

Seam, un framework nouvelle génération

JBoss Seam est un framework permettant de construire beaucoup plus simplement qu'auparavant des applications Web complexes. Seam s'appuie sur diverses technologies de Java EE 5, telles que : Java Server Faces (JSF), EJB 3, *Java Persistence API* (JPA). Seam fournit également un ensemble d'API et d'annotations pour aider et simplifier le développement d'application Java EE 5.

Gavin King, le créateur d'Hibernate, est à l'origine du projet Seam sur lequel il travaille activement. La version 2.0 de Seam est disponible depuis la fin de l'année 2007.

Au travers d'un exemple, nous allons montrer l'utilisation de JBoss Seam et comment il intègre les différents éléments de Java EE. Cet exemple est une application simple permettant de visualiser un ensemble d'étudiants, et de saisir et de supprimer des étudiants.

Techniquement, l'application se compose de :

- une IHM composée de deux pages JSF,
- une entité « étudiant » implémentée grâce à JPA,
- un gestionnaire des étudiants implémenté en EJB 3 par un EJB session stateful,
- une intégration de l'ensemble via Seam.

L'entité 'étudiant'

L'entité « étudiant » est un POJO annoté par diverses annotations de JPA. JPA est une spécification de Java EE 5 qui définit une API de mapping objet / relationnel. Dans notre exemple, nous utilisons Hibernate, qui est, à ce jour, une des implémentations de JPA.

```
@Entity
@Table(name="STUDENT")
public class Student implements Serializable {
    @Id
    @Column(name="STD_LOGIN")
    private String login;

    @Column(name="STD_NAME")
    private String name;

    @Column(name="STD_PASSWORD")
    private String password;

    @Column(name="STD_EMAIL")
    private String email;

    ...
}
```

Listing 1 : Entité étudiant

Le Listing 1 présente le code source de l'entité « étudiant ». La classe `Student` est déclarée comme étant une entité grâce à l'annotation `@Entity`. L'identifiant de l'entité est défini grâce à l'annotation `@Id`. Le mapping de l'entité dans la base est définie grâce à d'autres annotations. L'annotation `@Table` désigne la table dans laquelle l'entité est stockée. Les annotations `@Column` désignent les colonnes de cette table.

Le gestionnaire des étudiants

Le Listing 2 présente le code source de l'implémentation du gestionnaire des étudiants par un EJB *session stateful*. L'EJB se présente globalement comme un POJO (*Plain Old Java Object*) décoré par des annotations.

```
@Stateful
public class StudentManagerBean implements StudentManager {
    @PersistenceContext(type = EXTENDED)
    private EntityManager entityManager;
    private List<Student> students;

    public StudentManagerBean() {
        this.students = new ArrayList<Student>();
    }

    public List<Student> getStudents() {
        return students;
    }

    public void findAllStudents() {
        students = entityManager
            .createQuery("select s from Student s")
            .getResultList();
    }

    public String addStudent(Student student) {
        entityManager.persist(student);
        students.add(student);
        return "student-added";
    }

    public void deleteStudent(Student student) {
        entityManager.remove(student);
        students.remove(student);
    }

    @Remove
    public void end() {
    }
}
```

Listing 2 : Gestionnaire des étudiants

L'annotation `@Stateful` indique qu'il s'agit d'un EJB *session stateful*. Avec un tel EJB, l'état est conservé entre les appels de méthode. Une instance de l'EJB est créée pour chaque client de l'EJB. Le client s'adresse ensuite à cette instance dédiée, en invoquant diverses méthodes. L'appel à une méthode spéciale étiquetée par l'annotation `@Remove` permet au client de signaler qu'il a fini d'utiliser l'EJB. L'état disparaît alors avec l'instance de l'EJB.

L'annotation `@PersistenceContext` de JPA permet d'injecter un contexte de persistance JPA dans l'EJB. Le contexte de persistance permet d'accéder à des entités persistées dans une base de données (lecture, ajout, suppression, modification ...).

```

@Local
public interface StudentManager {
    List<Student> getStudents();
    void findAllStudents();
    String addStudent(Student student);
    void deleteStudent(Student student);

    void end();
}

```

Listing 3 : Interface locale du gestionnaire des étudiants

Le Listing 3 présente le code de l'interface de l'EJB. L'annotation `@Local` indique que `StudentManager` est une interface qui permet d'accéder à l'EJB en local, par opposition au mode distant (*remote*).

Composants Seam

Jusqu'alors, les composants mis en œuvre font partie de Java EE 5. Pour en faire des composants Seam, il convient de les décorer avec des annotations spécifiques.

```

@Name("student")
...
public class Student implements Serializable {
    ...
}

```

Listing 4 : Déclaration du composant `student`

Le Listing 4 montre les ajouts effectués pour déclarer l'entité `Student` en tant que composant Seam. Ainsi, l'annotation `@Name("student")` permet de déclarer un composant Seam baptisé `student`. Le composant est une instance de la classe `Student` identifiée par le nom `student`. Le composant peut ensuite être référencé par son nom, par exemple dans une page JSF.

```

@Stateful
@Name("studentManager")
@Scope(SESSION)
public class StudentManagerBean implements StudentManager {
    ...

    @Out("students")
    public List<Student> getStudents() { ... }

    @Factory("students")
    public void findAllStudents() { ... }

    public String addStudent(Student student) { ... }

    public void deleteStudent(Student student) { ... }

    ...
}

```

Listing 5 : Déclaration du composant `studentManager`

Le Listing 5 montre les ajouts effectués pour déclarer l'EJB *session stateful* `StudentManagerBean` en tant que composant Seam.

L'annotation `@Name("studentManager")` permet de déclarer un composant Seam baptisé `studentManager`. Le composant est une instance de la classe `StudentManagerBean` identifiée par le nom `studentManager`. L'annotation `@Scope(SESSION)` indique, qu'une fois créée, l'instance est associée au contexte Seam avec la portée « session », c'est-à-dire la session HTTP. Le composant cesse d'exister à la fin de la session. Ainsi, Il y a autant d'instances du composant `studentManager` que de sessions HTTP en cours, et chaque utilisateur voit sa propre instance du composant.

L'annotation `@Factory("students")` sur la méthode `findAllStudents()` informe Seam qu'il peut appeler cette méthode pour créer une instance du composant Seam baptisé `students`. Seam récupère ensuite l'instance du composant via la méthode `getStudents()`. L'annotation `@Out("students")` sur la méthode `getStudents()` indique à Seam d'associer le composant au contexte Seam sous le nom `students`.

Règles de validation

Le Listing 6 montre la définition de règles de validation sur l'entité `Student`. Pour cela, Seam met à contribution *Hibernate Validator* et ses annotations.

```
public class Student implements Serializable {
    ...

    @NotNull @Length(min=5, max=15)
    public String getPassword() {
        return password;
    }

    @NotNull
    @Pattern(regex="^[A-z0-9._%+-]+@[A-z0-9.-]+\.[A-z]{2,4}$",
            message="EMail isn't well formed,example toto@titi.com")
    public String getEmail() {
        return email;
    }

    ...
}
```

Listing 6 : Règles de validation du composant `student`

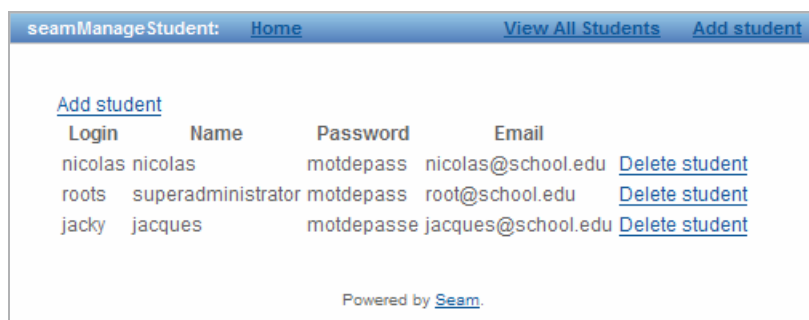
L'annotation `@NotNull` indique que la propriété `password` doit être obligatoirement renseignée. L'annotation `@Length(min=5, max=15)` impose une contrainte de longueur sur la propriété `password`. La taille de la chaîne doit être comprise entre 5 et 15 caractères. L'annotation `@Pattern` impose que la propriété `email` respecte l'expression régulière spécifiée.

L'interface de l'application

L'interface de l'application se compose de deux pages JSF :

- `view_all_students.xhtml`, une page de visualisation des étudiants (voir Figure 1),
- `add_student.xhtml`, une page de saisie d'un nouvel étudiant (voir Figure 2).

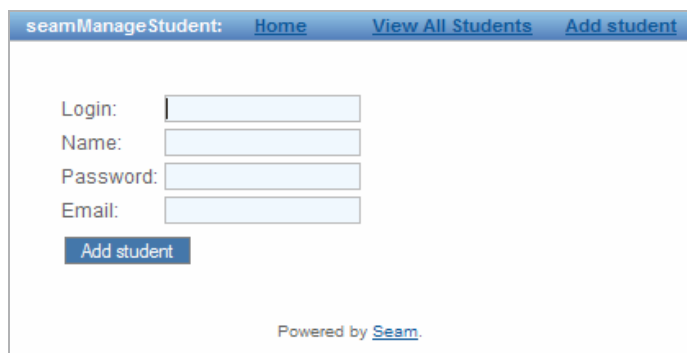
Les pages JSF se composent de composants JSF et utilisent diverses expressions EL. *Expression Language* (EL) est un standard commun à JSP et JSF. Seam fournit une extension de l'EL permettant l'appel de méthodes (avec passage de paramètres) et pas seulement l'accès aux propriétés.



Login	Name	Password	Email	
nicolas	nicolas	motdepass	nicolas@school.edu	Delete student
roots	superadministrator	motdepass	root@school.edu	Delete student
jacky	jacques	motdepasse	jacques@school.edu	Delete student

Powered by [Seam](#).

Figure 1 : Page de visualisation des étudiants



Login:

Name:

Password:

Email:

[Add student](#)

Powered by [Seam](#).

Figure 2 : Page de saisie d'un nouvel étudiant

Le Listing 7 présente le code source de la page `view_all_students.xhtml` de visualisation des étudiants.

```

<f:view>
<h:form>
  <h:commandLink value="Add student" action="add-student" /> <br />

  <h:outputText value="No Student Found
    rendered="#{students.size==0}" />

  <h:dataTable var="student" value="#{students}"
    rendered="#{students.size>0}">

    <h:column>
      <f:facet name="header">
        <h:outputText value="Login" />
      </f:facet>
      <h:outputText value="#{student.login}" />
    </h:column>

    <h:column>
      <f:facet name="header">
        <h:outputText value="Name" />
      </f:facet>
      <h:outputText value="#{student.name}" />
    </h:column>

    <h:column>
      <f:facet name="header">
        <h:outputText value="Password" />
      </f:facet>
      <h:outputText value="#{student.password}" />
    </h:column>

    <h:column>
      <f:facet name="header">
        <h:outputText value="Email" />
      </f:facet>
      <h:outputText value="#{student.email}" />
    </h:column>

    <h:column>
      <h:commandLink value="Delete student"
        action="#{studentManager.deleteStudent(student)}" />
    </h:column>
  </h:dataTable>
</h:form>
</f:view>

```

Listing 7 : Page `view_all_students.xhtml`

La page comporte un lien permettant de naviguer vers la page de saisie d'un nouvel étudiant. Ce lien est implémenté grâce au composant JSF `commandLink`.

La page comporte également un tableau qui affiche les étudiants. Ce tableau utilise un composant JSF `datatable` qui affiche le contenu de la collection `students`. L'expression EL `#{students}` fait référence à la variable `students` publiée par le composant Seam `studentManager`.

Chaque ligne du tableau correspond à un objet `Student` référencé symboliquement par la variable `student`. Cette variable est ensuite utilisée dans les diverses colonnes du tableau, notamment dans le cas de l'expression EL `#{student.login}`. Cette expression peut être évaluée de deux façons : soit en

consultant la valeur de la propriété, soit en affectant une nouvelle valeur à la propriété. Dans le premier cas, cela permet l'affichage, dans l'autre cas, la saisie.

La dernière colonne du tableau contient un lien qui permet de supprimer l'étudiant correspondant à la ligne. Ce lien est également implémenté grâce au composant JSF `commandLink`. L'action associée au lien est décrite par l'expression EL `#{studentManager.deleteStudent(student)}` qui fait référence à la méthode `deleteStudent` du composant Seam `studentManager`, ainsi qu'à la variable `student`.

Le Listing 8 présente le code source de la page `add_student.xhtml` de saisie d'un nouvel étudiant.

```
<f:view>
<h:form>
  <s:validateAll>
    <h:panelGrid columns="2">
      Login:
      <h:inputText value="#{student.login}" required="true"/>
      Name:
      <h:inputText value="#{student.name}" required="true"/>
      Password:
      <h:inputSecret value="#{student.password}" required="true"/>
      Email:
      <h:inputText value="#{student.email}" required="true"/>
    </h:panelGrid>
  </s:validateAll>

  <h:messages/>

  <h:commandButton value="Submit"
    action="#{studentManager.addStudent(student) }"/>
</h:form>
</f:view>
```

Listing 8 : Page `add_student.xhtml`

La page se compose principalement de composants JSF tels que `inputText` et `inputSecret`. Ces composants font référence au composant Seam `student` via des expressions EL telles que `#{student.login}`. Au moment de la soumission des données par l'utilisateur, les expressions EL sont évaluées en écriture. L'évaluation déclenche la création préalable du composant Seam `student`.

Les composants JSF sont encadrés par la balise Seam `validateAll`, qui indique de vérifier les règles de validation au moment de la soumission des données par l'utilisateur. Les erreurs éventuelles apparaissent au niveau du composant JSF `messages`.

La page contient un lien permettant de soumettre les données. L'action associée au lien est déclenchée si les règles de validation sont respectées. Cette action est décrite par l'expression EL `#{studentManager.addStudent(student)}` qui fait référence à la méthode `addStudent` du composant Seam `studentManager`, ainsi qu'au composant Seam `student`.

La navigation

JSF comporte en standard un langage de description de la navigation sous forme XML. Seam fournit en fait une extension de ce langage. Il permet également de décrire la navigation avec jPDL, le langage du moteur de processus jBPM de JBoss.

Le Listing 9 présente la description de la navigation entre les pages JSF de l'application.

```

<page view-id="*">
  <navigation>
    <rule if-outcome="home">
      <redirect view-id="/home.xhtml"/>
    </rule>

    <rule if-outcome="add-student">
      <redirect view-id="/add_student.xhtml"/>
    </rule>

    <rule if-outcome="student-added">
      <redirect view-id="/view_all_students.xhtml" />
    </rule>
  </navigation>
</page>

<exception class="org.jboss.seam.framework.EntityNotFoundException">
  <redirect view-id="/error.xhtml">
    <message>Not found</message>
  </redirect>
</exception>

```

Listing 9 : Description de la navigation entre les pages JSF

La navigation dans l'application se compose principalement de deux règles :

- une règle de navigation correspondant au lien 'Add student' de la page de visualisation des étudiants (view_all_students.xhtml),
- une règle de navigation correspondant au lien 'Submit' de la page de saisie d'un nouvel étudiant (add_student.xhtml).

La 1^{re} règle correspond au scénario suivant. Sur la page view_all_students.xhtml, l'utilisateur clique sur le lien 'Add student'. Conformément au Listing 7, le lien 'Add student' génère le dénouement add-student qui déclenche la 1^{re} règle de navigation et la redirection vers la page add_student.xhtml.

La 2^e règle correspond au scénario suivant. Sur la page add_student.xhtml, l'utilisateur clique sur le bouton 'Submit'. La méthode addStudent du composant Seam studentManager est alors appelée (voir Listing 8). Cette méthode retourne le dénouement student-added qui déclenche la 2^e règle de navigation et la redirection vers la page view_all_students.xhtml, conformément au Listing 2.

Pour conclure

Seam propose une approche élégante et naturelle du développement d'applications Web riches. Nous n'avons fait ici qu'entrevoir les possibilités de Seam qui couvre bien d'autres aspects : composants JSF, AJAX, workflow ...

Avec l'avènement de Java EE 5, un vent de simplification a soufflé sur J2EE, qui rejoint la tendance POJO. Néanmoins, les diverses technologies ne sont pas toujours très bien intégrées entre elles. C'est notamment le cas de JSF dont le manque d'intégration avec EJB et JPA est flagrant.

Plutôt que de fournir des technologies entièrement nouvelles, Seam part de Java EE 5 et fournit l'intégration et les extensions qui manquent cruellement. Notamment, Seam étend et intègre JSF : EL étendu, langage étendu de description de la navigation, etc. Seam permet également d'éviter les redoutables *managed beans*, véritable talon d'Achille de JSF.

Seam ne fait néanmoins pas de Java EE 5 une obligation. Il peut également intégrer sans difficulté particulière des POJO classiques. En plus de JSF, il s'est également ouvert à GWT pour l'aspect présentation.

N. le C.