Système de gestion de listes

On se propose dans ce TP de réaliser un système de gestion de listes. Ces listes sont génériques, et peuvent servir pour lister les choses à faire, les courses, les idées de cadeau, les choses que vous avez prêtées, etc.

Spécification fonctionnelle

Une liste possède un titre, une description, et un ensemble d'éléments. Un élément possède une date de création, une date de dernière modification, un titre et une description.

L'utilisateur peut créer et supprimer des listes. Il peut modifier le titre et la description d'une liste existante. Il peut aussi ajouter des éléments à une liste, en supprimer, et modifier le titre et la description d'un élément existant. Les dates de l'élément ne peuvent pas être modifiées par l'utilisateur.

Spécification technique

Nous allons réaliser ce système sous la forme d'une application Java autonome, c'est-à-dire qu'elle embarquera le serveur web. Ceci facilitera sa mise en place. Dans l'environnement de développement, la base de données sera aussi embarquée. Le système doit cependant être conçu de manière à permettre l'utilisation d'un serveur de donnée une fois en production si nécessaire.

Pour la base de donnée, nous utiliserons H2 [1] qui est une base de donnée relationnelle embarquée purement en Java. Afin de ne pas formuler les requêtes directement en JDBC, et faciliter la création des objets du modèle, nous utiliserons sql2o [2].

1. https://h2database.com/
2. https://www.sql2o.org/

La génération du HTML se fera à l'aide de FreeMarker [3] qui est un système de *template*.

La partie contrôleur, qui analyse les requêtes et les fait correspondre à des actions sur les ressources, se fera avec Spark [4].

Les *jar* de ces bibliothèques sont fournies dans l'archive du TP. Il suffit de l'extraire et ajouter le dossier à votre `CLASSPATH`. Vous pouvez aussi bien sûr utiliser Maven ou Gradle pour gérer vos dépendances.

Travail

Pour ce TP, vous remettrez le code source de votre travail (application, tests unitaires et fonctionnels, etc.). Ce code doit en outre contenir un script de compilation/exécution (maven, gradle, bash...) permettant de lancer facilement le logiciel et les tests. Vous remettrez également un bref résumé au format PDF expliquant l'architecture de votre système, votre modèle de donnée et la démarche suivie, en donnant des éléments de réponses aux questions ci-dessous. Vous préciserez bien les noms des membres du groupe ! Tous ces documents seront remis sur l'ENT sous la forme d'une archive au format ZIP.

Exercice 1 (Conception) Avant de se lancer dans la réalisation, il faut concevoir certains éléments.

Question 1 (Données) *:* Réfléchissez au modèle de données que vous allez mettre en œuvre.

Question 2 (Resources) *:* Définissez les différentes ressources que votre service exposera, ainsi que les interactions possibles avec celles-ci.

Question 3 (Espaces de noms) *:* Définissez les différents schéma d'URL qui identifierons vos ressources.

Exercice 2 (Modèle et accès aux données)

Question 1 (Base de donnée) *:* Initialisez la base de donnée, en créant les tables nécessaires. Vous pouvez pour cela utiliser l'outil d'administration fourni avec H2 (`java -jar lib/h2-1.4.196.jar`). Vous utiliserez un accès type fichier à la base, et non un accès réseau.

3. https://freemarker.apache.org/
4. http://sparkjava.com/

Question 2 (Objets du modèle) *:* Créez les classes nécessaires pour représenter vos éléments métiers. Pour faciliter la suite, elles suivront la convention *beans*, c'est-à-dire auront un constructeur par défaut (sans paramètre) et des accesseurs sur tous les attributs.

Question 3 (DAO) *:* En utilisant sql2o, créez un DAO (*Data Access Object*). C'est un objet (en général statique ou singleton) exposant des méthodes qui permettent d'effectuer des requêtes sur la base de données et retournent des objets métiers. Par exemple, on pourrait avoir une méthode

```
public static List<Item> getItemsByCreateDate(Date createDate) {
  // execution d'une requete SQL...
}
```

(ce n'est qu'un exemple)

Exercice 3 (Représentations) En utilisant FreeMarker, créez des vues pour vos différents types de ressources.

Exercice 4 (Interactions) Maintenant que vous disposez des différents éléments, créez le lien dans le contrôleur avec Spark. Ceci passe par la définition des *routes*, qui décrivent les actions à effectuer pour chaque méthode HTTP et chaque schéma d'URL

Exercice 5 (Ajout de fonctionnalités)

Question 1 *:* Assurez vous de gérer correctement les requêtes conditionnelles et les en-têtes de cache.

Question 2 *:* Ajouter un statut optionnel sur les éléments des listes. Le statut peut être :

— à faire,
— fait,
— supprimé.

Il pourra se matérialiser dans l'interface par une case à cocher, ou rayer l'élément.

Question 3 *:* Ajouter la possibilité pour une liste de contenir des sous-listes

Question 4 *:* Ajouter la possibilité pour un élément d'appartenir à plusieurs listes.

Question 5 *:* Ajouter la notion d'étiquette (*tag*) sur un élément d'une liste. D'un point de vue interface, chaque étiquette implique une liste « virtuelle » de tous les éléments marqué par celle-ci.

Question 6 *:* Ajouter la gestion des utilisateurs.

Question 7 *:* Ajouter la possibilité de partager une liste entre plusieurs utilisateurs.