- TD 2 -
Gérer des personnes avec des EJBs

On veut développer maintenant une application permettant non seulement de saluer des personnes mais également de les stocker dans une bases de données. Dans cette application on utilisera une méthode (utilisant JDBC) relativement naïve pour assurer la persistance ; une approche plus adaptée sera mise en place dans plus tard.

1. La gestion de ressources

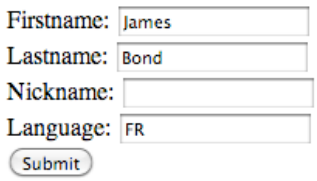
L'application proposera des fonctionnalités qui nécessitent parfois l'accès à différentes *ressources* comme, par exemple, une base de données permettant d'enregistrer des informations liées à des personnes. Pour faciliter la mise en place, nous utiliserons le serveur Derby intégré à NetBeans mais vous pouvez tester l'application avec une autre base de données installée sur votre machine. Les indications pour la création d'une base de données, pour la création de tables et pour leur visualisation (et remplissage) sont disponible à l'adresse http://netbeans.org/kb/docs/ide/java-db.html. On doit créer une BDD appelée `GLA` (gla/gla) et une table `PERSONS` contenant les colonnes `nickname`, `firstname`, `lastname` (un script pour la création de la table correspondante est disponible sur la page du cours).

Il faut également créer la ressource correspondante dans le serveur GlassFish en utilisant la console d'administration : on doit créer d'abord un JDBC Resource Pool et ensuite la JDBC Ressource. On va appeler le pool (de type `DataSource` et driver `Derby-30`) GLAPool - une fois créé le pool doit être testé avec un Ping (en cas d'échec vous devez probablement passer la propriété `SecurityMechanism` en 3 ou l'enlever). Il faut ensuite créer une ressource jdbc/gla dans le pool GLAPool. *Les ressources peuvent également être créées en utilisant NetBeans (catégorie Glasfish)*. Nous pouvons maintenant utiliser cette ressource pour assurer la persistance de notre application.

2. Des personnes avec leurs surnoms

On souhaite développer des pages permettant d'enregistrer des personnes dans une BDD et de consulter la liste des personnes déjà enregistrés. La logique métier de l'application est implantée en utilisant des EJBs.

Les personnes sont caractérisées par un nom, un prénom et un identifiant (l'identifiant est unique). Vous pouvez faire évoluer votre application existante ou en créer une nouvelle appelée, par exemple, GestionPersonnes (vous pouvez structurer l'application en utilisant une librairie pour les interfaces des EJBs et éventuellement pour d'autres classes/interfaces). Une première page (`index.xhtml`) doit permettre l'enregistrement de nouvelles personnes; cette page peut avoir la forme:

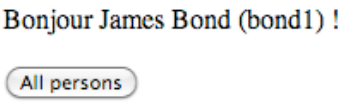


Dans un premier temps, le résultat de la validation du formulaire devrait être l'affichage (dans une autre page) d'un message de bienvenu pour la personne. Si l'utilisateur ne remplit pas le champ `Nickname` alors un identifiant est généré automatiquement à partir du nom (de la personne) et d'un entier aléatoire. Par exemple, le résultat pour les données ci-dessus devrait être l'affichage de « `Bonjour James Bond (bond20) !` » dans une page.

Pour implanter ce comportement on peut développer un EJB NameHandlerBean avec une méthode permettant de générer des identifiants à partir d'un nom. Ce bean doit également fournir la méthode greetingsMessage permettant (comme dans le TD précédent) de saluer la personne.

3. Lister les personnes

On développera maintenant un EJB PersonManagerBean qui permet d'ajouter des personnes dans la base données crée précédemment (vous pouvez utiliser éventuellement la classe Person fournie sur la page du cours). Dans un premier temps on suppose que la génération d'identifiants est suffisamment aléatoire et on ne vérifie pas l'unicité de l'identifiant; à terme il faudra générer des identifiants uniques et/ou gérer les éventuelles clés dupliquées. Suite à un ajout avec succès on doit être dirigé vers une page de la forme:

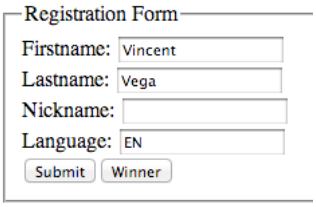


En utilisant le bouton « `All persons` » on peut obtenir la liste de toutes les personnes déjà enregistrées:

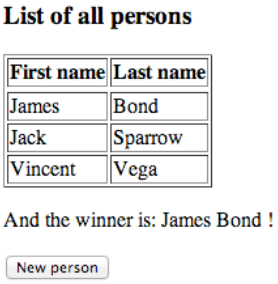
Commencez par identifier l'interface de l'EJB PersonManagerBean et implantez ce bean. Les beans ont évidement besoin d'accéder à différentes ressources - il faut penser à utiliser les callbacks pour optimiser leur utilisation.

4. Tirage au sort

Tous les jours une des personnes enregistrées est sélectionnée pour recevoir un prix - le choix est soit aléatoire soit via l'interface d'enregistrement. Pour sélectionner la personne explicitement il faut ajouter un bouton dans l'interface précédente:



Compléter l'application pour permettre l'affichage de cette personne efficacement. Par exemple, on peut imaginer que la personne gagnante est affichée au moment où on doit lister toutes les personnes:



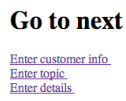
Pour être certain qu'une personne est sélectionnée dès le démarrage de l'application, on peut utiliser un bean qui permet d'enregistrer automatiquement un certain nombre de personnes.

5. Help desk

Les utilisateurs peuvent soumettre des tickets concernant différents problèmes. La soumission du ticket s'effectue en deux étapes : l'utilisateur fournit les informations personnelles (nom et prénom) et ensuite remplit les données (le topic et la description détaillée du problème). La soumission réalisée en plusieurs étapes permet la mise en place d'un certain workflow lié au métier de l'application (par exemple, les topics possibles sont proposés en fonction de l'identité de l'utilisateur). La soumission des tickets doit être réalisée avec une interface web et éventuellement avec un client classique.

Il faudra donc développer un Stateful EJB TicketManagerBean permettant de réaliser ce comportement. Le Bean sera appelé depuis un client d'application ainsi que depuis des pages web (JSP et JSF). Attention, l'appel depuis un client sans état (Servlet ou Managed Bean) doit être fait avec précaution.

Pour l'interface web, on pourrait avoir un enchainement d'écrans de la forme:



Si le workflow n'est pas respecté alors une exception `WorkflowViolationException` est lévée et doit être gérée dans les différents clients en affichant des messages/pages d'erreur. Les classes `Ticket` et `WorkflowViolationException` sont fournies. Pensez à structurer l'application en utilisant des packages.

En ce qui concerne l'interface du client d'application peut avoir la forme:

```
First name : James
Last name : Bond
Topic:  JEE
Issue: Big problem with my EJBs
```

et, dans un premier temps, le résultat pourrait être un affichage dans la console de la forme (à terme, on peut imaginer que les tickets sont sauvegardés dans une BDD):

```
Ticket{person=Person{firstName=James,lastName=Bond},topic=JEE,issue=Big problem with my EJBs}
```

On peut imaginer que l'enregistrement de chaque ticket est relativement long. Proposer, à terme, une solution où la durée de l'enregistrement est transparente pour le client, c'est-à-dire l'affichage du résultat (dans la console ou dans la page web) est fait sans attendre la création effective du ticket.

L'utilisation d'un STSB est justifiée dans notre cas pour la diversité de clients possibles. Ce type de session bean est également envisageable si les paramètres des méthodes invoquées ont des tailles importantes et par conséquent on souhaite minimiser leur utilisation - on pourrait, par exemple, imaginer une méthode qui prend en paramètre l'identité (qui contient entre autres une photo haute-définition) et fournit les topics correspondants. On peut également utiliser un STSB si on a besoin d'un `Extended Persistence Context`.