Artus VUATRIN
Apolline LAFARGE

SR03 – Projet Java EE

Ce deuxième projet a pour objectif de nous familiariser avec la technologie Java EE.
Pour ce faire, il nous a été proposé de réaliser une application web d'évaluation de stagiaires.

I – Cahier des charges

Description de l'application

L'application réalisée permet d'évaluer les compétences de stagiaires par le biais de différents questionnaires à choix multiples.
Elle gère deux types d'utilisateurs : les administrateurs et les stagiaires, et propose des fonctionnalités différentes pour chacun d'entre eux.
Les administrateurs ont la possibilité de gérer les utilisateurs et les questionnaires : ils peuvent les visualiser, les modifier ou encore en créer. Il leur est également permis de consulter les résultats obtenus par les stagiaires aux différents questionnaires.
Quant aux stagiaire, l'application leur permet de répondre aux questionnaires ainsi que de consulter l'historique de leur résultats.

Technologies utilisées

Nous avons utilisé les technologies suivantes pour le développement de notre application.

Environnement de développement	Eclipse
Serveur d'application	Apache Tomcat
SGBD	MySQL
Langages utilisés	Java, HTML

Artus VUATRIN
Apolline LAFARGE

II – Base de données et architecture

Base de données

Le modèle conceptuel de la base de données associée à notre application est le suivant :

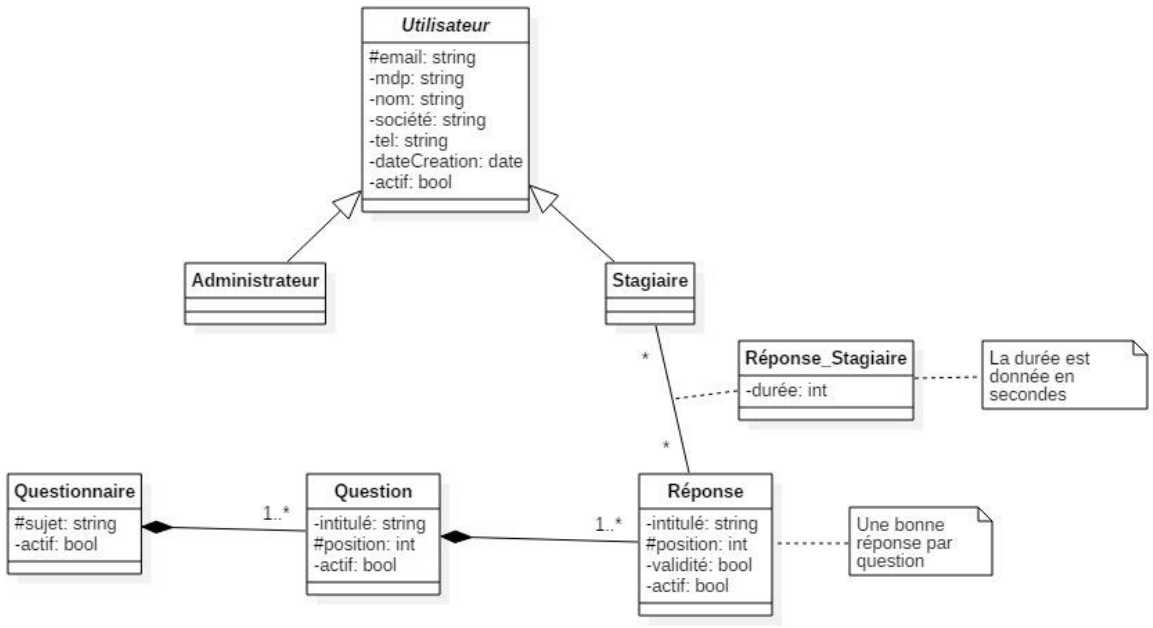


Figure 1 : Modèle conceptuel de la base de données

Après avoir établit les modèles conceptuel et logique de données, nous avons implémenté notre base de données au sein du SGBD MySQL.

Nous pouvons faire différentes remarques sur la base de données réalisée :

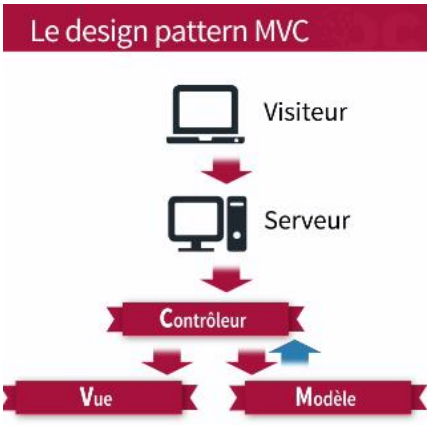
- Comme on peut le voir sur le diagramme UML, la classe `Utilisateur` est abstraite, et seules les tables `Administrateur` et `Stagiaire` sont implémentées dans la base de données. Cependant, il peut nous être utile d'avoir une table répertoriant l'ensemble des utilisateurs, notamment pour la fonction de login de l'application. Nous avons résolu ce problème par la création d'une vue `Utilisateur`, faisant l'union des tables `Administrateur` et `Stagiaire` et spécifiant le type d'utilisateur.
- Le parcours des stagiaires sur les différents questionnaires est stocké par une simple classe d'association entre les classes `Stagiaire` et `Réponse` (on suppose alors qu'un Stagiaire ne peut effectuer plusieurs fois un même questionnaire). Un questionnaire étant composé de questions, et une question étant composée de réponse, la table `Réponse_Stagiaire` suffit à stocker l'ensemble des informations relatives à un parcours. En effet, la requête SQL suivante permet de récupérer l'ensemble des parcours réalisés par un stagiaire donné :

```
SELECT DISTINCT Qr.id AS questionnaire, Qr.sujet AS sujetQuestionnaire,
SUM(RS.duree) AS duree, SUM(R.valide) AS score, COUNT(*) AS nb_questions
FROM reponse_stagaire RS, reponse R, question Q, questionnaire Qr
WHERE RS.stagiaire = idStagiaire
AND RS.reponse = R.id
AND R.question = Q.id
AND Q.questionnaire = Qr.id
GROUP BY Q.questionnaire;
```

- Pour simplifier notre l'écriture de nos requêtes SQL, nous avons écrit quelques fonctions permettant d'accéder directement à certaines informations. En effet, l'utilisation de clés artificielles (id) nous a permis d'obtenir des clés primaires plus simples, mais a compliqué l'accès à certaines informations. Par exemple, l'obtention du questionnaire associé à une réponse demande de nombreuses opérations de jointure, et la fonction `get_questionnaire_of_reponse(id_reponse int(6))` nous permet d'accéder directement à cette information.

Design Pattern MVC

Afin d'avoir un code plus propre et plus structuré pour faciliter le travail en groupe, nous avons choisi d'utiliser le design pattern Modèle Vue Contrôleur.



Cette structure nous demande de séparer chaque classe et fichier afin que chacune aie un rôle bien précis. Le contrôleur doit recevoir la requête du client, transmet les données au modèle s'il est nécessaire de récupérer des données en base de données, ou si des modifications de la base doivent être faits. Ensuite, le contrôleur récupère ces données et les formate avant de les renvoyer à la vue. La vue se charge de placer les données, dans la page. C'est cette page complétée qui sera renvoyée à l'utilisateur.

Dans notre implémentation du modèle MVC en Java EE, nous avons dû trier chaque classe selon leur rôle. Les servlets se chargent du contrôle, selon la requête de l'utilisateur, le fichier web.xml donne le nom du servlet à instancier. Ce servlet appellera la fonction `DoGet` ou `DoPost` selon la requête de l'utilisateur. Les fonctions du modèle sont réalisées par les JavaBeans. Il existe un objet pour chaque type de donnée en base ainsi que des classes pour gérer ces données comme expliqué dans la partie suivante. Enfin, les fichiers JSP permettent de réaliser les vues.

Artus VUATRIN
Apolline LAFARGE

Modèle DAO

L'échange d'informations avec la base de données a été implémenté selon le modèle DAO (Data Access Object). Celui-ci vise à séparer les requêtes SQL des objets et traitements métier (soit le Modèle de MVC) et permet de gérer de manière transparente différents systèmes de stockage.

Nous avons donc implémenté différentes classes permettant la communication de données concernant les utilisateurs, les questionnaires, les questions et les réponses. Pour chacun de ces objets, nous disposons de :

- Une interface définissant les méthodes de manipulation des données, sans pour autant les implémenter :
  - `lister()` permet de récupérer l'ensemble des objets ;
  - `trouver()` permet de récupérer un objet à partir de son id ;
  - `creer()` permet d'ajouter un objet à la base de données ;
  - `modifier()` permet de mettre à jour un objet existant ;
  - `supprimer()` permet de supprimer un objet de la base de données.

  L'interface liée aux utilisateurs comporte des méthodes supplémentaires pour le login et la gestion des parcours des stagiaires.

  Ces interfaces permettent de faire abstraction du système de stockage.
- Une implémentation de l'interface, contenant les requêtes SQL.

Enfin, nous avons implémenté une classe DaoFactory. Elle a pour rôle d'initialiser le DAO en chargeant les drivers nécessaires et de se connecter à la base de données. Elle fournit également des différents DAO décrit précédemment. Pour ce faire, elle dispose des méthodes suivantes :

- `getInstance()` appelle de driver JDBC et se connecte à la base de données. Elle est utilisée par les servlets pour récupérer les différents DAO.
- `getConnexion()` récupère à tout moment la connexion à la base de données. Elle est utilisée par les implémentations des DAO pour exécuter les requêtes SQL.
- Les méthodes de récupération des DAO, utilisées par les servlet pour communiquer avec la base de données.

Finalement, pour échanger des informations avec la base de données, les servlets instancient la classe `DaoFactory` et récupèrent les objets DAO utiles lorsque leur fonction `init()` est appelée. Ensuite, elle fait simplement appel aux méthodes des objets DAO stockés.

Artus VUATRIN
Apolline LAFARGE

III – Application réalisée

Page de login

Cette page permet à un utilisateur de se connecter pour accéder à son espace de travail. Le servlet Test se charge de cette fonctionalité. La méthode DoGet de ce servlet est appelée en premier. Elle se charge d'afficher la page sans appel à la base de données. La méthode DoPost est appelée quand l'utilisateur a entré ses identifiants. Le servlet teste si ces données existent en base, si oui, les informations sur l'utilisateur sont enregistrées en session puis on affiche une page d'accueil, sinon, on réaffiche la page de login avec un message d'erreur.

Figure 2: Page de login

Fonctionnalités administrateur

- **Page de gestion des utilisateurs *(servlet Utilisateurs)***

A l'arrivée sur cette page, la `méthode DoGet` est appelée, elle se charge de demander au modèle de récupérer tous les utilisateurs en base. La vue affiche ensuite tous ces utilisateurs ainsi que leurs caractéristiques et des boutons permettant de gérer ces utilisateurs. Une petite astuce javascript permet d'avoir des formulaires dans des tableaux via la fonction `submitForm(element)`. Au clic sur un des boutons, le servlet va analyser la demande de l'utilisateur. Une première vérification est faite pour s'assurer que l'utilisateur est bien un administrateur. La demande peut être : ajout d'utilisateur, suppression ou modification de ces caractéristiques. En fonction cette la demande, le servlet va récupérer les données transmises en POST puis, communique avec la base via les objets du modèle pour effectuer ces changements. Si une exception est levée, un message sera affiché à l'utilisateur, sinon, un message indiquant le succès sera affiché. Une fois la modification faite, le servlet effectue les mêmes actions que le `DoGet` pour réafficher la liste des utilisateurs modifiée.

Figure 3: Page de gestion des utilisateurs

- **Page de gestion des questionnaires *(servlet Questionnaires)***

La méthode DoGet se charge de récupérer tous les questionnaires en base avant de les afficher dans la vue avec les boutons nécessaires à la modification ou la suppression, ainsi qu'un formulaire d'ajout de questionnaire. A l'envoi d'un des formulaires, on analyse la demande de l'utilisateur s'il est administrateur, puis on récupère les données envoyées et on traite sa demande, tout comme dans la page précédente.

Figure 4: Page de gestion des questionnaires

- **Page de gestion d'une question *(servlet Question)***

La page affichant les détails d'un questionnaire doit avoir un paramètre GET qui n'est autre que l'identifiant du questionnaire. Dans un premier temps, la fonction DoGet doit donc vérifier si ce paramétré est bien renseigné et s'il pointe vers un questionnaire existant. Une fois cette vérification faite, le servlet demande aux Java Beans d'instancier toutes les questions ainsi que leurs réponses associées. Elles sont ensuite placées dans la vue. Encore une fois, chaque question et chaque réponse peut être modifiée ou supprimée via les boutons associés. On peut aussi ajouter une question ou une réponse ou encore changer la bonne réponse à une question.

A l'envoi d'un des formulaires, la méthode DoPost analyse la demande et la traite comme les pages précédentes avant de réafficher les questions et réponses mises à jour, accompagné d'un message d'information.

Figure 5: Page de gestion d'une question

Fonctionnalités stagiaire

L'application propose deux fonctionnalités aux stagiaires :

- Le remplissage de questionnaires ;
- La consultation de leurs résultats.

- **Ecran d'accueil *(servlet Stagiaire)***

A sa connexion, le stagiaire est redirigé vers une page lui affichant les sujets des différents questionnaires ainsi que les résultats qu'il a obtenu précédemment.
L'affichage des résultats inclut : le sujet du questionnaire, le score obtenu, ainsi que la durée du parcours.
Lorsque le stagiaire clique sur un sujet de questionnaire, il débute alors un nouveau parcours.
Par manque de temps, la pagination et la fonction de recherche sur les questionnaires n'a pas été implémentée.

Figure 6: Page d'accueil des stagiaires

- **Réponse aux questionnaires *(servlet Parcours)***

Après avoir choisi un questionnaire, le stagiaire est redirigé vers un formulaire répertoriant ses questions et réponses. Le choix d'afficher l'intégralité des questions sur une page unique a été motivé par des raisons de facilité : à la validation du formulaire, l'ensemble des réponses du stagiaire sont alors envoyées simultanément en POST au servlet.

Figure 7: Page de réponses à un questionnaire (ici composé d'une question)

Les variables envoyées sont identifiées par l'id de la question, et contiennent l'id de la réponse. Le servlet prend connaissance du nom des variables grâce à la méthode `request.getParameterNames()`, et effectue l'insertion de chaque réponse dans la base de données. Le temps de réponse du stagiaire n'est pas géré (un temps de réponse aléatoire compris entre 5 et 200 secondes est stocké en base de données).

Après la validation du formulaire, le stagiaire est redirigé vers son écran d'accueil (la page d'affichage des bonnes réponses n'a pas été réalisée par manque de temps).

Conclusion

Finalement, ce projet nous a permis de découvrir Java EE ainsi que l'environnement de travail Eclipse. Nous avons également pu nous accoutumer à de bonnes pratiques de développement par l'étude et la mise en œuvre des modèles MVC et DAO.