



Formation Esup-Commons V2

Manipulations et exercices

Date et lieu de la formation : 3 et 4 Novembre 2011 à Paris

Version du document : 0.0.1 (28/10/11 14:09)

Intervenant : Raymond Bourges et Céline Didier

Introduction

Le but de cette formation est de parvenir réaliser une application de bout en bout à l'aide du framework *esup-commons*. Ceci inclut la maîtrise de l'environnement de développement et des méthodologies préconisées par *esup-commons*.

Pour cela, nous allons réaliser une application simplifiée de gestion de tâches de type *todo-list*.

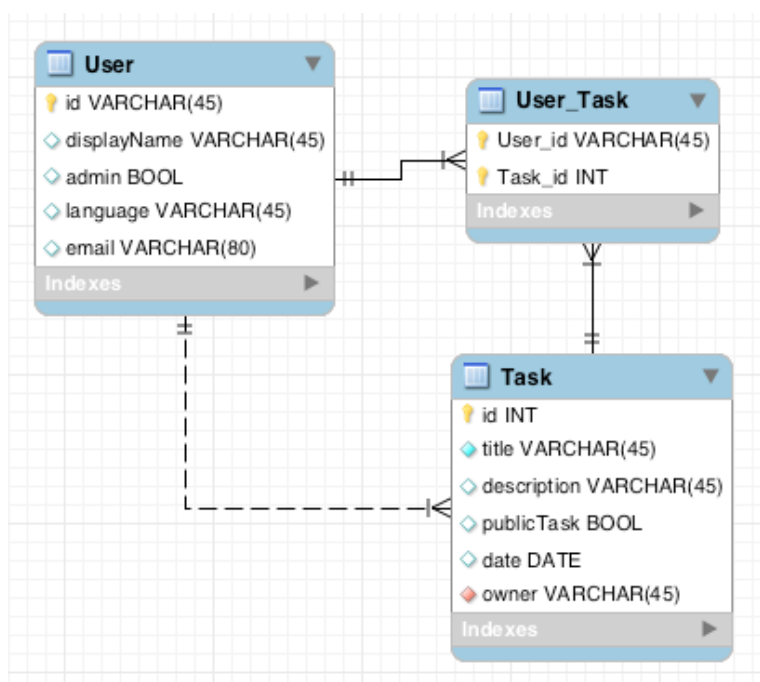
Fonctionnement final attendu

Une application web autonome qui proposera ;

- Une page d'accueil publique, une page de gestion des tâches et une page de gestion des utilisateurs.
- La page d'accueil listera les tâches publiques et offrira un bouton de connexion
- La page de gestion des tâches affichera les tâches qui appartiennent ou qui sont assignées à l'utilisateur.
- Une fois l'utilisateur authentifié il pourra procéder à la gestion des tâches (ajout, édition, modification et suppression de tâches).
- On pourra afficher le détail d'une tâche et l'application fournira un lien direct vers cette tâche et offrira la possibilité d'envoyer le descriptif de la tâche par e-mail.
- La page de gestion des utilisateurs proposera des options d'administration et notamment l'ajout d'utilisateurs en base après recherche dans un annuaire *LDAP*.
- Un *web service* exposera entre autre une méthode permettant de récupérer la liste des 10 dernières tâches d'un utilisateur.

Une portlet d'accroche qui interrogera le web service et n'affichera que la liste des n dernières tâches de l'utilisateur connecté au portail.

Schéma de la base de données



Sommaire

INTRODUCTION	2
SOMMAIRE	3
INDEX DES EXERCICES	6
PRISE EN MAIN DE L'ENVIRONNEMENT	8
1 LA MACHINE VIRTUELLE	8
1.1 PRESENTATION	8
1.2 INSTALLATION DE LA MACHINE VIRTUELLE	8
1.3 DEMARRAGE	8
2 L'ENVIRONNEMENT DE DEVELOPPEMENT ECLIPSE	9
3 DECOUVERTE AVEC ESUP-EXAMPLE	9
3.1 CHECKOUT DEPUIS SVN	9
3.2 ORGANISATION DES FICHIERS	10
3.3 FONCTIONNEMENT EN MODULES MAVEN	11
3.4 DEMARRAGE	13
CREATION D'UN PROJET	16
1 CREATION D'UN PROJET MAVEN A PARTIR DE ESUP-BLANK	16
2 UN COUP DE MENAGE...	18
2.1 PREMIER LANCEMENT	18
2.2 MENAGE DANS LES LIBRAIRIES JSF	20
2.3 MENAGE DES MODULES INUTILES	21
BEANS SPRING	22
1 LE FICHIER DE CONFIGURATION PRINCIPAL	22
2 L'INJECTION	22
3 ACCES AUX PARAMETRES DE CONFIGURATION	25
GESTION DES LOGS ET TESTS UNITAIRES	27
1 GESTION DES LOGS	27
1.1 UTILISATION DANS LE CODE JAVA	27
1.2 ACTIVATION DU MECANISME DE LOG	27
2 LES TESTS UNITAIRES	27
2.1 EXECUTION DES TESTS UNITAIRES VIA MAVEN	28
2.2 EXECUTION DES TESTS UNITAIRES DANS ECLIPSE	32
ACCES AUX DONNEES	33
1 L'OBJET METIER	33
2 LA COUCHE DAO	33
3 LA COUCHE SERVICES	35
4 PREMIERS TESTS D'ECRITURE ET LECTURE EN BASE	37
LES VUES	41
1 JSF ET SES LIBRAIRIES	41
2 FACELET	41
3 PAGES ET NAVIGATION	42

INTERNATIONALISATION	47
1 CONFIGURATION	47
2 DECLARATION ET UTILISATION DES ENTREES	47
2.1 DECLARATION	47
2.1.1 Via un éditeur de texte	47
2.1.2 Via ResourceBundleEditor dans eclipse	47
2.2 UTILISATION	48
2.2.1 Du côté de la vue	48
2.2.2 Du côté du code Java	48
3 SURCHARGE DES ENTREES	48
4 DEFINITION DES LANGAGES	49
5 LES MESSAGES D'ERREUR PAR DEFAUT DE JSF	49
FORMULAIRES ET VALIDATION	50
1 FORMULAIRES ET BINDING	50
2 LES CONVERTISSEURS	51
3 LES VALIDATEURS	53
GESTION DES EXCEPTIONS	56
ENVOI D'E-MAIL	59
AUTHENTIFICATION	61
ACCES A UN ANNUAIRE LDAP	65
1 PARAMETRAGE DU LDAP	65
2 RECHERCHE ET UTILISATION DE L'ANNUAIRE	66
GESTION DES URL	69
WEBSERVICES	71
1 LES WEBSERVICES CXF	71
2 LES SERVICES <i>REST</i>	73
DEPLOIEMENT EN PORTLET	77
1 CREATION D'UN NOUVEAU PROJET POUR LA PORTLET	77
2 PREMIER LANCEMENT	79
2.1 CREATION DE L'ARCHIVE WAR	79
2.2 DEPLOIEMENT DE LA PORTLET DANS LE PORTAIL	79
2.3 PUBLICATION DE LA PORTLET	80
2.4 TEST DE LA PORTLET	82
2.4.1 Ajout de la portlet à l'environnement	82
2.4.2 Récupération d'informations	83
3 DEVELOPPEMENT D'UN PORTLET A L'AIDE DE <i>SPRING-MVC</i>	85
3.1 UTILISATION DU WEBSERVICE	85
3.1.1 Démarrage des webservices	85
3.1.2 Génération des objets métiers à partir du WSDL	86
3.1.3 Interrogation du webService	87
3.1.4 Affichage des resultats	89
3.1.5 Tests et amélioration	89
3.2 UTILISATION DU SERVICE <i>REST</i>	90
ANNEXES	92

LEGENDE	94
1 TITRE NIVEAU 1	94
1.1 TITRE NIVEAU 2	94
1.1.1 Titre niveau 3	94
INDEX	95

Index des exercices

EXERCICE N°1 :	RECUPERATION D'UN PROJET DEPUIS SVN	9
EXERCICE N°2 :	LANCEMENT D'UNE APPLICATION MAVEN	13
EXERCICE N°3 :	CREATION D'UN PROJET A PARTIR D'UN ARCHETYPE MAVEN	16
EXERCICE N°4 :	INSTANCIATION D'UN BEAN SIMPLE	23
EXERCICE N°5 :	PERSONNALISATION DES CONFIGURATIONS GRACE A L'INJECTION	25
EXERCICE N°6 :	TEST UNITAIRE SIMPLE	28
EXERCICE N°7 :	TEST UNITAIRE AVANCE	30
EXERCICE N°8 :	CREATION D'UN OBJET METIER SIMPLE.	33
EXERCICE N°9 :	CREATION D'UNE RELATION ENTRE OBJETS METIERS	39
EXERCICE N°10 :	TEST DE LA COUCHE DOMAIN DANS UN TEST UNITAIRE	39
EXERCICE N°11 :	AJOUT D'UN MENU VIA UN TEMPLATE FACELET	41
EXERCICE N°12 :	AJOUT D'UNE NOUVELLE PAGE AVEC REGLE DE NAVIGATION	42
EXERCICE N°13 :	PARCOURS D'UN TABLEAU	43
EXERCICE N°14 :	DECLARATION ET UTILISATION DES ENTREES	47
EXERCICE N°15 :	SURCHARGE D'UN BUNDLE	48
EXERCICE N°16 :	AJOUT D'UN LANGAGE	49
EXERCICE N°17 :	CREATION D'UN FORMULAIRE DE SAISIE SIMPLE	50
EXERCICE N°18 :	UTILISATION D'UN CONVERTISSEUR PREDEFINI	51
EXERCICE N°19 :	CREATION D'UN CONVERTISSEUR	52
EXERCICE N°20 :	VALIDATION DES CHAMPS GRACE A UN VALIDATEUR	53
EXERCICE N°21 :	VALIDATION DES CHAMPS GRACE A JSR 303	54
EXERCICE N°22 :	AMELIORATION DU FORMULAIRE : EDITION ET SUPPRESSION	54
EXERCICE N°23 :	AJOUT DE FONCTIONS AJAX POUR L'ERGONOMIE	55
EXERCICE N°24 :	L'AFFICHAGE DES EXCEPTIONS	56
EXERCICE N°25 :	AJOUT DES FONCTIONNALITES D'E-MAIL	59
EXERCICE N°26 :	METTRE EN PLACE UNE AUTHENTIFICATION CAS	61
EXERCICE N°27 :	CREATION DE BOUTONS DE CONNEXION ET DECONNEXION	63

EXERCICE N°28 :	RECHERCHE DES INFORMATIONS D'UNE PERSONNE DANS LE LDAP	66
EXERCICE N°29 :	RECHERCHE D'UNE OU PLUSIEURS PERSONNES DANS L'ANNUAIRE	67
EXERCICE N°30 :	CREATION D'UN LIEN DIRECT	69
EXERCICE N°31 :	EXPOSER UN WEBSERVICE CXF	71
EXERCICE N°32 :	EXPOSER UN SERVICE <i>REST</i>	73
EXERCICE N°33 :	DEPLOIEMENT ET INSTALLATION D'UNE PORTLET	77
EXERCICE N°34 :	INTERROGATION DES WEBSERVICES	85
EXERCICE N°35 :	INTERROGATION DU SERVICE <i>REST</i>	90
EXERCICE N°36 :	TITRE	94

Prise en main de l'environnement

1 La machine virtuelle

1.1 Présentation

Pour l'ensemble de cette formation nous utiliserons une machine virtuelle autonome sur laquelle sont installés :

- Java 6 (JDK)
- Maven
- Un serveur de base de données MySQL
- Un annuaire LDAP
- Un serveur CAS
- Un Portail Esup v3.2

1.2 Installation de la machine virtuelle

Créer une machine virtuelle dans VirtualBox

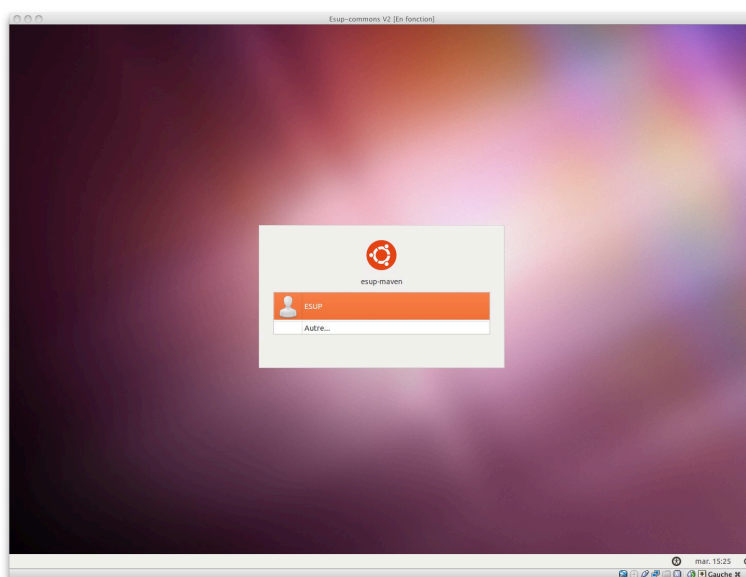
- Système : Linux / Version : Ubuntu
- Affecter au minimum 1024Mo de mémoire
- Utiliser un disque dur existant
 - Nouveaux supports virtuels
 - Pointer vers le fichier **esup-commons-formation.vdi**
- Terminer

Activer les Virtualisations de processeur (si votre matériel le permet)

1.3 Démarrage

Lancer la machine virtuelle.

- Login / mot de passe : esup/esup



2 L'environnement de développement Eclipse

Eclipse est déjà installé sur la machine virtuelle. (Documentation d'installation : <http://www.esup-portail.org/x/fgEABg>)

Plugins installés :

- Spring IDE
- Checkstyle
- Maven
- Subclipse
- Resource Bundle Editor

☞ **Démarrer *Eclipse* (raccourci sur le bureau)**

3 Découverte avec esup-example

Nous allons commencer par nous familiariser avec *maven* et la structure d'un projet *esup-commons* grâce à l'application *esup-example*.

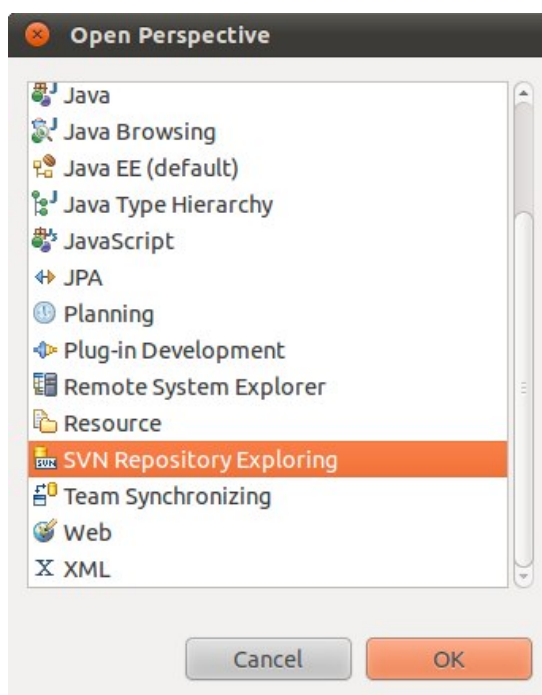
3.1 Checkout depuis SVN

Exercice N°1 : Récupération d'un projet depuis SVN

Récupérer le projet *esup-example* depuis le répertoire *trunk* du dépôt SVN de *esup-commons*.

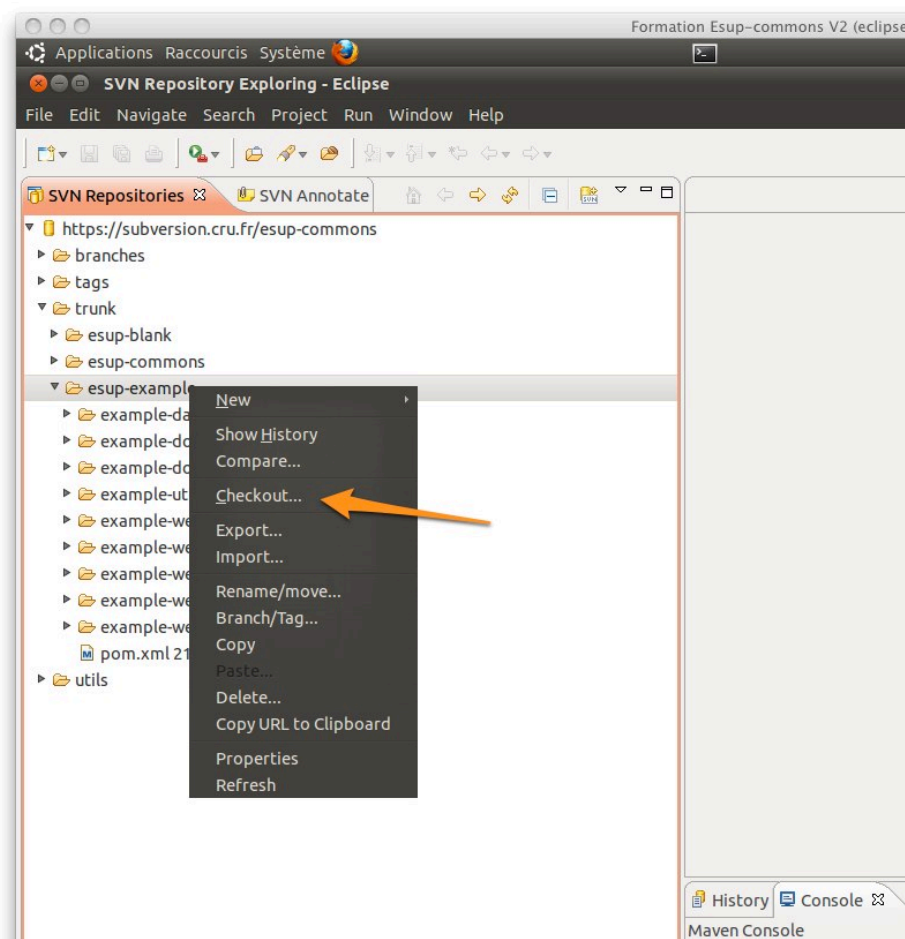
Passer en perspective *SVN Repository Exploring* dans Eclipse

☞ **Windows > Open Perspective > Other...**



Le dépôt <https://subversion.cru.fr/esup-commons> doit déjà être configuré.

Faire un *checkout* sur `/trunk/esup-example`

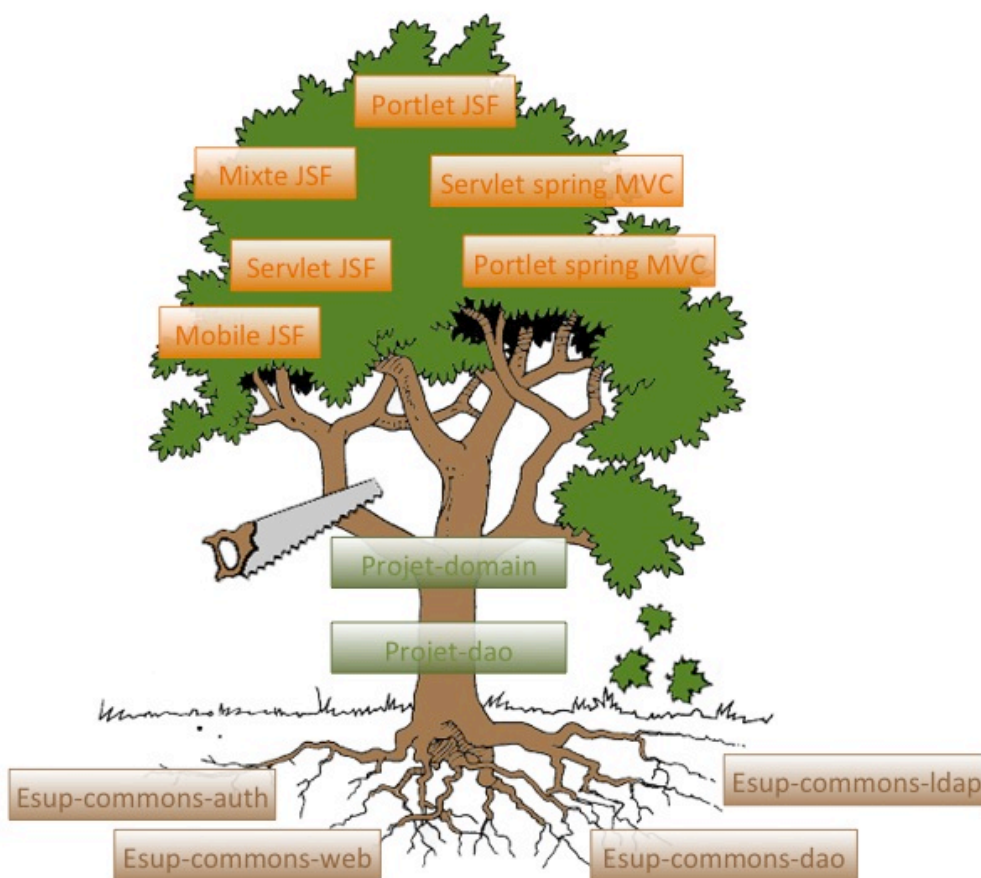


3.2 Organisation des fichiers

Passer en perspective *Java EE*.

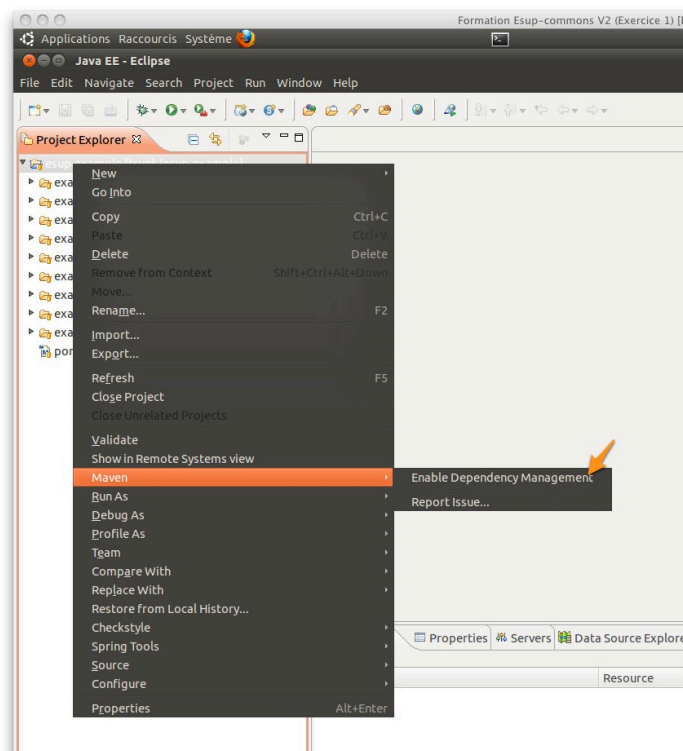


3.3 Fonctionnement en modules Maven



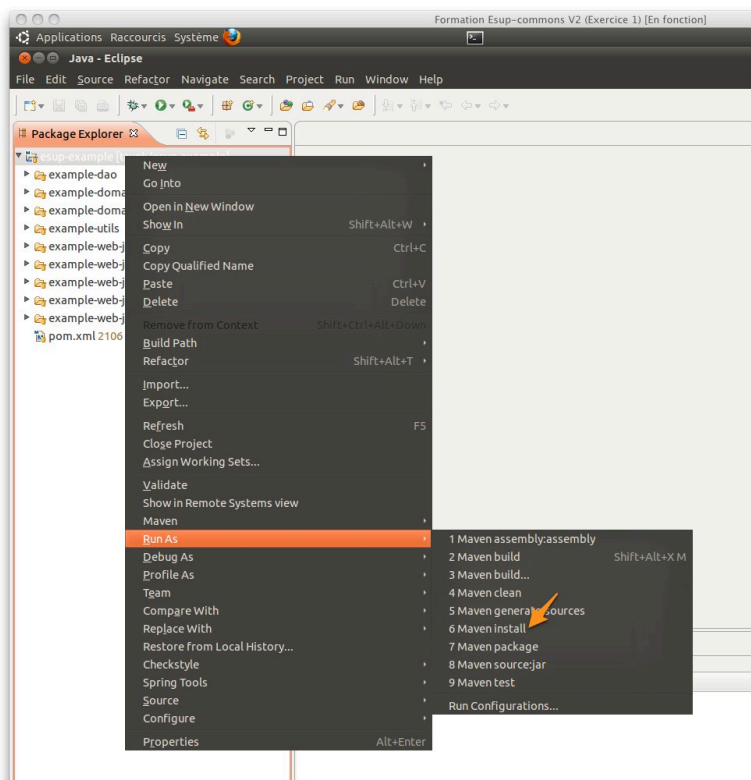
Activer la gestion des dépendances *maven* sur le projet

☞ **Clique droit > Maven > Enable Dependency Management**



Puis construire le projet :

 **Clique droit > Run as... > 6 Maven install**



On observera dans la console *Java* :

```
13/10/11 16:01:57 CEST: /home/esup/workspace/esup-example
13/10/11 16:01:57 CEST: mvn -B install
```

Ceci prend un certain temps si les librairies ne figurent pas dans votre répertoire **.m2**

```
[INFO] -----
[INFO] Reactor Summary:
[INFO] -----
[INFO] esup-example .....
SUCCESS [43.472s]
[INFO] example-domain-beans .....
SUCCESS [1:12.760s]
[INFO] example-dao .....
SUCCESS [20.388s]
[INFO] example-domain-services .....
SUCCESS [1:40.356s]
[INFO] example-utils .....
SUCCESS [0.115s]
[INFO] example-web-jsf-shared .....
SUCCESS [44.605s]
[INFO] example-web-jsf-servlet .....
SUCCESS [19.458s]
[INFO] Unnamed - org.esupportail:example-web-jsf-mixed:war:1.0-
SNAPSHOT SUCCESS [1:12.226s]
[INFO] example-web-jsf-mobile .....
SUCCESS [4.589s]
[INFO] example-web-jsf-portlet .....
SUCCESS [37.499s]
[INFO] -----
[INFO] -----
```

```
[INFO] BUILD SUCCESSFUL
[INFO] -----
[INFO] Total time: 7 minutes 1 second
[INFO] Finished at: Thu Oct 13 16:08:59 CEST 2011
[INFO] Final Memory: 56M/169M
[INFO] -----
```

3.4 Démarrage

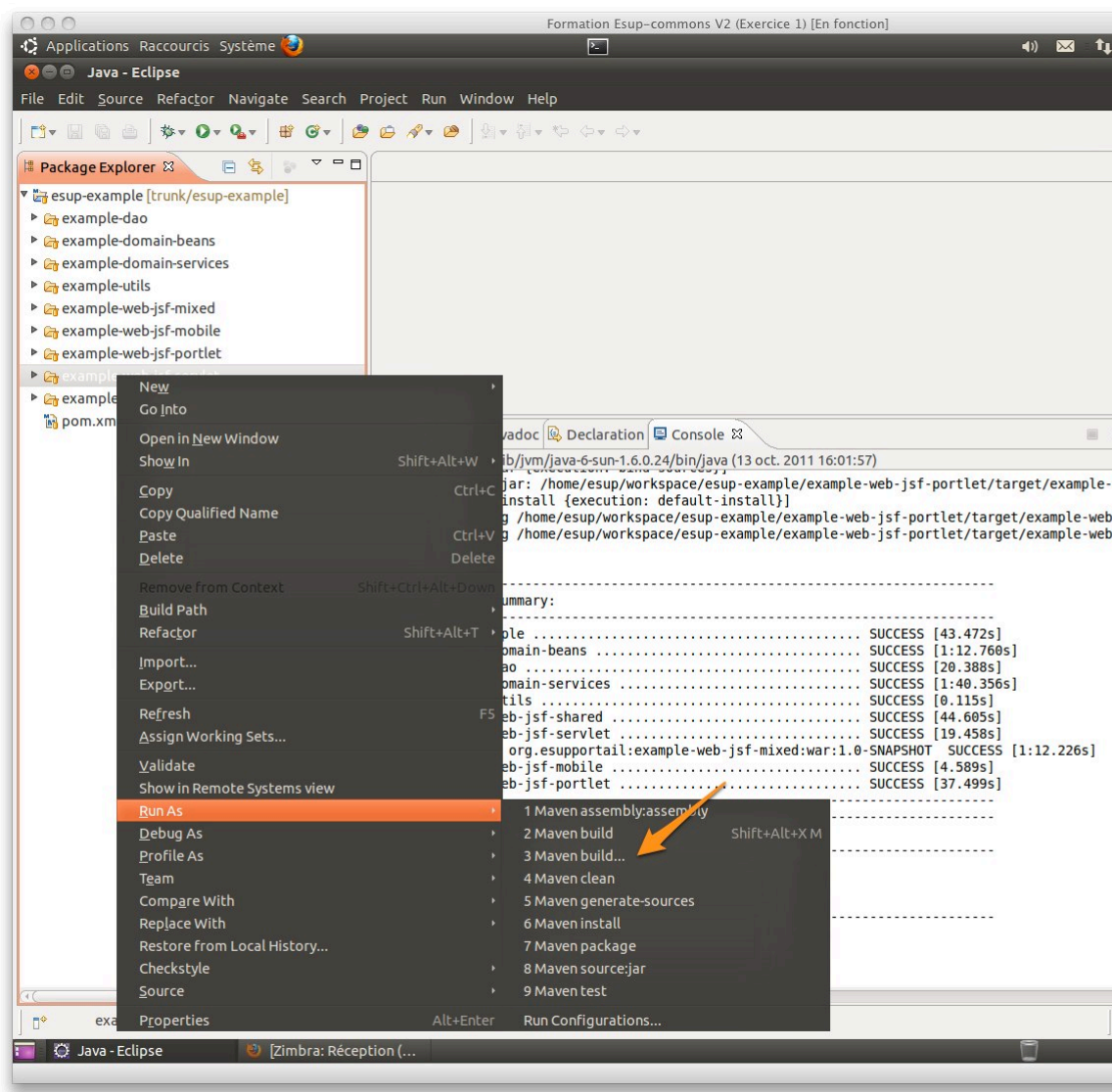
Exercice N°2 : Lancement d'une application *Maven*

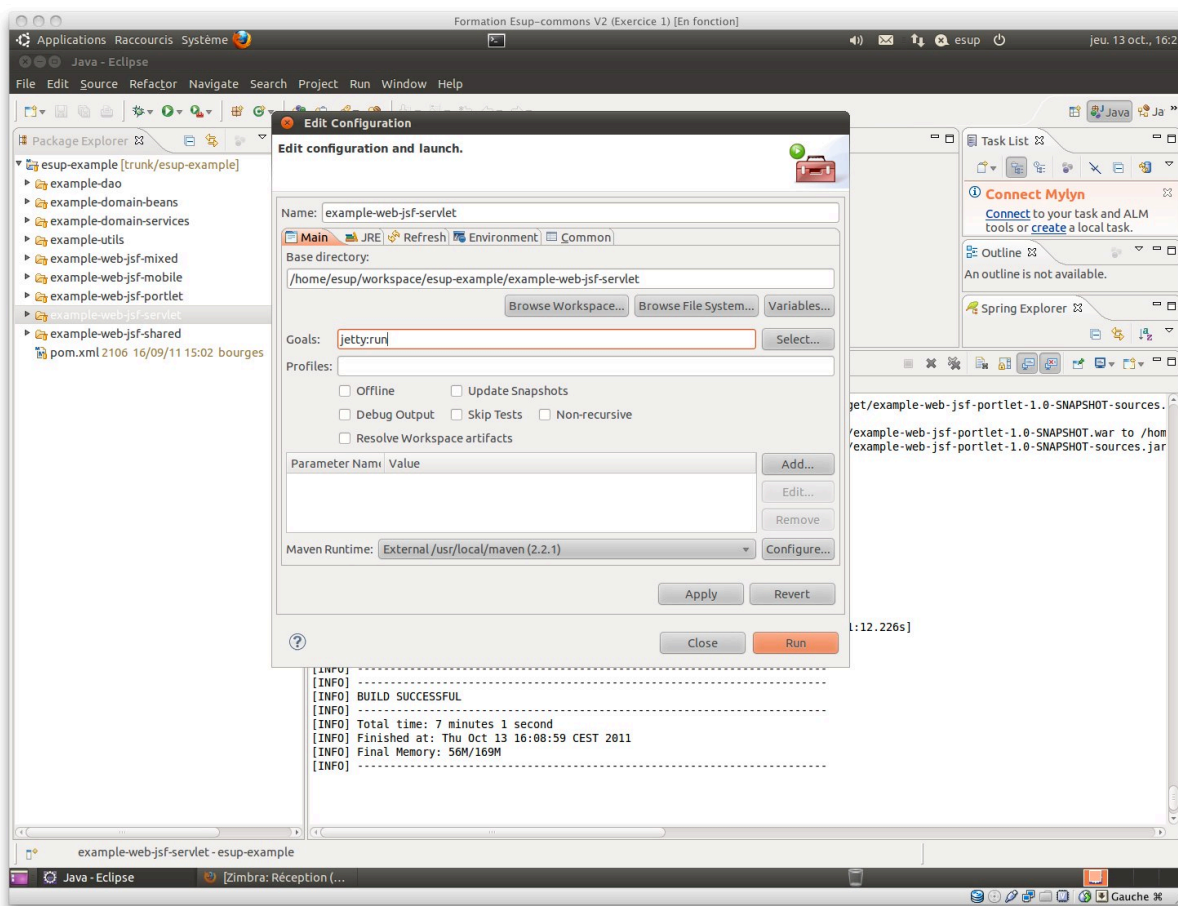
Nous allons commencer nos premiers tests sur une application de type *servlet* traditionnelle.

Sur le module `web-jsf-servlet`.

```
mvn jetty:run
```

- ☞ Cliquez droit sur le projet `web-jsf-servlet` > Run As > 5 Maven build...
- ☞ Saisir un nom de tâche et goals : `jetty:run`





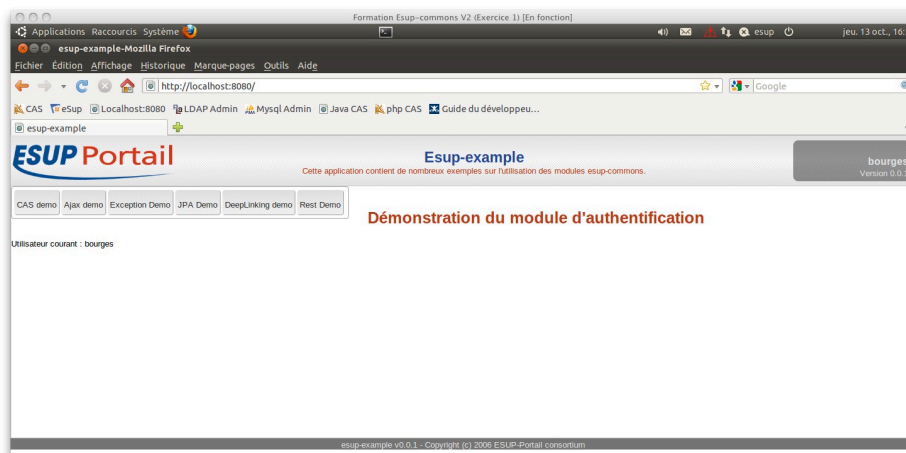
On observera dans la console *Java* :

```
13/10/11 16:28:41 CEST: /home/esup/workspace/esup-exemple/example-
web-jsf-servlet
13/10/11 16:28:41 CEST: mvn -B jetty:run
```

L'exécution doit se terminer par les messages suivants dans la console :

```
[...]
2011-10-13 16:29:53.395:INFO::Started
SelectChannelConnector@0.0.0.0:8080
[INFO] Started Jetty Server
```

Démarrer un navigateur et se rendre sur <http://localhost:8080>



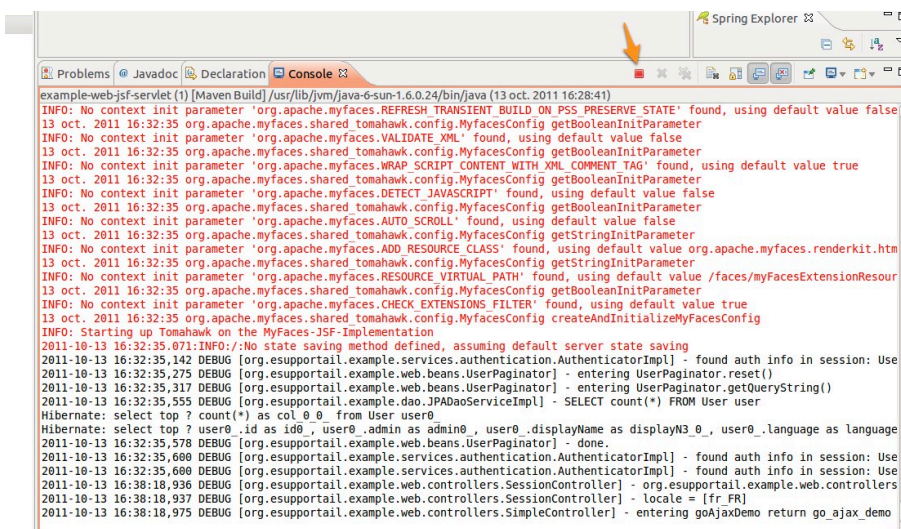
Tester les différentes options du menu



Stopper le serveur :

- Soit en lançant une execution *maven jetty* : *stop* equivalente à la commande :

```
mvn jetty:stop
```
- Soit via Eclipse



Création d'un projet

Exercice N°3 : Création d'un projet à partir d'un archetype *maven*

Esup-commons propose un archetype qui permet de créer automatiquement toute l'arborescence d'un projet vide qui respectera les préconisations du consortium pour le développement d'une application.

1 Création d'un projet *maven* à partir de esup-blank

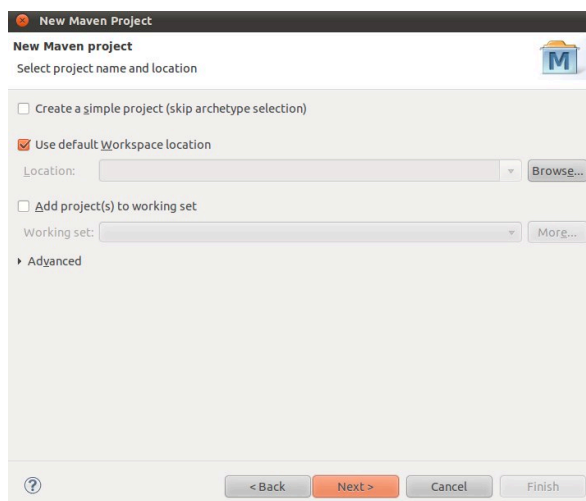
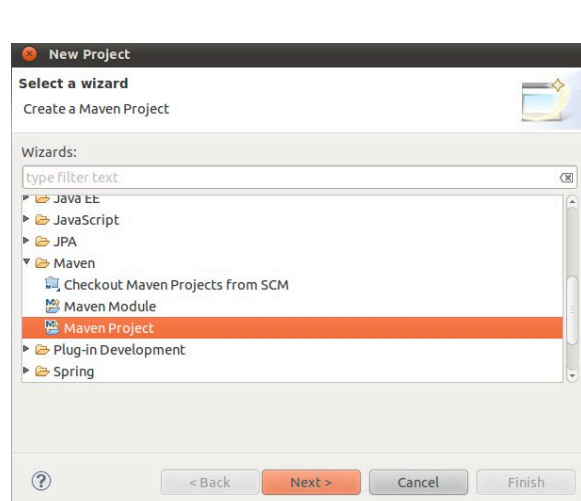
La documentation :

<http://www.esup-portail.org/pages/viewpage.action?pageId=100663444>

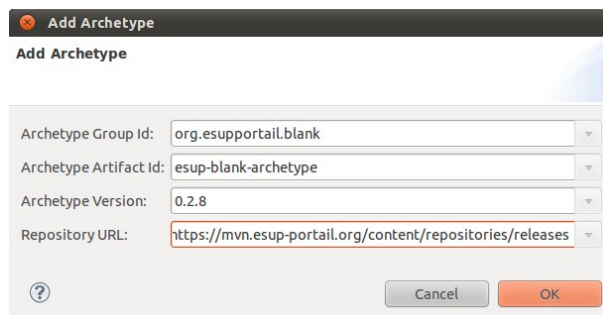
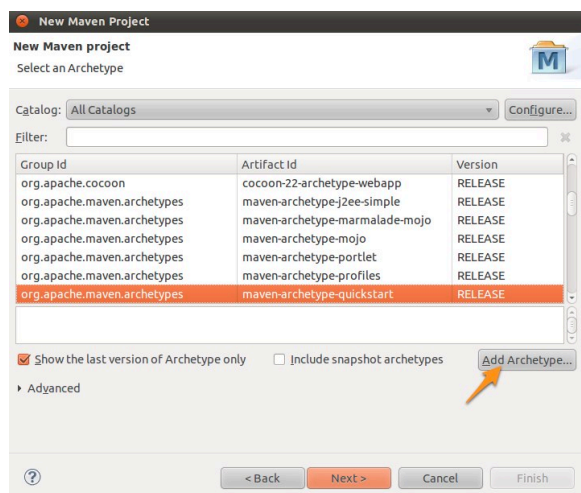
La structure du projet va être construite à partir d'un archetype *maven* dont voici les paramètres :

- `archetypeRepository=https://mvn.esup-portail.org/content/repositories/releases`
- `archetypeGroupId=org.esupportail.blank`
- `archetypeArtifactId=esup-blank-archetype`
- `archetypeVersion=0.2.8`
- `groupId=org.esupportail.formation`
- `artifactId=esup-formation`
- `package=org.esupportail.formation`
- `version=0.0.1-SNAPSHOT`

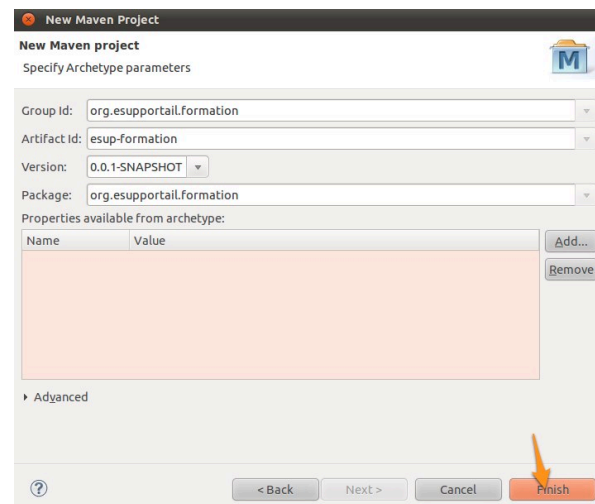
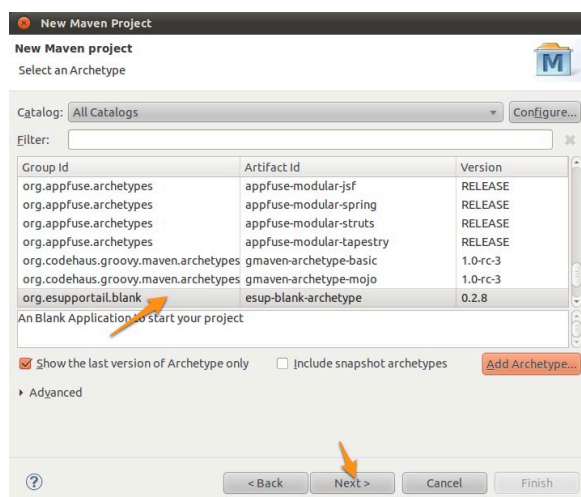
 **File > new > project > Project... > Maven > Maven project**



Add Archetype



Création du projet



Soit l'équivalent en ligne de commande :

```
mvn archetype:generate -B
-DarchetypeRepository=https://mvn.esup-
portail.org/content/repositories/releases
-DarchetypeGroupId=org.esupportail.blank
-DarchetypeArtifactId=esup-blank-archetype
-DarchetypeVersion=0.2.8
-DgroupId=org.esupportail.formation
-DartifactId=esup-formation
-Dpackage=packageMonProjet
-Dversion=0.0.1-SNAPSHOT
```

puis...

```
mvn eclipse:eclipse
```

On constate alors qu'*eclipse* fait un *Build maven dependencies* et un *Build maven project*

Eclipse a alors construit plusieurs projets. Il s'agit en fait d'un projet racine, ici *esup-formation* et de l'ensemble de ses modules, ici *esup-formation-xxx*.

```

esup-formation
esup-formation-dao
esup-formation-domain-beans
esup-formation-domain-services
esup-formation-utils
esup-formation-web-jsf-mixed
esup-formation-web-jsf-servlet
esup-formation-web-springmvc-portlet
esup-formation-web-springmvc-servlet

```

Le projet est alors complètement autonome et déconnecté du core *esup-commons* et de SVN. *Maven* ne sert qu'à gérer les dépendances. Ce projet pourra ensuite être partagé via SVN ou être lui-même déposé sur un repository *maven* (pour d'autres projets qui en dépendraient) et même potentiellement devenir archetype *maven* (modèle de projet).

2 Un coup de ménage...

2.1 Premier lancement

Sur le projet racine :

```
mvn install
```

☞ Cliquez droit sur le projet racine *esup-formation* > Run As > 6 Maven Install

```

[INFO] Reactor Summary:
[INFO]
[INFO] esup-formation ..... SUCCESS
[0.933s]
[INFO] esup-formation-domain-beans ..... SUCCESS
[4.283s]
[INFO] esup-formation-dao ..... SUCCESS
[1.562s]
[INFO] esup-formation-utils ..... SUCCESS
[0.580s]
[INFO] esup-formation-domain-services ..... SUCCESS
[0.961s]
[INFO] esup-formation-web-jsf-mixed ..... SUCCESS
[14.104s]
[INFO] esup-formation-web-jsf-servlet ..... SUCCESS
[7.868s]
[INFO] esup-formation-web-jsf-mobile ..... SUCCESS
[8.539s]
[INFO] esup-formation-web-jsf-portlet ..... SUCCESS
[8.928s]
[INFO] esup-formation-web-springmvc-servlet ..... SUCCESS
[4.476s]
[INFO] esup-formation-web-springmvc-portlet ..... SUCCESS
[10.112s]
[INFO] -----
[INFO] BUILD SUCCESS

```

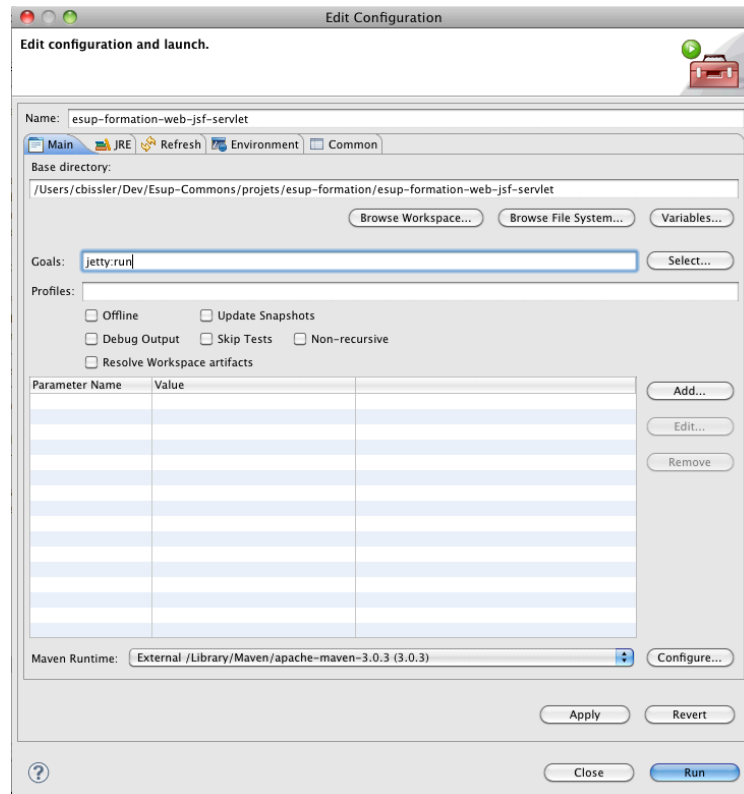
Si la trace est un peu différente et fait référence à des modules *unnamed*, c'est parce que sur la machine virtuelle on utilise *Maven 2* qui recommande de préciser une balise `<name>` dans le `pom.xml`. En *Maven 3*, ce n'est plus nécessaire

Nous allons commencer nos premiers tests sur une application de type *servlet* traditionnelle.

Sur le module `web-jsf-servlet`.

```
mvn jetty:run
```

- ☞ **Clique droit sur le projet web-jsf-servlet > Run As > 5 Maven build...**
- ☞ **Saisir un nom de tâche et goals : jetty:run**

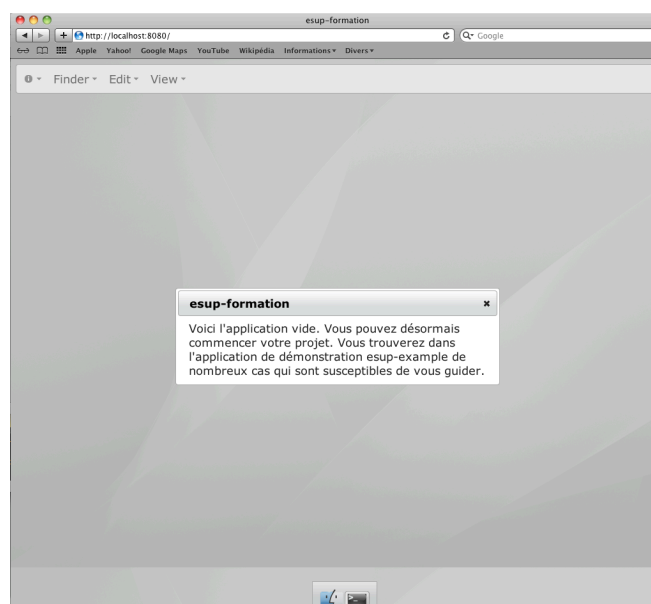


Vérifier que cela fonctionne

```
[INFO] Started Jetty Server
```

Et lancer dans un navigateur

☞ <http://localhost:8080/>



2.2 Ménage dans les bibliothèques JSF

Stopper le serveur.

Dans le module `web-jsf-servlet` le fichier `/src/main/webapp/stylesheets/welcome.xhtml` ôter toutes les déclarations et les utilisations des bibliothèques *primefaces*, *tomahawk* (corps + déclaration) et utiliser plutôt les balises *JSF* standard.

On ôte donc les déclarations des bibliothèques *tomahawk*, *primefaces*, *esupportail* :

- `http://primefaces.prime.com.tr/ui`
- `http://commons.esup-portail.org`
- `http://myfaces.apache.org/tomahawk`

On laisse les déclarations des bibliothèques *JSF* standard :

- `http://java.sun.com/jsf/core`
- `http://java.sun.com/jsf/facelets`
- `http://java.sun.com/jsf/html`

Faire de même dans `template.xhtml`, `_include/_header.xhtml` et `exception/exception.xhtml`

Oter la balise *primefaces* dans `exception.xhtml` en remplaçant...

```
<p:commandButton value="#{msgs['_BUTTON.BACK_WELCOME']}"
action="#{exceptionController.restart}" />
```

... par...

```
<h:commandButton value="#{msgs['_BUTTON.BACK_WELCOME']}"
action="#{exceptionController.restart}" />
```

Oter la balise *tomahawk* dans `template.xhtml` en remplaçant ...

```
<t:stylesheet path="#{path}" />
```

... par...

```
<link rel="stylesheet" type="text/css" href="#{path}"/>
```

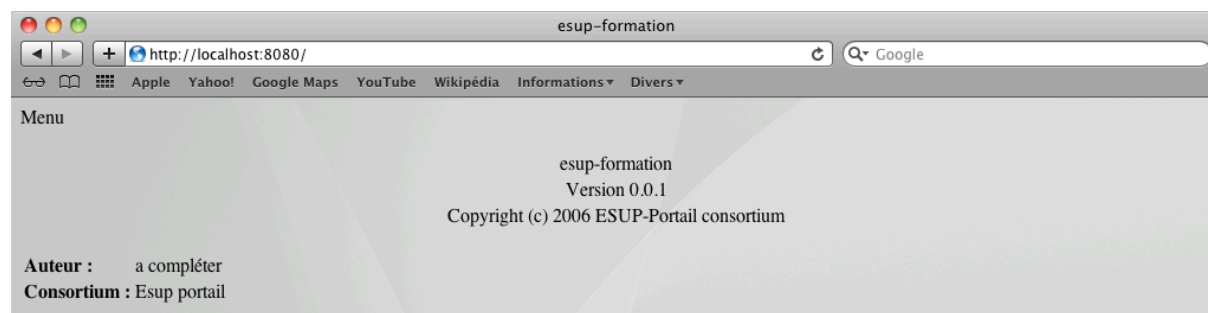
Sur le module `web-jsf-servlet`

```
mvn install jetty:run
```

Eventuellement, faire une nouvelle tâche :

- ☞ **Clique droit sur le projet `esup-formation-web-jsf-servlet` > Run As > 5 Maven build...**
- ☞ **Saisir un nom de tâche et goals : `install jetty:run`**

Vérifier que cela fonctionne.



2.3 Ménage des modules inutiles

Enfin, supprimer les modules dont nous ne nous servons pas au cours de cette formation à savoir :

- esup-formation-web-jsf-mixed
- esup-formation-web-springmvc-servlet
- esup-formation-web-springmvc-portlet

Puis dans **pom.xml** du projet racine esup-formation, ôter ou commenter les modules fils correspondants :

```
<modules>
  <module>esup-formation-dao</module>
  <module>esup-formation-utils</module>
  <module>esup-formation-domain-beans</module>
  <module>esup-formation-domain-services</module>
  <!-- <module>esup-formation-web-jsf-mixed</module> -->
  <module>esup-formation-web-jsf-servlet</module>
  <!-- <module>esup-formation-web-springmvc-servlet</module> -->
  <!-- <module>esup-formation-web-springmvc-portlet</module> -->

</modules>
```

Sur le projet racine :

```
mvn install
```

Beans Spring

1 Le fichier de configuration principal

Spring permet de créer des objets (appelés alors *beans*) en les déclarant dans un fichier de configuration *XML*.

Le fichier de configuration principal est déclaré dans le fichier **web.xml** sous forme d'un paramètre de l'application. Il se trouvera donc dans le module de vue qui sera lancé.

Ouvrir par exemple dans le module `web-jsf-servlet` le fichier **src/main/resources/properties/applicationContext.xml**

Dans *esup-commons* ce fichier de configuration principal contient seulement des inclusions de fichiers de configurations spécialisés par domaine

L'import des différents fichiers

```
<import resource="i18n/i18n.xml" />
<import resource="smtp/smtp.xml" />
```

Cas des fichiers embarqués dans les modules sous-jacents

Afin de limiter la duplication des fichiers de configuration dans chaque module de vue. On peut décider de positionner des fichiers de configuration directement au niveau du module qui les utilise.

```
<import resource="classpath*:META-INF/esup-formation-domain-
services-auth.xml" />
<import resource="classpath*:META-INF/esup-formation-domain-
services-domain.xml" />
```

On retrouvera par exemple ces fichiers dans le module `domain-services` et plus précisément dans **src/main/resources/META-INF** afin qu'ils se retrouvent à la racine du fichier *jar* généré à la compilation du module.

2 L'injection

Parcourir les différents fichiers de configuration.

Injection d'une chaîne de caractères

Exemple :

```
<property name="recipientEmail" value="webmaster@domain.edu"/>
```

Injection d'un autre bean

Exemple :

```
<property name="authenticationService" ref="authenticationService"/>
```

Injection d'une liste

Exemple :

```
<property name="servers">
```

```
<list>
  <ref bean="smtpServer1" />
  <ref bean="smtpServer2" />
</list>
</property>
```

Externalisation dans un fichier

Ouvrir le fichier `src/main/resources/properties/smtp/smtp.xml` on remarque des paramètres externalisés par l'intermédiaire d'une variable.

Exemple :

```
<property name="interceptAll" value="${smtp.interceptAll}" />
```

La variable `smtp.interceptAll` sera renseignée dans un fichier de propriété distinct. Ce mécanisme permet de simplifier la tâche des exploitants, qui pourront configurer une application directement en éditant des fichiers de propriétés, plus simples que des fichiers XML de configuration *spring*.

Dans le module `web-jsf-servlet` le fichier `src/main/resources/applicationContext.xml` on trouve :

```
<bean id="propertyConfigurer"
      class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
  <property name="locations">
    <list>
      <value>classpath:/properties/defaults.properties</value>
      <value>classpath:/properties/config.properties</value>
    </list>
  </property>
</bean>
```

L'utilisation du ***PropertyPlaceholderConfigurer*** de *spring* propose un mécanisme de surcharge des fichiers de propriétés. On voit ainsi dans l'exemple l'utilisation d'un fichier `/properties/defaults.properties` qui contiendra la configuration par défaut de l'application. Il suffira alors de surcharger dans le fichier `/properties/config.properties` les paramètres qui ne conviennent pas du fichier `/properties/defaults.properties`.

— — — — —
 | *On constate que config.properties est vide. En effet, par défaut une application*
 | *vierge utilise les paramètres par défaut.*

Exercice N°4 : Instanciation d'un bean simple

L'objectif de cet exercice est de créer un bean simple et d'y injecter les valeurs des attributs grâce à *spring*.

Dans `domain-beans` ajouter le bean ***Task***

```
package org.esupportail.formation.domain.beans;

public class Task{
  private boolean publicTask;

  private String title;

  private String description;

  public String getTitle() {
    return title;
  }
}
```

```

public void setTitle(String title) {
    this.title = title;
}
public String getDescription() {
    return description;
}
public void setDescription(String description) {
    this.description = description;
}

public boolean isPublicTask() {
    return publicTask;
}
public void setPublicTask(boolean publicTask) {
    this.publicTask = publicTask;
}
}

```

Récupérer le fichier Task.java-exo4

Déclarer une instance de ce bean dans **src/main/resources/properties/beans.xml**

```

<bean id="task1"
      class="org.esupportail.formation.domain.beans.Task"
      scope="session">
  <property name="title" value="titre 1"/>
  <property name="publicTask" value="true"/>
  <property name="description" value="description blabla"/>
</bean>

```

Importer **beans.xml** dans **src/main/resources/properties/applicationContext.xml**

```
<import resource="web/beans.xml" />
```

Pour l'exemple, injecter ce bean dans *sessionController* du module web-jspf-servlet.

Pour cela, ajouter une propriété de type *Task* à l'objet *sessionController* ainsi que ses accesseurs.

```

private Task task;
[...]
public Task getTask() {
    return task;
}
public void setTask(Task task) {
    this.task = task;
}

```

Injecter par référence *task1* dans le *sessionController* par configuration *Spring*

Dans **src/main/resources/properties/web/controllers.xml**

```

<bean id="sessionController"
      class="org.esupportail.formation.web.controllers.SessionController"
      parent="abstractDomainAwareBean" scope="session">
  [...]
    <property name="task" ref="task1">
      <description>The task</description>
    </property>
</bean>

```

Faire afficher ce bean dans la méthode *afterPropertiesSetInternal()* de *sessionController*

```

import org.esupportail.formation.domain.beans.Task;
import org.esupportail.commons.utils.BeanUtils;

```



```
[...]
public class SessionController extends AbstractDomainAwareBean {
[...]
    @Override
    public void afterPropertiesSetInternal() {
        [...]
        System.out.println("Test injection => "+task.getTitle()+
"+task.getDescription());
    }
[...]
```

Sur le projet racine :

```
mvn install
```

Sur le module web-jsf-servlet.

```
mvn jetty:run
```

Se connecter sur l'application pour instancier le *SessionController*. Pour cela, lancer dans un navigateur

☞ <http://localhost:8080/>

On trouvera alors dans les logs :

```
Test injection => titre 1 description blabla
```

3 Accès aux paramètres de configuration

Exercice N°5 : Personnalisation des configurations grâce à l'injection

Dans cet exercice nous allons observer le fonctionnement global du paramétrage d'une application et modifier pour l'exemple la configuration du système d'authentification (qui sera vu plus tard) grâce aux mécanismes d'injection *spring*.

Dans le module `domain-services` vérifier dans le fichier `src/main/resources/META-INF/esup-formation-domain-services-auth.xml` que la méthode d'authentification est bien « offline » (utilisation de la classe *org.esupportail.commons.services.authentication.OfflineFixedUserAuthenticationService* au niveau de *authenticator*)

Changer le login qui sera positionné en dur par défaut et le type d'authentification qui sera simulé.

Exemple :

```
<bean id="OfflineFixedUserAuthenticationService"
    class="org.esupportail.commons.services.authentication.OfflineFixedUserAuthenticationService">
    <property name="authId" value="cbissler" />
    <property name="authType" value="application" />
</bean>
```

Faire afficher la version de l'application dans la méthode *afterPropertiesSetInternal()* de *sessionController* du module `web-jsf-servlet`.

On constate que dans `src/main/resources/properties/misc/application.xml` on peut paramétrer la version de l'application.

Changer le numéro de la version.

```

        <property name="versionMajorNumber" value="0" >
            <description>
                The major number of the application (1 for version
1.2.3).
            </description>
        </property>
        <property name="versionMinorNumber" value="0" >
            <description>
                The minor number of the application (2 for version
1.2.3).
            </description>
        </property>
        <property name="versionUpdate" value="2" >
            <description>
                The update of the application (3 for version
1.2.3).
            </description>
        </property>

```

Ensuite dans **sessionController** du module `web-jsf-servlet` on constate que *esup-commons* fournit par héritage la méthode `getApplicationService()`.

```

public void afterPropertiesSetInternal() {
    [...]
    System.out.println("Test Version =>
"+getApplicationService().getVersion().toString()); }

```

`getVersion()` fournira bien une concaténation des 3 numéros passés en configuration :

```

public Version getVersion() {
    return new Version(versionMajorNumber + "." +
versionMinorNumber + "." + versionUpdate);
}

```

Sur le module `web-jsf-servlet`.

```
mvn jetty:run
```

Se connecter sur l'application pour instancier le **SessionController**. Pour cela, lancer dans un navigateur

 <http://localhost:8080/>

On trouvera alors dans les logs :

```
Test Version => 0.0.2
```

Nous avons fait ceci à titre d'exemple. Supprimer la déclaration du bean dans **beans.xml** et sa récupération dans **SessionController**. Eventuellement, remettre le numéro de version 0.0.1. Nous ne garderons que la classe **Task** et l'authentification *offline* pour la suite.

Gestion des Logs et tests unitaires

1 Gestion des logs

Esup-commons utilise la librairie standard *commons-logging* d'Apache qui permet d'utiliser différents mécanismes de log (standard Java, Log4j, etc.).

1.1 Utilisation dans le code Java

Pour pouvoir utiliser un logger dans une classe de votre application vous devez le définir. Suivant le cas, vous aurez deux définitions. Exemple :

```
private static final Logger LOGGER = new  
LoggerImpl(NonClasse.class);
```

ou

```
private final Logger logger = new LoggerImpl(getClass());
```

Le premier exemple est adapté à l'utilisation d'un logger à l'intérieur d'une classe utilitaire constituée de méthodes définies static. Le second exemple est adapté aux classes dynamiques. Dans ce cas l'utilisation de `getClass()` permet d'avoir une information sur la classe concrète utilisée. C'est particulièrement utile en cas d'héritage de classes.

Ensuite vous pouvez utiliser ce logger dans vos méthodes pour logger en mode TRACE, DEBUG, INFO, WARN ou ERROR. Exemple :

```
logger.error("Nous avons un problème");
```

Afin de ne pas pénaliser les performances avec la gestion des logs en mode DEBUG et TRACE il est conseillé de tester leur activation. Exemple :

```
if (logger.isDebugEnabled()) {  
    logger.debug("set language " + locale + " for user '" +  
        currentUser.getId() + "'");  
}
```

1.2 Activation du mécanisme de log

Le mécanisme de gestion des logs est configuré dans le fichier **src/main/resources/log4j.properties** du module `web-jsf-servlet`

Dans **config.properties** paramétrer la gestion des logs à votre convenance.

Ajouter par exemple :

```
log.level=DEBUG  
log.output=stdout
```

2 Les test unitaires

Les tests unitaires sont (ou devraient être !) une partie importante de toute application *Java*.

2.1 Execution des tests unitaires via *Maven*

Maven intègre complètement les tests unitaires dans le cycle de développement.

Pour exécuter l'ensemble des tests unitaires, on appelle la phase *test* du cycle de vie :

```
mvn test
```

Exercice N°6 : Test unitaire simple

Afin de voir comment *maven* intègre la gestion des tests unitaires nous allons mettre en place un test unitaire simple sur le bean *Task*.

Dans le fichier **pom.xml** du module `domain-beans` ajouter la dépendance vers *JUnit*.

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.8.2</version>
  <type>jar</type>
  <scope>test</scope>
</dependency>
```

☞ **clique droit sur le projet > Maven > Update dépendances**

On notera l'utilisation du scope *test* qui indique à *maven* que cette librairie ne sera utilisée que pour les tests unitaires du module. De ce fait, il est impossible de généraliser l'utilisation de la librairie *JUnit* en déclarant la dépendance au niveau du module racine de l'application puisque la portée reste dans les limites du module.

Puis créer le répertoire `src/test/java` (convention maven)

☞ **clique droit sur java > Build path > Build as source folder**

Créer une classe de test (*testCase*) sur *Task*

☞ **clique droit sur task > new > JUnit Test Case**

☞ **Attention ! changer le répertoire source proposé par défaut**

☞ **Next > Choisir une ou deux méthodes**

Laisser la classe proposée par défaut (qui provoque un échec à chaque fois)

```
package org.esupportail.formation.domain.beans;

import static org.junit.Assert.*;
import org.junit.Test;

public class TaskTest {

    @Test
    public void testSetTitle() {
        fail("Not yet implemented");
    }

    @Test
    public void testGetDescription() {
        fail("Not yet implemented");
    }

}
```

Lancer

```
mvn test
```

🔗 Cliquez droit sur le module > Run as > Maven Test

On constate alors dans la console :

```
Running org.esupportail.formation.domain.beans.TaskTest
[...]
Failed tests:
  testSetTitle(org.esupportail.formation.domain.beans.TaskTest): Not
  yet implemented
[...]
Tests run: 2, Failures: 2, Errors: 0, Skipped: 0
[...]
Failed to execute goal org.apache.maven.plugins:maven-surefire-
plugin:2.7.2:test (default-test) on project esup-formation-domain-
beans: There are test failures.
```

Et si on tente de compiler ou lancer l'application on obtient la même erreur :

- Sur le module ou sur le projet racine

```
mvn install
```

- Sur le module web-jsf-servlet

```
mvn jetty:run
```

En effet, *maven* exige que l'ensemble des tests unitaires soit validé avant de poursuivre.

Ajouter maintenant un test qui passe (ou qui peut passer pour être exact !)

```
@Test
public void testSetTitle() {
    Task t=new Task();
    t.setTitle("test titre");
    assertEquals("test titre",t.getTitle());
}
```

Et là ça passe

```
Running org.esupportail.formation.domain.beans.TaskTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed:
0.051 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

Il est toujours possible d'indiquer explicitement à *maven* d'ignorer les tests

- En ligne de commande

```
-Dmaven.test.skip=true
```

ou

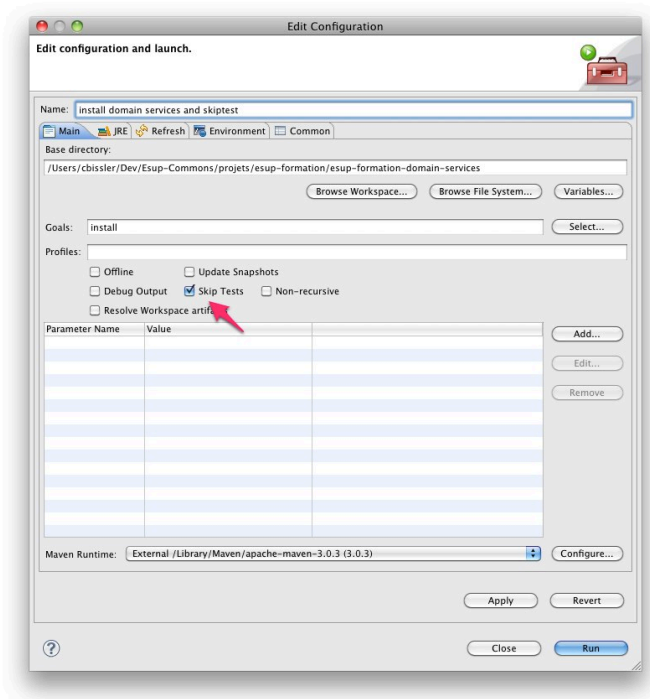
```
-DskipTests=true
```

- Dans le fichier **pom.xml**

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-surefire-plugin</artifactId>
      <configuration>
        <skip>true</skip>
      </configuration>
    </plugin>
  </plugins>
```

```
</build>
```

- Dans *eclipse*
- ☞ Fenêtre Run > Run configurations > cocher *Skip Test*



Exercice N°7 : Test unitaire avancé

On souhaite par exemple tester que l'authenticator injecté est bien de la classe *AuthenticatorImpl*. Ce test est complètement absurde dans un cas réel mais il va nous permettre de voir comment lancer des tests en mode batch (hors conteneur *J2EE*) dépendant d'une arborescence de fichiers de configurations traditionnellement gérés par le conteneur *J2EE* dans un contexte web.

Démarrer comme l'exercice précédent dans le module `domain-services`

Créer une classe de test sur *Authenticator*

```
package org.esupportail.formation.domain;

import org.esupportail.formation.services.auth.Authenticator;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;

[...]
public class AuthenticatorTest {
    @Autowired
    private Authenticator authenticator;

    @Test
    public void testSetAuthenticationService() {

        Assert.assertEquals("org.esupportail.formation.services.auth.AuthenticatorImpl", authenticator.getClass().getName());
    }
}
```

Dans **src/test/resources/META-INF** (à créer) créer un fichier de configuration spring similaire à **applicationContext.xml** (**testApplicationContext.xml** par exemple) qui contiendra :

```
<?xml version="1.0" encoding="UTF-8" ?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
  http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">

<import resource="classpath*:META-INF/esup-formation-domain-
services-auth.xml" />
<import resource="classpath*:META-INF/esup-formation-domain-
services-domain.xml" />
</beans>
```

Si ces fichiers importés utilisent des paramètres externalisés dans un fichier de propriétés, copier/coller ces fichiers **config.properties** et **default.properties** ou créer un fichier minimal contenant uniquement les propriétés nécessaires.

Il faudra alors ajouter un *propertyConfigurer*

```
<bean id="propertyConfigurer"
  class="org.springframework.beans.factory.config.PropertyPlaceho
lderConfigurer">
<property name="locations">
  <list>
    <value>classpath:META-INF/defaults.properties</value>
    <value>classpath:META-INF/config.properties</value>
  </list>
</property>
</bean>
```

Simuler enfin le travail du contexte d'application en y ajoutant les fichiers de configurations nécessaires.

Cela se traduira de la façon suivante dans la classe de test

```
[...]
import org.springframework.test.context.ContextConfiguration;
import
org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations="classpath*:META-
INF/testApplicationContext.xml")
public class AuthenticatorTest {
[...]
```

Pour aller plus loin...

On pourra, pour le test vérifier la classe d'implémentation de l'*authenticationService* est bien **OfflineFixedUserAuthenticationService**

Pour cela, on déclarera *getAuthenticationService()* de **Authenticator** comme méthode *public* et non plus *protected* :

```
public AuthenticationService getAuthenticationService() {
  return authenticationService;
}
```

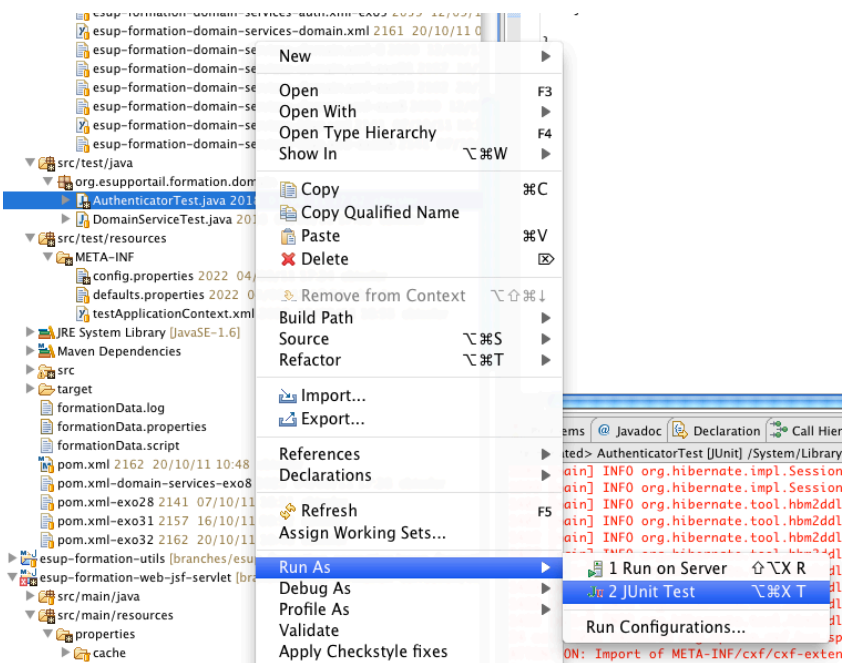
... et on testera :

```
Assert.assertEquals("org.esupportail.commons.services.authentication
.OfflineFixedUserAuthenticationService", authenticator.getAuthenticat
ionService().getClass().getName());
```

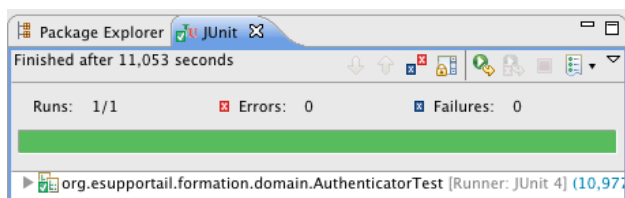
2.2 Execution des tests unitaires dans *Eclipse*

Eclipse permet de lancer directement un test unitaire grâce à un :

☞ **Clique droit sur la classe de test > Run as > JUnit Test**



Le résultat est alors visuel...



Accès aux données

Exercice N°8 : Création d'un objet métier simple.

L'objectif de ce premier exercice est de créer l'objet métier représentant une tâche et de mettre en place les mécanismes d'écritures et lecture en base de donnée.

Pour cela on utilisera une base de donnée embarquée *HSQL*.

1 L'objet métier

Créer un objet métier qui sera mappé en base.

Dans le **pom.xml** du module `domain-beans` ajouter la dépendance vers le module *JPA* de *esup-commons*.

```
<dependency>
  <groupId>org.esupportail</groupId>
  <artifactId>esup-commons2-jpa</artifactId>
  <version>${esupcommons.version}</version>
  <scope>provided</scope>
</dependency>
```

Dans `domain-beans` adapter le bean **Task** créé précédemment avec :

- `id` (long généré en base) : Identifiant
- `title` (chaîne obligatoire) : Titre
- `date` (type date) : Date de limite d'exécution de la tâche
- `description` (type chaîne) : Description de la tâche
- `publicTask` (type boolean) : Booléen précisant si la tâche est publique ou pas

Ajouter 2 requêtes de récupération des tâches :

- `allTasks` : pour récupérer toutes les tâches.
- `publicTasks` : pour récupérer les tâches publiques.

 **Récupérer le fichier Task.java-exo8**

2 La couche DAO

Créer une classe d'accès DAO **JPADaoServiceImpl** qui étendra **org.esupportail.commons.dao.AbstractGenericJPADaoService** ainsi que son interface **DaoService** qui devra se présenter comme suit :

```
package org.esupportail.formation.dao;

import java.io.Serializable;
import java.util.List;

import org.esupportail.formation.domain.beans.Task;

/**
 * The DAO service interface.
```

```

    */
public interface DaoService extends Serializable {

    /**
     * Get all public task.
     */
    public List<Task> getPublicTasks() ;
    /**
     * Get all task.
     */
    public List<Task> getTasks();

    /**
     * Add a task.
     * @param task
     */
    void addTask(Task task);

    /**
     * Delete a task.
     * @param task
     */
    void deleteTask(Task task);

    /**
     * Update a task.
     * @param task
     */
    Task updateTask(Task task);
    /**
     * @param id
     * @return the Task instance that corresponds to an id.
     */
    Task getTask(long id);
}

```

On ajoute ensuite les fichiers de configuration nécessaires **dao-dao.xml** et **dao-persistence.xml**

Pour cela 2 façon de faire :

- Soit **dans le module de vue** qui sera lancé (ex : `web-jsf-servlet`) à faire à chaque module de vue s'il y en a plusieurs (ex : une vue *portlet* et une vue *servlet*)
- Soit **dans le module dao** directement de manière à avoir la configuration dans le jar généré par *maven* et s'assurer d'avoir la même choses quel que soit le module de vue lancé.

On constate dans **dao-dao.xml** que les paramètres sont passés par l'intermédiaire de variables qui seront positionnées dans le fichier de propriétés.

 | *Esup-commons préconise de localiser les fichiers de configuration spring au plus*
 | *près de leur utilisation c'est à dire au niveau du module.*

Copier/coller/renommer respectivement les fichiers **dao.xml** et **persistence.xml** vers **src/main/resources/META-INF** (à créer) en **esup-formation-dao-dao.xml** et **esup-formation-dao-persistence.xml**

Dans **dao-dao.xml** vérifier le chemin vers **dao-persistence.xml**:

```

<property name="persistenceXmlLocation" value="classpath*:META-INF/
esup-formation-dao-persistence.xml" />

```

On peut d'ores et déjà prévoir que la couche *dao* sera appelée par la couche *domain-services*. On devra donc ajouter l'import de la *dao* dans le fichier **src/main/resources/META-INF/esup-formation-domain-services-domain.xml** du module *domain-services* de la façon suivante

```
<import resource="classpath*:META-INF/esup-formation-dao-dao.xml" />
```

Pour cela

 **Récupérer les fichiers dao-dao.xml-exo8 et dao-persitence.xml-exo8**

Enfin sur le module *dao* faire

```
mvn install
```

3 La couche Services

La couche *service* va faire appel à la couche *dao* pour récupérer les tâches en base.

Dans le fichier **pom.xml** du module *domain-services* ajouter la dépendance vers le module *dao*

```
<dependency>
  <groupId>org.esupportail.formation</groupId>
  <artifactId>esup-formation-dao</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <type>jar</type>
</dependency>
```

Ajouter dans **DomainService** et son implémentation **DomainServiceImpl** l'attribut *daoService* et les accesseurs correspondants

```
private DaoService daoService;
```

En cas d'erreurs de reconnaissance des classes dao générer à nouveau le jar du module dao.

Ajouter les méthodes permettant de lister les tâches et d'ajouter une tâche.

```
public List<Task> getTasks() {
    return daoService.getTasks();
}

public void addTask(Task task) {
    Task tmp = daoService.getTask(task.getId());
    if (tmp == null) {
        // task does not already exists in database
        logger.debug("addTask -> not found "+task.getId());
        daoService.addTask(task);
    }
    else {
        daoService.updateTask(task);
        logger.debug("addTask -> found "+task.getId());
    }
}
```

Ajouter dans *afterPropertiesSet()* la vérification de l'injection correcte de l'objet **DaoService**.

```
Assert.notNull(this.daoService,
    "property daoService of class " +
    this.getClass().getName() + " can not be null");
```

Ajouter l'injection *spring* du *daoService* dans **META-INF/esup-formation-domain-services-domain.xml**

```
<bean id="domainService"
class="org.esupportail.formation.domain.DomainServiceImpl">
    <property name="daoService" ref="daoService" />
</bean>
```

Ainsi que *AOP* (aspect-oriented programming via *Spring*) pour gérer les connexions aux bases de données et les transactions.

```
<aop:config>
    <aop:pointcut id="domainMethods"
expression="execution(*
org.esupportail.*.domain.DomainServiceImpl.*(..))" />
    <aop:advisor advice-ref="txAdvice" pointcut-
ref="domainMethods" />
</aop:config>

<tx:advice id="txAdvice" transaction-manager="txManager">
    <tx:attributes>
        <tx:method name="add*" propagation="REQUIRED" />
        <tx:method name="delete*" propagation="REQUIRED" />
        <tx:method name="update*" propagation="REQUIRED" />
        <tx:method name="*" propagation="SUPPORTS" read-
only="true"/>
    </tx:attributes>
</tx:advice>
```

La balise **<aop:config>** permet de préciser deux choses :

- Le point de coupe (pointcut). C'est-à-dire l'emplacement où va être inséré dynamiquement le code AOP. L'attribut *expression* permet de préciser que l'on va cibler toutes les méthodes, quelle qu'en soit la signature, de la classe *DomainServiceImpl* dans un package donc le path commence par *org.esupportail* et fini par *domain*.
- La référence au gestionnaire de transactions à brancher sur ce point de coupe.

La balise **<tx:advice>** permet de préciser la nature de la transaction.

Ici on précise, via *propagation="REQUIRED"* que l'on veut créer une transaction, si elle n'est pas encore présente, sur toutes les méthodes dont le nom commence par *add*, *delete* ou *update*.

Dans **src/main/resources/properties/config.properties** configurer la connexion à la base de données (cf. variables utilisées dans **dao-dao.xml**)

```
#####
###
# Database
#

jdbc.connection.driver_class=org.hsqldb.jdbcDriver
jdbc.connection.url=jdbc:hsqldb:file:formationData
jdbc.connection.username=sa
jdbc.connection.password=

# JNDI/JDBC
# for JDBC datasource.bean=JDBCDataSource
# for JNDI datasource.bean=JNDIDataSource
datasource.bean=JDBCDataSource

jpa.database.type=HSQL
```

Enfin, si cela n'a pas été fait ajouter l'import du fichier de propriétés DAO

```
<import resource="classpath*:META-INF/dao-dao.xml" />
```

4 Premiers tests d'écriture et lecture en base

Dans `src/main/java` du module `web-jsp-servlet` créer ***TaskController*** du package ***org.esupportail.formation.web.controllers***.

Le rôle du contrôleur sera abordé dans le chapitre suivant sur les vues

```
package org.esupportail.formation.web.controllers;
import java.util.Date;
import org.esupportail.formation.domain.beans.Task;

public class TaskController extends AbstractContextAwareController {

    /**
     * VersionId
     */
    private static final long serialVersionUID = -
872218604638760392L;

    public String getTestDao(){
        Task task=new Task();
        task.setDate(new Date());
        task.setTitle("titre de test");
        task.setPublicTask(true);
        getDomainService().addTask(task);
        return "OK";
    }
}
```

Déclarer le contrôleur dans `src/main/resources/properties/web/controllers.xml`

```
<bean id="taskController"
      class="org.esupportail.formation.web.controllers.TaskController"
      parent="abstractContextAwareController"
      scope="session"/>
```

Et l'appeler dans `src/main/webapps/stylesheets/welcome.xhtml` en ajoutant un

```
<h:outputText value="#{taskController.testDao}" />
```

Sur le projet racine :

```
mvn install
```

Sur le module `web-jsp-servlet`

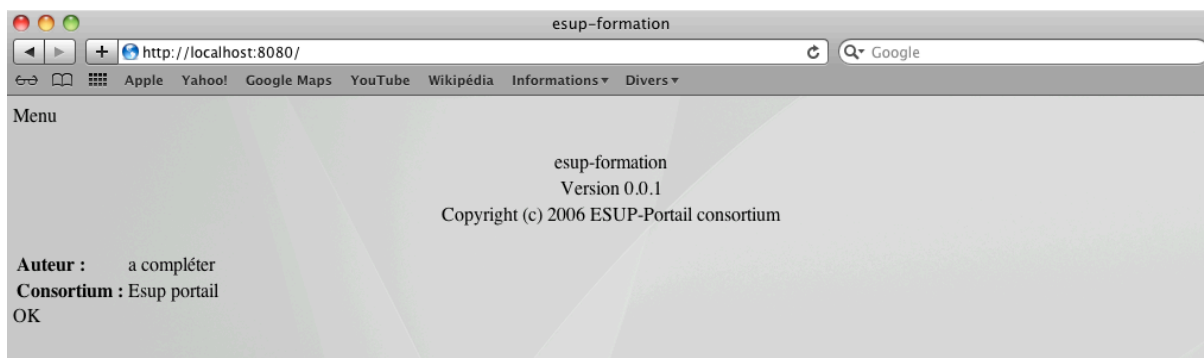
```
mvn jetty:run
```

Vérifier que cela fonctionne

```
[INFO] Started Jetty Server
```

et

☞ <http://localhost:8080/>



"OK" s'affiche la méthode a été appelée

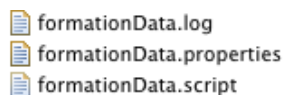
Dans la console on a ...

```

Hibernate: select task0_.id as id0_0_, task0_.date as date0_0_,
task0_.description as descript3_0_0_, task0_.publicTask as
publicTask0_0_, task0_.title as title0_0_ from Task task0_ where
task0_.id=?
Hibernate: insert into Task (id, date, description, publicTask,
title) values (null, ?, ?, ?, ?)
Hibernate: call identity()
    
```

... ce qui semble dire que la tâche a été ajoutée.

Dans Eclipse faire un *refresh* sur `web-jsf-servlet`. On constate que 3 fichiers ont été créés à la racine du module :



Dans `formationData.script` on trouve le script de création de la base généré à partir de ce qu'on a déclaré par annotations *JPA*.

```

CREATE SCHEMA PUBLIC AUTHORIZATION DBA
CREATE MEMORY TABLE TASK(ID BIGINT GENERATED BY DEFAULT AS
IDENTITY(START WITH 1) NOT NULL PRIMARY KEY,DATE
TIMESTAMP,DESCRIPTION VARCHAR(255),PUBLICTASK BOOLEAN NOT NULL,TITLE
VARCHAR(255) NOT NULL)
ALTER TABLE TASK ALTER COLUMN ID RESTART WITH 2
CREATE USER SA PASSWORD ""
GRANT DBA TO SA
SET WRITE_DELAY 10
SET SCHEMA PUBLIC
    
```

En revanche, il ne contient pas les données.

Si on stoppe jetty, on n'a toujours pas de données dans ce fichier.

On redémarre et là on a

```

INSERT INTO TASK VALUES(1,'2011-07-27
13:17:27.957000000',NULL,TRUE,'titre de test')
    
```

En fait, toutes les actions faites en base sont d'abord conservées temporairement dans un fichier de log `formationData.log`. A chaque démarrage les logs sont rejoués et l'état de la base est alors enregistré dans `formationData.script`.

Ajouter la récupération de l'ensemble des tâches en base dans la méthode `getTestDao()` et le lister dans la console

```

List<Task> liste=getDomainService().getTasks();
for (Task t : liste) {
    
```

```
        System.out.println(t.getId()+" : "+t.getTitle());
    }
```

On doit alors retrouver les lignes suivantes

```
Hibernate: select task0_.id as id0_, task0_.date as date0_,
task0_.description as descript3_0_, task0_.publicTask as
publicTask0_, task0_.title as title0_ from Task task0_
1 : titre de test
2 : titre de test
```

Exercice N°9 : Création d'une relation entre objets métiers

Par défaut, on trouve dans le module *domain-beans* la classe *User*. Nous allons maintenant gérer ce type d'objet en base et ajouter une relation avec l'objet *Task*. Il s'agira donc d'ajouter la propriété *owner* qui sera le *User* qui a créé la tâche.

Pour aller plus loin :

- Ajouter et tester la modification d'une Tâche
- Ajouter et tester la suppression d'une Tâche
- Ajouter la recherche des tâches d'un user
- Faire en sorte que lorsqu'on supprime un User en base, l'ensemble de ses tâches soient supprimées également

Dans *User* :

- Ajouter *id* comme identifiant généré automatiquement grâce à l'annotation

```
@GeneratedValue
```
- Ne stocker que *id*, et *login* les autres propriétés seront ignorées grâce à

```
@Transient
```
- Ajouter le bean *User* dans **dao-persistence.xml**

Dans *Task*

- Ajouter la relation sur une propriété *User owner* en utilisant l'annotation

```
ManyToOne(optional = true)
```

Dans *domain-services* et *dao* :

- Ajouter les accès à la persistance des objets *User* (add, remove, etc.)

Tester en logant dans la console comme précédemment

Exercice N°10 : Test de la couche *domain* dans un test unitaire

Pour valider l'intégrité de cette couche d'accès aux données qui est amenée à changer souvent au cours du développement d'une application, on ajoutera quelques tests unitaires. (Exercice facultatif)

Créer un *testCase* sur *domainService* qui pourra ressembler à ça :

```
package org.esupportail.formation.domain;

import java.util.Date;
import java.util.List;

import junit.framework.Assert;

import org.esupportail.formation.domain.beans.Task;
import org.junit.Test;
import org.junit.runner.RunWith;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import
org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations="classpath*:META-
INF/testApplicationContext.xml")
public class DomainServiceTest {
    @Autowired
    DomainService domainService;

    @Test
    public void testAddTask() {
        Task t=new Task(true, "Test de tache", "Tache de test",
new Date());
        domainService.addTask(t);
        List<Task> listedestaches = domainService.getTasks();
        //on cherche la tâche en base
        Task task=null;
        for (int i = 0; i < listedestaches.size(); i++) {
            if(listedestaches.get(i).getTitle().equals("Test de
tache"))
                task=listedestaches.get(i);
        }
        Assert.assertNotNull("La tache de test créée en base n'a
pas été retrouvée" , task);
    }
}
```

Documentation JPA :

- <http://tahe.developpez.com/java/jpa/>
- <http://www.dil.univ-mrs.fr/~massat/ens/jee/tp-JPA.html>
- <http://docs.jboss.org/hibernate/annotations/3.5/reference/en/html/entity.html>

Les vues

Esup-commons laisse la liberté au développeur de choisir la technologie de vue qu'il souhaite :

- Spring MVC
- Java Faces : JSF

Ces dernières étant sensiblement différentes, nous ne les détaillerons pas dans cette séance les différentes technologies. Elles pourront faire l'objet de sessions de formation de second niveau spécialisées sur l'une ou l'autre de ces technologies.

1 JSF et ses librairies

Pour commencer et rester relativement proche de ce que l'on pouvait faire en *esup-commons V1* nous allons utiliser *JSF*. Cependant, nous nous limiterons aux librairies *JSF* standard et nous n'utiliserons pas les librairies plus évoluées telles que *tomahawk*, *primefaces* etc.

Dans les prochains chapitres nous allons essentiellement travailler sur le module de vue.

2 Facelet

Exercice N°11 : Ajout d'un menu via un template facelet

On souhaite ajouter un menu et qui sera fixe pour toutes les pages de notre application. Pour cela nous allons créer un template facelet que l'on chaînera sur le template proposé par défaut.

Se placer dans l'arborescence **src/main/webapps/stylesheets**

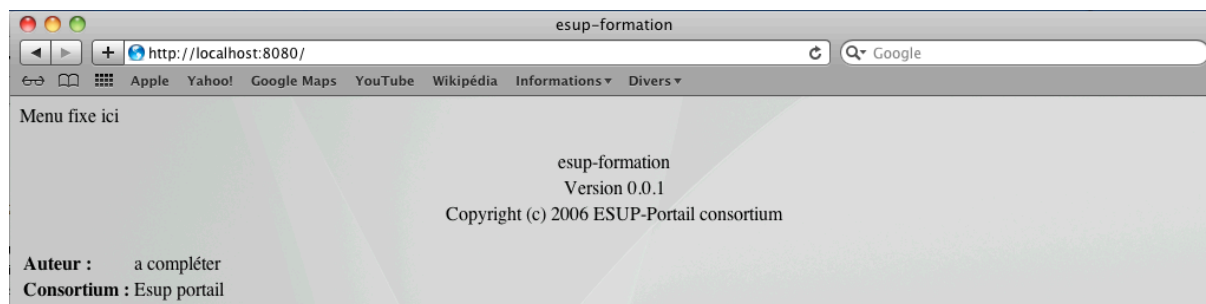
Dans créer un fichier **pageTemplate.xhtml** qui lui-même appellera **template.xhtml** et dans lequel on ne définira que le menu que l'on souhaite fixe

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<ui:composition xmlns:f="http://java.sun.com/jsf/core"
  xmlns:ui="http://java.sun.com/jsf/facelets"
  xmlns:h="http://java.sun.com/jsf/html"
  template="/stylesheets/template.xhtml">
  <ui:define name="navigation">
    Menu fixe ici
  </ui:define>
</ui:composition>
```

Dans **welcome.xhtml**

- supprimer la définition de navigation puisqu'elle est définie dans le template
- remplacer le template utilisé par **pageTemplate.xhtml**

Tester



3 Pages et navigation

Exercice N°12 : Ajout d'une nouvelle page avec règle de navigation

Nous allons créer un lien de redirection vers une nouvelle page : le Task Manager qui permettra de gérer les tâches.

Nous allons d'abord créer une nouvelle page *JSP* qui n'affichera rien (d'intéressant) pour l'instant.

Dans **src/main/webapps/stylesheets** créer un fichier **taskManager.xhtml** qui ressemblera à ceci :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<ui:composition xmlns:f="http://java.sun.com/jsf/core"
  xmlns:ui="http://java.sun.com/jsf/facelets"
  xmlns:h="http://java.sun.com/jsf/html"
  template="/stylesheets/pageTemplate.xhtml">
  <ui:define name="content">
    Ici le Task Manager !
  </ui:define>
</ui:composition>
```

Dans **src/main/java** du module **web-jsp-servlet** ajouter la méthode **doToTaskManager()** à **org.esupportail.formation.web.controllers.TaskController**.

Cette méthode retourne la chaîne « *go_taskManagerPage* »

```
public String goToTaskManagerPage(){
    return "go_taskManagerPage";
}
```

Enfin on ajoute la règle de navigation dans **src/main/webapps/WEB-INF/navigation-rules.xml**.

Cette déclaration doit préciser que lorsque la chaîne *go_taskManagerPage* est retournée par *JSP* il faut alors rediriger vers la page **taskManager.xhtml**

```
<navigation-rule>
  <description>Gestion des tâches</description>
  <navigation-case>
    <from-outcome>go_taskManagerPage</from-outcome>
    <to-view-id>/stylesheets/taskManager.xhtml</to-view-id>
    <redirect/>
  </navigation-case>
</navigation-rule>
```

Tester en ajoutant un lien sur la page d'accueil

Dans **src/main/webapps/stylesheets/welcome.xhtml** en ajoutant un lien

```
<h:form>
  <h:commandLink action="#{taskController.goToTaskManagerPage}">
    Task Manager
  </h:commandLink>
</h:form>
```

La balise `<h:form>` est obligatoire pour un `<h:commandLink>`. Nous verrons plus loin la création de formulaires.

Ne compiler que le module `web-jsp-servlet`

```
mvn install jetty:run
```

Vérifier que cela fonctionne

Pour aller plus loin faites en sorte que ce lien soit le premier onglet du menu de notre application.

Exercice N°13 : Parcours d'un tableau

Afficher la liste des tâches stockées en base et ajouter un tri par date grâce à l'utilisation d'objets comparateurs (qui implémente la classe *Comparable*).

Dans `src/main/java` du module `web-jsp-servlet` ajouter la méthode `getTasks()` à *org.esupportail.formation.web.controllers.TaskController*

```
private List<Task> sortedTasks=null;
[...]
public List<Task> getTasks() {
    if (this.sortedTasks==null)
        sortedTasks = getDomainService().getTasks();
    return sortedTasks;
}
```

Dans `src/main/webapps/stylesheets/taskManager.xhtml` on aura ceci :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<ui:composition xmlns:f="http://java.sun.com/jsf/core"
  xmlns:ui="http://java.sun.com/jsf/facelets"
  xmlns:h="http://java.sun.com/jsf/html"
  template="/stylesheets/pageTemplate.xhtml">
  <ui:define name="content">
    <h:dataTable var="task" value="#{taskController.tasks}">
      <h:column>
        <h:outputText value="#{task.id}"></h:outputText>
      </h:column>
      <h:column>
        <h:outputText value="#{task.name}"></h:outputText>
      </h:column>
      <h:column>
        <h:outputText value="#{task.date}"></h:outputText>
      </h:column>
    </h:dataTable>
  </ui:define>
</ui:composition>
```

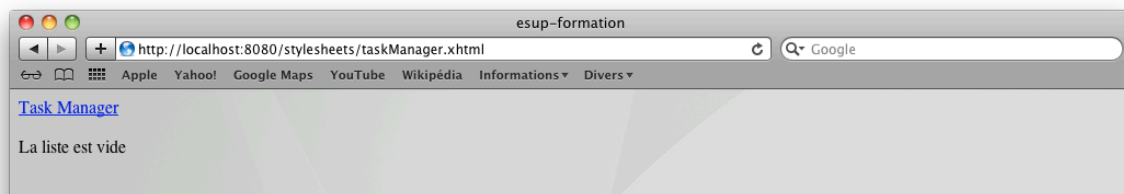
Améliorer le rendu ...

... en ajoutant les entêtes aux colonnes dans les balises `<h:column>`

```
<f:facet name="header">
  <h:outputText value="id" />
</f:facet>
```

...en testant si la liste contient des éléments dans le cas contraire on affichera un message :

```
rendered="#{taskController.tasks != null and !empty
taskController.tasks}"
```



... en ajoutant un bouton qui permet d'ajouter des tâches factices décrites en dur ainsi qu'un bouton qui supprime tout.

```
<h:form>
  <h:commandButton action="#{taskController.addFakeTask}"
value="Ajouter une tâche bidon"/>
  <h:commandButton action="#{taskController.supprimeTout}"
value="Supprimer toutes les tâches"/>
</h:form>
```

Et dans **src/main/java** de **web-jsp-servlet**, ajouter les méthodes *addFakeTask()* et *supprimerTout()* à **org.esupportail.formation.web.controllers.TaskController**

```
public void addFakeTask() {
    User user1=getDomainService().getUser("cbissler");
    if(user1 == null){
        user1=new User();
        user1.setLogin("cbissler");
        getDomainService().addUser(user1);
    }
    //On ajoute la tâche
    getDomainService().addTask(new Task(true, "titre de tache
bidon", "", new Date(), user1));
}
public void supprimeTout() {
    List<Task> listeTask=getDomainService().getTasks();
    for (Task t : listeTask) {
        getDomainService().deleteTask(t);
    }
    List<User> listeUser=getDomainService().getUsers();
    for (User u : listeUser) {
        getDomainService().deleteUser(u);
    }
}
```

... en triant les tâches par ordre alphabétique, date à l'aide de comparateurs

Pour cela dans le module **utils** ajouter la dépendance dans **pom.xml**

```
<dependencies>
  <dependency>
    <groupId>org.esupportail.formation</groupId>
    <artifactId>esup-formation-domain-beans</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <type>jar</type>
  </dependency>
</dependencies>
```

Sur le module **utils**

```
mvn install
```

Dans **src/main/java** du module `utils`, créer les classes ***TaskDateComparator*** et ***TaskTitleComparator*** à l'image de celle-ci :

```
package org.esupportail.formation.utils;

import java.util.Comparator;

import org.esupportail.formation.domain.beans.Task;

public class TaskDateComparator implements Comparator<Task> {
    public int compare(Task t1, Task t2)
    {
        int result = t1.getDate().compareTo(t2.getDate());
        //si la date est identique on trie ensuite par le titre
        if(result==0)
            result = t1.getTitle().compareTo(t2.getTitle());
        return result;
    }
}
```

Sur le module `utils` refaire

```
mvn install
```

Ensuite, il va falloir faire dépendre module `web-jsp-servlet` de `utils` en ajoutant dans le **`pom.xml`** du module `web-jsp-servlet`

```
<dependency>
    <groupId>org.esupportail.formation</groupId>
    <artifactId>esup-formation-utils</artifactId>
    <version>0.0.1-SNAPSHOT</version>
</dependency>
```

Dans **src/main/java** de `web-jsp-servlet` ajouter les méthodes `sortTasksByDate ()` et `sortTasksByTitle ()` à ***org.esupportail.formation.web.controllers.TaskController***

```
public void sortTasksByDate() {
    if (this.sortedTasks==null)
        sortedTasks = getDomainService().getTasks();
    Collections.sort(sortedTasks, new TaskDateComparator());
}
public void sortTasksByTitle() {
    if (this.sortedTasks==null)
        sortedTasks = getDomainService().getTasks();
    Collections.sort(sortedTasks, new TaskTitleComparator());
}
```

Enfin ajouter les boutons de tri à la vue

```
<h:commandButton action="#{taskController.sortTasksByTitle}"
value="titre" />
[...]
<h:commandButton action="#{taskController.sortTasksByDate}"
value="date" />
```

The screenshot shows a web browser window titled "esup-formation" with the address bar displaying "http://localhost:8080/stylesheets/taskManager.xhtml". The browser's search bar contains "Google". Below the browser window, the page content is titled "Task Manager" and features a table with the following data:

id	titre	date
45	fb - Titre de tache bidon	5 sept. 2011
46	ue - Titre de tache bidon	5 sept. 2011
47	bw - Titre de tache bidon	5 sept. 2011
48	re - Titre de tache bidon	5 sept. 2011
49	sh - Titre de tache bidon	5 sept. 2011
50	zi - Titre de tache bidon	5 sept. 2011
51	oz - Titre de tache bidon	5 sept. 2011
52	vw - Titre de tache bidon	5 sept. 2011
53	ew - Titre de tache bidon	5 sept. 2011
54	jd - Titre de tache bidon	5 sept. 2011
55	rv - Titre de tache bidon	5 sept. 2011
56	fù - Titre de tache bidon	5 sept. 2011

Below the table, there are two buttons: "Ajouter une tâche bidon" and "Supprimer toutes les tâches".

Internationalisation

1 Configuration

L'internationalisation est définie dans le fichier de configuration `src/main/resources/properties/i18n/i18n.xml`. On y trouvera par exemple :

```
<bean id="i18nService"
      class="org.esupportail.commons.services.i18n.ResourceBundleMessageSourceI18nServiceImpl">
    <property name="messageSource" ref="msgs" />
</bean>

<bean id="msgs"
      class="org.springframework.context.support.ReloadableResourceBundleMessageSource">
    <property name="basenames">
    <list>
        <value>classpath:properties/i18n/bundles/Custom</value>
        <value>classpath:properties/i18n/bundles/Messages</value>
        <value>classpath:properties/i18n/bundles/Commons</value>
    </list>
    </property>
    <property name="cacheSeconds" value="60" />
</bean>
```

2 Déclaration et utilisation des entrées

Exercice N°14 : Déclaration et utilisation des entrées

Adapter la vue `taskManager.xhtml` précédemment créée de manière à ce que tous les libellés de l'interface soient externalisés et non plus en « durs » dans les pages *JSP*.

2.1 Déclaration

2.1.1 Via un éditeur de texte

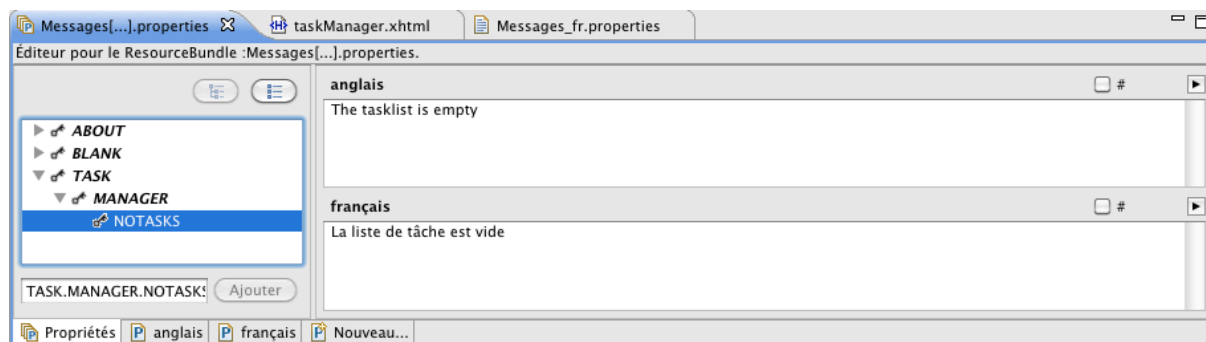
Par exemple nous allons ajouter par exemple l'entrée `TASK.MANAGER.NOTASKS` dans le fichier `bundles/Messages_fr.properties`.

```
TASK.MANAGER.NOTASKS = La liste de tâches est vide
```

2.1.2 Via `ResourceBundleEditor` dans eclipse

Editer `bundles/Messages_fr.properties` avec `resourceBundleEditor`

☞ Cliquez droit sur le fichier > open with... > éditeur de resourceBundle



2.2 Utilisation

2.2.1 Du côté de la vue

Dans `src/main/webapps/stylesheets/taskManager.xhtml` on déclarera les clés de traduction de la manière suivante :

```
<h:outputText value="#{msgs['TASK.MANAGER.NOTASKS']}"...
```

Changer la langue dans les préférences de votre navigateur et tester.

Et dans Firefox :

🔗 **Préférences > Contenus > Langues > Choisir...**

Tester

2.2.2 Du côté du code Java

Essayer maintenant d'envoyer directement une chaîne traduite depuis le contrôleur

```
public String getChaineTraduite(){
    return getString("TASK.MANAGER.NOTASKS");
}
```

Et

```
<h:outputText value="#{taskController.chaineTraduite}"...
```

3 Surcharge des entrées

Exercice N°15 : Surcharge d'un bundle

Surcharger dans `Custom_fr.properties` une des entrées de `Messages_fr.properties`.

Par exemple nous allons modifier le nom de l'auteur présent dans le fichier `bundles/Messages_fr.properties`.

```
ABOUT.AUTHOR.NAME = a compl\u00E9ter
```

Dans `bundles/Custom_fr.properties` ajouter la clé `ABOUT.AUTHOR.NAME` et modifier son contenu :

```
ABOUT.AUTHOR.NAME = C\u00E9line Didier
```

Vérifier dans la page d'accueil que le nom de l'auteur est bien celui déclaré dans `Custom`

4 Définition des langages

Exercice N°16 : Ajout d'un langage

Ajouter le langage japonais et traduire toute l'application ☺.

L'ajout d'un langage se fait dans le fichier de configuration `src/main/webapps/WEB-INF/jsf/faces-config.xml`. Il suffit ensuite d'écrire le bundle correspondant.

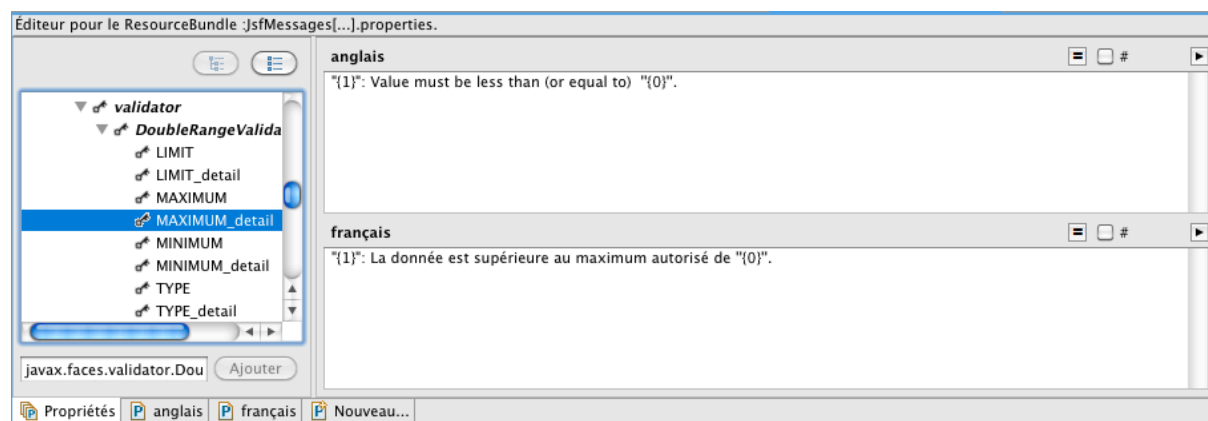
5 Les messages d'erreur par défaut de JSF

Dans `src/main/webapps/WEB-INF/jsf/faces-config.xml` on remarque

```
<message-bundle>properties.i18n.bundles.JsfMessages</message-bundle>
```

... qui indique que les messages utilisés par *JSF* sont déclarés dans `properties/i18n/bundles/JsfMessages`.

On notera dans l'exemple ci-après, l'usage du passage de variables.



Formulaires et Validation

1 Formulaires et binding

Exercice N°17 : Création d'un formulaire de saisie simple

Nous allons ajouter un formulaire de saisie d'une nouvelle tâche.

Les données saisies à travers le formulaire vont être bindées sur un objet.

Dans *org.esupportail.formation.web.controllers.TaskController* ajouter une propriété *currentTask* de type *Task* ainsi que les accesseurs correspondants.

```
private Task currentTask=null;
[...]
public Task getCurrentTask() {
    return currentTask;
}

public void setCurrentTask(Task currentTask) {
    this.currentTask = currentTask;
}
```

Nous allons d'abord créer un formulaire simple qui permet la saisie du titre (obligatoire), de la description et du caractère public ou non de la tâche.

Créer de formulaire dans *taskManager.xhtml*. On fera afficher les messages d'erreur correspondant à chaque champ.

```
<h:form id="newTask">
    <h:panelGrid id="grid" columns="1">
        <f:facet name="header">
            <h:outputText
value="#{msgs['TASK.MANAGER.ADD.TASK']}" />
        </f:facet>
        <h:panelGroup>
            <h:outputLabel for="title"
value="#{msgs['TASK.TITLE']}" />
            <h:inputText id="title"
value="#{taskController.currentTask.title}" required="#{true}" />
            <h:message for="title" />
        </h:panelGroup>
        <h:panelGroup>
            <h:outputLabel for="description"
value="#{msgs['TASK.DESCRPTION']}" />
            <h:inputTextarea id="description"
value="#{taskController.currentTask.description}" />
            <h:message for="description" />
        </h:panelGroup>
        <h:panelGroup>
            <h:outputLabel for="publicTask"
value="#{msgs['TASK.PUBLICTASK']}" />
            <h:selectBooleanCheckbox id="publicTask"
value="#{taskController.currentTask.publicTask}" />
            <h:message for="publicTask" />
        </h:panelGroup>
    </h:panelGrid>
    <h:commandButton action="#{taskController.addTask}"
value="#{msgs['TASK.MANAGER.ADD.TASK']}" />
</h:form>
```

✎ Récupérer le fichier `taskManager.xhtml-exo17`

Enfin dans ***TaskController*** ajouter la méthode `addTask()` appelée à la validation du formulaire et qui ajoute la tâche saisie en base de données :

```
public void addTask() {
    getDomainService().addTask(currentTask);
    currentTask=new Task();
    sortedTasks = getDomainService().getTasks();
}
```

Sur le module `web-jsf-servlet`

```
mvn install jetty:run
```

Tester le formulaire et l'affichage des erreurs de saisie.

2 Les convertisseurs

JSF propose des convertisseurs par défaut (*DateTimeConverter* et *NumberConverter*). Ceux-ci permettent de transformer une date ou un nombre suivant différentes règles.

Exercice N°18 : Utilisation d'un convertisseur prédéfini

Ajouter le champ de saisie de la date et convertir ce type complexe grâce au convertisseur `DateTime` fourni par JSF qui transformera la date saisie en objet `Date`.

```
<h:panelGroup>
  <h:outputLabel for="date" value="#{msgs['TASK.DATE']}" />
  <h:inputText id="date"
value="#{taskController.currentTask.date}">
    <f:convertDateTime type="date" pattern="ddMMyyyy" />
  </h:inputText>
  <h:message for="date" />
</h:panelGroup>
```

Tester le formulaire et l'affichage des erreurs de saisie.

Notez que si vous souhaitez changer le format de saisie il faudra également aller changer le message d'erreur correspondant dans **`bundles/JsfMessages_fr.properties`**

```
javax.faces.convert.DateTimeConverter.CONVERSION = Veuillez
saisir les dates sous la forme jjmmaaaa.
javax.faces.convert.DateTimeConverter.CONVERSION_detail = "{1}":
Conversion en Date impossible, Veuillez saisir sous la forme
jjmmaaaa.
```

Dans certains cas, il est aussi nécessaire de définir des convertisseurs manuellement. C'est notamment le cas pour les listes déroulantes.

Exercice N°19 : Création d'un convertisseur

Ajouter une liste déroulante pour le choix de l'utilisateur propriétaire de la tâche. La liste proposera l'ensemble des *Users* en base de données.

On souhaite mapper directement l'utilisateur qui sera sélectionné avec la propriété *owner* du bean *task* mappé au formulaire. D'où la nécessité d'un convertisseur qui pour un *login* fourni le *User* correspondant en base et pour un *User* le *login* correspondant

Pour cela, ajouter un élément `<h:selectOneListbox>` dans le formulaire.

```
<h:selectOneListbox value="#{taskController.currentTask.owner}"
  converter="#{userConverter}">
  <f:selectItems value="#{taskController.userItems}" />
</h:selectOneListbox>
```

Ajouter la méthode `getUserItems()` qui permet de récupérer l'ensemble des utilisateurs en base dans *TaskController*

```
public List<SelectItem> getUserItems() {
    List<User> les_users= getDomainService().getUsers();
    ArrayList<SelectItem> userItems = new ArrayList<SelectItem>();
    for (User user : les_users) {
        if (user.getDisplayName() !=null)
            userItems.add(new SelectItem(user,
            user.getDisplayName()+" (" +user.getLogin()+")"));
        else
            userItems.add(new SelectItem(user, "Inconnu
            (" +user.getLogin()+")"));
    }
    return userItems;
}
```

Dans *org.esupportail.formation.web.converters* écrire une classe *UserConverter*. Elle devra implémenter l'interface *javax.faces.convert.Converter* et notamment les méthodes `getAsObject()` et `getAsString()`.

```
package org.esupportail.formation.web.converters;
[...]
public class UserConverter implements Converter, InitializingBean {
    DomainService domainService;

    [...]

    @Override
    public Object getAsObject(FacesContext arg0, UIComponent arg1,
    String arg2) throws ConverterException {
        User u=domainService.getUser(arg2);
        return u;
    }

    @Override
    public String getAsString(FacesContext arg0, UIComponent arg1,
    Object arg2) throws ConverterException {
        if (arg2!=null && (arg2 instanceof User))
            return ((User)arg2).getLogin();
        else
            return null;
    }
    [...]
}
```

On souhaite aller rechercher l'objet *User* en base de données. On n'utilisera la méthode `getUser()` de *domainService* que l'on injectera via la déclaration des beans *Spring*.

Déclarer ce convertisseur dans **converters.xml**.

```
<bean id="userConverter"
      class="org.esupportail.formation.web.converters.UserConverter">
  <description>A converter for User.</description>
  <property name="domainService" ref="domainService" />
</bean>
```

Tester le formulaire.

3 Les validateurs

Exercice N°20 : Validation des champs grâce à un validateur

Tester la taille du champ titre grâce à un validateur prédéfini.

Ajouter le validateur dans **taskManager.xhtml**

```
<h:inputText id="title" value="#{taskController.currentTask.title}"
             required="#{true}">
  <f:validateLength minimum="5" />
</h:inputText>
```

Tester le formulaire et l'affichage des erreurs de saisie.

Faire l'équivalent avec un validateur personnalisé.

Ajouter la méthode `validateTitle()` dans **TaskController**

```
public void validateTitle(FacesContext ctx, UIComponent ui, Object
obj)
  throws ValidationException {
  if (obj != null){
    if (((String)obj).length()<5)
      throw new ValidationException("Erreur le titre
ne doit pas comporter moins de 5 caractères");
  }
}
```

Puis l'appel :

```
<h:inputText id="title" value="#{taskController.currentTask.title}"
             required="#{true}" validator="#{taskController.validateTitle}" />
```

Tester le formulaire et l'affichage des erreurs de saisie.

Exercice N°21 : Validation des champs grâce à JSR 303

Gérer la validation de la classe métiers `Task` grâce la normalisation *Bean Validation* (JSR303). Le titre de la tâche sera obligatoire et devra contenir de 5 et 15 caractères. La description devra contenir un maximum de 30 caractères.

Commencer par ajouter la dépendance dans le `pom.xml` du module `domain-beans`

```
<dependency>
  <groupId>javax.validation</groupId>
  <artifactId>validation-api</artifactId>
  <version>1.0.0.GA</version>
  <scope>provided</scope>
</dependency>
```

Dans `Task.java` du module `domain-beans`, ajouter les annotations

```
@Column(nullable = false)
@NotNull
@Size(max = 15, min = 5)
private String title;

@Size(max = 30)
private String description;
```

Sur le module `domain-beans`

```
mvn install
```

Tester le formulaire et l'affichage des erreurs de saisie.

Exercice N°22 : Amélioration du formulaire : édition et suppression

Ajouter un bouton d'édition et un bouton de suppression d'une tâche dans le tableau listant les tâches.

Dans `TaskController` ajouter la propriété `taskToEditOrDelete` de type `Task` ainsi que les accesseurs correspondants.

```
private Task taskToEditOrDelete=null;
[...]
```

```

public Task getTaskToEditOrDelete() {
    return taskToEditOrDelete;
}

public void setTaskToEditOrDelete(Task taskToEditOrDelete) {
    this.taskToEditOrDelete = taskToEditOrDelete;
}

```

Puis les méthodes *deleteTask()* et *editTask()*

```

public void deleteTask() {
    getDomainService().deleteTask(currentTask);
    taskToEditOrDelete=new Task();
    sortedTasks = getDomainService().getTasks();
}

public void editTask() {
    currentTask = taskToEditOrDelete;
    taskToEditOrDelete=new Task();
    sortedTasks = getDomainService().getTasks();
}

```

Dans **taskManager.xhtml** ajouter les deux boutons *Editer* et *Supprimer* pour chaque tâche

```

<h:commandButton action="#{taskController.editTask}"
value="#{msgs[ 'TASK.MANAGER.EDIT.TASK' ]}">
    <f:setPropertyActionListener value="#{task}"
target="#{taskController.taskToEditOrDelete}"/>
</h:commandButton>

<h:commandButton action="#{taskController.deleteTask}"
value="#{msgs[ 'TASK.MANAGER.DELETE.TASK' ]}">
    <f:setPropertyActionListener value="#{task}"
target="#{taskController.taskToEditOrDelete}"/>
</h:commandButton>

```

On notera l'utilisation de la balise *<f:setPropertyActionListener* pour mapper la tâche à éditer ou à supprimer sur une propriété du contrôleur.

Exercice N°23 : Ajout de fonctions Ajax pour l'ergonomie

Ajouter l'affichage en direct via ajax de ce qui a été tapé dans le titre de la tâche.

JSF permet l'ajout de fonction *ajax* de bas niveau.

Ajouter *<f:ajax* dans **taskManager.xhtml**

```

<h:inputText id="title" value="#{taskController.currentTask.title}"
required="#{true}">
    <f:ajax event="keyup" render="outAjax" />
</h:inputText>
<h:outputText id="outAjax"
value="#{taskController.currentTask.title}" />

```

Des bibliothèques proposent des composants JSF intégrant directement de l'*ajax* comme l'auto-complétion, par exemple.

Ajouter un champ texte pour indiquer l'uid du User attaché à la tâche et proposer une auto-complétion Ajax qui proposera les utilisateurs existants déjà en base.

A faire pour le plaisir ☺

Gestion des exceptions

Exercice N°24 : L'affichage des exceptions

Adapter la vue des exception actuelle afin qu'elle affiche les informations qui nous intéressent par exemple, la pile d'erreurs java complète.

Pour cela créer, un bouton sur la page d'accueil de l'application `welcome.xhtml` et dont l'action générera forcément une exception.

```
<h:commandButton action="#{taskController.genereException}"
value="#{msgs['GENERATE.EXCEPTION']}" />
```

Accompagné de

```
public void genereException() throws Exception {
    throw new Exception("Bonjour je suis une erreur !");
}
```

Dans la vue des exceptions `exceptions/exception.xhtml`

```
[...]
<e:subSection value="FULL TRACE" />
<h:outputText value="#{exceptionController.exceptionStackTrace}" />
<hr />
[...]
```

Tester

On notera que le bouton de retour ne fonctionne pas correctement car il manque la règle de navigation correspondant à l'issue `applicationRestarted`

```
<navigation-case>
  <from-outcome>applicationRestarted</from-outcome>
  <to-view-id>/stylesheets/welcome.xhtml</to-view-id>
  <redirect />
</navigation-case>
```

Créer une nouvelle exception de type `TaskException` pour laquelle on souhaitera un affichage différent de l'erreur à l'utilisateur. Configurer l'application pour utiliser cette nouvelle vue en cas d'exception.

Créer la classe ***TaskException*** :

```
package org.esupportail.formation.web.exceptions;
```



```
public class TaskException extends Exception {
    public TaskException() {
        super();
    }
    public TaskException(String message) {
        super(message);
    }
}
```

Pour cela créer, un nouveau bouton sur la page d'accueil de l'application **welcome.xhtml** et dont l'action générera forcément une *TaskException*.

```
<h:commandButton action="#{taskController.genereTaskException}"
value="#{msgs[ 'GENERATE.TASK.EXCEPTION' ]}"/>
```

Accompagné de

```
public void genereTaskException() throws TaskException {
    //imaginons un code qui génère une TaskException
    throw new TaskException("Moi je suis une Task Exception !");
}
```

Créer une nouvelle vue pour ce type d'exception, **exceptions/task-exception.xhtml** qui proposera par exemple un affichage simplifié.

Ajouter une nouvelle entrée à *exceptionViews* (mapping entre le type de l'exception et la vue) au niveau de la configuration du service de gestion des exceptions **properties/exceptionHandling/exceptionHandling.xml**.

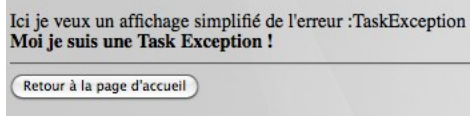
```
<bean id="exceptionServiceFactory"
class="org.esupportail.commons.services.exceptionHandling.CachingEmailExceptionHandlerServiceFactoryImpl"
parent="abstractApplicationAwareBean"
>
<property name="smtpService" ref="smtpService" />
<property name="recipientEmail"
value="{exceptionHandling.email}" />

<property name="exceptionViews" >
<map>
<entry key="java.lang.Exception" value="go_exception" />
<entry
key="org.esupportail.formation.web.exceptions.TaskException"
value="go_task_exception" />
</map>
</property>
<property name="logLevel"
value="{exceptionHandling.logLevel}"/>
<property name="cacheManager" ref="cacheManager" />
<property name="cacheName" value="" />
</bean>
```

Enfin ajouter la règle navigation correspondante.

```
<navigation-case>
<from-outcome>go_task_exception</from-outcome>
<to-view-id>/stylesheets/exceptions/task-exception.xhtml</to-view-id>
<redirect/>
</navigation-case>
```

Tester.



On constate que le service de gestion des exceptions offre aussi la possibilité de recevoir par e-mail les exceptions survenues (avec un mécanisme de cache).

Il faudra alors configurer le service d'envoi d'e-mail (cf. le chapitre suivant).

Et préciser dans **config.properties**

```
exceptionHandling.email=celine.didier@uhp-nancy.fr
```

Enfin on améliorera notre application en réinitialisant les contrôleurs en cas d'exception.

Implémenter la méthode reset dans le contrôleur **TaskController** :

```
@Override
public void reset() {
    super.reset();
    currentTask = new Task();
    taskToEditOrDelete = new Task();
    System.out.println("Reset task controller");
}
```

Tester en éditant une tâche puis en provoquant l'exception.

Envoi d'e-mail

Nous avons vu précédemment que les exceptions pouvaient être envoyée par e-mail à une adresse précisée dans la variable *exceptionHandling.email*.

Exercice N°25 : Ajout des fonctionnalités d'e-mail

Nous allons maintenant configurer les fonctionnalités d'e-mail pour d'une part faire envoyer par e-mail les exceptions levées par l'application et d'autre part faire envoyer un e-mail par un controller.

La configuration des fonctionnalités de mail se situe dans le fichier *spring smtp/smtp.xml*

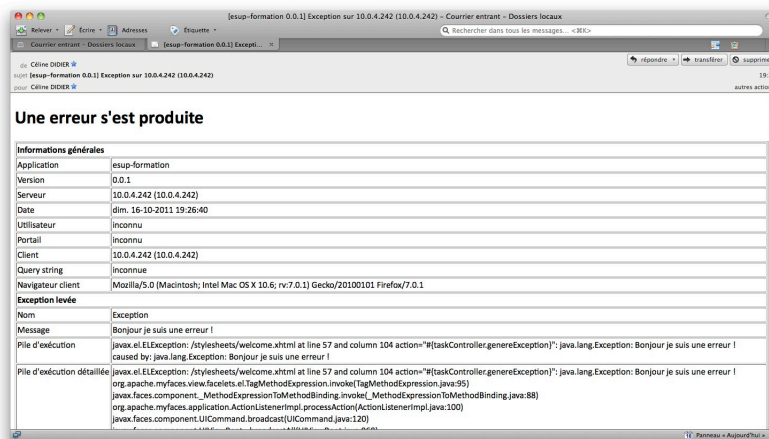
On passera par le fichier de propriétés **config.properties**

```
exceptionHandling.email=celine.didier@uhp-nancy.fr

smtp.host=smtp.uhp-nancy.fr
smtp.port=25
smtp.charset=utf-8
smtp.fromEmail=celine.didier@uhp-nancy.fr
smtp.fromName=Esup-Formation Application
smtp.interceptAll=true
smtp.interceptEmail=celine.didier@uhp-nancy.fr
smtp.interceptName=Céline DIDIER
smtp.notInterceptedAddresses=
smtp.testEmail=celine.didier@uhp-nancy.fr
smtp.testName=Céline DIDIER
```

Tester en provoquant une exception

```
2011-10-16 19:26:40,676 ERROR
[org.esupportail.commons.services.application.SimpleApplicationServiceImpl] - property quickStart is not set!
2011-10-16 19:26:40,830 ERROR
[org.esupportail.commons.services.exceptionHandling.CachingEmailExceptionHandlerServiceImpl] - javax.el.ELException: /stylesheets/welcome.xhtml
at line 57 and column 104
action="#{taskController.genereException}": java.lang.Exception:
Bonjour je suis une erreur !
2011-10-16 19:26:40,831 ERROR
[org.esupportail.commons.services.exceptionHandling.CachingEmailExceptionHandlerServiceImpl] - caused by: java.lang.Exception: Bonjour je suis
une erreur !
[...]
2011-10-16 19:26:45,836 INFO
[org.esupportail.commons.services.smtp.SmtUtils] - an email has
been sent to '?UTF-
8?Q?C=C3=AF=C2=BF=C2=BDline_DIDIER_=28normally_sent?= =?UTF-
8?Q?_to_celine=2Edidier=40uhp-nancy=2Efr=29?= <celine.didier@uhp-
nancy.fr>'.
```



Ajouter l'envoi d'un e-mail à chaque création d'une nouvelle tâche

Injecter *smtpService* au **TaskController**

```
<bean id="taskController"
    class="org.esupportail.formation.web.controllers.TaskController"
    parent="abstractContextAwareController"
    scope="session"
    >
    <property name="authenticator" ref="authenticator" />
    <property name="urlGenerator"
ref="servletUrlGenerator"></property>
    <property name="smtpService" ref="smtpService" />
</bean>
```

Ajouter l'envoi de l'e-mail dans la méthode *addTask()*

```
public void addTask() {
    System.out.println("ADD"+currentTask.getId());
    getDomainService().addTask(currentTask);
    try {
        smtpService.send(new InternetAddress("celine.didier@uhp-
nancy.fr"),
        "CREATION D'UNE TACHE",
        "<b>Une nouvelle tâche vient d'être crée</b><br/>" +
currentTask.getTitle()+"<br/>" +currentTask.getDescription(),
        "**Une nouvelle tâche vient d'être crée**\n\n" +
currentTask.getTitle()+"\n"+currentTask.getDescription());
    } catch (AddressException e) {
    }
    currentTask=new Task();
    sortedTasks = getTasksFromDomainService();
}
```

Tester

```
2011-10-16 19:49:49,696 INFO
[org.esupportail.commons.services.smtp.SmtpUtils] - an email has
been sent to '=?UTF-
8?Q?C=C3=AF=C2=BF=C2=BDline_DIDIER_=28normally_sent?=?UTF-
8?Q?_to_celine=2Edidier=40uhp-nancy=2Efr=29?= <celine.didier@uhp-
nancy.fr>'.
```



Authentification

Exercice N°26 : Mettre en place une authentification CAS

Activer l'authentification CAS et faire en sorte qu'elle ne soit demandée que sur les pages de gestion des tâches et des utilisateurs, la page d'accueil sera publique et affichera la liste des tâches publiques.

Pour vérifier que l'on est authentifié on affichera dans le menu, le login de la personne connecté sinon on affichera « invité »

Démarrer le serveur CAS installé sur la machine virtuelle en lançant le script `start.sh` situé dans `/home/esup/`.

Créer un contrôleur intitulé *UserController* qui se chargera de la gestion des utilisateurs de notre application.

Injecter dans ce contrôleur le bean *authenticator* déclaré dans le module `domain-services`.

Pour cela adapter le fichier `src/main/resources/META-INF/esup-formation-domain-services-auth.xml` afin que le service d'authentification soit de type CAS :

```
<bean id="authenticator" lazy-init="true"
      class="org.esupportail.formation.services.auth.AuthenticatorImpl">
  <property name="authenticationService"
    ref="servletAuthenticationService" />
</bean>

<bean id="servletAuthenticationService" lazy-init="true"
      class="org.esupportail.commons.services.authentication.CasFilterAuthenticationService">
</bean>
```

Puis sur le module `domain-services` :

```
mvn install
```

Déclarer le contrôleur :

```
<bean id="UserController"
      class="org.esupportail.formation.web.controllers.UserController"
      parent="abstractContextAwareController"
      scope="session">
  <property name="authenticator" ref="authenticator"/>
</bean>
```

Ajouter la vue `userManager.xhtml` (ainsi que la navigation nécessaire + un lien dans le menu) qui nous permettra par la suite d'ajouter/modifier/supprimer des utilisateurs. Pour l'instant, ne faire afficher qu'un texte indiquant qu'on est bien sur la page de gestion des utilisateurs.

Dans `web.xml`

```
<filter>
  <filter-name>CAS Authentication Filter</filter-name>
  <filter-class>org.jasig.cas.client.authentication.AuthenticationFilter</filter-class>
  <init-param>
    <javaee:param-name>casServerLoginUrl</javaee:param-name>
```

```

        <javaee:param-value>
https://localhost/cas/login</javaee:param-value>
    </init-param>
    <init-param>
        <javaee:param-name>serverName</javaee:param-name>
        <javaee:param-value>http://localhost:8080</javaee:param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>CAS Authentication Filter</filter-name>
    <url-pattern>/stylesheets/taskManager.xhtml</url-pattern>
</filter-mapping>
<filter-mapping>
    <filter-name>CAS Authentication Filter</filter-name>
    <url-pattern>/stylesheets/userManager.xhtml</url-pattern>
</filter-mapping>
<filter>
    <filter-name>CAS Validation Filter</filter-name>
    <filter-
class>org.jasig.cas.client.validation.Cas10TicketValidationFilter</f
ilter-class>
    <init-param>
        <javaee:param-name>casServerUrlPrefix</javaee:param-name>
        <javaee:param-value>http://localhost/cas</javaee:param-value>
    </init-param>
    <init-param>
        <javaee:param-name>serverName</javaee:param-name>
        <javaee:param-value>http://localhost:8080</javaee:param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>CAS Validation Filter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

```

On pourra déplacer les fichiers **taskManager.xhtml** et **userManager.xhtml** dans un sous répertoire afin de filtrer plus proprement la partie privée de la façon suivante :

```
<url-pattern>/stylesheets/private/*</url-pattern>
```

Ajouter l'affichage du *Login* de la personne authentifiée :

```

<h:outputText
    value="#{msgs['CURRENT.USER']}"
    #{userController.currentUserLogin}" />

```

Avec dans **UserController**:

```

public String getCurrentUserLogin(){
    try {
        if (authenticator.getUser()!=null)
            return authenticator.getUser().getLogin();
        else
            return "Invité";
    } catch (Exception e) {
        return "Invité";
    }
}

```

Tester l'authentification.

Adapter la page d'accueil et la page de gestion des tâches pour qu'elles affichent respectivement les tâches publiques et les tâches de l'utilisateur connecté.

Exercice N°27 : Création de boutons de connexion et déconnexion

Faire en sorte que le menu ne s'affiche que pour une personne authentifiée et s'accompagne d'un bouton de déconnexion. Dans le cas contraire seul un bouton de connexion sera proposé.

On ajoutera dans la vue les boutons :

```
<h:outputText value="#{msgs['CURRENT.USER']}"
#{userController.currentUserLogin}"
rendered="#{userController.userLogged}"/>rendered="#{! userController.userLogged}"/>></h:commandLink>

<h:commandLink action="#{userController.goLogout}"><h:outputText
value="#{msgs['USER.LOGOUT']}"
rendered="#{userController.userLogged}"/>></h:commandLink>
]
```

Et les rendered sur les lien

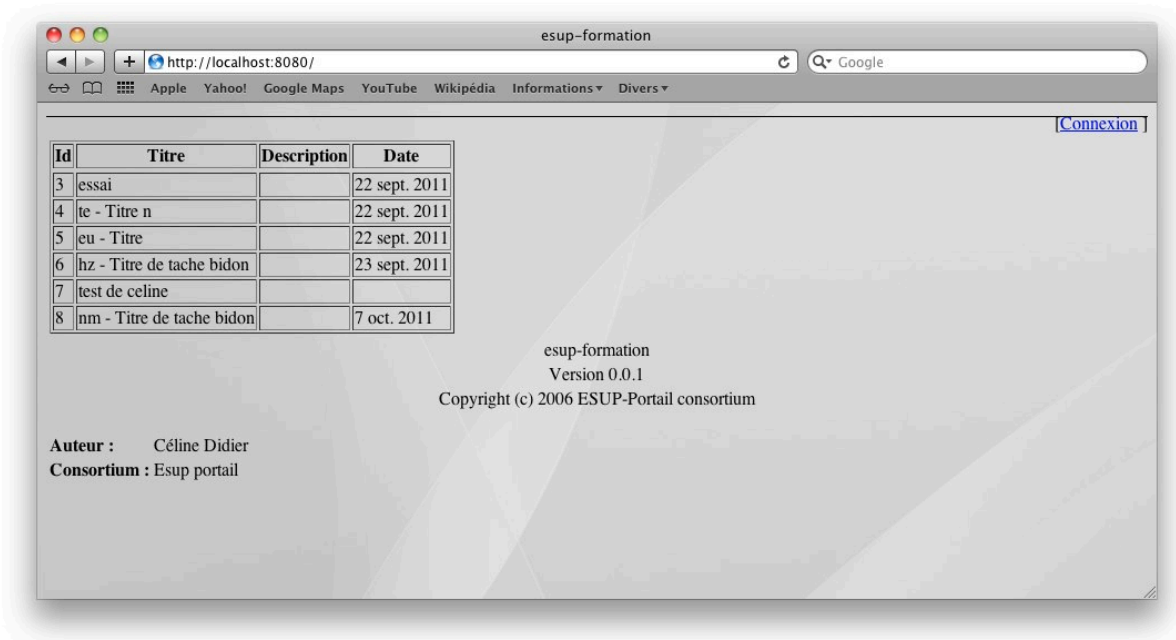
```
<h:commandLink action="#{taskController.goToTaskManagerPage}"
rendered="#{userController.userLogged}"><h:outputText
value="#{msgs['TASK.MANAGER']}" /></h:commandLink></li>
<h:commandLink action="#{userController.goToUserManagerPage}"
rendered="#{userController.userLogged}"><h:outputText
value="#{msgs['USER.MANAGER']}" /></h:commandLink>
```

Dans *userController* :

```
public boolean isUserLogged(){
    try {
        if (authenticator.getUser()!=null)
            return true;
        else
            return false;
    } catch (Exception e) {
        return false;
    }
}

public String goLogout(){
    try {
        getSessionController().logout();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return "go_home";
}
```

Quelques traductions et css plus tard...



Pour améliorer ceci on pourra faire en sorte que le logout entraine un logout du serveur CAS grâce à une redirection, comme dans l'exemple suivant :

<https://sourcesup.cru.fr/scm/viewvc.php/trunk/src/org/esupportail/reunion/web/controllers/SessionController.java?root=esup-reunion>

On aura dans le web.xml

```

<filter>
    <filter-name>CASLogout</filter-name>
    <filter-
class>org.jasig.cas.client.session.SingleSignOutFilter</filter-
class>
</filter>
[...]
<filter-mapping>
    <filter-name>CASLogout</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
    
```


Accès à un annuaire LDAP

Esup-commons propose un certain nombre d'objets facilitant l'accès à un annuaire LDAP.

1 Paramétrage du LDAP

Ajouter la dépendance vers le module LDAP de *esup-commons* dans le module où vous en aurez besoin et créer un fichier de configuration spring spécifique pour la gestion du LDAP.

Dans le fichier pom.xml :

```
<dependency>
  <groupId>org.esupportail</groupId>
  <artifactId>esup-commons2-ldap</artifactId>
  <version>${esupcommons.version}</version>
  <exclusions>
    <exclusion>
      <artifactId>spring-tx</artifactId>
      <groupId>org.springframework</groupId>
    </exclusion>
  </exclusions>
</dependency>
```

! Un conflit de librairies nous oblige à exclure ici la librairie spring-tx

Dans le module domain-services créer le fichier **src/main/resources/META-INF/esup-formation-domain-services-ldap.xml**

Le bean de connexion à l'annuaire LDAP utilise le *ldapTemplate* de spring.

```
[...]
<bean id="ldapUserService"
  class="org.esupportail.commons.services.ldap.SearchableLdapUser
ServiceImpl" lazy-init="true">
  <property name="i18nService" ref="i18nService"/>
  <property name="ldapTemplate" ref="ldapTemplate"/>
  <property name="dnSubPath" value="{ldap.dnSubPath}"/>
  <property name="idAttribute"
value="{ldap.uidAttribute}"/>
  <property name="attributesAsString"
value="{ldap.attributes}"/>
  <property name="searchAttribute"
value="{ldap.searchAttribute}"/>
  <property name="searchDisplayedAttributesAsString"
value="{ldap.searchDisplayedAttributes}"/>
  <property name="cacheManager" ref="cacheManager"/>
  <property name="cacheName" value=""/>
  <property name="testFilter" value="{ldap.testFilter}"/>
</bean>

<bean id="ldapTemplate"
  class="org.springframework.ldap.core.LdapTemplate"
  lazy-init="true">
  <property name="contextSource" ref="contextSource"/>
</bean>

<bean id="contextSource"
```

```

        class="org.esupportail.commons.services.ldap.MultiUrlLdapContextSource" lazy-init="true">
            <property name="url" value="{ldap.url}"/>
            <property name="userDn" value="{ldap.userDn}"/>
            <property name="password" value="{ldap.password}"/>
            <property name="base" value="{ldap.base}"/>
            <property name="baseEnvironmentProperties">
                <map>
                    <entry key="com.sun.jndi.ldap.connect.timeout"
                        value="{ldap.connectTimeout}"/>
                </map>
            </property>
        </bean>
    [...]
```

- **Récupérer le fichier esup-formation-domain-services-ldap.xml-exo28**

On ajoutera l'import au niveau de du contexte d'application

```
<import resource="classpath*:META-INF/esup-formation-domain-services-ldap.xml" />
```

... et les propriétés dans **config.properties**

```

ldap.url=ldap://localhost:389
ldap.userDn=
ldap.password=
ldap.base=dc=univ,dc=fr
ldap.dnSubPath=ou=people
ldap.uidAttribute=uid
ldap.displayNameAttribute=displayName
ldap.emailAttribute=mail
ldap.searchAttribute=cn
ldap.attributes=cn,displayName,sn,mail,employeeType,department,homedirectory
ldap.searchDisplayedAttributes=cn,sn,mail,displayName,employeeType,department
```

2 Recherche et utilisation de l'annuaire

Exercice N°28 : Recherche des informations d'une personne dans le LDAP

Au moment où l'utilisateur se connecte sur notre application, faire en sorte que celle-ci renseigne la propriété *displayName* de l'objet user par la valeur *displayName* de l'utilisateur récupérée dans l'annuaire LDAP.

Dans le bean **Authenticator** du module domain-services déclarer une propriété de type **LdapUserService** qui sera renseignée par le mécanisme d'injection *spring*.

Les tests unitaires mis en place dans les exercices précédents vont maintenant échouer à cause des fichiers de configuration spring qu'il aurait fallu ajouter dans les répertoires de test. Pour palier ce problème, nous allons passer outre.

- **Clique droit sur le projet esup-formation-domain-services > Run As > 5 Maven build...**
- **Saisir un nom de tâche et goals : "install" et cocher "skip Tests"**

Dans la classe **AuthenticatorImpl** adapter la méthode *getUser()*.

```

public User getUser() throws Exception {
    [...]
    if (AuthUtils.CAS.equals(authInfo.getType())) {
```

```

        if (logger.isDebugEnabled()) {
            logger.debug("CAS authentication");
        }
        User user = new User();
        user.setLogin(authInfo.getId());
        LdapUser uLdap=ldapUserService.getLdapUser(user.getLogin());
        String displayName=uLdap.getAttribute("cn");
        user.setDisplayName(displayName);

        storeToSession(authInfo, user);
        return user;
    }

```

Du coté de la vue `getCurrentUserLogin()` deviendra `getCurrentUserName()`

```

public String getCurrentUserName(){
    try {
        if (authenticator.getUser()!=null)
            if(authenticator.getUser().getDisplayName()!=null)
                return authenticator.getUser().getDisplayName();
            else
                return
getString("USER.UNKNOWN",authenticator.getUser().getLogin());
        else
            return getString("USER.GUEST");
    } catch (Exception e) {
        return getString("USER.GUEST");
    }
}

```

Que l'on appellera de la même manière

```

<h:outputText value="#{msgs['CURRENT.USER']}"
    #{userController.currentUserName}"
    rendered="#{userController.userLogged}"/>

```

Tester

Exercice N°29 : Recherche d'une ou plusieurs personnes dans l'annuaire

Compléter la page de gestion des utilisateurs en ajoutant une recherche des utilisateurs dans l'annuaire qui affichera pour résultat une liste de personnes avec pour chaque entrée un bouton qui nous permettra d'importer l'utilisateur dans la base de donnée de l'application.

Pour réaliser cela il faudra :

- Créer un formulaire de recherche dans `userManager.xhtml`
- Ajouter un tableau qui affichera les résultats
- Ajouter les variables nécessaires à ***UserController***
- Injecter le *LdapUserService*
- Déclarer une méthode de recherche

```

public void searchUserInLdap(){
    List<LdapUser> listOfLdapUser =
ldapUserService.getLdapUsersFromToken(searchUser);
    resultSearch=new ArrayList<User>();
    for (LdapUser ldapUser : listOfLdapUser) {
        User user=new User();
        user.setLogin(ldapUser.getAttribute("uid"));
        user.setDisplayName(ldapUser.getAttribute("displayName"));
        resultSearch.add(user);
    }
}

```

- Et une méthode d'ajout :

```
public void addUserFromLdap(){  
    getDomainService().addUser(chosenUser);  
}
```

Tester en recherchant "ens" ou "Antoine".

Gestion des URL

Exercice N°30 : Création d'un lien direct

Ajouter une page qui affiche le détail d'une tâche et qui propose une URL d'accès direct à cette tâche.

Pour cela créer une nouvelle vue qui affichera le détail d'une tâche ainsi qu'un lien sur chaque ligne du **taskManager** qui appellera cette vue.

Il faudra ajouter une méthode permettant de récupérer une tâche à partir de son identifiant au niveau de la couche domaine

La page de détail devra afficher le titre, la description, la date etc. de la tâche ainsi qu'une URL construite grâce à la classe **UrlGenerator** de *esup-commons*.

Dans le module de vue `web-jsf-servlet` créer le répertoire **src/main/resources/properties/deeplinking**

Créer ensuite le fichier **urlGenerator.xml** qui permettra de configurer la génération de l'url en suivant ce modèle.

```
<bean id="servletUrlGenerator"
class="org.esupportail.commons.services.urlGeneration.ServletUrlGene
ratorImpl" lazy-init="true">
  <property
    name="servletGuestUrl"
    value="http://localhost:8080/stylesheets/welcome.xhtml" />
</bean>
```

On ne paramètrera pour l'instant que l'URL permettant un accès non authentifié.

Dans un cas réel on préférera passer par une variable positionnée dans un fichier de propriétés afin qu'un exploitant n'ait pas à venir dans ce fichier pour adapter l'URL au moment de l'installation

Ainsi que le fichier **deepLinking.xml** qui permettra de configurer le **UrlPatternDescriptor** nécessaire au décryptage de l'URL, au traitement des paramètres et à la redirection.

```
<bean id="urlTaskDetail"
class="org.esupportail.commons.jsf.UrlPatternDescriptor">
  <property name="params">
    <list>
      <value>taskId</value>
    </list>
  </property>
  <property name="actionBinding" >
    <bean class="org.esupportail.commons.jsf.ActionBinding">
      <property name="action"
value="taskController.goUrlTask"/>
      <property name="args">
        <list><value>java.lang.String</value></list>
      </property>
    </bean>
  </property>
</bean>
```

Pour des question de simplification, on choisi un argument de type String pour le passage de paramètre. En effet, on évite ainsi les problèmes de cast et de

conversion des types simples (long, ient etc.) au niveau de la signature de la méthode qui sera appelée par EL.

On pensera à ajouter l'import de ces fichiers dans `ApplicationContext.xml`

Enfin, dans **TaskController** ajouter :

- La méthode qui fourni l'URL

```
public String getUrlTask(){
    Map<String, String> params = new HashMap<String,
String>();
    params.put("taskId", new Long(taskId).toString());
    String url = getUrlGenerator().getUrl(params);

    return url;
}
```

On pensera à injecter le bean `urlGenerator` au niveau du contrôleur

- La méthode qui redirige positionne le taskId et redirige vers la bonne page

```
public String goUrlTask(String taskId){
    this.taskId = new Long(taskId).longValue();
    detailedTask = getTaskFromDomainService(new
Long(taskId).longValue());
    return "go_taskDetailPage";
}
```

On utilisera un filtre *JSF* proposé par *esup-commons* pour faire la redirection à partir de l'**UrlPatternDescriptor**.

Ainsi on ajoutera dans `/src/main/webapp/webapp/WEB-INF/faces-config.xml`

```
<lifecycle>
  <phase-
listener>org.esupportail.commons.jsf.ResourceBundlePhaseListener</ph
ase-listener>
  <phase-
listener>org.esupportail.commons.jsf.DeepLinkingPhaseListener</phase
-listener>
</lifecycle>
```

On obtient alors une URL construite de la façon suivante :

☞ <http://localhost:8080/stylesheets/welcome.xhtml?args=taskId%3d21>

Pour aller plus loin...

On pourra améliorer ceci en proposant une URL qui nécessite de passer par une authentification CAS. (Voir correction exo30bis)

On pourra également proposer, grâce à un formulaire, d'envoyer à une adresse e-mail saisie, le descriptif de la tâche ainsi que le lien d'accès direct à cette dernière.

Webservices

1 Les webservices CXF

Exercice N°31 : Exposer un webservice CXF

Nous allons exposer l'interface *domainService* comme un *webservice* qui pourra être appelé par un programme tiers.

Créer une méthode du service domaine qui affichera les *n* dernières tâches d'un utilisateur.

Dans un premier temps ajouter dans *daoService* et *domainService* la méthode *getLastTasksForUser()* qui retournera les *n* dernières tâches (en date) de l'utilisateur.

On adaptera la requete

```
@NamedQuery(
    name="tasksForUser",
    query="SELECT t FROM Task t WHERE t.owner.login =
:userLogin ORDER BY t.date"
)
```

Et on limitera le nombre de résultats

```
public List<Task> getLastTasksForUser(User u, int limit) {
    Query q = entityManager.createNamedQuery("tasksForUser");
    q.setParameter("userLogin", u.getLogin());
    q.setMaxResults(limit);
    List<Task> ret = (List<Task>)q.getResultList();
    return ret;
}
```

Dans le fichier **pom.xml** du module **domain-service** ajouter les dépendances nécessaires.

```
<dependency>
    <groupId>org.esupportail</groupId>
    <artifactId>esup-commons2-ws-cxf</artifactId>
    <version>${esupcommons.version}</version>
    <type>pom</type>
</dependency>
<dependency>
    <groupId>org.esupportail</groupId>
    <artifactId>esup-commons2-rs-cxf</artifactId>
    <version>${esupcommons.version}</version>
    <type>pom</type>
</dependency>
```

Puis dans l'interface *domainService*

```
[...]
import javax.jws.WebService;
[...]

@WebService
public interface DomainService extends Serializable {
```

Ajouter dans le fichier de configuration *spring* **src/main/resources/META-INF/esup-formation/domain-service-domain.xml**

```
[...]
xmlns:util="http://www.springframework.org/schema/util"
xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:tx="http://www.springframework.org/schema/tx"
xmlns:jaxws="http://cxf.apache.org/jaxws"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-3.0.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-3.0.xsd
http://cxf.apache.org/jaxws
http://cxf.apache.org/schemas/jaxws.xsd">
  [...]
  <import resource="classpath:META-INF/cxf/cxf.xml" />
  <import resource="classpath:META-INF/cxf/cxf-extension-
soap.xml" />
  <import resource="classpath:META-INF/cxf/cxf-servlet.xml" />

  <jaxws:endpoint id="domainService.remoteService"
implementor="#domainService"
  address="/ToDoService">
  </jaxws:endpoint>
  [...]

```

Les fichiers de configuration cxf importés ici sont embarqués dans les jar cxf

Enfin il va falloir déclarer la servlet CXF au niveau de fichier **web.xml** de la vue.

```
<servlet>
  <servlet-name>CXFServlet</servlet-name>
  <servlet-class>
  org.apache.cxf.transport.servlet.CXFServlet
  </servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>CXFServlet</servlet-name>
  <url-pattern>/services/*</url-pattern>
</servlet-mapping>

```

Tester l'URL :

☞ <http://localhost:8080/services>

Available SOAP services:

<p>DomainService</p> <ul style="list-style-type: none"> • getTasksForUser • getTasksForUser • getPublicTasks • getTasks • addTask • getTask • deleteTask • deleteTask • getUsers • updateTask • deleteTask • getUser • addUser 	<p>Endpoint address: http://localhost:8080/services/ToDoService WSDL : (http://domain.formation.esupportail.org/DomainServiceImplService) Target namespace: http://domain.formation.esupportail.org/</p>
---	---

☞ Et <http://localhost:8080/services/ToDoService?wsdl>


```
Aucune information de style ne semble associée à ce fichier XML. L'arbre du document est affiché ci-dessous.
- <wsdl:definitions name="DomainServiceImplService" targetNamespace="http://domain.formation.esupportail.org/">
- <wsdl:types>
- <xs:schema elementFormDefault="unqualified" targetNamespace="http://domain.formation.esupportail.org/" version="1.0">
  <xs:element name="addTask" type="tns:addTask"/>
  <xs:element name="addTaskResponse" type="tns:addTaskResponse"/>
  <xs:element name="addUser" type="tns:addUser"/>
  <xs:element name="addUserResponse" type="tns:addUserResponse"/>
  <xs:element name="deleteTask" type="tns:deleteTask"/>
  <xs:element name="deleteTaskResponse" type="tns:deleteTaskResponse"/>
  <xs:element name="deleteUser" type="tns:deleteUser"/>
  <xs:element name="deleteUserResponse" type="tns:deleteUserResponse"/>
  <xs:element name="getLastTasksForUser" type="tns:getLastTasksForUser"/>
  <xs:element name="getLastTasksForUserResponse" type="tns:getLastTasksForUserResponse"/>
  <xs:element name="getPublicTasksResponse" type="tns:getPublicTasksResponse"/>
  <xs:element name="getTask" type="tns:getTask"/>
  <xs:element name="getTaskResponse" type="tns:getTaskResponse"/>
  <xs:element name="getTasks" type="tns:getTasks"/>
  <xs:element name="getTasksForUser" type="tns:getTasksForUser"/>
  <xs:element name="getTasksForUserResponse" type="tns:getTasksForUserResponse"/>
  <xs:element name="getTasksResponse" type="tns:getTasksResponse"/>
  <xs:element name="getUser" type="tns:getUser"/>
  <xs:element name="getUserResponse" type="tns:getUserResponse"/>
  <xs:element name="getUsers" type="tns:getUsers"/>
  <xs:element name="getUsersResponse" type="tns:getUsersResponse"/>
  <xs:element name="updateTask" type="tns:updateTask"/>
  <xs:element name="updateTaskResponse" type="tns:updateTaskResponse"/>
- <xs:complexType name="getLastTasksForUser">
```

2 Les services REST

Exercice N°32 : Exposer un service REST

Créer un service REST qui permettra de récupérer au format JSON les 10 dernières tâches d'un utilisateur grâce à l'appel d'une URL de type `/tasks/login`.

Ajouter dans **domainService** la méthode `get10LastTasksForUserLogin()` qui retournera les 10 dernières tâches (en date) de l'utilisateur à partir de son login.

```
public List<Task> get10LastTasksForUserLogin(String uid) {
    User u=getUser(uid);
    return daoService.getLastTasksForUser(u,10);
}
```

On déclarera le service JSON grâce aux annotations `@Path` et `@Produces` et les URL REST qui pourront être appelée grâce aux annotation `@GET` ainsi que `@Path`.

```
[...]
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;

@Path("/JSONServices/")
@Produces("application/json")
public interface DomainService extends Serializable {

[...]
    @GET
    @Path("/users/{login}")
    public User getUser(@PathParam("login") String uid);
    [...]
    @GET
    @Path("/tasks/{login}")
    public List<Task>
    get10LastTasksForUserLogin(@PathParam("login") String uid);
```

Dans le fichier **pom.xml** du module `domain-service` ajouter les dépendances nécessaires.

```
<dependency>
  <groupId>org.esupportail</groupId>
  <artifactId>esup-commons2-rs-cxf</artifactId>
  <version>${esupcommons.version}</version>
```

```
<type>pom</type>
</dependency>
```

Ajouter dans le fichier de configuration *spring* **src/main/resources/META-INF/esup-formation/domain-service-domain.xml**

```
[...]
  xmlns:jaxws="http://cxf.apache.org/jaxws"
  xmlns:jaxrs="http://cxf.apache.org/jaxrs"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
  http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
  http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-3.0.xsd
  http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-3.0.xsd
  http://cxf.apache.org/jaxws
http://cxf.apache.org/schemas/jaxws.xsd
  http://cxf.apache.org/jaxrs
  http://cxf.apache.org/schemas/jaxrs.xsd">

[... ]
  <import resource="classpath:META-INF/cxf/cxf.xml" />
  <import resource="classpath:META-INF/cxf/cxf-extension-soap.xml" />
  <import resource="classpath:META-INF/cxf/cxf-servlet.xml" />
[... ]
  <jaxrs:server id="domainServiceRest" address="/rest">
    <jaxrs:serviceBeans>
      <ref bean="domainService" />
    </jaxrs:serviceBeans>
    <jaxrs:providers>
      <ref bean="jsonProvider" />
    </jaxrs:providers>
  </jaxrs:server>

<bean id="jsonProvider"
class="org.codehaus.jackson.jaxrs.JacksonJaxbJsonProvider" />

</beans>
[...]
```

Pour sérialiser les objets Jax-RS appelle toute les méthodes *get* des objets ce qui va nous poser deux problèmes :

- Une boucle car une tâche référence un utilisateur propriétaire et un user référence une liste de tâches dont il est propriétaire : *getTasks>getOwner>getTaches*. On commentera *User.getTaches()*
- Une *nullPointerException* à corriger.

Ceci sera corrigé dans la prochaine version d'esup-commons et notamment d'esup-blank.

```
org.codehaus.jackson.map.JsonMappingException: (was
java.lang.NullPointerException) (through reference chain:
java.util.ArrayList[0]-
>org.esupportail.formation.domain.beans.Task["owner"]-
>org.esupportail.formation.domain.beans.User["displayLanguage" ])
[... ]
Caused by: java.lang.NullPointerException
  at java.util.Locale.toLowerCase(Locale.java:1060)
  at java.util.Locale.convertOldISOCodes(Locale.java:1083)
  at java.util.Locale.<init>(Locale.java:272)
  at java.util.Locale.<init>(Locale.java:302)
```

```
at
org.esupportail.formation.domain.beans.User.getDisplayLanguage(User.
java:181)
```

On corrigera la classe *User* du module domain-beans

```
public String getDisplayLanguage() {
    if (language!=null){
        Locale locale = new Locale(language);
        return locale.getDisplayLanguage(locale);
    }
    else
        return null;
}
```

Tester :

- ☞ <http://localhost:8080/services>
- ☞ <http://localhost:8080/services/rest? wadl& type=xml>
- ☞ <http://localhost:8080/services/rest/JSONServices/tasks>
- ☞ <http://localhost:8080/services/rest/JSONServices/tasks/cbissler>

Available SOAP services:

DomainService <ul style="list-style-type: none"> • getLastTasksForUser • getTasksForUser • getPublicTasks • get10LastTasksForUserLogin • getTasks • addTask • getTask • deleteTask • getUsers • updateTask • deleteTask • getUser • addUser 	Endpoint address: http://localhost:8080/services/ToDoService WSDL: {http://domain.formation.esupportail.org/}DomainServiceImplService Target namespace: http://domain.formation.esupportail.org/
--	--

Available RESTful services:

Endpoint address: http://localhost:8080/services/rest WADL: http://localhost:8080/services/rest?_wadl&type=xml

Aucune information de style ne semble associée à ce fichier XML. L'arbre du document est affiché ci-dessous.

```
-<application>
  <grammars/>
  <resources base="http://localhost:8080/services/rest">
    <resource path="/JSONServices/">
      <resource path="tasks">
        <method name="GET">
          <response>
            <representation mediaType="application/json"/>
          </response>
        </method>
      </resource>
      <resource path="tasks/{login}">
        <param name="login" style="template" type="xs:string"/>
        <method name="GET">
          <request/>
          <response>
            <representation mediaType="application/json"/>
          </response>
        </method>
      </resource>
      <resource path="user/{login}">
        <param name="login" style="template" type="xs:string"/>
        <method name="GET">
          <request/>
          <response>
            <representation mediaType="application/json"/>
          </response>
        </method>
      </resource>
    </resources>
  </resources>
</application>
```

On obtient alors des flux qui ressemblent à cela :

```
[{"id":3,"owner":{"language":null,"id":1,"displayName":null,"display
name":null,"login":"cbissler","admin":false},"date":131664960000
0,"description":"","title":"essai","publicTask":true}, {"id":10,"owne
r":{"language":null,"id":2,"displayName":null,"displayName":null
,"login":"tartempion","admin":false},"date":1318003628462,"descripti
on":"","title":"rk - Titre de tache
bidon","publicTask":true}, {"id":11,"owner":{"language":null,"id":2,"
displayName":null,"displayName":null,"login":"tartempion","admin
```

```
":false}, "date":1318003629540, "description":"","title":"pc - Titre  
de tache bidon", "publicTask":true}]
```

Nous verrons comment interroger ce *webservice* et ce service *REST* dans le chapitre suivant.

Déploiement en Portlet

Nous allons maintenant voir comment développer et déployer une portlet dans un environnement *Esup-portail*.

Le but de cette formation n'étant pas d'installer un portail, nous utiliserons le portail pré-installé sur la machine virtuelle.

1 Création d'un nouveau projet pour la portlet

Exercice N°33 : Déploiement et installation d'une portlet

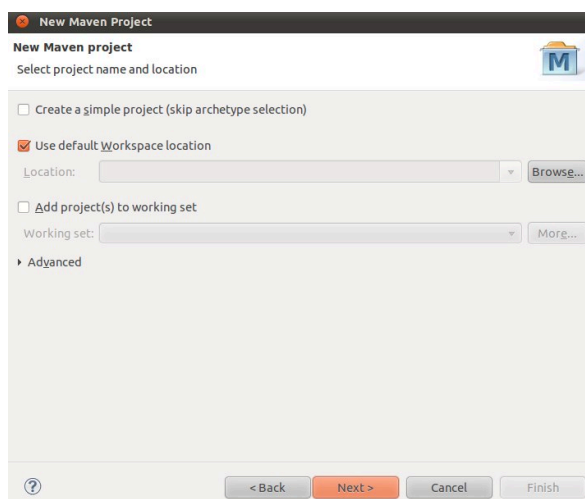
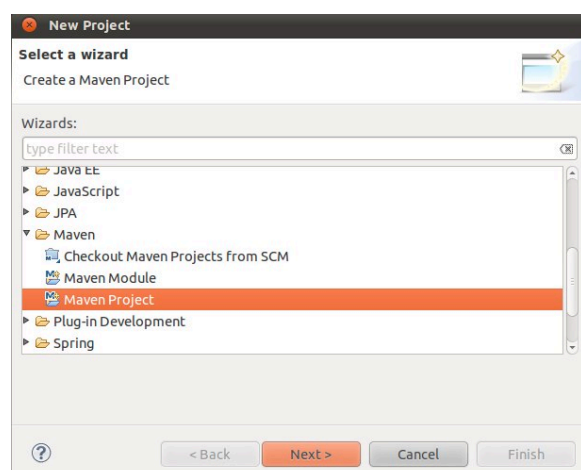
Comme précédemment, créer un nouveau projet à partir de l'archetype blank. Déployer la portlet dans le portail et tester l'application fournie par défaut.

Pour cela, créer un nouveau projet *maven*

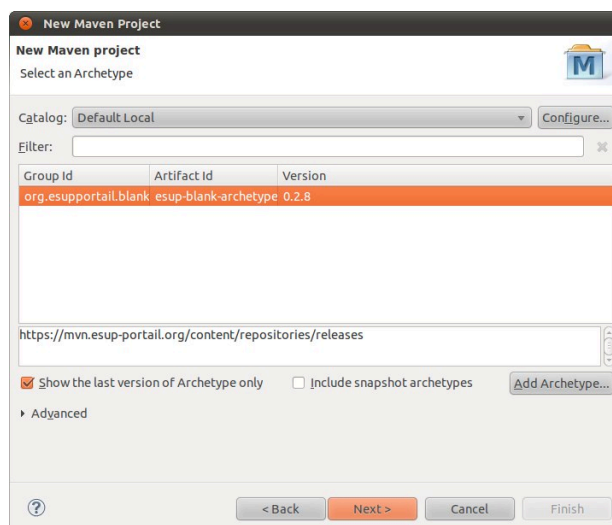
La structure du projet va être construite à partir du même archétype *maven* que celui du projet précédent et dont voici les paramètres :

- `archetypeRepository=https://mvn.esup-portail.org/content/repositories/releases`
- `archetypeGroupId=org.esupportail.blank`
- `archetypeArtifactId=esup-blank-archetype`
- `archetypeVersion=0.2.8`
- `groupId=org.esupportail.formation`
- `artifactId=esup-portlet-formation`
- `package=org.esupportail.formation.portlet`
- `version=0.0.1-SNAPSHOT`

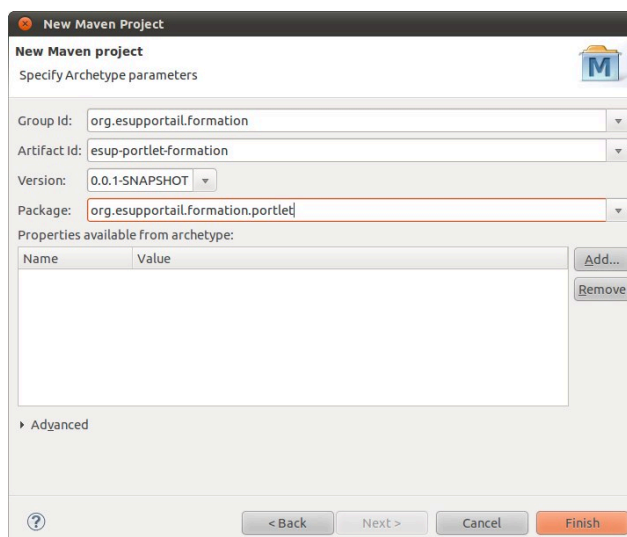
 **File > new > project > Project... > Maven > Maven project**



🖱️ Choix de l'archetype



🖱️ Création du projet



Soit l'équivalent en ligne de commande :

```
mvn archetype:generate -B
-DarchetypeRepository=https://mvn.esup-
portail.org/content/repositories/releases
-DarchetypeGroupId=org.esupportail.blank
-DarchetypeArtifactId=esup-blank-archetype
-DarchetypeVersion=0.2.8
-DgroupId=org.esupportail.formation
-DartifactId=esup-portlet-formation
-Dpackage=p org.esupportail.formation.portlet
-Dversion=0.0.1-SNAPSHOT
```

puis...

```
mvn eclipse:eclipse
```

On constate alors qu'*eclipse* fait un *Build maven dependencies* et un *Build maven project*. *Eclipse* a alors construit les différents projets *esup-portlet-formation-xxx*.

```

  ▶ esup-portlet-formation
  ▶ esup-portlet-formation-dao
  ▶ esup-portlet-formation-domain-beans
  ▶ esup-portlet-formation-domain-services
  ▶ esup-portlet-formation-utils
  ▶ esup-portlet-formation-web-jsf-mixed
  ▶ esup-portlet-formation-web-jsf-servlet
  ▶ esup-portlet-formation-web-springmvc-portlet
  ▶ esup-portlet-formation-web-springmvc-servlet

```

Supprimer les modules qui nous seront inutiles à savoir :

- dao
- web-jsf-mixed
- web-jsf-servlet
- web-springmvc-servlet

Commenter ou supprimer les modules dans le fichier **pom.xml** du projet racine :

```

<modules>
  <!-- <module>esup-formation-dao</module> -->
  <module>esup-formation-utils</module>
  <module>esup-formation-domain-beans</module>
  <module>esup-formation-domain-services</module>
  <!-- <module>esup-formation-web-jsf-mixed</module> -->
  <!-- <module>esup-formation-web-jsf-servlet</module> -->
  <!--
  <module>esup-formation-web-springmvc-servlet</module>-->
  <module>esup-formation-web-springmvc-portlet</module>
</modules>

```

2 Premier lancement

2.1 Création de l'archive war

Créer le fichier `config.properties` qui restera vide dans un premier temps dans le répertoire `/src/main/resources` du module `web-springmvc-portlet`

Corriger le `/src/main/webapp/jsp/view.jsp` en remplaçant

```

| ${userFromEC2.id} !

```

... par...

```

| ${userFromEC2.login} !

```

Ceci sera corrigé dans la prochaine version d'esup-commons et notamment d'esup-blank.

Sur le module racine lancer

```

| mvn install

```

Sur le module `web-springmvc-portlet` lancer

```

| mvn package

```

Vérifier qu'un fichier `.war` a bien été créé.

2.2 Déploiement de la portlet dans le portail

Ouvrir un terminal (raccourci sur le bureau)

```
cd /home/esup/src/esup-install
ant portlet.deploy -DportletApp=/home/esup/workspace/esup-portlet-formation/esup-portlet-formation-web-springmvc-portlet/target/esup-portlet-formation.war
```

Déclarer le contexte dans `/home/esup/tomcat/conf/server.xml`

```
<Context path="/esup-portlet-formation" docBase="/esup-portlet-formation" reloadable="false">
  <Manager pathname="" />
</Context>
```

Démarrer le portail

```
cd /home/esup/
./start.sh
```

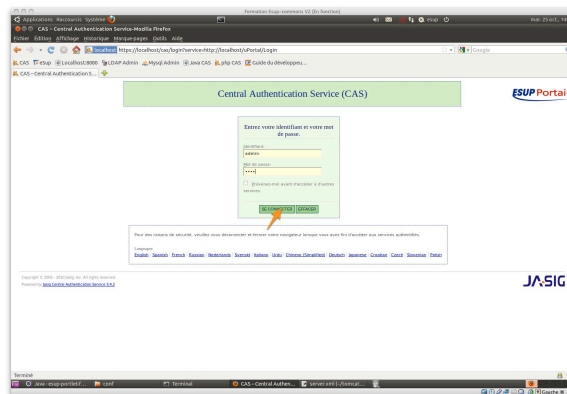
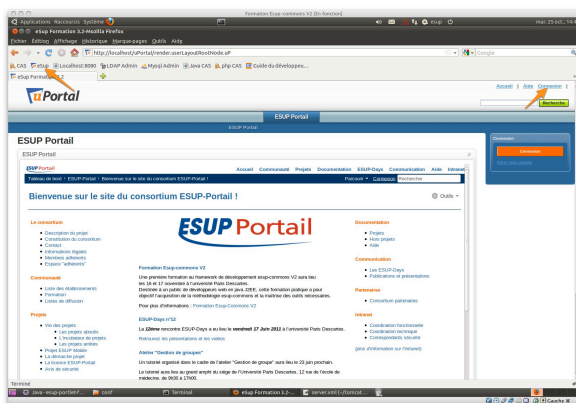
Lancer un navigateur et se connecter sur :

<http://localhost/uPortal>

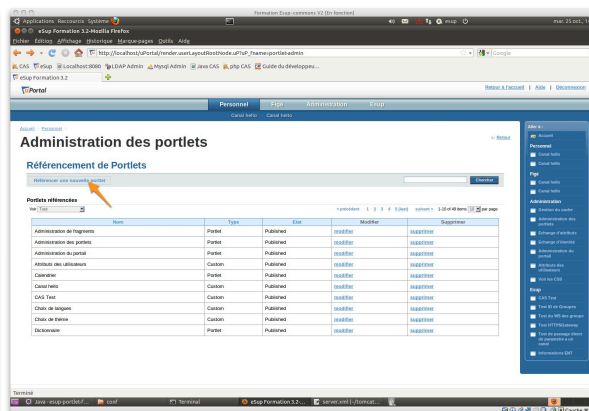
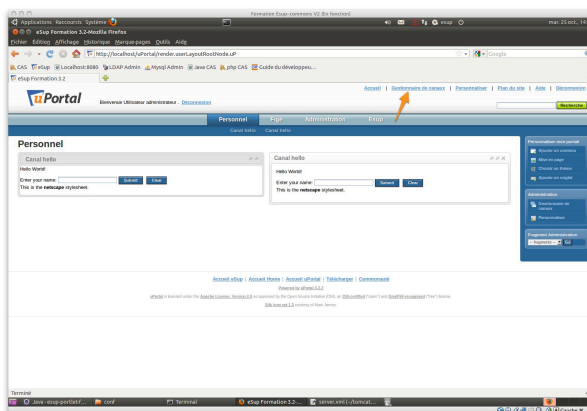
2.3 Publication de la portlet

Nous allons maintenant publier la portlet via l'interface du portail.

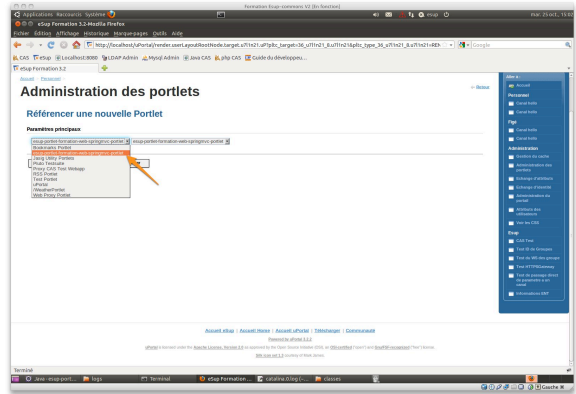
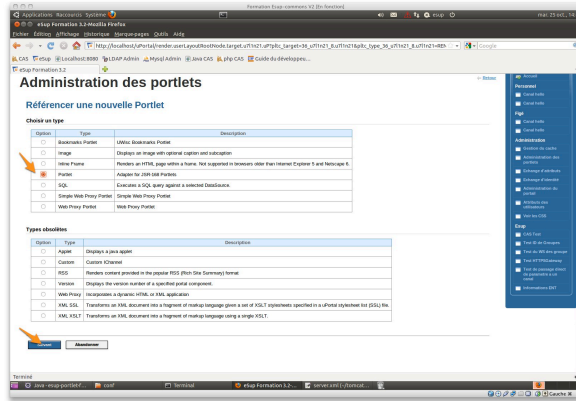
🖱️ **Se connecter au portail**



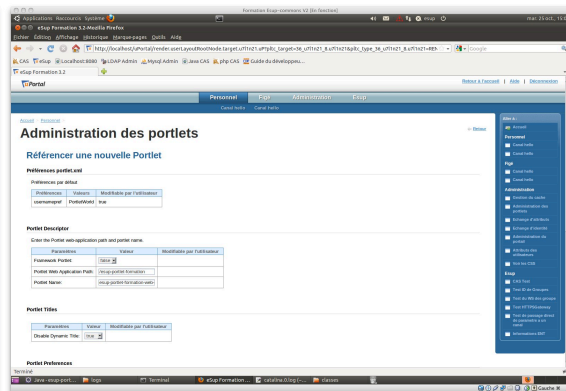
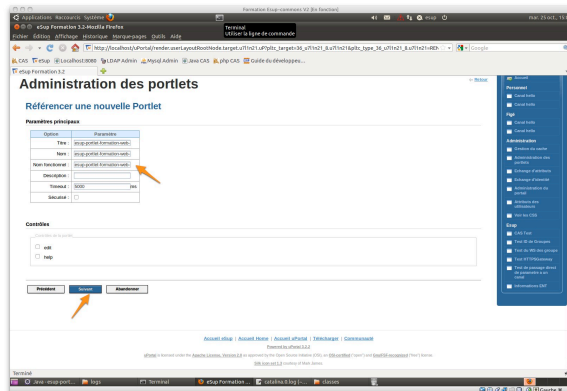
🖱️ **Choisir "Gestionnaire de canaux" et choisir "Referencer une nouvelle portlet"**



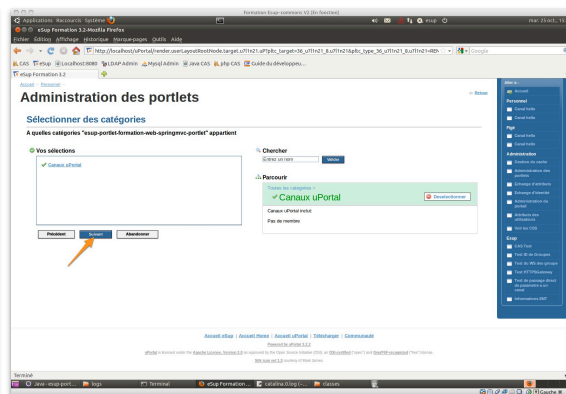
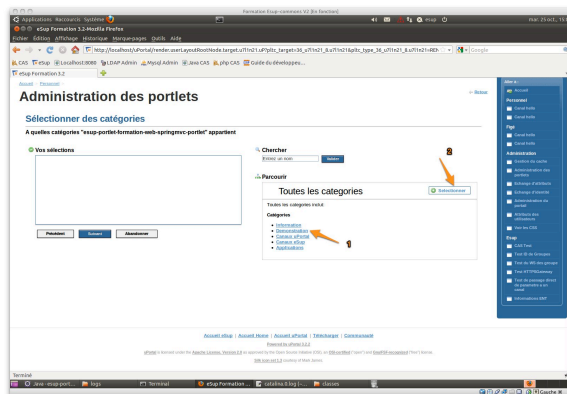
🖱️ **Choisir le type "Portlet" et sélectionner notre portlet**



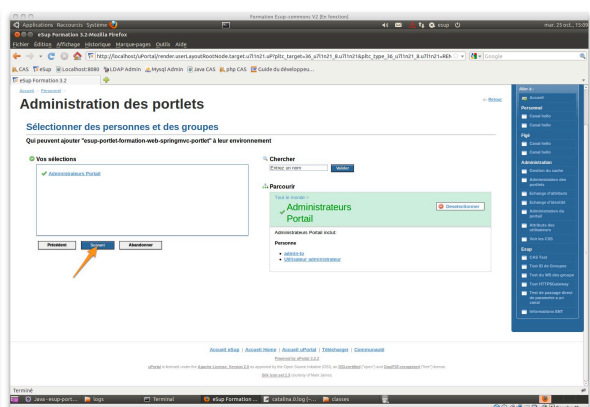
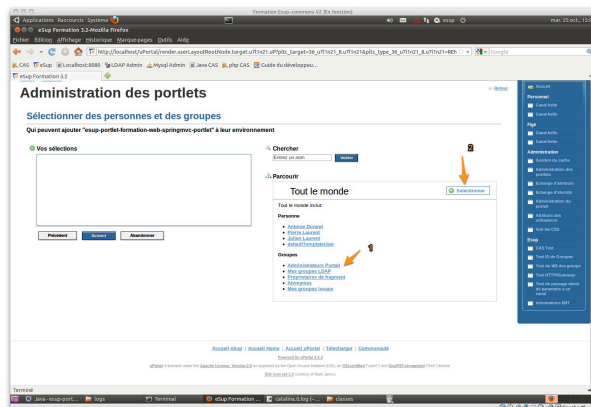
Spécifier un nom fonctionnel (le reste des informations est récupéré automatiquement) et laisser le reste des paramètres proposés par défaut



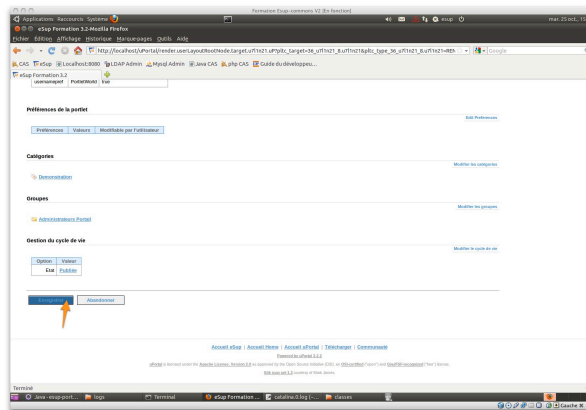
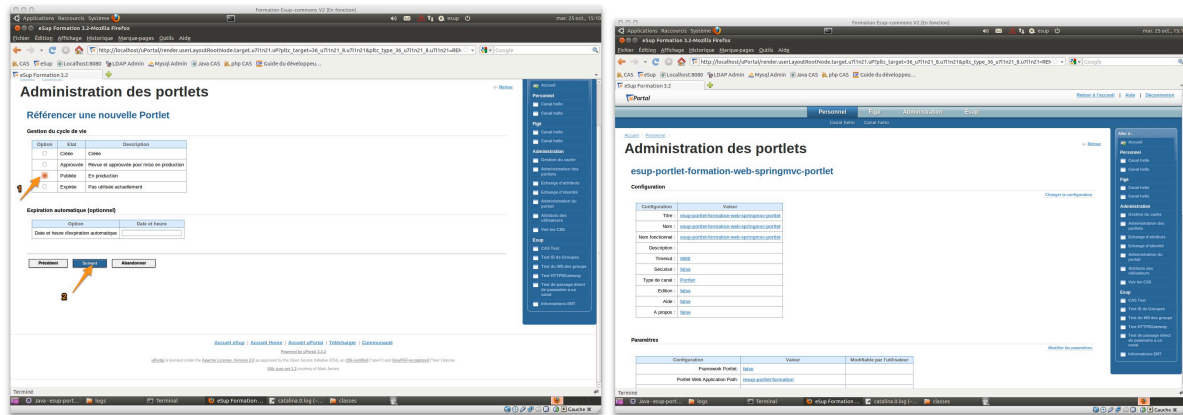
Choisir la catégorie dans laquelle se trouvera le canal,



Choisir la population autorisée à accéder au canal



Choisir l'état "publiée" et valider

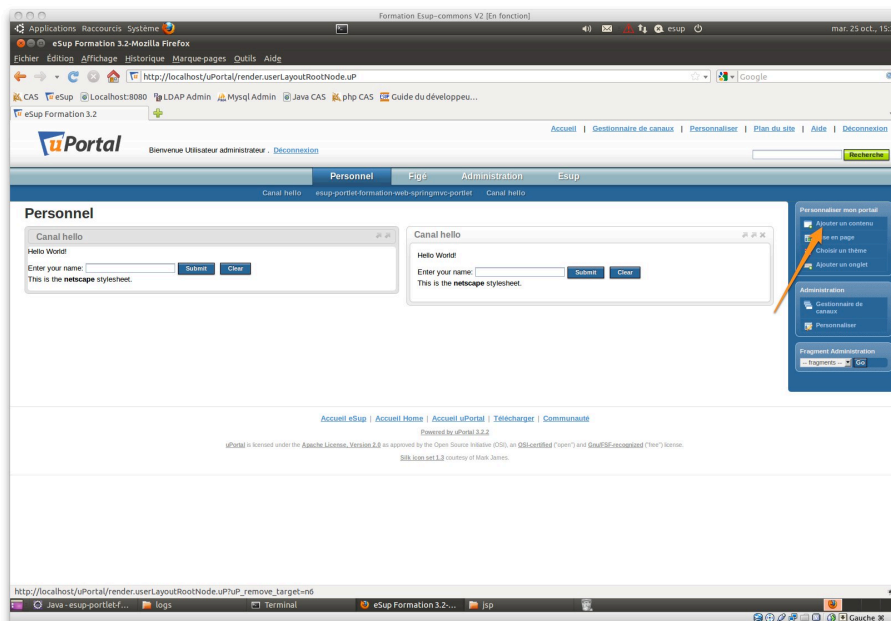


2.4 Test de la portlet

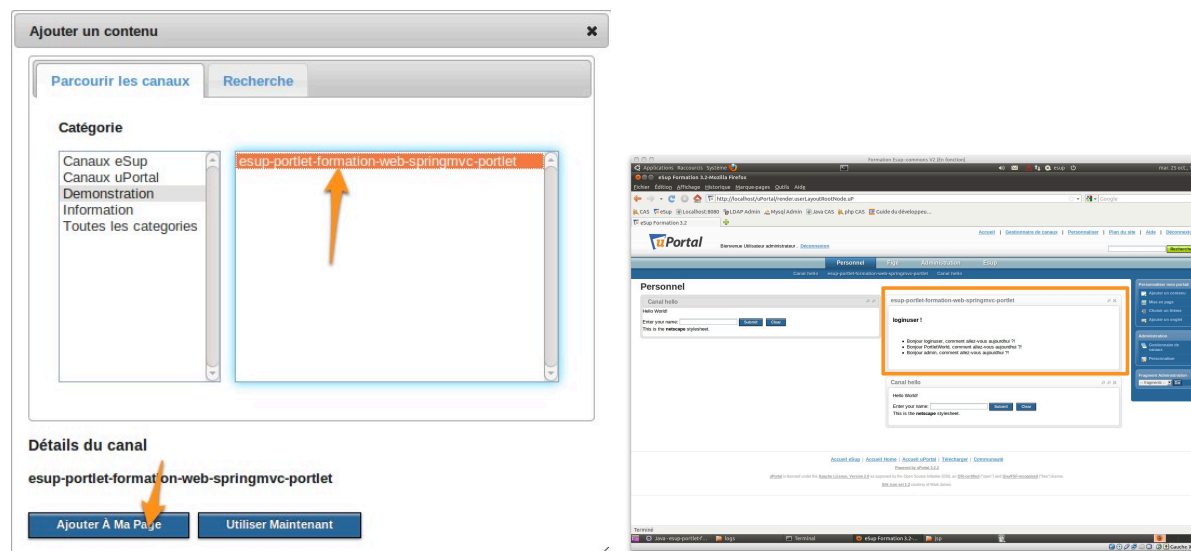
2.4.1 Ajout de la portlet à l'environnement

Il nous reste à ajouter la portlet à notre environnement.

Choisir "Ajouter un contenu"



Choisir le canal et l'ajouter



2.4.2 Récupération d'informations

Les 3 identifiants utilisateurs sont récupérés de la manière suivante :

Dans view.jsp

```
[...]
<ul>
    <li><spring:message code="view.helloString"
arguments="{userFromEC2.login}"/></li>
    <li><spring:message code="view.helloString"
arguments="{usernamePref}"/></li>
    <li><spring:message code="view.helloString"
arguments="{remoteUser}"/></li>
</ul>
[...]
```

Dans WebController.java

```
[...]
final PortletPreferences prefs = request.getPreferences();
String usernamePref = prefs.getValue(PREF_USERNAME,
DEFAULT_USERNAME);
model.put("usernamePref", usernamePref);

String remoteUser = request.getRemoteUser();
model.put("remoteUser", remoteUser);

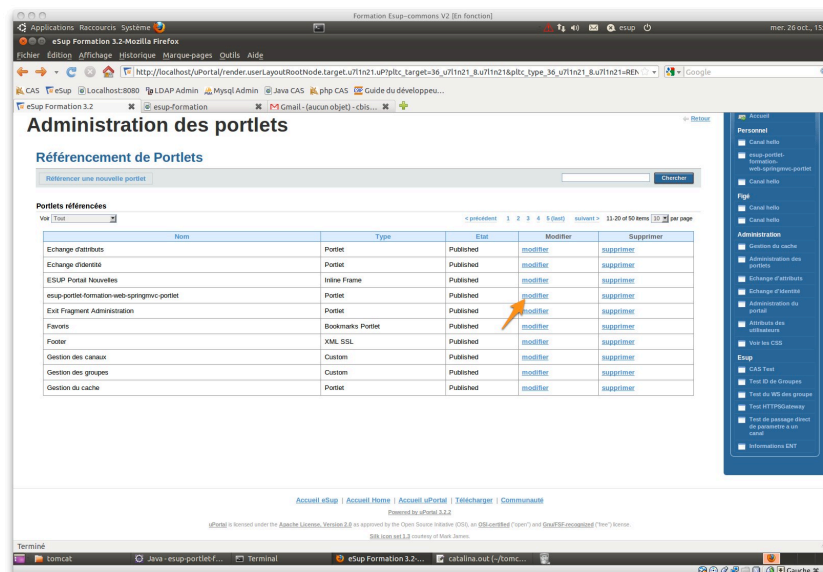
User userFromEC2 = authenticator.getUser();
model.put("userFromEC2", userFromEC2);
[...]
```

On constate que le troisième identifiant affiché est celui de l'utilisateur connecté au portail, ici *admin*.

Le deuxième identifiant affiché est celui déclaré dans **src/mais/webapp/WEB-INF/portlet.xml**

```
<portlet-preferences>
  <preference>
    <name>usernamepref</name>
    <value>PortletWorld</value>
  </preference>
</portlet-preferences>
```

On pourra changer ces valeurs par défaut dans le fichier **src/main/webapp/WEB-INF/portlet.xml** mais également le surcharger via l'interface de gestion des canaux.



Et ajouter une préférence usernamepref et donner lui une valeur différente.

Préférences portlet.xml

Préférences	Valeurs	Modifiable par l'utilisateur
usernamepref	PortletWorld	true
wSDLLocation	http://localhost:9090/services/DomainService?wsdl	true

Préférences de la portlet

Préférences	Valeurs	Modifiable par l'utilisateur
usernamepref	Céline	true

Enfin le premier identifiant affiché est l'identifiant de la personne authentifiée sur la portlet grâce à l'authenticator.

En effet on déclare dans le **src/main/resources/default.properties** du module **springmvc-portlet** une authentification *Offline* :

```
auth.bean=OfflineFixedUserAuthenticationService
```

Et dans **src/main/resources/META-INF/esup-portlet-formation-domain-services-auth.xml** du module **domain-service**

```
<bean id="OfflineFixedUserAuthenticationService"
class="org.esupportail.commons.services.authentication.OfflineFixedUserAuthenticationService">
  <property name="authId" value="loginuser" />
  <property name="authType" value="cas" />
</bean>
```

Changer l'authentification et tester une authentification *CAS* en déclarant dans le **src/main/resources/config.properties**

```
auth.bean=portletAuthenticationService
cas.url=http://localhost/cas
```

Déclarer dans **src/main/webapp/portlet.xml**

```
[...]
```

```

        </preference>
    </portlet-preferences>
</portlet>
<user-attribute>
    <description>uid</description>
    <name>uid</name>
</user-attribute>
</portlet-app>

```

Vérifier que vous obtenez le même identifiant que le *remoteUser*



3 Développement d'un portlet à l'aide de *spring-MVC*

Exercice N°34 : Interrogation des webservices

Réaliser une portlet qui interrogera les webservices réalisés dans le chapitre précédent afin d'afficher les *n* dernières tâches de l'utilisateur connecté.

On positionnera *n* et l'URL du webService au niveau des préférences de la portlet.

3.1 Utilisation du webservice

3.1.1 Démarrage des webservices

Faire tourner l'application sur le port *9090*.

Dans le **pom.xml** du projet racine *esup-formation*

```

<plugin>
  <groupId>org.mortbay.jetty</groupId>
  <artifactId>maven-jetty-plugin</artifactId>
  <version>6.1.26</version>
  <configuration>
    <contextPath></contextPath>
    <connectors>
      <connector
implementation="org.mortbay.jetty.nio.SelectChannelConnector">
        <port>9090</port>
      </connector>
    </connectors>
  </configuration>
</plugin>

```

Dans le fichier **web.xml** du module *web-jsf-servlet* on adaptera les ports des URL de retour.

```

<javaee:param-value>http://localhost:9090</javaee:param-value>

```

Démarrer l'application

```
2011-10-27 12:50:26.153:INFO::Started
SelectChannelConnector@0.0.0.0:9090
[INFO] Started Jetty Server
```

3.1.2 Génération des objets métiers à partir du WSDL

Ajouter au fichier **pom.xml** du module `domain-beans` le plugin `cxf-codegen-plugin` qui va nous permettre de générer les objets métiers à partir du fichier de description du `webservice`.

```
<build>
  <finalName>${pom.artifactId}</finalName>
  <plugins>
    <plugin>
      <groupId>org.apache.cxf</groupId>
      <artifactId>cxfr-codegen-plugin</artifactId>
      <executions>
        <execution>
          <id>generate-sources</id>
          <phase>generate-sources</phase>
          <configuration>
            <wsdlOptions>
              <wsdlOption>
                <wsdl>http://localhost:9090/services/ToDoService?wsdl</wsdl>
              </wsdlOption>
            </wsdlOptions>
          </configuration>
          <goals>
            <goal>wsdl2java</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

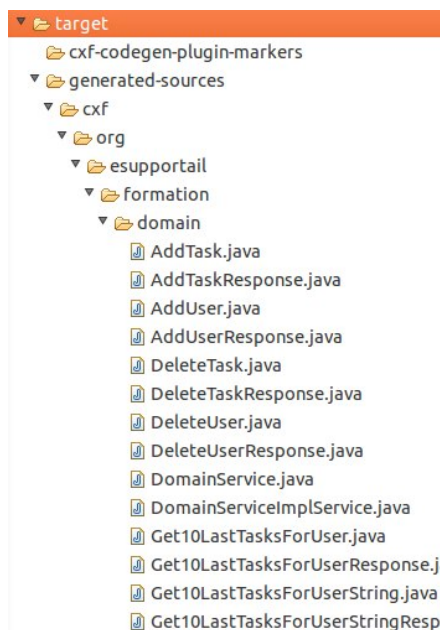
Lancer la commande *maven*

```
Mvn generate-sources
```

On obtient alors

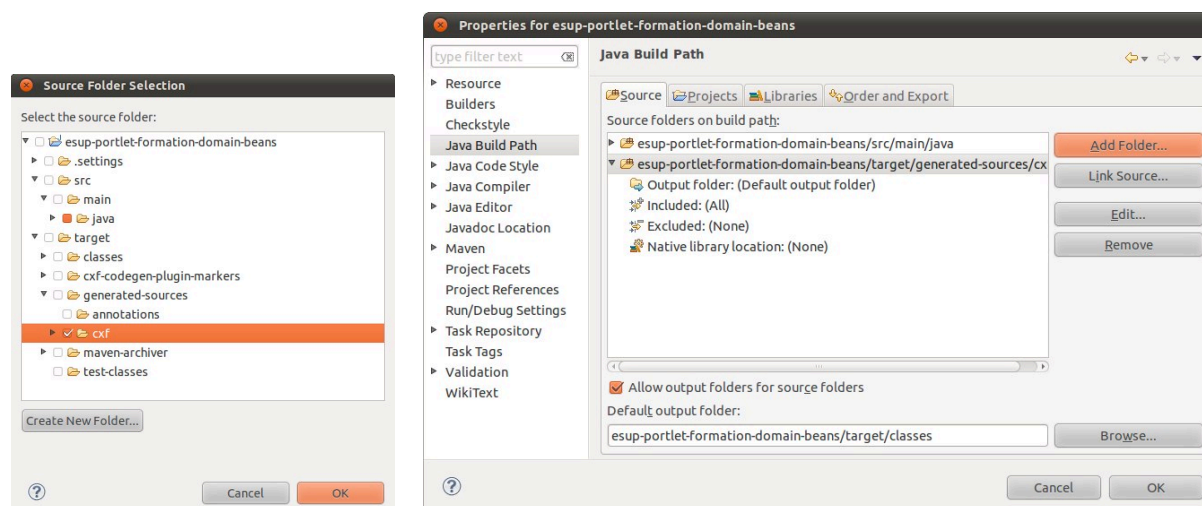
```
[INFO] [cxfr-codegen:wsdl2java {execution: generate-sources}]
[INFO] -----
[INFO] BUILD SUCCESSFUL
[INFO] -----
[INFO] Total time: 25 seconds
[INFO] Finished at: Tue Oct 25 16:33:06 CEST 2011
[INFO] Final Memory: 21M/56M
[INFO] -----
```

Vérifier après un *refresh* de **target** la présence du répertoire **generated-sources**



Ajouter le répertoire **target/generated-sources/cxf** comme *source folder* dans les propriétés de votre projet *eclipse*.

☞ **Clique droit > properties**



3.1.3 Interrogation du webservice

Dans la classe **DomainServiceImpl** du module **domain-service**

```
public List<Task> getTasks(String login,String wsdl,int limit){
    org.esupportail.formation.domain.DomainService ds =
getWs(wsdl);
    org.esupportail.formation.domain.User u=new
    org.esupportail.formation.domain.User();
    u.setLogin(login);
    return ds.getLastTasksForUser(u,limit);
}

public List<Task> getAllTasks(String wsdl){
    org.esupportail.formation.domain.DomainService ds =
    getWs(wsdl);
    return ds.getTasks();
}
```

```
private org.esupportail.formation.domain.DomainService getWs(String
wsdl) {
    org.esupportail.formation.domain.DomainService ds = null;
    try {
        ds = new DomainServiceImplService(new
URL(wsdl)).getDomainServiceImplPort();
    } catch (MalformedURLException e) {
        e.printStackTrace();
    }
    return ds;
}
```

Dans l'interface *DomainService* du module domain-service

```
public List<Task> getTasks(String login,String wsdl,int limit);
public List<Task> getAllTasks(String wsdl);
```

Dans la méthode *renderView()* classe *WebController* du module web-springmvc-portlet

```
[...]
private static final String PREF_WSDL = "wsdlLocation";
private static final String PREF_LIMIT = "nbLimit";
private static final String PREF_URL_TASK = "urlTask";
@Autowired
private Authenticator authenticator;

@Autowired
private DomainService domainService;

@RequestMapping("VIEW")
protected ModelAndView renderView(RenderRequest request,
RenderResponse response) throws Exception {
    [...]
    String wsdl = prefs.getValue(PREF_WSDL,null);
    String limit = prefs.getValue(PREF_LIMIT,null);

    List<Task> list= domainService.getTasks(remoteUser,
wsdl,Integer.parseInt(limit));
    model.put("taskList", list);

    // Affichage dans les logs
    System.out.println("Tâche de "+remoteUser);
    for (Task task : list) {
        System.out.println(task.getId()+" : "+task.getTitle());
    }
    System.out.println("Toutes les tâches");
    List<Task> list2= domainService.getAllTasks(wsdl);
    for (Task task : list2) {
        System.out.println(task.getId()+" : "+task.getTitle());
    }
    return new ModelAndView("view", model);
}
```

Dans `src/main/webapp/WEB-INF/portlet.xml`

```
<portlet-preferences>
    <preference>
        <name>usernamepref</name>
        <value>PortletWorld</value>
    </preference>
    <preference>
        <name>wsdlLocation</name>
        <value>http://localhost:9090/services/ToDoService?wsdl</value>
    </preference>
</portlet-preferences>
```



```

        <name>nbLimit</name>
        <value>8</value>
    </preference>
</portlet-preferences>

```

Lancer un premier test en déployant à nouveau le *war*

```

cd /home/esup/src/esup-install
ant portlet.deploy -DportletApp=/home/esup/workspace/esup-portlet-formation/esup-portlet-formation-web-springmvc-portlet/target/esup-portlet-formation.war

```

Relancer le portail

```

cd /home/esup
./stop.sh
./start.sh

```

Vérifier dans les logs `/home/esup/tomcat/logs/catalina.out` que le webservice a bien été appelé.

```

Tâche de admin
1 : tache 1
3 : Tâche 3
Toutes les tâches
1 : tache 1
2 : tache 2
3 : Tâche 3
4 : ua - Titre de tache bidon

```

3.1.4 Affichage des resultats

Dans `src/main/webapp/jsp/view.jsp` du module `web-springmvc-portlet`

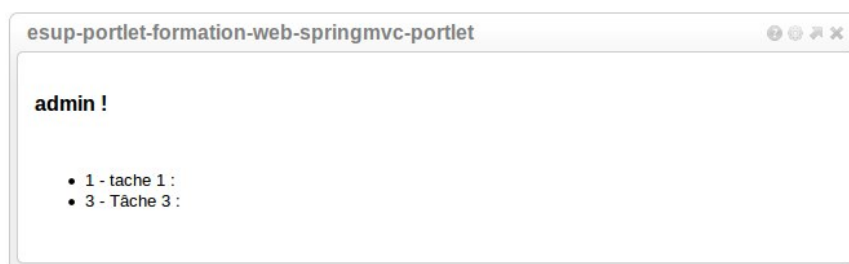
```

<ul>
  <c:forEach var="task" items="${taskList}" >
    <li>${task.id} - ${task.title} : ${task.description}</li>
  </c:forEach>
</ul>

```

3.1.5 Tests et amélioration

Tester en déployant à nouveau le *war* et en relançant le portail.



Améliorer en ajoutant un lien vers la page de détail de la tâche de notre application servlet.

On ajoutera dans `src/main/webapp/WEB-INF/portlet.xml`

```

<preference>
  <name>urlTask</name>
  <value>http://localhost:9090/stylesheets/welcome.xhtml?args=taskId%3d</value>
</preference>

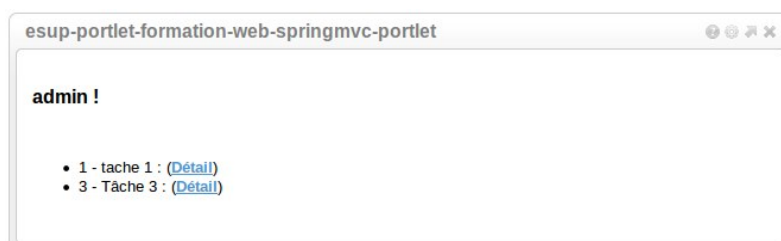
```

... dans la classe **WebController** du module `web-springmvc-portlet`

```
[...]
private static final String PREF_URL_TASK = "urlTask";
String urlTask = prefs.getValue(PREF_URL_TASK, null);
model.put("urlTask", urlTask);
```

... et dans `src/main/webapp/WEB-INF/jsp/view.jsp` du module `web-springmvc-portlet`

```
<ul>
  <c:forEach var="task" items="${taskList}" >
    <li>${task.id} - ${task.title} : ${task.description}
    (<a href="${urlTask}${task.id}">Détail</a></li>
  </c:forEach>
</ul>
```



Pour aller plus loin...

On peut rendre le nombre limite de tâche éditable par l'utilisateur via le menu *edit* du portail

3.2 Utilisation du service REST

Exercice N°35 : Interrogation du service REST

Faire afficher la liste des tâches de l'utilisateur récupérée grâce au service *Rest* de notre application. Pour cela on utilisera des méthodes *jQuery*.

On ajoutera dans `src/main/webapp/WEB-INF/portlet.xml`

```
<preference>
  <name>urlJSON</name>
  <value>http://localhost:9090/services/rest/JSONServices/tasks/</value>
</preference>
```

... dans la classe **WebController** du module `web-springmvc-portlet`

```
[...]
private static final String PREF_URL_JSON = "urlJSON";
String urlJSON = prefs.getValue(PREF_URL_JSON, null);
model.put("urlJSON", urlJSON);
```

... et dans `src/main/webapp/WEB-INF/jsp/view.jsp` du module `web-springmvc-portlet`

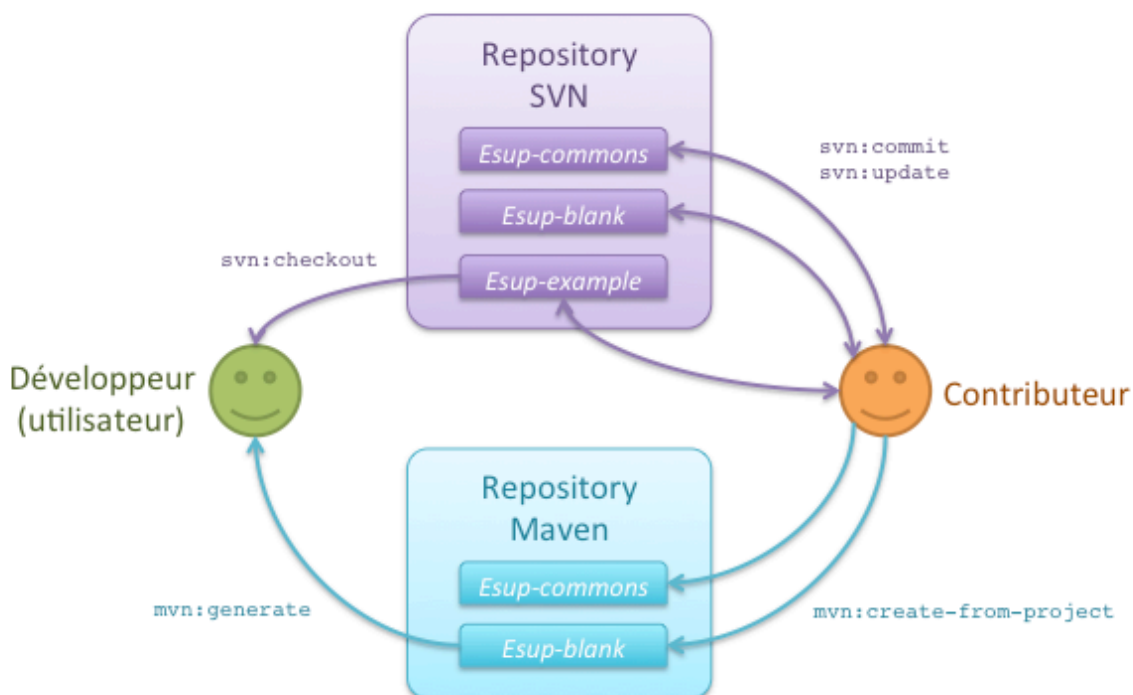
```
<script type="text/javascript">
function getTasks(){
  $('#tasksJSON').empty();
  alert('${urlJSON}${remoteUser}');
  $.getJSON('${urlJSON}${remoteUser}', setTasks);
}
function setTasks(data) {
  var taskList = data;
  alert(data);
```

```
        var ulNode = $('<ul></ul>');
        $.each(data.items, function(i,item){
            var taskNode = $('<li></li>');
            taskNode.append(item['id'] + ' - ' + item['title']
+ ' : ' + item['description']);
            ulNode.append(taskNode);
        });
        $('#tasksJSON').append(ulNode);
    }
</script>

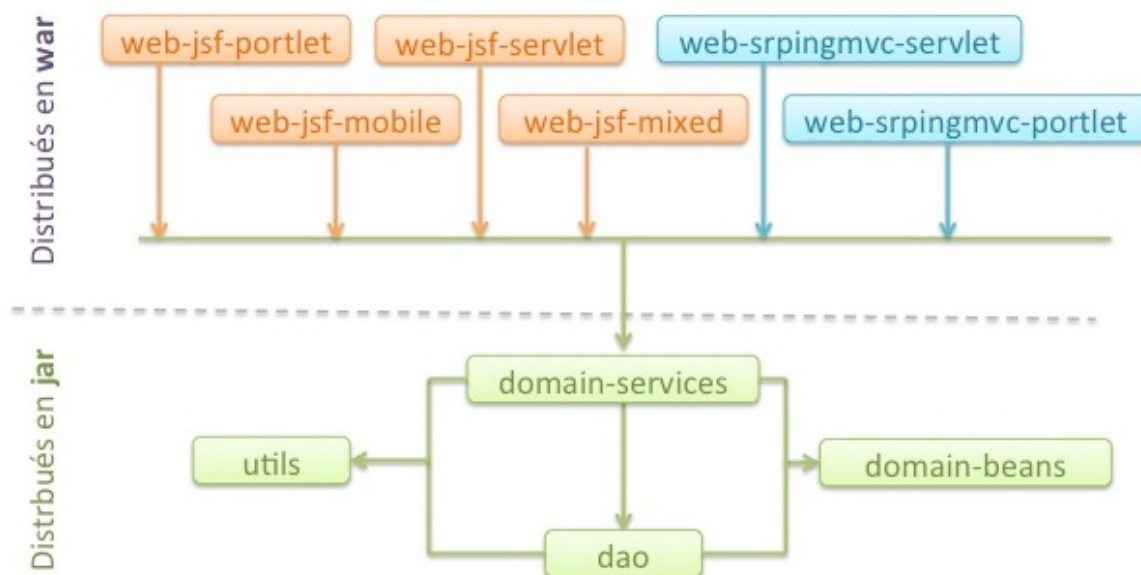
<a href="#" onclick="getTasks();">Actualiser en JSON</a> <div
id="tasksJSON">
```

! A debugger ☺

Annexes



Utilisation des différents repository esup-commons



Les différents modules d'un projet esup-commons

README :

Une image VirtualBox (faite sur une version 3.0.10)

Elle contient :

- Un jdk sun 6 (sun-java6-jdk : /usr/lib/jvm/java-6-sun) avec les commandes java et javac dans le PATH
- Un Ant (apache-ant-1.7.1 : /usr/local/ant) avec la commande ant dans le PATH
- Un Tomcat (apache-tomcat-6.0.24 : /usr/local/tomcat)
- Un serveur MYSQL (mysql-server)
- Un serveur LDAP (slapd)
- Des certificats HTTPS : /etc/ssl/private/ssl-cert-snakeoil.key et /etc/ssl/certs/ssl-cert-snakeoil.pem
- Un keystore java : /etc/ssl/certs/local.keystore
- Un serveur APACHE2 avec mod_proxy et mod_proxy_ajp
- Les modules APACHE2
- Un moteur PHP (php5) avec LDAP, mysql
- Une interface d'admin phpldapadmin
- Une interface d'admin phpmyadmin

Compte Utilisateur :

- Console : login "esup" mot de passe "esup"
- Mysql : login "root" mot de passe "esup"
- Mysql : login "uportal" mot de passe "uportal"
- Administrateur Ldap : dn "cn=admin,dc=univ,dc=fr" mot de passe "esup"
- Bind LDAP anonyme autorisé
- Compte utilisateur LDAP :
 - login "uid=admin,ou=People,dc=univ,dc=fr" mot de passe "esup"
 - Enseignants (X de 0 à 100) : "uid=ensX,ou=People,dc=univ,dc=fr" mot de passe "esup"
 - Etudiant (X de 0 à 100) : "uid=etudX,ou=People,dc=univ,dc=fr" mot de passe "esup"
 - Personnel (X de 0 à 100) : "uid=persX,ou=People,dc=univ,dc=fr" mot de passe "esup"

Installation de la Machine Virtuelle

Légende

1 Titre niveau 1


1.1 Titre niveau 2

1.1.1 Titre niveau 3


Exercice N°36 : Titre

Sujet de l'exercice

Normal

 **Navigation > de > ce > style**

 **Tips, astuces et gain de temps**

 **http://urlquelconque**

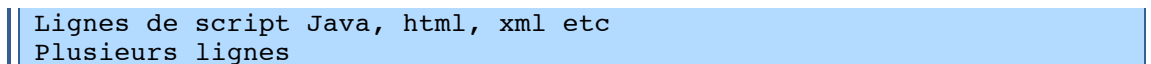
Nom de fichier ou chemin/comme/ceci

Nom de projet ou de module Maven

Nom de classe ou package

Nom de méthode ou de variable

 Ligne de commande ou run eclipse

 Lignes de script Java, html, xml etc
Plusieurs lignes

 Ligne de log
Plusieurs lignes

 *Alerte*

 *Note*

 *Infos*

-

Index

INTRODUCTION	2
SOMMAIRE	3
INDEX DES EXERCICES	6
PRISE EN MAIN DE L'ENVIRONNEMENT	8
1 LA MACHINE VIRTUELLE	8
1.1 PRESENTATION	8
1.2 INSTALLATION DE LA MACHINE VIRTUELLE	8
1.3 DEMARRAGE	8
2 L'ENVIRONNEMENT DE DEVELOPPEMENT ECLIPSE	9
3 DECOUVERTE AVEC ESUP-EXAMPLE	9
3.1 CHECKOUT DEPUIS SVN	9
Exercice N°1 : Récupération d'un projet depuis SVN	9
3.2 ORGANISATION DES FICHIERS	10
3.3 FONCTIONNEMENT EN MODULES MAVEN	11
3.4 DEMARRAGE	13
Exercice N°2 : Lancement d'une application <i>Maven</i>	13
CREATION D'UN PROJET	16
Exercice N°3 : Création d'un projet à partir d'un archetype <i>maven</i>	16
1 CREATION D'UN PROJET MAVEN A PARTIR DE ESUP-BLANK	16
2 UN COUP DE MENAGE...	18
2.1 PREMIER LANCEMENT	18
2.2 MENAGE DANS LES LIBRAIRIES JSF	20
2.3 MENAGE DES MODULES INUTILES	21
BEANS SPRING	22
1 LE FICHIER DE CONFIGURATION PRINCIPAL	22
2 L'INJECTION	22
Exercice N°4 : Instanciation d'un bean simple	23
3 ACCES AUX PARAMETRES DE CONFIGURATION	25
Exercice N°5 : Personnalisation des configurations grâce à l'injection	25
GESTION DES LOGS ET TESTS UNITAIRES	27
1 GESTION DES LOGS	27
1.1 UTILISATION DANS LE CODE JAVA	27
1.2 ACTIVATION DU MECANISME DE LOG	27
2 LES TEST UNITAIRES	27
2.1 EXECUTION DES TESTS UNITAIRES VIA <i>MAVEN</i>	28
Exercice N°6 : Test unitaire simple	28
Exercice N°7 : Test unitaire avancé	30
2.2 EXECUTION DES TESTS UNITAIRES DANS <i>ECLIPSE</i>	32
ACCES AUX DONNEES	33
Exercice N°8 : Création d'un objet métier simple.	33
1 L'OBJET METIER	33
2 LA COUCHE DAO	33

3 LA COUCHE SERVICES	35
4 PREMIERS TESTS D'ECRITURE ET LECTURE EN BASE	37
Exercice N°9 : Création d'une relation entre objets métiers	39
Exercice N°10 : Test de la couche <i>domain</i> dans un test unitaire	39
LES VUES	41
1 JSF ET SES LIBRAIRIES	41
2 FACELET	41
Exercice N°11 : Ajout d'un menu via un template facelet	41
3 PAGES ET NAVIGATION	42
Exercice N°12 : Ajout d'une nouvelle page avec règle de navigation	42
Exercice N°13 : Parcours d'un tableau	43
INTERNATIONALISATION	47
1 CONFIGURATION	47
2 DECLARATION ET UTILISATION DES ENTREES	47
Exercice N°14 : Déclaration et utilisation des entrées	47
2.1 DECLARATION	47
2.1.1 Via un éditeur de texte	47
2.1.2 Via ResourceBundleEditor dans eclipse	47
2.2 UTILISATION	48
2.2.1 Du côté de la vue	48
2.2.2 Du côté du code Java	48
3 SURCHARGE DES ENTREES	48
Exercice N°15 : Surcharge d'un bundle	48
4 DEFINITION DES LANGAGES	49
Exercice N°16 : Ajout d'un langage	49
5 LES MESSAGES D'ERREUR PAR DEFAUT DE JSF	49
FORMULAIRES ET VALIDATION	50
1 FORMULAIRES ET BINDING	50
Exercice N°17 : Création d'un formulaire de saisie simple	50
2 LES CONVERTISSEURS	51
Exercice N°18 : Utilisation d'un convertisseur prédéfini	51
Exercice N°19 : Création d'un convertisseur	52
3 LES VALIDATEURS	53
Exercice N°20 : Validation des champs grâce à un validateur	53
Exercice N°21 : Validation des champs grâce à JSR 303	54
Exercice N°22 : Amélioration du formulaire : édition et suppression	54
Exercice N°23 : Ajout de fonctions Ajax pour l'ergonomie	55
GESTION DES EXCEPTIONS	56
Exercice N°24 : L'affichage des exceptions	56
ENVOI D'E-MAIL	59
Exercice N°25 : Ajout des fonctionnalités d'e-mail	59
AUTHENTIFICATION	61
Exercice N°26 : Mettre en place une authentification CAS	61
Exercice N°27 : Création de boutons de connexion et déconnexion	63
ACCES A UN ANNUAIRE LDAP	65

1 PARAMETRAGE DU LDAP	65
2 RECHERCHE ET UTILISATION DE L'ANNUAIRE	66
Exercice N°28 : Recherche des informations d'une personne dans le LDAP	66
Exercice N°29 : Recherche d'une ou plusieurs personnes dans l'annuaire	67
GESTION DES URL	69
Exercice N°30 : Création d'un lien direct	69
WEBSERVICES	71
1 LES WEBSERVICES CXF	71
Exercice N°31 : Exposer un webservice CXF	71
2 LES SERVICES REST	73
Exercice N°32 : Exposer un service <i>REST</i>	73
DEPLOIEMENT EN PORTLET	77
1 CREATION D'UN NOUVEAU PROJET POUR LA PORTLET	77
Exercice N°33 : Déploiement et installation d'une portlet	77
2 PREMIER LANCEMENT	79
2.1 CREATION DE L'ARCHIVE WAR	79
2.2 DEPLOIEMENT DE LA PORTLET DANS LE PORTAIL	79
2.3 PUBLICATION DE LA PORTLET	80
2.4 TEST DE LA PORTLET	82
2.4.1 Ajout de la portlet à l'environnement	82
2.4.2 Récupération d'informations	83
3 DEVELOPPEMENT D'UN PORTLET A L'AIDE DE SPRING-MVC	85
Exercice N°34 : Interrogation des webservices	85
3.1 UTILISATION DU WEBSERVICE	85
3.1.1 Démarrage des webservices	85
3.1.2 Génération des objets métiers à partir du WSDL	86
3.1.3 Interrogation du webService	87
3.1.4 Affichage des resultats	89
3.1.5 Tests et amélioration	89
3.2 UTILISATION DU SERVICE <i>REST</i>	90
Exercice N°35 : Interrogation du service <i>REST</i>	90
ANNEXES	92
LEGENDE	94
1 TITRE NIVEAU 1	94
1.1 TITRE NIVEAU 2	94
1.1.1 Titre niveau 3	94
Exercice N°36 : Titre	94
INDEX	95