Partie 1 : Questions de cours (8 points)

Q1 : Cochez les méthodes HTTP qui sont sûres (safe) dans la liste suivante
■ GET
■ HEAD
■ OPTIONS
□ PUT
□ DELETE
□ POST

Q2 : Une méthode HTTP est idempotente lorsque
□ l'effet obtenu sur le serveur est le même que la requête soit exécutée 0 à N fois
■ l'effet obtenu sur le serveur est le même que la requête soit exécutée 1 à N fois
□ le serveur ne garantit pas que l'effet de la requête sera le même si elle est exécutée à nouveau
□ la requête modifie l'état du serveur

Q3 : En effectuant une requête HTTP POST vers un serveur, vous obtenez le code statut 461 en réponse. Que pouvez-vous en déduire ?
□ rien du tout
□ il y a un bug dans le serveur car le code 461 n'est pas répertorié dans la liste des codes HTTP
□ une erreur s'est produite après le traitement de la requête par le serveur
□ il faut émettre à nouveau la même requête plus tard
■ la requête n'a pas été acceptée par le serveur

Q4 : L'architecture REST est une architecture
□ orientée représentation
■ orientée ressource
□ orientée service
□ orientée objet

Q5 : Dans une architecture REST, une ressource est
□ une information échangée entre le client et le serveur
□ une donnée qui peut être mise en cache
■ une information qui porte un nom
□ un espace mémoire alloué par le système

Q6 : Parmi les propositions suivantes, quelles sont celles qui définissent un service Web RESTful
□ il offre une représentation des ressources au format JSON
■ il est accessible en utilisant le protocole HTTP
□ il n'utilise pas le format HTML
□ il est implémenté grâce à l'API JAX-RS

Q7 : Pourquoi un service Web RESTful doit-il être sans état (stateless) ?

Un service sans état signifie que le serveur n'a pas besoin de connaître les requêtes précédentes pour répondre à la requête entrante. Le serveur n'a donc pas besoin de sauvegarder le contexte des échanges avec un client. Un service sans état est plus simple à implémenter et à maintenir : il est plus robuste. Un service sans état est plus scalable : une partie des requêtes peut être prise en charge par un nouveau serveur sans modification du service et un client peut basculer d'un serveur vers un autre de manière transparente. Enfin un service sans état est plus évolutif : le client n'est pas contraint par le serveur à effectuer une séquence précise de requêtes pour aboutir à un état cohérent nécessaire à l'exécution d'une requête particulière.

Partie 2 : services Web RESTful (8 points)

Cette partie présente des exemples de requêtes/réponses HTTP décrivant l'API d'un service Web. Commentez et corrigez ces exemples pour rendre le service conforme à une architecture RESTful.

Ajout d'une série TV

```
PUT /series HTTP/1.1
Content-type: application/xml; charset=UTF-8
Content-length: 117
Host: serie-tv.fr

<serie>
 <titre>Game of Thrones</titre>
 <diffusion>2011</diffusion>
 <genres>
  <genre>fantasy</genre>
 </genres>
</serie>

HTTP/1.1 201 Created
```

La méthode HTTP PUT permet de créer ou de modifier une ressource identifiée par une URI. Pour une création, cela signifie que le client DOIT décider de l'URI de la ressource. Dans cette API, l'URI utilisée pour ajouter une série TV est donc incomplète.
Dans la réponse retournée par le serveur, il est possible d'ajouter l'en-tête Content-location pour donner l'URI de la ressource. Dans ce cas, cet en-tête est optionnel car le client a lui-même décidé de l'URI de la ressource.
Les en-têtes ETAG et Last-Modified sont fortement recommandés pour permettre les requêtes conditionnelles et la mise en cache côté client.

```
PUT /series/Game+of+Thrones HTTP/1.1
Content-type: application/xml; charset=UTF-8
Content-length: 117
Host: serie-tv.fr

<serie>
 <titre>Game of Thrones</titre>
 <diffusion>2011</diffusion>
 <genres>
  <genre>fantasy</genre>
 </genres>
</serie>

HTTP/1.1 201 Created
ETAG: 92194be5754d4b059ec
Last-modified: Fri, 3 Jan 2014 14:05:32 GMT
Content-location: http://serie-tv.fr/series/Games+of+Thrones
```

Il est possible également d'utiliser la méthode POST pour créer une ressource. Dans ce cas, c'est le serveur qui décidera de l'URI de la ressource. L'en-tête Content-location est obligatoire dans la réponse pour que le client connaisse l'URI de la ressource qu'il vient de créer.

```
POST /series HTTP/1.1
Content-type: application/xml; charset=UTF-8
Content-length: 117
Host: serie-tv.fr

<serie>
 <titre>Game of Thrones</titre>
 <diffusion>2011</diffusion>
```

```
 <genres>
  <genre>fantasy</genre>
 </genres>
</serie>
```

```
HTTP/1.1 201 Created
ETAG: 92194be5754d4b059ec
Last-modified: Fri, 3 Jan 2014 14:05:32 GMT
Content-location: http://serie-tv.fr/series/Games+of+Thrones
```

L'inconvénient de la méthode POST est qu'elle n'est pas idempotente. Si le client ne reçoit pas de réponse du serveur, il ne peut pas savoir si la ressource a été créée ou non et il n'a aucun moyen de connaître avec certitude l'URI de la ressource (puisqu'elle est déterminée par le serveur). On réservera donc l'usage de POST en création pour des cas très particuliers. Pour cette API, il n'y a pas de justification particulière à l'usage de POST, il faut donc privilégier la méthode PUT.

Modification d'une série TV

```
PUT /series/Game+of+Thrones HTTP/1.1
Content-type: application/xml; charset=UTF-8
Content-length: 142
Host: serie-tv.fr

<serie>
 <titre>Game of Thrones</titre>
 <diffusion>2011</diffusion>
 <genres>
  <genre>fantasy</genre>
  <genre>mediéval</genre>
 </genres>
</serie>
```

```
HTTP/1.1 204 No content
```

La méthode PUT fonctionne parfaitement pour la modification. Il est possible de fournir également une interface avec la méthode PATCH. L'inconvénient de la méthode PATCH est qu'il faut également définir une représentation permettant de préciser les éléments de la ressource que l'on souhaite modifier. Pour cette API, l'usage de PUT se révèle plus simple.
Dans la réponse, les en-têtes ETAG et Last-Modified sont fortement recommandés pour permettre les requêtes conditionnelles et la mise en cache côté client.

```
HTTP/1.1 204 No content
ETAG: c2ef3f6a1cd0e6450b
Last-modified: Fri, 3 Jan 2014 14:07:14 GMT
```

Recherche d'une série TV

```
POST /series/rechercher HTTP/1.1
Content-type: application/xml; charset=UTF-8
Content-length: 60
Host: serie-tv.fr

<critere maxReponse="1">
  <titre>Game of *</titre>
</critere>
```

```
HTTP/1.1 200 Ok
Content-type: application/xml; charset=UTF-8
Content-length: 142
```

```
<serie>
 <titre>Game of Thrones</titre>
 <diffusion>2011</diffusion>
 <genres>
  <genre>fantasy</genre>
  <genre>mediéval</genre>
 </genres>
</serie>
```

Une recherche est une activité qui normalement ne modifie pas le serveur et est idempotente. La méthode GET est parfaite pour ce type d'interface. Les critères de recherche peuvent être transmis *via* des paramètres de requête plutôt que dans le corps. Ainsi on peut créer virtuellement une infinité de ressources puisque les paramètres de requête font partie de l'URI.
Si on considère l'URI d'accès, on constate qu'il ne s'agit pas d'un service orienté ressource mais d'un appel RPC.
Cet appel n'est pas conforme à la contrainte REST de l'interface uniforme puisque le verbe utilisé est dans l'URI : /rechercher.
Le corps de la réponse ne devrait pas contenir directement une représentation des ressources mais des liens vers ces ressources. Retourner uniquement une représentation de la ressource présuppose que le client ne souhaite que consulter des informations.

```
GET /series?titre=Game+of+*&maxReponse=1 HTTP/1.1
Host: serie-tv.fr


HTTP/1.1 200 Ok
Content-type: text/plain; charset=UTF-8
Content-length: 42

http://serie-tv.fr/series/Games+of+Thrones
```

Ajout d'une saison

```
POST /serie/Game+of+Thrones/saison HTTP/1.1
Host: serie-tv.fr
Content-type: application/xml; charset=UTF-8
Content-length: 52

<saison>
  <diffusion>17/04/2011</diffusion>
</saison>

HTTP/1.1 201 Created
Content-location: http://serie-tv.fr/serie/Game+of+Thrones/saison/1
```

Le choix de la méthode POST est problématique car si le client ne reçoit pas de réponse du serveur, il ne peut pas déterminer si la ressource a été créée (il ne connaît pas son URI puisqu'elle est déterminée par le serveur dans ce cas). Pire, si un problème réseau survient, le client peut décider de retenter de créer une saison, ce qui peut aboutir à la création d'un doublon. Le plus simple est donc d'utiliser la méthode PUT et de laisser au client le soin de déterminer l'URI de la saison.
Dans cette API, le numéro de la saison sert à construire l'URI de la ressource. On peut donc déduire le numéro de la saison à partir de son URI. Cependant, un client ne doit pas avoir à connaître les détails de construction des URI. Il est donc préférable d'ajouter le numéro de la saison dans la représentation.
L'URI a changé (serie n'a plus de « s »). Il est préférable d'établir une arborescence correcte entre les ressources pour les rendre plus simple à appréhender pour les humains.
Comme pour la création et la modification d'une série, les en-têtes ETAG et Last-Modified devraient apparaître dans la réponse pour permettre les requêtes conditionnelles et la mise en cache côté client.

```
PUT /series/Game+of+Thrones/saisons/1 HTTP/1.1
Host: serie-tv.fr
```

```
Content-type: application/xml; charset=UTF-8
Content-length: 72

<saison>
  <numero>1</numero>
  <diffusion>17/04/2011</diffusion>
</saison>

HTTP/1.1 201 Created
ETAG: 98f71825a81f1a79ea
Last-modified: Fri, 3 Jan 2014 14:09:38 GMT
Content-location: http://serie-tv.fr/series/Game+of+Thrones/saisons/1
```

Consultation d'une serie TV

```
GET /serie/Game+of+Thrones HTTP/1.1
Host: serie-tv.fr

HTTP/1.1 200 Ok
Content-type: application/xml; charset=UTF-8
Content-length: 177

<serie>
 <titre>Game of Thrones</titre>
 <diffusion>2011</diffusion>
 <genres>
  <genre>fantasy</genre>
  <genre>mediéval</genre>
 </genres>
 <nombreDeSaison>1</nombreDeSaison>
</serie>
```

Dans la requête, il est recommandé d'utiliser l'en-tête Accept pour permettre la négociation de contenu avec le serveur. Ainsi, il est possible d'implémenter des représentations de la ressource dans un autre format que le format XML.
Comme pour la création et la modification d'une série, les en-têtes ETAG et Last-Modified devraient apparaître dans la réponse pour permettre les requêtes conditionnelles et la mise en cache côté client.
La consultation d'une série devrait permettre d'établir un lien avec les saisons associées. Une solution possible est d'utiliser un en-tête Link pointant vers la ressource des saisons. Cette nouvelle ressource retourne la liste des liens vers chaque saison.

```
GET /series/Game+of+Thrones HTTP/1.1
Host: serie-tv.fr
Accept: application/xml

HTTP/1.1 200 Ok
Content-type: application/xml; charset=UTF-8
Link: http://serie-tv.fr/series/Game+of+Thrones/saisons/; rel="saisons"
ETAG: c2ef3f6a1cd0e6450b
Last-modified: Fri, 3 Jan 2014 14:07:14 GMT
Content-length: 177

<serie>
 <titre>Game of Thrones</titre>
 <diffusion>2011</diffusion>
 <genres>
  <genre>fantasy</genre>
  <genre>mediéval</genre>
 </genres>
 <nombreDeSaison>1</nombreDeSaison>
</serie>
```

La nouvelle ressource pour obtenir la liste des saisons :

GET /series/Game+of+Thrones/saisons/ HTTP/1.1
Host: serie-tv.fr

HTTP/1.1 200 Ok
Content-type: text/plain; charset=UTF-8
Link: http://serie-tv.fr/series/Game+of+Thrones; rel="serie"
ETAG: 98f71825a81f1a79ea
Last-modified: Fri, 3 Jan 2014 14:09:38 GMT
Content-length: 51

http://serie-tv.fr/series/Game+of+Thrones/saisons/1

Suppression d'une saison

DELETE /serie/Game+of+Thrones/saisons/1 HTTP/1.1
DELETE /series/Game+of+Thrones/saisons/1 HTTP/1.1
Host: serie-tv.fr

HTTP/1.1 204 No content

Si on supprime une saison qui n'existe pas on obtiendra :

HTTP/1.1 404 Not found

La méthode DELETE est idempotente : répéter la même requête 1 à N fois devrait avoir le même comportement. Il est donc préférable que DELETE retourne toujours le code 204 même si la ressource n'existe pas.

Partie 3 : Implémentation d'un service Web RESTful (4 points)

Annotez la classe Java ci-dessous avec les annotations JAX-RS afin de déclarer un service Web RESTful.
Pour rappel, les annotations JAX-RS disponibles sont :
@ApplicationPath, @Consumes, @CookieParam, @DefaultValue, @DELETE, @Encoded, @FormParam, @GET, @HEAD, @HeaderParam, @HttpMethod, @MatrixParam, @OPTIONS, @Path, @PathParam, @POST, @Produces, @PUT, @QueryParam

```
package fr.mycompany.myapp.user;
import javax.ws.rs.*;

// Ressource racine d'un utilisateur accessible à partir de l'URI /user/*

@Path("user")
public class UserAccountResource {

  private UserRepository userRepository = new UserRepository();

  // Création ou mise à jour d'un utilisateur au format application/x-www-form-urlencoded
  @PUT
  @Path("{userId}")
  @Consumes("application/x-www-form-urlencoded")
  public Response createOrUpdate (@PathParam("userId") String userId,
                                  @FormParam("name") String name, @FormParam("email") String email) {
    User user = new User();
    user.setName(name);
    user.setEmail(email);
    return createOrUpdate(userId, user);
  }

  // Création ou mise à jour d'un utilisateur au format application/xml ou application/json
  @PUT
  @Path("{userId}")
  @Consumes({"application/xml", "application/json"})
  public Response createOrUpdate (@PathParam("userId") String userId, User user) {
    user.setId(userId);
    userRespository.createOrUpdate(user);
    return Response.status(Status.CREATED).build();
  }

  // Consultation d'un utilisateur au format application/xml ou application/json
  @GET
  @Path("{userId}")
  @Produces({"application/xml", "application/json"})
  public User getUser(@PathParam("userId") String userId) {
    return userRepository.get(userId);
  }

  @DELETE
  @Path("{userId}")
  public void deleteUser(@PathParam("userId") String userId) {
    userRepository.delete(userId);
  }
}
```