Yasmine Saifi
Alexandre Léonardi

RAPPORT

Yasmine Saifi
Alexandre Léonardi

Table of Contents

Yasmine Saifi
Alexandre Léonardi

Introduction

L'objectif de ce projet, dont les détails sont plus amplement décrits dans le cahier des charges, était de fournir une interface web permettant de gérer une base de données simples, contenant des personnes et des groupes, chaque personne faisant partie d'un groupe.

Nous avons repris les points importants des cours de JEE et de Fiabilité I pour mener à bien le projet : d'une part, utilisation des différentes techniques vues en cours ou apprises à l'aide de ressources en ligne, et d'autre part l'application d'un plan de test au fur et à mesure du développement de la partie back-end.

Dans ce rapport, nous allons présenter les fonctionnalités particulières de Spring ou de Java EE que nous avons mises à contribution, ainsi que les problèmes rencontrés au cours du développement. Enfin, nous expliquerons rapidement comment nous avons procédé par les tests, et les limitations qui subsistent dans le plan de test que nous avons appliqué.

Yasmine Saifi
Alexandre Léonardi

I-Choix d'implémentation

Un certain nombre de points vus en cours et que nous avons mis en pratique dans le projet. La liste n'est pas exhaustive mais présente les fonctionnalités les plus importantes, intéressantes ou juste pratiques, indépendamment de la complexité que présente leur mise en œuvre.

1) Tags

C'est peut-être la fonctionnalité non-obligatoire que nous avons le plus utilisée, les annotations de Spring facilitent un grand nombre de fonctionnalités et surtout la rédaction du fichier de configuration *spring.xml*.

Notamment, les tags de la famille `@Component` permettent de ne pas spécifier manuellement les beans que Spring doit gérer, ce qui fait que la partie du fichier de configuration concernée est compacte et se résume à :

```
<context:component-scan base-package="fr.projet.jee.dao" />
<context:component-scan base-package="fr.projet.jee.utils" />
```

Nous avons aussi largement utilisé l'injection de dépendance avec `@Autowired` mais cela est quasiment obligatoire dans un contexte JEE avec Spring.

2) Properties

Les fichiers *.properties* permettent, dans Spring, l'externalisation des constantes utilisées dans le programme. Un fichier *.properties* regroupe donc une ou plusieurs variables qui, sans ça, seraient par exemple des variables statiques. Typiquement, on crée un *.properties* pour un package.

Bien que dans notre cas il se soit principalement agi d'un exercice pour apprendre la méthode associée, dans un projet de plus grande envergure les intérêts sont multiples, principalement :

- La pratique incite à utiliser des noms parlants pour les constantes plutôt que d'inscrire leur valeur « en dur »
- Comme la valeur de la constante n'est saisie qu'une seule fois et ensuite partagée par tout le package, sa modification est facilitée (un seul endroit à modifier, il n'y a pas à traquer les occurrences de notre constante)
- Dans le cas de chaînes de caractère, le portage dans une autre langue est facilité car il suffit de traduire les *.properties* correspondant aux IHM

Dans notre cas, cela a été utilisé dans le package utils pour externaliser les noms de groupes et de personnes possibles ainsi que, de manière très sécurisée, le mot de passe commun que toutes les personnes de la base utilisent.

De façon plus technique il faut dans un premier temps indiquer l'emplacement du fichier dans le *spring.xml* :

```
<context:property-placeholder location="classpath:fr/projet/jee/utils/utils.properties"/>
```

Puis de récupérer les valeurs des différentes constantes contenues dans le *.properties* à l'aide du tag `@Value`, comme par exemple :

```
@Value("${names}")
private String[] names;
```

3) Try-with-resources

Le try-with-resources est une modification du try-catch auquel est associé une ressource qui sera automatiquement fermée à la fin du bloc try. Pour que cela soit possible, la ressource doit implémenter une interface précise, `java.lang.AutoClosable`.

Encore une fois les avantages sont multiples :

- Possibilité de faire un multi-catch pour raccourcir la gestion des exceptions si le traitement est le même
- Code plus concis et avec une meilleure lisibilité
- Plus de risque d'oubli de fermer une ressource, qui emmènerait à un bug ultérieur

Le try-with-resources est indépendant de Spring et fait partie de Java depuis la version 7. Nous l'avons majoritairement utilisé pour la partie DAO, où chaque fonction impliquait d'établir une connexion à la base de données, puis de la fermer.

Yasmine Saifi
Alexandre Léonardi

II-Problèmes rencontrés

Encore une fois, il ne s'agit pas d'une liste exhaustive mais des problèmes que nous avons rencontrés et qui nous ont semblé les plus intéressants.

1) Tester les exceptions

Il ne s'agit pas vraiment d'une difficulté de programmation, mais plutôt d'une incompréhension de notre part quant à la manière dont les tests unitaires portant sur les exceptions fonctionnent.

Il y a plusieurs façon de réaliser un TU sur une exception, celle que nous avons utilisée consiste à déclarer une variable permettant de tester une exception :

```
@Rule
public ExpectedException thrown = ExpectedException.none();
```

Puis ensuite, dans la fonction de test concernée, déclarer l'attention d'une exception précise :

```
thrown.expect(PersonDoesNotExistException.class);
```

Il ne reste qu'à écrire une instruction produisant l'exception mentionnée pour que le test soit validé. La situation s'est compliquée pour nous quand ce genre d'instruction a lieu dans un try-catch : nous pensions que le thrown.expect aurait la priorité mais c'est en fait le contraire, si le catch peut attraper l'exception, il le fera, et de ce fait le test échouera car il n'aura pas vu l'exception attendue.

L'objectif est donc que la méthode de test renvoie l'exception, et non pas que l'exception soit levée et traitée à l'intérieur de cette fonction. C'est donc un problème qui a été assez simple à régler une fois compris, mais qui nous a occupé pendant un bon moment à essayer de saisir le fonctionnement de JUnit sur le sujet.

2) Utiliser l'injection de dépendance dans le servlet

Le servlet doit, pour les besoins du projet, peupler la base de données d'un certain nombre de personnes. Cela pourrait être fait de plusieurs façon différentes, notamment à l'aide des méthodes de DAO que nous avions justement rédigées.

Cela a été, cependant, moins simple que nous ne l'avions tout d'abord pensé, car les servlets ne sont pas gérés par Spring par défaut. Il fallait donc indiquer manuellement à la classe de servlet de chercher parmi les beans gérés par Spring ceux que nous voulions injecter.

Nous avons longuement tenté de mettre cette solution en pratique, car elle nous semblait la plus adaptée au contexte, mais finalement et par manque de temps nous l'avons abandonnée au profit d'une autre plus simple : peupler la base à l'aide d'un script pré-écrit.

La classe qui devait gérer cette partie est ceci dit toujours présente dans le code source de l'application : Populate. Les différentes approches pour pouvoir injecter cette classe vues sur le net consistaient souvent à utiliser `SpringBeanAutowiringSupport`, par exemple de la manière suivante :

```
public void init(ServletConfig config) {
    super.init(config);
    SpringBeanAutowiringSupport.processInjectionBasedOnServletContext(this,
      config.getServletContext());
  }
```

Mais chaque solution entraînait de nouveaux problèmes qu'il fallait régler, raison pour laquelle nous avons finalement préféré la solution du script.

Yasmine Saifi
Alexandre Léonardi

III-À propos des tests

1) Couverture de tests

La couverture de tests est, sur le projet, de 73%. Bien que ce score puisse sembler peu élevé, nous l'avons jugé acceptable compte tenu de limitations en termes de tests du projet. Tout d'abord, cette couverture prend en compte des portions du code qui ne sont pas pertinente comme la partie servlet que nous n'avons pas testée.

Ensuite, même au sein des packages que nous avons testés, il y avait des portions de code impossibles à tester (e.g. générer une SQLException dans la DAO).

Les pourcentages dans les packages les plus pertinents sont donc comme suit :

- Dao : 76.2%
- Beans : 87.9%
- Utils : 97.6%

Une moyenne de ces trois parties nous donne une couverture de 87.2% : si l'on excepte les exceptions qu'il était difficile de reproduire comme mentionné plus haut, l'intégralité du code a été couvert, y compris les différents embranchements en cas de test de condition.

2) Tests d'intégration

Les tests d'intégration ont consisté, majoritairement, en une utilisation du logiciel en essayant de se mettre dans l'état d'esprit d'un utilisateur lambda. Il s'agissait de tester les fonctionnalités interdites (modifier les informations de quelqu'un sans être connecté/en étant connecté comme quelqu'un d'autre), ou encore de rentrer des informations erronées.

Par ailleurs, nous avons testé tous les cheminements qu'un utilisateur peut prendre dans l'interface, c'est-à-dire tous les endroits où il était possible de cliquer.

Enfin nous avons vérifié *via* l'interface en ligne de commande de MySQL que les modifications que nous faisions se répercutaient bien en base.

Il est à noter que nous n'avons pas réussi à résoudre tous les problèmes rencontrés, notamment quant à la connexion d'un utilisateur et à la modification des informations.

Yasmine Saifi
Alexandre Léonardi

Conclusion

Pour clore ce rapport, il est bon de noter que ce projet nous a apporté beaucoup, non seulement en étant un prétexte pour réviser et mettre en pratique les notions vues en cours, comme c'est le cas avec tous les projets universitaires, mais aussi en cela qu'il est ouvert et large. Il nous a donc confrontés à des situations auxquelles nous ne nous attentions pas à la lecture du sujet, comme l'externalisation des constantes ou le peuplement de la base de données.

Du point de vue de la fiabilité, il y a des raffinements que nous aurions souhaité réaliser et que nous n'avons pas eu le temps de faire, comme chercher des APIs spécialisées dans les tests impliquant des bases de données et qui nous auraient permis de faciliter les vérifications consécutives à une modification de la base.

Malgré cela le constat est le même que pour la partie JEE : bien qu'il ne s'agisse encore que d'un projet universitaire, il a été l'occasion d'une part de réviser le programme vu en cours et d'autre part d'improviser et de se projeter dans ce à quoi pourrait ressembler un véritable projet industriel.