|  |
| --- |
| UNiversite de montpellier ii |
| T.E.R. : Outil de C.M.S. sur la plateforme Seam |
|  |
| Réalisé par |
|  |
|  |

**AHAMADA Soulé, DUVAL Simon, KEIRA Mamadou et TOUIL Mehdi**

**Année 2008-2009**

|  |
| --- |
| [Tapez le résumé du document ici. Il s'agit généralement d'une courte synthèse du document. Tapez le résumé du document ici. Il s'agit généralement d'une courte synthèse du document.] |

Tuteur s :

Mr DONY Christophe

Mr CHATEL Thierry

# Introduction

# Méthodologie de travail

## Gestion des ressources

## Gestion de projet

### Planning prévisionnel

### Outil de collaboration et de versionning : SVN

# Technologies utilisées et concepts

## Framework Seam

## Le modèle de composant Seam

A l'origine le modèle de composants de Seam a été conçu pour permettre l'intégration directe entre JSF et les EJB3.

Quand on développe une application web à partir de JSF, on implémente généralement deux types de composants :

* une page JSF qui va constituer la vue et contenir le code HTML,
* une classe Java appelée backing bean JSF qui va constituer le modèle et fournir les méthodes pour implémenter les actions utilisateurs (soumission formulaire ...).

Les méthodes du backing bean JSF liées aux actions utilisateurs accèdent en général à un composant métier. Si celui-ci est déployé sous la forme d'un EJB 3 de type Session Bean, il est alors nécessaire d'implémenter une couche d'adaptation entre ces deux types de composants.



Figure 1: Intégration JSF/EJB 3 sans le framework Seam

Avec Seam, il n'y a plus de distinction entre le composant backing bean et le composant métier ; ils forment un seul et unique composant qu'on appelle composant Seam. Une page JSF peut donc invoquer directement un composant métier déployé sous la forme d'un Session Bean.



Figure 2: Intégration JSF/EJB3 avec le framework Seam

Un composant Seam est une classe java du type POJO ou EJB 3.0 qui contient les annotations @name pour identifier le composant et @Scope pour le contexte applicatif.

Exemple : déclaration d'un composant Seam :

@Name("sessionUtilisateur")

@Scope(ScopeType.*SESSION*)

**public** **class** SessionUtilisateur {

…

}

Chaque composant Seam est déclaré sous un nom logique (ici sessionUtilisateur) et l'appel de ce composant depuis une page JSF s'effectue de la façon suivante :

<h:inputText id="login" value="#{sessionUtilisateur.utilisateur.login}" required="true"/>

Avec ce principe, les développements JSF sont simplifiés, puisque il n'y a plus besoin de déclarer les backing beans dans le fichier de configuration faces-config.xml.

La spécification Web Beans JSR 299 (http://jcp.org/en/jsr/detail?id=299) est basée sur les concepts de ce modèle de composant et sera très certainement intégrée dans la **spécification JEE6.**

## Contextes Seam

On accède à un composant Seam depuis son contexte. Chaque type de composant est déclaré par défaut dans un contexte mais il est facile d'étendre le contexte d'un composant. Pour cela, Seam met à disposition différents contextes :

* **stateless** qui est le contexte par défaut des composants EJB Stateless, sert pour les composants sans état.
* **request** (idem J2EE correspond à la requête HTTP) ou **event** qui est le contexte par défaut des composants du type POJO ou JavaBean,
* **page** (idem J2EE), le composant est accessible sur la durée de vie d'une page JSF,
* **conversation** qui correspond à un enchainement de pages.
* **session** (idem J2EE), le composant est accessible depuis la session HTTP,
* **process**, contexte utilisé pour la gestion des processus (par exemple JBPM)
* **application** (idem J2EE) est commun à toute l’application web et dure jusqu’à l’arret du serveur d’application.

## Inversion de contrôle et injection de dépendances :

Pour que les composants puissent interagir entre eux, Seam propose un mécanisme d'injection des dépendances bijectif :

* un composant peut être injecté automatiquement dans un autre composant (équivalent au mécanisme d'injection des dépendances du framework Spring),
* un composant peut également créer et projeter un autre composant dans un contexte

(outjection) qui deviendra ainsi accessible par tous les autres composants.

Il est également possible de forcer la création ou non des composants au moment de l'injection.

Avec Seam, nous avons donc un mécanisme d'injection des dépendances complètement dynamique, ne nécessitant pas de configuration XML.

L’idée de l’inversion de contrôle en anglais Inversion Of Control (IoC) est de ne pas gérer manuellement dans chaque composant la récupération des autres composants dont il a besoin, mais de le laisser faire au framework. Les composants vont ainsi pouvoir être développés indépendamment les uns des autres, en laissant le soin au framework de faire le lien entre eux à l’exécution.

Chaque composant va signaler ses dépendances, et le framework va les lui injecter.

On indique une dépendance devant être injectée en ajoutant une annotation @In à une variable d’instance d’un composant.

Le framework va au-delà de l’injection de dépendance, puisqu’il va aussi gérer l’opération inverse, via l’annotation @Out, pour (re-)publier après l’exécution d’une méthode une dépendance si sa référence a été modifiée par la méthode.

## Gestion des conversations

Actuellement, pour gérer l'état conversationnel dans une application web, la seule solution est de stocker les informations au niveau de la session HTTP, ce qui entraîne souvent des problèmes de réinitialisation ou de purge des informations.

Avec Seam, on utilise ce qu'on appelle les conversations longues soit de manière implicite à partir de composants du type EJB3 stateful ou bien avec des JavaBeans en les déclarants dans le contexte de conversation.

Une conversation Seam va donc englober plusieurs requêtes et un utilisateur pourra utiliser plusieurs conversations sachant que chaque conversation est complètement indépendante.

Le démarrage d'une conversation longue s'effectue de manière explicite en positionnant par exemple une annotation Java @Begin au niveau d'une méthode métier.

De la même façon une conversation est terminée à partir de l'annotation @End sur une méthode ou bien dès qu'elle tombe en timeout.

## Intégration avec JPA

JPA (Java Persistence API) est l'API de persistance fournie par JEE5. Cette API se base en grande partie sur les concepts apportés par Hibernate, le framework incontournable de mapping relationnel objet (ORM).

JPA et Hibernate ont un mécanisme de chargement des données à la demande (Lazy Loading), pour éviter par exemple qu'un seul select sur une table parent récupère toutes les données des tables filles. Ce chargement des données à la demande n'est possible qu'à partir d'un contexte de persistance (ou session hibernate) et généralement celui-ci n'est plus accessible depuis une page JSF de l'application web, ce qui peut parfois poser des problèmes pour l'affichage des données.

Pour contourner ce problème, il y a deux solutions :

* implémenter une couche d'objets (DAO pour Data Acces Object) permettant de charger les objets de persistance avec toutes les données nécessaires pour l'affichage,
* utiliser le contournement OpenSessionInView qui consiste à maintenir le contexte de persistance ouvert pour chaque requête HTTP.

Ces deux solutions, si elles fonctionnent, ont leurs limites. Dans le cas des DAO, on duplique le code et, avec la deuxième solution, on maintient le contexte de persistance ouvert pour toutes les requêtes HTTP et ce mécanisme ne convient pas vraiment aux applications JSF.

Avec Seam, la solution est beaucoup plus simple : le contexte de persistance sera réutilisé si la page web a besoin d'afficher des données supplémentaires. On a donc plus besoin d'implémenter de **couche DAO.**

## Framework Hibernate

Hibernate, comme tous les autres outils de mapping objet/relationnel, nécessite des méta-données qui régissent la transformation des données d'une représentation vers l'autre (et vice versa). Dans Hibernate 2.x, les méta-données de mapping sont la plupart du temps déclarées dans des fichiers XML.

Maintenant, avec la spécification java EE5, les annotations sont disponibles pour arriver au même résultat avec une très grande élégance.

Eclipse, par exemple, prend en charge la complétion automatique et la coloration syntaxique des annotations du JDK 5.0. Les annotations sont compilées en bytecode et lues au moment de l'exécution (dans le cas d'Hibernate, au démarrage) en utilisant la réflexion, donc pas besoin de fichiers XML externes.

La spécification EJB3 reconnaît l'intérêt et le succès du paradigme du mapping objet/relationnel transparent. La spécification EJB3 standardise les APIs de base et les méta-données requises par n'importe quel mécanisme de persistance objet/relationnel. Hibernate EntityManager implémente les interfaces de programmation et les règles de cycle de vie telles que définies par la spécification de persistance EJB3. AvecHibernate Annotations, ce wrapper implémente une solution de persistance EJB3 complète (et autonome) au-dessus du noyau mature d'Hibernate.

On peut utiliser soit les trois ensembles, soit les annotations sans le cycle de vie et les interfaces de programmations EJB3, ou même Hibernate tout seul, selon les besoins techniques et fonctionnels du projet. On peut à tout moment recourir aux APIs natives d'Hibernate ou même, si besoin est, à celles de JDBC et au SQL.

Dans ce projet, nous avons utilisé Hibernate tout seul pour le mapping objet/relationnel car, le cycle de vie est géré par le framework Seam.

Conformément à la documentation officielle de Hibernate, nous avons créer un wraper connu sous le nom de HibernateUtil dans un contexte statique pour la gestion des transactions.

## Frameworks JSF, Facelets

# Configuration de l’environnement de développement

## Serveur d’application Tomcat 6.0

## Hibernate

# Mise en œuvre et réalisation

## Diagrammes des cas d’utilisation

### Back end

### Front end

## Gestion du contenu

### Structure de données

### Design pattern composite

## Gestion des droits d’utilisateurs

## Templates de pages web

## Templates de pages web :

Le templating consiste à factoriser la structure commune d'un ensemble de pages et à l'extraire dans une nouvelle page, appelée template ou modèle. Les autres pages utilisent alors le template comme structure et y injectent leur contenu spécifique.

Le templating offre plusieurs avantages :

* Uniformiser la structure des pages.
* Simplifier la mise à jour : une modification dans le template se propage automatiquement dans toutes les pages qui l'utilisent.
* Gain en productivité : moins de code à écrire : une page ne contient que ce qui lui est propre.

Grace à l’utilisation des facelets dans notre application, nous avons pu faire du templating, qui est non seulement utile pour éviter la recopie de code mais aussi pour la réutilisation des composants. Les deux tempates se trouve dans le répertoire /template.

Chaque page de l’application (.xhtml) est donc une composition d’une page template-admin.xhtml, qui affiche l’entête de la page, le menu de navigation, le pied de page, dont on redéfinie le contenu. Dans le cas de la partie « Backend » chaque page est une composition de template-admin.xhtml qui est une composition de header-admin.xhtml en y ajoutant le menu.xhmtl et footer.xhtml. La page admin-xhtml utilise ce template et rédéfinit le contenu afin d’acquérir l’élément courant grâce à un composant seam qui retourne la page courante à insérer.

Le squelette du template de la partie administration est le suivant :

<body>

<ui:insert name=*"header"*>

<ui:include src=*"header.xhtml"*/>

</ui:insert>

<ui:insert name=*"navigation"*>

<ui:include src=*"navigation.xhtml"*/>

</ui:insert>

<ui:insert name=*"content"* />

<ui:insert name=*"footer"*>

<ui:include src=*"footer.xhtml"*/>

</ui:insert>

</body>

Les balises importantes qu’il convient d’expliquer sont :

<ui :include> : Permet

<ui :insert> : Permet

<ui :composition> : Permet

Nous remarquons que seule la partie <ui:insert name=*"content"* /> sera redéfinit par la page cliente utilisant ce template, ce qui rend cette partie du template dynamique et réutilisable.

On commence par spécifier que l'on utilise un template avec la balise <ui:composition> qui prend comme paramètre le chemin vers le fichier contenant le template. Pour définir les différents blocs qui seront injectés dans le template, on utilise la balise <ui:define>. Cette balise prend comme paramètre le nom logique du bloc correspondant dans le template et contient dans son corps le contenu à injecter dans le template.

Ainsi la page cliente *admin.xhtml* définira la balise :

<ui:composition template=*"template/template-admin.xhtml"*>

</ui:composition>

## Gestion des exceptions

Seam nous permet d’intercepter toutes les exceptions qui se produisent en dehors du contexte applicatif d’un composant de deux manières qui peuvent être utilisées conjointement :

* Par annotation des classes d’exceptions, Exemple :

*@Redirect(viewId="/error.xhtml", message="erreur contenu")*

*public class ContenuException extends RuntimeException { ... }*

Cette exception se traduit par un message à l’utilisateur et l’annulation de la transaction courante avec une redirection vers la page *error.xhtml.*

* Par le fichier de configuration *pages.xml :* Cette méthode a l’avantage de pouvoir intercepter toute sorte d’exceptions.

Ces deux possibilités sont complémentaires car, on ne peut annoter toutes les exceptions qu’on veut, c’est pourquoi Seam autorise l’utilisation du fichier XML afin d’intercepter ces dernières.

Pour que Seam soit capable d’intercepter des exceptions, il faut tout d’abord déclaré le master servlet dans le fichier web.xml de la manière suivante :

*<filter>*

*<filter-name>Seam Filter</filter-name>*

*<filter-class>org.jboss.seam.servlet.SeamFilter</filter-class>*

*</filter>*

*<filter-mapping>*

*<filter-name>Seam Filter</filter-name>*

*<url-pattern>\*.seam</url-pattern>*

*</filter-mapping>*

Dans ce projet, nous avons choisi la deuxième méthode celle à l’aide du fichier XML car, elle permet d’intercepter toute sorte d’exception se produisant en dehors du contexte applicatif d’un composant Seam.

Par exemple pour intercepter une exception et afficher un message compréhensible à l’utilisateur quand sa session a expiré :

# Conclusion

# Glossaire

# Bibliographie

# Annexes

Back end

Screnshots

Front end

Screnshots

## Templates de pages web :

### Back end :

Grace à l’utilisation des facelets dans notre application, nous avons pu faire du templating, qui est non seulement utile pour éviter la recopie de code mais aussi pour la réutilisation des composants. Les deux tempates se trouve dans le répertoire /template.

Chaque page de l’application (.xhtml) est donc une composition d’une page template-admin.xhtml, qui affiche l’entête de la page, le menu de navigation, le pied de page, dont on redéfinie le contenu. Dans le cas de la partie « Backend » chaque page est une composition de template-admin.xhtml qui est une composition de header-admin.xhtml en y ajoutant le menu.xhmtl et footer.xhtml. La page admin-xhtml utilise ce template et rédéfinit le contenu afin d’acquérir l’élément courant grâce à un composant seam qui retourne la page courante à insérer.