SCIENCES SORBONNE UNIVERSITÉ

SRCS : Systèmes Répartis Client/Serveur

TME 9 : Compagnie aérienne

Objectifs pédagogiques

- Web Service REST
- Utilisation du framework Talend Restlet

Objectif

L'objectif est de programmer un web service permettant de gérer (très sommairement) le système d'information d'une compagnie aérienne. La compagnie possède une flotte d'avions et un ensemble d'emplacements dans des aéroports internationaux. Elle vend des vols à des clients.

Modèle de données

Un avion (classe `Aricraft` fournie dans les ressources) est caractérisé par :
- une immatriculation (identifiant unique)
- le nom du modèle (ex : Airbus-A380)
- une capacité en passagers

Un aéroport (classe `Airport` fournie dans les ressources) est caractérisé par :
- un code AITA, (sur trois lettres) qui permet d'identifier l'aéroport au niveau mondial
- un nom
- la ville/la métropole qu'il dessert
- la région dans lequel il se situe
- son pays

Un vol (classe `Flight` fournie dans les ressources) est caractérisé par
- un identifiant interne à la compagnie
- un aéroport de départ, desservi par la compagnie
- un aéroport d'arrivée, desservi par la compagnie
- une date et une heure de départ
- une date et une heure prévue d'arrivée
- l'ensemble des passagers associés à leur numéro de place respective
- un avion de la compagnie

Enfin, un passager (classe `Passenger`, fournie dans les ressources) caractérisé et identifié par un nom et un prénom. Pour simplifier, on considérera qu'il n'existe pas d'homonyme. Nous prendrons l'hypothèse forte que nous sommes dans un monde idéal ne nécessitant aucun mécanisme de sécurité.

Spécification du service

Le service sera accessible par deux ports d'écoute via le protocole HTTP 1.1 : un port d'écoute pour les utilisateurs classiques et un port d'écoute pour les administrateurs. Nous ne traiterons pas ici les problématiques d'authentification. Toute requête d'écriture sur la base de données se fera via les méthodes HTTP *POST* ou *PUT* et que toute requête de lecture se fera exclusivement par la méthode HTTP *GET*. Le format de transmission est le JSON. Les requêtes d'écriture se font uniquement via le port d'administration. Toute requête d'administration a */admin* comme préfixe du path de son

URI. En revanche les requêtes de lecture peuvent aussi bien se faire via le port classique que via le port d'administration. Dans tout les cas, il n'est pas possible d'accéder au service via le port classique et de préfixer le path par */admin*

Ressources offertes par le service :

- */admin/airports* :
  - ◇ accessible exclusivement par le port d'administration. Si on tente un accès via le port classique, une réponse HTTP ayant le statut *404 Not Found* est renvoyée.
  - ◇ *POST* ou *PUT* : ajouter des aéroports à la base de données
  - ◇ *GET* : récupérer la liste des aéroports desservis par la compagnie
- */airports* :
  - ◇ accessible sur les deux ports
  - ◇ *POST* renvoie une réponse HTTP ayant le statut *405 Method Not Allowed*
  - ◇ *GET* : récupérer la liste des aéroports desservis par la compagnie
- */admin/aircrafts* et */aircrafts* qui ont respectivement les même spécifications que */admin/airports* et */airports* mais pour les avions.
- */admin/flights* :
  - ◇ accessible exclusivement par le port d'administration. Si on tente un accès via le port classique, une réponse HTTP ayant le statut *404 Not Found* est renvoyée.
  - ◇ *POST* ou *PUT* : ajouter un nouveau vol à la base de données. Attention un seul vol est attendu dans la requête (et non une collection). Le serveur renvoie une réponse HTTP *412 Precondition Failed* (et par conséquent, ne modifie pas la base) si une des conditions suivantes est remplie :
    - · il existe déjà un vol du même identifiant dans la base
    - · l'aéroport de départ n'existe pas ou n'est pas desservi par la compagnie
    - · l'aéroport d'arrivée n'existe pas ou n'est pas desservi par la compagnie
    - · l'avion est déjà utilisé dans un autre vol dont les horaires entrent en conflit (on peut savoir si deux vols entrent en conflit grâce à la méthode `isInConflict` de la classe `Flight`).
  - ◇ *GET* : récupérer la liste des vols existant dans la base. On pourra raffiner les critères de recherche grâce à la partie query de l'URI en renseignant le paramètre `to` et/ou `from` pour spécifier respectivement un aéroport d'arrivé et de départ (pour simplifier, on ne traitera pas la recherche par date).
- */flights* :
  - ◇ accessible sur les deux ports
  - ◇ *POST* renvoie une réponse HTTP ayant le statut *405 Method Not Allowed*
  - ◇ *GET* : mêmes spécifications que `/admin/flights` pour la méthode `GET`
- */admin/flight/<id_vol>/passenger* :
  - ◇ accessible exclusivement par le port d'administration. Si on tente un accès via le port classique, une réponse HTTP ayant le statut *404 Not Found*
  - ◇ *POST* ajoute un passager à un vol donné. Le numéro de place du passager se renseignera grâce à la partie query de l'URI avec le paramètre `place`.
- */admin/flight/<id_vol>/passengers* :
  - ◇ accessible exclusivement par le port d'administration. Si on tente un accès via le port classique, une réponse HTTP ayant le statut *404 Not Found* est renvoyée.
  - ◇ *GET* : obtenir la liste de tous les passagers d'un vol donné
- */admin/flight/<id_vol>/place* :
  - ◇ accessible exclusivement par le port d'administration. Si on tente un accès via le port classique, une réponse HTTP ayant le statut *404 Not Found* est renvoyée.
  - ◇ *GET* : obtenir la place la place d'un passager dont le nom et le prénom sont passés dans la partie query de l'URI (paramètre *firstname* pour le prénom, paramètre *lastname* pour

le nom).

Travail à effectuer

Programmer un web service REST à l'aide du framework Restlet qui respecte les spécifications décrites ci-dessus. L'implémentation est libre. Vous utiliserez le fichier de test JUnit `AirlineServiceTest`. L'ensemble des méthodes de tests de ce fichier sont exécutées dans l'ordre de leur déclaration. Elles respectent une logique de progression incrémentale c'est-à-dire que pour valider une méthode de test, il est nécessaire de valider l'ensemble des méthodes de test précédentes. Le fichier de test considère lors de son lancement que la base de données de la compagnie est vide. Les différentes méthodes permettant de remplir cette base au fur et à mesure, il vous est conseillé, pour simplifier, de stocker les données uniquement en mémoire vive. Le fichier de test utilise une interface `SRCSWebService` qui offre les méthodes suivantes :

- `String getName();` renvoie le nom du web service
- `void deploy();` permet de démarrer le web service
- `void undeploy();` permet d'arrêter le web service

Votre classe principale du web service devra donc implanter cette interface afin de faire le lien entre le fichier de test et le déploiement de votre code. Il sera également nécessaire de définir une méthode statique `buildAirline(String name, int portuser, int portadmin)` dans une classe `SRCSWebServiceFactory` qui permet de construire une instance de votre classe principale. Si vous êtes sur une version supérieure à java 8, vous aurez besoin d'inclure dans vos dépendance le fichier jar *jaxb-api-2.3.1.jar* (fourni dans les ressources du TP).