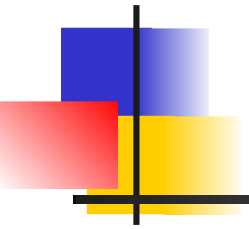
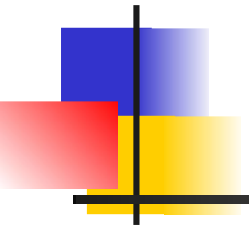

PLB
FORMATION INFORMATIQUE ET MANAGEMENT



Liferay 7.x - Développer un portail d'entreprise

David THIBAU – 2025

david.thibau@gmail.com



Agenda

- **Fondamentaux**
  - Panorama Liferay DXP / Portal 7.4
  - Installation
  - Sites, Pages, Navigation
  - CMS
- **Administration, Exploitation**
  - Base de données
  - Configuration, logs, sessions
  - Exploitation modules OSGI
  - Patching et mise à jour
  - Environnements, Staging, Import/Export
- **Développement**
  - Mise en place IDE
  - Développement Portlet
  - Service Builder et Rest Builder
  - Sécurité
  - Extensions

Introduction

Panorama Liferay DXP / Portal 7.4

Installation

Site, pages, navigation

CMS : web content, templates, documents

Portail et Digital Experience Platform

- Un **portail d’entreprise** est une plate-forme web qui centralise l’accès aux applications, contenus et services numériques de l’organisation.
- Une **« Digital Experience Platform (DXP) »** ajoute à ces fonctions des capacités avancées : personnalisation, APIs, intégration *omnicanal* (web, mobile, applications tierces).
- L’objectif est de fournir à chaque utilisateur (client, partenaire, collaborateur) une expérience unifiée malgré la diversité des systèmes sous-jacents.

Fonctions clés d’un portail d’entreprise

- **Agrégation de contenus et d’applications** : tableau de bord, pages composites, vue synthétique d’indicateurs issus de plusieurs systèmes.

- **Administration centralisée** : gestion des utilisateurs, des rôles, des droits d’accès et des espaces (sites, communautés, organisations).

- **Cadre d’architecture** : normes, modèles de développement et de sécurité homogènes pour les applications intégrées au portail.

Personnalisation, identité et SSO

- **Personnalisation par profil** : chaque utilisateur appartient à un ou plusieurs rôles, qui conditionnent les contenus visibles et les fonctionnalités disponibles.
- **Personnalisation par préférences** : choix des applications affichées, agencement des portlets, sélection d'un thème graphique.
- **Identité et SSO** :
  - L'authentification s'appuie sur un référentiel central (annuaire LDAP ou IdP type Azure AD / Keycloak).
  - Le portail met en œuvre des standards de Single Sign-On : SAML2, OpenID Connect, OAuth2, permettant à l'utilisateur de s'authentifier une fois et d'accéder à plusieurs applications.

Modes d’intégration d’applications et de contenus

- **Applications intégrées** dans le portail (modules / portlets / remote apps) développées selon le modèle technique de la plate-forme.
- **Intégration légère :**
  - liens vers des applications externes,
  - iframes ou widgets intégrés dans les pages,
  - flux RSS/Atom pour remonter des actualités ou contenus éditoriaux.
- **Intégration par APIs :**
  - consommation ou exposition de services REST/JSON pour échanger des données avec des applications métier, ERP, CRM, etc.
  - sécurisation via OAuth2 lorsque les APIs sont exposées à des applications externes.

Positionnement de Liferay

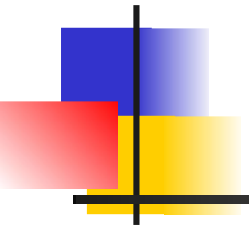
- **Liferay DXP (édition entreprise)** : plate-forme commerciale avec support, mises à jour contrôlées et fonctionnalités avancées (notamment Commerce et certains connecteurs).
- **Liferay Portal CE (Community Edition)** : édition communautaire libre (LGPL), adaptée aux projets moins critiques ou aux phases d'étude.
- Dans les deux cas, Liferay fournit :
  - un noyau portail,
  - des fonctionnalités CMS,
  - des fonctions de collaboration et de gestion d'accès,
  - des APIs headless pour l'intégration avec des applications externes.

Liferay comme CMS et plate-forme collaborative

- **CMS intégré :**
  - gestion des pages, des gabarits, des contenus web,
  - workflow éditorial, staging, planification de publication,
  - taxonomies, tags, segmentation.
- **Collaboration :**
  - blogs, wikis, forums, agendas, bibliothèque documentaire,
  - communautés internes ou externes,
  - fonctionnalités sociales (suivi, notifications, commentaires).

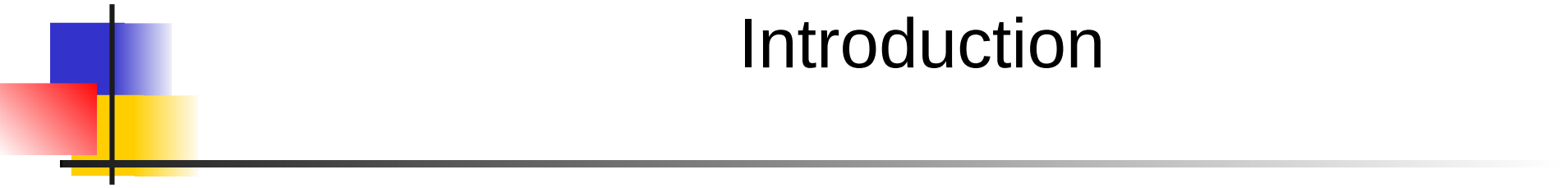
Liferay comme plate-forme d’intégration

- Liferay joue le rôle de couche d’exposition des données et services :
  - **APIs REST headless** pour la plupart des fonctionnalités standard.
  - **Connecteurs** et intégrations avec des systèmes tiers (ERP, CRM, GED, moteurs de recherche).
- La recherche full-text s’appuie sur Elasticsearch / OpenSearch avec indexation des contenus, documents et données applicatives.



Liferay pour les développeurs

- Modèle de développement basé sur des **modules OSGi** :
  - modules MVC portlet,
  - modules Service Builder pour la couche de services,
  - modules REST Builder pour exposer des APIs headless.
- Couche présentation :
  - taglibs Liferay + AlloyUI + Clay pour l'interface,
  - Remote Apps (React/Angular…) pour des applications full-JS.
- Extension et personnalisation :
  - Configuration par modules, thèmes, fragment modules,
  - Intégration SSO (SAML, OIDC) et sécurisation OAuth2 pour les APIs.



Introduction

Panorama Liferay DXP / Portal 7.4

Installation

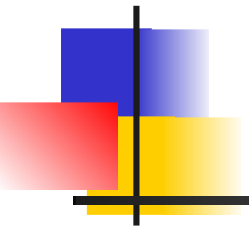
Site, pages, navigation

CMS : web content, templates, documents

Pré-requis pour l’installation

- **Système d’exploitation : Linux, Windows, macOS**
  - Minimum recommandé : 2 CPU, 4 Go RAM (8 Go recommandé pour production), 10 Go disque libre
- **Java**
  - JDK17 ou 21
  - Variable d’environnement : JAVA_HOME correctement configurée
- **Base de données**
  - MySQL / MariaDB
  - PostgreSQL
  - Oracle
  - SQL Server
  - Embedded HSQLDB disponible uniquement pour tests / dev
- **Préparer la base :**
  - Créer un utilisateur dédié
  - Accorder droits CREATE / UPDATE / DELETE

Télécharger et installer un bundle Liferay

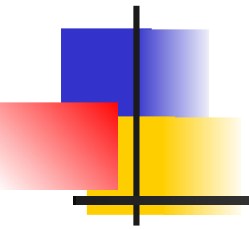


- Télécharger le bundle Liferay 7.4 Tomcat (DXP ou Portal CE) depuis le site officiel.
- Décompresser l’archive dans un répertoire dédié, qui devient le Liferay Home.
- Vérifier la variable d’environnement JAVA_HOME et le chemin du JDK.

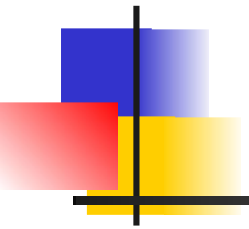
Structure de Liferay Home

- Le dossier racine contient notamment :
  - ***data*** : données persistées par Liferay (documents, configuration, éventuelle base de démonstration) ;
  - ***deploy*** : répertoire pour le déploiement automatique de modules/LPKG ;
  - ***logs*** : journaux applicatifs ;
  - ***osgi*** : modules standard et extensions déployées.
- Cette structure est commune quel que soit le serveur applicatif sous-jacent.

Démarrage et assistant de configuration



- Lancer le serveur :
  - sous Linux : ***./tomcat-*/bin/startup.sh,***
  - sous Windows : ***tomcat-*\bin\startup.bat.***
- Accès au portail via http://localhost:8080.
- À la première connexion :
  - choix de la langue et du nom du portail ;
  - création du compte administrateur ;
  - configuration de la base de données (base embarquée pour les TPs ou base dédiée de développement).



Alternative Docker

- Liferay fournit des images Docker facilitant l’installation en environnement isolé.
- Intérêt :
  - Reproductibilité de l’environnement ;
  - Gestion simplifiée pour les ateliers lorsqu’un moteur Docker est disponible.

Atelier 1 : Installation

Introduction

Panorama Liferay DXP / Portal 7.4

Installation

Site, pages, navigation

CMS : web content, templates, documents

Company (Instance Liferay)

- Une **Company** représente une instance Liferay (tenant) :
  - point d'entrée du portail
  - périmètre de sécurité et de configuration
  - racine de tous les sites, utilisateurs et rôles

- Un serveur Liferay peut héberger plusieurs *Company*, mais dans la majorité des projets :
  1 serveur = 1 Company

Rôle de la Company

- Définit :
  - le site par défaut
  - le nom du portail
  - les paramètres globaux (authentification, sécurité, timeout…)
- Contient :
  - les utilisateurs
  - les sites
  - les rôles globaux
- Table : ***company***

Utilisateurs dans Liferay

Un utilisateur représente une personne accédant au portail.

- appartient à une Company
- peut être membre de plusieurs sites
- se voit attribuer un ou plusieurs rôles

- Utilisateur **anonyme**
  - non authentifié
  - accès aux pages publiques uniquement
- Utilisateur **authentifié**
  - accès selon ses rôles
- **Administrateur**
  - gestion des sites, utilisateurs et configurations

Regroupement des utilisateurs

Les utilisateurs peuvent optionnellement être associés à :

- Des **organisations** : structure hiérarchique métier
  Par exemple : Direction, Département, Filiale, Equipe
  Peut contenir des utilisateurs ou des sous-organisation
  Modélise l'entreprise

- Des **groupes** : Un groupe transversal non hiérarchique
  Par exemple : Rédacteur, Chef de projet, etc..
  Sert à affecter des rôles en masse
  Simplifie la gestion des utilisateurs

Rôle et permissions

- Un utilisateur n’a pas de droits en direct
- Les droits sont toujours portés par des **rôles**
- Un même utilisateur peut jouer plusieurs rôles
- Un rôle est un ensemble de **permissions**.
- Les permissions définissent :
  - Ce que l’on peut faire (voir, créer, modifier…)
  - Sur quoi (site, page, contenu, application)

Principaux types de rôles

L’interface Liferay distingue plusieurs types de rôles, chacun avec une portée précise.

- Rôles **réguliers** (Regular Roles)
  - Portée : Globale à la Company
  - Usage : Accès à l’administration, Accès transversal
- Rôles de **site** (Site Roles)
  - Portée : Limitée à un site donné
  - Usage : Droits sur les pages, contenus, staging
  - Exemples : Administrateur de site, Éditeur, Contributeur, Membre du site
- Rôles **d’organisation** (Organization Roles)
  - Portée : Limitée à une organisation
  - Usage : Gestion des utilisateurs d’une entité métier, Délégation administrative
  - Exemple : Administrateur d’organisation

Les sites

Un **site** Liferay est un espace fonctionnel autonome qui regroupe des pages, des contenus et des utilisateurs, avec ses propres règles d'accès et de publication.

- Structure le portail en domaines fonctionnels
- Sépare espaces publics et privés
- Gère le multilingue
- Définit des équipes locales, incluant des attributions (rôles)
- Accueille des contenus spécifiques

Types de Sites

- **Sites Publics**
  - Visibles par tous
  - Pages publiques / navigation publique
- **Sites Privés**
  - Accessibles seulement à un sous-ensemble d’utilisateurs
- **Site Global (Global Group)**
  - Contenus transverses, tags globaux, documents partagés
- **Sites Organisationnels**
  - Liés aux structures hiérarchiques de l’entreprise

Tables BD d’un Site

- **Table Group_** :
  Représente un **site, un espace, une communauté ou une organisation**. La colonne classNameId + le champ type permettent d’identifier qu’il s’agit d’un Site.

- **Table Layout** : Chaque entrée correspond à une **page (publique ou privée)**. Contient les informations structurelles : URL, template, widgets placés, hiérarchie…

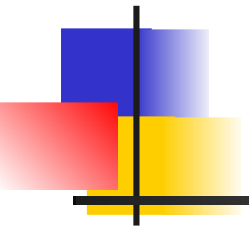
- **Table LayoutSet** : Définit **un ensemble de pages publiques ou privées pour un site**. Stocke aussi le thème, le look & feel, les paramètres globaux du page set.

- **Table JournalArticle** : Stocke les **Web Contents (articles CMS)**. Contient le XML/JSON du contenu + métadonnées.

- **Table DLFileEntry** : Chaque ligne = un document avec ses métadonnées, versions liées, fichiers binaires en stockage.

Comprendre ces tables aide à diagnostiquer les erreurs (pages corrompues, URLs cassées, droits incohérents

Les Master Pages



- Comparable à un “**gabarit**” :
  - En-tête commun
  - Pied de page
  - Éléments répétés
  - Structure globale
- Utilisée pour :
  - Harmoniser plusieurs pages
  - Déléguer aux équipes métiers la création des pages sans toucher au template
- On démarre avec une Master Page : Page vierge
- Lors de la création de la page on choisit son gabarit
- Création via le menu

Conception – Modèles de pages

Pages de contenu dans Liferay

Widget Pages

- Pages historiques
- Contiennent des portlets
- Mise en page via colonnes / layout
- Extrêmement flexibles pour du développement custom

= > Idéal pour les pages dynamiques

Content Pages

- Pages créées avec l’éditeur visuel (React)
- Basées sur les fragments
- Responsive natif
- Pas de portlets par défaut (mais portlets utilisables via fragments portlets)

= > Idéal pour les pages de contenu

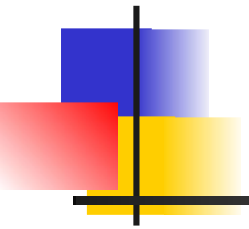
Widget Pages

- Les **Widget Pages** sont le modèle historique de Liferay :
  - Basées sur des portlets (applications)
  - Mise en page via layouts en colonnes (1, 2, 3 colonnes, etc.)
  - Chaque zone de la page accueille une ou plusieurs portlets
  - Adaptées aux applications métier et écrans riches (portlets custom, listes, formulaires)
  - Compatibles avec :
    - permissions fines par portlet
    - configuration par instance (préférences portlet)
    - personnalisation utilisateur (ajout/retrait de portlets si autorisé)

Content Pages : avantages

- Éditeur visuel riche : drag & drop
- Aucune compétence technique nécessaire
- Reposent sur des fragments réutilisables
- Personnalisation par audience (segments)
- Intégration des contenus web, media, formulaires Liferay

Recommandé pour les sites CMS modernes.



Structure des pages dans Liferay

Une page est définie par :

- une Friendly URL (ex : /actualites)
- un layout (1 colonne, 2 colonnes, vide...)
- une template de base (master page)
- des permissions propres

Friendly URLs

Chaque page possède une URL lisible. Exemples :

/produits
/contact
/a-propos

Caractéristiques :

- Automatiques mais modifiables
- Déclinables par langue
- Influencent le SEO
- Uniques dans leur ensemble (public ou privé)

Display Pages (Pages de présentation)

- Les **display pages** servent à afficher un contenu individuel (ex : un article de blog, un événement, une actualité).
  - Elles sont reliées à une structure de contenu
  - Sélectionnées automatiquement par Asset Publisher ou widgets équivalents
  - Permettent un site complètement headless-friendly

Conception – Modèles de page – Modèles de page d'affichage

Redirections et gestion avancée

Possibilités :

- Redirection vers une URL externe
- Redirection vers une autre page du portail
- Redirection conditionnelle (Audience Targeting)
- Masquer la page dans le menu tout en la laissant accessible via son URL

Nouvelle Page – URL ou Lien vers une page interne

Navigation dans un Site

- La navigation repose sur :
  - La hiérarchie des pages
  - La visibilité des pages (public/privé, permissions)
  - Les menus de navigation configurables
  - Les “friendly URLs”

Gestion de la hiérarchie

- Dans l'UI : ***Créateur de site – Page de site***
  - Glisser-déposer une page pour changer sa position
  - Créer des sous-pages pour organiser l'information
  - Utiliser des pages "conteneurs" non affichées pour structurer

Bonne pratique : Ne jamais créer des dizaines de pages à la racine : segmenter !

Permissions sur les pages

- Chaque page peut définir :
  - Qui peut voir la page
  - Qui peut modifier la page
  - Qui peut ajouter des sous-pages
  - Qui peut gérer les applications / fragments
- Les **permissions** sont gérées :
  - au niveau site
  - au niveau page individuelle
  - via rôles généraux ou rôles de site

Menus de navigation

Deux possibilités :

1. Navigation automatique

- Basée sur l’arborescence des pages
- Recommandée pour sites simples

2. Navigation manuelle ***Créateur de site – Menu de navigation***

- Menus personnalisés
- Possibilité de mélanger pages, URLs externes, catégories CMS, etc.

Les menus sont affichés via un fragment de navigation ou un widget Navigation Menu souvent placé dans une Master Page (header)

Organisation recommandée d’un Site

Pour un projet d’entreprise :

- 1 site principal
- 1 site “documentation / DM” si volumétrique
- Pages limitées et hiérarchisées
- Master pages pour tout le site
- Display pages pour tous les contenus importants
- Portlets seulement si besoin métier

Éviter les pièges :

- Trop de pages à la racine
- Navigation manuelle pour tout → maintenance lourde
- Master pages complexes
- Mélanger Content Pages et Widget Pages sans logique

Atelier 2 : Site, Page, Navigation

Introduction

Panorama Liferay DXP / Portal 7.4

Installation

Site, pages, navigation

CMS : web content, templates, documents

Le CMS de Liferay : un socle central

Le CMS intégré permet :

- La gestion de contenus multilingues
  Site Settings → Languages
- La création de contenus structurés via *Structures & Templates*
  Contenu → Contenu Web → Structure
- La gestion documentaire (DM)
  Contenu → Document et média
- L'organisation de l'information grâce à la taxonomie (tags / catégories)
  Catégorisation
- La publication dynamique (Asset Publisher, Fragments, Display Pages)
- L'intégration workflow

Liferay est à la fois un CMS et un framework applicatif.

Architecture CMS

Un contenu publié dans Liferay peut être :

- Un contenu Web (article HTML, texte, actualité…)
- Un contenu structuré : Structure + Template
- Un Document & Media File (image, PDF, vidéo)
- Un Asset personnalisé (ex : Event, produit…)

Tous ces éléments sont des **Assets**, réutilisables dans :

- Fragments
- Widgets
- Display Pages
- Search
- Asset Publisher

Web Content : définition

Un **contenu web** est un contenu textuel ou HTML pouvant contenir :

- du texte
- des images
- des vidéos
- des variables dynamiques (si structure)
- de la mise en forme simple ou avancée (éditeur wysiwyg)

Il constitue l'unité de contenu principale dans un site Liferay.

Web Content : types et usages

Utilisations classiques

- Actualités / communiqués
- Pages institutionnelles
- Blocs éditoriaux
- Contenu multilingue
- Bannière / snippet

Stockage basé sur :

- Table *JournalArticle*
- Versioning intégré
- Workflow optionnel

Création d’un Web Content

Étapes :

- Aller dans ***Site > Contenu > Web Content***
- Cliquer ***Add > Basic Web Content***
- Renseigner :
  - Titre
  - Langues
  - Contenu (éditeur HTML)
- Publier ou démarrer un workflow

Le contenu peut ensuite être affiché via un *fragment* ou un *widget*.

Intégration d’un contenu dans une page

Deux possibilités :

Widget “Web Content Display”

- Sélection d’un contenu unique
- Simple et direct

Fragments

- Utiliser un fragment “contenu web” dans une page de contenu
- Choisir le contenu directement dans l’éditeur visuel
- Mise en forme plus flexible
- Approche moderne et responsive

Structures

Avantages :

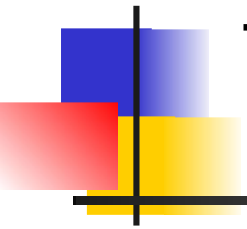
- Séparer contenu et mise en forme
- Éviter le HTML dans les contenus
- Garantir l’homogénéité : toutes les actualités ont le même modèle
- Publication automatisée via Asset Publisher ou Display Page
- Réutilisation multi-sites
- Multilingue propre

Structure : le modèle de données

Une structure définit les champs d’un contenu :

- Texte
- Image
- Date
- Sélecteur
- Géolocalisation
- Liste dynamique
- Rich Text
- Champs répétés (repeatable fields)

Les structures sont stockées dans la table *DDMStructure*.



Template : le rendu d’un contenu (FreeMarker)

Un template définit comment afficher le contenu dans un widget:

- HTML + FTL
- Accès aux champs via ${fieldName.getData()}
- Mise en forme personnalisée
- Intégration CSS
- Conditionnels, boucles, formats

Les templates sont stockés dans la table *DDMTemplate*.

Un template FreeMarker ne crée pas de page :
il définit uniquement le rendu d’un contenu à l’intérieur d’un widget

Exemple structure + template

Structure “Actualité”

- titre (text)
- image (image)
- résumé (text)
- corps (rich text)
- datePublication (date)

Template free marker:

```
<h1>${titre.getData()}</h1>
<img src="${image.getData()}" />
<p class="summary">${resume.getData()}</p>
<div class="content">${corps.getData()}</div>
<small>Publié le ${datePublication.getData()}</small>
```

Publication automatisée via Display Page

Une structure peut être liée à un **modèle de page d'affichage (Display Page)** :

- L'utilisateur crée un contenu → il est automatiquement associé à une page
- URL dédiée par contenu
- Idéal pour blogs, actualités, produits, FAQ
- Fragments de page = design flexible

La Display Page est une vraie page, construite avec des fragments, qui remplace avantageusement les templates FreeMarker pour l'affichage détaillé d'une structure.

Structure, Template ou Display Page : quand utiliser quoi ?

- Définir les champs d’un contenu :
  Structure
- Rendu dans un widget (liste, bloc) :
  Template FreeMarker
- Page dédiée avec URL, SEO, navigation :
  Display Page
- Site moderne / CMS :
  Structure + Display Page

Documents & Media : rôle

Module de gestion documentaire :

- Stockage d’images, PDF, vidéos, archives…
- Arborescence (dossiers, sous-dossiers)
- Prévisualisation automatique
- Versioning natif
- Métadonnées personnalisées
- Permissions granulaire
- Publication via fragments, widgets ou URLs publiques

Méta-données et Taxonomie

Chaque document peut avoir :

- Tags
- Catégories
- Métadonnées spécifiques (ex : type de facture, client, date)
- Informations techniques (MIME, taille, checksum)

Récupération d’un document dans une page

Méthodes :

- ***Fragment “Image” → choisir un document***
  Liferay gère le responsive
- **Fragment “Content”**
  Utiliser des documents comme assets
- **Widget Document & Media**
  affichage de dossiers complets
- URL publique (si activée)

Bonnes pratiques Documents & Media

- Organiser en dossiers logiques (éviter un dossier “images” avec 1000 éléments)
- Utiliser catégories + métadonnées
- Activer les permissions selon usage (édition restreinte)
- Préférer WebP/JPEG optimisés
- Séparer contenus éditoriaux / fichiers techniques

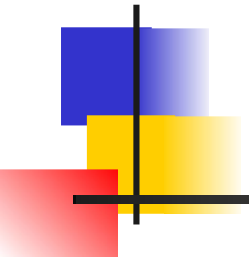
Tags vs Catégories

Tags

- Libres
- Créés par les utilisateurs
- Permettent un regroupement ponctuel
- Utiles pour le filtrage dans Asset Publisher ou Search

Catégories

- Organisées dans des vocabulaires
- Contrôlées (taxonomie métier)
- Permettent :
  - Segmentation
  - récupération personnalisée
  - création d’automatisations (display page conditionnelle)



Administration & Exploitation

Base de données

Optimisation configuration / logs / sessions

Exploitation des modules OSGI

Patching DXP / mises à jour

Environnements, staging, import/export

Organisation générale de la base

La base Liferay est composée de quatre grands ensembles :

I) Données cœur du portail : users, roles, groups, permissions, layouts

II) Modules standards (CMS, Documents & Media, Blogs, etc.)

III) Configuration et métadonnées système

IV) Données applicatives personnalisées (Service Builder)

Liferay applique systématiquement :

- une gestion stricte des clés (companyId, groupId, userId)
- un modèle multi-tenant intégré

Les trois identifiants fondamentaux

Ces 3 colonnes sont présentes dans presque toutes les tables Liferay :

- ***companyId*** → identifie l'instance Liferay
- ***groupId*** → identifie un site, une communauté, un scope
- ***userId*** → identifie l'utilisateur ayant créé / modifié la ressource

Compréhension obligatoire pour diagnostiquer :

- contenu invisible (mauvais groupId)
- erreurs de permission
- duplications ou conflits de Site

Company, Groupes, Sites

Table Company

- Représente l’instance Liferay (tenant)
- Contient le nom du portail (ex. Portail des événements)
- Porte le branding global (nom, domaine, configuration générale)

Table Group_

- Représente un site Liferay (Guest, Global, site métier, organisation…)
- 1 enregistrement = 1 site logique
- Lié à une Company via companyId

Table LayoutSet

- Représente un ensemble de pages pour un site
- Toujours 2 enregistrements par site :
  - pages publiques (privateLayout = false)
  - pages privées (privateLayout = true, héritage historique)
- Porte les paramètres globaux : thème, logo, CSS, paramètres communs aux pages

Pages (Layouts)

Table : Layout

Colonnes clés :

- *plid* → ID de la page
- *groupId* → site auquel la page appartient
- *friendlyURL*
- *parentLayoutId* → hiérarchie
- *type* → widget / content
- *hidden* → page non affichée dans navigation

Diagnostics utiles :

- page non visible → vérifier *hidden* ou *typeSettings*
- URL cassée → vérifier *friendlyURL*

Utilisateurs & Rôles

Users : ***User_***

Organisations : ***Organization_***

Rôles : ***Role_***

Permissions : ***ResourcePermission***

Membres de site : ***UserGroupRole*** & ***Group_***

Liferay mélange rôles globaux, rôles de site, rôles d’organisation

Les permissions finales sont matérialisées dans *ResourcePermission*

Permissions : tables critiques

Permission = combinaison de :

- *RoleId*
- *ResourceId*
- *ActionId*

Tables clés :

- ***ResourceAction*** : Liste des actions possibles (VIEW, UPDATE…)
- ***ResourcePermission*** : Autorisation effective
- ***ResourceTypePermission*** : Permissions par type

Diagnostic classique :

- Contenu invisible → vérifier *ResourcePermission* + rôle utilisateur.

CMS : Web Content (Journal)

JournalArticle : Contenu (toutes versions)

JournalArticleLocalization : Multilingue

JournalFolder : Arborescence

DDMStructure : Structure

DDMTemplate : Template Freemarker

Points clés :

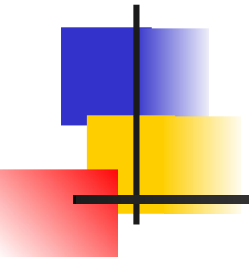
- Un Web Content possède plusieurs versions (version)
- Le champ *resourcePrimKey* relie toutes les versions entre elles
- Workflow géré via *status* + *statusByUserId*

Documents & Media

- ***DLFileEntry*** : Métadonnées document
- ***DLFileVersion*** : Versioning
- ***DLFolder*** : Dossiers
- ***DLFileEntryMetadata*** : Métadonnées personnalisées
- ***Repository*** : Dépôts externes (CMIS, S3…)

Diagnostics :

- Document invisible → mauvais groupId
- Mauvaise version → vérifier DLFileVersion.isLatest



Administration & Exploitation

Base de données

Optimisation configuration / logs / sessions

Exploitation des modules OSGI

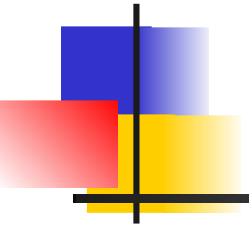
Patching DXP / mises à jour

Environnements, staging, import/export

Deux niveaux de configuration

1. Configuration **globale**
   - Options de la JVM
   - Fichiers *portal-ext.properties*
   - Paramètres haute disponibilité / cluster
   - LDAP / SSO / Proxy

2. Configuration **modulaire** (OSGi)
   - Fichiers .*config dans osgi/configs*
   - GogoShell
   - Activation/désactivation des modules
   - Configuration à chaud



Options JVM

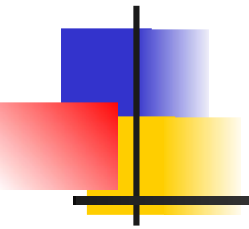
Les options de la JVM se positionne dans Tomcat.
bin/setenv.sh

Extrait :

```
CATALINA_OPTS="$CATALINA_OPTS -Dfile.encoding=UTF-8 -
Djava.net.preferIPv4Stack=true -Duser.timezone=GMT -Xms2560m -Xmx2560m -
XX:MaxNewSize=1536m -XX:MaxMetaspaceSize=768m -XX:MetaspaceSize=768m -
XX:NewSize=1536m -XX:SurvivorRatio=7"

export JDK_JAVA_OPTIONS="${JDK_JAVA_OPTIONS}
--add-opens=java.base/java.lang=ALL-UNNAMED
--add-opens=java.base/java.lang.invoke=ALL-UNNAMED
--add-opens=java.base/java.lang.reflect=ALL-UNNAMED
--add-opens=java.base/java.net=ALL-UNNAMED
--add-opens=java.base/sun.net.www.protocol.http=ALL-UNNAMED --add-
opens=java.base/sun.net.www.protocol.https=ALL-UNNAMED
--add-opens=java.base/sun.util.calendar=ALL-UNNAMED
--add-opens=jdk.zipfs/jdk.nio.zipfs=ALL-UNNAMED"
```

Recommandations

- **Heap** : Xms = Xmx
  - DEV : 2 Go
  - INT / REC : 4 Go
  - PROD : 6–8 Go (selon charge)
- **Garbage collecting :**
  - -XX:+UseG1GC
  - -XX:MaxGCPauseMillis=200
- **Metaspace** (Important pour OSGI) :
  - -XX:MetaspaceSize=256m
  - -XX:MaxMetaspaceSize=512m
- **Diagnostic / Debug**
  - -XX:+HeapDumpOnOutOfMemoryError
  - -XX:HeapDumpPath=/opt/liferay/heapdumps
  - -XX:+PrintGCDetails
  - -XX:+PrintGCDateStamps
  - -Xloggc:/opt/liferay/logs/gc.log



portal-ext.properties : rôle et limites

portal-ext.properties :

- surcharge les propriétés de base
- global à toute l'instance
- appliqué au démarrage uniquement
- destiné aux paramètres techniques

Exemples fréquents :

```
jdbc.default.driverClassName=com.mysql.cj.jdbc.Driver
jdbc.default.username=liferay
company.default.web.id=liferay.com
theme.css.fast.load=false
```

Référence fichier ***portal.properties*** dans le bundle ***portal-impl.jar***

Configuration via Control Panel

Accessible depuis :

Control Panel > Configuration > System Settings

Permet de configurer :

- recherche
- sessions
- documents & media
- authentification
- sécurité
- modules spécifiques (blogs, web content, DM…)

➡ Modifiable sans restart, stocké dans la table configuration.

Configuration OSGi (.config)

Emplacement : *LIFERAY_HOME/osgi/configs/*.config*

- Chaque fichier correspond à une configuration d'un module OSGI
- Le nom du fichier est l'ID OSGi de la configuration.
  Ex :
  com.liferay.portal.search.elasticsearch7.configuration.CompanyIndexConfiguration.config

Le contenu contient des lignes clé/valeurs :

```
indexNamePrefix=mycompany
indexStorageType=remote
```

Architecture des logs Liferay

Liferay utilise :

- log4j pour la logique globale
- slf4j pour la majorité des modules
- logs catégorisés par package Java
- logs complémentaires OSGi

Fichier clé :

LIFERAY_HOME/tomcat/logs/catalina.out (ou équivalent serveur)

Modifier la verbosité des logs

Deux méthodes :

1) Via Control Panel

System Settings → Logging

Créer un logger :

Category: *com.liferay.journal.service*

Level: *DEBUG*

2) Via *portal-ext.properties* (rarement utilisé)

log4j.logger.com.liferay.journal=DEBUG

Lire une stacktrace Liferay

Une erreur se compose de :

- Type d’erreur : *PermissionException, PrincipalException, NoSuchLayoutException*…
- Module source
- Cause profonde (root cause)
- stack trace complète

Méthodologie :

- Repérer la première ligne “meaningful”
- Identifier module et composant
- Vérifier OSGi “component not satisfied”
- Rejouer scénario en logs DEBUG

Erreurs fréquentes & diagnostic

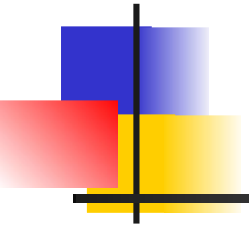
"*NoSuchLayoutException*"
Page supprimée / mauvais friendly URL
Voir Table Layout

"*PrincipalException*"
Permissions manquantes
Voir ResourcePermission

"*ClusterMasterExecutorImpl*"
Problème cluster System Settings

“Could not resolve module”
Dépendances OSGi manquantes
Gogo Shell

Gestion des sessions dans Liferay



Session HTTP + Session Portal

Stockage en mémoire (par défaut)

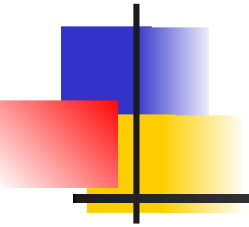
Synchronisation cluster possible via :

- Sticky sessions (recommandé)
- Persistence (Redis, Infinispan) dans DXP

Paramètres clés :

session.timeout=30

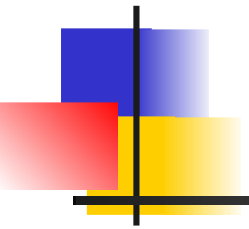
session.timeout.auto.extend=true



Optimisation de la gestion des sessions

Bonnes pratiques :

- Timeout = 15 à 30 min pour intranet, moins en extranet
- Activer session.timeout.auto.extend uniquement si nécessaire
- Surveiller la taille des objets en session
- Éviter de stocker des collections lourdes en session



Optimisations de démarrage / déploiement

Désactiver modules inutiles via Gogo Shell

Supprimer modules marketplace non utilisés

Vérifier :

- DNS
- connexions JDBC
- index de recherche
- état OSGi (“unsatisfied components”)

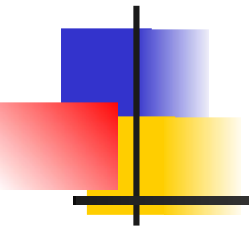
Optimiser la recherche (Elasticsearch)

Options clés :

- Externaliser Elasticsearch
- Multi-Node pour ressource critique
- Activer le monitoring Kibana
- Régénérer les index régulièrement :

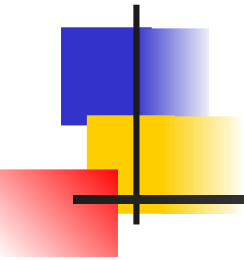
Panneau de contrôle > Recherche > Indexer les actions

À éviter : Elasticsearch embarqué en production.



Optimisation UI (thèmes & front-end)

- Activer minification JS/CSS
- Utiliser Clay plutôt qu'AUI
- Limiter le nombre de portlets par page
- Privilégier les Content Pages pour les pages marketing
- Optimiser les images



Administration & Exploitation

Base de données

Optimisation configuration / logs / sessions

Exploitation des modules OSGI

Patching DXP / mises à jour

Environnements, staging, import/export

Introduction

- Dans Liferay, toute fonctionnalité est un **module OSGi** :
  - CMS
  - Pages
  - Staging
  - Recherche
  - Marketplace
  - Commerce

- Les modules sont chargés à l'exécution
- Les problèmes pouvant arrivés lors de l'exploitation sont souvent un module OSGI mal résolu

Qu’est ce qu’un module OSGI

- **OSGi (Open Services Gateway initiative)** est un système de modules pour Java.
- Il permet de transformer une application monolithique en une collection de composants dynamiques et réutilisables
- Les avantages recherchés :
  - **La modularité** : Le JAR classique devient un Bundle. Chaque bundle possède son propre ClassLoader (isolation totale) et choisit explicitement ce qu’il expose (Export-Package) et ce dont on a besoin (Import-Package).
  - **Cycle de vie dynamique** : On peut installer, démarrer, arrêter ou mettre à jour un module sans redémarrer le serveur.
  - **Registre de Services** : Les modules communiquent via des Services (interfaces). Découplage total : on demande une interface, OSGi injecte la meilleure implémentation disponible.

Annotations OSGI

- Déclaration de composant

```
@Component(
    immediate = true,                // Démarre dès que possible
    property = {
        "osgi.command.function=add", // Exemple pour une commande Gogo Shell
        "osgi.command.scope=event"
    },
    service = EventLocalService.class // Définit l'interface du service publié
)
public class EventLocalServiceImpl implements EventLocalService { ... }
```

- Injection de dépendance

```
@Reference
private UserLocalService _userLocalService;
```

- Cycle de vie

```
@Activate
protected void activate(Map<String, Object> properties) {
    System.out.println("Le composant Event est démarré !");
}

@Deactivate
protected void deactivate() {
    System.out.println("Nettoyage avant l'arrêt...");
}
```

Bnd.bnd

```
Bundle-Name: events-api
Bundle-SymbolicName: org.formation.event.api
Bundle-Version: 1.0.0
Export-Package:\
       org.formation.event.exception,\
       org.formation.event.model,\
       org.formation.event.service,\
       org.formation.event.service.persistence
-check: EXPORTS
-includeresource:
META-INF/service.xml=../events-service/service.xml
```

États des modules OSGi

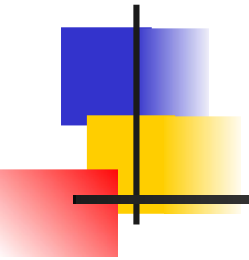
- Un module peut être dans les états suivants :
  - **Active** : Fonctionne normalement
  - **Resolved** : Chargé mais pas actif
  - **Installed** : Présent mais inutilisable
  - **Unsatisfied** : Dépendances manquantes

Diagnostic avec le Gogo Shell

- **Gogo Shell (OSGi Shell)** est l'outil principal pour le diagnostic. Il est accessible via ***Panneau de contrôle → Système → Gogo shell***
- Commandes utiles :
  - ***lb*** : Lister les modules
  - ***lb -s*** : Lister avec les noms symboliques
  - ***lb -s \| grep document*** : Filtrer un domaine fonctionnel
  - ***B [bundle_id]*** → Détail des packages d'un bundle
  - ***install*** → Installation à partir d'un jar
  - ***system:check*** : Vérification complète de tous les modules
  - ***start \| stop*** : Démarrer, arrêter un module
  - ***dm na*** : Liste les modules non résolus
  - ***scr*** : Les composants OSGI @Component, @Reference
  - ***services*** : Liste des services exposés par les bundle, attribut service des @Component

En cas de problème

- Symptômes typiques
  - une application ne s’ouvre pas
  - un menu existe mais renvoie une erreur
  - une fonctionnalité CMS / staging / search est inactive
  - Des exceptions dans les traces
- Lister les modules et identifier ceux liés au problème
- Vérifier le statut
- Si statut *unsatisfied*
  scr:list | grep <nom_du_bundle>
- Activer les logs de debug du logger pour comprendre
- Essayer d’arrêter et redémarrer le module



Administration & Exploitation

Base de données

Optimisation configuration / logs / sessions

Exploitation des modules OSGI

Patching DXP / mises à jour

Environnements, staging, import/export

Pourquoi appliquer des patchs ?

Les patchs **Liferay DXP** fournissent :

- Corrections de bugs
- Correctifs de sécurité (critiques)
- Corrections de performances
- Mise à jour des compatibilités (Java, ES…)
- Préparation aux upgrades mineurs

Recommandé : maintenir un cycle de patching régulier.

Types de patchs DXP

Hotfix Correctif ciblé fourni par Liferay Support À la demande

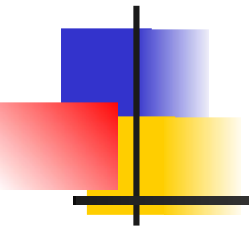
Security Patch Correctifs de vulnérabilités Mensuel / urgent

Fix Pack / GA Update Regroupement de correctifs généraux Trimestriel

Service Pack Métapack de consolidation 1–2 fois par an

En 7.4, la tendance est à : Security Updates + Updates Packs

Patching Tool : rôle



Le Patching Tool permet :

- d'analyser le bundle
- d’installer un patch
- de vérifier les patchs appliqués
- de restaurer un patch
- de générer des rapports de compatibilité

Emplacement :

<LIFERAY_HOME>/patching-tool

Vérifier l'état du serveur

Commande :

./patching-tool.sh info

Affiche :

- version DXP installée
- patchs appliqués
- patchs disponibles
- niveau de support
- état du Patching Tool

Télécharger un patch

Les patchs sont fournis :

- via le Liferay Customer Portal
- via abonnement DXP

Chaque patch = archive ZIP contenant :

- les fichiers modifiés
- un fichier *patch.info*
- scripts de contrôle

Installer un patch

1. Copier le patch dans :

<LIFERAY_HOME>/patching-tool/patches/

2. Exécuter :

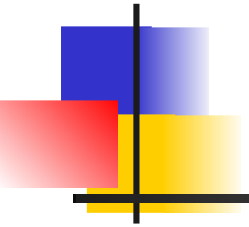
./patching-tool.sh install

3. Redémarrer le serveur Liferay.

L’outil vérifie :

- compatibilité version
- dépendances
- conflits éventuels

Désinstaller / rollback



Deux méthodes :

Si le patch est encore présent :

./patching-tool.sh revert

Sinon :

Supprimer manuellement le patch + réinstaller une version propre.

Toujours faire un snapshot avant patching.

Diagnostiquer un patch mal appliqué

Indicateurs typiques :

- *patching-tool info* ne liste pas le patch
- erreurs au démarrage Tomcat
- modules OSGi en état Unsatisfied
- version incorrecte dans /o/system-status

Solutions :

- patching-tool.sh revert
- vérifier droits d’écriture
- supprimer .liferay dans le dossier du patching tool (réinitialisation)
- réinstaller patch proprement

Patch vs Upgrade

Patch Corrige le système existant Faible

Upgrade mineur GA → GA ou FixPack → FixPack Moyen

Upgrade majeur 7.2 → 7.3 → 7.4 Élevé

Processus d’upgrade majeur

Étapes recommandées :

1. Préparer un environnement de test
2. Mettre à jour :
   - JDK
   - Tomcat
   - modules custom
3. Vérifier compatibilité des apps Marketplace
4. Lancer les scripts upgrade :

./gradlew upgrade

ou

/opt/liferay/tools/portal-tools-db-upgrade-client

5. Vérifier index de recherche
6. Valider via tests fonctionnels

Compatibilité code custom lors de l’upgrade

Checklist :

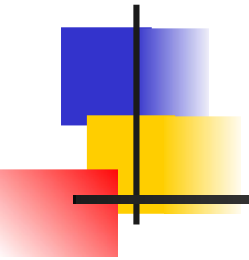
- API obsolètes remplacées
- Service Builder régénéré
- REST Builder mis à jour
- OSGi : vérifier bnd.bnd
- Clay / React : mise à jour front
- Thème recompilé avec nouvelle base Clay

Bonnes pratiques de mise à jour

- Migrer progressivement (version par version)
- Régénérer tous les modules Service Builder
- Ré-indexer après upgrade
- Tester chaque étape (unité / intégration / charge)
- Vérifier les Display Pages & Fragments
- Mettre à jour la base Elasticsearch

Version CE

- L’outil patch n’est pas disponible, on upgrade la version en remplaçant le bundle et en gardant les données
- Ce que l’on garde :
  - BD, répertoire data/, osgi/modules, osgi/configs, portal-ext.properties
- Ce que l’on remplace :
  - Le bundle tomcat
  - Les répertoires techniques : osgi/state, osgi/tmp, tomcat/work, tomcat/temp
- Bonnes pratiques
  - Stopper Liferay avant copie
  - Ne pas copier l’état OSGi
  - Vérifier DB et liferay.home
  - Tester sur un environnement intermédiaire



Administration & Exploitation

Base de données

Optimisation configuration / logs / sessions

Exploitation des modules OSGI

Patching DXP / mises à jour

Environnements, staging, import/export

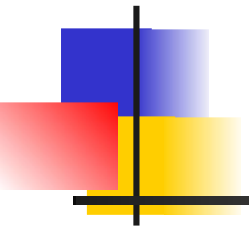
Pourquoi plusieurs environnements ?

Une organisation standard Liferay utilise :

- Dev : développement des modules
- Intégration : assemblage + tests fonctionnels
- Préproduction : tests de charge / validation client
- Production : site final, disponibilité garantie

Objectifs :

- Sécurité
- Qualité
- Continuité de service
- Déploiements contrôlés



Ce qui doit varier selon l’environnement

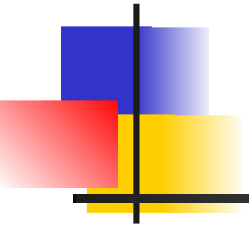
Les paramètres suivants ne doivent jamais être codés en dur :

- URLs externes
- Identifiants LDAP / SSO
- Paramètres DB
- Paramètres Elasticsearch
- Configurations d’email
- Secrets / tokens OAuth2

Paramètres cluster

➡ Externaliser via .config OSGi + variables d’environnement.

Configuration externalisée



Méthodes recommandées :

1. Configurations OSGi

osgi/configs/*.config

Versionnables

Déployables via CI/CD

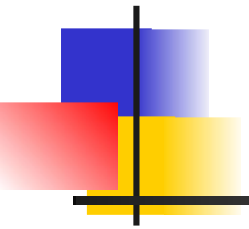
2. Variables d’environnement

Docker / Kubernetes

Remplacent les propriétés dynamiquement

3. portal-ext.properties

Pour les réglages fondamentaux seulement.



Ce qui doit être identique entre environnements

- Modules OSGi déployés
- Version du thème
- Structures / templates du CMS
- Modèle de données
- Permissions globales
- Configuration des rôles

Objectif : éviter la dérive de configuration.

Présentation Export / Import

- Fonctionnalité accessible via :

Site Menu → Publishing → Export / Import

Permet de transférer entre environnements :

- Pages
- Web contents
- Documents & Media
- Structures & Templates
- Catégories & Tags
- Données applicatives compatibles

Export d’un site

- Étapes :
- Aller dans Export
- Sélectionner :
  - Pages
  - Contenus
  - Documents
  - Métadonnées
- Choisir compression & options
- Télécharger le fichier .lar

Le fichier .lar contient :

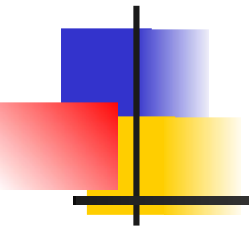
- données sérialisées
- fichiers accompagnants
- permissions facultatives

Import dans un autre environnement

- Étapes :
- Aller dans ***En cours de publication - Import***
- Importer le ***.lar***
- Choisir comportement :
  - Merge with existing
  - Replace
  - Mirror (synchronisation)
- Lancer l'import

Attention : dépendances entre contenus.

- (Règle : importer les structures AVANT les contenus).



Limites de l’Export / Import

- Ne migre pas les configurations système (.config)
- Ne migre pas les utilisateurs (hors LDAP)
- Ne migre pas les configurations applicatives complexes
- Peut créer des doublons si mal utilisé
- Sensible aux permissions

➡ Pour un portail volumineux, préférer le Staging.

Qu’est-ce que le Staging ?

Fonctionnalité permettant de :

- préparer le site dans un environnement “brouillon”
- valider les contenus et pages
- publier manuellement ou automatiquement vers le site “live”

Deux modes :

- Local Staging
- Remote Staging

Local Staging

Les environnements “staged” et “live” sont sur la même instance.

Caractéristiques :

- Facile à configurer
- Idéal pour un intranet ou un petit portail
- Impact minimal sur réseau
- Prévisualisation en direct

Limite :

même base → pas d’isolation complète

Remote Staging

Environnements “staged” et “live” sur deux serveurs différents.

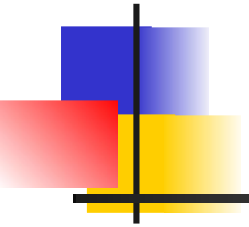
Avantages :

- Isolation complète
- Perf du front intacte
- Séparation totale des données
- Recommandé pour gros volume & sites publics

Limites :

- Configuration réseau
- Cohérence des versions
- Gestion des échecs de réplication

Ce que le Staging publie



- Pages (structure, layouts, fragments)
- Web contents
- Documents & Media
- Structures / Templates
- Taxonomies
- Permissions (optionnel)

Ne publie pas :

- Configurations système
- Modules OSGi
- Paramétrages d’infrastructure

Cycle de publication

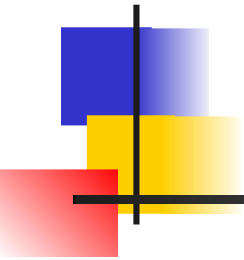
1) Création dans l’environnement de staging
2) Prévisualisation
3) Validation (workflow selon paramétrage)
4) Publication vers live
5) Contenu immédiatement disponible

Modes de publication :

- Manuelle
- Programmée (cron-like)
- Automatique (rare)

Choisir entre Staging et Import/Export

- Site en forte évolution : Staging
- Migration ponctuelle : Export / Import
- Copie de site existant : Export / Import
- Travail collaboratif sur pages : Staging
- Gros portail public : Remote Staging



Développement

Mise en place IDE

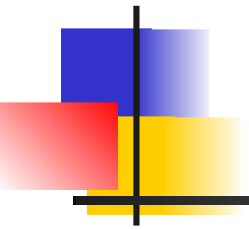
Développement Portlet

Service Builder

REST Builder et Client Extensions

Sécurité

Extensions : Asset, Thèmes, OSGi



Pré-requis pour le poste de développement

- Poste de développement : Windows, Linux ou macOS.
- JDK 8 ou 11 selon la distribution Liferay utilisée.
- Accès à une base de données (PostgreSQL, MariaDB/MySQL, etc.) pour les environnements pérennes.
- Outils de développement :
  - Java, Git,
  - un IDE (Eclipse / Liferay Developer Studio / IntelliJ).

Mode développeur

- Activation (portal-ext.properties) :

theme.css.fast.load=false

layout.template.cache.enabled=false

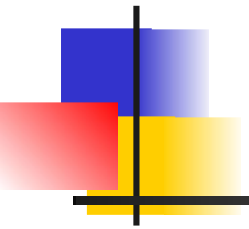
browser.launcher.url=http://localhost:8080

Avantages :

- Affichage des templates
- Réduction du cache
- Logs plus explicites

Mise en place de l'environnement de développement

- Installer ***Blade CLI*** sur le poste de développement.
  - Créer un **Liferay Workspace** dans un répertoire de travail :
    `blade init liferay-workspace`
    - Initialisation Gradle ou Maven
    - configuration de la version produit
- Ce workspace contiendra les modules (apps, thèmes, services, etc.) du projet.



Intégration avec l'IDE

- Importer le workspace dans l'IDE :
  - Liferay Developer Studio (Eclipse packagé) ou Eclipse + plugins Liferay
    Peu maintenu
  - IntelliJ IDEA avec le plugin Liferay.
- L'IDE reconnaît les projets Gradle/Maven du workspace et fournit :
  - assistants de création de modules,
  - support pour le déploiement sur le serveur Liferay,
  - complétion des APIs Liferay.

Atelier 8 : IDE et Premier portlet

Développement

Mise en place IDE

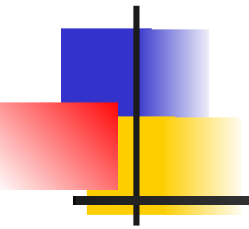
Développement Portlet

Service Builder

REST Builder et Client Extensions

Sécurité

Extensions : Asset, Thèmes, OSGi



Qu’est-ce qu’un portlet ?

- Composant web générant un fragment de page
- Cycle de vie géré par le portlet container
- Contenu final agrégé par le portail
- Interaction via des URLs portlet (*action*, *render*, *resource*)
- Plusieurs instances possibles sur une même page

Structure d’un module MVC Portlet

- Module généré via Blade :
  `blade create -t mvc-portlet ...`
  - ***bnd.bnd*** : informations OSGi du module ;
  - ***build.gradle*** ou ***pom.xml*** : dépendances et configuration de build ;
  - ***src/main/java*** : classe portlet *@Component* étendant MVCPortlet ;
  - ***src/main/resources/META-INF/resources/view.jsp*** (et autres JSP) pour la vue ;
  - ressources additionnelles (i18n, fichiers de configuration).

- Le déploiement s’effectue via Gradle/Maven (tâche deploy) ou en copiant l’archive dans *deploy*.

Cycle de vie / JSR-286

Phases principales :

- **Action**
  - Traitement d’une action utilisateur
  - Mise à jour du modèle + validation
- **Render**
  - Production du HTML final
  - Chargement de la JSP définie par mvcPath
- **Resource**
  - Requêtes AJAX / JSON / flux binaire
  - Implémentée via *MVCResourceCommand*
- **Event**
  - Communication inter-portlets (peu utilisée)

Modes & États

Modes fonctionnels

- ***view*** : affichage
- ***edit*** : configuration
- ***help*** : aide (rare)

États de fenêtre

- ***normal***
- ***maximized***
- ***minimized***

Portlet MVC : principes Liferay 7.4

- Basé sur des modules **OSGi**
- Classe historique contrôleur : **MVCPortlet** avec surcharge des méthodes doView, processAction et serveResource
- Vues : JSP sous ***META-INF/resources***
- Sélection de la JSP via :
  - Attribut d'annotation ou paramètre d'URL ***mvc-path***
- Pattern commande permettant une séparation claire, impliquant 3 classes :
  - Action = *MVCActionCommand*
  - Render = *MVCRenderCommand*
  - Resource = *MVCResourceCommand*

Structure d’un module MVC Portlet

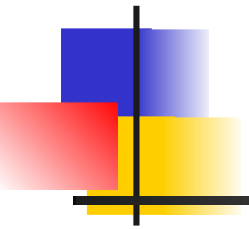
```
hello-web/
  ├─ bnd.bnd
  ├─ build.gradle
  └─ src/main/
       ├─ java/com/acme/hello/HelloPortlet.java
       └─ resources/META-INF/resources/
            ├─ view.jsp
            ├─ edit.jsp
            └─ css/js/img…
```

Exemple minimal : classe portlet

```
@Component(
  property = {
    "javax.portlet.name=demo_mvc",
    "javax.portlet.init-param.mvc-path=/view.jsp",
    "javax.portlet.display-name=Demo MVC"
  },
  service = Portlet.class
)
public class DemoMVCPortlet extends MVCPortlet { }
```

MVCActionCommand : traitement & validation

```
@Component(
  property = {
    "javax.portlet.name=demo_mvc",
    "mvc.command.name=saveEntry"
  },
  service = MVCActionCommand.class
)
public class SaveEntryMVCActionCommand 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, "entry-saved");
    return true;
  }
}
```



Navigation entre vues : *mvcPath*

Dans une JSP :

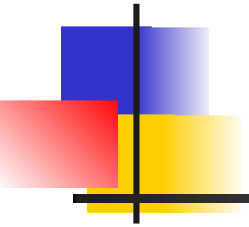
```
<portlet:renderURL var="editURL">
    <portlet:param name="mvcPath" value="/edit.jsp"/>
</portlet:renderURL>
```

Dans une *ActionCommand* :

```
resp.setRenderParameter("mvcPath", "/edit.jsp");
```

Validation : concepts Liferay

- Validation « métier » dans *MVCActionCommand*
- Utilitaires :
  - ***ParamUtil***
  - ***Validator***
- Messages :
  - Erreurs : SessionErrors.add
  - Information : SessionMessages.add
- Affichage dans la JSP via :
  - `<liferay-ui:error`
  - `<liferay-ui:success`



Validation côté service

- La couche service gère la validation avancée
- En cas d’erreur → exception métier (ex. *EntryValidationException*)
- Permet une validation centralisée, réutilisable via APIs REST

Atelier 9 : Portlet MVC avec validation

Développement

Mise en place IDE

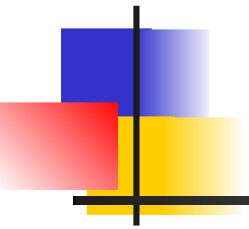
Développement Portlet

Service Builder

REST Builder et Client Extensions

Sécurité

Extensions : Asset, Thèmes, OSGi



Objectifs

- Définir un modèle métier avec ***service.xml***
- **Générer** tables, modèle, persistance & services OSGi
- **Implémenter** règles métier dans la couche service
- **Exposer** des APIs distantes modernes RestFul

Etapes de mise en place

1) Décrire le domaine dans ***service.xml***

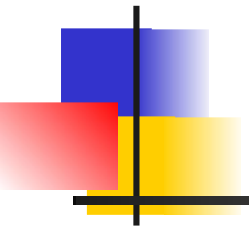
2) Exécuter ***gradlew buildService***

3) Compléter ***LocalServiceImpl*** avec la logique métier

4) Appeler le service depuis les portlets / remote apps

Exemple d’entité

```
<entity name="Event" local-service="true" remote-service="false">
    < !-- Colonnes identifiantes -->
    <column name="eventId" type="long" primary="true" />
    <column name="groupId" type="long" />
    <column name="companyId" type="long" />
    <column name="userId" type="long" />
    < !-- Champs spécifiques -->
    <column name="name" type="String" />
    <column name="date" type="Date" />
    < !-- Requêtes -->
    <finder name="GroupId" return-type="Collection">
        <finder-column name="groupId"/>
    </finder>
</entity>
```

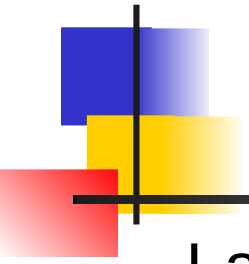


Tables générées

Table SQL : même nom que l’entité préfixé par le namespace

Avec colonnes auto-gérées :

- *companyId*
- *groupId*
- *userId*
- *uuid_*
- *createDate, modifiedDate*



Fichiers générés

La commande *blade* de création de projet génère 2 modules :

- Module **Service** contenant *service.xml*
- Module **API** qui reflète les méthodes qui sont implémentées

Après avoir mis au point service.xml, on génère les classes via une commande Gradle
./gradlew buildService

Les classes sont générées dans les 2 modules :

- Service : Classes **d'implémentation** appliquant des patterns classiques
- API : **Interfaces** utilisées

Principales interfaces

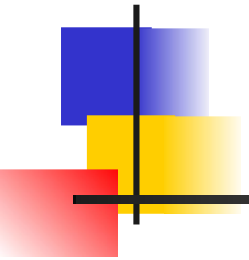
- Les interfaces sont utilisées par les autres modules. Elles sont organisées en package
  - Model : ***<Entity>***
  - Service :
    - ***EventLocalService*** : Les méthodes exposées et utilisables par les autres modules
    - ***EventPersistence*** : Interface pour gérer la persistance de l'entité utilisée par l'implémentation du service

Implémentation du Service local

Dans le module service, *ServiceBuilder* génère l'implémentation ***<Entity>LocalServiceImpl*** :

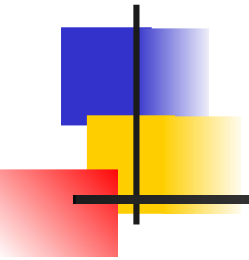
- C'est la classe que l'on peut modifier en y ajoutant des méthodes métier.
- ServiceBuilder synchronise automatiquement l'interface publiée.

- Le code gère automatiquement :
  - Transactions
  - Injections OSGi
  - Permissions (si intégrées)



Exemple

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

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

    Event event = eventPersistence.create(eventId);

    event.setGroupId(groupId);
    event.setCompanyId(sc.getCompanyId());
    event.setUserId(userId);
    event.setName(name);
    event.setDate(date);

    return super.addEvent(event);
}
```



Utiliser des services

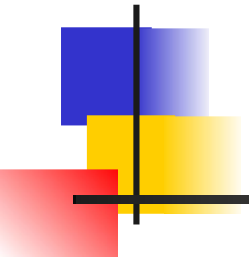
- 2 alternatives :
  - Utiliser directement le service en **local**.
    Le module portlet a une dépendance sur l'api et se fait injecter le service
  - Accéder au service en mode remote :
    - **SOAP** = héritage, non recommandé
    - **JSON Web Services/JAX-RS**, éventuellement à désactiver
    - Solution moderne = **REST Builder** : Documentation automatique OpenAPI, Sécurisation OAuth2

Intégration portlet du serviceLocal

- Ajouter la dépendance vers l'API dans build.gradle
- S'injecter l'interface via OSGI

```
@Reference
private EventLocalService _eventLocalService;
```

- Utiliser les méthodes dans les méthodes des portlets



Développement

Mise en place IDE

Développement Portlet

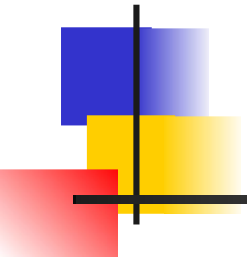
Service Builder

REST Builder et Client Extensions

Sécurité

Extensions : Asset, Thèmes, OSGi

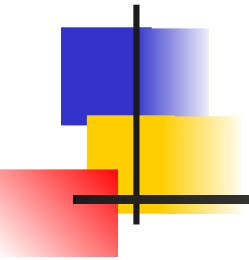
RestBuilder



- La commande blade de démarrage de projet crée 2 modules principaux :
  - **Api** : Contient les interfaces de service REST et les DTO.
    C'est là que l'on définit les ressources (méthodes GET, POST, etc.).
  - **Impl** : Contient l'implémentation de ces services REST.
    C'est là que l'on injecte le Service local et que l'on fait le mapping entre vos entités Service Builder et les DTO Headless.

Définition du contrat

- Le fichier -api/src/main/resources/***rest-openapi.yaml*** permet de définir le contrat en OpenAPI
- Ensuite, ***./gradlew clean buildRest*** permet de générer :
  - Les classes DTO Java
  - L'interface Java de votre service Headless
  - Le squelette de la classe d'implémentation dans le sous-module ***-impl***.
  - Toute la configuration JAX-RS (Application avec les propriétés OSGi) nécessaire



Implémentation

- De la même façon, l’implémentation a une dépendance sur l’API du service et se fait injecter le service local existant par OSGI
- Elle est également responsable de mapping entre classe du modèle et DTP

Exemple

```
public class EventResourceImpl extends BaseEventResourceImpl {

@Reference
private EventLocalService _eventLocalService;

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

// Appel du service local
List<org.formation.event.model.Event> events =
_eventLocalService.getEvents(groupId, 0, 20);

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

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

Atelier 12.1 : Service Restful via RestBuilder

Client extensions

- Dans les récentes versions de Liferay, une alternative au portlet est proposée
- Le frontend est alors développé avec des technologies standards (React, Vue, Angular ou Vanilla JS) et communique avec Liferay via des APIs REST.
- Avantages :
  - Indépendance technologique : Pas besoin de compétences Java pour créer une interface.
  - Sécurité du Core : Le code personnalisé s'exécute dans le navigateur ou sur un serveur distant, sans risque de faire "tomber" le portail.
  - Mise à jour simplifiée : On peut modifier le frontend sans redémarrer le serveur Liferay

Anatomie d'un "Élément Personnalisé"

- Pour que Liferay affiche l’application JS comme un Widget, 3 éléments sont nécessaires :
  - **Le Code (index.js)** : Un Web Component standard (CustomElement) qui contient la logique et le rendu HTML.
  - **La Déclaration (YAML ou UI)** : On définit l'identité de l'application :
    - ***htmlElementName*** : Le tag HTML unique (ex: <my-app>).
    - ***url*** : Le chemin vers le fichier JavaScript.
  - **Le Rendu** : Liferay injecte automatiquement la balise et le script sur la page. Le navigateur instancie alors l'application.

Communication Liferay et Authentification

- Appels API : Utilisation de l'API fetch() vers les endpoints natifs (ex: /o/c/evenements).

- Sécurité native :
  - L'application étant sur le même domaine, elle bénéficie des cookies de session.
  - Le Jeton (Auth Token) : Liferay fournit un objet global Liferay.authToken pour sécuriser les requêtes contre les failles CSRF.
- Cycle de vie :
  - L'utilisateur charge la page.
  - Le composant JS est monté.
  - Le composant appelle l'API REST des événements.
  - Les données sont affichées dans le widget.

Autres types de client extension

- Les extensions d'UI
  - ***Custom Element***
  - ***JS / CSS*** : Permet d'injecter des fichiers JavaScript ou des feuilles de style globalement sur le portail ou sur une page spécifique
  - ***Theme CSS*** : Spécifique pour surcharger les variables CSS d'un thème sans avoir à créer un thème complet.
- Les extensions de "Logique" et Data
  - ***Batch*** : Permet d'importer massivement des données (Utilisateurs, Objets, Entrées de blog) via des fichiers JSON au moment du déploiement.
  - ***Object Action*** : Permet d'appeler une URL externe (une fonction Lambda, une API Node.js) dès qu'un événement survient dans Liferay
  - ***Workflow Action*** : Similaire aux actions d'objet, mais déclenché lors des étapes d'un flux de validation.
- Les extensions de Configuration et Traduction
  - ***I18n*** : Permet d'ajouter de nouvelles clés de traduction pour le portail ou d'écraser les traductions natives de Liferay.
  - ***Configuration de l'éditeur*** : Permet de personnaliser les barres d'outils de l'éditeur de texte riche (CKEditor) pour ajouter ou supprimer des boutons.

Atelier 12.2 : CustomElement

Développement

Mise en place IDE

Développement Portlet

Service Builder

REST Builder et Client Extensions

Sécurité

Extensions : Asset, Thèmes, OSGi

Principes des permissions

- une permission répond toujours à la question :
  “Qui peut faire quoi sur quoi ?
- Liferay combine :
  - des utilisateurs
  - des rôles
  - des ressources
  - des actions (VIEW, UPDATE, DELETE…)
- Le même moteur de permissions est utilisé pour :
  - les pages
  - les contenus
  - les portlets
  - les APIs REST

Modèle

- Ressource : un objet à sécuriser
  (page, contenu, document, entité métier, API…)
- Action : ce que l’on peut faire
  (VIEW, ADD, UPDATE, DELETE…)
- Rôle : Affecté à un utilisateur
  Utilisateur, Administrateur, Rédacteur…

Une permission = Rôle + Action + Ressource

Déclaration des permissions

- Chaque module peut (et doit) déclarer ses propres permissions dans un fichier défini par portlet.properties :
  `resource.actions.configs=resource-actions/default.xml`
- On y définit :
  - les actions possibles et visibles dans l'interface
  - les actions autorisées par défaut

Exemple portlet

```
<resource-action-mapping>
    <portlet-resource>
        <portlet-name>com_acme_events_web_portlet_EventsPortlet</portlet-name>
        <permissions>
             < !-- actions supportées -->
            <supports>
                <action-key>VIEW</action-key>
                <action-key>ACCESS_IN_CONTROL_PANEL</action-key>
                <action-key>CONFIGURATION</action-key>
                <action-key>PERMISSIONS</action-key>
            </supports>

            <site-member-defaults>
                <action-key>VIEW</action-key>
            </site-member-defaults>

            <guest-defaults>
                <action-key>VIEW</action-key>
            </guest-defaults>
             < !-- actions visibles et assignables dans l'interface -->
            <action-keys>
                <action-key>VIEW</action-key>
                <action-key>ACCESS_IN_CONTROL_PANEL</action-key>
                <action-key>CONFIGURATION</action-key>
                <action-key>PERMISSIONS</action-key>
            </action-keys>
        </permissions>
    </portlet-resource>
</resource-action-mapping>
```

Exemple Modèle métier

```
<model-resource>
    <model-name>com.acme.events.model.Event</model-name>
<!-- Permissions gérées par le portlet -->
    <portlet-ref>
        <portlet-name>com_acme_events_web_portlet_EventsPortlet</portlet-name>
    </portlet-ref>
    <permissions>
        <supports>
            <action-key>VIEW</action-key>
            <action-key>UPDATE</action-key>
            <action-key>DELETE</action-key>
            <action-key>PERMISSIONS</action-key>
            <action-key>REGISTER</action-key>  <!-- Actions “métier” optionnelles -->
        </supports>
        <site-member-defaults>
            <action-key>VIEW</action-key>
        </site-member-defaults>
        <guest-defaults>
            <action-key>VIEW</action-key>
        </guest-defaults>
        <guest-unsupported>
            <action-key>UPDATE</action-key>
            <action-key>DELETE</action-key>
            <action-key>REGISTER</action-key>
        </guest-unsupported>
        <owner-defaults>
            <action-key>VIEW</action-key>
            <action-key>UPDATE</action-key>
            <action-key>DELETE</action-key>
        </owner-defaults>
        <action-keys>
            <action-key>VIEW</action-key>
            <action-key>UPDATE</action-key>
            <action-key>DELETE</action-key>
            <action-key>PERMISSIONS</action-key>
            <action-key>REGISTER</action-key>
        </action-keys>
    </permissions>
</model-resource>
```

Application des permissions

- Les permissions s’appliquent à plusieurs niveaux :
  - Dans l’interface :
    - cacher une ligne
    - désactiver un bouton
    - masquer une action
  - Dans la couche service
    - bloquer une modification
    - empêcher une suppression
  - Dans les APIs REST
    - autoriser ou refuser l’accès à une ressource

ServiceContext

- Le contexte d’exécution ***ServiceContext*** véhicule les informations sur l’utilisateur et donc ses permissions associées.
- Les champs de serviceContext contiennent entre autres
  - ***userId***: Indique l’utilisateur qui effectue l’opération
  - ***scopeGroupId*** :Groupe/site d’exécution
  - ***companyId*** : Société concernée
  - ***permissions*** : Définit les droits donnés à la ressource créée
  - ***assetCategoryIds*** : Catégories associées
  - ***assetTagNames*** : Tags associés
  - ***workflowAction*** : Ex : publier ou enregistrer comme brouillon

Récupération du ServiceContext

- Depuis une portlet MVC :

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

- Depuis un service, appelé dans un flux web:

```
ServiceContext sc =

   ServiceContextThreadLocal.getServiceContext();
```

- Ou mieux, le portlet passe l’argument ServiceContext au service

- Hors Flux Web, il faut l’instancier et positionner le groupId et le userId

```
ServiceContext sc = new ServiceContext();
sc.setScopeGroupId(groupId);
sc.setUserId(userId);
```

Vérification des permissions

- La vérification des permissions nécessite :
  - De récupérer l'utilisateur courant
  - D'évaluer ses rôles
  - De vérifier s'il a le droit demandé
- Cela se fait via 2 classes Java :
  - *PermissionChecker*
  - *ModelResourcePermission* qui offre une méthode ***check***
- Le résultat est un booléen :
  - autorisé → action possible
  - refusé → action bloquée ou masquée

ModelResourcePermission

- Pour chaque modèle métier (Event, Article, etc.), un ***ModelResourcePermission*** centralise :
  - les règles
  - la notion de propriétaire
  - les rôles
  - les règles personnalisées
- Toute vérification passe par lui :
  - UI
  - Service
  - REST / GraphQL

Enregistrement ModelResourcePermission

- Le développeur doit exposer un service OSGI qui crée l'instance de *ModelResourcePermission*
- Lors de l'instanciation, le développeur doit renseigner :
  - La classe métier impliquée
  - La méthode permettant de récupérer l'identifiant
  - La méthode permettant de charger l'objet

Enregistrement d'un ModelResourcePermission

```
@Component(immediate = true)
public class EventModelResourcePermissionRegistrar {

@Activate
public void activate(BundleContext bundleContext) {
  Dictionary<String, Object> properties = new HashMapDictionary<>();
  properties.put("model.class.name", Event.class.getName());

// Enregistrement manuel dans le registre OSGi
bundleContext.registerService(
  ModelResourcePermission.class,
  ModelResourcePermissionFactory.create(
        Event.class,
        Event::getEventId,
       _eventLocalService::getEvent,
       _portletResourcePermission,
        (modelResourcePermission, consumer) -> { // Possibilité d'ajouter des validateurs ici }),
       properties);
}

@Deactivate
public void deactivate() { _serviceRegistration.unregister(); }

@Reference
private EventLocalService _eventLocalService;
@Reference(target = "(resource.name=com_acme_events_web_portlet_EventsPortlet)")
private PortletResourcePermission _portletResourcePermission;
}
```

Configuration des permissions d'un Widget (Interface)

- Accès à la configuration : Chaque widget (portlet) déposé sur une page possède un menu "Droits d’accès" (icône engrenage).
- Onglet Permissions : Permet de définir manuellement quel rôle peut :
  - Voir, Ajouter à la page, Configurer, Droits d’accès, ..
  - Configuration : Modifier les réglages propres au widget.
  - Permissions : Déléguer la gestion des droits de ce widget à d'autres rôles.
- Portée : Ces réglages s'appliquent à l'instance spécifique du widget sur la page ou au niveau du site selon le contexte.

Déclaration des permissions (resource-actions)

Le fichier portlet.properties indique le fichier de définition des permissions :

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

Le fichier ***resource-actions/default.xml*** déclare :

- les actions supportées
- les actions autorisées par défaut

Permissions d’un portlet

```
<?xml version="1.0"?>
<!DOCTYPE resource-action-mapping PUBLIC "-//Liferay//DTD Resource Action
Mapping 7.4.0//EN" "http://www.liferay.com/dtd/liferay-resource-action-
mapping_7_4_0.dtd">
<resource-action-mapping>
    <portlet-resource>
        <portlet-name>org_formation_event_EventPortlet</portlet-name>
        <permissions>
            <supports>
                <action-key>ADD_EVENT</action-key>
                <action-key>VIEW</action-key>
            </supports>
            <site-member-defaults>
                <action-key>VIEW</action-key>
                <action-key>ADD_EVENT</action-key>
            </site-member-defaults>
        </permissions>
    </portlet-resource>

</resource-action-mapping>
```

Permissions pour une entité

```
<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>
  </permissions>
</model-resource>
```

Fonctionnement

Lorsque Liferay reçoit une demande de permission comme

- afficher une ligne dans une liste
- accéder à un détail
- exécuter une action via un bouton
- REST / GraphQL / Service Builder check

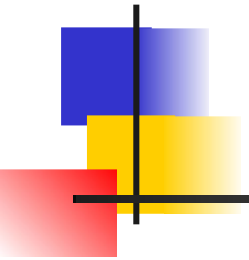
Il appelle : ***modelResourcePermission.check.***
Par exemple :

```
modelResourcePermission.check(permissionChecker, event, ActionKeys.VIEW);
```

modelResourcePermission

- consulte les règles définies dans *resource-actions/default.*xml
- Vérifie si l'utilisateur a les permissions nécessaires
- Applique éventuellement les règles personnalisées
- Retourne Autorisé ou refusé

En fonction de la valeur de retour, Liferay cache la ligne si VIEW est refusé, désactive un bouton si UPDATE est refusé, bloque un appel REST/GraphQL, masque une action dans la management toolbar, ...



Vérification des permissions

La vérification des permissions doit être implémentée

- Au niveau **présentation** : Ne pas présenter des liens ou boutons non autorisés
- Au niveau **service** : Vérifier que l'utilisateur a le droit d'effectuer telle opération sur la base
- = > Aucun code n'est nécessaire au niveau portlet, Liferay automatiquement effectue les contrôles

Vérification dans les Portlets (UI & Contrôleur)

- Côté Contrôleur (Java) : Liferay effectue des contrôles automatiques pour l'accès général, mais le développeur peut affiner la logique en utilisant le PermissionChecker.

- Côté Vue (JSP) : Utilisation des tags de sécurité pour masquer les éléments :

```
<c:if test="<%=
_eventModelResourcePermission.contains(permissionChecker, event,
ActionKeys.UPDATE) %>">

    <button>Modifier</button>

</c:if>
```

- ***<liferay-security:permissionsURL>*** pour offrir un lien direct vers l'interface de gestion des droits d'une entité précise

Vérification des permissions dans un service

La vérification des permissions au niveau **service** consiste à vérifier que l'utilisateur a les permissions pour l'action demandée

```
_modelResourcePermission.check(
    permissionChecker, eventId, ActionKeys.UPDATE);
```

L'instance *_modelResourcePermission* est injecté :

```
@Reference private ModelResourcePermission<Event>
_eventModelResourcePermission;
```

L'instance permissionChecker s'obtient via :

```
PermissionChecker permissionChecker =
PermissionThreadLocal.getPermissionChecker();
```

Sauvegarde d’une permission pour une entité

- Lors de la création d’une entité, il faut également sauvegarder les permissions associées
- Il faut utiliser l’attribut hérité *resourceLocalService* et appeler :

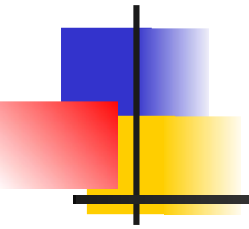
```
resourceLocalService.addModelResources(event, sc);
```

Exemple : Contrôle de permission dans un contrôleur Rest

```
@Component(
    properties = "OSGI-INF/liferay/rest/v1_0/events.properties",
    service = EventsResource.class
)
public class EventsResourceImpl extends BaseEventsResourceImpl {


    @Reference
    private ModelResourcePermission<Event> _eventModelResourcePermission;


    @Override
    public Event getEvent(Long eventId) throws Exception {


        // Vérification permission VIEW
        _eventModelResourcePermission.check(
            getPermissionChecker(), eventId, ActionKeys.VIEW);


        return _eventLocalService.getEvent(eventId);
    }
}
```

Atelier 13 : Permissions et ServiceContext

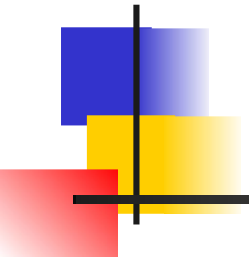
Mise en place OAuth2 dans Liferay



- Liferay embarque un serveur OAuth2 complet conforme à la norme.
- On l’active à partir du Control Panel :
  Applications → OAuth2 Administration
- Dans Liferay, OAuth2 sert à :
  - Autoriser l’accès à des API REST Builder
  - Délivrer des jetons d’accès aux utilisateurs ou aux applications
  - Protéger les endpoints grâce aux scopes et aux permissions du modèle

3 cas d’usage OAuth2 typiques sur Liferay

- Application Java/Backend → Liferay
  - Utilise Client Credentials
  - Obtenir un token → appeler API REST Builder
- Remote App React → Liferay
  - Utilise Authorization Code
  - L’utilisateur s’authentifie → Liferay renvoie le token → la Remote App appelle l’API
- Un utilisateur authentifié → App externe → Liferay
  - OAuth2 Authorization Code
  - Permet un “Connexion via Liferay” / SSO API



Développement

Mise en place IDE

Développement Portlet

Service Builder + REST Builder

Sécurité

REST Builder et Client Extensions

Extensions : Asset, Thèmes, OSGi

Asset Framework

- Permet d’ajouter à tout contenu :
  - Tags
  - Catégories
  - Commentaires
  - Notes / Ratings
  - Relations entre contenus
  - Workflow
  - Diffusion via Asset Publisher
- Support toujours 100% opérationnel en 7.4.

Intégration Asset pour une entité custom

- Ajouter AssetEntry lors de la création / mise à jour d'un modèle.
- Utiliser les services OSGi :

```
@Reference
private AssetEntryLocalService assetEntryLocalService;

assetEntryLocalService.updateEntry(
    userId, groupId,
    Event.class.getName(), eventId,
    categoryIds, tagNames
);
```

- Permet la publication via les widgets standard Liferay.

Composants UI associés

Options disponibles :

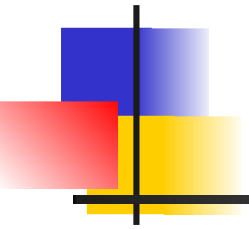
- Taglibs héritées :
  - *<liferay-ui:asset-tags-selector>*
  - *<liferay-ui:asset-categories-selector>*
- Composants modernes :
  - Clay React (sélecteurs de tags/categories)
- Asset Publisher :
  - Tri
  - Filtrage
  - Gabarits d’affichage

Atelier 14.1 : Asset framework

Thèmes Liferay

Les apports d’un thème

- Uniformiser l’identité visuelle
- Modifier gabarits de page / navigation
- Adapter le responsive design
- Ajouter des variables Clay / custom CSS
- Intégration possible avec outils front modernes



Technologies pour les thèmes

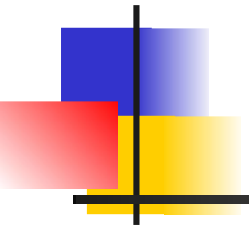
- SCSS / CSS moderne
- Clay / Bootstrap 4
- Freemarker ou JSP pour les templates
- Customisation via :
  - *portal_normal.ftl*
  - *navigation.ftl*
  - *init_custom.ftl*

Structure d’un thème

```
src/
  css/
    _custom.scss
  templates/
    portal_normal.ftl
    navigation.ftl
  images/
liferay-look-and-feel.xml
```

Recommandations :

- Centraliser les styles dans *_custom.scss*
- Ne jamais surcharger directement *_unstyled* ou *_styled*



Personnalisation visuelle

- Utiliser variables SCSS Clay (couleurs, hauteurs, bordures…)
- Ajouter vos propres règles CSS
- Modifier la structure via Freemarker
- Ajouter des fragments de thème (header/footer)
- Support complet du responsive via Bootstrap 4

Atelier 14.2 : Thème

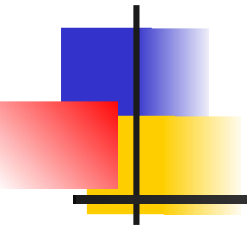
OSGi vs Hooks et Ext

Hooks & EXT (Liferay 6.x / 6.2) :

- Lourds
- Fragiles aux mises à jour
- Couplage fort au portail
- Compatibilité réduite en 7.x

Modernisation :

- Modules OSGi légers
- Déploiement dynamique
- Overrides ciblés
- stable, maintenable, recommandé



OSGi : les 4 mécanismes de personnalisation

- Fragment Modules
  - → Override de JSP / ressources statiques
- Service Wrappers
  - → Surcharge de la logique des services Liferay
- Model Listeners
  - → Réagir aux événements CRUD
- Configuration Admin
  - → Modifier le comportement via des fichiers .config

Fragment Modules (remplace JSP Hook)

Objectif : remplacer une JSP Liferay sans EXT.

Structure :

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

Utilisation :

- Modifier l’IHM native du portail
- Injecter du HTML ou Clay
- Corriger l’affichage d’un module natif

Service Wrappers OSGi

Remplace *UserLocalServiceWrapper* des Hooks historiques.

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

    @Override
    public User addUser(...) {
        // custom logic
        return super.addUser(...);
    }
}
```

Utilisation :

- Auditer
- Ajouter des règles métier
- Étendre les services internes

Model Listeners OSGi

- Écoute les événements d’un modèle (CRUD).

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

    @Override
    public void onAfterUpdate(Event event) {
        // log, sync, external notification…
    }
}
```

Exemples d’usage :

- Log métier
- Notifications
- Vérifications supplémentaires

Atelier 15 : Personnalisation OSGI