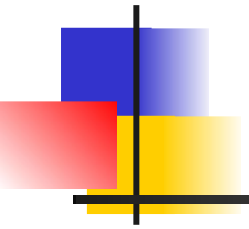
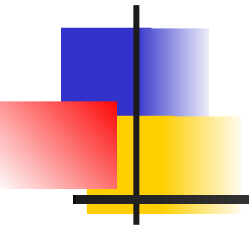

PLB
FORMATION INFORMATIQUE ET MANAGEMENT



# Liferay 7.x - Développer un portail d'entreprise

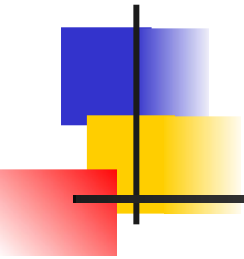
David THIBAU – 2025

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

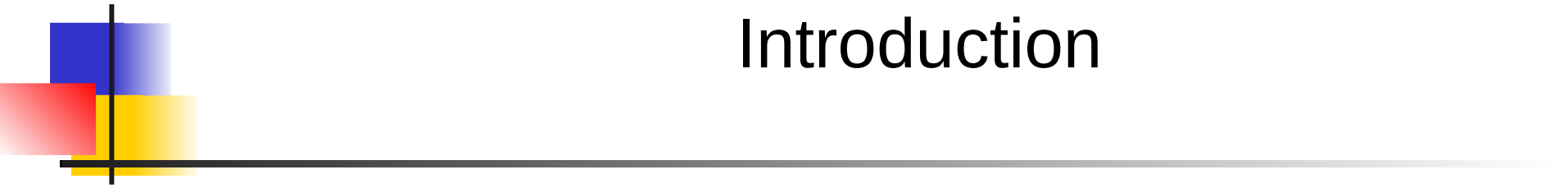
- **CMS intégré :**
  - gestion des pages, des gabarits, des contenus web,
  - workflow éditorial, staging, planification de publication,
  - taxonomies, tags, segmentation.
- **Collaboration :**
  - blogs, wikis, forums, agendas, bibliothèque documentaire,
  - communautés internes ou externes,
  - fonctionnalités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

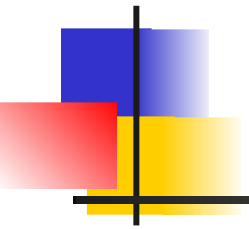
# Pré-requis pour l’installation

- **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

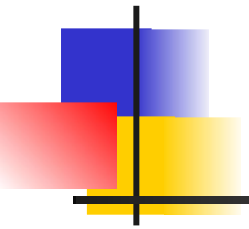
- Télécharger le bundle Liferay 7.4 Tomcat (DXP ou Portal CE) depuis le site officiel.
- Décompresser l’archive dans un répertoire dédié, qui devient le Liferay Home.
- Vérifier la variable d’environnement JAVA_HOME et le chemin du JDK.

# Structure de Liferay Home

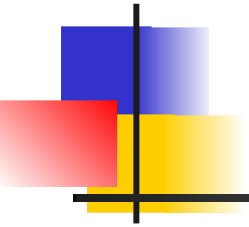


- Le dossier racine contient notamment :
  - ***data*** : données persistées par Liferay (documents, configuration, éventuelle base de démonstration) ;
  - ***deploy*** : répertoire pour le déploiement automatique de modules/LPKG ;
  - ***logs*** : journaux applicatifs ;
  - ***osgi*** : modules standard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

# Démarrage et assistant de configuration



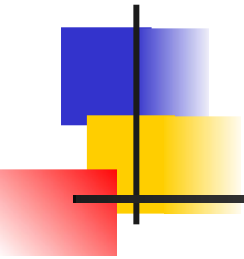
- Lancer le serveur :
  - sous Linux : ***./tomcat-*/bin/startup.sh,***
  - sous Windows : ***tomcat-*\bin\startup.bat.***
- Accès au portail via http://localhost:8080.
- À la première connexion :
  - choix de la langue et du nom du portail ;
  - création du compte administrateur ;
  - configuration 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

# Pages de contenu dans Liferay

## Widget Pages

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

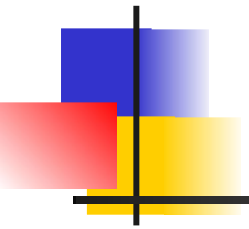
= > Idéal pour les pages dynamiques

## 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

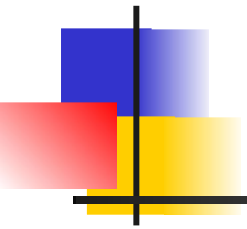


- Les **Widget Pages** sont le modèle historique de Liferay :
  - Basées sur des portlets (applications)
  - Mise en page via layouts en colonnes (1, 2, 3 colonnes, etc.)
  - Chaque zone de la page accueille une ou plusieurs portlets
  - Adaptées aux applications métier et écrans riches (portlets custom, 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

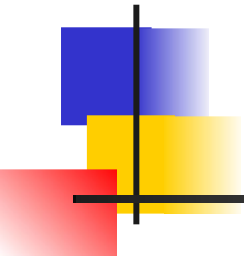
Pour un projet d’entreprise :

- 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 :

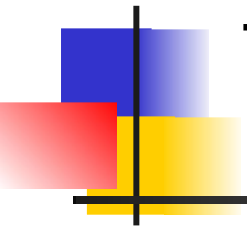
- Séparer contenu et mise en forme
- Éviter le HTML dans les contenus
- Garantir l’homogénéité : toutes les actualités ont le même modèle
- Publication automatisée via Asset Publisher ou Display Page
- Réutilisation multi-sites
- Multilingue propre

# 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

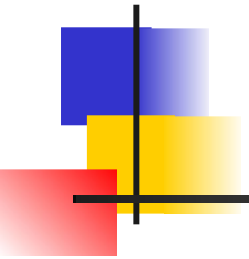
# Tags vs Catégories

## 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**

Configuration – logs, sessions, OSGI

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, Pages

## 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 conteneur de ressources (documents, rôles, permissions, contenus). Un site Liferay est un conteneur de ressources

- Lié à la table *Company* via *companyId*

## Table LayoutSet

- Représente la configuration d’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** représente une page

Colonnes clés :

- *plid* → ID de la page
- *groupId* → site auquel la page appartient
- *privateLayout* → booléen : page publique ou privée
- *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, Organisations et Groupes

- L'Utilisateur (***User_***) : L'entité centrale (compte, email, mot de passe).
  Chaque utilisateur possède automatiquement son propre espace de pages (dans la table Group_).

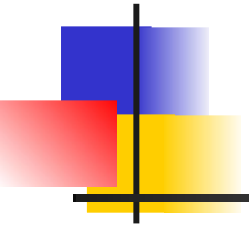
- L'Organisation (***Organization_***) :
  - Structure hiérarchique (Parent/Enfant) reflétant souvent l'organigramme de l'entreprise.
  - Un utilisateur peut appartenir à plusieurs organisations.

- Le Groupe d'Utilisateurs (***UserGroup***) :
  - Regroupement transverse (ex: "Tous les chefs de projet").
  - Contrairement à l'organisation, il n'y a pas de hiérarchie stricte, c'est un ensemble "plat".

# Les rôles

- 3 Types de Rôles (***Role_***) :
  - **Rôle Régulier** : Global à toute l'instance (ex: Administrateur, Utilisateur).
  - **Rôle de Site** : Droits valables uniquement au sein d'un site spécifique (ex: Gestionnaire de contenu de site).
  - **Rôle d'Organisation** : Droits valables uniquement au sein d'une organisation.

- Les rôles réguliers sont associés aux utilisateurs via ***Users_Roles***,
  les rôles contextuels via ***UserGroupRole*** :



# Permissions sur les ressources

- Dans Liferay, tout est ressource (une Page, un Document, un Portlet, un Article, une entité).
- La table ***ResourcePermission*** définit l'accès en croisant 3 données :
  - La Ressource : Quel objet est protégé ? (***name + primKey***).
  - Le Rôle : Qui possède le droit ? (***roleId***). (global ou contextuel)
  - L'Action : Que peut-il faire ? (***actionIds*** : Voir, Modifier, Configurer, etc.) sous forme de bitmask

- Exemple :  Quand un utilisateur tente d'accéder à une page, Liferay vérifie si l'un des rôles de l'utilisateur possède la permission VIEW sur la ressource Layout.

# Permissions : tables critiques

Tables liées au permissions :

- ***ResourceAction*** : Liste des actions possibles (VIEW, UPDATE…)
- ***ResourcePermission*** : Autorisation sur des instance précise.
  Exemple : Tel rôle a le droit de voir telle page
- ***ResourceTypePermission*** : Permissions par type de ressource
  Exemple ; Tel rôle a le droit de créer telle ressource

Diagnostic classique :

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

# Table du CMS

***JournalArticle*** : Contenu (toutes versions)

***JournalArticleLocalization*** : Multilingue

***JournalFolder*** : Arborescence

***DDMStructure*** : Structure

***DDMTemplate*** : Template Freemarker

Points clés :

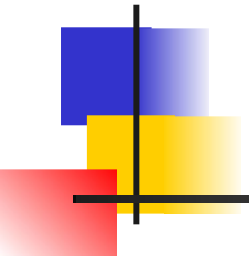
- Un Web Content possède plusieurs versions (version)
- Le champ *resourcePrimKey* relie toutes les versions entre elles
- Workflow géré via *status* + *statusByUserId*

# 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

**Configuration logs, sessions, OSGI**

Exploitation des modules OSGI

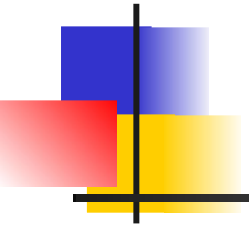
Patching DXP / mises à jour

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

# Architecture des logs Liferay

Frameworks :

- ***SLF4J*** : API utilisée par les développeurs dans les modules.
- ***Log4j*** : Moteur interne qui traite et redirige les flux.

Granularité Java :

- Niveaux configurables par package ou par classe (DEBUG, INFO, WARN, ERROR).
- Modifiable à chaud via :
  ***Control Panel > Configuration > Server Administration > Log Levels***.

Logs OSGi :

- Gestion des cycles de vie des bundles et erreurs de dépendances gérés par OSGI Log Service
- Liferay implémente un pont qui redirige vers Log4j

Fichiers de sortie :

- ***LIFERAY_HOME/logs/liferay.log*** (Journal principal de l'application).
- ***LIFERAY_HOME/tomcat/logs/catalina.out*** (Console du serveur d'application).

# Modifier la verbosité des logs

Deux méthodes :

1) Via Control Panel

***System Settings → Logging***

Créer un logger :

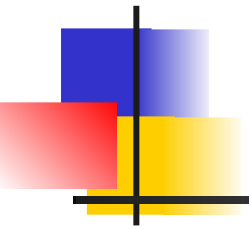
Category: *com.liferay.journal.service*

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, NoSuchLayoutExceptio*n…
- 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

2 sessions dans un contexte Liferay :

- **Session HTTP** standard géré par le conteneur (Tomcat)
- **Session Portail** géré par Liferay contenant les préférences, l'état des portlets, les permissions en cache)

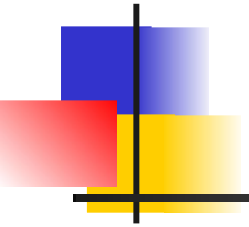
Si la session HTTP expire, la session Portal est détruite.

Stockage en mémoire (par défaut), nécessite synchronisation si cluster

2 Paramètres clés :

- ***session.timeout=30***
  Passé ce délai, l'utilisateur est déconnecté automatiquement.
  Attention il doit être inférieur à la session de tomcat définie dans *web.xml*
  Liferay l'utilise pour afficher l'avertissement à l'utilisateur

- ***session.timeout.auto.extend=true***
  Option de confort. Si l'utilisateur a une page Liferay ouverte, la session est étendue via du code Javascript



# 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

# *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

***Control Panel > Configuration > System Settings***

UI permettant de configurer certains module OSGI :

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

➡ Modifiable sans restart, stocké dans la table ***configuration_*** ou via fichiers ***.config***

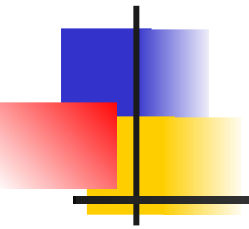
# Configuration OSGi (.config)

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
```



# Configuration effective

- Les configuration OSGI peuvent donc se trouver à plusieurs endroits
- Priorité 1 (Fichiers .config) : Si un administrateur dépose un fichier .config ou .cfg dans le dossier osgi/configs, il est prioritaire.
- Priorité 2 (Base de données) : Si la configuration est faite via l'interface "System Settings", elle est stockée dans la table Configuration_.
- Au runtime : Le moteur OSGi de Liferay surveille ces deux sources et "injecte" les modifications dans les modules concernés.

# Optimisation & Maintenance

Gestion des modules (Prudence !) :

- Prioriser le Blacklisting plutôt que la désinstallation. (Voir + loin)
- Nettoyer les modules Marketplace et Custom inutilisés.

Règle d'or : Ne jamais toucher aux modules com.liferay.portal.* sans audit d'impact.

Diagnostic Santé (Health Check) :

- Gogo Shell : scr:list pour repérer les "unsatisfied components".
- JDBC : Vérifier la taille du pool de connexion (éviter la saturation au boot).

Infrastructure :

- DNS local / fichier hosts pour éviter les latences de résolution. (BD, ElasticSearch, Liferay lui-même)
- Vérifier que l'index de recherche (Elasticsearch) est bien sur un serveur séparé.

# Optimiser la recherche (Elasticsearch)

Options clés :

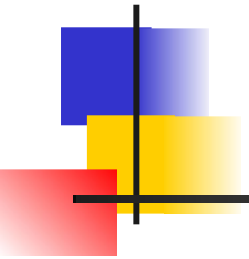
- Externaliser Elasticsearch
- Multi-Node pour ressource critique
- Activer le monitoring Kibana
- Régénérer les index régulièrement :

***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

Configuration logs, sessions, OSGI

**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

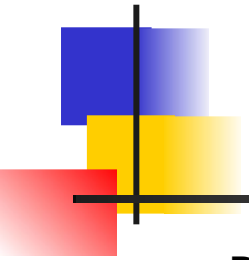
```
@Reference
private UserLocalService _userLocalService;
```

- 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...");
}
```

# Fichier .bnd

- Ce fichier donne les instructions à l'outil ***Bnd*** (utilisé par Gradle/Maven) pour générer le MANIFEST.MF du JAR
  - Instructions fondamentales :
    Bundle-Name, Bundle-SymbolicName, Bundle-Version,
    Export-Package, Import-Package
  - Instructions spécifiques Liferay :
    Web-ContextPath (Chemin des ressources statiques web), -metatype: * (Classe de Configuration)



# Exemple

```
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

# Configuration des modules

- Liferay n'utilise plus de fichiers .properties mais des interfaces Java pour configurer les modules.
- Cela permet à Liferay de générer automatiquement une interface de configuration dans son UI d'administration

```
@ExtendedObjectClassDefinition(
    category = "communication",
    scope = Status.Scope.SYSTEM  // Menu Système Settings
)
@Meta.OCD(id = "com.projet.NewsletterConfig", name = "Newsletter Settings")
public interface NewsletterConfiguration {


    @Meta.AD(deflt = "smtp.mon-entreprise.com", name = "Serveur SMTP")
    public String smtpServer();


    @Meta.AD(deflt = "true", name = "Activer l'envoi")
    public boolean enabled();

    @Meta.AD(deflt = "100", name = "Nombre max d'emails/heure")
    public int maxEmails();
}
```

# Exemple de composant configuré

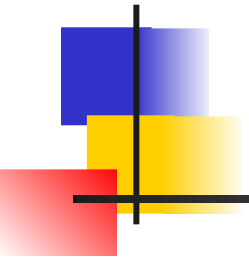
```
@Component(
    configurationPid = "com.projet.NewsletterConfig", // <--- Le lien avec l'OCD
    immediate = true,
    service = NewsletterService.class
)
public class NewsletterServiceImpl implements NewsletterService {

    private volatile NewsletterConfiguration _configuration;

    // Appelée au démarrage ou dès que l'Admin change une valeur dans System Settings
    @Activate
    @Modified
    protected void activate(Map<String, Object> properties) {

        _configuration = ConfigurableUtil.createConfigurable(
            NewsletterConfiguration.class, properties);

        System.out.println("Configuration Newsletter mise à jour ! SMTP : " +
_configuration.smtpServer());
    }

    @Override
    public void sendEmail(String message) {
        if (_configuration.enabled()) {
            // Logique d'envoi utilisant _configuration.smtpServer()
        }
    }
}
```

# 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 et leurs statuts
  - ***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
  - ***diag [ID]*** : Indique quel est le package java manquant
  - ***scr*** : Les composants OSGI et leur statut ENABLED ou UNSATISFIED
  - ***services*** : Liste des services exposés par les bundles, (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 : *org.apache.felix.scr, com.liferay.portal.osgi*
- Essayer d’arrêter et redémarrer le module



# Administration & Exploitation

Base de données

Configuration logs, sessions, OSGI

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 :

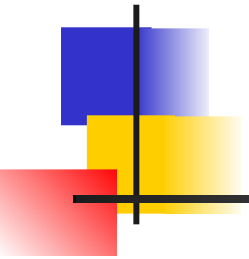
- API obsolètes remplacées
- Service Builder régénéré
- REST Builder mis à jour
- OSGi : vérifier bnd.bnd
- Clay / React : mise à jour front
- Thème recompilé avec nouvelle base Clay

# 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

Configuration logs, sessions, OSGI

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

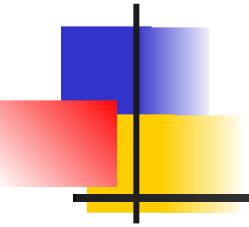
Les paramètres suivants ne doivent jamais être codés en dur :

- 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

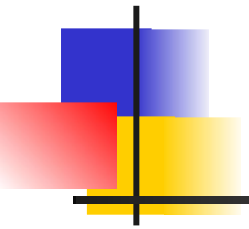
2. Variables d’environnement

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.

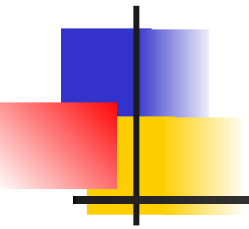
Caractéristiques :

- 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.

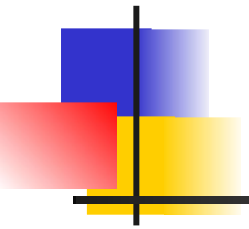
Avantages :

- 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

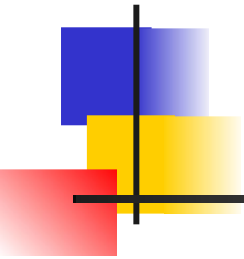
1) Création dans l’environnement de staging
2) Prévisualisation
3) Validation (workflow selon paramétrage)
4) Publication vers live
5) Contenu immédiatement disponible

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

Taglibs

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,
  - Blade : Générateur de projet et de modules OSGI basé sur Maven ou Gradle
  - un IDE (Eclipse / IntelliJ / VSCode ).
  - Des plugins IDE pour Liferay

# Configuration développeur

- Activation (portal-ext.properties) :

*include-and-override=portal-developer.properties*

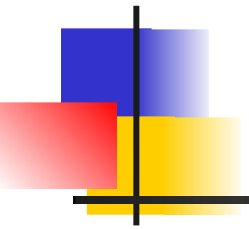
*module.framework.properties.osgi.console=11311*

Avantages :

- Affichage des templates
- Réduction du cache
- Logs plus explicites
- Gogoshell accessible par telnet ou blade via le port 11311

# Mise en place de l'environnement de développement

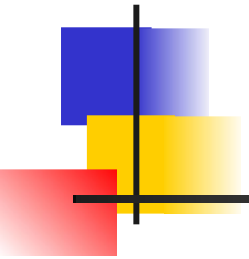
- Installer ***Blade CLI*** sur le poste de développement.
  - Créer un **Liferay Workspace** dans un répertoire de travail :
    ***`blade init liferay-workspace`***
    - Initialisation Gradle ou Maven
    - Configuration de la version produit
- Ce workspace contiendra les modules (apps, thèmes, services,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

***Atelier 8 : IDE et Premier portlet***



# Développement

Mise en place IDE

**Développement Portlet**

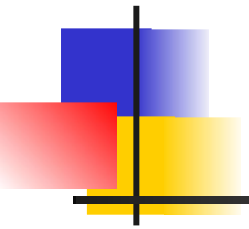
Taglibs

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*.

# 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…
```

# 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

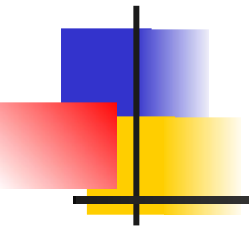
La spécification portlet JSR-286 (Portlet 2.0), toujours respectée par Liferay définit :

- Des modes fonctionnels du portlet
  - ***view*** : Affichage, modèle par défaut
  - ***edit*** : Configuration, peu utilisé au détriment de la configuration OSGI s'affichant dans le menu configuration du portlet
  - ***help*** : aide (quasiment jamais implémenté)

- Des états pour la fenêtre englobant le portlet dépend énormément du thème utilisé
  - ***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 command permettant une séparation claire, impliquant 3 classes :
  - Action = *MVCActionCommand*
  - Render = *MVCRenderCommand*
  - Resource = *MVCResourceCommand*



# Le Modèle MVC "Standard"

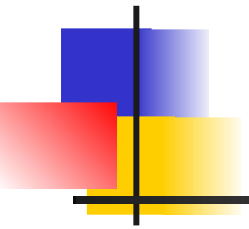
- Le portlet hérite de la classe MVCPortlet.
- Toute la logique est regroupée dans une seule classe Java.
  - La méthode ***processAction*** (ou de propres méthodes *actionName*) gèrent les changements d'état (écriture en base).
  - La méthode ***render*** prépare les données pour la vue.
  - ***serveResource*** gère les appels AJAX ou la génération de fichiers.
- Idéal pour : Les portlets très simples avec 1 ou 2 vues maximum.

# Exemple MVCPortlet minimal

```
@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 {


}
```

# Implémentation méthodes standards

```
@Override
public void processAction(ActionRequest req, ActionResponse res) {
  String action = ParamUtil.getString(req, "javax.portlet.action");
  if (action.equals("addEvent")) { ... }
  else if (action.equals("deleteEvent")) { ... }
}


@Override
public void render(RenderRequest renderRequest, RenderResponse renderResponse)
    throws IOException, PortletException {

    // 1. Récupération via méthode métier
    List<String> events = service.getEvents();
    // 2. Positionnement des attributs
    renderRequest.setAttribute("eventList", events);
    int totalEvents = events.size();
    renderRequest.setAttribute("hasEvents", totalEvents > 0);

    // 4. Appel au moteur de rendu Liferay
    super.render(renderRequest, renderResponse);
}
}
```



# Pattern MVC Command

- Principes : Éclater la logique en petites classes spécialisées (les Commands) enregistrées via OSGi.
- 3 types de Commands :
  - ***MVCActionCommand*** : Gère une action spécifique (ex: *AddEventActionCommand*).
  - ***MVCRenderCommand*** : Gère l'affichage d'une vue spécifique (ex: *ViewDetailsRenderCommand*).
  - ***MVCResourceCommand*** : Gère une ressource spécifique (ex: *ExportExcelResourceCommand*).
- Avantage OSGi : On peut surcharger ou remplacer une seule action d'un portlet existant sans toucher au reste du code.

# *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 (Côté JSP)

- La navigation repose sur des URLs de portlet générées par des tags spécifiques.
- Changer de vue (Render) :

```
<portlet:renderURL var="detailsURL">
    <portlet:param name="mvcRenderCommandName"
value="/event/details" />
    <portlet:param name="eventId" value="${event.id}" />
</portlet:renderURL>
<a href="${detailsURL}">Voir les détails</a>
```

- Déclencher une action (Action) :

```
<portlet:actionURL name="/event/add" var="addEventURL" />
<aui:form action="${addEventURL}" method="post"> ...
</aui:form>
```

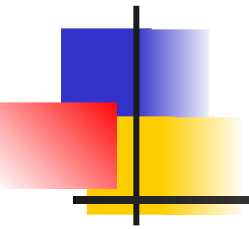
# Navigation (Côté Java)

- L'aiguillage est assuré par la propriété ***mvc.command.name*** déclarée dans l'annotation @Component des classes Command.

```
@Component(
    property = {
        "javax.portlet.name=my_portlet_ID",
        "mvc.command.name=/event/details" // <--- La clé de navigation
    },
    service = MVCRenderCommand.class
)
public class ViewDetailsRenderCommand implements MVCRenderCommand { ... }
```

Règle Liferay :

- Si un paramètre ***mvcRenderCommandName*** est présent dans l'URL, Liferay cherche la classe correspondante.
- Si le paramètre est absent il se rabat sur mvcPath qui peut indiquer directement la vue JSP
- Sinon, il se replie sur le mvc-path par défaut (souvent view.jsp).



# Navigation entre vues via *mvcPath*

Dans une JSP :

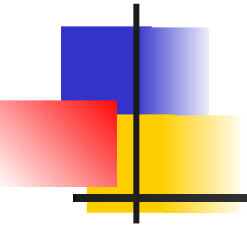
```
<portlet:renderURL var="editURL">
    <portlet:param name="mvcPath" value="/edit.jsp"/>
</portlet:renderURL>
```

Dans une *ActionCommand* :

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

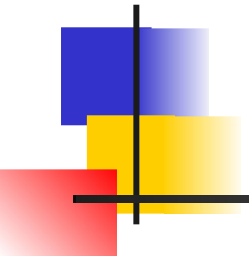
# Validation : concepts Liferay

- 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



# Développement

Mise en place IDE

Développement Portlet

**Taglibs**

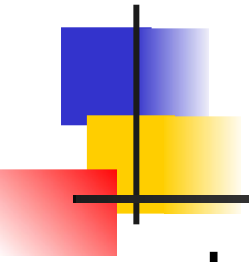
Service Builder

REST Builder et Client Extensions

Sécurité

Extensions : Asset, Thèmes, OSGi

# Introduction

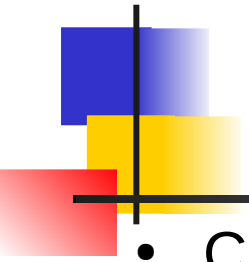


- Liferay a progressivement remplacé son framework historique par des standards plus ouverts.
  - **Liferay Util (AUI)** : Historiquement basé sur YUI (Yahoo User Interface). En maintenance, mais encore très présent dans les anciens projets.
  - **Clay UI (Le présent)** : C'est l'implémentation de Liferay du design system Lexicon.
    Basé sur Bootstrap 4, il est plus léger et "responsive".
  - **Taglibs Standards** : Liferay utilise toujours les JSTL (c:, fmt:) et les portlet tags (portlet:).

# Les bases de AlloyUI

- ***<aui:form>*** : Gère automatiquement l'encapsulation des namespaces (évite les conflits d'ID entre portlets).
- ***<aui:input>*** : Génère le label, l'ID unique, et gère les types (text, checkbox, hidden).
- ***<aui:button-row>*** : Aligne proprement les boutons en bas de formulaire.

```
<aui:form action="${actionURL}" name="fm">
    <aui:input name="userName" label="Nom d'utilisateur" />
    <aui:button-row>
        <aui:button type="submit" value="Enregistrer" />
    </aui:button-row>
</aui:form>
```



# Clay UI

- Clay UI est qu'une couche graphique (Lexicon) appliquée sur les composants Bootstrap.
  - Toutes les classes de Bootstrap4 peuvent être utilisées dans les JSP Liferay
    - Grille responsive immédiate.
    - Utilitaires de marges et d'espacement (spacing).
    - Composants familiers pour les développeurs Front-end.
- Il permet également d'afficher des composants complexes avec très peu de code en passant des objets Java respectant des interfaces.
  - ***<clay:management-toolbar>*** : La barre d'outils standard avec recherche, filtre et tri.
  - ***<clay:navigation-bar>*** : Pour créer des onglets ou des menus de navigation.
  - ***<clay:alert>*** : Pour les messages de succès ou d'erreur stylisés Lexicon.

# Exemple classes Bootstrap

```
<div class="container-fluid-1230">
   <div class="sheet">
      <div class="sheet-header">
         <h2 class="sheet-title">${event == null ? 'Créer un événement' : 'Modifier l\'événement'}</h2>
      </div>

      <aui:form action="${saveEventURL}" name="fm">
         <aui:model-context bean="${event}" model="<%= Event.class %>" />
         <div class="sheet-item">
            <div class="row">
               <div class="col-md-8">
                  <aui:input name="title" label="Titre de l'événement" placeholder="Ex: Conférence Java" />
               </div>
               <div class="col-md-4">  <aui:input name="date" label="Date" type="date" /> </div>
            </div>
            <aui:input name="location" label="Lieu"> <aui:validator name="required" /> </aui:input>
         </div>
         <div class="sheet-footer">
            <aui:button-row>
               <aui:button type="submit" value="Enregistrer" cssClass="btn-primary" />
               <aui:button type="cancel" onClick="${viewURL}" cssClass="btn-secondary" />
            </aui:button-row>
         </div>
      </aui:form>
   </div>
</div>
```

# Les Objets Java pour ClayUI

- Les composants dynamiques de Clay attendent des objets qui implémentent des interfaces  du package ***com.liferay.frontend.taglib.clay.***

- Le taglib Clay se charge de les transformer en JSON pour que le composant React/Javascript de Liferay les affiche

- Les 3 objets majeurs :
  - ***CreationMenu*** : Définit le bouton "+" (Ajouter).
  - ***DropdownItem*** : Définit les éléments d'un menu déroulant ou d'actions.
  - ***InfoPanel*** : Définit les panneaux de détails latéraux.

# Exemple *DropdownItem*

- **RenderCommand**

```
List<DropdownItem> dropdownItems = new ArrayList<>();

// Option Modifier
DropdownItem editItem = new DropdownItem();
editItem.setHref(renderResponse.createRenderURL(), "mvcPath", "/edit.jsp", "eventId", eventId);
editItem.setLabel("Modifier");
editItem.setIcon("pencil");
dropdownItems.add(editItem);

// Option Supprimer
DropdownItem deleteItem = new DropdownItem();
deleteItem.setHref(renderResponse.createActionURL(), "mvcActionCommandName", "/event/delete");
deleteItem.setLabel("Supprimer");
deleteItem.setIcon("trash");
dropdownItems.add(deleteItem);

renderRequest.setAttribute("eventActionItems", dropdownItems);
```

- **JSP**

```
<clay:dropdown-actions
    items="${eventActionItems}"
/>
```

# Affichage de données – Search Container

- Pour afficher des listes ***liferay-ui:search-container***.

Il est moderne, supporte la pagination et le tri.

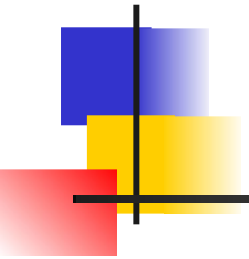
```
<liferay-ui:search-container total="${eventCount}" delta="10">
  <liferay-ui:search-container-results results="${eventList}" />
  <liferay-ui:search-container-row className="org.formation.model.Event"
modelVar="event">
    <liferay-ui:search-container-column-text name="Nom" value="$
{event.name}" />
     <liferay-ui:search-container-column-text name="Date" value="${event.date}"
/>
  </liferay-ui:search-container-row>
  <liferay-ui:search-iterator />
</liferay-ui:search-container>
```

# Tag et définition d’objets

- Certains tags sont quasiment tout le temps présent. Leur rôle est d'injecter des variables Java directement dans le contexte de la page.
- **<portlet:defineObjects />** : Spécification Portlet.
  - Objets injectés : renderRequest, renderResponse, portletConfig, portletPreferences.
  - Indispensable pour créer des URLs (renderURL, actionURL) ou lire les préférences du portlet dans la JSP.
- **<liferay-theme:defineObjects />**
  - Objets injectés : *themeDisplay*, user, layout (la page actuelle), scopeGroupId.
  - themeDisplay. (le couteau suisse) Permet de récupérer le logo, le nom du site, vérifier si l'utilisateur est connecté, ou obtenir l'URL du portail.
- **<liferay-frontend:defineObjects />** : Apparu avec Liferay 7, il complète les deux précédents pour le développement moderne.
  - Principalement des utilitaires pour les nouveaux composants (comme les contextes d'affichage).
  - Souvent nécessaire pour que les *taglibs clay:* ou *liferay-frontend:*

# Taglib *liferay-frontend*

- Cette bibliothèque fournit des composants de structure qui "habillent" les formulaires et les pages au look Liferay.
- Les composants majeurs :
  - ***<liferay-frontend:edit-form>*** : Une version améliorée de <aui:form> optimisée pour les écrans de configuration.
  - ***<liferay-frontend:fieldset-group>*** : Regroupe des ensembles de champs dans un panneau rétractable ou stylisé.
  - ***<liferay-frontend:sidebar-panel>*** : Crée des panneaux latéraux coulissants.



# Développement

Mise en place IDE

Développement Portlet

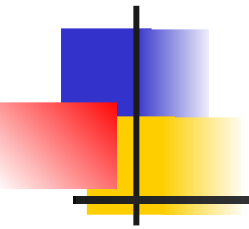
Taglibs

**Service Builder**

REST Builder et Client Extensions

Sécurité

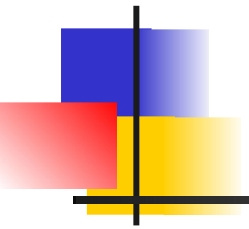
Extensions : Asset, Thèmes, OSGi



# Objectifs

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

# Etapes de mise en place



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

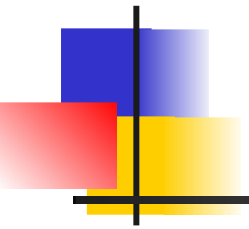
2) Exécuter ***gradlew buildService***

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

4) Appeler le service depuis les classes portlets ou les 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>
```

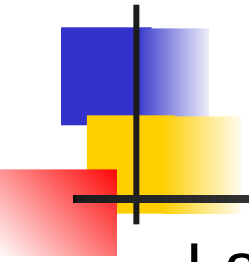


# Tables générées

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>*** : getter/setter
  - Service :
    - ***<Entity>LocalService*** : Les méthodes exposées et utilisables par les autres modules
    - ***<Entity>Persistence*** : Interface pour gérer la persistance de l'entité utilisée par l'implémentation du service

# Implémentation du Service local

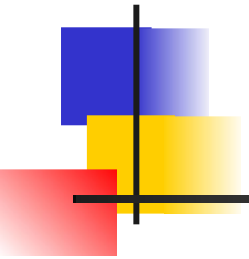
Dans le module service, *ServiceBuilder* génère l'implémentation ***<Entity>LocalServiceImpl*** :

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

- Le code gère automatiquement :
  - Transactions
  - Injections OSGi des services cœur de Liferay

# Exemple

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

    // Service coeur de Liferay permettant de trouver un nouvel ID
    long eventId = counterLocalService.increment(Event.class.getName());

    // Interface autorisation les opérations persistence
    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

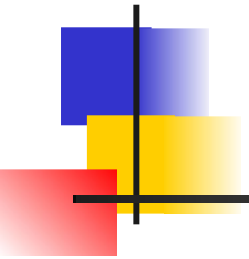
- 2 alternatives :
  - Utiliser directement le service en **local**.
    Le module portlet définit une dépendance sur l'api et se fait injecter le service
  - Accéder au service en mode remote :
    - **SOAP** = héritage, non recommandé
    - **JSON Web Services/JAX-RS**, éventuellement à désactiver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

# Intégration portlet du serviceLocal

- Ajouter la dépendance vers le module 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

Taglibs

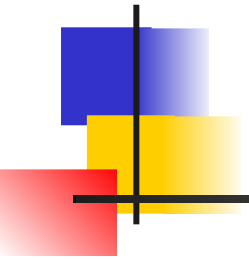
Service Builder

**REST Builder et Client Extensions**

Sécurité

Extensions : Asset, Thèmes, OSGi

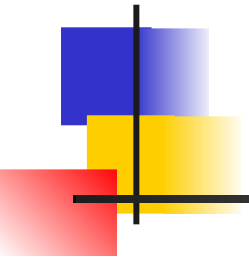
# RestBuilder



- 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 les entités de 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 du mapping entre classe du modèle et DTO

# 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);
}
```

# Intégration dans Liferay

- Après un déploiement avec *./gradlew deploy*, le service est intégré dans l'écosystème Liferay qui intègre nativement Swagger UI ***Menu Global → Applications → API Explorer***.
  - Visualisation de tous les endpoints déployés (natifs et personnalisés).
  - Test des requêtes (GET, POST, etc.) avec authentification automatique de la session.
  - Génération de jetons d'authentification pour les tests.

- URLs Utiles
  - Swagger UI : ***http://localhost:8080/o/api***
  - Définition OpenAPI (JSON) : ***http://localhost:8080/o/openapi/v1.0/openapi.json***
  - Liste des schémas REST : ***http://localhost:8080/o/rest-openapi/v1.0/openapi.json***

***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 et on spécifie :
    - ***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.

# Exemple WebComponent

```
class MyEventWidget extends HTMLElement {
  connectedCallback() {
    // Le rendu HTML est injecté ici
    this.innerHTML = `
      <div class="card">
        <div class="card-body">
          <h5>Mes Événements</h5>
          <button id="btn" class="btn btn-primary">Rafraîchir</button>
        </div>
      </div>`;

    // La logique (ex: clic) est encapsulée
    this.querySelector('#btn').addEventListener('click', () => {
      alert('Appel API Liferay imminent !');
    });
  }
}

// Enregistrement du tag HTML <my-event-list>
customElements.define('my-event-list', MyEventWidget);
```

# Définition .yaml

```
# L'identifiant de l’extension dans Liferay
my-event-widget:
    name: Liste des Événements
    type: customElement
    # Le tag que le navigateur va instancier
    htmlElementName: my-event-list
    # Le chemin vers le fichier JS compilé
    url: index.js
    # Les fichiers CSS associés
    cssURLs:
        - main.css
```

# Communication Liferay et Authentification

- Cycle de vie :
  - L'utilisateur charge la page.
  - Le composant JS est monté.
  - Le composant appelle l'API REST en utilisant ***fetch()*** vers le endpoint Liferay.
  - Les données sont affichées dans le widget.

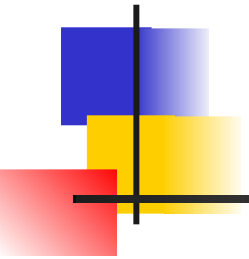
- Sécurité :
  - Si application est sur le même domaine, elle bénéficie des cookies de session.
  - Sinon il faut implémenter CORS + un flux oAuth2

- Contexte : Liferay fournit l'objet global ***Liferay*** qui contient :
  - ***themeDisplay :*** Qui permet d'accéder à *groupId, companyId,* etc..
  - ***authToken*** pour sécuriser les requêtes contre les failles CSRF qui doit être utilisé dans l'entête ***x-csrf-token***

# 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

Taglibs

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** ?*
- Une permission combine :
  - Des **rôles** affectés à un utilisateur : Qui
  - des **actions** (VIEW, UPDATE, DELETE…) : Quoi
  - Des **ressources** : Sur Quoi
- Le même moteur de permissions est utilisé pour :
  - les pages
  - les contenus
  - les portlets
  - les APIs REST

# 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.

# 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
    - Filtrer des résultats

# 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>
             < !-- permissions par défaut, lors de l'installation du module -->
            <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>    <!-- Chaque ligne de la table Event est une resource sécurisable -->
    <model-name>com.acme.events.model.Event</model-name>
    <portlet-ref>    <!-- Permissions gérées par le portlet -->
        <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>   <!-- Actions que ne pourra jamais avoir guest -->
            <action-key>UPDATE</action-key>
            <action-key>DELETE</action-key>
            <action-key>REGISTER</action-key>
        </guest-unsupported>
        <owner-defaults> <!-- Le créateur de l’entit » -->
            <action-key>VIEW</action-key>
            <action-key>UPDATE</action-key>
            <action-key>DELETE</action-key>
        </owner-defaults>
        <action-keys>  <!-- Les actions configurables -->
            <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>
```

# *ModelResourcePermission*

- Pour chaque modèle métier (Event, Article, etc.), un ***ModelResourcePermission*** centralise :
  - les règles
  - la notion de propriétaire
  - les rôles
  - les règles personnalisées
- Il permet à Liferay de faire le lien entre un ID d'événement et le moteur de permissions.
- Toute vérification passe par lui :
  - UI
  - Service
  - REST / GraphQL

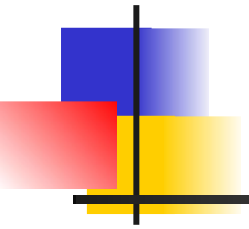
# Enregistrement de *ModelResourcePermission*

- 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

# Exemple

```
@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
_serviceRegistration = 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;
}
```

# Responsabilité de la couche service



- La couche service doit préparer les données pour qu'elles soient compatibles avec le moteur de sécurité.
- Lors de l'ajout d’une entité, il doit enregistrer l'objet dans le système de ressources. resourceLocalService.addModelResources( entity, serviceContext);

# 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

# Exemple Ajout de ressource

```
public Event addEvent(long userId, long groupId, String title,
ServiceContext serviceContext) {

    // 1. Persistance
    long eventId = counterLocalService.increment(Event.class.getName());
    Event event = eventPersistence.create(eventId);
    event.setTitle(title);
    eventPersistence.update(event);


    // 2. Enregistrement dans le moteur de permissions de Liferay
    resourceLocalService.addModelResources(event, serviceContext);


    return event;
}
```

# La couche "Interface" (Le Contrôle d'Accès)

- La couche d’interface a la responsabilité de  vérifier les permissions avant d'autoriser l'accès au service.
  - Si on est en création, elle doit vérifier le droit que l’utilisateur a le droit de créer en s’apputant sur PermissionChecker
  - Si on est sur une mise à jour ou une suppression, elle utilise ModelResourcePermission.
- Cas de l'API REST (ResourceImpl) :
  - On injecte le vérificateur via @Reference.
  - Action : Avant chaque opération, on appelle .check().

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

_eventLocalService.updateEvent(...);
```

# Exemple : Droit de création dans un portlet

```
public void addEvent(ActionRequest actionRequest, ActionResponse
actionResponse)
    throws Exception {

    ThemeDisplay themeDisplay = (ThemeDisplay)
actionRequest.getAttribute(WebKeys.THEME_DISPLAY);
    PermissionChecker permissionChecker = themeDisplay.getPermissionChecker();

    // On vérifie le droit "ADD_EVENT" sur la ressource Portlet du site actuel
    boolean hasAddPermission = permissionChecker.hasPermission(
        themeDisplay.getScopeGroupId(),
        "com.acme.events", // Le nom de la ressource défini dans default.xml
        themeDisplay.getScopeGroupId(),
        "ADD_EVENT"
    );

    if (!hasAddPermission) {
        SessionErrors.add(actionRequest, "permission-error");
        return; // On arrête l'exécution
    }

    // Si OK, on appelle le service
    _eventLocalService.addEvent(...);
}
```

# Exemple : Droit d’update dans Endpoint REST

```
@Override
public Event putEvent(Long eventId, Event event) throws Exception {

    // 1. Récupèration PermissionChecker de l'utilisateur
PermissionChecker permissionChecker =
PermissionThreadLocal.getPermissionChecker();

    // 2. Vérification de droit UPDATE sur l'ID
    _eventModelResourcePermission.check(
        permissionChecker,
        eventId,
        ActionKeys.UPDATE
    );

    // 3. Si aucune exception, mise à jour
    return _eventLocalService.updateEvent(eventId, event.getTitle());
}

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

# Construction de l'UI dans les portlets

- 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

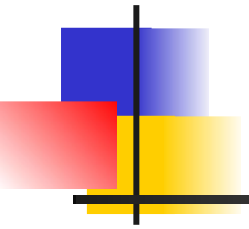
# Client Extension

- Dans le cas d’une Client Extension, le frontend ne peut pas "deviner" les permissions.
- La bonne pratique est que L'API REST doit renvoyer, pour chaque objet, la liste des actions permises (ex: actions: ["view", "update"]).
- Le Code JS peut alors effectuer des test pour masquer des boutons :

```
if (event.actions.includes('update')) {
    showEditButton();
}
```

***Atelier 13 : Permissions et ServiceContext***

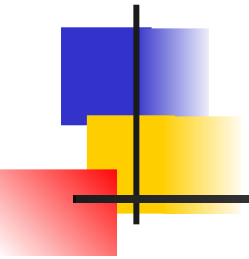
# Mise en place OAuth2 dans Liferay



- 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

Taglibs

Service 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
);
```

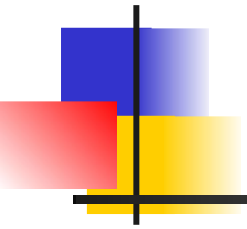
- Permet la publication via les widgets standard Liferay. (Asset Publisher)

# 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***

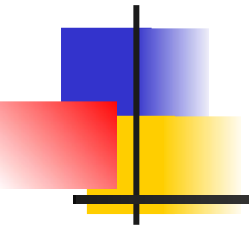


# Thèmes Liferay

## 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

- Alternative : Liferay Objects & Client Extensions



# 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*

# Principes

- Création de module via blade :
  blade create -t theme mon-theme
- Par défaut, le thème hérite du thème ***_styled***. Il est "vide" car il utilise les fichiers parents. On ne crée que ce que l'on veut modifier.

- Arborescence Standard :
  - ***src/main/webapp/css/_custom.scss*** : Pour vos styles.
  - ***src/main/webapp/templates/*** : Pour vos structures Freemarker (portal_normal.ftl).
  - ***WEB-INF/liferay-look-and-feel.xml*** : Définit le nom et les paramètres du thème.

# Personnalisation (SCSS & FTL)

- Design avec SCSS : Le fichier ***_custom.scss***.
  Liferay compile automatiquement vos fichiers SASS en CSS.
  On peut écraser les variables de base de Liferay (comme les couleurs de la charte Clay/Bootstrap).
- Structure avec Freemarker (.ftl) :
  - Le fichier ***portal_normal.ftl*** est le cœur du thème. Il contient le header, le footer et la zone de contenu.
    Insertion des des éléments globaux (bannières, scripts de tracking, logos spécifiques).
  - Variables Liferay : Utilisez les objets fournis par le moteur (ex: ${is_signed_in}, ${site_name}, ${the_title}) pour rendre le thème dynamique.

***Atelier 14.2 : Thème***

# OSGi vs Hooks et Ext

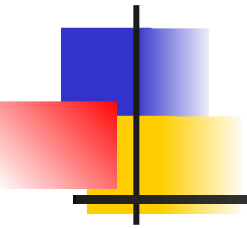
Hooks & EXT (Liferay 6.x / 6.2) :

- 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***