**Spécification technique – Socle applicatif**

**support d’une solution open source d’ENT pour les EPLE**

**de la région Île-de-France**

**Spécification Technique   
Socle applicatif**

**Auteur** :

**Version** : 1.0



**Gestion des changements de version**

*Ce tableau gère les modifications apportées au document au-delà de sa version initiale. Les petites modifications de type erreurs de frappe ou changements de syntaxe ne font pas l’objet d’un suivi. Toute nouvelle version du document ne conserve pas systématiquement les changements apportés lors de la version précédente.*

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Auteur** | **Objet de la mise à jour** |
| 1.0 | 04/03/2011 | MMAU | Initialisation du document |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Sommaire

[Objet du document 5](#_Toc299375113)

[1. Fonctionnalités proposées par le socle 5](#_Toc299375114)

[1.1. Qu’est-ce que le socle ? 5](#_Toc299375115)

[1.2. Sécurité 5](#_Toc299375116)

[1.3. Autour de l’annuaire 6](#_Toc299375117)

[1.4. Console d’administration de l’ENT 7](#_Toc299375118)

[1.5. Socles techniques 7](#_Toc299375119)

[1.6. Utilitaires divers pour les modules 8](#_Toc299375120)

[1.7. Vue d’ensemble 8](#_Toc299375121)

[2. Découpage technique et fonctionnement du socle 9](#_Toc299375122)

[2.1. Organisation générale 9](#_Toc299375123)

[2.2. CAS 10](#_Toc299375124)

[2.3. Batchs 11](#_Toc299375125)

[2.3.1. Description des batchs 11](#_Toc299375126)

[2.3.2. Packaging et déploiement 11](#_Toc299375127)

[2.3.3. Fonctionnement général 12](#_Toc299375128)

[2.4. APIs 12](#_Toc299375129)

[2.4.1. Description des apis 12](#_Toc299375130)

[2.4.2. Packaging et déploiement 13](#_Toc299375131)

[2.4.3. Utilisation des apis par un module 13](#_Toc299375132)

[2.4.4. Configuration interne des apis 15](#_Toc299375133)

[2.5. Frameworks core 17](#_Toc299375134)

[2.6. Customisation de serveurs 19](#_Toc299375135)

[2.6.1. Portail 19](#_Toc299375136)

[2.6.2. Solr 19](#_Toc299375137)

[2.6.3. Jabber 19](#_Toc299375138)

[2.7. Web Services 20](#_Toc299375139)

[2.8. Console d’administration 21](#_Toc299375140)

[2.8.1. Rôle de la console 21](#_Toc299375141)

[2.8.2. Ajout d’un nouveau module 21](#_Toc299375142)

[2.8.3. Fonctionnement général 22](#_Toc299375143)

[2.9. Projet modèle pour création d’une webapp 22](#_Toc299375144)

[2.9.1. Principe 22](#_Toc299375145)

[2.9.2. Utilisation pour création d’un module 22](#_Toc299375146)

[2.9.3. Modification de l’archetype 23](#_Toc299375147)

[2.10. Tableau de correspondance des fonctionnalités 23](#_Toc299375148)

[2.11. Interactions et flux entre les projets 25](#_Toc299375149)

# Objet du document

Le but de ce document est de décrire le socle applicatif de l’ENT selon ces trois aspects :

* le contenu et les services fournis
* le fonctionnement technique sous-jacent
* l’utilisation du socle par les modules

L’utilisation et les configurations communes à différentes parties du socle (utilisation générale des apis par exemple) sont décrites dans ce document. Par contre, le détail de chaque projet, quand nécessaire, fait l’objet d’une documentation technique spécifique supplémentaire.

# Fonctionnalités proposées par le socle

## Qu’est-ce que le socle ?

Le socle se veut être une base stable commune à tous les modules de l’ENT :

* Il permet de fournir un certain nombre de fonctionnalités et choix techniques, sous la forme d’un ensemble de projets (librairies, webapps, batchs…) avec chacun sa spécialité.
* Cela assure une centralisation des règles métier communes et incite à garder une cohérence à la fois fonctionnelle et technique entre tous les modules.
* Il simplifie également le développement d’un module, car celui-ci peut directement réutiliser les outils proposés.

## Sécurité

Plusieurs aspects de sécurité sont fournis par le socle. L’utilisation de l’outil CAS permet de gérer :

* L’authentification/connexion
* Le SSO (Single-Sign-On) avec les différents services externes reliés à l’ENT
* La sécurisation des droits d’accès dans le portail

D’autre part, les flux et accès aux données sont également sécurisés par d’autres moyens gérés dans le socle :

* Utilitaire de sécurisation des urls http pour appeler les parties serveur (accès aux données) en fonction des droits de la personne connectée
* Utilitaire de cryptage MD5 et AES pour les différents besoins de stockage de mots de passe

## Autour de l’annuaire



La gestion des utilisateurs et groupes dans l’ENT est basé sur l’annuaire fédérateur fourni par les académies. Ces données sont importées dans une base de données du socle. Elles sont adaptées et réorganisées afin d’être utilisées par les différents services. Les services proposés par le socle au sujet de l’annuaire sont donc :

* Alimentation à partir des fichiers xml de l’annuaire fédérateur
* Synchronisation des données vers le portail
* Encapsulation des appels au portail pour gérer les accès et pages des personnes
* Services d’accès aux données de l’annuaire
* Règles de gestion métier autour des données de l’annuaire
* Centralisation des droits des personnes sur l’annuaire (en fonction des groupes et profils)
* Identifiants de connexion des utilisateurs

## Console d’administration de l’ENT

La console d’administration permet de gérer le paramétrage et l’activation des différents modules par les administrateurs des établissements. Celle-ci est considérée comme faisant partie du socle dans le sens où elle centralise un certain nombre d’informations utiles à tous les modules. Elle contient notamment les fonctionnalités suivantes :

* Visualisation/export des données de l’annuaire
* Création de groupes supplémentaires
* Gestion de comptes supplémenatires comme les comptes invités et administrateurs
* Activation/désactivation des modules
* Paramétrage et droits d'accès aux modules selon les profils
* Remontée d'incidents
* Services d'accès à ces données de paramétrage utilisables par les modules

## Socles techniques

Le socle est également un socle technique proposant des encapsulations de frameworks techniques et des traitements centralisés permettant de :

* garder une cohérence technique entre les modules
* faciliter les développements

Il contient les aspects suivants :

* socles d’abstraction des frameworks et classes de référence pour les différentes couches : persistance Ibatis, métier, contrôleur/vue (Struts2, GWT, PureMVC)
* gestion des exceptions
* gestion des fichiers de configuration
* gestion du principe de conversations pour les données en session
* cache métier
* gestion des logs
* centralisation des feuilles de style ENT
* template de génération d’un projet module

## Utilitaires divers pour les modules

Le socle propose également des outils plus fonctionnels à utiliser par les modules qui ont un besoin fonctionnel similaire. Dans le cas d’un besoin fonctionnel correspondant à ce que fournit l’un de ces outils, il est préférable - pour des raisons d’homogénéité, de maintenance évolutive, et de simplicité - d’utiliser ce dernier plutôt que de refaire un traitement similaire en doublon dans chaque module.

Les fonctionnalités sont les suivantes :

* Récupération des informations de l’utilisateur connecté
* Moteur de recherche Solr et api associée (indexation + recherche)
* Gestion des droits sur les entités et contenu des modules (traitements métier + interface avec onglet commun aux modules concernés)
* Gestion des échanges de données entre modules (flux REST)
* Gestion des fichiers uploadés sur le serveur applicatif
* Utilitaire de cryptage MD5 et AES
* Envoi d’emails
* Api d’interfaçage avec Jira
* Customisation d’un serveur Jabber par rapport à l’annuaire ENT

## Vue d’ensemble



# Découpage technique et fonctionnement du socle

## Organisation générale

Techniquement, le socle se découpe en plusieurs projets de différents types. On retrouve des batchs, des APIs, des frameworks-socles et divers projets de customisation. Selon leur type, ils sont utilisés et déployés différemment pour les modules.



## CAS

L’outil CAS est utilisé pour toute la partie authentification et SSO des applications ENT. Dans les sources Lilie, cela représente sept projets regroupés dans le dossier parent « cas ». Afin de répondre aux besoins spécifiques, certaines parties du logiciel ont été redéveloppées. On peut distinguer trois aspects différents :

* La partie cas-server : déployée indépendamment, elle permet d’authentifier les utilisateurs et d’en conserver les informations. Les projets cas-server-webapp-lvl1 et cas-server-webapp-lvl2 définissent les interfaces de connexion pour l’utilisateur (lvl1 et lvl2 car il y a deux niveaux de connexion pour les enseignants).
* La partie cas-client-ent customisée pour les webapps : elle est déployée avec les webapps pour intercepter les appels vers celles-ci et les envoyer au cas-server
* La partie cas-client-portail customisée pour le portail : elle a le même rôle que la précédente mais pour le portail



## Batchs

### Description des batchs

Les batchs nécessaires à l’ENT sont regroupés dans le dossier « bat » du socle. Ce sont pour l’instant principalement des batchs autour de l’annuaire.

L’annuaire ENT est représenté sous la forme d’une base de données alimentée par des fichiers de l’annuaire fédérateur fournis par les académies. Cette base de données est enrichie par des données provenant de l’ENT lui-même. Ces données servent ensuite à toutes les webapps de l’ENT, ainsi qu’au portail pour gérer ses pages.

* bat-importfederateur : il s’occupe de récupérer les fichiers xml des académies pour les mettre à disposition pour l’alimentation, puis de renvoyer le fichier d’erreurs fonctionnelles de l’alimentation aux académies. Il est complété par des scripts shell pour gérer l’ordonnancement, le passage des « deltas » par rapport aux « complets » etc…
* bat-alimentation : c’est celui qui fait l’alimentation effective des données en base depuis les fichiers xml. Il se découpe en trois parties :
  + la purge : elle remanie les fichiers xml d’entrée qui ne seraient pas tout à fait d’aplomb (enlever les établissements que l’on ne veut pas déployer, transformer des ordres d’ ajout en ordres de modification lorsqu’un complet fourni des ordres d’ajout de personnes déjà présentes en base…)
  + l’alimentation : elle parse les différents xml dans l’ordre logique en fonction des relations entre les types de données. Des contrôles fonctionnels sont appliqués puis l’ordre est soit rejeté, soit appliqué en base.
  + le chargement des groupes : les données insérées en base sont remaniées par des procédures pour alimenter une table de « groupes » orientée performances pour la lecture des données
* bat-initportail : il sert à synchroniser le portail après chaque alimentation. Le portail se sert des données de l’annuaire pour gérer la création de ses pages et les accès.

### Packaging et déploiement

Les batchs sont tous packagés de la même manière : deux paquets sont générés

* un jar contenant les sources et configurations du batch
* un zip contenant le jar lui-même plus les librairies dont dépend le batch. Ces librairies sont à ajouter au classpath lors de l’utilisation du batch.

### Fonctionnement général

Ces batchs sont sous la forme d’une « main » java, avec un back suivant les mêmes règles de développement en couche que les autres projets, et faisant des appels aux apis.

L’import des différents contextes Spring utilisés par un batch (contexte du batch lui-même et contextes des apis) est fait par l’intermédiaire d’une classe *SpringFactory* propre au batch. Celle-ci est dans le package *utils* de chaque batch.

La propriété *DATABASE\_DEFINITION\_TYPE* du *config.properties* du batch doit être définie avec la valeur « *dataSource* ». Cela signifie que les accès aux bases de données éventuelles (par l’intermédiaire notamment des apis) se feront par les datasources définies dans le contexte Spring de l’api en question, et non par la datasource JNDI de Tomcat. (la valeur « *dataSourceJNDI »* servant aux webapps déployées dans Tomcat). A terme, si les batchs sont déployés avec Quartz au sein d’un war, ils pourront également utiliser les datasources JNDI.

## APIs

### Description des apis

Un certain nombre d’apis permettent d’encapsuler les différents traitements et accès aux ressources utiles pour les modules. Elles sont pour la plupart principalement constituées d’une partie back, et suivent le même modèle en couches (business/dao) que les modules web.

* api-annuaire : elle regroupe tous les traitements d’accès, règles de gestion et droits concernant les données de la base annuaire. Elle met à disposition un certain nombre de méthodes publiques pour utilisation par les autres projets du socle ou les modules web.
* api-admin : elle regroupe tous les traitements d’accès et règles de gestion concernant les données de la base d’administration. Les données sont principalement mises à jour par la console d’admin, mais consultées par tous les projets.
* api-liferay : cette api permet d’encapsuler les règles de gestion relatives au portail et les appels aux web-services de celui-ci. Les méthodes à disposition sont surtout des méthodes de mises à jour pour synchroniser ponctuellement le portail avec les informations saisies dans l’ENT.
* api-recherche : elle permet d’encapsuler les traitements d’indexation et de recherche avec Solr. Les méthodes publiques à disposition permettent de modifier les indexations de contenu propres à chaque webapp. La recherche peut ensuite être faite sur la globalité.
* api-portail : celle-ci est une api façade, servant d’intermédiaire entre les modules web et les apis. Elle dépend des quatre apis précédentes. Lorsqu’une fonctionnalité nécessite l’appel à différentes apis, elle permet de centraliser et ordonnancer les traitements entre les différentes apis, afin que cela soit transparent pour le module appelant. Les méthodes publiques à disposition sont principalement de la consultation qui se présentent sour la forme de « business » utilisables par les modules.
* api-portail-gestion : de manière similaire à l’api-portail, elle permet de centraliser et ordonnancer les traitements entre les différentes apis, mais regroupe cette fois-ci principalement les méthodes de mises à jour. Elle n’est donc pas disponible pour les modules, mais seulement pour les autres projets du socle (console d’admin, batchs…).
* api-web-droits : cette api est un peu particulière car elle n’est pas constituée que d’une partie back. Elle contient également toute une partie front (interface JSP + contrôleur Struts) équivalente à celle des modules webapps ENT. La partie back peut être utilisée de la même manière que les autres apis, mais l’intégration totale avec le front se fait en suivant plusieurs recommandations précisées dans la spécification technique de l’api elle-même.

Cette api permet donc de centraliser toute la gestion des droits propres aux entités des modules (contenus spécifiques des webapps). Elle centralise à la fois les règles de gestion métier et l’onglet d’interface commun à tous les modules, permettant d’affecter des droits aux personnes et groupes sur les contenus. Les actions Struts de l’api peuvent ainsi être appelées par les webapps qui intègrent la configuration nécessaire.

Le projet api-web-droits-portlet redéfinit certaines classes et configurations de l’api-web-droits dans le but de fonctionner avec les modules de type portlet (version de Struts différente).

* api-logging : cette api fournit simplement une classe d’extension de log4j pour gérer le back up journalier des fichiers de logs. Elle est utilisée dans les lo4j.xml de chaque projet du socle et des modules web.
* api-jira : elle permet d’encapsuler les appels aux web-services Jira. Les méthodes proposées permettent de créer des comptes et d’ajouter des demandes dans Jira.

### Packaging et déploiement

Les apis sont toutes packagées sous forme de jar que les modules peuvent ajouter en dépendance. Elles sont alors intégrées au war de l’application et déployées avec chaque webapp.

### Utilisation des apis par un module

A part l’api-logging, les apis fonctionnent toutes de la même façon. Elles sont configurées notamment avec Spring qui permet de mettre facilement à disposition les services : par l’intermédiaire de son interface, chaque implémentation de business contenant des méthodes publiques utilisables par d’autres projets, est définie par un bean Spring. Ces beans peuvent être ajoutés au contexte Spring du module appelant, et injectés dans les business du module.

* Ajout de la dépendance sur l’artifact Maven dans le pom.xml du module

*<dependency>*

*<groupId>org.lilie.socle</groupId>*

*<artifactId>api-portail</artifactId>*

*</dependency>*

* Ajout de l’import du contexte Spring de l’api dans le web.xml (ou dans une SpringFactory pour un batch)

*<context-param>*

*<param-name>contextConfigLocation</param-name>*

*<param-value>*

*classpath:spring/applicationContext.xml*

*classpath:spring/applicationContext\_api-portail.xml*

*classpath:spring/applicationContext\_fmk-core-web.xml*

*</param-value>*

*</context-param>*

* Ajout du business api à utiliser en attribut du business du module (ne pas oublier le setter pour l’injection Spring, le getter n’est pas nécessaire). Il faut utiliser le type interface et non l’implémentation directement, car celle-ci sera injectée par Spring.

*private ElevePortailBusiness elevePortailBusiness;*

*public void setElevePortailBusiness(ElevePortailBusiness elevePortailBusiness) {*

*this.elevePortailBusiness = elevePortailBusiness;*

*}*

* Ajout du business api en propriété du bean Spring du module

*<bean id="moduleBusiness" class="org.lilie.services.module.business.impl.ModuleBusinessImpl">*

*<property name="elevePortailBusiness" ref="elevePortailBusiness" />*

*</bean>*

Les DTOs des apis peuvent être utilisés directement par les modules.

Précisions concernant les fichiers de configuration :

* La gestion des fichiers de configuration des modules et apis (paramétrage de propriétés à externaliser des sources) est centralisée par la classe *ConfigAppli* du fmk-core-ent, qui est décrite dans la spécification technique du core. Pour les apis, l’appel de la méthode *ConfigAppli.getProprieteStr()* doit prendre en paramètre le nom de l’api dont on veut récupérer un paramètre de configuration. Le bean *ConfigAppli* peut également être injecté par Spring dans le business du module de la même manière que les business des apis.
* Comme pour les batchs, les modules utilisant des apis ou une base de données spécifique doivent préciser la propriété *DATABASE\_DEFINITION\_TYPE* dans le *config.properties* du module (pas des apis). La valeur doit être à *dataSourceJNDI* pour utiliser les datasources JNDI définies dans Tomcat. Sinon ce sont les datasources définies dans le contexte Spring qui sont utilisées (par exemple sur les postes de développement).

### Configuration interne des apis

Nommage

Afin de garder une homogénéïté entre les différentes apis et un bon fonctionnement lorsque plusieurs sont utilisées en même temps, certaines règles de nommage sont définies :

Les différents fichiers de configuration ont tous une racine commune et un suffixe portant le nom de l’api :

* Configuration de l’application : *src/main/resources/config\_<nomApi>.properties*
* Mapping DTO/POJO avec dozer : *src/main/resources/dozer/dozerMappings\_<nomApi>.xml*
* Chargement des map Ibatis : *src/main/resources/SqlMapConfig\_<nomApi>.xml*
* Les fichiers de contextes Spring sont découpés par couche et contiennent encore une fois le nom de l’api en suffixe : *src/main/resources/spring/applicationContext\_<nomApi>.xml*

De même, quand tous les contextes Spring sont chargés en même temps, il ne faut pas qu’il y ait de conflits dans les noms des beans. Le nom de l’api est donc présent dans chacun (ou au moins dans ceux pouvant prêter à confusion car similaires dans chaque api) : par exemple, « *annuaireAdapter* », « *annuaireSqlMap* » ou « *personnePortailBusiness* ».

Configuration des datasources

La configuration de l’accès aux données pour les apis fonctionne de la même manière que pour les bases de données de chaque module ENT. Deux types de datasources différents sont gérés systématiquement :

* JNDI définie dans Tomcat
  + Nom du bean spring : *dataSourceJNDI<nomApi>*
  + Le nom JNDI utilisé est valué dans le *config.properties* de l’api par la propriété *DATABASE\_JNDI*.
* Datasource simple définie dans le contexte Spring : *org.postgresql.ds.PGPoolingDataSource*
  + Nom du bean spring : *dataSource<nomApi>*
  + Les propriétés de la source de données sont définies dans le *config.properties* de l’api : *DATABASE\_SERVER\_NAME, DATABASE\_PORT\_NUMBER, DATABASE\_NAME, DATABASE\_USER, DATABASE\_PASSWORD, DATABASE\_DATASOURCE\_NAME, DATABASE\_INITIAL\_CONNECTIONS, DATABASE\_MAX\_CONNECTIONS*

Les deux beans spring sont déclarés en *lazy-init* afin que seul celui qui est réellement utilisé au lancement soit effectivement chargé.

Le bean *<nomApi>sqlMap* permet de définir la datasource et le fichier de mapping Ibatis à utiliser. Le choix de la datasource entre JNDI ou non est variabilisé, afin que le module puisse décider de manière globale pour toutes les datasources qu’il utilise. C’est le paramètre *DATABASE\_DEFINITION\_TYPE* qui est à mettre dans le *config.properties* du module (et non de l’api), comme précisé dans le paragraphe précédent.

Chargement des fichiers de configuration

Le chargement du fichier *config.properties* se fait de manière centralisée par un bean Spring customisé pour l’ENT. Cela permet de regrouper tous les paramètres de toutes les apis nécessaires au module et d’y accéder de manière centralisée. Chaque api précise donc à ce bean spécifique les informations qui lui sont propres :

Dans le *applicationContext\_<nomApi>.xml*, on définit le bean suivant :

*<bean id="propertyConfigurerApiAnnuaire"*

*class="org.lilie.socle.core.utils.spring.CustomPropertyPlaceholderConfigurer">*

*<property name="configAppli" ref="configAppli"/>*

*<property name="fichierConfigVarEnvirStr" value="annuaire.config"/>*

*<property name="fichierConfigInterneStr" value="config\_api-annuaire.properties"/>*

*<property name="placeholderPrefix" value="$annuaire{"/>*

*<property name="nomApi" value="api-annuaire"/>*

*</bean>*

* *configAppli* : instance d’un objet du core associé au CustomPropertyHolder. C’est le point d’entrée ensuite pour accéder aux propriétés dans le code
* *fichierConfigVarEnvirStr* :le nom de la variable d’environnement contenant le chemin du fichier de config.
* *fichierConfigInterneStr* : s’il n’y a pas de variable d’environnement (en environnement de dev), cette propriété définit le nom du fichier en interne dans le jar de l’api. Dans le packaging cible des releases, les fichiers de config sont externalisés et il n’y a rien dans le jar.
* *placeHolderPrefix*: c’est le préfixe utilisé dans les fichiers de contexte Spring pour accéder à une variable du fichier *config.properties* de l’api (par exemple pour les informations de la datasource).
* *nomApi*: c’est le nom de l’api utilisé dans *ConfigAppli* lorsque l’on veut récupérer une propriété de l’api dans le code

Configuration pour les tests d’une api

Pour tester une api avec de simples *main()* Java, il faut configurer le lancement de cette classe de la même manière qu’un module utilisant l’api (comme un batch utilisant l’api), afin que les données entrantes nécessaires à l’api soient présentes :

* un fichier *config.properties* de module contenant *DATABASE\_DEFINITION\_TYPE*
* un log4j.xml
* un fichier de contexte spring de type « module » : applicationContext.xml, contenant un bean de chargement de fichier de configuration customisé pour un module (légèrement différent de celui des apis) :

*<bean id="propertyConfigurer"*

*class="org.lilie.socle.core.utils.spring.CustomPropertyPlaceholderConfigurer">*

*<property name="configAppli" ref="configAppli"/>*

*<property name="fichierConfigVarEnvirStr" value="vide.config"/>*

*<property name="fichierConfigInterneStr" value="config.properties"/>*

*<property name="fichierLog4jVarEnvirStr" value="vide.log4j"/>*

*<property name="fichierLog4jInterneStr" value="log4j.xml"/>*

*<property name="delaiRechargementLog4jStr" value="60000"/>*

*</bean>*

Le préfixe et le nom de l’api ne sont pas nécessaires pour les modules, car par défaut sans préfixe on considère toujours celui du module englobant.

* Dans la classe, avant de commencer le test, il faut charger le contexte Spring avec le contexte du module et celui de l’api à tester :

*String FICHIER\_APPLICATION\_CONTEXT\_TEST = "spring/applicationContext.xml";*

*String [] tabCtx = {FICHIER\_APPLICATION\_CONTEXT\_TEST,*

*ConstantesApiAnnuaire.FICHIER\_APPLICATION\_CONTEXT};*

*ClassPathXmlApplicationContext classPathXmlApplicationContext = new ClassPathXmlApplicationContext(tabCtx);*

*PersonneBusiness pBusiness = (PersonneBusiness) classPathXmlApplicationContext.getBean("personneBusiness");*

## Frameworks core

Le dossier « fmk » contient plusieurs projets plutôt orientés socles techniques. Ce sont des frameworks spécifiques ENT qui ont plusieurs rôles :

* Donner une couche d’abstraction des outils techniques utilisés
* Proposer un socle technique de départ pour les nouveaux projets
* Centraliser les traitements qui peuvent être commun à tous les projets

Il en existe pour l’instant 6 (3 pour les clients riches, 1 pour tous les projets, 2 pour les clients légers) :

* *fmk-ria-ihm* :

framework de gestion d’interface pour les applications ria de l’ENT. Ce framework propose une interface plus ou moins générique servant de cadre pour les applications GWT. L’interface est paramétrable pour être adaptée au besoin du projet.

* *fmk-ria-serveur* :

framework permettant la mise en place de la partie back d’une application ria en GWT (configuration des services RPC…)

* *fmk-ria-puremvc*:

framework permettant de mettre en place le concept MVC pour les applications ria de l’ENT. Ce projet contient les traitements et classes de référence d’abstraction du framework externe PureMVC (échanges entre les couches par notifications).

L’utilisation de ces projets est plus détaillée dans la STD spécifique.

* *fmk-core-ent*:

framework core de tous les projets ENT, aussi bien pour les modules web que les apis et les batchs.

* + Il sert de socle technique de départ afin que tous les projets partent sur les mêmes bases techniques : gestion des exceptions, des logs, des fichiers de configuration, classes de référence d’abstraction des frameworks du back (Ibatis, Dozer…), cache métier…
  + Il regroupe également des constantes et DTOs de données ENT utiles à tous les niveaux (données utilisateurs, constantes de profils etc).
  + Un certain nombre d’utilitaires sont également disponibles afin de faciliter le développement et d’homogénéïser l’utilisation des outils externes : upload de fichiers, échanges de données entre modules (REST), récupération des informations de connexion, cryptage des données, envoi d’email…

Tous les projets doivent normalement utiliser ce framework core.

* *fmk-core-web* :

framework core de toutes les webapps ENT en client léger (Struts 2). Il a les mêmes objectifs que fmk-core-ent mais pour la partie front. Il regroupe donc :

* + des parties de socle technique pour le front : classes de référence d’encapsulation de Struts, gestion des pages d’erreur, gestion des données en session par le principe des conversations, centralisation des css et images des webapps, centralisation des tlds pour les librairies de tags et sitemesh
  + des constantes et données utiles à toutes les webapps dans la partie front : noms des paramètres passés dans la request…
  + des utilitaires divers pour faciliter le développement de la vue et du contrôleur : récupération des informations de connexion dans la request, partie contrôleur et vue de l’upload/download de fichiers, éditeur WYSIWYG, utilisation d’Ajax et de JQuery dans la vue

Ces deux projets core sont à utiliser de la même manière que celle décrite pour les apis dans le chapitre précédent (notamment avec Spring). Le nommage des fichiers et la configuration interne est également similaire à celle des apis. Ces projets suivent les mêmes règles de développement en couche que les autres projets (définies dans la documentation des normes de développement).

Les détails d’utilisation de chaque fonctionnalité se trouvent dans la spécification technique du projet lui-même.

* *fmk-core-web-portlet* :

Comme pour l’api-web-droits-portlet par rapport à l’api-web-droits, ce projet est le pendant de fmk-core-web pour les portlets. Il a comme dépendance fmk-core-web, et redéfinit seulement les classes et configurations à adapter pour les portlets, en raison surtout de la version de Struts qui diffère. Les portlets font donc référence à celui-ci plutôt qu’à fmk-core-web (aussi bien dans le pom.xml que dans les fichiers de configuration d’imports divers).

## Customisation de serveurs

Plusieurs projets sont en réalité des adaptations de serveurs développés hors ENT afin de les intégrer de manière complète au socle ENT et à ses besoins. Ces projets sont open-source, la modification de leur code source est donc autorisée.

### Portail

Deux projets servent à adapter le portail Liferay aux specificités de l’ENT :

* *prt-custom* : il est packagé sous forme d’un jar déployé dans le portail comme une nouvelle librairie disponible. Il permet de définir tous les traitements java spécifiques, comme l’interaction avec CAS (filter, servlets de login/logout), les traitements d’affichage du bandeau, l’interaction avec Xiti…

La gestion des fichiers de configuration *log4j.xml* et *config.properties* du projet est la même que pour les modules ENT (bean spring customisé + *ConfigAppli*).

* *prt-lookeliot* : il permet de définir le look (images, css) du portail. Un zip de static différent est packagé pour chaque porteur grâce à des configurations de build Maven.

Les spécifications techniques du portail permettent de rentrer plus en détail sur ces customisations.

### Solr

Certains modules, pas tous, indexent des données dans un moteur de recherche afin de pouvoir être disponibles de manière globale par l’intermédiaire d’un service de recherche.

Le projet solr-custom permet d’adapter le serveur SolR, utilisé pour l’indexation et la recherche des différents contenus, aux besoins de l’ENT. En plus de quelques modifications/ajouts du code source, ce projet sert surtout à stocker les différentes configurations du serveur et les schémas des contenus ENT à indexer.

La specification technique sur l’api-recherche rentre plus en détail sur ce sujet.

### Jabber

Le dossier « openfire » contient plusieurs projets relatifs à l’implémentation du serveur Jabber pour les ENT. Jabber est le protocole standard pour les communications de type messagerie instantanée. Nous utilisons le serveur *Openfire* comme implémentation de la partie serveur du protocole.

Cependant, afin de bien s’intégrer à notre socle et plus particulièrement à l’annuaire, des adaptations du serveur ont été nécessaires :

* *openfire-custom* : customisation des sources du serveur pour pointer sur l’annuaire ENT par l’intermédiaire des apis ENT. Les comptes de chat découlent alors directement des utilisateurs de l’annuaire. La connexion au serveur Jabber est transparente lorsque l’utilisateur est déjà connecté à l’ENT.

Ce projet est packagé dans un jar déployé avec les librairies du serveur Openfire.

Plusieurs plugins Openfire ont été également adaptés ou créés. Le modèle de packaging de ceux-ci étant prêt, d’autres pourraient éventuellement facilement être ajoutés dans l’avenir : le contenu, le packaging et le déploiement d’un plugin suivent des règles définies par Openfire. Maven nous permet de gérer cela au moment du build.

* *plugin-search-custom* : adaptation du plugin de recherche des comptes pour pointer sur l’annuaire par l’intermédiaire des apis ENT.
* *plugin-entpacketfilter* : permet de filtrer tous les paquets envoyés entre les clients et serveurs Jabber en fonction des droits des utilisateurs définis par l’api-annuaire. L’interface du client chat ENT empêche déjà les communications non autorisées, mais ce plugin est une sécurité dans le cas d’utilisation d’autres clients de chat par les utilisateurs.

Toujours en suivant le principe de cohérence entre tous les projets ENT, le chargement des fichiers de configuration se fait sur le même modèle que les autres projets, avec Spring et l’utilisation du fmk-core-ent.

La spécification technique du chat permet de rentrer plus dans les détails.

## Web Services

Plusieurs services externes proposent des web services pour accéder à leurs données. La génération des services Java à partir des WSDL fournis a été centralisée dans le dossier « ws ». Cela permet de regrouper les projets ayant besoin des mêmes dépendances relatives aux web services :

* ws-prt… : web services du portail Liferay. Permet de mettre à jour le portail lors des modifications de données dans l’annuaire.
* ws-jiraclient : web services d’accès à Jira pour créer ou modifier des demandes

Ils fonctionnent de la même façon :

* Utilisation d’Axis 1.4 comme librairie Java pour les appels aux web services
* Fichiers WSDL dans *src/main/resources/wsdl*
* Utilisation du plugin Maven pour générer les classes :

*<properties>*

*<dir.wsdl>${basedir}/src/main/resources/wsdl</dir.wsdl>*

*</properties>*

*<build>*

*<plugins>*

*<!-- Generation du java a partir des wsdl -->*

*<plugin>*

*<groupId>org.codehaus.mojo</groupId>*

*<artifactId>axistools-maven-plugin</artifactId>*

*<configuration>*

*<sourceDirectory>${dir.wsdl}</sourceDirectory>*

*<wsdlFiles>*

*<!-- Ne pas faire le lien en http mais en file Pour la récupération :*

*ssh VM-ENTIDF-DEV2 cd /tmp mkdir wsdl cd wsdl for i in `cat list.txt`;*

*do wget $i; done rename ?wsdl .wsdl \*wsdl -->*

*<wsdl>jirasoapservice-v2.wsdl</wsdl>*

*</wsdlFiles>*

*<packageSpace>org.lilie.socle.wsjiraclient</packageSpace>*

*<verbose>true</verbose>*

*<allElements>true</allElements>*

*<testCases>false</testCases>*

*</configuration>*

*<executions>*

*<execution>*

*<phase>process-resources</phase>*

*<goals>*

*<goal>wsdl2java</goal>*

*</goals>*

*</execution>*

*</executions>*

*</plugin>*

*</plugins>*

*</build>*

* Les appels aux classes Java générées sont ensuite encapsulés par l’intermédiaire d’apis (api-liferay, api-jira…) pour permettre de rendre leur utilisation transparente au code appelant.

## Console d’administration

### Rôle de la console

Comme vu dans le premier chapitre, bien qu’étant une webapp avec une interface utilisateur, la console d’administration fait partie du socle. En effet, elle permet de centraliser les paramétrages des différents modules et de consulter/modifier certaines données de l’annuaire. Elle sert de point d’ancrage pour chaque module qui doit s’interfacer avec elle pour « s’enregistrer » en tant que nouveau module de l’ENT. Chaque module peut ensuite être activable au besoin par chacun des établissements. Par l’intermédiaire de son api-admin (utilisée par l’api-portail en façade vue plus haut), la console fournit également un certain nombre de méthodes publiques accessibles pour les modules et le portail, afin de récupérer les paramétrages et informations diverses disponibles.

### Ajout d’un nouveau module

Lors de l’ajout d’un nouveau module, il faut :

* Dans le projet de la console d’admin : créer un écran d’activation et paramétrage similaire à ceux des autres modules (description précise du code dans la spécification technique de la console d’administration)
* Dans le module ajouté : récupérer les informations de paramétrage saisies grâce à l’api-portail (description de l’utilisation de l’api dans les chapitres précédents)
* Dans les projets « prt » : modifier le bandeau du portail pour faire apparaître le nouveau module (description du code du bandeau dans la spécification du portail)

### Fonctionnement général

La console est composée de trois projets dans le dossier *ria/ria-admin*. C’est une webapp client riche développée en GWT et PureMVC. Elle est déployée de la même manière que les webapps des différents modules. Elle peut accéder aux apis du socle au besoin (sans forcément passer par l’api-portail). Le code de la console utilise également les 3 projets socle fmk-ria décrits plus haut.

Des spécifications techniques sur la console, ainsi que sur l’utilisation du socle client riche, sont disponibles.

## Projet modèle pour création d’une webapp

### Principe

L’outil Maven permet de créer des projets « template » servant de base de départ pour la création d’un nouveau projet. Ceux-ci s’appellent des « *archetype* ». lls permettent de prédéfinir l’arborescence du projet, les différents packages de base, avec certaines classes, ressources et configurations déjà présentes.

Nous avons utilisé ce principe pour créer un modèle pour les modules ENT. Ainsi, il n’est pas nécessaire de recréer un projet de zéro à chaque fois. Les dépendances et configurations de base d’un module sont automatiquement créées à l’aide de *l’archetype*. Pour l’instant, il n’existe qu’un modèle pour les modules en client léger (Struts 2).

### Utilisation pour création d’un module

L’archetype est un artifact Maven déployé comme toutes les autres librairies, avec son groupId et sa version :

*<groupId>org.lilie.services.archetype</groupId>*

*<artifactId>tem-web-archetype</artifactId>*

* En ligne de commande, se placer dans le workspace là où l’on veut créer le projet
* Utiliser la commande : *mvn archetype:generate*

Le catalogue utilisé peut être précisé (voir la documentation en ligne du *maven-archetype-plugin* pour les détails), sinon il est demandé en prompt.

* On choisit l’archetype parmi ceux proposés par la commande, puis on saisit les informations suivantes demandées :
  + groupId : *org.lilie.services*
  + artifactId : *web-<nommodule>*
  + version en cours
  + package : org.lilie.services.web<nommodule>
* Le projet est alors créé. Toutes les configurations sont déjà pré-remplies. Les noms de packages, fichiers et beans Spring ont été automatiquement adaptés pour prendre en compte le nom du module.

Seule une configuration est à changer à la main : la configuration du *CustomPropertyHolder* dans *applicationContext.xml*. Il faut changer les noms des variables d’environnement du config et du log4j pour les adapater avec le nom du module.

* Pour importer le projet dans Eclipse, ne pas oublier de supprimer d’abord le dossier « *.settings »* s’il existe dans le projet nouvellement créé.

### Modification de l’archetype

Le projet servant de source pour la création de l’archetype est le projet *tem-web*, dans la partie *services/services-initiaux-web.*

* Modifier les sources et ressources à créer
* Ne pas oublier de supprimer les dossiers et fichiers qui sont générés par le build, car ils ne doivent pas être présents dans l’archetype déployé (target, fichiers copiés du core…)
* Se placer dans le dossier du projet tem-web
* Lancer la commande :

*mvn archetype:create-from-project -Darchetype.languages=java,xml,txt,groovy,cs,mdo,*

*aj,jsp,gsp,vm,html,xhtml,properties,.classpath,.project,resources*

Ne pas oublier le “*resources*” pour que les noms de package soient également changés dans les dossiers de ressources.

* Se placer dans …\tem-web\target\generated-sources\archetype\
* Lancer la commande *mvn install* pour le mettre dans le repository local (ou *mvn deploy* pour le déployer sur le Nexus)

## Tableau de correspondance des fonctionnalités

Voici un tableau faisant correspondre les fonctionnalités décrites dans le premier chapitre au découpage technique des projets.

|  |  |
| --- | --- |
| **FONCTIONNALITE** | **PROJETS CONCERNES** |
| ***Sécurité*** |  |
| authentification | cas |
| SSO | cas |
| Droits d’accès portail | prt-custom |
| Outils de sécurisation des urls http | api-web-droits,  api-annuaire |
| Outil de cryptage | fmk-core-ent |
| ***Annuaire*** |  |
| Alimentation | bat-alimentation,  bat-importfederateur |
| Synchronisation portail | bat-initportail |
| Encapsulation appels portail | api-liferay  (par api-portail) |
| Accès aux données de l’annuaire | api-annuaire  (par api-portail) |
| RG et droits sur l’annuaire | api-annuaire |
| Données de connexion | api-annuaire |
| **Administration** |  |
| Visualisation/export annuaire | ria-admin |
| Création de groupes supplémentaires | ria-admin |
| Gestion de comptes invités et administrateurs | ria-admin |
| Activation/désactivation des modules | ria-admin |
| Paramétrage et droits d'accès aux modules selon les profils | ria-admin |
| Remontée d'incidents | ria-admin |
| Services d'accès à ces données de paramétrage utilisables par les modules | api-admin (api-portail) |
| ***Socles techniques*** |  |
| abstraction frameworks et classes de référence : persistance Ibatis, métier | fmk-core-ent |
| Abstraction frameworks et classes de référence : contrôleur/vue (Struts2) | fmk-core-web |
| Abstraction frameworks et classes de référence : contrôleur/vue (GWT, PureMVC) | fmk-ria-ihm,  fmk-ria-puremvc |
| gestion des exceptions | fmk-core-ent |
| gestion des fichiers de configuration | fmk-core-ent |
| gestion du principe de conversations pour les données en session | fmk-core-web |
| cache métier | fmk-core-ent |
| gestion des logs | fmk-core-ent |
| centralisation des feuilles de style ENT | fmk-core-web |
| template de génération d’un projet module | tem-web |

|  |  |
| --- | --- |
| ***Utilitaires*** |  |
| Récupération informations utilisateur connecté | fmk-core-web, fmk-core-ent |
| Moteur de recherche | api-recherche (par api-portail), solr-custom |
| Gestion des droits sur les contenus des modules | api-web-droits |
| Gestion des échanges de données entre modules | fmk-core-ent |
| Gestion des fichiers uploadés sur le serveur applicatif | fmk-core-ent, fmk-core-web |
| Utilitaire de cryptage MD5 et AES | fmk-core-ent |
| Envoi d’emails | fmk-core-ent |
| Api d’interfaçage avec Jira | api-jira |
| Customisation d’un serveur Jabber par rapport à l’annuaire ENT | openfire |

## Interactions et flux entre les projets

* Point de départ : fmk-core-ent est le projet central commun à tous les projets. Il sert de framework de base pour tous les autres qui ont une dépendance sur lui. Il n’a donc aucune dépendance sur les autres projets ENT.
* Le projet fmk-core-web dépend de fmk-core-ent et l’étend pour compléter la partie web. Les autres projets du socle ne dépendent pas de ce projet qui est fait pour les modules.
* Accès aux apis :
  + Les fonctionnalités des apis ne sont pas accédées en direct par les modules mais par l’intermédiaire de l’api-portail qui centralise le tout et sert de façade pour proposer seulement ce qui est utile.
  + Les projets du socle (console d’admin, batchs, cas…) peuvent par contre accéder directement aux apis sous-jacentes sans passer par la façade.
* Les apis ont par contre des dépendances entre elles :
* l’api-portail pointe donc sur les quatres apis proposant des fonctionnalités aux modules : api-recherche, api-annuaire, api-admin, api-liferay
* l’api-portail-gestion a les mêmes dépendances que l’api-portail. Elle est utilisée par les projets du socle (console d’admin, batchs…) pour partager des traitements communs d’appel aux apis sous-jacentes.
* l’api-web-droits, qui est une api également de front, est à un niveau au dessus, comme les modules, car elle dépend de ce que propose l’api-portail dans son ensemble
  + les autres apis ne communiquent pas entre elles, elles gèrent chacune leurs fonctionnalités spécifiques.
* Les projets customisation de serveurs (openfire, prt-custom, solr-custom) sont au dessus des apis dont ils se servent pour intégrer notamment les données annuaire.
* Interactions avec les modules :
  + Les projets ws ne sont pas accédés directement par les modules. Ils sont encapsulés par les apis (api-liferay, api-jira).
  + Les projets du socle n’accèdent pas aux modules. Aucun flux n’est prévu dans ce sens là par l’architecture mise en place. Ce sont les modules qui utilisent le socle. De même pour le cas particulier de la webapp de la console d’admin : l’api-admin permet aux modules d’accéder aux données de la console, mais la console n’appelle pas les différents modules qui gèrent tous seuls leurs données spécifiques. Les paramétrages d’activation et droits par profil sont bien des données internes à la console, consultables par les modules.
  + Les modules sont au plus haut niveau et dépendent des différents projets du socle, qu’ils embarquent dans leur war pour le déploiement.
  + Les modules sont indépendants les uns des autres. Pour des besoins ponctuels d’échanges de données entre ces modules distants, des flux de type HTTP sont prévus : les traitements métiers restent du côté du module contenant les données concernées, qui met à disposition des services REST pour les autres modules.

