**Travail pratique 3 – Clinique vétérinaire**

**Liste de contrôle**

***Noms étudiants-es: George Panu, Vincent Boies \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

**Exigences de fonctionnalités**

***Si vous inscrivez qu’une fonctionnalité est réalisée, cela signifie qu’elle est opérationnelle, complète et testée. Inscrivez les initiales de la ou des personnes qui l’ont réalisée.***

***IMPORTANT :*** *au fur et à mesure que vous développez des user stories, vous devez injecter des données fictives (« seed data ») dans les dépôts de données (les « repositories ») pour les classes entités ajoutées de la couche modèle.*

Gestion des clients

|  |  |
| --- | --- |
| **Réalisé par** | **User Story** |
| XX | On doit pouvoir consulter la liste de tous les clients (no, nom, prénom, téléphone1), triée sur le nom des clients.  1 Téléphone : type String (sans validation) |
| XX | On doit pouvoir consulter la liste des clients (no, nom, prénom, téléphone), triée sur le téléphone des clients. |
| XX | À partir du numéro de client, on doit pouvoir rechercher un client et consulter toutes les informations du client (no, nom, prénom, téléphone, courriel). |
| XX | Lorsqu’on consulte la fiche d’un client, on doit pouvoir modifier toutes les informations du client, sauf son no. Il faut toujours indiquer un nom, un prénom et un téléphone.  On informe l’utilisateur que les modifications ont bien été effectuées. |
| XX | On doit pouvoir ajouter un nouveau client en fournissant obligatoirement les informations de base suivantes : nom, prénom et téléphone.  Un no identifiant (unique) est attribué automatiquement à chaque nouveau client.  On informe l’utilisateur que l’ajout a bien été effectué. |
| XX | On doit pouvoir rechercher un client par son nom. Si la recherche trouve plusieurs clients, on affiche le no, nom, prénom et téléphone des clients trouvés. |
| XX | Au démarrage de l’application, des données fictives, pertinentes et réalistes, permettent d’utiliser correctement les fonctionnalités reliées aux clients. |

Gestion des chiens

|  |  |
| --- | --- |
| **Réalisé par** | **User Story** |
| XX | On veut pouvoir consulter la liste de tous les chiens (no, nom et race du chien, nom et prénom du propriétaire), en ordre alphabétique sur le nom du chien. |
| XX | À partir du numéro d’un chien, on doit rechercher un chien et consulter toutes les informations sur le chien (no, nom, race) et toutes les informations du propriétaire (no, nom, prénom, téléphone). |
| XX | Lorsqu’on consulte la fiche d’un chien, on doit pouvoir modifier le nom et la race du chien.  On informe l’utilisateur que les modifications ont bien été effectuées. |
| XX | Lorsqu’on consulte la fiche d’un chien, on doit pouvoir supprimer le chien de la liste de la clinique. Attention, il ne faut pas supprimer le propriétaire.  On demande une confirmation à l’utilisateur avant d’effectuer la suppression, et d’en informer l’utilisateur. |
| XX | On doit pouvoir inscrire un chien : il faut indiquer son nom et sa race ainsique son propriétaire. Le propriétaire doit déjà faire partie de la liste des clients de la clinique.  Un no identifiant (unique) est attribué automatiquement à chaque nouveau chien. Il n’est pas possible d’inscrire un chien qui n’a pas de propriétaire.  On informe l’utilisateur que l’ajout a bien été effectué. |
| XX | On doit pouvoir rechercher les chiens d’une même race. On affiche le no et le nom du chien, ainsi que le no, le nom, le prénom et le téléphone du propriétaire. |
| XX | Au démarrage de l’application, des données fictives, pertinentes et réalistes, permettent d’utiliser correctement les fonctionnalités reliées aux chiens. |

EXCEPTION QUAND AJOUTER CHIEN, PASSER UN ID DE PROPRIETAIRE INEXISTANT

**Exigences de qualité**

**Application**

* Il n’y a aucune erreur ni avertissement lors de la compilation de l’application.  
  Voir commentaires dans DogDataSeed.java
* L’application s’exécute sans erreurs.  
  Si non, précisez dans quels cas ou quel contexte l’application plante : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
* Au démarrage de l’application, un « data seeder » peuple les « repositories » avec des données fictives, valides et représentatives pour chacune des classes entités de la couche modèle.

**Interface**

* L’interface est en français.
* L’interface est dans un français de qualité.
* L’interface est convenable. Et très belle 😊

**Conception objet**

* L’architecture MVC est respectée pour réaliser l’application.
* Il y a au moins 1 contrôleur pour chaque entité de la couche modèle.
* Les vues ne communiquent jamais avec la couche modèle, et vice-versa.
* Les contrôleurs assurent la communication entre les vues et la couche modèle, notamment par l’utilisation de DTO (Data Transfert Objects).
* Encapsulation : toutes les propriétés non finales d’une classe sont privées.
* Le principe de conception suivant est respecté: Single Responsability Principle.
* Le principe de conception suivant est respecté: Loi de Demeter.
* Le principe de conception suivant est respecté: Query-Command Separation.
* Le principe de conception suivant est respecté: Tell, don’t ask.
* Le principe de conception suivant est respecté: programmer avec des interfaces (ou des classes abstraites) pas avec leurs implémentations.
* Le patron de conception suivant est utilisé : Repository pattern.
* La classe Repository est générique (classe typée ou « template »)
* Les patrons de conception suivants ont aussi été utilisés :
  + Stratégie. Précisez : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
  + Fabrique. Précisez : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Code**

* La classe Repository utilise une collection de type Map.
* La méthode forEach() de la classe Collection est utilisée, avec des expressions lambdas, pour parcourir des collections.

Fait d’une autre façon

* Nommage : toutes les méthodes, les classes, les attributs, les variables et les constantes ont un nom significatif indiquant leur raison d’être ou leur contenu selon le cas.
* Il n’y a pas de conditions complexes (elles sont encapsulées).
* Il n’y aucun chiffre magique.
* Il n’y a aucune chaine de caractères magique.
* Les commentaires au travers du code sont réellement informatifs.
* Les conventions Java et les normes de programmation, telles que montrées, sont respectées.
* Toutes les classes sont petites (moins de 100 lignes).

Respecté le plus possible : classe ClientController 150 lignes, car contient 20 méthodes, classe DogController 150 lignes, car contient 18 méthodes.

* Toutes les méthodes sont les plus petites possibles (moins de 15 lignes).
* Les méthodes ont peu de paramètres.
* L’indentation du code est irréprochable.
* Dans l’ensemble, le code est clair, facile à lire et à comprendre.
* Dans l’ensemble, le code est simple (KISS : Keep it Simple, Stupid)

**Exceptions**

* Des exceptions « Runtime » sont utilisées pour gérer les erreurs de programmation.
* Les exceptions sont les plus spécifiques possibles.

**Tests**

* Toutes les méthodes de tests ont un nom significatif indiquant ce qui est testé, quitte à avoir un nom de méthode long.
* Chaque méthode de test ne teste qu’un seul comportement.
* Chaque méthode de test respecte l’organisation : Arrange/Act/Assert.
* Toutes les classes des couches modèle et contrôleur sont testées (entités, « repositories », contrôleurs et autres classes).
* Les vues et les DTO sont les seules classes qui ne sont pas testés unitairement.
* Des objets simulés (« mocks ») ont été utilisés dans les tests au besoin.  
  Précisez dans quelles classes de tests : \_\_\_DogControllerTest, ClientControllerTest\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* Tous les tests s’exécutent sans erreurs.
* Combien de tests sont exécutés ? \_\_\_17\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_