

# README

**Binôme:** Daniel Bensihmon et Mustapha Nezzari

## Contenu de l'archive

- Un script SQL permettant de créer la BDD et les tables en local et un autre pour créer celles du WEBTP (qui est déjà initialisée mais au cas où vous voudriez la réinitialiser pour diverses raisons).
- Les trois projets Maven qui contiennent le code source.
- Un dossier avec les dépendances au cas où elles ne se téléchargeraient pas.
- Un fichier contenant le mot de passe de la base de données WEBTP.
- Un jar de l'application compilée (toku.jar).
- Ce fichier README.

## Pré-requis

- Un ordinateur avec un JRE 8.
- Maven OU Eclipse munit de Maven (sinon bon courage pour le compiler à la main ..)
- De la patience.
- Avoir regardé l'intégrale de la petite maison dans la prairie.

## Etapas à suivre

- Extraire l'archive
- Exécutez le script SQL dans la BDD MySQL de votre choix (la structure existe déjà sur le Webtp avec des données dummy et la session "admin"). Le script de la vdd local ne fonctionnera pas sur le WEBTP (c'est pour cela qu'il y'a deux script).
- Ouvrez une fenêtre de commande dans le dossier et exécuter la commande "mvn clean install; java -cp toku-swing/target/toku-swing-0.0.2-SNAPSHOT-jar-with-dependencies.jar fr.lille1.univ.coo.tp.Main" pour nettoyer, compiler et exécuter le programme. Sinon si vous n'avez pas Maven vous pouvez utiliser la commande "java -jar toku.jar" pour lancer le programme à partir du jar compilé.
- Choisissez la façon dont vous voulez vous connecter :
  - Soit à une nouvelle base de données MySQL (locale)
  - Soit à la base de données Webtp où vous aurez juste à taper le mot de passe fourni et où la structure existe déjà avec le compte administrateur du programme.
- Connectez vous avec le login "admin" et le mot de passe "admin".
- Créez des amis imaginaires et parlez avec eux !

# Architecture générale du projet

---

Une architecture par feature (<http://www.javapractices.com/topic/TopicAction.do?Id=205>)

Tout d'abord le projet s'appelle Toku ! (Talk en japonais car je suis un garçon très original). Le projet est en fait divisé en trois répertoires principaux. Un répertoire parent "toku" et deux sous répertoire "toku-core" et "toku-swing" qui sont en fait des modules Maven du projet parent "Toku".

Comme vous l'aurez compris le dossier toku-core contient tous le modèle du programme et le dossier toku-swing contient la partie vue + contrôleurs.

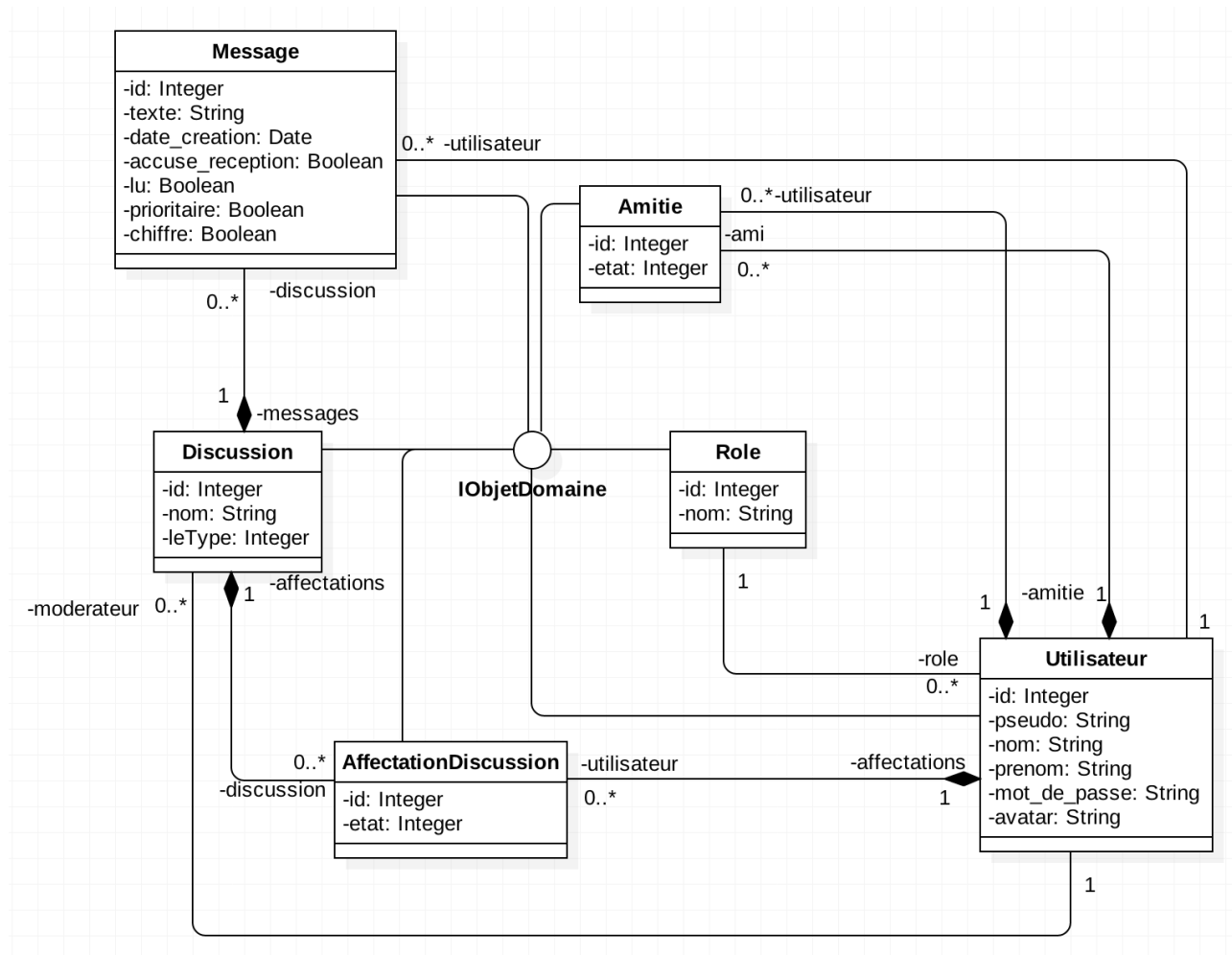
Dans les deux dossier j'ai respecté une certaines normes : Toutes les classes en rapport avec un thème se trouve dans un même package. Par exemple les classes Utilisateur, UtilisateurService et UtilisateurValideur se trouve dans un package "utilisateur". Il me semble plus facile de s'y retrouver plutôt que d'avoir des package "model", "service" et autres .. C'est une façon comme une autre de structurer les packages.

---

## La couche Domaine (UML)

La couche domaine contient toutes les classes du domaine. Ainsi, nous y retrouvons les classes suivantes :

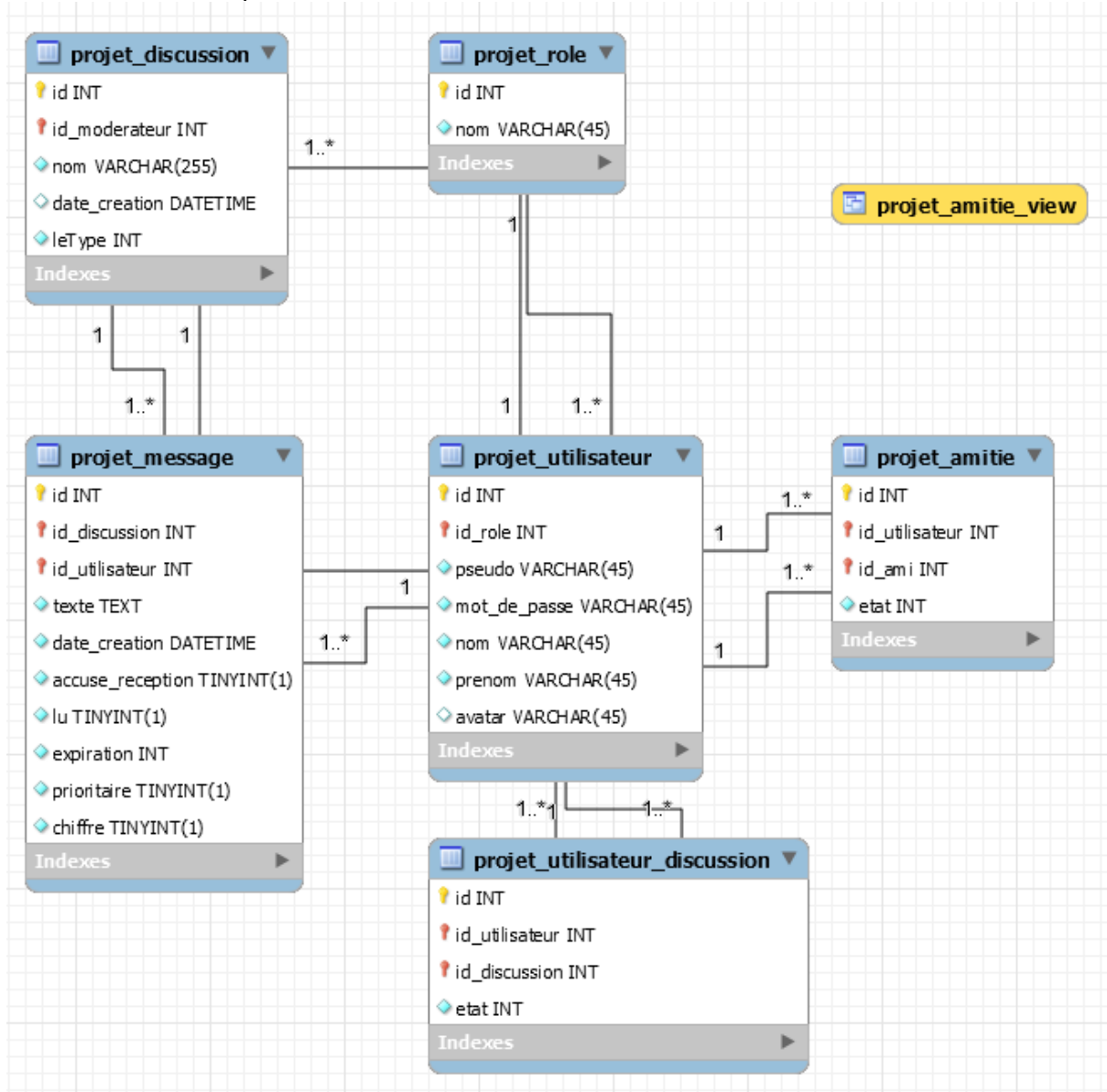
- Utilisateur : Un utilisateur
- Rôle : Le rôle d'un utilisateur (Normal ou Admin)
- Discussion : Une discussion (En groupe ou privée)
- Message : Un message de discussion
- Amitié : Un lien d'amitié entre deux utilisateurs
- AffectationDiscussion : L'affectation d'un utilisateur à une discussion à laquelle il participe (peut importe le type de discussion)



Une classe **IObservableList** existe afin de stocker des liste de ces objets. Elle contient deux fonctions **ajouter** et **supprimer** pour ajouter et supprimer des objets du modèle et notifier le **Unit Of Work** (dont je parlerais plus loin) afin qu'il ajoute ces objets en base.

## La couche persistance (MCD)

Voici le MCD correspondant à l'UML du Domaine :



Afin de mapper les tables de la BDD avec mes classes du Domaine, j'ai créé des annotations que j'ai placées sur ces dernières. Puis, grâce à un DAO générique, j'ai fait en sorte que l'on puisse, de façon générique, communiquer avec chaque table associée à sa classe. Ces annotations se trouvent dans le package "annotations" du modèle. Voici leurs détails :

- **@Table** : Cette annotation est à placer au dessus d'une classe. Par défaut le nom de la table a le même nom que la classe mais on peut spécifier le nom de la table s'il est différent du nom de la classe. On peut aussi spécifier le nom de la clé primaire de la table.
- **@Id** : Cette annotation est à placer au dessus de l'attribut qui représente la colonne contenant la clé primaire de la table. Le nom de la colonne est le même que le nom de l'attribut mais on peut en spécifier un s'il est différent.
- **@Colonne** : Cette annotation représente une colonne de la table. Le nom de celle-ci est le même que le nom de l'attribut mais on peut en spécifier un s'il est différent.
- **@Transient** : Cette annotation est à placer sur tout attribut qui ne doit pas être récupéré de la base de données pour des raisons de sécurité (Par exemple un mot de passe).
- **@Vue** : Cette annotation est à coupler avec une annotation @Table sur une vue. On a ainsi l'information de la table qui a servi à créer la vue. Il faut alors spécifier le nom de la vue en base de données et spécifier dans l'annotation @Table, le nom de la table et le nom de la colonne.

représentant la clé primaire. Ceci est important car les requêtes SELECT se feront sur la Vue tandis que les autres types de requêtes se feront sur la table (car on ne peut pas faire de mise à jour sur la Vue à moins qu'elle ne respecte certaines règles : <http://dev.mysql.com/doc/refman/5.7/en/view-updatability.html>).

- **@ColonneVue** : Cette annotation est à placer au dessus d'un attribut qui représente une colonne de la Vue qui elle prend les valeurs d'une colonne de la table. Ceci permet par exemple d'avoir les clés primaires de la table dans la Vue et de lier ces deux colonnes afin de savoir quelles clés de la table doivent être utilisées pour mettre à jour la table lorsque l'on fera nos requêtes de mise à jour sur la Vue. Ce sont un peu des colonnes "proxy".
- **@UnAUn** : Cette annotation n'est plus utilisée car elle fait la même chose que **@PlusieursAUn** mais je m'en suis rendu compte trop tard. Je l'ai donc passée en **@Deprecated**.
- **@PlusieursAUn** : Elle représente une relation n - 1. Par exemple : Plusieurs Discussions peuvent être créées par 1 utilisateur. Cette annotation prend en paramètre le nom de la clé étrangère et le type de la classe représentant la partie "1" de la relation. Le DAO générique injecte donc 1 objet du type spécifié dans les objets contenant celui-ci. Cette annotation se place sur l'attribut représentant l'objet à injecter.
- **@UnAPlusieurs** : Elle représente une relation 1 - n. Par exemple : Une Discussion peut avoir plusieurs messages. Cette annotation prend en paramètre le nom de la clé étrangère contenu dans la table représentant les objets de la partie "n" de la relation et le type de ces objets. Le DAO générique injecte donc une liste d'objets du type spécifié dans l'objet contenant cette liste. Cette annotation se place sur l'attribut représentant la liste à injecter.
- **@PlusieursAPlusieurs** : Cette annotation représente une relation n - n. Par exemple : Plusieurs utilisateurs peuvent être affectés à plusieurs Discussions. On aura ainsi d'un côté la liste des Discussions dans l'objet de type Utilisateur et de l'autre côté, la liste des membres dans l'objet de type Discussion. Cette annotation prend en paramètre le type de la classe représentant la table d'association, le nom de la colonne représentant la clé des objets contenant la liste, le nom de la colonne représentant la clé des objets contenus dans la liste.

Tout objet dont le type est spécifié en paramètre d'une annotation va être créé de façon paresseuse grâce au virtual proxy générique et à des factories génériques qui effectuent cette remontée grâce aux informations des 4 annotations de cardinalités (**UnAUnFactory**, **PlusieursAPlusieursFactory**, etc ...). Les autres objets de type plus primitif seront récupérés de façon normale grâce aux resultsets.

Lors d'une requête, le DAO va utiliser la réflexivité pour scanner la classe représentant la table à questionner et va utiliser les factories pour créer des Proxies.

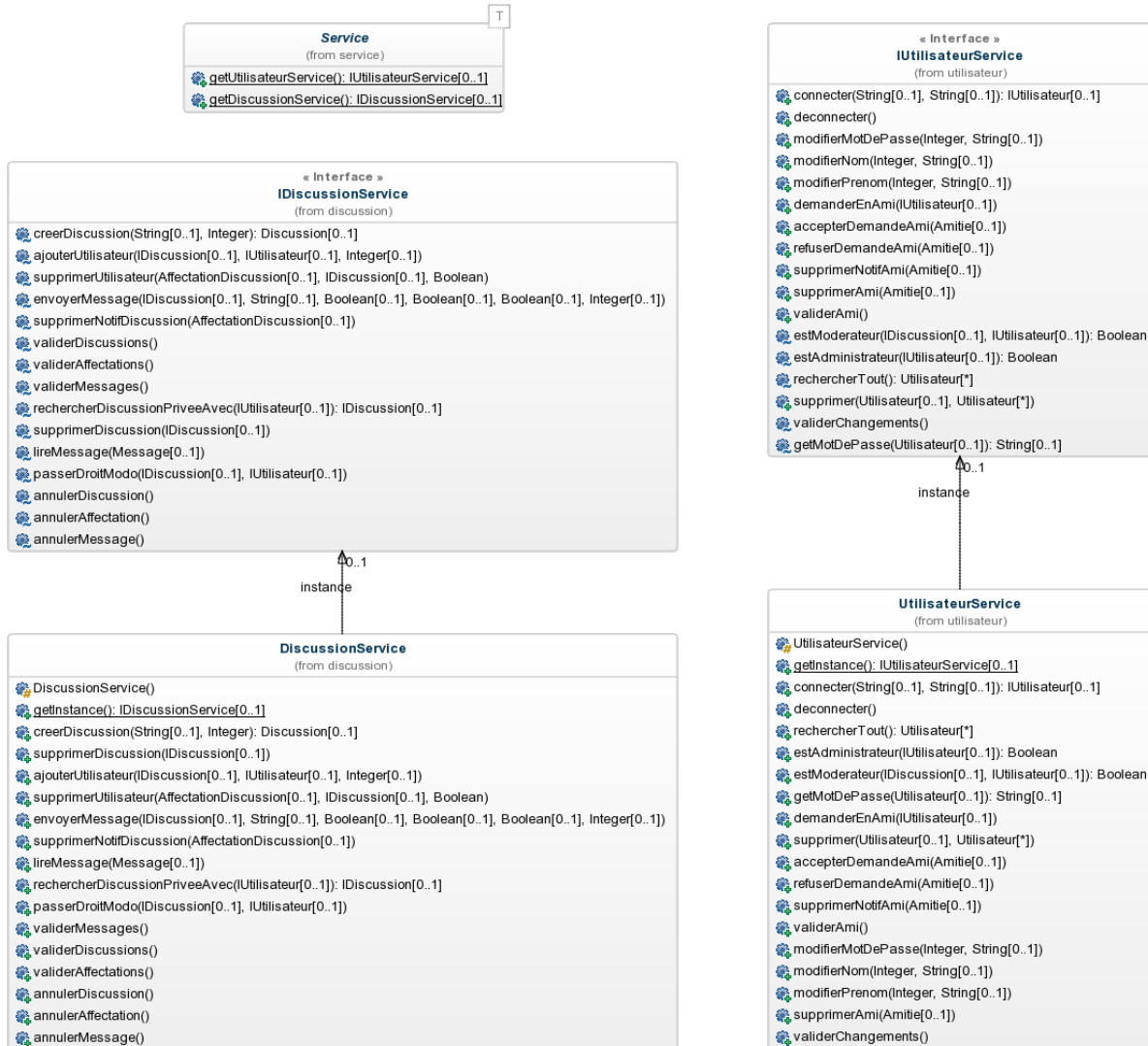
Enfin les objets du domaine sont stockés en mémoire grâce à des Map d'identité (Identity Map) afin de ne pas dupliquer les références vers ces objets. Cela permet d'optimiser les requêtes sur la base de données mais aussi d'avoir une seule référence par objet et ainsi d'économiser la mémoire.

[illegible]

Mots clé : Design pattern Proxy, Design pattern Factory, Design pattern identity map

## La couche Service

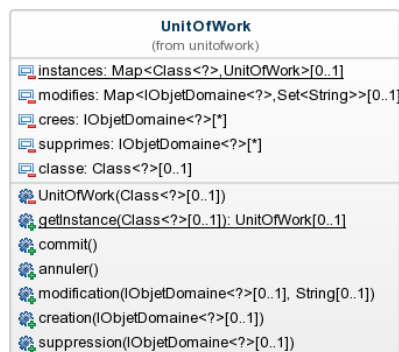
La couche Service est composé de deux classes : UtilisateurService et DiscussionService. Comme leur nom l'indique, la première gère tout ce est en rapport avec les utilisateurs comme la création de compte, la connexion, la gestion des rôle, l'ajout d'ami, etc, tandis que la deuxième gère tout ce qui touche aux discussions comme la création de discussion, l'envoi de message, l'affectation à une discussion etc. Ces classes sont utilisées par les contrôleurs de la couche Présentation.



## Le Unit of work

Afin de limiter les appels à la base de données, j'ai utilisé un Unit Of Work (UOW) qui est une sorte de cache où sont stockés tous les objets du Domaine créés, modifiés ou supprimés. Ce UOW suit le design pattern Observer pour observer tout les objets du Domaine sur lesquels ont effectue des changement. Il est notifié de la création, de la modification de la suppression de tous ces objets. Dès qu'un objet est créé ou supprimé dans la couche service, il est envoyé à ce UOW. Dès qu'un objet est modifié, il notifie lui même le UOW de ces changements en précisant le ou les paramètres modifiés. Enfin, j'ai créé une classe IObservableList qui notifie de l'ajout ou la suppression d'un objet du domaine comme par exemple lors de l'ajout d'un ami, le UOW est notifié de la création d'un nouvel objet Amitié qui a été ajouté à l'IObservableList "amitiés" de la classe Utilisateur.

Une fois que l'on veut faire persister ces changements en base, on appelle la fonction "commit" de ce UOW qui va utiliser le design pattern Visitor pour connaître le type des objets à traiter et donc à faire persister. Il se sert de trois objet : Le CreationCommitter, le ModificationCommitter et le SuppressionCommitter. Ce sont ces trois objets qui implément la classe Visitor ce qui leur permet de savoir comment créer/modifier/supprimer chaque type d'objet du Domaine.



Mots clés : Design Pattern Visitor, Design pattern Observer



---

## La couche Présentation

Cette couche rassemble toutes les classes représentant l'interface utilisateur. Elle est divisé en deux catégorie : Les fenêtres créés avec des composants de base ou custom et les contrôleurs qui interceptent les actions de l'utilisateur, récupèrent les informations à traiter et les passent à la couche Service. L'interface utilisateur sera décrite dans un prochain chapitre.

La JObservableList se sert des IObservableList pour se mettre à jour. Les composants graphiques sont génériques.

# Cas d'utilisation

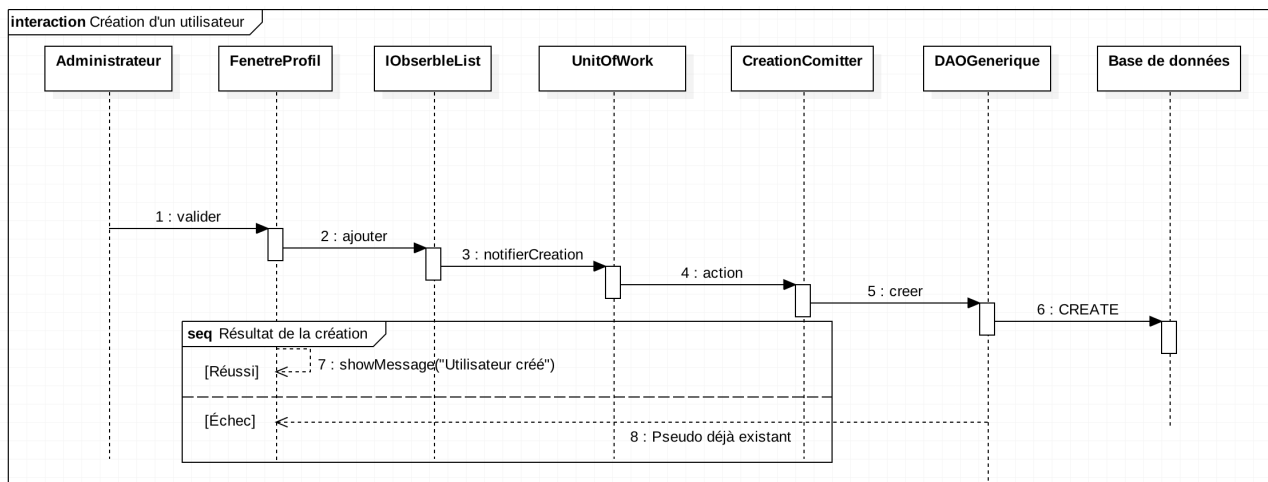
Tous les cas d'utilisation ont été implémentés.

## Création et suppression de compte utilisateur (CU 1)

Un administrateur peut créer et supprimer un utilisateur. Il doit pour cela renseigner les infos suivantes :

- Pseudo
- Mot de passe
- Nom
- Prénom
- Rôle (Admin ou utilisateur normal)

Voici le diagramme montrant les fonctions appelées lors de la création d'un utilisateur. Nous avons la même chose pour la suppression sauf que la fonction appelée est `notifierSuppression` et la classe du commiter est `SuppressionComitter`.



(Ce diagramme de séquence sera le seul car je n'ai pas eu le temps pour en faire d'autre)

---

## Modification de compte utilisateur (CU 2)

Un administrateur peut modifier les informations de n'importe quel compte y compris le pseudo. Un utilisateur ne peut modifier que son mot de passe, son nom et son prénom.

Lorsque l'utilisateur ou l'admin valide la modification, la classe UtilisateurValideur valide les changements et lève des exception si les champs ne sont pas bien remplis. Puis la modification est appliquée à l'objet représentant l'utilisateur, qui lui va notifier le Unit of Work de ce changement de propriété. Puis on valide les changements avec la fonction validerChangements qui va commit et faire persister ces changements en base.

---

## Connexion et déconnexion (CU 3)

Un utilisateur peut se connecter à son compte et se déconnecter. Lors de la connexion, on récupère les identifiants saisis et on recherche en base, grâce à l'UtilisateurService qui appelle le DAO générique, s'il existe un compte qui correspond à ces identifiants. S'il y en a un on stocke la session dans la classe Application sinon on lève une exception.

---

## Affichage de la liste des discussions disponibles (CU 4)

Lorsque l'on se connecte, un proxy de la liste des affectations aux discussions est créé. Cela permet d'afficher la liste des discussions dans la vue. Ceci est fait dans le DAO générique lors de la construction de l'objet Utilisateur.

---

## Envoi et réception de messages (CU 5)

Lors de l'envoi d'un message, on utilise la fonction envoyerMessage du DiscussionService pour ajouter un message dans l'ObservableList de message de la discussion. Cela va provoquer la notification de création d'un message par cette liste au Unit of Work qui va commuter ensuite le message en base. On met alors à jour toutes les affectations des utilisateurs pour les prévenir qu'il y a un nouveau message.

---

## Affichage des notifications (CU 6)

Lorsque l'on veut afficher la fenêtre des notifications, on itère sur la liste des demandes d'amis et la liste des affectations pour créer des NotificationPanel représentant les notifications. Ceci est fait dans la classe PopupNotification. J'aurais dû le gérer dans le modèle mais je n'ai pas eu le temps de changer ce comportement.

---

## Création d'un groupe (CU 7)

Un utilisateur peut créer une discussion privée ou en groupe. Dans le cas de la discussion privée, on vérifie d'abord s'il existe une discussion comprenant les deux membres grâce à la fonction DiscussionService.rechercherDiscussionPriveeAvec(). S'il y a une, on l'utilise pour ouvrir la fenêtre de discussion privée sinon on utilise la fonction creerDiscussion pour la créer. Dans le cas d'une discussion de groupe, on utilise directement la fonction creerGroupe. Une clause UNIQUE existe pour le nom des discussions. On a donc des exceptions lorsque l'on duplique les noms.

---

## Modération du groupe (CU 8)

Un administrateur peut modérer n'importe quel groupe auquel il appartient. Un admin ne voit pas toutes les discussions qui existe car il n'est pas censé travailler pour la NSA .. Un groupe n'a qu'un seul modérateur principal. Un modérateur peut affecter des amis à une discussion (`DiscussionService.ajouterUtilisateur()`), passer ses droit de modo (`DiscussionService.passerDroitModo()`) et supprimer un utilisateur (`DiscussionService.supprimerutilisateur()`).

---

## Recherche d'amis (CU 9)

Un utilisateur peut rechercher des amis. Lorsque l'on ouvre la fenêtre de recherche, on souhaite lister tout les utilisateurs de la base. On utilise donc le `RechercherToutFactory` pour récupérer tous les utilisateurs. Lorsque l'on sélectionne un ami à ajouter, la fonction `UtilisateurService.demanderEnAmi` crée une demande d'ami et l'insère dans la liste des demande d'amis de l'utilisateur. L'état de cet objet est mis à `EN_ATTENTE`. Puis lorsque l'ami accepte, l'état est mis à `VALIDEE` sinon à `REFUSEE`. Enfin lorsque l'utilisateur demandeur à vu la réponse de l'ami sollicité, la demande d'ami passe à l'état `TRAITEE`.

---

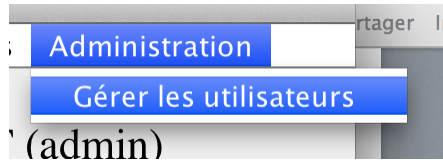
## Suppression d'un ami et départ d'un groupe (CU 10)

Un utilisateur peut supprimer des amis. Pour cela la fonction `UtilisateurService.supprimerAmi()` supprime l'amitié de la liste d'ami de l'utilisateur, ce qui notifie le Unit Of Work de la suppression. Un modérateur ne peut pas quitter le groupe tant qu'il reste des personnes dans ce groupe. C'est encore la fonction `DiscussionService.supprimerUtilisateur()` qui s'en occupe.

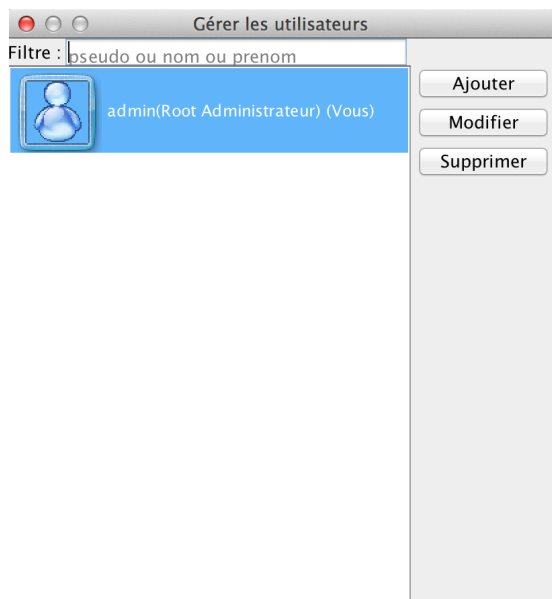
# Description de l'interface graphique

## Création d'un utilisateur

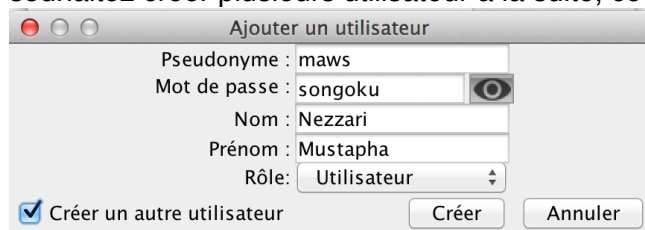
Connectez vous avec un compte administrateur. Allez dans le menu "administration" -> "gérer les utilisateurs"

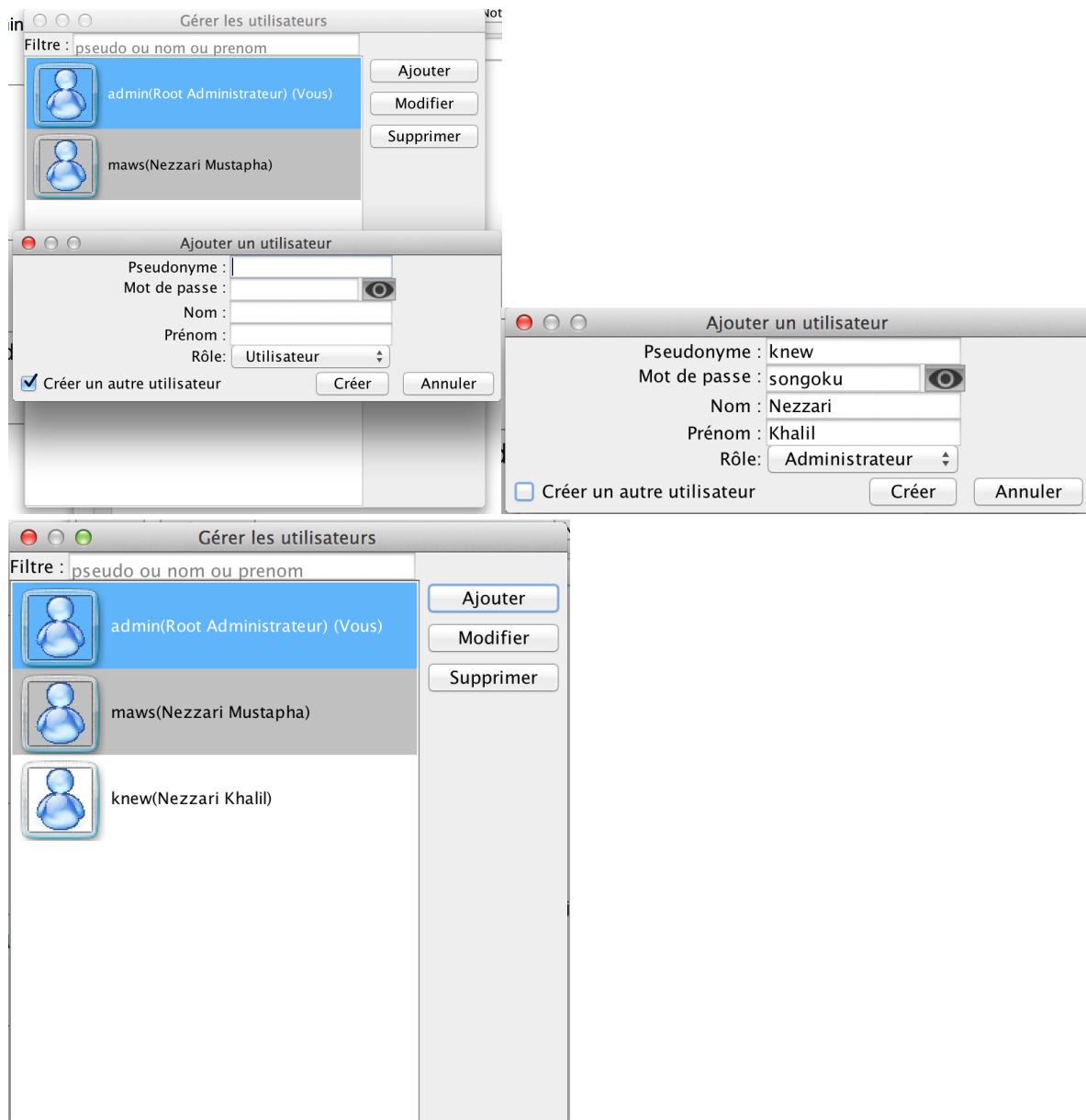


Vous arrivez sur la fenêtre suivante :

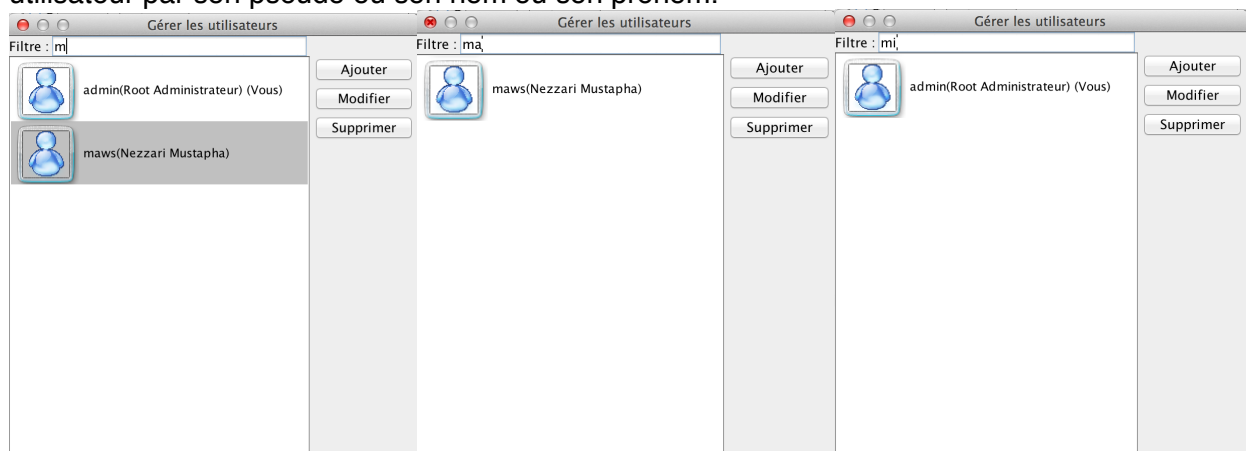


Cliquez sur le bouton "ajouter". Une fenêtre apparait. Remplissez les informations. Si vous souhaitez créer plusieurs utilisateur à la suite, cochez la case "créer un autre utilisateur".





Au passage, vous disposez d'une barre de filtre des utilisateur. Vous pouvez rechercher un utilisateur par son pseudo ou son nom ou son prénom.



---

Modification d'un utilisateur

En mode Administrateur

Toujours dans le menu “Gérer les utilisateur”, cliquez sur un utilisateur puis sur le bouton “Modifier” ou double cliquez sur un utilisateur. Modifiez les informations puis cliquez sur “Modifier”.

En mode Utilisateur

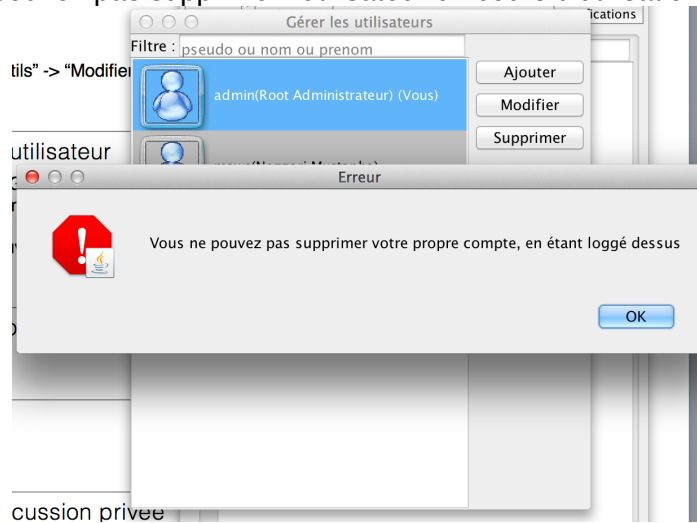
Cliquez sur le menu “Outils” -> “Modifier mon profil”.

---

## Suppression d'un utilisateur

Toujours dans le menu “Gérer les utilisateur”, cliquez sur un utilisateur puis sur le bouton “Supprimer”. Puis confirmez la suppression.

Sachez que vous ne pouvez pas supprimer l'utilisateur en cours d'utilisation de session.



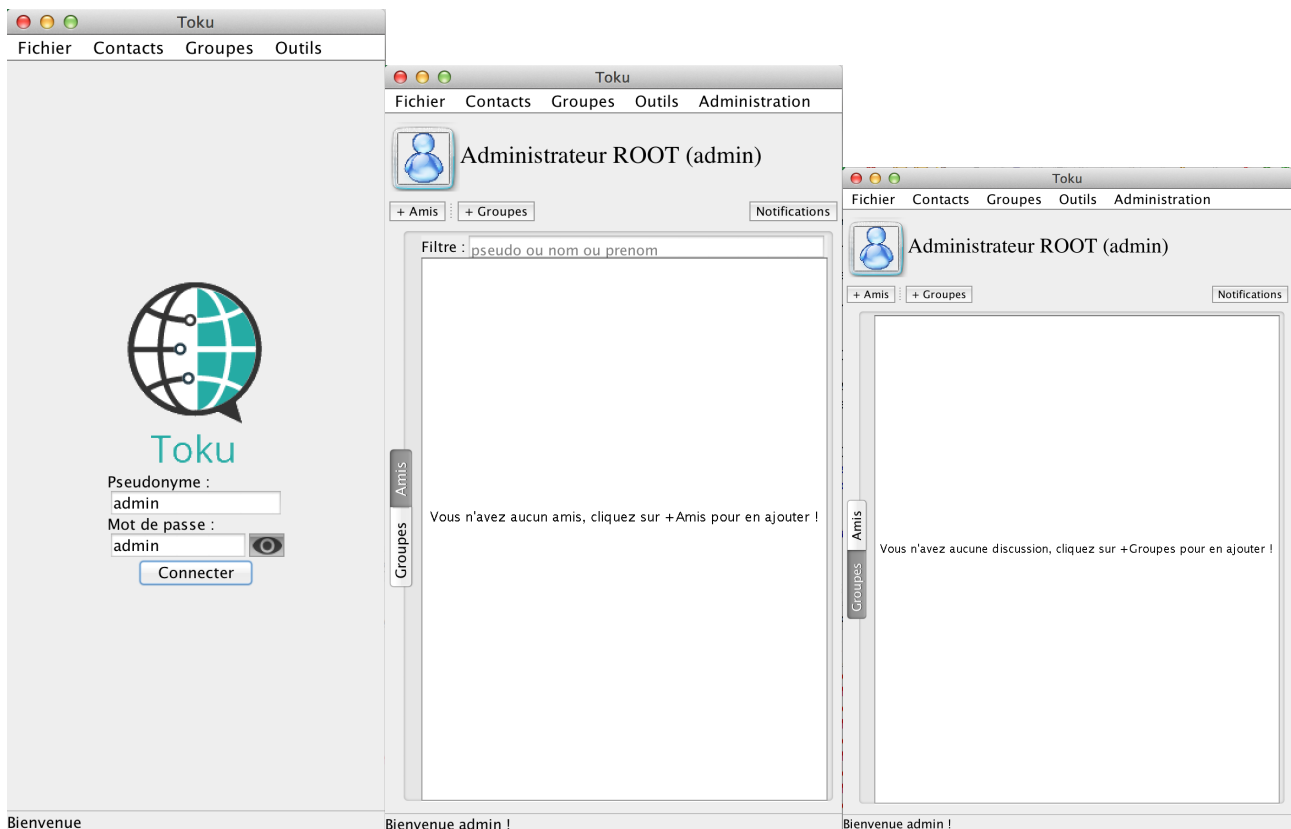
## Connexion et déconnexion d'un compte

Lorsque vous vous connectez à votre compte intergalactique, vous arrivez sur le tableau de bord de Toku. Votre nom, prénom et pseudo sont affiché en haut avec un jolie avatar vide (pas eu le temps pour ça ..).

Juste en dessous, une toolbar qui permet de rechercher un utilisateur pour le demander en amis, créer un nouveau groupe et afficher les notifications.

Encore en dessous, vous disposez d'un panel contenant deux onglets affichant vos amis et vos discussion privées et en groupe. (Ne pleurez pas car vous n'avez pas d'amis, ce n'est pas ma faute.. ajoutez en ! Mais bon, gardez bien à l'esprit que ce sont des amis qui ne vous répondront pas, sauf si vous le faites à leur place ..).

Enfin tout en bas une barre d'état non exploitée.



## Ajout d'un ami

Pour ajouter un amis, cliquez sur le bouton “+Amis”. Recherchez une personne grâce à la barre de recherche puis double cliquez sur la personne à inviter. Elle recevra une notification de demande d'amis.

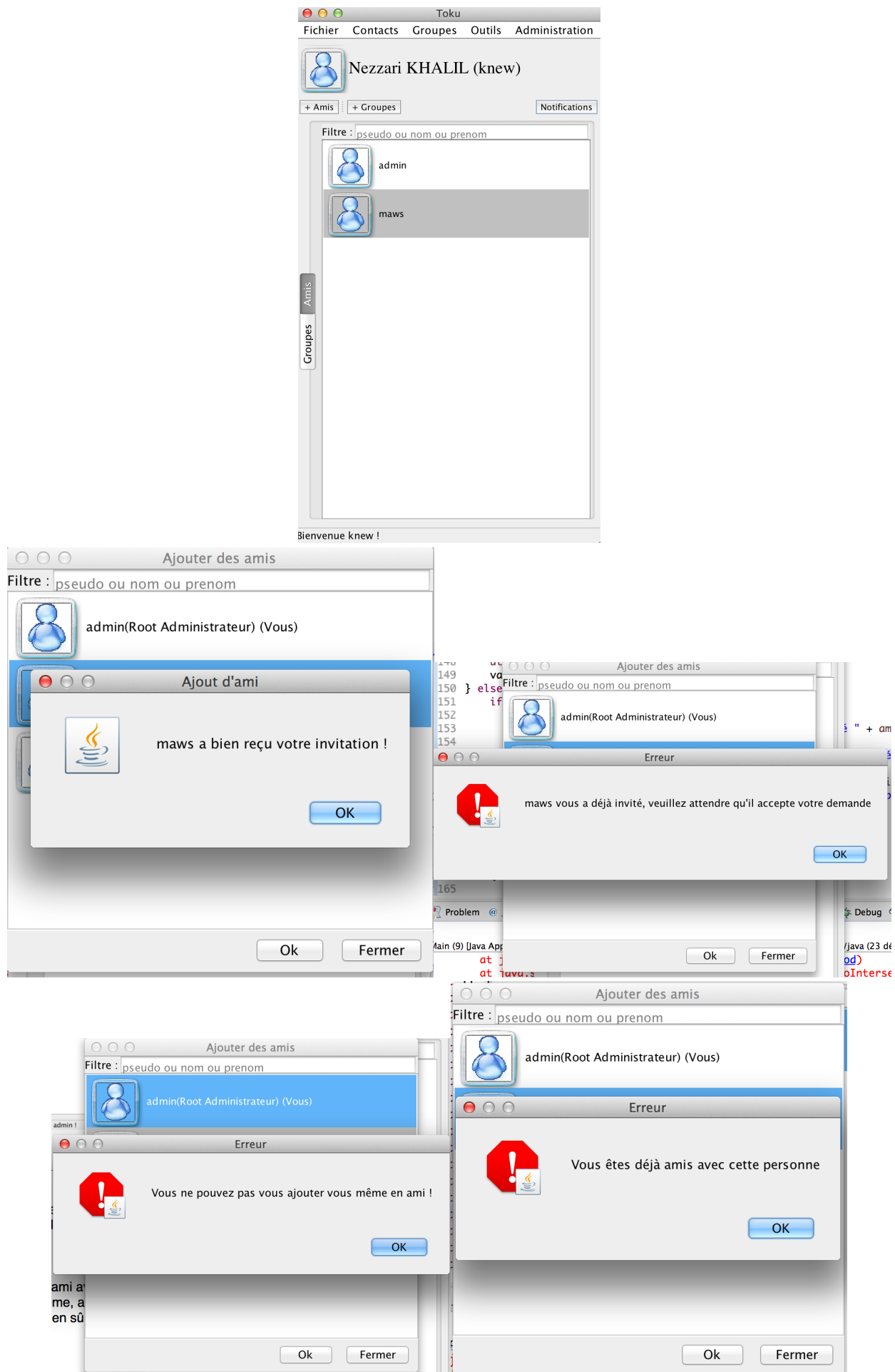
Si vous réinvitez la personne alors qu'elle n'a pas accepté, vous serez averti du fait que vous l'avez déjà invité.

Si vous invitez une personne déjà ami avec vous, vous serez averti.

Si enfin vous vous invitez vous même, allez voir un psy car vous êtes schizophrène (et mon application vous le dira poliment bien sûr).

Une fois ajouté et l'invitation acceptée, l'ami apparaît dans la liste d'amis.



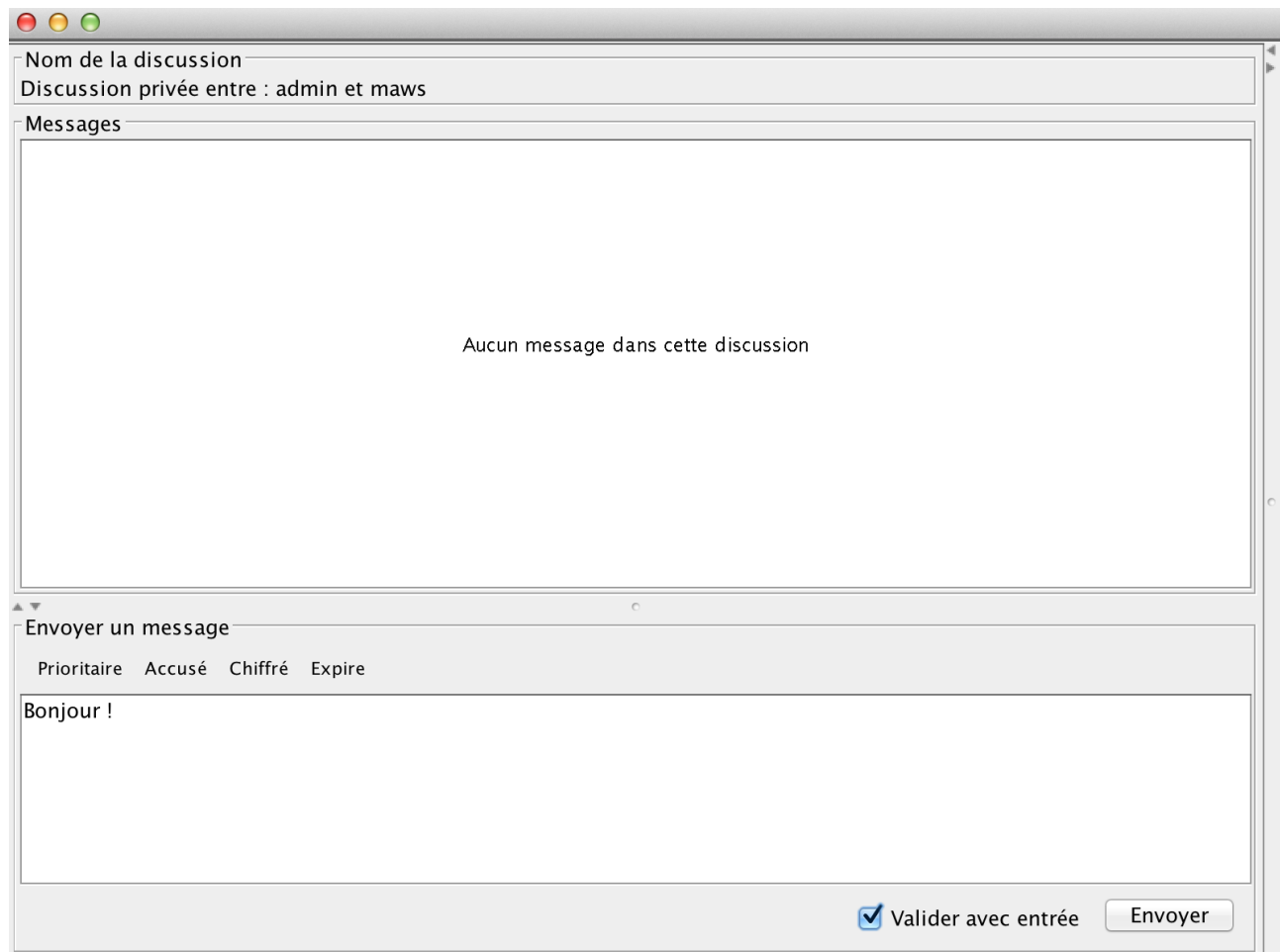


---

## Création d'une discussion privée

Double cliquez sur un amis pour démarrer une conversation privée.

Une discussion privée ne peut pas être supprimée même si les deux personnes redeviennent amis, l'archive est conservée.



The screenshot shows a web-based chat interface. At the top, there's a title bar with three colored buttons (red, yellow, green). Below it, a header section contains the text 'Nom de la discussion' followed by 'Discussion privée entre : admin et maws'. The main area is labeled 'Messages' and contains the text 'Aucun message dans cette discussion'. At the bottom, there's a section titled 'Envoyer un message' with four radio buttons: 'Prioritaire', 'Accusé', 'Chiffré', and 'Expire'. Below these is a text input field containing 'Bonjour !'. At the bottom right, there is a checkbox labeled 'Valider avec entrée' which is checked, and an 'Envoyer' button.

---

## Création d'une discussion de groupe

Il est possible de créer une discussion de groupe en cliquant sur le bouton “+Groupe”. Comme demandé dans le cahier des charge, une discussion ne peut avoir le même nom qu’une autre. Le panel de modération sur la droite est caché pour les utilisateurs non administrateur/modérateur du groupe.

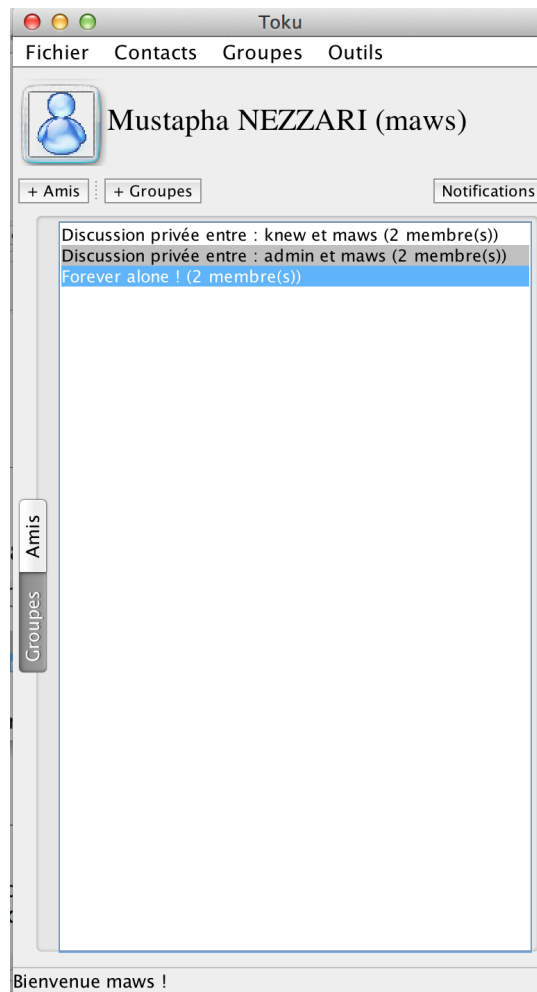
The screenshot shows a web application window with a title bar containing three colored buttons (red, yellow, green). The main content area is divided into three sections:

- Nom de la discussion**: A text field containing "Groupe : Forever alone !".
- Messages**: A large text area displaying "Aucun message dans cette discussion".
- Envoyer un message**: A section with a header bar containing "Prioritaire", "Accusé", "Chiffré", and "Expire". Below this is a large text input field. At the bottom right of this section are two buttons: "Valider avec entrée" (disabled) and "Envoyer".

On the right side of the window, there is a sidebar with two sections:

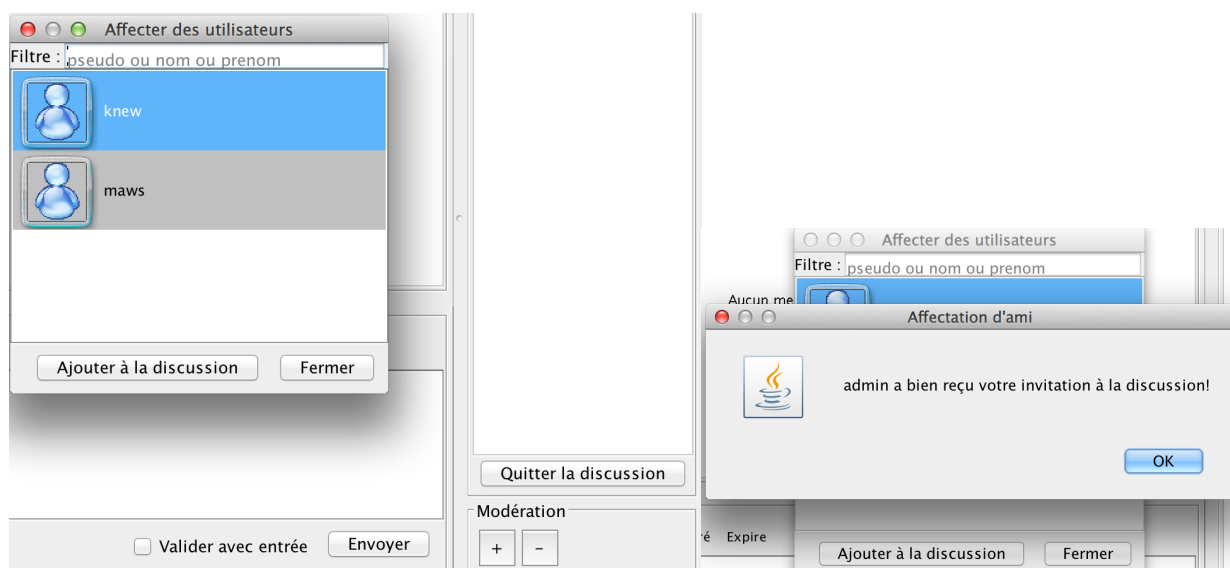
- Liste des membres**: A list containing "admin (Admin) (Modérateur)".
- Modération**: A section containing two buttons, "+" and "-", and a "Quitter la discussion" button below them.

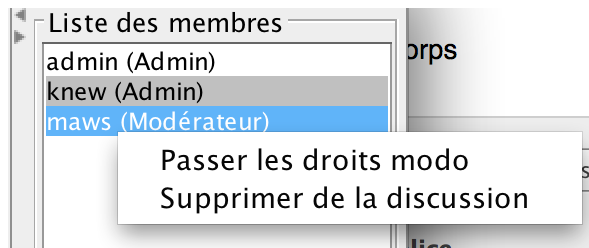
Toutes les discussion sont affichées dans l'onglet Groupes sur l'écran d'accueil.



## Modération d'un groupe de discussion

Vous pouvez ajouter ou supprimer des personnes en utilisant le panneau de modération en bas à droite ou le menu contextuel lorsque vous êtes connecté avec un compte administrateur. Pour ajouter des utilisateur, cliquez sur “+” puis double cliquez sur les amis à ajouter à la discussion. Pour les supprimer cliquez droit ou cliquez sur “-”. Pour passer les droits modérateur à un autre utilisateur, cliquez droit sur celui-ci et cliquez sur “Passer les droit modo”. Tout utilisateur ajouté à une discussion est prévenu par une notification.

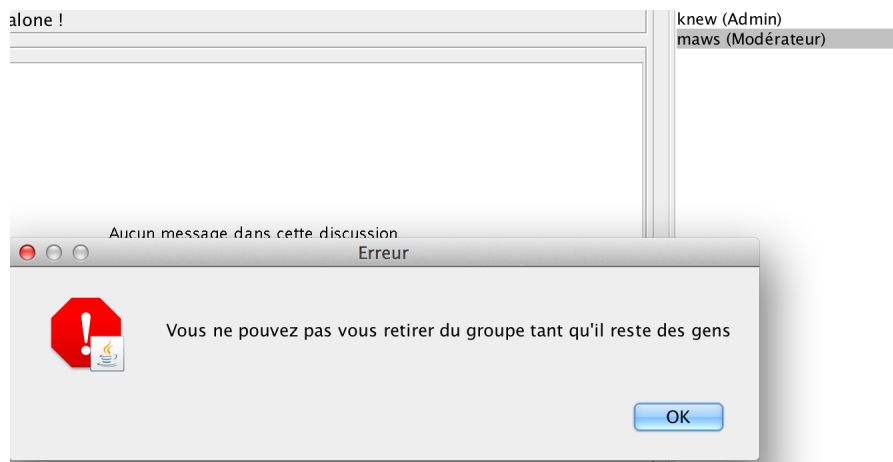




---

## Départ d'une discussion

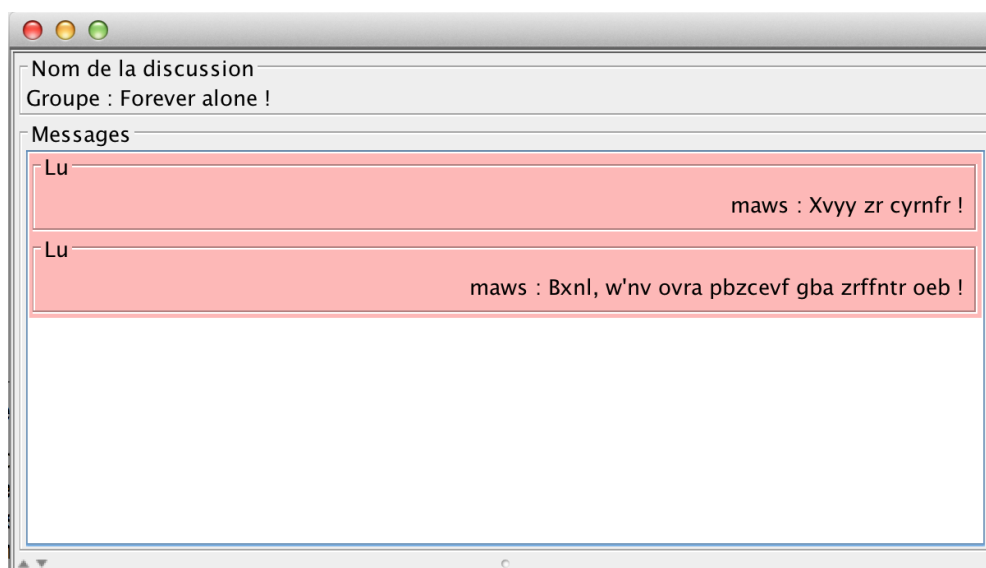
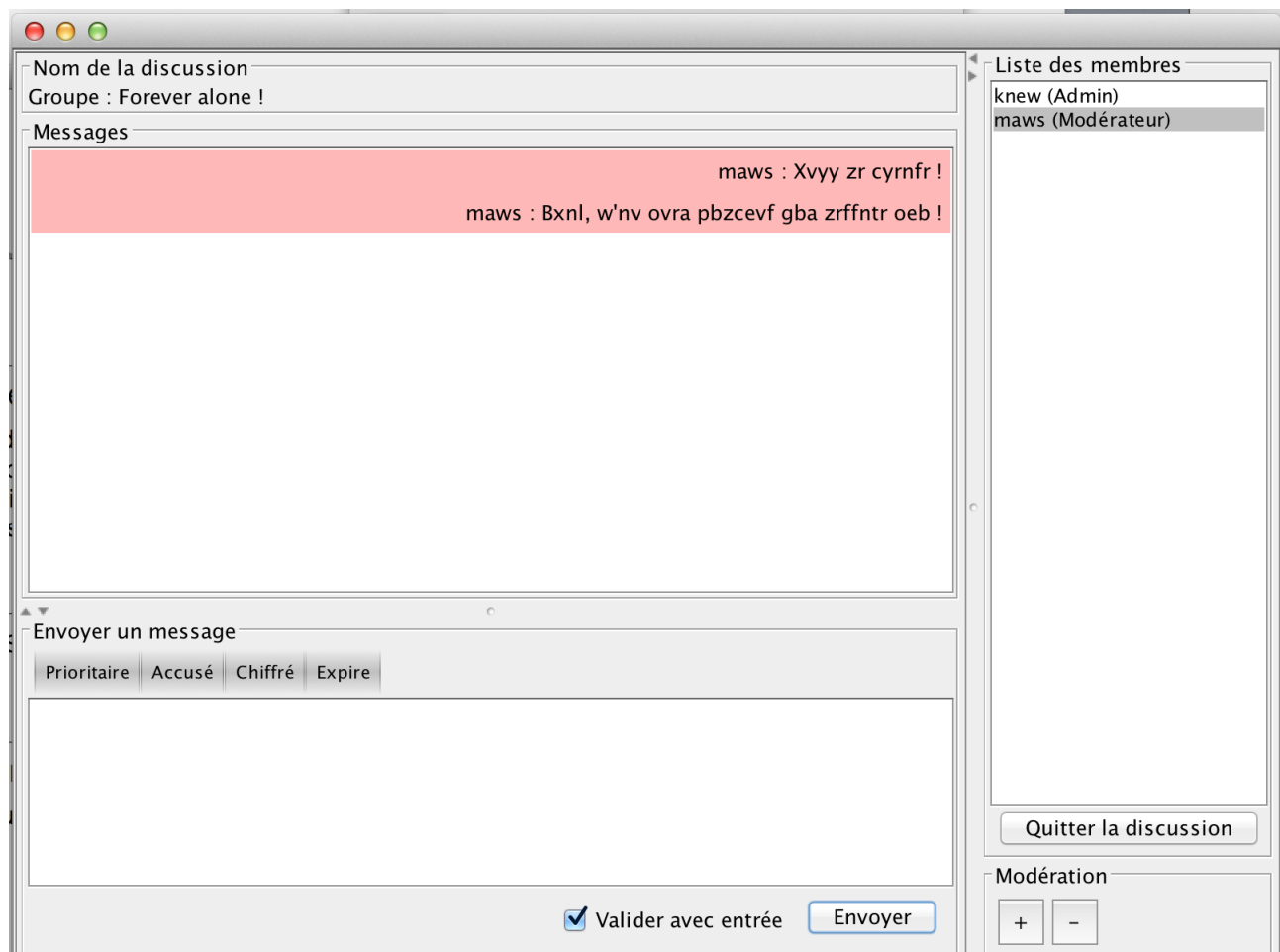
Comme demandé, on ne peut quitter un groupe que s'il n'y a plus de personne dedans lorsque l'on en est le modérateur. Lorsque l'on peut enfin quitter le groupe, il est supprimé.



## Envoi de message dans un discussion (Privée ou en groupe)

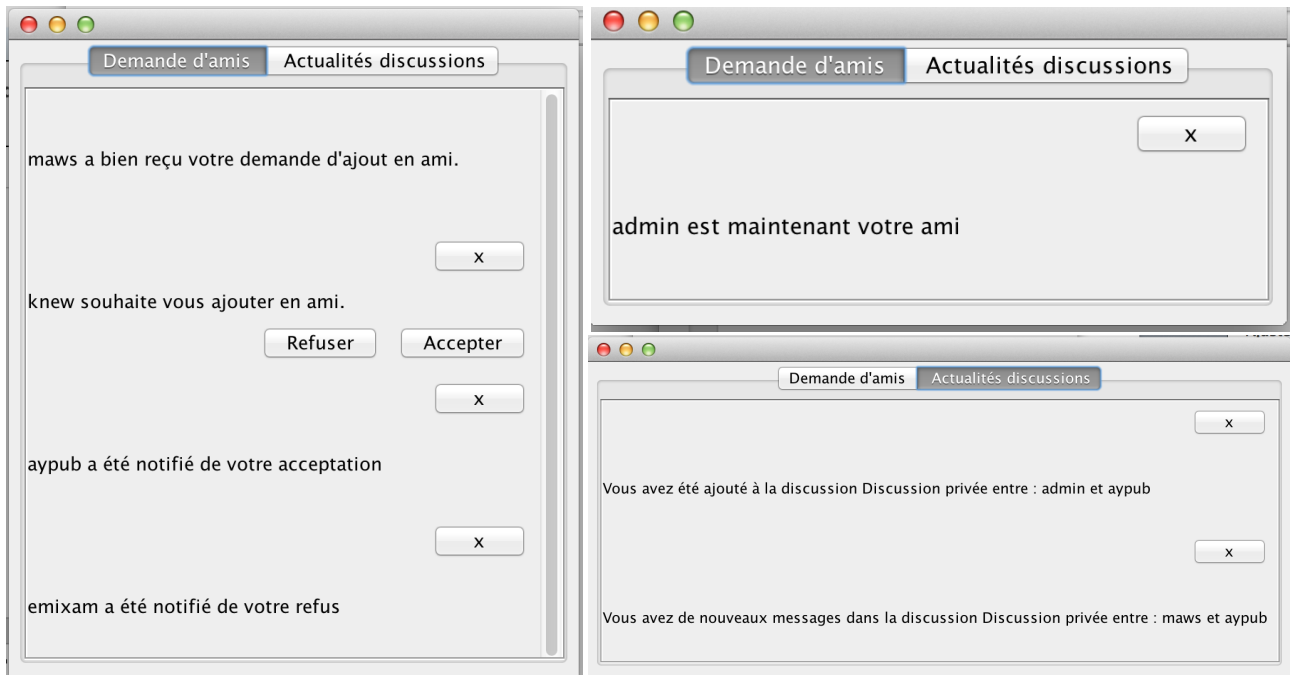
Il est possible de poster plusieurs type de messages : Prioritaire (en rouge), avec accusé de réception ("Lu" lorsqu'il est lu par les destinataires) , chiffré et avec une expiration (de 2 jours par défaut). Pour cela, il suffit de cocher les boutons associés aux options au dessus de la zone de saisie.

Lorsqu'un message est envoyé, une notification est envoyé à tous les utilisateurs.



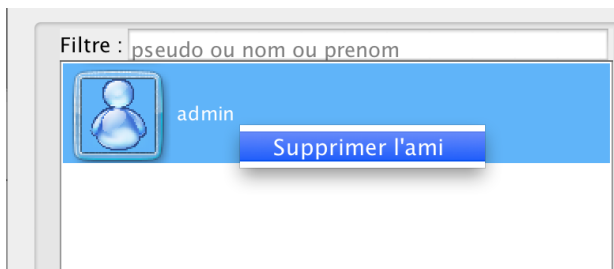
## Gestion des notifications

Un panneau de notification est accessible grâce au bouton “notifications” du panneau d’accueil. Il permet de suivre les fils d’actualités des discussions ainsi que l’état des demande d’ajout en ami.



## Suppression d'un ami

Cliquez droit sur l'ami et supprimez le !



## Axes d'amélioration

- Ajouter la gestion des avatar. Il suffit de demander à l'utilisateur de sélectionner une image. A ce moment on la copie dans un dossier dont on va stocker l'url en base.
- Utiliser le design pattern Decorator pour les message au moment de la création du message dans le service. Comment je ne gère pas encore l'héritage dans mon Framework, je n'ai pas pu l'intégrer.
- Améliorer l'interface graphique avec du material design !
- Ajouter la possibilité de supprimer les notifications. Pour la table Amitié, il faudrait rajouter une colonne pour connaître l'état de la demande du côté du demandeur et utiliser la colonne existante pour connaître l'état du côté du sollicité.
- Déplacer certains bouts de code qui ne sont pas à leur place comme ceux dans les fonctions valider() des fenêtres qui devraient être dans la couche service.
- Ne pas permettre que la session admin puisse modifier son rôle car sinon on peut se retrouver sans admin au final !
- Gérer les notifications en base plutôt que de les recréer à chaque fois dans la vue. Ceci est du au fait que cela a été la dernière chose que j'ai faites et je n'ai pas eu le temps de la peaufiner. Le gestionnaire de notification aurait permis d'illuminer le bouton de notifications lorsque l'on a de nouvelle notifications.
- Nettoyer et commenter le code.
- Corriger certains bug restant ?

## Problèmes rencontrés

- Afin de pouvoir gérer les amis, j'ai créé une vue projet\_amitie\_view. Sauf que comme une vue ne peut pas être mise à jour avec UPDATE j'ai du créé des annotations spécifiques à mes vue. Cela n'a pas été facile de gérer ce cas car j'ai essayé au maximum de rester générique dans mon DAO..
- Le Framework développé pour cette application était peut-être un peu gros pour une appui de cette envergure. J'ai eu beaucoup de cas qui se sont rajouté au fur et à mesure que j'avais ce qui m'a beaucoup ralenti. Cependant, j'ai fait mon maximum pour rester le plus générique possible afin de pouvoir réutiliser ce framework si hibernante n'est pas autorisé.