Yasmine Saifi
Alexandre Léonardi

Plan de tests

Yasmine Saifi
Alexandre Léonardi

Table of Contents

Yasmine Saifi
Alexandre Léonardi

Introduction

Les tests de ce projet vont se découper de manière très classique en deux grandes parties : tests unitaires et tests d'intégration à la fin de chaque lot (ceux-ci représentant des grands pans de fonctionnalité de notre programme).

Le test d'intégration du troisième lot sera fusionné avec un test général de toutes les fonctionnalités de l'application, qui fera office de test d'application avant le rendu définitif.

Les techniques mises en place pour les tests unitaires seront dans un premier temps JUnit et spring-test pour l'interfaçage avec Spring et l'utilisation des fonctions associées telles qu'*autowired*. Si le temps le permet, il est envisagé d'intégrer DbUnit pour améliorer la qualité des tests sur la base de données.

Enfin, l'outil EMMA sera utilisé en parallèle pour avoir une indication de la qualité de la couverture de code par nos tests unitaires.

Yasmine Saifi
Alexandre Léonardi

Tests unitaires

Chaque méthode de la classe de DAO que nous avons créée doit être testée en fonction de ses domaines de validité. Les domaines valides et invalides sont ici détaillés, accompagnés d'un ou plusieurs tests permettant de vérifier que les résultats sont conformes aux attentes.

Il est à noter que les tests supposent pour leur bon fonctionnement que l'utilisateur a correctement initialisé la base annuaire conformément au script mySQL fourni.

Par ailleurs *dao* est le nom donné à l'instance d'une implémentation d'IPersonDao utilisée pour les tests (en pratique, il s'agit d'une instance de GroupDao, mais cet aspect est géré par Spring et n'apparaît pas explicitement).

Enfin, les classes Group et Group surchargent *equals* et rendent possible l'utilisation de *assertEquals* (deux instances de ces classes sont égales si et seulement si elles ont le même id).

1) Interface IPersonDao

```
public interface IPersonDao {
	Collection<Group> findAllGroups();
	Collection<Person> findAllPersons(long groupId);
	Group findGroup(long id) throws GroupDoesNotExistException;
	Person findPerson(long id) throws PersonDoesNotExistException;
	void savePerson ( Person p) throws InvalidPersonException;
	void saveGroup(Group g) throws InvalidGroupException;
	void clearTables();
	void clearPersons();
	boolean groupExists(long id);
	boolean personExists(long id);
	void deletePerson (long id) throws PersonDoesNotExistException;
}
```

Yasmine Saifi
Alexandre Léonardi

2) Classes de test

findAllGroups

Table GROUPS vide	OK	Retourne un ensemble vide	assertTrue(dao.findAllGroups().isEmpty())
Table GROUPS non-vide	OK	Retourne un ensemble de groupes	assertTrue(expected,dao.findAllGroups()) //expected contient les groupes qui ont été ajoutés à la base

findAllPerson

Table PERSONS vide	OK	Retourne un ensemble vide	assertTrue(dao.findAll`Person`().isEmpty())
Table PERSONS non-vide	OK	Retourne un ensemble de personnes	assertTrue(expected,dao.findAll`Persons`()) //expected contient les personnes qui ont été ajoutées à la base

findGroup(groupId)

Un groupe avec cet id existe	OK	Retourne le groupe	dao.saveGroup(g) assertEquals(g,dao.findGroup(g.getGroupId())
Aucun groupe n'existe avec cet id	KO	GroupDoesNotExistException	dao.findGroup(-1)

findPerson(personId)

Une personne avec cet id existe	OK	Retourne la personne	dao.save`Person`p) assertEquals(p,dao.findPerson(p.getPersonId())
Aucune personne n'existe avec cet id	KO	`Person`DoesNotExistException	dao.findPerson(-1)

Yasmine Saifi
Alexandre Léonardi

savePerson(person)

Personne non présente en base	OK	Personne ajoutée	assertFalse(dao.personExists(p.getPersonId())) dao.savePerson(p) assertTrue(dao.personExists(p.getPersonId()))

Personne présente en base	OK	Personne modifiée	//modif de p
			assertNotEquals(p,dao.findPerson(p))
			dao.savePerson(p)
			assertEquals(p,dao.findPerson(p.getPersonId()))

Personne non-valide	KO	InvalidPersonException	//p a un id invalide (e.g. déjà utilisé)
			dao.savePerson(p)

saveGroup(group)

Groupe non présent en base	OK	Groupe ajouté	assertFalse(dao.groupExists(g.getGroupId())) dao.saveGroup(g) assertTrue(dao.groupExists(g.getGroupId()))

Groupe présent en base	OK	Groupe modifié	//modif de g
			assertNotEquals(g,dao.findGroup(g.getGroupId()))
			dao.saveGroup(g)
			assertEquals(g,dao.findGroup(g.getGroupId()))

Groupe non-valide	KO	InvalidGroupException	//g a un id invalide (e.g. déjà utilisé)
			dao.saveGroup(g)

clearTables

/	OK	Tables vidées	Dao.saveGroup(g) dao.savePerson(p) dao.clearTables assertFalse(dao.existsPerson(p.getPersonId())) assertFalse(dao.existsGroup(g.getGroupId()))

Yasmine Saifi
Alexandre Léonardi

clearPersons

/	OK	PERSONS vidée	Dao.saveGroup(g) dao.savePerson(p) dao.clearTables assertFalse(dao.existsPerson(p.getPersonId()))

personExists(personId)

Une personne avec cet id existe	OK	Retourne true	Dao.savePerson(p) assertTrue(Dao.personExists(p.getPersonId())))
Aucune personne avec cet id n'existe	OK	Retourne false	assertFalse(dao.personExists(p.getPersonId())))

groupExists(groupId)

Un groupe avec cet id existe	OK	Retourne true	Dao.saveGroup(g) assertTrue(Dao.groupExists(g.getGroupId()))
Aucun groupe avec cet id n'existe	OK	Retourne false	assertFalse(dao.groupExists(g.getGroupId()))

deletePerson(personId)

Une personne avec cet id existe	OK	Personne supprimée	Dao.savePerson(p) dao.deletePerson(p.getPersonId()) assertFalse(dao.personExists(p))
Aucune personne avec cet id n'existe	KO	PersonDoesNotExistException	Dao.deletePerson(p)

3) Couverture de code

Nous avons actuellement une couverture de code de 80%, ce qui est encore insuffisant : le plan de test que nous avons établi n'est probablement pas assez complet et va donc devoir être remanié pour le lot 2.

On remarque néanmoins qu'un certain nombre des lignes non-couvertes sont des exceptions SQL qui seraient produites seulement

si la base n'était pas correctement initialisée ou encore si les requêtes étaient erronnées.

Yasmine Saifi
Alexandre Léonardi

Tests d’intégration

1) Lot n°1

Le test d’intégration dans ce cas a été relativement rapide : il s’agit de tester chacune des fonctions, vérifiant directement en base en ligne de commande que les modifications étaient correctes.

Le lot n°1 étant majoraitairement du backend, et partant de rien, les tests unitaires sont ici plus important que les tests d’intégration.