Références

- HTTP servlets : http://docs.oracle.com/javaee/1.2.1/api/javax/servlet/http/package-summary.html
- Jetty : https://eclipse.org/jetty/, base de donnés H2 : http://www.h2database.com/
- parser json :
  - http://developer.android.com/reference/org/json/JSONObject.html
  - http://developer.android.com/reference/org/json/JSONArray.html

Exercice 60 – MVC

Le but de ce TP est de construire le côté serveur d'une application web, en utilisant l'approche modèle-vue-contrôleur. En fait, il s'agit de l'application appelée depuis le client dans l'exercice 50.

Pour mieux découvrir les mécanismes mis en jeu dans un serveur web, nous vous proposons d'adopter une démarche « artisanale », et implémenter à la main des fonctionnalités habituellement gérées par des bibliothèques standard.

A Spécifications fonctionnelles

A.1. Objets manipulés

Le service web manipule deux types d'objets : des tags (tags) et des marque-pages (bookmarks). Il ne communique qu'au format JSON (type de données `application/json`).

- Un tag possède un identifiant unique attribué par le système (`id`) et un nom (`name`). En JSON, un tag est représenté par un objet `{ 'id': <id>, 'name': <name> }`. Si l'identifiant n'est pas connu (p.ex. lors de la création d'un nouveau tag, et avant que le système lui attribue son numéro unique), on peut spécifier juste `{ 'name': <name> }`.
- Une liste de tags est représentée par un tableau : `[ tag1, tag2,..., tagn ]`.
- Un marque-page possède un identifiant unique attribué par le système (`id`), un titre (`title`), une URL (`link`), une description optionnelle (`description`) et une liste de tags attachés (`tags`), cette liste pouvant être vide. Sa représentation en JSON est : `{ 'id': <id>, 'title': <title>, 'description': <description>, 'link': <link>, 'tags': <liste-de-tags> }`.
- Enfin, une liste de marque-pages est un tableau : `[ bookmark1, bookmark2,...]`.

A.2. Service web à implémenter

Les URLs du service sont de la forme http://serveur/bmt/<User>/<Path>[?Param].

Le paramètre Param permet de spécifier le nom d'une méthode du protocole HTTP sous la forme `x-http-method=<nom-de-methode>`. Cette construction permet de contourner la limitation des navigateurs, qui proposent seulement des requêtes GET et POST : en spécifiant par exemple http://serveur/bmt/<User>/<Path>?x-http-method=put dans la barre d'adresse d'un navigateur, la requête GET résultante sera interprétée par le service web comme s'il s'agissait d'une requête PUT.

La composante <User> des URLs désigne l'utilisateur (plus précisément son login). Dans le TP, on considère **trois utilisateurs possibles : titi, tata et toto**.

La composante <Path> des URLs peut prendre plusieurs valeurs, présentées dans le tableau suivant. Les cellules grisées correspondent à des appels interdits (erreur 405). Pour les autres cas, le code de retour attendu est indiqué (CR).

<Path>	GET	POST	PUT	DELETE
/bookmarks	donne la liste des marque-pages CR 200	crée un nouveau marque-page, dont la définition aura été passée dans le paramètre json CR 201 ou 304		
/bookmarks/<bid>	donne la description du marque-page dont l'ID est <bid> CR 200		modifie le marque-page avec la nouvelle définition passée en paramètre json CR 204 ou 403	efface le marque-page dont l'ID est <bid> CR 204 ou 403
/tags	donne la liste des tags CR 200	crée un nouveau tag à partir de la définition passée par le paramètre json CR 201 ou 304		
/tags/<tid>	donne la description du tag dont l'ID est <tid> CR 200 ou 404		modifie le tag avec la nouvelle définition passée en paramètre json CR 204 ou 403	efface le tag dont l'ID est <tid> CR 204 ou 403
/tags/<tid> /bookmarks	donne la liste des marque-pages attachés au tag dont l'ID est <tid> CR 200			
/tags/<tid> /bookmarks/<bid>	donne l'information si le marque-page <bid> est attaché au tag <tid> : oui CR 204, non CR 404		attache le marque-page <bid> au tag <tid> CR 204, 304 ou 403	efface l'attachement du tag <tid> au marque-page <bid> CR 204 ou 403

Quelques exemples :

- l'URL http://serveur/bmt/<User>/tags?x-http-method=get permet de récupérer la liste de tous les tags
- l'URL http://serveur/bmt/<User>/tags/12?x-http-method=get donne la description du tag dont l'identifiant unique est « 12 »
- l'URL http://serveur/bmt/<User>/tags/12?x-http-method=put &json={'id':12,'name':"toto"} permet de modifier le nom du tag « 12 » à « toto »
- l'URL http://serveur/bmt/<User>/tags/12?x-http-method=delete efface le tag « 12 »

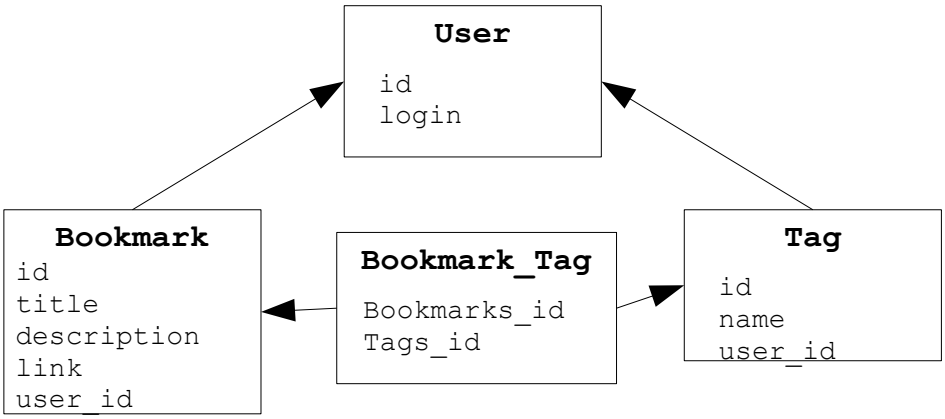
En outre, pour faciliter les tests, deux autres URLs sont fournies :

- http://serveur/bmt/<User>/clean?x-http-method=post permet d'effacer tous les tags et tous

les marque-pages

- http://serveur/bmt/<User>/reinit?x-http-method=post permet de réinitialiser les tags et les marque-pages (quelques objets sont fournis par défaut)

A.3. Schéma de la base de données



Il y a quatre tables dans la base de données, représentées schématiquement ci-dessus. Les flèches représentent des clés étrangères.

- Table User : représente les utilisateurs. Chaque utilisateur possède un id (clé primaire) et un login (unique au sein de la table).
- Table Bookmark représente les marque-pages. Les colonnes de la table sont : id (clé primaire), title, description et link (textes jusqu'à 255 caractères), puis user_id qui stocke l'id de l'utilisateur possédant le marque-page (c'est la clé étrangère de la table). Il existe une contrainte d'unicité sur le couple user_id-link.
- Table Tag représente des tags. La colonne id constitue la clé primaire, la colonne name est la chaîne de caractères qui contient le nom du tag et user_id qui stocke l'id de l'utilisateur possédant le marque-page (c'est la clé étrangère de la table). Il existe une contrainte d'unicité sur le couple user_id-name.
- Table Bookmark_Tag permet d'associer des tags à des marque-pages. Les deux colonnes, Tags_id et Bookmarks_id sont des clés étrangères qui contiennent les ids de Tag, respectivement de Bookmark.

  Remarque : Dans votre implémentation certaines requêtes peuvent entre en contradiction avec des contraintes d'intégrité, dans ce cas le code de retour HTTP pourra être 500 ou 403.

Voici la définition formelle de ces tables :

```
CREATE TABLE `user` (
  `id` bigint(20) NOT NULL auto_increment,
  `login` varchar(255) default NULL,
  PRIMARY KEY  (`id`),
  UNIQUE KEY `login` (`login`)
)

CREATE TABLE `bookmark` (
  `id` bigint(20) NOT NULL auto_increment,
  `description` longtext,
  `link` varchar(255) default NULL,
  `title` varchar(255) default NULL,
  `user_id` bigint(20) default NULL,
  PRIMARY KEY  (`id`),
  UNIQUE KEY (`user_id`,`link`),
  FOREIGN KEY (`user_id`) REFERENCES `user` (`id`)
)
```

```
CREATE TABLE `tag` (
  `id` bigint(20) NOT NULL auto_increment,
  `name` varchar(255) default NULL,
  `user_id` bigint(20) default NULL,
  PRIMARY KEY  (`id`),
  UNIQUE KEY (`user_id`,`name`),
  FOREIGN KEY (`user_id`) REFERENCES `user` (`id`)
)
CREATE TABLE `bookmark_tag` (
  `bookmarks_id` bigint(20) NOT NULL,
  `tags_id` bigint(20) NOT NULL,
  PRIMARY KEY  (`bookmarks_id`,`tags_id`),
  FOREIGN KEY (`tags_id`) REFERENCES `tag` (`id`),
  FOREIGN KEY (`bookmarks_id`) REFERENCES `bookmark` (`id`)
)
```

B Environnement de travail

On vous propose d'implémenter le web service en vous appuyant sur la technologie « servlet HTTP » déployée dans le conteneur Jetty, et sur la base de données H2[1] .

En pratique, téléchargez le fichier sur Chamilo Exercice-60.tar.gz et déballez-le sur votre machine. Ce fichier installe le répertoire Exercice-60 contenant plusieurs fichiers et sous-dossiers :

- Le fichier TP5.h2.db est votre base de données (les tables et leur contenu sont dans ce fichier).
- Le fichier start.jar sert à lancer le serveur Jetty
  - la commande à utiliser est **`java -jar start.jar jetty.port=8080`**
  - votre TP sera alors en ligne à l'adresse http://localhost:8080/bmt/...
  - en cas de besoin, on peut opter pour un autre port en modifiant la valeur du paramètre jetty.port
  - il faut utiliser une machine virtuelle Java 6 ou supérieure
- le dossier webapps/bmt est votre dossier de travail : c'est là que l'application web que vous implémenterez est déployée
  - webapps/bmt/src contient les sources (*.java) de votre TP
  - webapps/bmt/WEB-INF/classes doit contenir les classes compilées par vos soins (*.class) pour qu'elles soient prises en compte
  - **attention !** (re)compiler une classe Java et mettre le fichier byte-code dans webapps/bmt /WEB-INF/classes ne suffit pas à le faire prendre en compte par Jetty : il faut arrêter le serveur et le relancer pour que les classes (re)compilées soient effectivement utilisées

Votre espace de travail se réduit donc aux deux dossiers `webapps/bmt/src` et `webapps/bmt/WEB-INF/classes` : en dehors de ces deux endroits, vous ne devez pas faire de modifications ! Vous pouvez utiliser Eclipse pour éditer et compiler (simplement en sauvegardant vos fichiers), et lancer dans un terminal le serveur jetty à chaque changement.

Pour utiliser Eclipse, créer un nouveau projet en spécifiant le chemin vers `webapps/bmt`, ensuite ajouter les .jar présents dans `lib/` et dans `webapps/bmt/WEB-INF/lib`. Enfin la sauvegarde des fichiers devraient générer les fichiers compilés .class dans le répertoire `webapps/bmt/WEB-INF/classes/` . Il ne reste plus qu'à relancer le serveur.

1 H2 est une base de données légère, qui s'appuie sur un unique fichier. Elle a été choisie pour sa simplicité, car la persistance n'est pas ce qui nous intéresse dans ce TP.

C Squelette de l'application web

On vous fournit un début de l'application web.

- L'entrée se fait par la classe `GlobalServlet` : cette servlet interagit avec le conteneur (Jetty) dans le sens où elle reçoit toutes les requêtes HTTP venant sur le serveur avec le préfixe http://localhost:8080/bmt/. La servlet se contente de décoder la requête entrante (elle décompose le chemin exact, elle décode les paramètres d'appel et elle détermine la méthode HTTP effectivement appelée), puis elle donne la main à la classe `Dispatcher`. **La classe `GlobalServlet` n'est pas à modifier**, seulement à comprendre.
- La classe `Dispatcher` sert à aiguiller le traitement de la requête vers la méthode qui correspond le mieux au chemin (path) de la requête.
- A titre d'exemple, la classe `SpecialActions` vous est donnée implémentée : elle réalise le traitement des actions *clean* et *reinit*.
- La classe `DBConnection` est un fournisseur (factory) de connexions à la base de données. Elle vous est donnée, vous n'avez qu'à l'utiliser en l'état. Veillez à bien fermer les connexions que vous ouvrez quand vous faites une requête, sans quoi vous saturerez vite le serveur. Pour ce faire vous pouvez utiliser les blocs `try-finally` pour encadrer le code exécutant vos requêtes.
- La classe `User` fournit le modèle d'un utilisateur. La classe `UserDAO` permet de retrouver les objets de type `User` dans le stockage persistant, et le crée si besoin s'il n'existe pas.

D Travail à faire

D.1. Création de nouveau tag

Implémenter le traitement des requêtes POST à l'adresse http://serveur/bmt/<User>/tags : dans la classe `Tags`, compléter la méthode `handleTagList` à l'endroit indiqué dans le code.

Cette action nécessitera de développer de nouvelles méthodes dans la classe TagDAO, p.ex.

- `Tag getTagByName(String name, User user)` pour vérifier si l'utilisateur ne possède pas déjà un tag avec le même nom (ce qui est un cas d'erreur)
- `void saveTag(Tag tag, User user)` pour sauvegarder un tag

D.2. Autres modifications de tags

Implémenter les autres méthodes de la classe `Tags` (`handleTag`, `handleTagBookmarks`, `handleTagBookmark`) en vous inspirant du point précédent. Ces méthodes servent à gérer les autres appels aux autres types d'adresses dont le préfixe est http://serveur/bmt/<User>/tags (cf. les appels à ces méthodes dans la classe `Dispatcher`).

D.3. Gestion des marque-pages

Implémenter les méthodes pour les appels au préfixe http://serveur/bmt/<User>/bookmarks : après avoir modifié la classe `Dispatcher`, vous pouvez p.ex. créer les classes `Bookmarks` et `BookmarkDAO`.

D.4. Tests

Proposez un moyen de tester au moins certaines parties de l'application. Vous pouvez vous baser sur le client écrit au TP précédent, ou sur un autre mécanisme.

E Résultat attendu

Il suffit de rendre le contenu du répertoire **webapps/bmt/src**, et la description et l'implémentation de la réponse à la question 4.