TP Web Hibernate, partie 1

Conception et la réalisation d'une application web

https://www.ersagun.ga/chat

Etudiant: Ersagun Yalcintepe

Intervenant: Gilles Halin

MIAGE M2 SID

Université de Lorraine

2016 - 2017

November 22, 2016

https://github.com/ersagun/chat.git

Résumé

L'objectif de ce TP1 était de conceptualiser et développer une application web en utilisant :

- Java EE pour la gestion des servlets,
- Hibernate pour la persistance des données,
- JSTL JSTL (Java server page Standard Tag Library) pour écrire du code java dans les pages jsp,
- Mysql (MariaDB) pour stockage des informations.

Sommaire

1 Introduction

Les principales cas d'utilisation de l'application web sont :

- Inscription d'un utilisateur,
- La connexion d'un utilisateur,
- Affichage de la liste des utilisateurs,
- Consultation des messages par un utilisateur,
- Envoi d'un message par un utilisateur,
- La déconnexion d'un utilisateur

Voici le schéma fourni qui représente les différents cas d'utilisations.

2 Conception

2.1 Diagrammes d'état stéréotypé décrivant la navigation mise en œuvre

2.1.1 Inscription

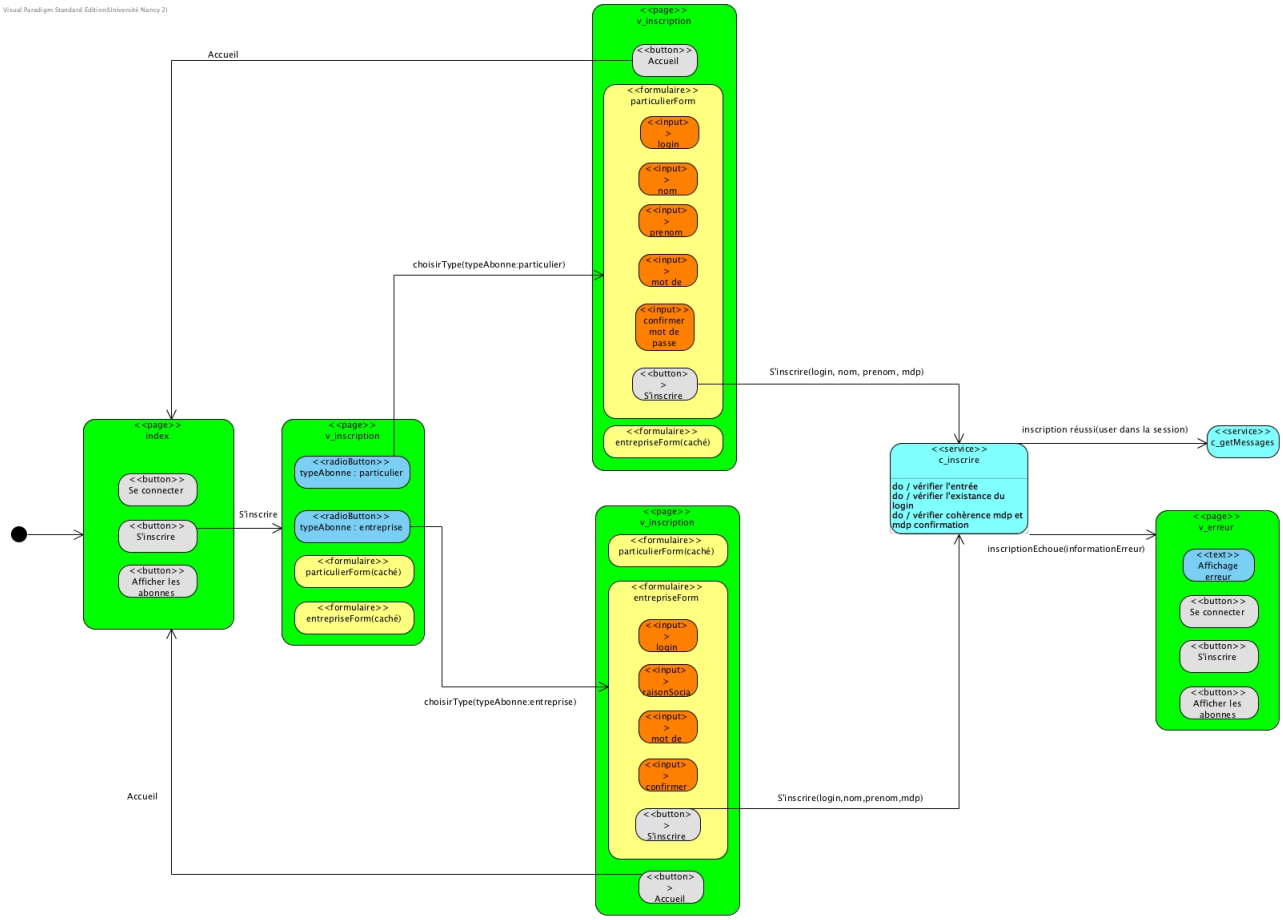

Figure 1 – Diagrammes d'état stéréotypé décrivant l'inscription d'un utilisateur

Nous pouvons voir que la page index contient 3 boutons nommé se connecter, s'inscrire et afficher les abonnes.

Dans ce diagramme nous allons simuler le clique de l'utilisateur sur l'inscription. Client dirigé vers la page v_inscription qui contient 2 formulaires nommé formulaireParticulier et formulaireEntreprise. Ils ont cachés au départ. V_inscription contient également 2 radio buttons l'utilisateur rend visible les formulaires on fonction de ces boutons. Si l'utilisateur est ce type particulier, il clique sur le radio bouton particulier. Le formulaireParticulier s'active input caché typeAbonne devient particulier. Utilisateur peut

mettre son login, nom, prénom et mot de passe. En cliquant sur le bouton utilisateur active le javascript qui fait la première vérification des inputs. S'il y a un problème l'utilisateur dirigé vers la page v_erreur qui affiche l'erreur sinon il accède au contrôleur contrôleur fait les vérifications. Si inscription réussie, il redirige l'utilisateur vers le chat, sinon il redirige vers v_erreur et affiche l'erreur.

2.1.2 Identification

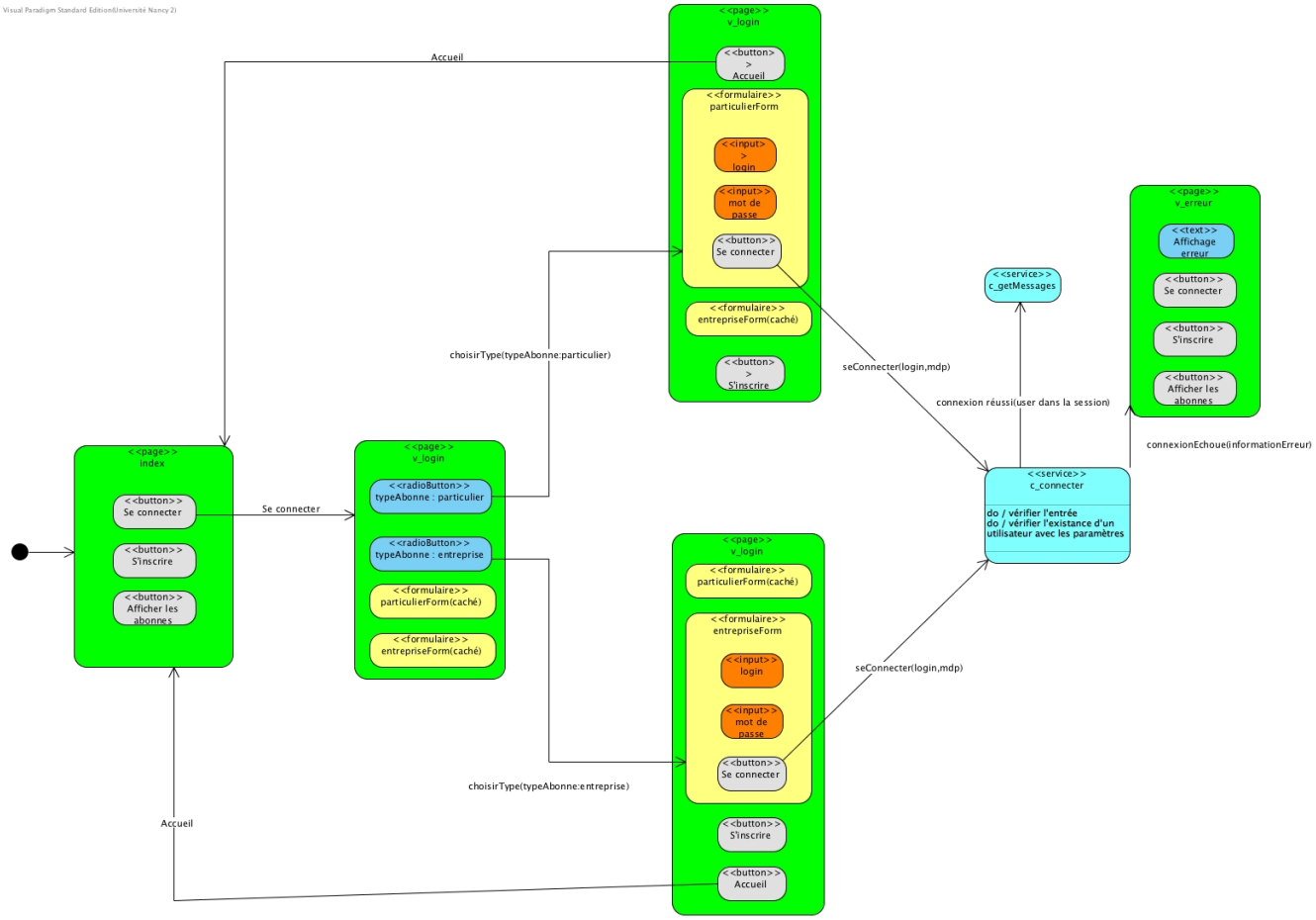

Figure 2 – Diagrammes d'état stéréotypé décrivant l'identification d'un utilisateur

Le mécanisme d'identification est identique à celle de l'inscription. Quand l'utilisateur accède au page v_login. Il peut rend visible les formulaires en cliquant formulaireParticulier et formulaireEntreprise. Les formulaires possèdent 2 input login et mot de passe. En cliquant sur le bouton utilisateur active le JavaScript qui fait la première vérification des inputs. S'il y a un problème l'utilisateur dirigé vers la page v_erreur qui affiche l'erreur sinon il accède au contrôleur contrôleur fait les vérifications. Si la connexion est réussie, il redirige l'utilisateur vers le chat, sinon il redirige vers v_erreur.

2.1.3 Consultation des messages

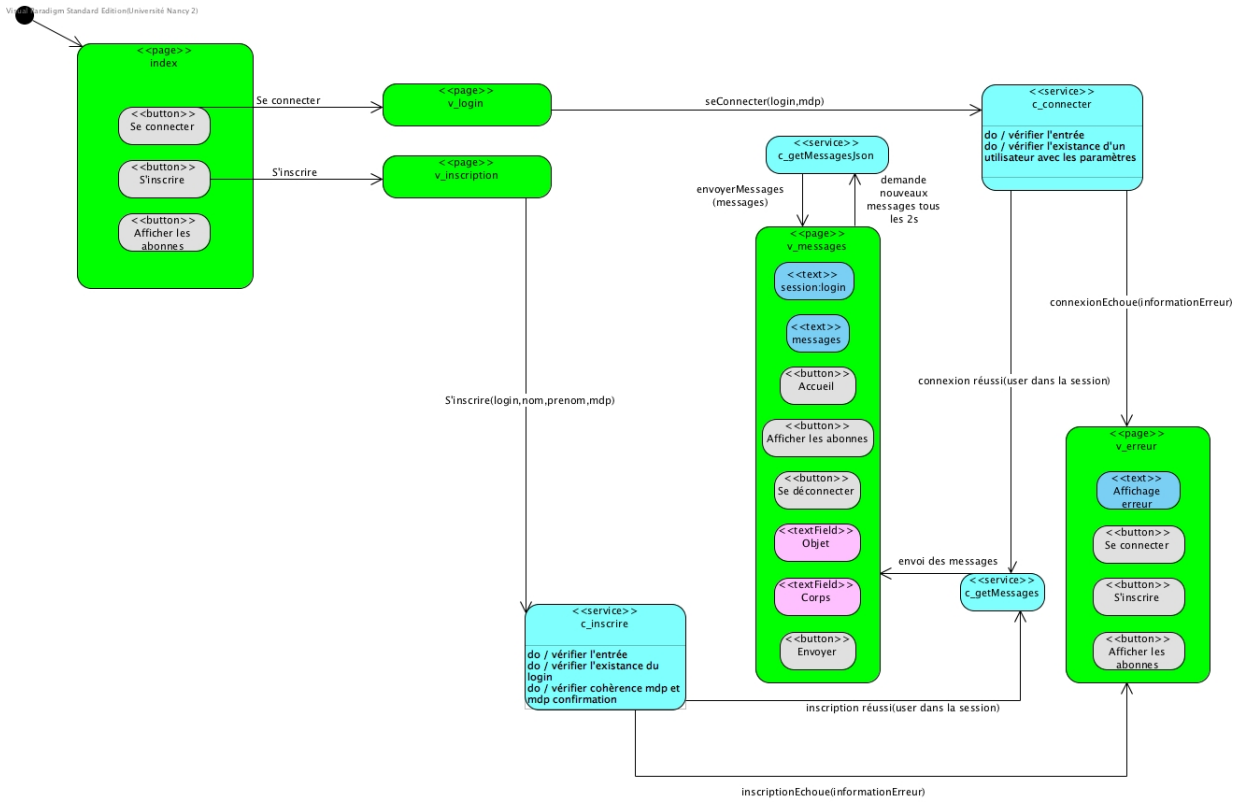

Figure 3 – Diagrammes d'état stéréotypé décrivant consultation des messages par un utilisateur

Pour pouvoir consulter le message il faut accéder au contrôleur c_getMessages donc il faut se connecter. La page d'inscription en cas de l'inscription réussie et la page de connexion en cas de connexion réussie nous emmènent vers le contrôleur c_getMessages. Ce contrôleur récupère les messages et ils passent les messages à la vue v_messages. V_messages est écrit en JSTL il parcourt les messages et les affiche. La vue v_messages contient du javascript qui fait une requête Ajax au serveur sur le contrôleur c_getMesssagesJson qui rècupere également les nouveaux messages les transforment en Json avec Gson et les proposent via response. Le JavaScript déserialise l'objet et modifie le contenu du JSTL en utilisant le jQuery.

2.1.4 Envoi des messages

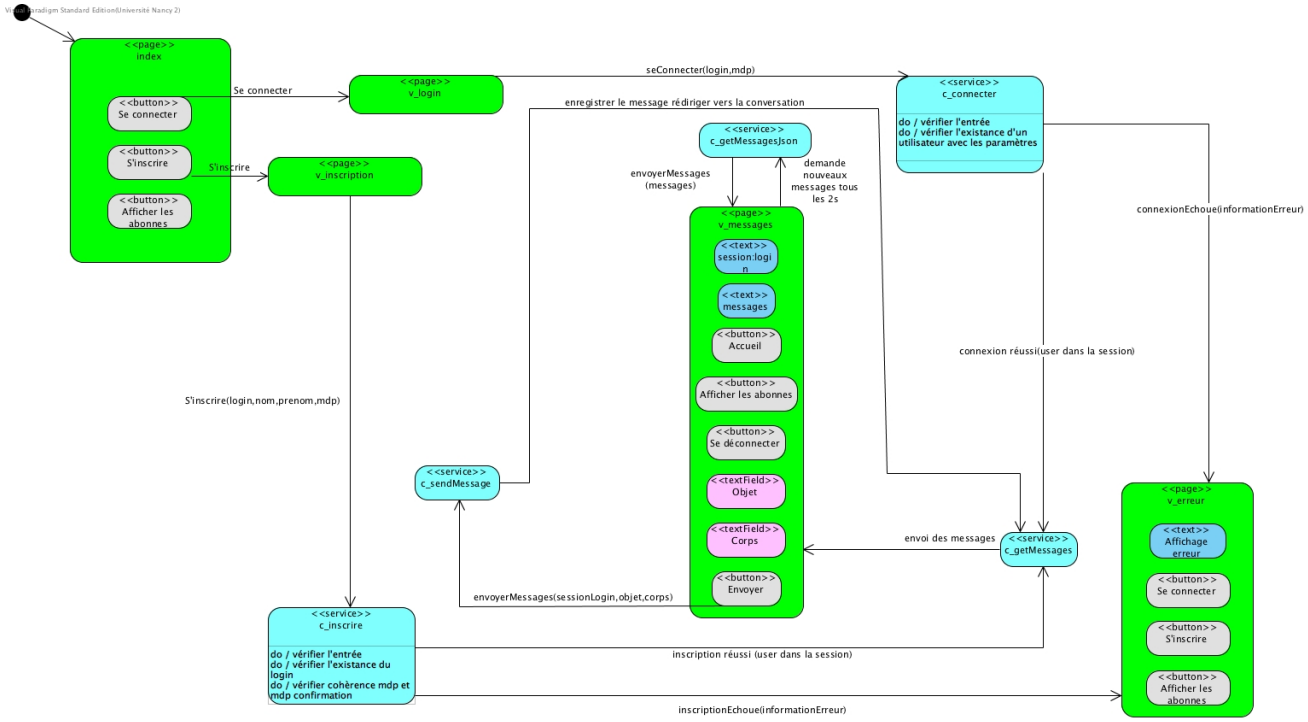

Figure 4 – Diagrammes d'état stéréotypé décrivant l'inscription d'un utilisateur

Le client connecté accède au page v_messages. La page v_messages contient un formulaire qui a comme input login, objet, corps. Un bouton envoyer à pour le but d'envoyer le formulaire. Login de l'utilisateur est automatiquement rempli car l'utilisateur est déjà dans la session. Une fois que l'utilisateur mets l'objet et corps, un JavaScript vérifient le contenu. En cas du problème il alerte avec Notify.js s'il n y a pas d'erreur l'utilisateur envoi le formulaire au contrôleur c_sendMessage qui enregistre le message dans la base de données et qui redirige vers la page c_getMessages.

2.1.5 Affichage des utilisateurs

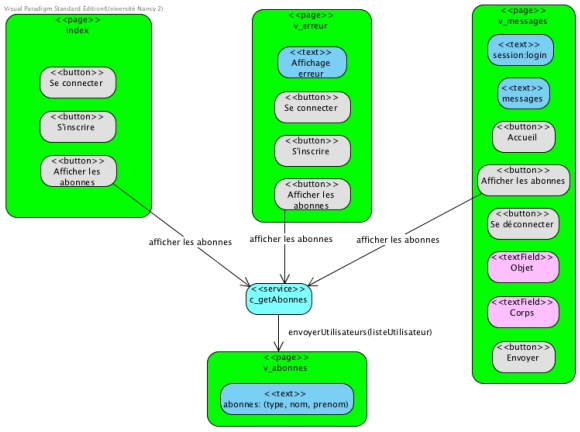

Figure 5 – Diagrammes d'état stéréotypé décrivant l'affichage des utilisateurs

L'utilisateur n'a pas besoin d'être connecté. A partir de l'index et v_messages et v_erreur utilisateur peut cliquer sur le bouton afficher les abonnes qui redirige l'utilisateur vers le contrôleur c_getAbonnes qui interroge la base de données, récupèrent les abonnes et redirige l'utilisateur vers la page v_abonnes qui à pour le but d'afficher les utilisateurs en jstl.

2.1.6 Déconnexion

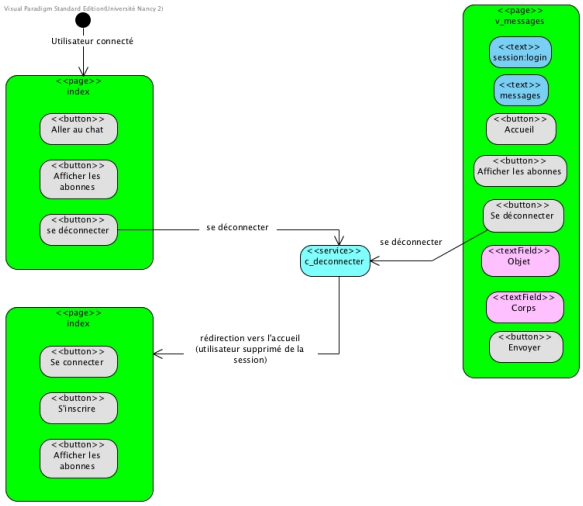

Figure 6 – Diagrammes d'état stéréotypé décrivant la déconnexion d'un utilisateur

L'utilisateur doit être connecté. Il peut voir à l'index et au v_messages un bouton se déconnecter qui le redirige vers c_deconnecter. Le contrôleur c_deconnecter supprime l'utilisateur de la session.

2.2 Diagrammes de classes stéréotypé décrivant les composants développés

2.2.1 Inscription

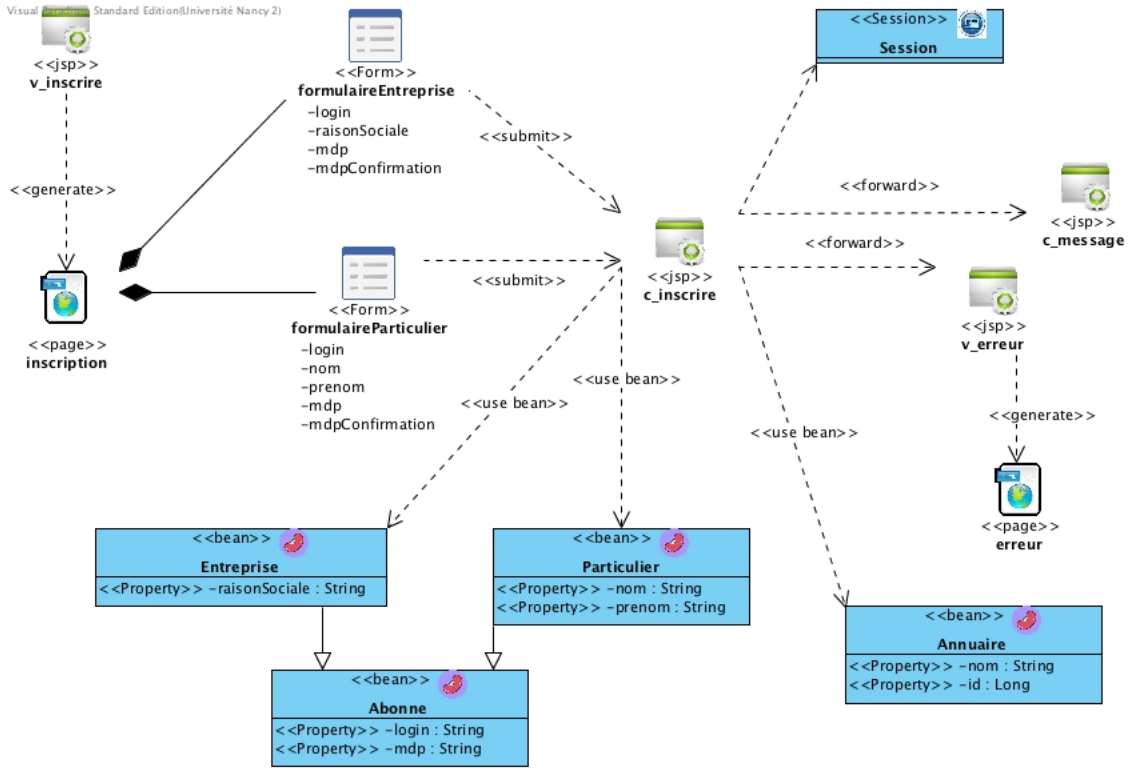

Figure 7 – Diagrammes de classes stéréotypé décrivant l'inscription d'un utilisateur

En cas d'une inscription réussie, l'utilisateur est inséré dans la session étant user. C_inscrire utilisent le bean Particulier et Entreprise pour vérifier s'il existe un utilisateur possédant le même login ou non. Si ce n'est pas le cas, il insère l'utilisateur. Ce n'est pas demandé mais dans le futur c_inscrire va récupérer les utilisateurs ou ajouter les utilisateurs dans l'annuaire de l'utilisateur courant. Pour l'instant nous ne faisons rien avec les annuaires dans c_inscription toutefois la table existe.

2.2.2 Identification

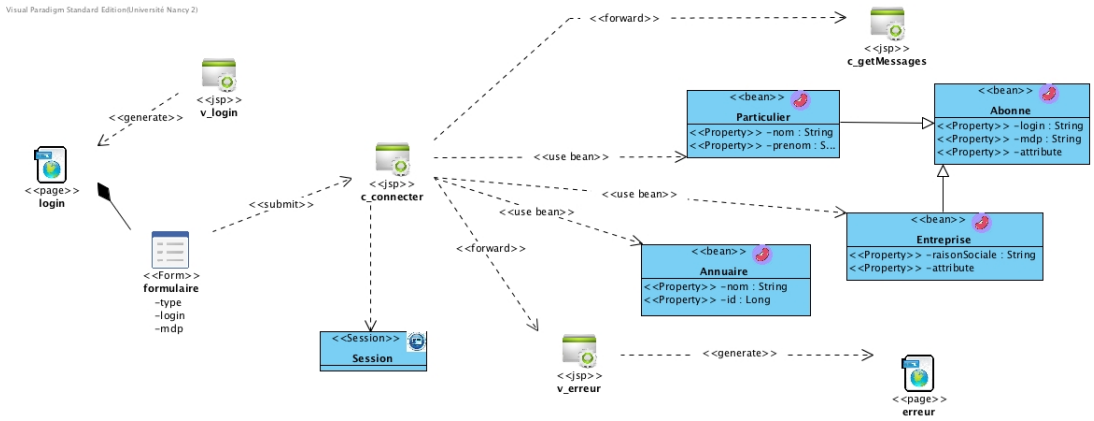

Figure 8 – Diagrammes de classes stéréotypé décrivant l'identification d'un utilisateur

La partie inscription et la partie connexion sont identique.

2.2.3 Consultation des messages

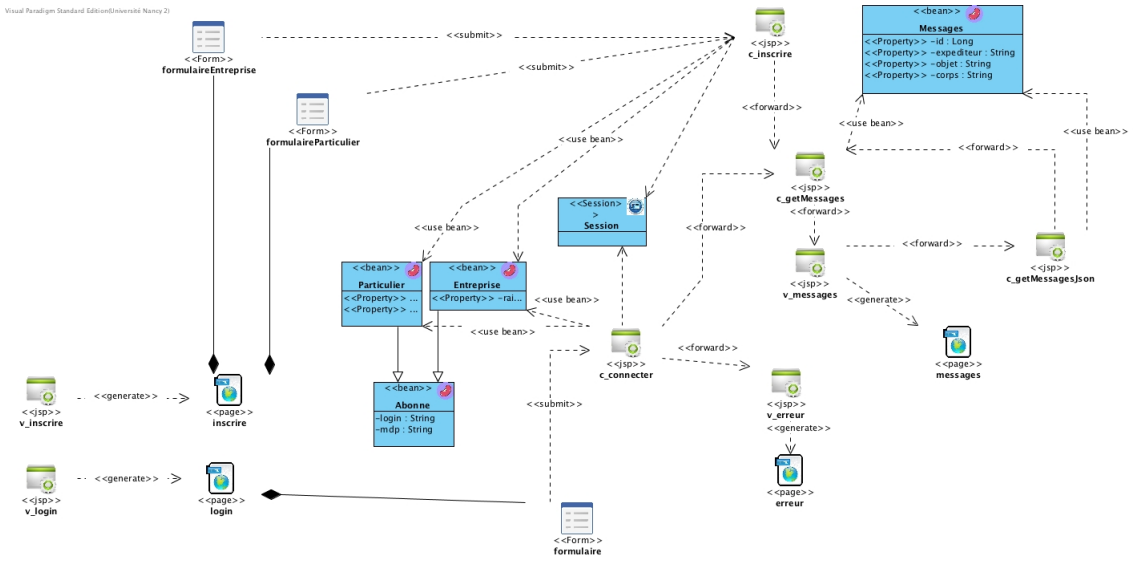

Figure 9 – Diagrammes de classes stéréotypé décrivant consultation des messages par un utilisateur

c_getMesssages et c_getMessagesJson utilisent le bean Message. C_connecter et c_inscrire utilisent la session pour stocker l'utilisateur en cas d'une connexion. Ils utilisent également les beans particulier et entreprise. Pour plus d'information voir l'explication dans la section de diagramme d'état stéréotypé décrivant consultation des messages par un utilisateur.

2.2.4 Envoi des messages

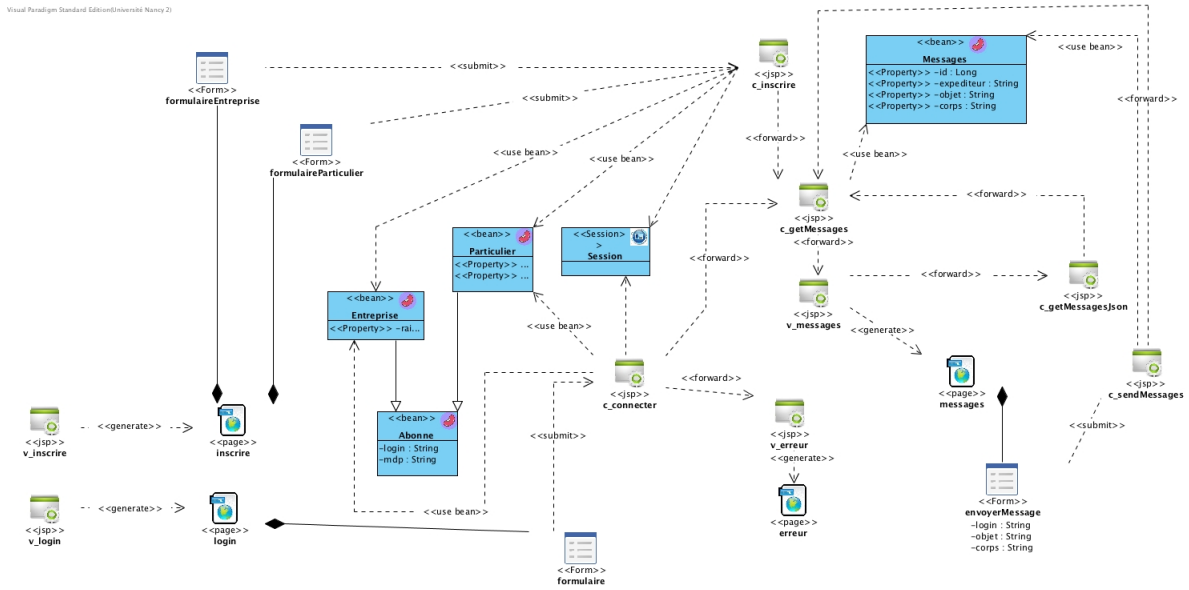

Figure 10 – Diagrammes de classes stéréotypé décrivant l'inscription d'un utilisateur

C_sentMessages utilise le bean message et inséré le message dans la base de données. Pour plus d'information veuillez voir l'explication dans la section de diagramme d'état stéréotypé décrivant envoi des messages par un utilisateur.

2.2.5 Affichage des utilisateurs

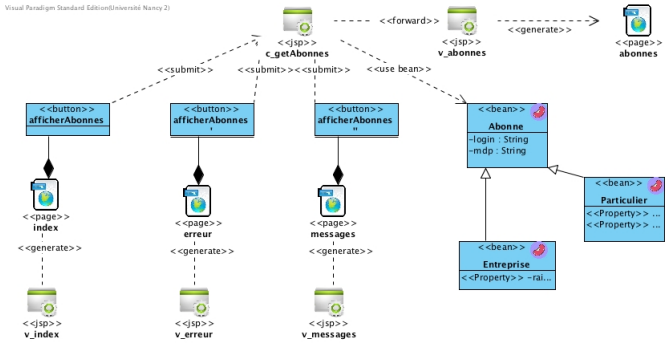

Figure 11 – Diagrammes de classes stéréotypé décrivant l'affichage des utilisateurs

Voir l'explication dans la section de diagramme d'état stéréotypé décrivant l'affichage des utilisateurs.

2.2.6 Déconnexion

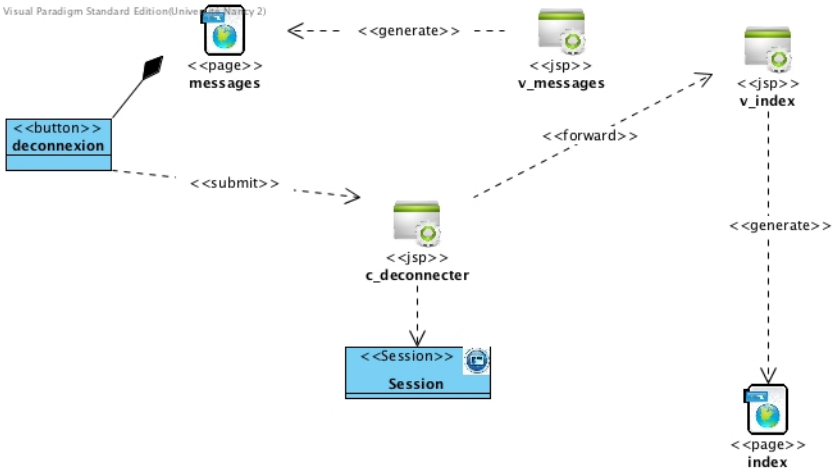

Figure 12 – Diagrammes de classes stéréotypé décrivant la déconnexion d'un utilisateur

Voir l'explication dans la section de diagramme d'état stéréotypé décrivant la déconnexion d'un utilisateur.

3 Implémentation

3.1 Choix de la stratégie pour l'héritage

Dans cette partie parmi les stratégies proposé par hibernate, nous allons choisir la stratégie qui réponds mieux à nos attentes.

- Une table par hiérarchie Dans notre cas on manipule plus les objets particulier et entreprise. On n'utilise pas abonné tout seul donc ce n'est pas interessant. De plus dans cette strategie pour chaque entreprise inseré on va devoir insérer le nom est le prenom en null. Pour chaque particulier inseré la raisonSociale on va devoir insérer la raisonSociale en null.
- Une table par classe fille On pouvait prendre une table par classe fille aussi mais dans ce cas on aurait eu une table entreprise qui possède qu'une colonne id et une colonne raison sociale. Cette stratégie est potentiellement intéressant pour notre cas.
- Une table par classe concrète Cette stratégie est similaire à une table par classe fille mais plus adapté pour notre cas. Au moment de la connexion je demande à l'utilisateur son type, ce qui me fait éviter au pire des cas de parcourir toutes les lignes de la table abonnés. A partir du login et type, je parcours Particulier ou Entreprise.

Dans le cas de L'héritage entre Abonne, Particulier et Entreprise, j'ai choisi dans la stratégie une table par classe concrète.

3.2 Paquetages de logiciels d'applications

- les jsps : Web Pages
- les fichiers css : Web Pages/css
- les fichiers js : Web Pages/js
- les objets persistants : org.miage.m2sid.chat
- org.miage.m2sid.util
- fichiers de mapping pour hibernate et hibernate config : src/main/resources

3.3 Dépendances Maven

- hibernate core 4.3.11.Final
- mysql-connector-java 5.1.23
- gson 2.2.2
- javaee-web-api version 7.0

Le client est fait en Bootstrap.

3.4 Execution

Vous pouvez accèder à l'application déployé par : https://www.ersagun.ga/chat

3.5 Structure du fichier ZIP déposé

- Visual paradigm project: Chat.vpp
- Projet Maven: TPWeb1
- Read-me: Informations

4 Conclusion

Grâce à ce projet, j'ai appris à ;

- Faire des diagrammes d'état stéréotypé décrivant la navigation
- Faire des diagrammes de classes stéréotypé décrivant
- Développer en Java EE + JSTL
- Mettre une persistance en place avec Hibernate

De plus grâce à ce projet, J'ai eu l'occassion de tester;

- le GitHub, logiciel de gestion de versions : https://github.com/ersagun/chat.git/
- Jenkins pour intégration continue, configuré avec github,
- Tomcat pour déployer l'application,
- Nginx pour mettre en place le reverse proxy et LaTeX pour la rédaction du rapport.