePermission<Event> _eventModelResourcePermission;
```

2. Dans une méthode de lecture ou de modification, insérer un contrôle, par exemple dans `updateEvent(...)` ou dans `getEvent(...)` :

```
public Event getEvent(long eventId) throws PortalException {

    Event event = eventPersistence.findByPrimaryKey(eventId);

    _eventModelResourcePermission.check(
        getPermissionChecker(), eventId, ActionKeys.VIEW);

    return event;
}
```

> Note formateur : ici, montrer que `getPermissionChecker()` est accessible via la

classe de base de la ressource REST ou via `PermissionThreadLocal` dans certains cas.

Adapter la portlet : masquer les actions non autorisées

Dans `view.jsp` de la portlet `events-web` :

1. Pour chaque ligne `event` d'un tableau, entourer les actions "Modifier/Supprimer" d'un test de permission simple.

Option basique : vérifier côté JSP au moyen de la balise `<liferay-security:permissionsURL>` + icône, ou d'un test sur `permissionChecker.hasPermission(...)`.
Exemple minimal :

```
<%
boolean canEdit = permissionChecker.hasPermission(
    themeDisplay.getScopeGroupId(),
    Event.class.getName(),
    event.getEventId(),
    ActionKeys.UPDATE);
%>

<c:if test="<%= canEdit %>">
   <!-- Bouton / lien d'édition -->
   <a href="<%= editURL %>">Modifier</a>
</c:if>
```

2. Vérifier que, connecté avec un utilisateur n'ayant pas l'action UPDATE sur ce type d'Event, le bouton n'apparaît plus.

Contrôle fonctionnel final

- **Scénario 1** : utilisateur "admin" ou rôle avec toutes les permissions :
    - Il peut créer, voir, modifier et supprimer des événements.
    - Les liens d'action (édition, suppression) sont visibles.
- **Scénario 2** : utilisateur "lambda" sans permission UPDATE :
    - Il voit la liste des événements (grâce à VIEW).
    - Il ne voit pas les liens/boutons de modification/suppression.
    - S'il force un appel à un service de modification (URL manuelle), la couche service doit lever une exception de permission (test `_eventModelResourcePermission.check(...)`).

Atelier 14 : Asset, Thème, OSGI

14.1 Asset Framework

Objectifs

- Associer des catégories et tags à une entité personnalisée (`Event`)
- Utiliser `ServiceContext` pour alimenter Asset Framework
- Afficher les tags/catégories dans l'interface
- Tester l'intégration avec Asset Publisher

Ajouter les sélecteurs de tags / catégories dans la JSP

Dans `edit.jsp` ou `form.jsp` :

```
<liferay-ui:asset-tags-selector
    className="com.acme.events.model.Event"
    classPK="${event.eventId}"
    hiddenInput="tagNames" />

<liferay-ui:asset-categories-selector
    className="com.acme.events.model.Event"
    classPK="${event.eventId}"
    hiddenInput="categoryIds" />
```

Récupérer tags & catégories dans l'ActionCommand

Dans l'ActionCommand :

```
String[] tagNames = ParamUtil.getStringValues(actionRequest, "tagNames");
long[] categoryIds = ParamUtil.getLongValues(actionRequest, "categoryIds");

serviceContext.setAssetTagNames(tagNames);
serviceContext.setAssetCategoryIds(categoryIds);
```

Appeler AssetEntryLocalService dans la couche service

```
@Reference
private AssetEntryLocalService assetEntryLocalService;

assetEntryLocalService.updateEntry(
    userId,
    serviceContext.getScopeGroupId(),
    Event.class.getName(),
    event.getEventId(),
    serviceContext.getAssetCategoryIds(),
    serviceContext.getAssetTagNames()
);
```

Vérifier l'affichage dans la vue

Dans `view.jsp` :

```
<p>Tags : <liferay-ui:asset-tags-summary className="com.acme.events.model.Event"
classPK="${event.eventId}" /></p>
<p>Catégories : <liferay-ui:asset-categories-summary
className="com.acme.events.model.Event" classPK="${event.eventId}" /></p>
```

Validation

- Créer un Event avec tags/catégories → vérifier dans Asset Publisher.
- Modifier un Event → tags mis à jour.

14.2 Thème

Objectifs

- Créer un thème minimaliste
- Modifier la structure via Freemarker
- Ajouter du style personnalisé via SCSS
- Tester le thème dans le portail

Générer un thème

Dans le Liferay Workspace :

```
blade create -t theme custom-theme
```

Importer dans l'IDE.

Modifier le SCSS

Dans `src/css/_custom.scss` :

```
body {
   background-color: #f5f7fa;
}

#header {
   background-color: #004b8d;
   color: white;
}
```

Modifier le template principal

Dans `src/templates/portal_normal.ftl` :

Ajouter après `<body>` :

```
<div class="alert alert-info text-center">
    Thème personnalisé chargé avec succès
</div>
```

Déployer

```
gradlew deploy
```

Appliquer le thème à un site.

Validation

- La bannière bleue apparaît dans toutes les pages du site.
- Le style personnalisé est appliqué.

Atelier 15 : Personnalisation OSGI

15.1 Personnalisation via Fragment Module (Override JSP)

Objectifs

- Apprendre la méthode moderne pour remplacer une JSP native
- Comprendre le fonctionnement des OSGi Fragment Modules
- Modifier l'IHM de Liferay sans EXT ni Hooks

Créer un fragment

```
blade create -t fragment -h com.liferay.login.web login-fragment
```

Cela crée un module dépendant du bundle com.liferay.login.web.

Identifier la JSP à remplacer

Exemple : remplacer le texte dans `login.jsp` (emplacement réel dépend de la version).

Accepter l'arborescence proposée :

```
src/main/resources/
   META-INF/resources/
      login.jsp
```

Modifier la JSP

Ajouter un message :

```
<div class="alert alert-warning">
   Message personnalisé : connexion au portail Liferay
</div>
```

Déployer

```
gradlew deploy
```

Aller sur `/c/portal/login` pour valider.

Validation

- Le message personnalisé apparaît au-dessus du formulaire de connexion.

15.2 Personnalisation via Service Wrapper OSGi

Objectifs

- Surcharger un service existant de Liferay
- Intercepter une opération métier
- Ajouter des règles ou logs personnalisés

Créer un module wrapper

```
blade create -t service-wrapper user-wrapper
```

Modifier la classe générée

Dans `UserLocalServiceWrapper` :

```
@Component(
```

```
  property = "model.class.name=com.liferay.portal.kernel.model.User",
  service = AopService.class
)
public class CustomUserLocalServiceWrapper
    extends UserLocalServiceWrapper {

    @Override
    public User addUser(...) throws PortalException {

        System.out.println(">>> Un utilisateur est en cours de création");

        User user = super.addUser(...);

        System.out.println(">>> Utilisateur créé : " + user.getUserId());

        return user;
    }
}
```

Déployer

Créer un utilisateur → observer les logs serveur.

Validation

- Le log avant/après création apparaît correctement.
- Aucune régression fonctionnelle.

15.3 : Model Listener OSGi

Objectifs

- Réagir aux événements CRUD sur une entité
- Implémenter la logique d'audit ou de synchronisation
- Découvrir le hook moderne remplaçant les "Value Object Listeners"

Créer un module listener

```
blade create -t api -p com.acme.events.listener events-listener
```

Ajouter un ModelListener

```
@Component(
    service = ModelListener.class
)
public class EventModelListener extends BaseModelListener<Event> {

    @Override
    public void onAfterCreate(Event event) {
        System.out.println("Événement créé : " + event.getName());
    }

    @Override
    public void onAfterRemove(Event event) {
        System.out.println("Événement supprimé : " + event.getName());
    }
}
```

Déployer et tester

Créer et supprimer un Event → vérifier les logs.

Validation

- Les messages apparaissent lors de la création et suppression.