Proposition de démarche d'évolution d'une application JEE : un exemple avec le projet Equida

Il s'agit d'une proposition permettant de faire évoluer une application JEE, ici l'application Equida. L'ordre de certaines phases peut être modifié. Prenez-soin d'utiliser à bon escient le **vocabulaire technique** (en rouge dans le texte) et fonctionnel adéquat. Pour chaque “couche de l'architecture applicative”, commencer par faire un état de l'existant pour rechercher les ressources manquantes ou à modifier.

Expression du besoin

Une nouvelle fonctionnalité doit être ajoutée à l’application Equida. Il faut ajouter la possibilité à l’utilisateur de l’application de consulter la liste des chevaux sélectionnés lors d’une vente. On partira donc de la page « lister les ventes » et on ajoutera un lien qui affichera une page listant les chevaux sélectionnés pour la vente sur laquelle on aura cliqué.

Préalables

Cette nouvelle **fonctionnalité** sera développée à partir de la dernière version de l’application **publiée** sur gitHub. On créera une nouvelle **branche** nommée `listeLotsParVente` pour implémenter cette nouvelle fonctionnalité.

Modélisation / Diagramme de classes :

Le **diagramme de classes** tel qu’il est conçu actuellement permet de gérer les chevaux sélectionnés lors d’une vente. Il s’agit d’utiliser **les classes métier** Cheval / Lot / Vente.

Hypothèse : Dans un développement ultérieur, nous pourrions aussi utiliser la classe TypeCheval pour afficher les informations concernant la race du cheval et le bloc Courses/Participer pour afficher les résultats du cheval aux différentes courses auxquelles il a participé.

Base de données :

A partir du **diagramme de classes**, nous pouvons concevoir le **modèle logique de données**. Dans la base de données actuelle, les tables Cheval et Lot n’existent pas. Il faut modifier la base pour suivre le modèle logique ci-dessous :

Modèle Logique de données Cheval (id, nom, sire, ……) clé primaire : id clé étrangère ; idPère en référence à id de la table Cheval clé étrangère ; idMère en référence à id de la table Cheval

Lot(id, ...) clé primaire : id clé étrangère : idVente en référence à id de la table Vente clé étrangère : idCheval en référence à id de la table Cheval

Ajout de data : afin de tester l'application, nous ajouterons des données pertinentes dans les nouvelles tables.

Scripts sql : les scripts de modification de la structure de la base et d'insertion des nouvelles données seront sauvegardés dans deux fichiers distincts nommés xxxxST.sql et xxxxDT.sql.

Ces fichiers seront utilisés pour la livraison en production de la nouvelle fonctionnalité.

Couche applicative ''Modele '':

Il faut ajouter les **classes métiers** Cheval et Lot et implémenter les **relations** entre ces classes. Il faudra aussi modifier la classe Vente pour gérer la relation Vente/Cheval.

La classe Cheval contiendra toutes les **propriétés primitives** (id, sire...), les getters, setters et au moins un constructeur vide. La classe Lot contiendra toutes les propriétés primitives (id, prixDepart), les **getters**, **setters** et au moins un **constructeur** vide.

Relation Lot/vente La classe Vente sera modifiée pour y ajouter une ArrayList de Lot (+ getter/setter et méthode add). L'ajoute de la **propriété complexe** Vente dans Lot n'est pas nécessaire pour le moment.

Relation Lot/Cheval On ajoutera dans la classe Lot une propriété complexe Cheval (+ getter/setter). La relation inverse consistant à ajouter dans la classe Cheval, une ArrayList de Lot n'est pas nécessaire pour le moment.

Tests : Les classes et les relations seront testées dans des classes dédiées : VenteTest, LotTest, ChevalTest.

Couche applicative DAO :

Dans la classe existante VenteDAO, nous ajouterons une **méthode statique** nommée getLesLotsParVente qui renverra une ArrayList de Lot et qui prendra en **paramètres** la connexion à la base de données et le code de la vente. Cette méthode utilise la connexion existante à la base de données et effectuera une **requête préparée** d'interrogation qui récupèrera en base de données les informations des lots (et donc du cheval lié à chaque lot) pour le code vente passé en paramètre. La requête SQL utilisera 3 **tables** (donc 2 **jointures**).

Tests :

La requête SQL sera d'abord réalisée et testée dans mysql. La méthode sera testée dans la classse de tests des DAO.

Partie Vues :

Maquette : dessin avec informations à afficher. Structure et charte graphique à respecter.

Modification vue listerLesVentes.jsp Pour chaque vente, on ajoutera un lien (url) redirigeant vers la nouvelle url : ServletVentes/listerLesLotsPourUneVente….et passant en paramètre de l'url le code de la vente.

Nouvelle page : Dans le dossiers Vues, nous ajouterons une nouvelle page nommée listerLesLotsPourUneVente.jsp. Cette page récupèrera de la **Request** (requête http) l'ArrayList des lots pour la vente sélectionnée. Pour chaque Lot, on affichera sous forme de tableau html les informations de chaque lot (donc de chaque cheval).

Déclaration dans web.xml Pour que cette page soit connue de l'application, il faut déclarer l'url redirigeant vers la vue dans le fichier de configuration web.xml entre les tags <servlet-mapping>. La nouvelle url sera liée à ServetVente.

Tests ; on s'assure que l'url redirige vers la bonne page.

Partie Controleur :

On utilisera la **servlet** existante ServletVente pour gérer cette fonctionnalité. Dans la méthode doGet, on ajoutera un test pour savoir si l'Url correspond à celle déclarée ci-dessous. Si tel est le cas : - on récupère l'ArrayList de Lot avec la méthode getLesLotPourUneVente de la classe VenteDAO créée précédemment - on insère cette ArrayList dans la request Http - on redirige vers la page jsp crée ci-dessous.

TESTS :

Une fois validée, la **branche** sera **fusionnée** à la branche principale et **publiée** sur gitHub dans le projet xxxx/

Une fois validée par le client, la fonctionnalité sera livrée en production.

La démarche de livraison est décrite dans le document xxxxx.

DOCUMENTS A MODIFIER

- diagramme de classe : pas de mise à jour
- diagramme de cas d'utilisation : mise à jour
- modèle logique de données : à modifier
- interfaces graphiques : maquettes à ajouter
- documentation de spécifications : mise à jour
- plan de tests : mise à jour