

Cours 6

Web services

HEI 2021 / 2022

Introduction

Qu'est-ce qu'un webservice ?

Introduction

- Un web service est une application disponible sur le web dont l'objectif est de fournir un service (répondre à des besoins).
- Les web services interviennent dans des communications entre machines : c'est ce qu'on appelle du Machine To Machine (M2M).
- Chaque service va proposer un contrat décrivant comment l'interroger et le format qui sera retourné.

Exemples de cas d'utilisation

1. Un site web utilise du code Javascript (Machine cliente) qui exécute des requêtes AJAX vers un web service.
2. Une application mobile Android est un logiciel (Machine cliente) qui envoie des requêtes HTTP vers un web service.

Architectures

- Un web service peut être développé de manière totalement spécifique sans suivre de règles particulières. L'important est que le serveur et le client soient d'accord sur comment discuter (API).
- On trouve néanmoins des principes ou standards dans le développement de web services.

Web services et Java EE

- Java EE propose deux API pour prendre en charge l'écriture de web services :
 - **JAX-WS** pour les web services en **SOAP**
 - **JAX-RS** pour les web services en **REST**
- On va principalement s'intéresser à JAX-RS.

REST

Introduction

- REST est un type d'architecture pour développer des web services.
- REST n'est pas un standard ou une norme.
- C'est un ensemble de principes définis au début des années 2000.

REST et HTTP

- REST s'appuie sur une bonne utilisation du protocole HTTP et notamment :
 - De l'URL pour identifier ce qu'on manipule
 - Du verbe HTTP pour identifier l'action effectuée
 - Du code retour pour identifier le résultat de l'action

Ressources

- En REST, les données manipulées sont appelées ressources.
- Une ressource est un concept logique et ne correspond pas forcément au format de stockage utilisé par l'application.
- Chaque ressource va être associée à une URL.

Ressources et URL

- Imaginons une applications gérant une bibliothèque. Une des ressources manipulées est le « livre ».
- Cette ressource est associée à l'URL suivante :

`http://biblio2000.com/api/ws/books`



Collections et éléments

- L'URL précédente représente une ressource de type Collection.
- Il existe deux types de ressource :
 - Les collections qui représentent une liste d'élément : tous les livres de la bibliothèques
 - Les éléments qui représentent un élément particulier : l'exemplaire du « Seigneur des anneaux »
- Un élément est contenu dans une collection.

URL

- URL d'une collection :

`http://biblio2000.com/api/ws/books`

Les noms de ressource sont au pluriel

- URL d'un élément :

`http://biblio2000.com/api/ws/books/42`

Identifiant de l'élément au sein de la collection

Ressources hiérarchiques

- Il est possible d'organiser ses ressources en hiérarchie.
- Une ressource peut ainsi correspondre aux auteurs d'un livre en particulier :

`/books/42/authors`

Actions

- Les différents verbes HTTP combinés aux ressources permettent d'effectuer les différentes actions.
- Contrairement aux sites web « classiques » qui utilisent principalement GET et POST, les web services peuvent utiliser tous les verbes disponibles.

Lister des éléments

- Utiliser le verbe **GET** sur une ressource de type **collection** permet de lister son contenu.
- Exemple : pour récupérer la liste des livres de la bibliothèque

GET /books

Récupérer un élément

- Utiliser le verbe **GET** sur une ressource de type **élément** permet de récupérer son contenu.
- Exemple : pour récupérer le livre d'identifiant 42 de la bibliothèque

GET /books/42

Ajouter un élément

- Utiliser le verbe **POST** sur une ressource de type **collection** permet d'y ajouter un élément.
 - Le corps de la requête contient l'élément à ajouter.
- Exemple : pour ajouter un nouvel auteur au livre d'identifiant 42 de la bibliothèque

POST /books/42/authors

Modifier un élément

- Utiliser le verbe **PATCH** ou **PUT** sur une ressource de type **élément** permet de le modifier.
 - Le corps de la requête contient les informations de modification.
- Exemple : pour modifier le livre d'identifiant 42 de la bibliothèque

PATCH /books/42

Supprimer un élément

- Utiliser le verbe **DELETE** sur une ressource de type **élément** permet de le supprimer.
- Exemple : pour supprimer le livre d'identifiant 42 de la bibliothèque

DELETE /books/42

Paramètres

- Il est possible d'ajouter des paramètres à l'URL. Cela va permettre d'ajouter des informations à la requête pour :
 - Filtrer le résultat
 - Trier les collections
 - Effectuer des actions particulières
 - ...

Filterer

- Exemple : pour récupérer la liste des livres de la bibliothèque qui possède le genre Science-Fiction

`GET /books?genre=sci-fi`

- Exemple : pour récupérer la liste des livres de la bibliothèque empruntés par l'utilisateur d'identifiant 42

`GET /books?borrower=42`

Tri

- Exemple : pour trier la liste des livres par titre

`GET /books?sort=title`

Code de retour

- Les codes de retour HTTP classiques sont utilisés mais un web service peut en déclarer des particuliers si nécessaire.
- La plupart du temps, on se contente des codes de retour classiques.

POST /books



201 Created

Codes d'erreur

- Exemple : Emprunter le livre d'identifiant 42 alors qu'il est déjà emprunté

PATCH /books/42?action=borrow → 409 Conflict

- Exemple : Supprimer le livre d'identifiant 42 alors qu'on est simple utilisateur

DELETE /books/42 → 403 Forbidden

Format d'échange

- REST n'impose aucun format d'échange particulier.
- Classiquement, JSON est utilisé.

JSON et Javascript

- Une variable **JSON** est disponible en javascript pour transformer des objet Javascript en chaine de caractères et vice-versa :
 - **JSON.stringify** transforme un objet en string ;
 - **JSON.parse** transforme une string en objet.

```
let book = {id: 1, title: "Foundation"};
let jsonBook = JSON.stringify(book);
console.log(jsonBook); // {"id":1,"title":"Foundation"}
let newBook = JSON.parse(jsonBook);
console.log(newBook.id); // 1
console.log(newBook.title); // Foundation
```

JSON et Java

- Java ne permet pas de manipuler du JSON sans importer une librairie.
- Jackson est une librairie permettant de créer des objets JSON en Java et de générer la chaîne de caractère associée.

Mapping

- La classe **ObjectMapper** permet de transformer un objet Java en une chaine JSON et vice-versa:

```
public class Book {  
    private Integer id;  
    private String title;  
    // ...  
}
```



The diagram illustrates the mapping process. A green box labeled "ObjectMapper" has two arrows pointing outwards. The left arrow points from the JSON object to the Java class, and the right arrow points from the Java class to the JSON object.

```
{  
    "id": 1,  
    "title": "American Gods"  
}
```

Transformer un objet Java en JSON

- La méthode `writeValueAsString` de `ObjectMapper` permet de générer un objet String contenant une représentation JSON de l'objet.

```
Book book = new Book(1, "Foundation");
ObjectMapper mapper = new ObjectMapper();
String bookJson = mapper.writeValueAsString(book);
System.out.println(bookJson);
// {"id":1,"title":"Foundation"}
```

Transformer un JSON en objet Java

- La méthode `readValue` de `ObjectMapper` permet de générer un objet Java à partir d'une chaîne contenant une représentation JSON de l'objet.

```
String json = "{\"id\":1,\"title\":\"Foundation\"}";  
ObjectMapper mapper = new ObjectMapper();  
Book book = mapper.readValue(json, Book.class);  
System.out.println(book.getId()); // 1  
System.out.println(book.getTitle()); // Foundation
```

JAX-RS

Implémenter un web service REST en Java EE

Introduction

- JAX-RS est une API Java EE permettant de mettre en place un web service REST.
- Comme JDBC, ce n'est qu'une API. Il faut choisir une implémentation pour pouvoir l'utiliser.

Jersey

- Jersey est une implémentation de JAX-RS.
- Deux nouvelles dépendances doivent être ajoutées au projet :

```
<dependency>
  <groupId>org.glassfish.jersey.containers</groupId>
  <artifactId>jersey-container-servlet</artifactId>
  <version>2.29.1</version>
</dependency>
<dependency>
  <groupId>org.glassfish.jersey.inject</groupId>
  <artifactId>jersey-hk2</artifactId>
  <version>2.29.1</version>
</dependency>
```

Contrôleur

- Chaque ressource REST va être associée à une classe Java qu'on appelle contrôleur.
- Un contrôleur est une simple classe Java annotée avec `@Path`.
 - Cette annotation prend en paramètre l'URL de la ressource.

```
@Path("/books")
public class BookController {
    // ...
}
```

Servlet JAX-RS

- JAX-RS fonctionne avec une servlet qui va intercepter l'ensemble des requête sur le web service.
- Cette servlet va ensuite se charger d'envoyer la requête au bon contrôleur en se basant sur l'URL.

Configurer la servlet

- La configuration de la servlet peut se faire dans le fichier web.xml.

```
<servlet>  
  <servlet-name>javax.ws.rs.core.Application</servlet-name>  
</servlet>
```

```
<servlet-mapping>  
  <servlet-name>javax.ws.rs.core.Application</servlet-name>  
  <url-pattern>/ws/*</url-pattern>  
</servlet-mapping>
```



URL de base du Web Service

Ajouter une méthode à un contrôleur

- Pour chaque requête faite sur une ressource, une méthode doit être ajoutée dans le contrôleur.
- Cette méthode est annotée avec le verbe HTTP utilisé : `@GET`, `@POST`, `@PATCH`, etc.

```
@GET
public Response listAllBooks() {
    // ...
}
```

Préciser une URL

- Il est possible d'annoter une méthode du contrôleur avec `@Path` pour préciser l'URL de l'action.

```
@GET
@Path("/authors")
public Response getBookAuthors() {
    // ...
}
```

URL d'une méthode

- L'URL associée à une méthode est la concaténation de :
 - L'URL de base de la servlet JAX-RS
 - L'URL du contrôleur
 - L'URL de la méthode

`<url-pattern>/ws/*</url-pattern>` + `@Path("/books")` + `@Path("/authors")`



`/ws/books/authors`

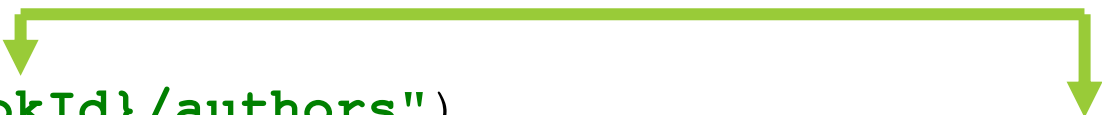
URL paramétrées

- Il est possible d'ajouter des parties variables dans les URL.

`/ws/books/42/authors`  `/ws/books/{bookId}/authors`

- Cette variable est ensuite récupérable en paramètre de la méthode avec l'annotation `@PathParam`.

```
@GET
@Path("/{bookId}/authors")
public Response getBookAuthors(@PathParam("bookId") Integer bookId) {
    // ...
    return null;
}
```

A green arrow originates from the `{bookId}` placeholder in the `@Path` annotation and points down to the `@PathParam("bookId")` annotation in the method signature. Another green arrow originates from the `"bookId"` string in the `@PathParam` annotation and points down to the `bookId` parameter in the method signature.

Type de la requête

- Il est possible de préciser le type du corps de la requête avec l'annotation `@Consumes`.

```
@POST
@Consumes(MediaType.APPLICATION_FORM_URLENCODED)
public void saveBook(/* ... */) {

}
```

Type de la réponse

- Il est possible de préciser le type du corps de la réponse avec l'annotation `@Produces`.

```
@GET
@Path("/{bookId}")
@Produces(MediaType.APPLICATION_JSON)
public Response getBook(@PathParam("bookId") Integer bookId) {
    // ...
}
```

Paramètres de la requête

- Les paramètres présents dans l'URL de la requête sont récupérables avec l'annotation `@QueryParam`.

`/ws/books?author=Tolkien`

```
@GET
public Response listBooks(@QueryParam("author") String author) {
    // ...
}
```



- Le paramètre `author` vaut `null` si il n'est pas présent dans l'URL.

Paramètres du corps de la requête

- Les paramètres présents dans le corps de la requête sont récupérables avec l'annotation `@FormParam`.
 - Cela n'est possible que si le corps de la requête correspond au format `application/x-www-form-urlencoded`.
- Les paramètres valent `null` si le paramètre n'est pas présent.

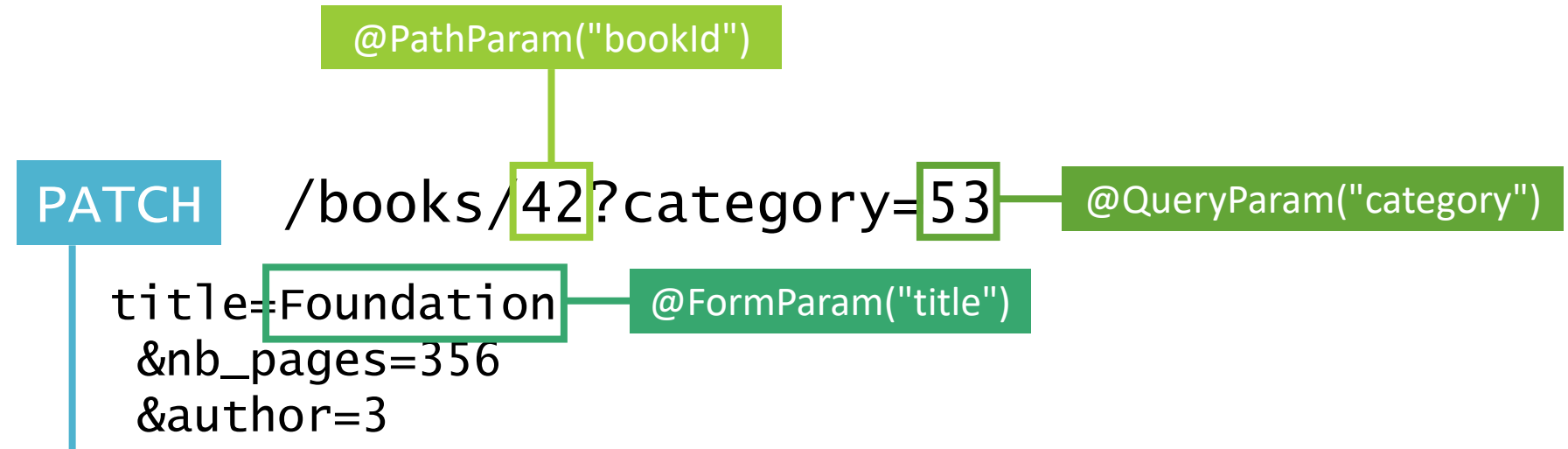
Paramètres de formulaire

- Il est possible de récupérer tous les paramètres du formulaire dans un objet `MultivaluedMap`.

```
@POST
@Consumes(MediaType.APPLICATION_FORM_URLENCODED)
public void saveBook(MultivaluedMap<String, String> formParams) {

}
```

Bilan des paramètres de la requête



Réponse

- Le retour le plus générique pour les méthodes des contrôleurs est un objet `javax.ws.rs.core.Response`.
- Cet objet est la modélisation de la réponse HTTP. Il est créé à partir d'un builder.

```
return Response.status(200).entity(jsonBooks).build();
```

Code de retour

Corps de
la réponse
(String)

Méthode qui crée
l'objet Response

Réponse

- Tous les types primitifs sont également acceptables en tant que réponse.
 - La valeur sera affichée directement dans le corps de la réponse avec un type **text/plain**.

```
@GET
@Path("/count")
public Integer countBooks(@QueryParam("author") String author) {
    // ...
    return books.size();
}
```

Codes d'erreur courants

- À l'utilisation d'un webservice JAX-RS, des codes d'erreurs sont envoyés automatiquement dans certains cas :
 - **404** : l'URL demandée n'existe pas dans le projet Java.
 - **405** : l'URL demandée existe mais n'est pas associée au bon verbe HTTP.
 - **415** : l'URL demandée existe avec le verbe indiqué mais le content-type de la requête ou réponse n'est pas correcte.
 - **500** : une exception a été levée côté serveur.

Intégrer Jackson à JAX-RS

- Il est possible d'intégrer directement Jackson à JAX-RS pour que le mapping d'objets se fasse automatiquement.
- Une nouvelle dépendance doit être ajoutée au projet :

```
<dependency>  
  <groupId>org.glassfish.jersey.media</groupId>  
  <artifactId>jersey-media-json-jackson</artifactId>  
  <version>2.29.1</version>  
</dependency>
```

Utiliser Jackson pour la réponse

- Il suffit de renvoyer des objets javas en retour d'une méthode du contrôleur pour qu'ils soient automatiquement transformés en JSON.
 - Il faut obligatoirement préciser le type de la réponse avec **@Produces**.

```
@GET
@Produces(MediaType.APPLICATION_JSON)
public List<Book> listBooks(@QueryParam("author") String author) {
    List<Book> books = new ArrayList<>();
    // ...
    return books;
}
```

Utiliser Jackson pour la requête

- Si un objet Java est ajouté en paramètre de la méthode, il sera automatiquement transformé à partir du JSON.
 - Il faut obligatoirement préciser le type de la requête avec `@Consumes`.

```
@POST
@Consumes(MediaType.APPLICATION_JSON)
public void saveBook(Book book) {

}
```

- Attention, le JSON présent dans la requête doit correspondre à l'objet Java demandé : tout champ inconnu dans le JSON va lever une erreur.

Sources

- https://en.wikipedia.org/wiki/Representational_state_transfer
- <https://docs.oracle.com/javaee/7/tutorial/jaxrs002.htm>
- <https://jersey.github.io/documentation/latest/index.html>