

Formation

Esup-Commons V2

Manipulations et exercices

Date et lieu de la formation :

Version du document : 0.0.1 (05/08/11 14:28)

Intervenant : à compléter

1. Introduction

Le But de cette formation est de parvenir à faire

Pour cela nous allons réaliser une application simplifiée de gestion de tâches de type todolist

Fonctionnement final attendu :

Schéma de la base de données finale :

1. Sommaire

Introduction 2

Sommaire 3

Index des exercices 5

Prise en main de l’environnement 6

1 La machine virtuelle 6

1.1 JDK 6

1.2 Maven 6

1.3 MySQL 6

1.4 LDAP 6

1.5 CAS 6

1.6 Porttail 6

2 L’environnement de développement Eclipse 6

3 Découverte avec esup-example 6

3.1 Checkout depuis SVN 6

3.2 Organisation des fichiers 6

3.3 Fonctionnement en modules Maven 6

3.4 Démarrage 6

Création d’un projet 7

1 Création d'un projet maven à partir de blank 7

2 Un coup de ménage… 8

2.1 Erreur sur web-jsf-mixed 8

2.2 Les fichiers auth.xml et domain.xml embarqués dans les jars 8

2.3 Erreur dans l’appel d’un template 10

2.4 Ménage dans les librairies JSF 10

Beans Spring 12

1 Le fichier de configuration principal 12

2 L’injection 12

3 Accès aux paramètres de configuration 15

Gestion des Logs et tests unitaires 16

1 Gestion des logs 16

1.1 Utilisation dans le code Java 16

1.2 Activation du mécanisme de log 16

2 Les test unitaires 17

Accès aux données 21

1 L’objet métier 21

2 La couche DAO 21

2.1 Dans le module vue : 22

2.2 Dans le module dao : 23

3 La couche Services 23

4 Premiers tests d’écriture et lecture en base 25

4.1 Création d’un contrôleur 25

Les vues 29

1 Généralités 29

2 JSF 29

2.1 Les différentes librairies 29

2.2 Facelet 29

Formulaires et Validation 31

Gestion des exceptions 32

Internationalisation 33

1 Configuration 33

2 Déclaration des entrées 33

2.1 Via un éditeur de texte 33

2.2 Via ResourceBundleEditor dans eclipse 33

3 Définition des langages 34

4 Les messages d’erreur par défaut de JSF 34

Authentification 36

Accès à un annuaire LDAP 37

Gestion des URL 38

Envoi d’e-mail 39

Téléchargement des fichiers 40

Webservices 41

Déploiement en Portlet 42

Distribuer une application 43

Légende Chapitre 44

1 Titre niveau 1 44

1.1 Titre niveau 2 44

1.1.1 Titre niveau 3 44

1. Index des exercices

Exercice N°1 : Récupérer un projet depuis SVN 6

Exercice N°2 : Lancement d’une application Maven 6

Exercice N°3 : Création d'un projet maven à partir de blank 7

Exercice N°4 : Instanciation d’un bean simple 13

Exercice N°5 : Personnaliser les configurations grâce à l’injection 15

Exercice N°6 : Mettre en place un test unitaire simple 17

Exercice N°7 : Réaliser un test unitaire avancé 18

Exercice N°8 : Créer un objet métier simple. 21

Exercice N°9 : Créer une relation entre objets métiers 27

Exercice N°10 : Tester la couche domain dans un test unitaire 27

Exercice N°11 : Ajouter un menu via un template facelet 29

Exercice N°12 : Créer une règle de navigation 30

Exercice N°13 : Créer un contrôleur 30

Exercice N°14 : Afficher un message sur une page JSF 30

Exercice N°15 : Lister les tâches en base avec un tri par date avec comparable 30

Exercice N°16 : Créer un formulaire de saisie d’une nouvelle tâche 31

Exercice N°17 : Ajouter le lien avec un User (Ajax) 31

Exercice N°18 : Ajouter la validation des champs grâce à un validator 31

Exercice N°19 : Ajouter la validation des champs grâce à JSR 303 31

Exercice N°20 : Changer la vue des exceptions 32

Exercice N°21 : Réinitialiser un contrôleur après une exception 32

Exercice N°22 : Surcharger un bundle 33

Exercice N°23 : Ajouter un langage 34

Exercice N°24 : Mettre en place une authentification CAS 36

Exercice N°25 : Titre 44

1. Prise en main de l’environnement

# La machine virtuelle

Installation et démarrage

## JDK

## Maven

## MySQL

## LDAP

## CAS

## Porttail

# L’environnement de développement Eclipse

Plugnis etc.

# Découverte avec esup-example

## Checkout depuis SVN

1. Récupérer un projet depuis SVN

## Organisation des fichiers

## Fonctionnement en modules Maven

Schéma de l’arbre

## Démarrage

1. Lancement d’une application Maven
2. Création d’un projet
3. Création d'un projet maven à partir de blank

# Création d'un projet maven à partir de blank

* File > new > project > Project… > Maven > Maven project

Suivre la documentation :

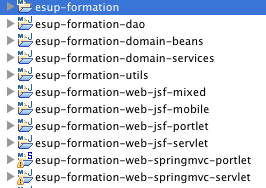
* http://www.esup-portail.org/pages/viewpage.action?pageId=100663444

La structure du projet va être construite à partir d’un archétype Maven dont voici les paramètres :

* archetypeRepository=https://mvn.esup-portail.org/content/repositories/releases
* archetypeGroupId=org.esupportail.blank
* archetypeArtifactId=esup-blank-archetype
* archetypeVersion=0.2.5
* groupId=org.esupportail.formation
* artifactId=esup-formation
* package=org.esupportail.formation
* version=0.0.1-SNAPSHOT

[Impression des ecrans]

On constate alors qu’éclipse 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.

# Un coup de ménage…

## Erreur sur web-jsf-mixed

Dans web-jsf-mixed la classe intitulée ServletOrPortletPhaseListener changer le nom de package en "package org.esupportail.formation.jsf;"

Voir mon mail "annuaire2 qui traine dans blank"

/\*\*

\* ESUP-Portail Directory Application - Copyright (c) 2011 ESUP-Portail consortium.

\*/

package org.esupportail.annuaire2.web.jsf;

import javax.faces.context.ExternalContext;

/!\ A corriger dans blank

## Les fichiers auth.xml et domain.xml embarqués dans les jars

Déplacer les fichiers auth.xml et domain.xml des modules de vue web vers le module domain-services.

Dans un module de vue (ex : web-jsf-servlet) copier/coller/renommer respectivement src/main/resources/properties/auth/auth.xml et src/main/resources/properties/domain/domain.xml vers src/main/resources/META-INF (à créer) domain-services-auth.xml et domain-services-domain.xml

Dans chacun des autres modules de vue web dans src/main/resources/properties ôter auth.xml et domain.xml

Puis dans applicationContext.xml de chaque vue web ôter…

<import resource="auth/auth.xml" />

<import resource="domain/domain.xml" />

… et mettre à la place

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

<import resource="classpath\*:META-INF/domain-services-domain.xml" />

/!\ Voir si on fait une livraison qui corrige ceci avant la formation

Sur le projet racine :

mvn install

* Clique 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]

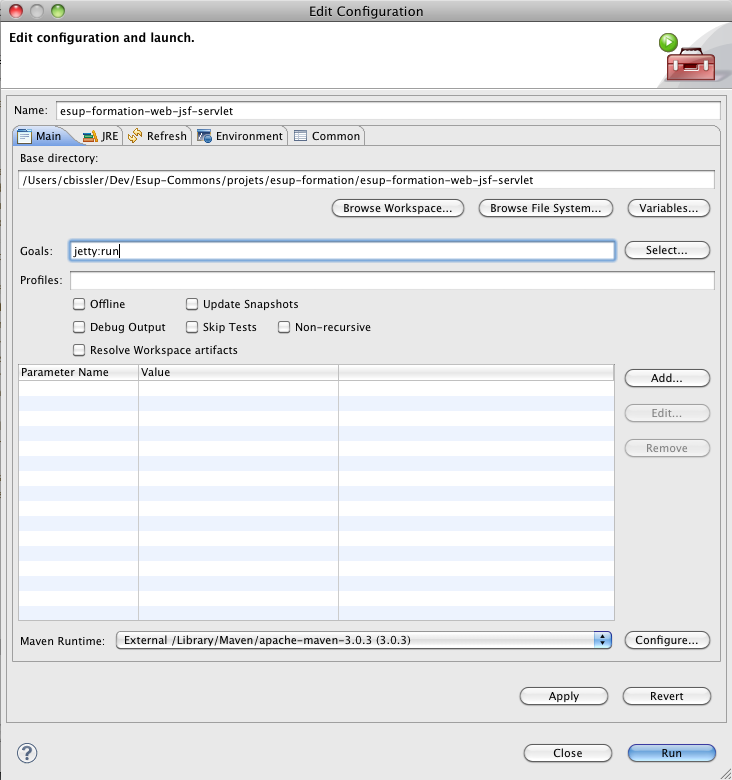
[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESS

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

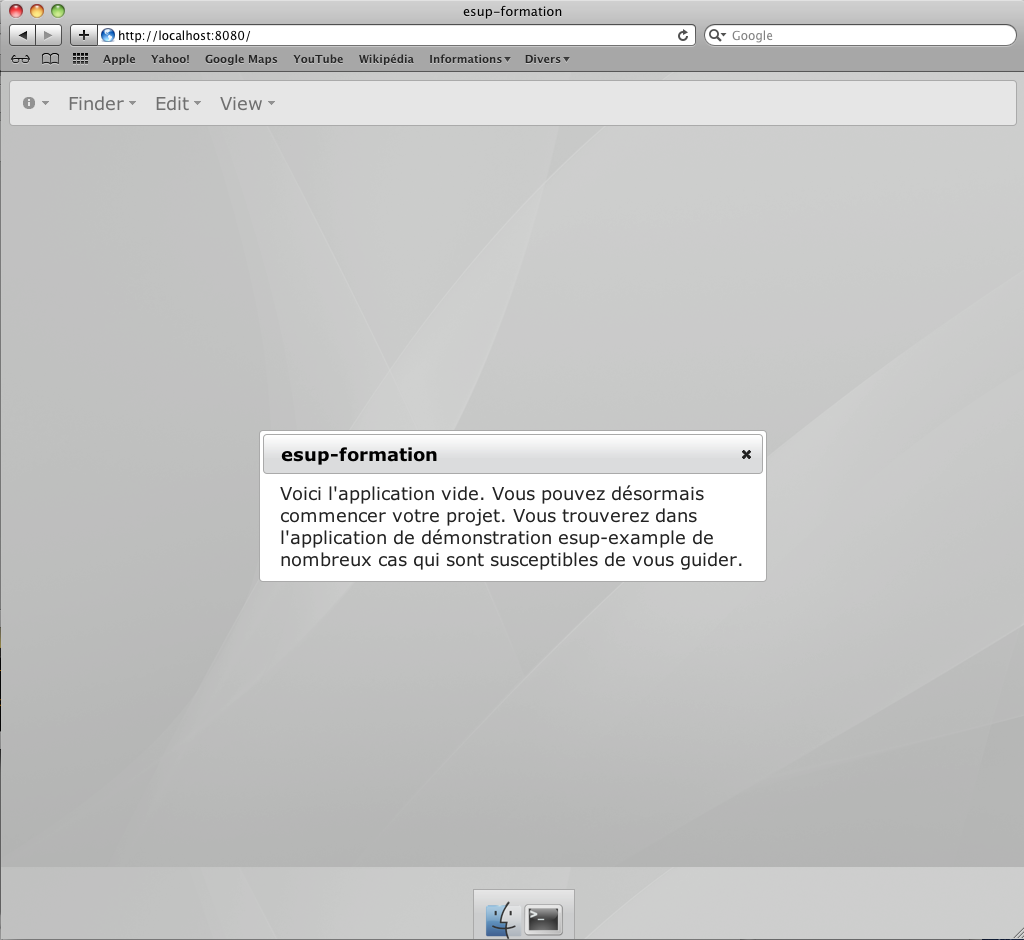


Vérifier que cela fonctionne

[INFO] Started Jetty Server

Et lancer dans un navigateur

* http://localhost:8080/



## Erreur dans l’appel d’un template

Corriger l'appel du template dans exception/exception.xhtml

Remplacer …

template="/stylesheets/simpleTemplate.xhtml"

… par …

template="/stylesheets/template.xhtml">

/!\ à corriger dans blank

## Ménage dans les librairies JSF

Stopper le serveur.

Dans le module web-jsf-servlet le fichier /src/main/webapps/stylesheets/welcome.xhtml ôter tout ce qui est primefaces, tomahawk (corps + déclaration) et mettre du JSF standard.

On ôte donc tomahawk, primefaces, esupportail :

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

On laisse les librairies 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}"/>

/!\ arriver à enlever les warnings sur les balises

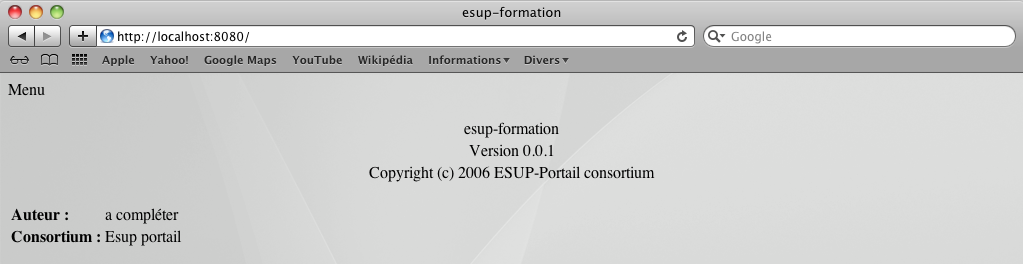
Sur le module web-jsf-servlet

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.



L'auteur qui apparait est « Cédric Leproust » à changer dans web-jsf-servlet le fichier src/main/resources/properties/i18n/bundles

Ouvrir Messages\_xx.properties avec RBE et corriger ABOUT > AUTHOR > NAME

/!\ à corriger dans blank

Est-ce qu’on supprime les modules qui ne nous serviront pas ?

Problème avec le module web-srpingmvc-portlet… impossible de remettre sur un SVN… erreur de lock ???

1. Beans Spring

# 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 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 utilisent.**

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

**<import resource="classpath\*:META-INF/domain-services-domain.xml" />**

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

# 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/properties/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 ficher /properties/config.properties les paramètres qui ne conviennent pas dans /properties/defaults.properties.

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

1. Instanciation d’un bean simple

L’objectif de cet exercice est de créer un bean simple et 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" />

Comment récupère-t-on ce bean ?

Récupérer ce bean dans la méthode afterPropertiesSetInternal() de sessionController du module web-jsf-servlet par exemple grâce à la méthode BeanUtils.getBean().

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

import org.esupportail.commons.utils.BeanUtils;

[...]

public class SessionController extends AbstractDomainAwareBean {

[...]

@Override

public void afterPropertiesSetInternal() {

[...]

Task t = (Task) BeanUtils.getBean("task1");

System.out.println("Test injection => "+t.getTitle()+" "+t.getDescription());

}

[...]

Dans un cas réel, il est recommandé pour des questions de lisibilité d’utiliser le moins possible la méthode BeanUtils.getBean() pour récupérer un bean. On préférera injecter directement ce dernier via la configuration spring dans le bean l’utilisant. Ici par exemple on aurait ajouté task comme attribut de SessionController et on aurait ajouter la référence au bean d’identifiant « task1 » dans la déclaration de sessionController de controllers.xml.

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

Nous avons fait ceci à titre d’exemple. Supprimer la déclaration du bean dans beans.xml et sa récupération dans SessionController. Nous ne garderons que la classe Task pour la suite.

Pour l’instant on fait afficher dans sessionController ? Faire afficher ceci dans un junit comment faire ? le paragraphe ne vient qu’apres

1. Personnaliser les 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/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>

# Accès aux paramètres de configuration

Faire afficher la méthode d’authentification dans la méthode afterPropertiesSetInternal() de sessionController du module web-jsf-servlet par exemple

Comment faire ?

GetAuthenticationService est protected Essai de test unitaire plus loin

1. Gestion des Logs et tests unitaires

# 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.).

## 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 loguer 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() + "'");

}

## Activation du mécanisme de log

Dans le fichier src/main/resources/log4j.properties du module web-jsf-servlet

Remplacer…

log4j.logger.org.esupportail.example=DEBUG

par…

log4j.logger.org.esupportail.formation=DEBUG

/ !\ A corriger dans blank

Dans config.properties paramétrer la gestion des logs à votre convenance. Ajouter par exemple :

log.level=DEBUG

log.output=stdout

# Les test unitaires

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

1. Mettre en place un 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>

</dependency>

* clique droit sur le projet > Maven > Update dépendencies

Si on souhaite généraliser les tests unitaires à l’ensemble des modules il serait peut-être plus judicieux de positionner cette dépendance dans le projet racine.

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

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

Créer une classe de test (tesCase) 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

* Clique 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.

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

* Dans eclipse

Fenêtre Run > Run configurations

* En ligne de commande

-Dmaven.test.skip=true

ou

-DskipTests=true

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

@Test

public void testSetTitle() {

fail("Not yet implemented");

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

1. Réaliser un 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.Test;

public class AuthenticatorTest {

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 applicationContex.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/domain-services-auth.xml" />

<import resource="classpath\*:META-INF/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 nécéssaires.

Il faudra alors ajouter un propertyConfigurer

<bean id="propertyConfigurer"

class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">

<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.junit.Before;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

[…]

@Before

public void setUp() throws Exception {

String[] springFiles = { "classpath\*:META-INF/testApplicationContext.xml" };

ApplicationContext applicationContext = new ClassPathXmlApplicationContext(springFiles);

authenticator = (Authenticator) applicationContext.getBean("authenticator");

}

Impossible de faire afficher le nom de la classe d’authentification choisie car elle est protected

1. Accès aux données
2. Créer 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 Compléter

# L’objet métier

Créer un objet métier qui sera mappé en base.

Dans le pom.xml du module domain-beans y mettre la couche JPA

<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 précédement avec :

* id (long généré en base) : Identifiant
* title (chaine obligatoire) : Titre
* date (type date) : Date de limite d’éxécution de la tâche
* description (type chaine) : Description de la tâche
* publicTask (type booleen) : 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-exo1

# 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.xml et 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 s'il y en a plusieurs
* 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.xml que les paramètres sont passés par l’intermédiaire de variables qui seront positionnées dans le fichier de properties.

## Dans le module vue :

Dans src/main/resources/properties/dao copier les fichiers dao.xml et persitence.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/domain-services-domain.xml du module domain-services de la façon suivante :

<import resource="classpath\*:/properties/dao/dao.xml" />

## Dans le module dao :

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

Dans dao-dao.xml adapter le chemin vers dao-persitence.xml en remplaçant :

<property name="persistenceXmlLocation" value="classpath:/properties/dao/persistence.xml" />

par

<property name="persistenceXmlLocation" value="classpath\*:META-INF/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/domain-services-domain.xml du module domain-services de la façon suivante

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

Que préconise-t-on ? les fichiers dao dans le module vue ? ou dans le module dao ?

Selon la méthodes choisie

* Récupérer les fichiers dao.xml-exo1 et persistence.xml-exo1 ou dao-dao.xml-exo1 et dao-persitence.xml-exo1

Enfin sur le module dao faire

mvn install

# 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 que l'objet DaoService est bien injecté

Assert.notNull(this.daoService,

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

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

<bean id="domainService" class="org.esupportail.formation.domain.DomainServiceImpl">

<property name="daoService" ref="daoService" />

</bean>

Ainsi que a compléter

<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>

Dans src/main/resources/properties/config.properties configurer la connexion à la base de données (cf. variables utilisées dans 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.datasource=jdbc/formationDB

# 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 selon la méthode choisie

<import resource="classpath\*:/properties/dao/dao.xml" />

Ou

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

# Premiers tests d’écriture et lecture en base

## Création d’un contrôleur

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

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-jsf-servlet

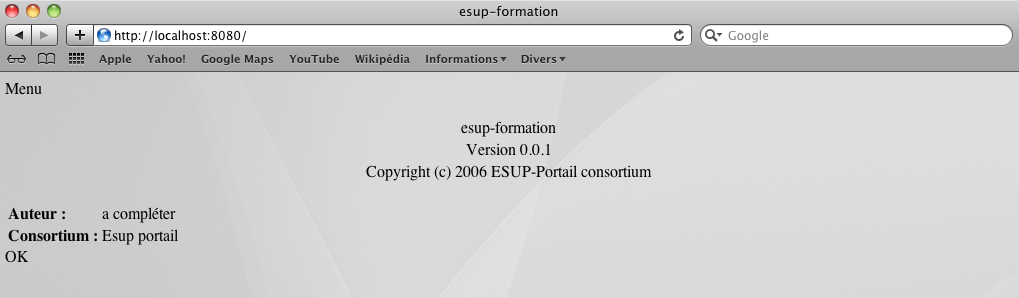
mvn jetty:run

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 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.

On stoppe jetty toujours pas de données.

On redémarre et là on a

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

/!\ expliquer pourquoi on a les inserts au redémarrage et pas au stop

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

1. Créer une relation entre objets métiers

On trouve par défaut dans le module domain-beans l’objet 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 :

* Pour cela renommer la propriété id en login

En effet, pour un objet métier il est important de savoir différencier : l’identifiant en base souvent clé primaire et propre à la base (ici id) et l’identifiant métier unique, qui pourra être public et commun à plusieurs base du système d’information (ici login).

/!\ cela nécessite de changer tous les appels setId en setLogin et notamment dans le module domain-services org.esupportail.formation.services.auth.AuthenticatorImpl méthode getUser pour les types CAS et Shibboleth

Pourquoi a-t-on mis id=uid dans l’objet user par défaut ?

* 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 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

1. Tester la couche domain dans un test unitaire

Pour être propre blablabla (voir Les test unitaires )

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.Before;

import org.junit.Test;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class DomainServiceTest {

DomainService domainService;

@Before

public void setUp() throws Exception {

String[] springFiles = { "classpath\*:META-INF/testApplicationContext.xml" };

ApplicationContext applicationContext = new ClassPathXmlApplicationContext(springFiles);

domainService = (DomainService) applicationContext.getBean("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);

}

}

Pourquoi reste en mémoire et pas de base comme dans module vue ?

* Documentation JPA :
* <http://tahe.developpez.com/java/jpa/>
* [http://www.dil.univ-mrs.fr/~massat/ens/jee/tp-JPA.html](http://www.dil.univ-mrs.fr/%7Emassat/ens/jee/tp-JPA.html)
* <http://docs.jboss.org/hibernate/annotations/3.5/reference/en/html/entity.html>

1. Les vues

# Généralités

Spring MVC vs JSF

Bien insister sur le fait que des formations supplémentaires spécifiques à chaque technologie de vues seront proposées.

# JSF

## Les différentes librairies

## Facelet

1. Ajouter 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 appelera 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

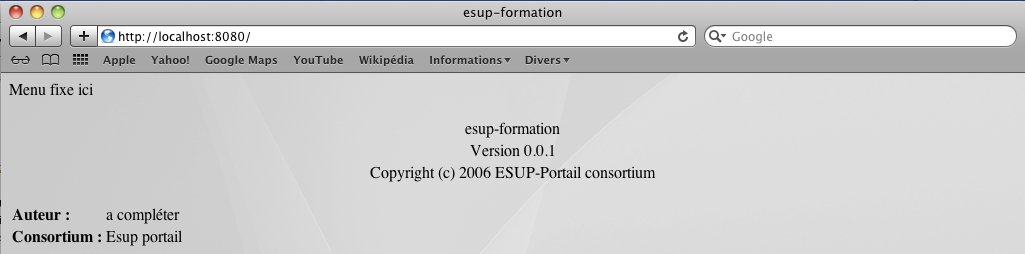
</ui:define>

</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



1. Créer une règle de navigation
2. Créer un contrôleur
3. Afficher un message sur une page JSF
4. Parcourir un tableau

Lister les tâches en base avec un tri par date avec comparable

1. Formulaires et Validation
2. Créer un formulaire de saisie d’une nouvelle tâche
3. Ajouter le lien avec un User (Ajax)

Ajouter le lien avec un User existant (Ajax)

1. Ajouter la validation des champs grâce à un validator

Création d’un validator

1. Ajouter la validation des champs grâce à JSR 303

et la validation

<dependency>

<groupId>javax.validation</groupId>

<artifactId>validation-api</artifactId>

<version>1.0.0.GA</version>

<scope>provided</scope>

</dependency>

1. Gestion des exceptions
2. Changer la vue des exceptions

Copier exception.jsp en exception-prod.jsp et simplifier la page pour ne garder que les informations qui vous intéressent. Configurer l'application pour utiliser cette nouvelle vue en cas d'exception.

A reprendre

1. Création de la page exception-prod.jsp dans le répertoire webapp/stylesheets/

2. Modification du fichier exceptionHandling.xml :

Ajout d'une entrée dans la property exceptionViews.

<entry key="java.lang.Exception" value="/stylesheets/exception-prod.jsp" />

3. Créer puis provoquer une exception afin de tester la redirection vers exception-prod.jsp.

1. Réinitialiser un contrôleur après une exception

Faire en sorte que sur une exception, l'attribut value de test2Controller soit vidé.

Implémenter la méthode reset dans le controller Test2Controller :

/\*\*

\* @see org.esupportail.formation.web.controllers.AbstractDomainAwareBean#reset()

\*/

@Override

public void reset() {

super.reset();

value = null;

}

1. Internationalisation

# 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="[...].commons.services.i18n.BundlesCachingI18nServiceImpl"

>

<property name="bundleBasenames">

<list>

<value>properties/i18n/bundles/Commons</value>

<value>properties/i18n/bundles/Messages</value>

<value>properties/i18n/bundles/Custom</value>

</list>

</property>

</bean>

# Déclaration des entrées

1. Surcharger un bundle

Surcharger dans Custom\_fr.properties une des entrées de Messages\_fr.properties et tester.

## Via un éditeur de texte

Par exemple nous allons modifier le nom de l’auteur présent dans le fichier bundles/Messages\_fr.properties.

ABOUT.AUTHOR.NAME = a compl\u00E9ter

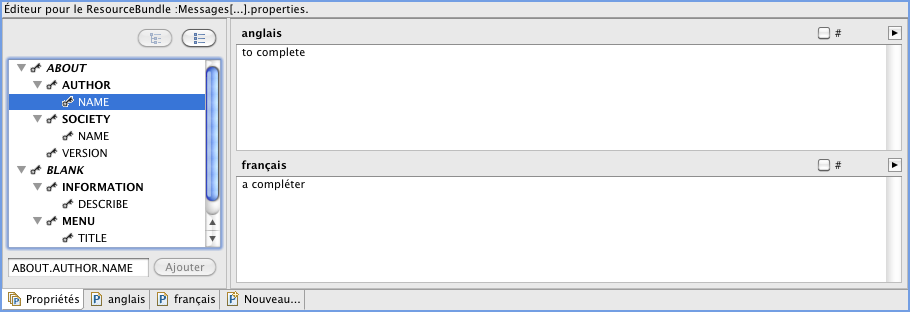
Dans bundles/Custom\_fr.properties ajouter la clef ABOUT.AUTHOR.NAME et modifier son contenu :

ABOUT.AUTHOR.NAME = C\u00E9line Didier

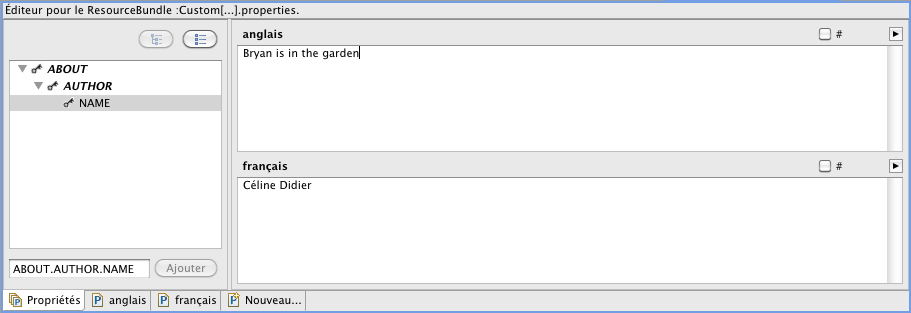
## Via ResourceBundleEditor dans eclipse

Editer bundles/Messages\_fr.properties avec resourceBundleEditor

* Clique droit sur le fichier > open with… > éditeur de reourceBundle



Editer bundles/Custom\_fr.properties avec resourceBundleEditor



# Définition des langages

1. Ajouter un langage

Ajouter le langage japonais et traduire tout esup-commons.

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.

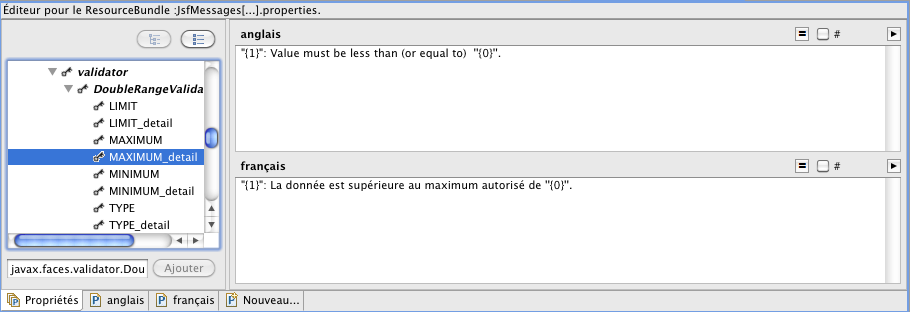
# Les messages d’erreur par défaut de JSF

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 JFaces sont déclaré dans properties/i18n/bundles/JsfMessages.

On notera dans l’exemple ci-après, l’usage du passage de variable.



1. Authentification
2. Mettre en place une authentification CAS
3. Accès à un annuaire LDAP
4. Gestion des URL
5. Envoi d’e-mail
6. Téléchargement des fichiers
7. Webservices
8. Déploiement en Portlet
9. Distribuer une application
10. Légende Chapitre

# Titre niveau 1

## Titre niveau 2

### Titre niveau 3

1. 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

1. Index

Introduction 2

Sommaire 3

Index des exercices 5

Prise en main de l’environnement 6

1 La machine virtuelle 6

1.1 JDK 6

1.2 Maven 6

1.3 MySQL 6

1.4 LDAP 6

1.5 CAS 6

1.6 Porttail 6

2 L’environnement de développement Eclipse 6

3 Découverte avec esup-example 6

3.1 Checkout depuis SVN 6

Exercice N°1 : Récupérer un projet depuis SVN 6

3.2 Organisation des fichiers 6

3.3 Fonctionnement en modules Maven 6

3.4 Démarrage 6

Exercice N°2 : Lancement d’une application Maven 6

Création d’un projet 7

Exercice N°3 : Création d'un projet maven à partir de blank 7

1 Création d'un projet maven à partir de blank 7

2 Un coup de ménage… 8

2.1 Erreur sur web-jsf-mixed 8

2.2 Les fichiers auth.xml et domain.xml embarqués dans les jars 8

2.3 Erreur dans l’appel d’un template 10

2.4 Ménage dans les librairies JSF 10

Beans Spring 12

1 Le fichier de configuration principal 12

2 L’injection 12

Exercice N°4 : Instanciation d’un bean simple 13

Exercice N°5 : Personnaliser les configurations grâce à l’injection 15

3 Accès aux paramètres de configuration 15

Gestion des Logs et tests unitaires 16

1 Gestion des logs 16

1.1 Utilisation dans le code Java 16

1.2 Activation du mécanisme de log 16

2 Les test unitaires 17

Exercice N°6 : Mettre en place un test unitaire simple 17

Exercice N°7 : Réaliser un test unitaire avancé 18

Accès aux données 21

Exercice N°8 : Créer un objet métier simple. 21

1 L’objet métier 21

2 La couche DAO 21

2.1 Dans le module vue : 22

2.2 Dans le module dao : 23

3 La couche Services 23

4 Premiers tests d’écriture et lecture en base 25

4.1 Création d’un contrôleur 25

Exercice N°9 : Créer une relation entre objets métiers 27

Exercice N°10 : Tester la couche domain dans un test unitaire 27

Les vues 29

1 Généralités 29

2 JSF 29

2.1 Les différentes librairies 29

2.2 Facelet 29

Exercice N°11 : Ajouter un menu via un template facelet 29

Exercice N°12 : Créer une règle de navigation 30

Exercice N°13 : Créer un contrôleur 30

Exercice N°14 : Afficher un message sur une page JSF 30

Exercice N°15 : Parcourir un tableau 30

Formulaires et Validation 31

Exercice N°16 : Créer un formulaire de saisie d’une nouvelle tâche 31

Exercice N°17 : Ajouter le lien avec un User (Ajax) 31

Exercice N°18 : Ajouter la validation des champs grâce à un validator 31

Exercice N°19 : Ajouter la validation des champs grâce à JSR 303 31

Gestion des exceptions 32

Exercice N°20 : Changer la vue des exceptions 32

Exercice N°21 : Réinitialiser un contrôleur après une exception 32

Internationalisation 33

1 Configuration 33

2 Déclaration des entrées 33

Exercice N°22 : Surcharger un bundle 33

2.1 Via un éditeur de texte 33

2.2 Via ResourceBundleEditor dans eclipse 33

3 Définition des langages 34

Exercice N°23 : Ajouter un langage 34

4 Les messages d’erreur par défaut de JSF 34

Authentification 36

Exercice N°24 : Mettre en place une authentification CAS 36

Accès à un annuaire LDAP 37

Gestion des URL 38

Envoi d’e-mail 39

Téléchargement des fichiers 40

Webservices 41

Déploiement en Portlet 42

Distribuer une application 43

Légende Chapitre 44

1 Titre niveau 1 44

1.1 Titre niveau 2 44

1.1.1 Titre niveau 3 44

Exercice N°25 : Titre 44

Index 45