Rapport itération 4

Groupe 10

Factorisation du code

Modèle

Une class abstraite nommée AbstractObject a été créé. Cette classe est mère de toutes les autres classes du modèle. Elle redéfinit des méthodes couramment utilisées dans nos objets. Cela évite de devoir override dans toute les classes.

Une classe nommée NamedObject été créé. Toutes les classes d'objets ayant un nom héritent de cette classe afin d'éviter une duplication du code.

Les classes "ShoppingList" et "Recipe" avaient beaucoup de méthodes en commun. En effet, dans les deux cas il s'agit d'objets qui vont contenir une série de produits associés à une quantité. Une super classe appelée "ProductsQuantity" a donc été créée. Cette classe contient toutes les méthodes permettant de gérer les produits présents dans Recipe ou ShoppingList. Stock hérite également de cette classe pour associer le nombre de produits présents dans un Stock.

Contrôleur

Après avoir créé une superclasse pour recette et liste de courses, nous avons remarqué que les contrôleurs "ShoppingListController" et "RecipeController" contenaient également du code dupliqué. Une super classe abstraite, AbstractProductController a donc été créée. Celle-ci s'occupe principalement de la gestion de plusieurs ProductsQuantity (ainsi que de la gestion d'un ProductsQuantity quand celui-ci est sélectionné). Ajout de produit, suppression, ...

Ajout de “sel” pour les mots de passe

Lors des précédentes itérations, le hash du mot de passe était stocké tel quel dans la base de données. Cela signifie que deux utilisateurs avec le même mot de passe auront la même valeur de hash sauvegardée. Des tables de hachage précalculées pouvaient également être utilisées afin de retrouver un mot de passe courant à partir du haché. Cela nous a semblé une priorité du point de vue de la sécurité.

Nous avons alors convenu avec le client de rajouter du “sel” au mot de passe afin de résoudre ce problème. Au moment de la création d’un utilisateur, une chaîne de caractères aléatoires “sel” est créée et sauvegardée dans la base de données. Le sel est ensuite concaténé au mot de passe avant de la hacher. De cette manière, les chaînes de caractères hachées sont moins facilement calculables à l’avance et deux utilisateurs ayant le même mot de passe auront deux hachés différents.

Histoire 3

Suite à la discussion avec le client, il a été convenu de compléter l’histoire n°3 (création d’une recette). Les tâches à effectuer pour cette itération étaient :

- convertir une recette en liste de course ;
- permettre à un utilisateur d’avoir plusieurs listes de course.

N’ayant pas assez de temps disponible, l’adaptation de la recette à un nombre de personnes différent a été abandonnée.

Notre architecture nous a permis d'implémenter assez facilement ces nouvelles fonctionnalités. Pour la conversion des recettes en liste de course, nous avons simplement ajouté une méthode dans l’objet recette qui complète une crée une liste de course minimale pour couvrir les ingrédients de la recette. Pour permettre à un utilisateur d’avoir plusieurs listes de course, l’attribut liste de courses dans l'utilisateur a été changé en une liste pouvant contenir plusieurs listes. Pour la base de données JPA s’occupe du reste.

Implémentation de DAO

Des objets de type DAO ont été utilisés afin de faciliter les interactions des objets avec la base de données. Ainsi pour chaque objet qu'il est possible de sauver/récupérer dans/depuis la base de données, un ObjectDAO a été créé. Ainsi, le contrôleur ne fait jamais appel directement ni à un objet du modèle ni à la base de données, mais uniquement à un DAO.

Base de données

La base de données en elle-même est gérée par Java Persistence Api (JPA). Le point positif de JPA, c'est que cela fait le boulot tout seul (pas besoin de trop configurer la sauvegarde d'un objet). En même temps, le point négatif de JPA, c'est que cela fait le boulot tout seul (quand une erreur survient ou que certains champs ne sont pas sauvé/sont dupliqués, on ne sait pas pourquoi).

Outils de gestion d'équipe

Pendant ce projet, nous avons tout d'abord utilisé Trello afin d'organiser notre travail. Trello était organisé en plusieurs colonnes : *todo, doing*, *merged request* et *done*. La colonne *todo* permettait de voir les tâches qu'il restait à effectuer. La colonne *doing* permettait de voir les tâches qui étaient en cours afin de ne pas travailler sur la même tâche à plusieurs équipes. La colonne *merge request* permettait était notre verrou. Une seule merge request à la foi pouvait être proposée afin de ne pas casser le master. La colonne *done* servait d'archivage.

Lien vers Trello https://trello.com/b/xf3lodAn/gl

L'assistant nous a montré les issues GitLab lors de la première réunion coach. Nous avons petit à petit remarqué que GitLab permettait plus de choses (citation de commit/merge request/fix d'issue dans un commit) et avons commencé notre migration vers GitLab.

Pair programming

Quand nous avons pu, nous avons réalisé du pair programming. Cependant, les horaires de tout le monde ne pouvant pas toujours concorder, cela n'a pas toujours été possible. Lorsque cela n'était pas possible, nous avons instauré du remote pair programming via un partage d'écran (partage TeamViewer et appel Facebook). En plus de cela, afin que le code soit testé et revu avant d'atteindre le master, il a été interdit à chacun de merger son propre code sur le master. Après avoir terminé une tâche, l'équipe impliquée devait créer une merge request et demander à autre équipe de revoir et tester le code nouvellement créé. Cette technique a permis de relever pas mal d'erreurs qui s'étaient glissées malgré le pair programming.

Améliorations possibles

- Utilisation d’EventBus pour l’échange de données entre les contrôleurs
- Utilisation d'observateurs pour mettre à jour l’interface
- Ne pas autoriser l’ajout de produit avec une quantité 0 dans recette et liste de course