  **Département de génie logiciel et des TI**

**LOG720 –Architecture distribuée orientée objet**

**Automne 2016**

LOG720 – Laboratoire #03  
Réponses aux questions d’analyses

|  |  |
| --- | --- |
| **Numéro de laboratoire** | 03 |
| **Étudiant (s)** | Khemmanivong, Anousonne Mustapha, Zaher |
| **Code (s) Permanent(s)** | KHEA24079004  MUSZ01029104 |
| **Numéro d'équipe** | 10 |
| **Session** | Automne 2016 |
| **Groupe** | 01 |
| **Responsable du cours** | Bernard, Michaël |
| **Chargé de laboratoire** | Redouane, Billel |
| **Date de remise** | Mercredi 14 décembre 2016 |

## Introduction

Dans le cadre du troisième laboratoire du cours d’architecture distribuée orientée objet (LOG720), nous avons comme tâche principale de reprendre le système développé dans le cadre du deuxième laboratoire et d’y intégrer des « *frameworks* ». L’intégration de ces « *frameworks* » à notre système aura pour but d’améliorer l’architecture de notre application afin de la rendre conforme aux règles de l’art et aux bonnes pratiques.

Concevoir et développer une application respectant les règles de l’art a plusieurs avantages. L’un de ces avantages est la simplification de la maintenabilité. Par exemple, une application conçue selon le patron *MVC* (Modèle Vue Contrôleur*)* sera plus facile à maintenir, car il permet de séparer les différentes fonctionnalités selon les trois domaines d'expertises. Une application dont l'architecture est bien structurée implique souvent une simplification de la lecture et de la compréhension du code. De ce fait, il est possible d’effectuer des travaux de maintenance et d’expansion (ajout de fonctionnalités) en limitant les ressources allouées à ces tâches.

Dans le cas d’une architecture MVC, un modèle contient les fonctionnalités qui ont pour objectif d’effectuer le traitement des données et d'établir les différentes interactions avec la base de données. Pour ce qui est d’une vue, sa fonction principale est de permettre l’interaction entre un humain et une machine. Elle doit donc être en mesure de présenter l’information renvoyée par le modèle et de gérer l’interaction de l’humain avec l’application. Le contrôleur quant à lui, effectuer le travail de synchronisation entre une vue et le modèle appropriés. C’est donc lui qui fait l’appel aux différentes actions à réaliser selon les événements obtenus.

Contrairement aux laboratoires précédents, celui-ci possède une structure différente. Dans le cadre des deux premiers laboratoires, nous devions réaliser trois sections différentes, dont la première section étant la réalisation d’un tutoriel. La majeure différence entre les deux premiers laboratoires et ce troisième laboratoire est le fait que la notion de tutoriel s’avère être abolie. Les deux autres sections restent cependant similaires. La première section consiste à effectuer un travail de documentation et de recherche nous permettant de nous familiariser avec les différents « *frameworks* » tandis que la dernière section consiste à l’intégration des « *frameworks* » retenues dans notre application.

## Réponses aux questions d’analyse

**Question 1 :**

En ce qui concerne le « *framework* » de persistance, nous pensons que *Hibernate* est un choix plus que judicieux. Avant même de nous lancer dans la documentation et la recherche, ce « *framework* » nous a été présenté dans la matière du cours d’architecture distribuée orientée objet (LOG720). Les membres de notre équipe possédaient également des connaissances de base concernant l’utilisation de ce « *framework* » de persistance.

Suite à une courte documentation et recherche sur cette technologie, nous pouvions affirmer qu’elle est largement supportée par la communauté web. Cela impliquait donc que si nous faisions face à des obstacles, il serait facile de trouver de documentations pour nous aider à les résoudre. De plus, nous avions appris qu’*Hibernate* fait partie des nombreuses dépendances supportées par *Maven*. En plus d’être populaire et qu’il répond à nos besoins, il était très simple de l’intégrer à notre projet grâce à notre implémentation de *Maven*.

Le principal avantage qu’offre ce « *framework* » est la persistance des données tout en offrant une transparence d’accès à la base de données. Lorsqu’on parle de transparence, on parle également de portabilité du code entre les divers types de bases de données. Cette transparence est atteinte grâce au fait qu’*Hibernate* permet de générer du *SQL* en fonction du système de gestion de base de données employé.

De plus, *Hibernate* est doté d’une interface de programmation *Java* permettant d’effectuer en toute simplicité des requêtes à la base de données sans avoir recours à l’utilisation du *SQL*. Cela permet donc à un programmeur moins expérimenter d’effectuer des appels à une base de données à l’aide de ce qu’on appelle des critères (*criteria*).

**Question 2 :**

Nous devons avouer que l’utilisation du « *framework* » de persistance a su démontrer de nombreux avantages et ce, sur plusieurs aspects. Dans cette section de ce rapport de laboratoire, nous allons vous faire part des principales améliorations observées suite à l’intégration d’*Hibernate* en tant que « *framework* » de persistance des données.

Tel que mentionné dans l’introduction de ce rapport de laboratoire, certains « *frameworks* » permettent de regrouper les fonctionnalités selon des domaines d’expertises définis. Dans le cas d’*Hibernate*, nous avons été forcés de déplacer l’ensemble des fonctionnalités concernant la configuration du système de gestion de la base de données dans un fichier *XML* commun. Ce transfert des fonctionnalités a permis d’augmenter de façon considérable, la maintenabilité de notre application. En d’autres termes, tout ce qui a trait au « *mapping* » des données de configuration et de connexion à la base de données se retrouvera dans un fichier *XML* géré par *Hibernate*. Sachant cela, le mainteneur ne perdra pas de temps à chercher ces fichiers de configuration.

Tel que mentionné précédemment, l’ensemble de la logique d’affaire concernant la connexion et la configuration de la communication avec une base de données se retrouve dans des fichiers de configurations gérés par *Hibernate*. On vous a donc affirmé que cette architecture logicielle permet d’accroître la maintenabilité de l’application. Mais cela n’est pas tout, elle permet également d’accroître l’extensibilité de celle-ci. La raison est simple. Adopter une architecture centrée sur des fichiers de configuration permet de faciliter l’ajout et la modification de relations entre les tables d’une base de données adoptant un schéma relationnel. Cela simplifie largement le processus d'extensibilité du logiciel.

Pour finir, dans le cadre du deuxième laboratoire, l’accès aux données impliquait d’effectuer des tâches de lecture et d’écriture dans la base de données. Le problème face à cette solution est au niveau de l’optimisation du processus. Lors d’une requête, afin de créer un objet, l’application devait lire une valeur à la fois dans la base de données. Suite à cette lecture, l’application devait passer par une boucle afin de créer un objet à partir des résultats obtenus. L’implémentation d’*Hibernate* a permis de résoudre ce problème, car il offre la notion de matérialisation et de dématérialisation d’objets qui permettent par exemple, d’accroître la performance des requêtes d’écriture.

**Question 3 :**

En ce qui concerne notre choix du « *framework* *Web* », notre équipe de travail s’est attardée sur le « *framework* » connu sous le nom de « *Spring MVC* ». Nous avons choisi ce « *framework* » pour plusieurs raisons. L’une de ces raisons est le fait que nous avions de l’expérience en ce qui concerne l’implémentation de ce « *framework* ». De plus, tout comme *Hibernate*, il fut un sujet de technologie étudié en classe.

Pour commencer, suite à l’analyse et à la documentation effectuées sur cette technologie, nous avons appris qu’elle s’intégrait bien avec le « *framework* » de persistance de données *Hibernate*. Nous avons ainsi trouvé une documentation abondante nous permettant d’accélérer le processus d’implémentation de cette technologie.

En ce qui concerne le côté technique, ce « *framework* » s’avère être un bon choix pour favoriser une architecture modulaire de l’application. Cela est dû au fait que ce « *framework* » implémente la notion de *MVC* (modèles, vues et contrôleurs). Une architecture implémentant la notion de *MVC*, permet d’accroître la maintenabilité et la modularité de l’application. Cette architecture permet en d’autres termes d’ajouter et de supprimer des composantes en limitant les impacts négatifs sur le fonctionnement de l’application logicielle.

Pour finir, afin de pallier la modularité d’une application, le « *framework* » *Spring MVC* offre la notion « d’inversion de contrôle ». Cette notion permet en effet d’effectuer ce qu’on appelle de l’injection de dépendances. Le principe de la modularité vient ainsi du fait que l’injection de dépendance permet de découpler les dépendances entre les objets.

**Question 4 :**

Dans le cadre du deuxième laboratoire, l’architecture de notre implémentation était centrée sur les servlets. Ceci étant dit, lorsqu’une page nécessite d’effectuer des requêtes, elle a recours à un servlet. Comme vous pouvez vous l’imaginer, cela va à l’encontre du principe de la modularité architecturale. La nécessité de lier une page à un servlet pour effectuer des requêtes présente une architecture adoptant la notion de fort couplage. Cela accroît considérablement la réutilisation et la maintenabilité du code.

Les servlets servaient ainsi d’intermédiaire entre une page et les contrôleurs. Les servlets avaient pour objectifs de capter toutes actions (commandes) entreprises dans une page et de retrouver les méthodes correspondantes à ces actions dans les contrôleurs, et ce, en fonction du nom de la méthode et de l’action en question.

Lorsque le servlet a trouvé la méthode correspondante à l’action entreprise, il s’occupe par la suite de trouver les dépendances nécessaires à cette méthode auprès du système des *DAO* étant l’abréviation des « *Data Access Objetcs* ». Ces dépendances permettront ainsi au contrôleur de bâtir la requête nécessaire permettant de répondre aux commandes entreprises par la page web concernée. Le contrôleur effectuera ainsi la demande des données nécessaires auprès des *DAO* qui retourneront les données au contrôleur. Ces données seront ensuite traitées et retournées au servlet qui les acheminera à la vue appropriée.

Comme vous pouvez le voir, c’est un processus assez complexe dû aux fortes dépendances architecturales. À l’aide du « *framework* » *Spring MVC*, il a été possible pour notre équipe de travail d’améliorer l’architecture de notre application à l’aide du découplage composantes architectural (réduction significative des dépendances). Cela nous a ainsi permis de créer une multitude de méthodes permettant de répondre à plusieurs types de requêtes.

De plus, la réutilisabilité de requêtes implémentées s’est avérée fortement augmentée grâce à la notion d’injection des dépendances décrite dans la section précédente (réponse à la question 3). L’injection des dépendances s’effectuant au niveau des divers paramètres possibles de l’application web.

Bref, l’implémentation du « *framework* » *Spring MVC* fut un excellent choix permettant à notre équipe de travail de rendre notre application web propice aux tâches de maintenance et d’extensibilité (ajout et modification de fonctionnalités).

## Conclusion

Tel que mentionné dans l’introduction de ce rapport de laboratoire, la structure de ce troisième laboratoire fut différente des deux derniers, car il ne comporte que deux sections au lieu de trois. Nous tenons à préciser que cela n’a en aucun cas eu un impact négatif sur notre apprentissage. La réalisation de ce troisième laboratoire a nécessité plusieurs heures d’analyse et de documentation sur les différents types de « *frameworks* ».

Ceci étant dit, ce troisième et dernier laboratoire du cours d’architecture distribuée orientée objet (LOG720) nous a permis d’étendre nos connaissances en termes de développement web implémentant des technologies offrant une structure architecturale performante, extensible et simple à maintenir.

À l’aide du « *framework* » de persistance de données *Hibernate* ainsi que le « *framework web* » *Spring MVC*, nous avons été en mesure de concevoir un système adoptant une architecture distribuée orientée objet. Ces technologies nous ont permis d’améliorer notre application web développée dans le cadre du deuxième laboratoire. Ces améliorations se retrouvent principalement au niveau de la réutilisabilité du code le rendant ainsi facilement extensible (ajout ou modification de fonctionnalité). De plus, la maintenabilité de l’application web a également subi une nette amélioration (plus simple à maintenir).

Bref, nous sommes en mesure d’affirmer que la nouvelle architecture implémentée présente une grande amélioration par rapport à l’architecture employée dans le cadre du deuxième laboratoire. Nous avons également été en mesure de valider l’apport positif d’une architecture axée sur distribution orientée objet.