Projet IHM

Rapport

Clément MARTINS
Mathis CHAIGNEAU
Ismail HANI

Table des matières

1) Explications Techniques

Pour ce projet nous avons essayé de répartir les Classes le plus clairement possible tout en essayant de respecter le modèle MVC .

a) Fonctionnement des classes

Au Lancement de l'application la classe IHM (le Main) nous affiche une fenêtre avec la sélection de l'utilisateur et son mot de passe :

-ROOT : « root »

-PROF : « prof »

-ETUDIANT : « etuiutsen »

Si le mot de passe est bon il nous demande quel est le modèle souhaité :

-PERSISTANT : les Factory charge les données dans la BD lors de l'initialisation

-NON PERSISTANT : On créer les Factory et lui donnons des données.

Enfin il instancie le Model Correspondant. Le model crée la fenêtre et la répartie en 2 .

Il crée ou récupère les Factory dépendamment de son modèle. Il affiche alors le premier Groupe de la Promo ou rien s'il n'existe pas encore. Il connaît la fenêtre contrôlant l'affichage des Groupes : FenetreGroupe, celle des étudiants : FenetreEtudiant, ainsi que peut-être celle des changements s'il est root : FenetreChangement.

Ces fenêtres connaissent un Jpanel avec les données à afficher et instancier aussi les listeners des Boutons. Elles connaissent le Model et les Liteners également.

Les listeners lors du clic affichent parfois des informations supplémentaires, des messages d'erreur en fonction de leur fonction. Lors de Modifications, elle appelle le Model pour les Réaliser.

En Modèle Persistant :

La Connexion à la Base de données se fait via les Factory qui lors de leur Fonction modifient la Base de Donnée pour correspondre à l'action. La connexion se fait à chaque requête pour pouvoir gérer les erreurs de connexion et afficher à l'utilisateur une fenêtre modale pour lui demander de se reconnecter ou non (fermera l'application).

Les factory mettent à jour leurs informations concernées à chaque demande ou modification (seulement l'information concernée, cela prendrait trop de temps de tout mettre à jour). Cela permet une utilisation en temps réel de l'application avec plusieurs instances sans avoir d'erreur (ex :root modifiant un groupe qu'un autre root a supprimé sans le savoir).

b) Fonctionnement de la Base de données

La Base de données est très simple :

-Table Groupe(id, nom, min, max, Type, id-parent)

-Table Etudiant(id, nom, prenom)

-Table Contient(idGroupe, idEt)

-Table Changement(id, idGroupeA, idGroupeB, idEtudiant, Raison)

c) Modifications de l'API

Nous avons modifié plusieurs fois l'API :

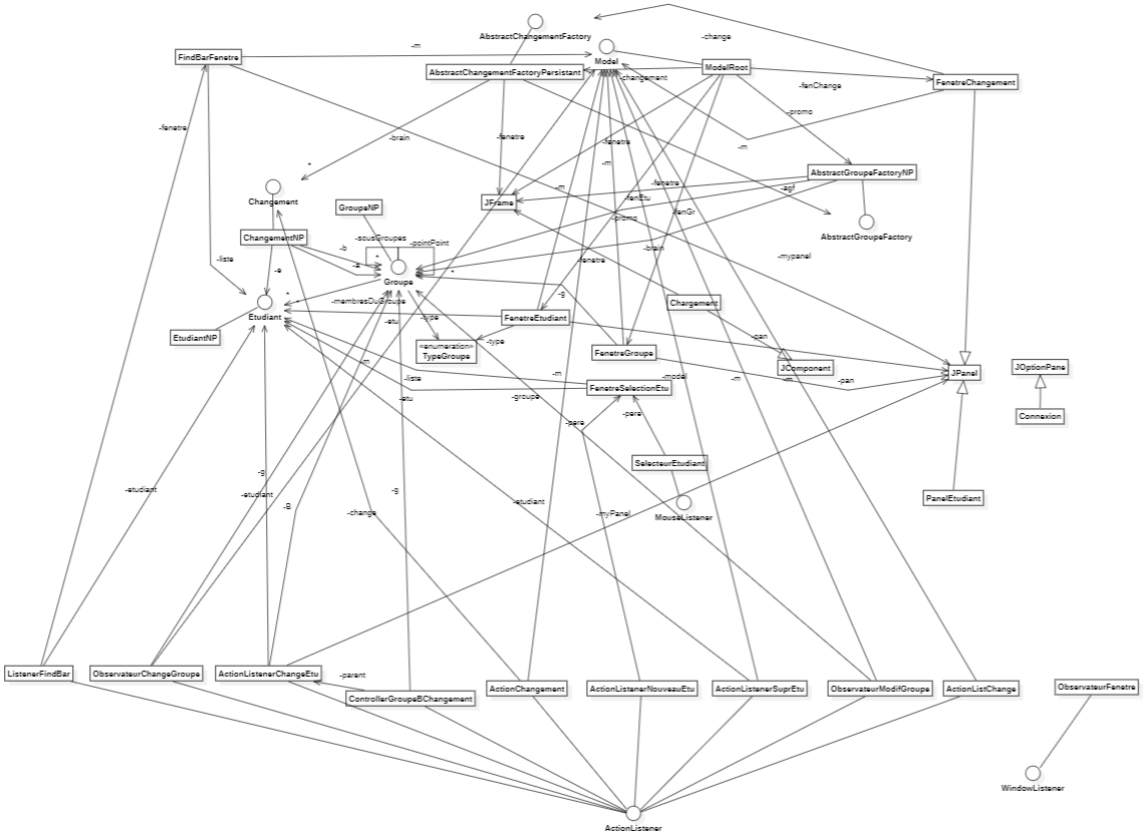
- Groupe : Ajouter une méthode setName, Pour modifier facilement son nom (pour éviter d'avoir à créer une copie de lui-même en changeant son nom pour changer de nom) ;
- AbstracGroupeFactory :
  - ➔ Ajouter une méthode setGroupeName pour changer ce nom en passant par la factory,
  - ➔ Ajouter une méthode refrreshAll qui demande à la factory de mettre à jour le groupe en argument (obligatoire pour pouvoir avoir plusieurs instances) ;
- Changement : Ajouter une méthode getRaison, retourne la raison du changement (null en Type1, rempli en Type2) ;
- AbsractChangementFactory : createChagement avec une raison maintenant.

2) Diagrammes

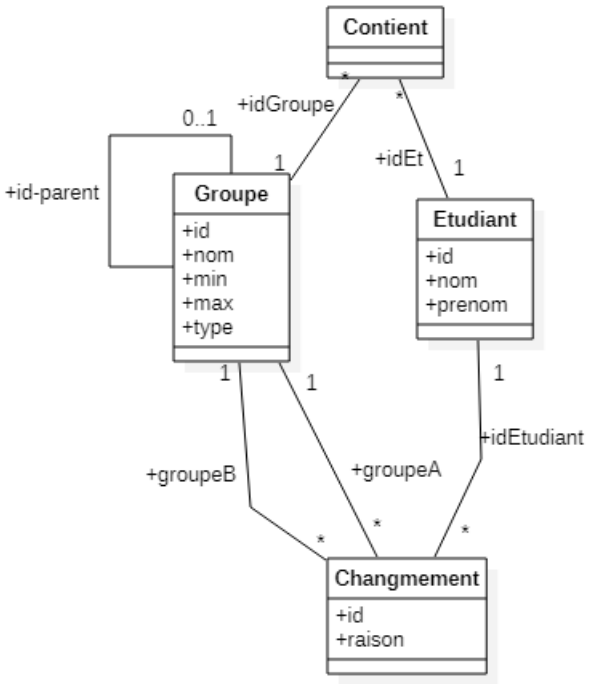
a) Diagramme de classes du programme

Voici le diagramme de classes de la partie persistante du programme (MP).

Nb : Nous avons conscience de la faible qualité de l'image qui suit. Ce pendant une image au format SVG était impossible sans les copyrights. C'est pourquoi nous avons ajouté au dépôt git la sauvegarde StarUML ainsi que l'image au format SVG du diagramme de classes.



b) Diagramme de la base de données



3) Fonctionnement

a) Cas de test

1) Se connecter en tant que ROOT, créer un nouveau groupe « tp1 », 5 étudiant minimum, 15 étudiants maximum. Ajouter 3 étudiants dans ce groupe. Un message apparait « nombre d'étudiant trop petit».

2) Se connecter en tant que ROOT, appuyer sur add, nouveau, entrer un nom et un prénom, valider puis. Appuyer sur « find etu » chercher l'étudiant en indiquant le nom et le prénom.

3) Se connecter en tant que Etudiant, aller dans son groupe de TD, appuyer sur « changer de groupe », choisir le groupe de TD que l'on veut rejoindre.

- ➔ « impossible, trop d'étudiant dans l'autre groupe »
- ➔ La demande a bien été prise en compte.

4) Se connecter en tant que PROF, aller dans un sous-groupe de la Promotion. Lancer une deuxième instance de l'application et se connecter en tant que ROOT. Supprimer le Groupe dans lequel le PROF est. Observer les changements sur l'instance du PROF : faire un refresh ou essayer de changer dans un sous-groupe de ce groupe → le PROF sera ramener au groupe parent encore existant le plus proche.

b) Fonctionnalités demandées et fonctionnalités réalisées

Voici la liste des fonctionnalités demandées dans le sujet du projet. Un ✓ indique que la fonctionnalité a été réalisée.

Déplacer un individu en validant sa demande ✓

Refus d'une demande ✓

Demander à passer dans un groupe qui est de même taille ou plus grand en ajoutant une explication Pour l'administrateur ✓

Voir les demandes de changement de groupe du type 2 ✓

Lister tous les Groupes✓

Rechercher les Groupes d'un étudiant avec les premières lettres de son nom✓

Échanger des individus entre 2 groupes à leur demande respectives ✓

Refus d'une demande ✓

Généralisation de l'échange mais avec des cycles de taille plus grande que 2✓

Déplacer un individu dans un autre groupe✓

Demander à passer dans un groupe qui est moins plein que le sien en ajoutant une explication ✓

Modèle persistant de données✓

Chercher le groupe d'un étudiant à partir des premières lettres de son nom✓

Ajouter un individu dans un groupe✓

Sauvegarde des modification✓

Fabriquer automatiquement une partition des étudiants en plusieurs groupes✓

Afficher la liste des groupes✓

Afficher la liste des étudiant d'un groupe donné✓

Créer, supprimer et renommer un groupe✓

Les groupes forment une arborescence. Tout en haut toute la promo, un groupe indestructible ensuite on peut soit ajouter des groupes qui ne forment pas une partition, soit un groupe qui forme une partition✓

Adapter toutes les opérations à cette arborescence complexe. ✓

Implémenter un modèle de données persistants. ✓

4) Conclusion

a) Améliorations possibles

Avec Plus de temps nous aurions aimé améliorer l'aspect graphique.

Nous aurions également aimé créer des triggers dans la base de données qui enregistrent chaque changement pour, lors du refresh, au lieu de refresh le groupe bêtement (ce qui en fonction de la connexion peut être long et vite énervant), regarder s'il y a eu de nouveaux changements et si oui les effectuer.

Pour finir nous aurions aimé mieux répartir les classes ainsi que diminuer la récurrence de code (par exemple au lieu de l'interface Model en faire une classe abstraite pour que tous les modèles n'aient pas à définir les méthodes qui sont souvent les mêmes).

b) Ressenti global sur le travail effectué

Globalement nous sommes très satisfaits du travail effectué qui répond à toutes les attentes demandées. Malgré l'aspect graphique simple il reste du moins instinctif et donc répond aussi aux attentes d'une interface Homme machine.