Architecture logicielle

Auteurs :
Simon Descamps
Claire Garcia

Tuteur :
Benoit Verhaeghe

Table des matières

1 Introduction

Dans le cadre de notre formation d'ingénieur statistique et informatique à Polytech Lille, nous sommes amenés à réaliser un projet d'architectures logicielles. Nous nous baserons sur les connaissances que nous avons précédemment acquises lors des cours d'ingénierie logicielle mais également sur les compétences acquises en programmation Java lors des cours de programmation par objet. En effet, ce projet est codé en Java.

Vous trouverez l'archive de notre projet en suivant ce lien :
`https://gitlab.univ-lille.fr/simon.descamps.etu/projet-al`

L'objectif de ce projet est de disposer d'une application JEE permettant de générer des questionnaires web et d'en évaluer les réponses. Un utilisateur pourra accéder à un questionnaire par une page web, y répondre et enfin vérifier l'exactitude de ses réponses.

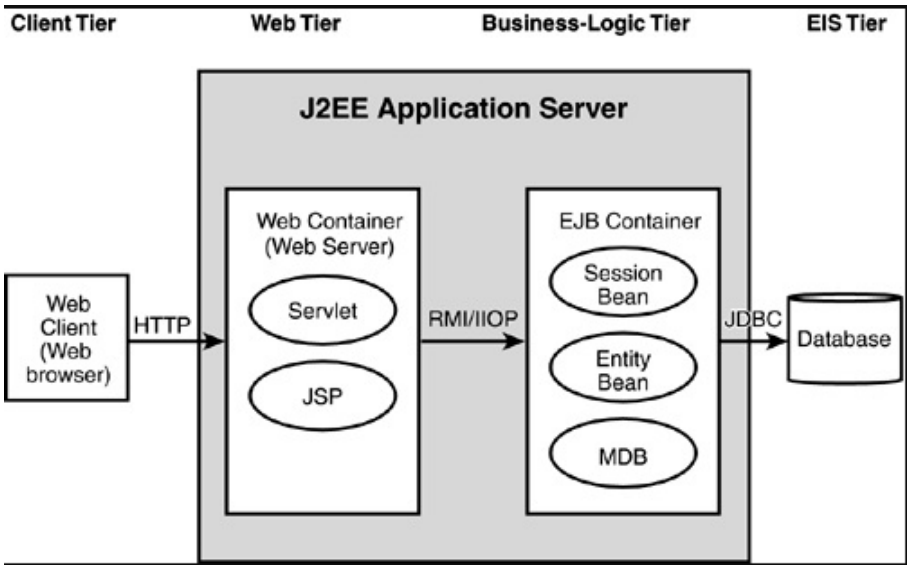
Dans ce rapport, nous décrirons les classes UML qui nous permettrons de répondre au sujet, puis nous commenterons la programmation de celle-ci. Ensuite, nous présenterons les différentes vues que nous avons utilisé pour la partie web avant d'expliquer comment exécuter nos fichiers. Pour finir, nous évoquerons les difficultés rencontrées tout au long de ce projet.

2 Analyse du problème

L'objectif est de créer une application qui permet la génération de questionnaires en ligne, leur remplissage par un utilisateur et leur évaluation. Un questionnaire peut contenir deux types de questions. En effet, il contiendra des questions dites ouvertes, ou encore textuelles, où l'utilisateur rentrera lui même sa réponse et des questions dites fermées où des choix de réponse seront proposés à l'utilisateur. Les question fermées sont elles aussi divisées en deux types. Elles peuvent avoir un seule bonne réponse ou plusieurs bonne réponses. C'est ce qu'on appelle des questions à choix unique ou multiple.

Nous allons stocker les questionnaires, les différents types de questions et leurs réponses dans une base de données. Il devra être possible de consulter, de créer, ou de modifier les questionnaires. Dans ce projet la modification possible est d'ajouter des nouvelles questions à un questionnaire. L'accès aux questionnaire se fera exclusivement par accès web. Il y aura une page web pour chaque questionnaire.

Pour mener à bien ce projet nous allons utiliser l'architecture J2EE.
Voici un schéma de l'architecture logicielle utilisée :



Source : https ://ejbvn.wordpress.com/category/week-3-advanced-ejb-applications/day-21-developing-a-complete-enterprise-application/

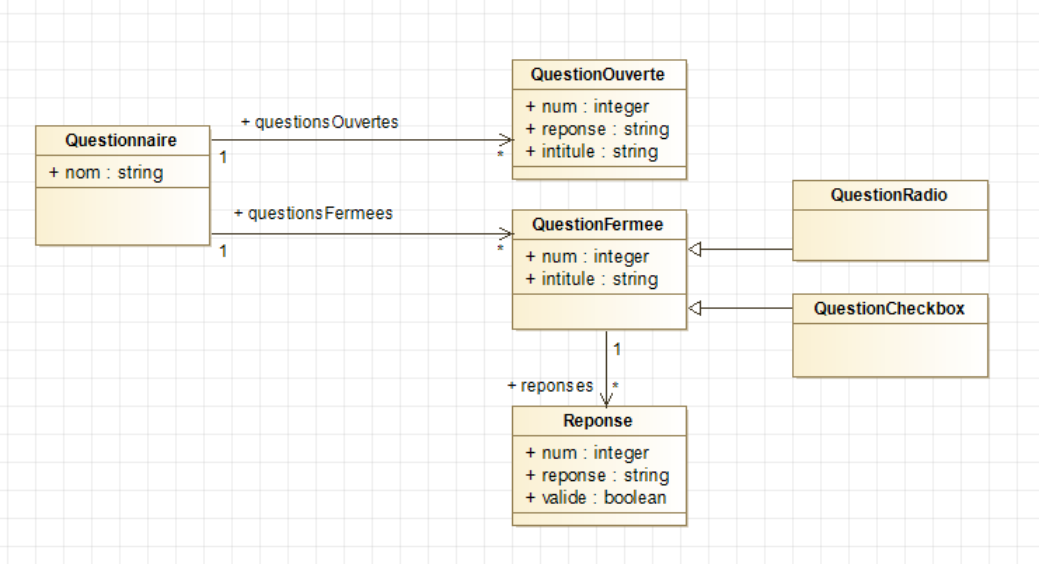
La couche client représente la partie de l'application avec laquelle l'utilisateur va interagir. On parle d'interface homme-machine. Ici, l'utilisateur va interagir avec des pages web.

La partie fonctionnelle de l'application va recouvrir les conteneurs de l'application et le conteneur web. Le conteneur d'application qui fournira tout ce qui est nécessaire pour l'exécution des JavaBeans. Le conteneur Web quant à lui servira d'environnement d'exécution des JSP et servlet

La couche d'accès aux données va regrouper ce qui permet de gérer l'accès à la base de données.

3 Analyse des classes

Pour répondre au sujet nous avons choisi la structure de données suivantes :



Nous avons la classe **Questionnaire** qui a pour clé primaire son nom. Cela implique qu'on ne pourra pas avoir deux questionnaires portant le même nom. En effet, nous souhaitons retrouver les questionnaire par leur nom. Cette classe est reliée aux classes **QuestionOuverte** et **QuestionFermee**. Bien qu'il s'agisse de deux classes représentant des questions, nous avons choisi de ne pas les mettre dans une sur-classe *Question* car nous ne souhaitons pas les stocker dans la même table de notre base de données. Nous estimions qu'il était donc plus compréhensible de les séparer dans notre schéma UML.

La classe **QuestionOuverte** permettra d'implémenter les questions dites *ouvertes*, c'est à dire où il n'y à qu'une seule bonne réponse et que c'est l'utilisateur lui-même qui saisie la réponse.

La classe **QuestionFermee** permettra d'implémenter les questions dites *fermées*. Une liste de réponses sera toujours proposée à l'utilisateur. Ces dernières correspondent à la classe **Reponse**. Nous avons ajouté l'attribut booléen valide qui permet de définir si une réponse est bonne ou fausse. Les questions fermées sont séparables en deux sous-classes :

— ***QuestionRadio*** qui peut contenir qu'une seule bonne réponse

— ***QuestionCheckbox*** qui peut contenir plusieurs bonnes réponses

Remarque : Puisqu'une question ouverte n'a qu'une unique réponse, nous stockons cette dernière dans un attribut reponse dans la classe.

Nous avons choisi de rendre les classes navigables dans un seul sens uniquement. En conséquence, on peut, depuis un questionnaire retrouver toutes les questions et les réponses mais on ne pas retrouver un questionnaire depuis une question.

Nous avons choisi d'utiliser un attribut *num* de type entier qui sera généré automatiquement pour les questions et les réponses. Cela permet de créer des questions qui ont le même intitulé dans le cas où on voudrait mettre deux questions du même

intitulé dans un questionnaire ou dans différents questionnaires, sans pour autant qu'elles aient les mêmes réponses ou le même type.

4 Programmation des EJB entités

Les composants Enterprise Java Bean (EJB) entités s'assurent de la représentation des données et de la persistance de celle-ci.

Chaque classe et sous-classe que nous avons définies dans la section précédente correspond à une entité. La traduction de ces classes UML en classes Java se fait par l'annotation @Entity. De plus, chaque classe Java doit contenir un constructeur sans paramètre et être serializable pour être considérée comme une entité.

Ces classes vont être reliées à la base de données grâce au descripteur XML persistence.xml.

Nous avons défini une propriété pour chaque éléments de chaque classe. En effet, nous avons respecté la convention de nommage et crée des getElt(), et isElt() pour les booléens, afin d'accéder aux éléments de la classe, et des setElt() afin de les modifier.

Comme vu précédemment, nous avons choisi de générer automatiquement un identifiant pour les classes QuestionOuverte, QuestionFermee et Reponse. Pour cela nous avons utilisé l'annotation @GeneratedValue. Pour définir cette valeur en tant qu'identifiant nous lui avons rajouté l'annotation @Id.

Pour représenter les relations *1-**entre les classes, nous stockerons dans la classe du côté du 1 un Set du type de la classe à laquelle elle est reliée. Ce Set sera annoté par @OneToMany pour traduire l'association *1-** des classes.

Nous stockons donc dans la classe QuestionFermee : Set<Reponse>.

Également nous stockons dans la classe Questionnaire : Set<QuestionFermee> et Set<QuestionOuverte>

L'utilisation de Set est plus intéressante que l'utilisation de Arrays car elle nous permet d'accéder directement à un élément du set grâce à sa clé.

5 Programmation des EJB sessions

Les composants Enterprise Java Bean (EJB) Session assurent les services métiers. Nous avons donc crée le composant serviceQuestionnaireBean qui contiendra les règles métiers de l'application.

Il accède aux composants entités avec différentes méthodes que nous décriront dans la partie suivante. Cela est rendu possible car, comme dit précédemment, nous avons défini un constructeur sans paramètre dans chaque classe.

Nous n'avons pas besoin de mémoriser un état entre deux appels de méthodes nous avons donc défini notre composant sans état transactionnel et donc annoté le composant par @Stateless.

Nous nous sommes assurer de la persistance des données en important le package EntityManager. Nous pouvons retrouver les entités grâce à leur clé primaire.

Le composant serviceQuestionnaireBean sera accessible localement et à distance avec des méthodes communes. Nous avons implémenté au composant deux interfaces qui sont annotées l'une par @Local et l'autre par @Remote.

5.1 Méthodes utilisées

Nous avons rassemblé les méthodes créées dans serviceQuestionnaireBean en fonction des classes qu'elles concernent dans les tableaux suivants.

Méthodes concernant la classe Questionnaire :

Méthodes	**Descriptions**
creerQuestionnaire(String nom)	Créer un questionnaire.
getQuestionnaire(String nom)	Renvoi un questionnaire à partir de sa clé primaire
getQuestionnaires()	Renvoie tous les questionnaires de la base.

Méthodes concernant la classe QuestionOuverte :

Méthodes	**Descriptions**
getQuestionOuverte(int id)	Renvoi une question ouverte à partir de sa clé primaire.
addQuestionOuverte(String questionnaire, String intitule, String reponse)	Ajoute une question ouverte à un questionnaire avec comme attribut l'intitulé et la réponse à la question.
testReponseOuverte(int num, String reponse)	Vérifie l'exactitude de la réponse à la question ouverte en renvoyant un booléen.

Méthodes concernant la classe QuestionFermee :

Méthodes	**Descriptions**
getQuestionFermee(int id)	Renvoi une question fermée à partir de sa clé primaire.
getQuestionsFermees()	Renvoie une collection de toutes les questions fermées.
addQuestionFermee(String questionnaire, TypeSpec type, String intitule)	Ajoute une question fermée à une questionnaire avec comme attribut l'intitulé à la question. Le type va permettre de définir si la question a une ou plusieurs bonnes réponses.
testReponseFermee(int num, String[] reponses)	Vérifie l'exactitude de la réponse à la question ouverte en comparant le tableau des réponses données aux réponses et en renvoyant un booléen.
addReponseFermee(int question, String reponse, boolean valide)	Ajoute une réponse à une question fermée avec comme attribut l'intitulé de la réponse et un booléen qui permet de définir si la réponse est bonne ou non.

5.2 Gestion des erreurs

Nous avons utilisé les exceptions suivantes afin de gérer aux mieux les éventuelles erreurs.

L'exception **QuestionnaireDejaCreeException** permet de vérifier lorsque l'on veut créer ou ajouter un questionnaire que celui-ci n'existe pas déjà.

L'exception **QuestionnaireInconnuException** est émise si le questionnaire recherché n'existe pas

L'exception **QuestionDejaAjouteeException** permet de vérifier lorsque l'on veut ajouter une question que son intitulé ne correspond pas déjà à une question du questionnaire.
Remarque : nous avons laissé la possibilité d'ajouter une question ouverte et une question fermée ayant le même intitulé.

L'exception **QuestionInconnueException** est émise si la question recherchée n'existe pas.

L'exception **ReponseDejaAjouteeException** est émise si l'intitulé de la réponse a déjà été ajoutée comme réponse à la question.

L'exception **UneReponseValideParQuestionRadioException** est émise si on souhaite ajouter une réponse valide à une questionRadio alors qu'on en a déjà ajoutée une.

Vous retrouverez dans le tableau suivant la liste de chaque méthodes et les exceptions qu'elles provoquent.

Méthodes	**Exceptions**
creerQuestionnaire(String nom)	QuestionnaireDejaCreeException / QuestionnaireInconnuException
getQuestionnaire(String nom)	QuestionnaireInconnuException
addQuestionnaire(String questionnaire)	QuestionnaireDejaCreeException
getQuestionFermee(int id) / getQuestionOuverte(int id)	QuestionInconnueException
addQuestionOuverte(String questionnaire, String intitule, String reponse) / addQuestionFermee(String questionnaire, TypeSpec type, String intitule)	QuestionnaireInconnuException / QuestionDejaAjouteeException
addReponseFermee(int question, String reponse, boolean valide)	QuestionInconnueException / ReponseDejaAjouteeException / UneReponseValideParQuesitonRadioException
testReponseFermee(int num, String[] reponses) / testReponseOuverte(int num, String reponse)	QuestionInconnueException

6 Premiers tests

Avant de se lancer dans le développement des composants web, il est impératif de s'assurer du bon fonctionnement de nos EJB entités et sessions.

Nous avons donc, dans le fichier main.java, créé deux questionnaires. Parmi ces deux questionnaires, nous avons recréé le questionnaire du sujet. Pour exécuter le fichier main.java, il faut en premier lieu déployer les entités et les sessions sur le serveur java. Cela est faisable très facilement en suivant les explications des TP pour afficher la fenêtre Ant et en double cliquant sur "Deploy entities and sessions". Ensuite, il est important de compiler et d'exécuter le fichier main à l'aide des actions "Compile client" et "Run client".

L'exécution de ce fichier nous permet de tester les premières méthodes de notre bean mais également de vérifier qu'elles renvoient bien des exceptions. Pour cela, il suffit de relancer une nouvelle fois le client et de constater que des exceptions sont capturées car les questionnaires et les questions sont déjà rentrés dans la base de données.

Il est possible de visualiser les questionnaires créés dans la console du serveur wildfly sur lequel on a déployé nos EJB. Pour cela, il faut se rendre à l'url suivante : `localhost:8080/h2-console` .

L'interface de la console est assez intuitive, on peut y exécuter des requêtes SQL.

7 Développement des composants WEB

Nous avons créé différentes vues WEB pour répondre au cahier des charges. L'interaction entre les vues est gérée par un contrôleur (Controleur.java), il permet de passer d'une vue à l'autre et de récupérer les valeurs des variables passées par les méthodes POST ou GET dans les formulaires.

Voici la liste de toutes les vues web :

Vue	**Description**
addQuestionQuestionnaire.jsp	Confirme ou non l'ajout d'une question à un questionnaire. Une redirection vers le questionnaire est effectuée en cas de succès.
addReponseQuestion.jsp	Confirme ou non l'ajout d'une réponse à un question. Une redirection vers le questionnaire est effectuée en cas de succès.
admin.html	Correspond au panneau d'accueil de la page d'administration depuis laquelle on peut consulter (et non répondre) un questionnaire et en créer de nouveaux.
creerQuestionnaire.jsp	Confirme ou non la création d'un questionnaire. Une redirection vers la page d'administration est effectuée en cas de succès.
index.html	Correspond à la page d'accueil du projet. Permet d'accéder à la page d'administration, de répondre à un questionnaire existant (en modifiant le nom dans l'URL) et de consulter un questionnaire.
questionnaire.jsp	Permet de répondre au questionnaire passé dans l'URL avec l'attribut nom (par exemple nom=serieTV)
reponseQuestionnaire.jsp	Permet de vérifier les réponses au questionnaire auquel vous avez répondu
viewQuestionnaire.jsp	Permet de consulter un questionnaire, d'y ajouter des questions et des réponses

Nous avons également rajouté une feuille de style afin de rendre le questionnaire un peu plus esthétique.

8 Exécution du projet

Le projet correspond au dossier questionnairesdescamp qui est téléchargeable à l'adresse suivante :`https://gitlab.univ-lille.fr/simon.descamps.etu/projet-al.git`.
Pour accéder à l'application, il faut en premier temps ouvrir le projet sous éclipse et le compiler. Il faut donc se rendre sur la fenêtre Ant et double-cliquer sur *earEntityAndSessionAndWeb*. Si vous n'avez pas accès à la fenêtre Ant, veuillez-suivre les indications dans les fiches de TP architecture logicielles. Une fois compilé, vous devez déployer le projet sur le serveur Wildfly préalablement lancé sur votre machine. Le déploiement se fait en double cliquant sur *deployAll*.

Une fois déployé, vous pouvez vous rendre à l'URL `http://localhost:8080/projet/` et vous serez immédiatement renvoyés vers la vue par défaut du projet qui correspond à l'accueil. Vous pouvez accéder à tout depuis cette page, cependant, si vous souhaitez accéder directement à un questionnaire, cela est possible en se rendant à l'URL `http://localhost:8080/projet/questionnaire?nom=serieTV`. Bien évidemment, il faut remplaçer serieTV par le nom du questionnaire auquel vous voulez accéder (à condition qu'il existe dans la base).

9 Difficultés rencontrées

Globalement, le projet ne paraissait pas difficile en apparence car il ne faisait que reprendre ce que nous avions vu en TP. Cependant, le confinement a apporté beaucoup de difficultés, notamment pour mettre en place le serveur Wildfly. La seconde difficulté liée au confinement est que nous n'avions pas les professeurs avec nous lorsque nous rencontrions des erreurs et cela nous a fait perdre beaucoup de temps à les débugger alors qu'il s'agissait parfois d'erreurs dans la configuration d'éclipse ou de wildfly. Il faut quand même admettre que la partie WEB était un peu plus compliquée que ce que nous avions fait en TP.

Deuxièmement, puisque nous avons passé les premières séances à mettre en place Eclipse, nous n'avons pas voulu perdre de temps et avons avancé sur le projet hors des créneaux. Bien évidemment, nous n'avions pas fait valider notre structure de données et nous avons dû tout reprendre alors que nous avions terminé les sessions et la partie web. Corriger cela n'a pas été très difficile, il fallait simplement tout corriger un par un. Le plus dur a été de repérer ce qui ne servait plus à rien dans le code. On avait par exemple des fonctions qui n'étaient plus utilisées. Il était tout de même important de corriger notre structure de données afin de limiter les éventuels problèmes sur l'application. En effet, telle que nous l'avions implémentée initialement, l'application empêchait l'ajout de multiples réponses à une question ouverte dans la partie web. Cela signifiait que si quelqu'un récupérait l'application pour développer ses propres vues web, il pouvait alors ajouter des réponses à une question ouverte. Nous avons donc réglé le problème plus localement en modifiant notre base de données et en effectuant des vérifications dans le code métier.

Enfin, il était difficile de travailler en binôme malgré l'utilisation du Git.

10 Conclusion

Nous avons apprécié passer du temps sur ce projet car il était très instructif. Nous avons su mettre en application ce que nous avons appris lors des cours et des séances de TP d'architecture logicielle. Nous avons construit une architecture J2EE qui permet de créer un questionnaire et de le lier à une base de données.

Ce projet est très enrichissant pour notre future carrière professionnelle car il allie la double compétence informatique et data à travers la gestion de la base des données.

Bilan personnel de Claire

Ce projet a permis de développer mes connaissances apprises lors des cours d'architecture logicielle en les appliquant sur un sujet concret. Il est agréable de travailler avec Simon car c'est un binôme très efficace.

Cette situation de confinement a peut être compliqué la communication mais je suis tout de même contente que nous ayons rendu un travail qui fonctionne et d'avoir développé mes compétences. :)

Bilan personnel de Simon

Le projet était intéressant, il nous a permis de travailler bien plus en autonomie que nous l'aurions fait si nous avions été à Polytech. Il nous a en conséquent fait beaucoup plus travailler les neurones. Personnellement, ça ne me dérange pas de passer du temps sur quelque chose qui m'intéresse. Enfin, il m'a permis de bien assimiler ce que nous avons vu en cours tout au long du semestre et de bien comprendre le fonctionnement des EJB.