Développement des web Services

Qu'est-ce qu'une API ?

Une API est une interface de programmation qui permet à un programme informatique de communiquer avec d'autres programmes. Elle permet de faire communiquer des applications entre elles.

C'est un ensemble de routines, de protocoles et de fonctions permettant à un logiciel de communiquer avec d'autres logiciels.

Comment fonctionne une API ?

Une API fonctionne en deux temps :

- **La première étape consiste à envoyer une requête à l'API**
- **La deuxième étape consiste à recevoir une réponse de l'API**

Envoyer une requête à l'API

Pour envoyer une requête à l'API, il faut utiliser un client.

Dans le cas d'une API REST, on utilise un client HTTP.

Les requêtes HTTP sont des requêtes de type GET, POST, PUT, DELETE ...

Elles sont envoyées au serveur de l'API.

Dans le format de la requête, on peut utiliser du JSON ou du XML.

requetes HTTP GET

```
GET /api/users HTTP/1.1
Host: api.example.com
```

```
HTTP/1.1 200 OK
Content-Type: application/json

[
        {
                "id": 1,
                "name": "John Doe"
        },
        {
                "id": 2,
                "name": "Jane Doe"
        }
]
```

```
GET /api/users/1 HTTP/1.1
Host: api.example.com
```

```
HTTP/1.1 200 OK
```

```
<?xml version="1.0" encoding="UTF-8"?>
<user>
        <id>1</id>
        <name>John Doe</name>
</user>
```

Recevoir une réponse de l'API

Pour recevoir une réponse de l'API, il faut utiliser un serveur.

Dans le cas d'une API REST, on utilise un serveur HTTP.

Les réponses HTTP sont des réponses de type 200, 400, 401, 404 ...

Elles sont envoyées par le serveur de l'API.

Dans le format de la réponse, on peut utiliser du JSON ou du XML.


```
{
    "id": 1,
    "name": "toto"
}
```

```xml
<user>
    <id>1</id>
    <name>toto</name>
</user>
```

Les fichiers json

JSON est un format de données textuel, basé sur le langage de programmation JavaScript, qui permet de représenter des données de façon simple et claire.

Les fichiers xml

XML est un langage de balisage conçu pour représenter des données de façon structurée et hiérarchisée.

Comment utiliser une API ?

Pour utiliser une API, il faut :

- **Connaître l'adresse de l'API**
- **Connaître la requête à envoyer à l'API**
- **Connaître le format de la requête à envoyer à l'API**
- **Connaître la réponse de l'API**
- **Connaître le format de la réponse de l'API**

Les différents types d'API

Il existe plusieurs types d'API :

- **API interne**
- **API externe**
- **API publique**
- **API privée**

API interne

Une API interne est une API qui est utilisée par les développeurs d'une entreprise pour communiquer entre eux.

API externe

Une API externe est une API qui est utilisée par les utilisateurs d'une entreprise pour communiquer avec les développeurs d'une entreprise.

API publique

Une API publique est une API qui est utilisée par tout le monde.

API privée

Une API privée est une API qui est utilisée par une entreprise pour communiquer avec ses clients.

Les différents types de requêtes

Il existe plusieurs types de requêtes :

- **GET : permet de récupérer des données**
- **POST : permet d'envoyer des données**
- **PUT : permet de modifier des données**
- **DELETE : permet de supprimer des données**

GET

La requête GET permet de récupérer des données.
Nous utiliserons cette requête afin de sélectionner des données dans une base de données, de récupérer des données d'un fichier, etc.

Exemple :

```
GET /api/users HTTP/1.1
Host: api.example.com
```

POST

La requête POST permet d'envoyer des données.
Nous utiliserons cette requête afin d'ajouter des données dans une base de données, d'envoyer des données dans un fichier, etc.

Exemple :

```
POST /api/users HTTP/1.1
Host: api.example.com
```

PUT

La requête PUT permet de modifier des données.
Nous utiliserons cette requête afin de modifier des données dans une base de données, de modifier des données dans un fichier, etc.

Exemple :

```
PUT /api/users HTTP/1.1
Host: api.example.com
```

DELETE

La requête DELETE permet de supprimer des données.
Nous utiliserons cette requête afin de supprimer des données dans une base de données, de supprimer des données dans un fichier, etc.

Exemple :

```
DELETE /api/users HTTP/1.1
Host: api.example.com
```

Les différents types de réponses

Il existe plusieurs types de réponses :

- **200**
- **201**
- **204**
- **400**
- **401**
- **403**
- **404**

200 : OK

La réponse 200 est une réponse positive.

Elle signifie que la requête a été traitée avec succès.

201 : CREATED

La réponse 201 est une réponse positive.
Elle signifie que la requête a été traitée avec succès et qu'une nouvelle ressource a été créée.

202 : ACCEPTED

La réponse 202 est une réponse positive.
Elle signifie que la requête a été traitée avec succès et qu'elle est en attente d'une autre requête.

400 : BAD REQUEST

La réponse 400 est une réponse négative.

Elle signifie que la requête est incorrecte.

401 : UNAUTHORIZED

La réponse 401 est une réponse négative.
Elle signifie que l'utilisateur n'est pas autorisé à accéder à la ressource.

403 : FORBIDDEN

La réponse 403 est une réponse négative.
Elle signifie que l'utilisateur est autorisé à accéder à la ressource mais qu'il n'a pas les droits pour effectuer l'action.

404 : NOT FOUND

La réponse 404 est une réponse négative.

Elle signifie que la ressource n'a pas été trouvée.

Les différents types de données

Il existe plusieurs types de données :

- **JSON**
- **XML**
- **CSV**
- **YAML**

L'api rest

L'API REST est une architecture de communication entre un client et un serveur.
REST signifie REpresentational State Transfer.
Elle se distingue des autres API par le fait qu'elle est basée sur les ressources.
De ce fait, elle est plus simple à utiliser et à comprendre, car elle se construit autour des ressources.

Les ressources

Une ressource est un objet qui peut être manipulé par l'API.

Une ressource peut être une personne, un produit, un article, etc.

Cette ressource est représentée par un identifiant unique.

Les méthodes

Notre API REST se base sur les méthodes HTTP.
Elle devra donc créer des méthodes HTTP pour chaque action que l'on souhaite effectuer sur une ressource.

- **GET**
- **POST**
- **PUT**
- **DELETE**

Les états

Les états sont des informations sur la ressource.
Ils permettent de savoir si la ressource est disponible, si elle est en cours de modification, etc.

Les principes de l'API REST

L'API REST se base sur 6 principes :

- **Uniformité**
- **Indépendance**
- **Cacheabilité**
- **Client/serveur**
- **Code sur demande**
- **Séparation des interfaces**

Uniformité

L'API REST doit être uniforme.
Cela signifie que chaque ressource doit être identifiée de la même manière.
Par exemple, si l'on souhaite récupérer une personne, on utilisera toujours la même URL.
Cela permet de faciliter la compréhension de l'API.

Indépendance

L'API REST doit être indépendante.

Cela signifie que chaque ressource doit être indépendante de l'API.

Elle sera donc accessible même si l'API est indisponible.

Cacheabilité

L'API REST doit être mise en cache.

Cela signifie que chaque ressource doit être mise en cache.

Cela permet de réduire le temps de réponse de l'API.

Client/serveur

L'API REST doit être basée sur le client/serveur.

Cela signifie que chaque ressource doit être accessible par le client.

Cela permet de faciliter l'utilisation de l'API.

Code sur demande

L'API REST doit être basée sur le code sur demande.
Cela signifie que les ressources doivent être accessibles par le client, mais que le client doit formuler une demande pour obtenir le code.
Cela permet de réduire le temps de réponse de l'API.

Séparation des interfaces

L'API REST doit être basée sur la séparation des interfaces. Cela signifie que des interfaces devront être créées pour chaque ressource.

La mise en place d'une API REST

Pour mettre en place une API REST, il faut :

- **Définir les ressources : les ressources sont les objets manipulés par l'API.**
- **Définir les méthodes : actions que l'on souhaite effectuer sur les ressources.**
- **Définir les états : les états sont les informations sur les ressources.**
- **Définir les principes : les principes sont les règles à respecter pour créer une API REST.**

Les avantages de l'API REST

Les avantages de l'API REST sont :

- **Simplicité**
- **Indépendance**
- **Cacheabilité**
- **Séparation des interfaces**
- **Securisation**

Les annotations de l'api rest

Les annotations de l'API REST sont :

- **@Path**
- **@Get**
- **@Post**
- **@Put**
- **@Delete**

Les annotations de l'api rest

Les autres annotations de l'API REST sont :

- **@Produces**
- **@Consumes**
- **@Context**

@produces

L'annotation @produces permet de définir le type de données que l'on souhaite obtenir en retour.

On peut utiliser les types suivants :

- **Application/json**
- **Application/xml**

```
@GET
@Path("/get")
@Produces("application/json")


---

@consumes

L'annotation @consumes permet de définir le type de données que l'on souhaite envoyer.
On peut utiliser les types suivants :

-   application/json
-   application/xml

```java
@POST
@Path("/add")
@Consumes(MediaType.APPLICATION_JSON)
```


```

@context

L'annotation @context permet de définir le contexte de l'API.

Elle permet de définir le chemin d'accès à l'API.

```
@context("/api")
```

HttpServletRequest

HttpServletRequest un objet qui permet de récupérer les informations de la requête HTTP.
Il permet de récupérer les informations suivantes :

- **Le chemin d'accès à l'API**
- **Le type de méthode HTTP**
- **Le type de données envoyées**
- **Le type de données attendues**

GetSession

La méthode getSession permet de récupérer la session HTTP.

Nous pouvons l'utiliser pour récupérer des informations sur l'utilisateur.

Et pour obtenir des attributs de la session.

```
HttpSession session = request.getSession();
String login = (String) session.getAttribute("login");
```

GetParameter

La méthode getParameter permet de récupérer les paramètres de la requête HTTP.
Nous pouvons l'utiliser pour récupérer des informations sur l'utilisateur.
Et pour obtenir des attributs de la session.

```
String nom = request.getParameter("nom");
```

TP : créer une API REST

Créer une API REST qui permet d'afficher sur la page web le message "Hello World".

Maintenant que nous avons vu les bases d'une api, nous allons revenir les api et leur utilisation.

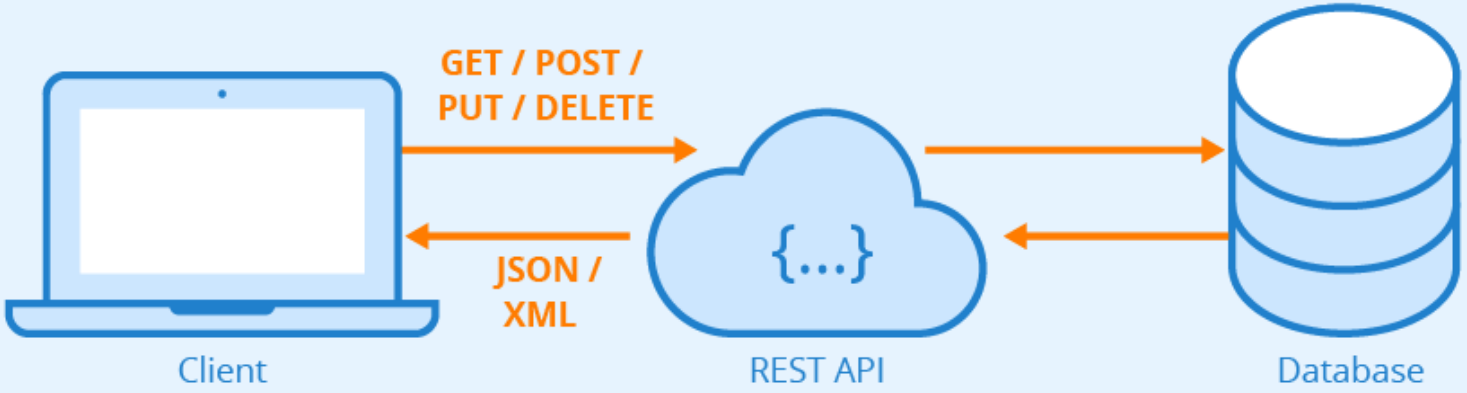
Les API

Les API sont des interfaces de programmation.

Elles sont utilisées au quotidien pour traiter des données.

Leurs utilisations les plus courantes sont :

- **Afficher des données dynamiques**
- **Envoyer des données**
- **Récupérer des données**


GET / POST /
PUT / DELETE
JSON /
XML
{...}
Client
REST API
Database


Java

Les API

exemple d'utilisation d'une API :

Si vous voulez laisser aux utilisateurs de notre site web la posibilité de faire des actions sur notre application, vous pouvez utiliser une API.

Nous pourrons alors laisser la possibilité aux utilisateurs de notre api de : créer des données, modifier des données, supprimer des données, récupérer des données...

L'authentification

Il est possible de sécuriser une API REST.
Il existe plusieurs méthodes pour sécuriser une API REST :

- Basic Auth : l'utilisateur doit envoyer son login et son mot de passe
    - 🐊 Peut sur mais simple à mettre en place
- API Key : l'utilisateur doit envoyer une clé d'API
    - 🐊 Plus sécurisé que le basic auth mais moins stable
- OAuth : l'utilisateur doit envoyer un token
    - 🐊 Plus sécurisé que le basic auth et l'api key mais plus complexe à mettre en place

Les API REST

Maintenant que nous savons ce qu'est une API, nous allons revenir sur les spécificités des API REST.

Les API REST

Les API REST se différencient des autres API par leur architecture.
Elle a été créée par Roy Fielding en 2000. Qui avait pour but de regler les problèmes de l'architecture des API.
Ces problèmes étaient :

- **La complexité**
- **La portabilité**
- **La performance**
- **La scalabilité**

REST / RESTful

REST est un acronyme qui signifie REpresentational State Transfer.
RESTful est un terme qui signifie que l'API respecte les principes de REST.

Les principes de REST sont :

- **Interface uniforme**
- **Architecture client/serveur**
- **Absence d'état**
- **Cacheabilité**
- **Code sur demande**
- **Système multi-plateforme**

Interface uniforme

Une interface est uniforme si :

- **Nous pouvons accéder aux ressources de manière indépendante**
- **Les demandes et les réponses sont autosuffisantes**

Architecture client/serveur

Une interface respecte l'architecture client/serveur si :

- **Les composants client et serveur sont indépendants et peuvent être développés indépendamment**
- **Les composants client et serveur peuvent être remplacés indépendamment**

le client est un programme qui envoie des demandes à un serveur.
le serveur est un programme qui reçoit des demandes et qui envoie des réponses.

Absence d'état

Une interface est sans état si :

- **Les interactions entre le client et le serveurs ne sauvegardent pas d'état.**
- **Les interactions sont indépendantes les unes des autres.**

Cacheabilité

Le serveur peut indiquer au client si les réponses peuvent être mises en cache.

Le but est de réduire le nombre de requêtes.
Les données sont mises en cache pour éviter de les réenvoyer à chaque fois, elle pourront donc être réutilisées sans avoir à les réenvoyer.

Code sur demande

C'est la seule contrainte qui n'est pas obligatoire. Oui. C'est une contrainte facultative.

Le serveur peut envoyer du code au client pour qu'il puisse l'exécuter.
Le code peut être du javascript, du css, du html...

Rest vs SOAP

SOAP est un protocole de communication.

Il permet de communiquer entre deux applications.

Il est basé sur le langage XML.

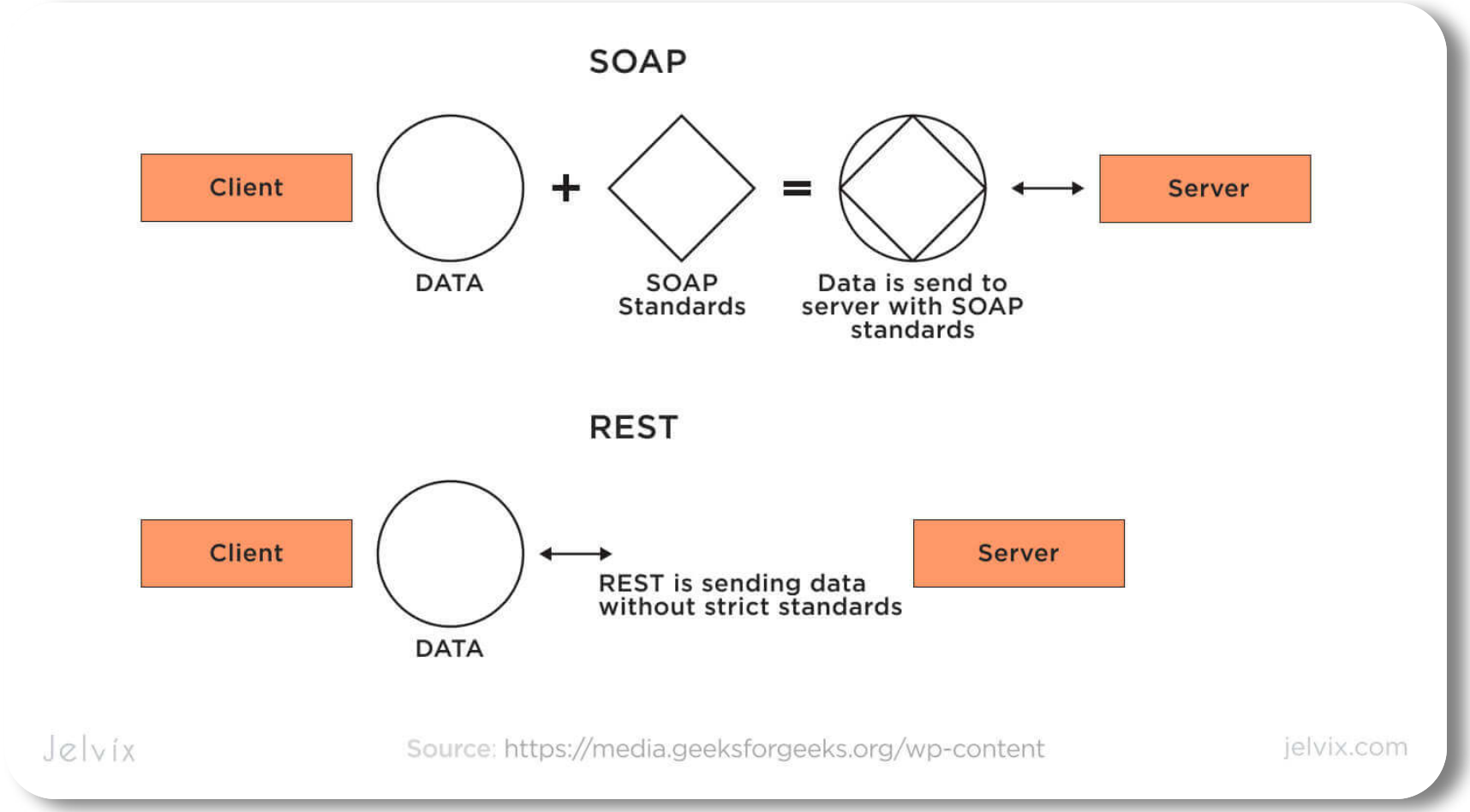
Il permet d'assurer la portabilité des applications.

REST est un style d'architecture.

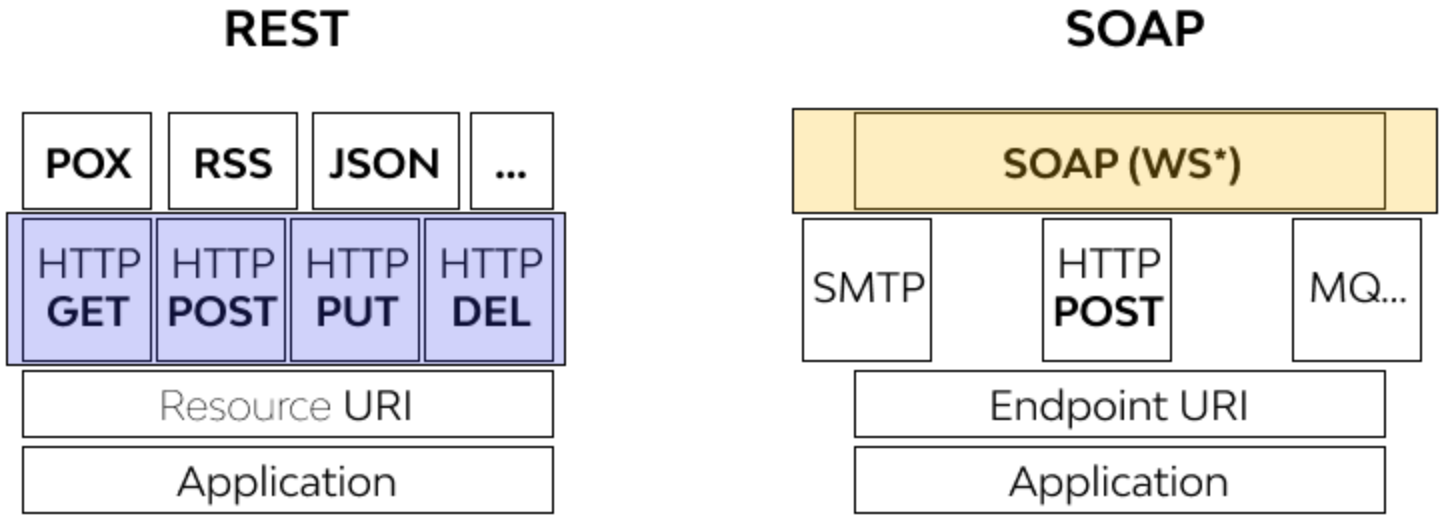
Il permet de communiquer entre deux applications.

Nous pouvons l'utiliser avec n'importe quel langage.

Rest vs SOAP


SOAP
Client
+
=
Server
DATA
SOAP Standards
Data is send to server with SOAP standards
REST
Client
Server
REST is sending data without strict standards
DATA
Jelvix
Source: https://media.geeksforgeeks.org/wp-content
jelvix.com

Protocol Layering



Les gets

Les gets sont des méthodes HTTP qui permettent de récupérer des données.

Nous voulons les utiliser pour récupérer des données.

Un exemple de get :

```
@Get
@Path("/films")
@Produces(MediaType.APPLICATION_JSON)
public List<Film> getFilms() {
        return filmService.getFilms();
}
```

TP : développer une API REST

Faire un pojo Film avec les attributs suivants :

- **Id**
- **Titre**
- **Genre**

Et créer une API REST qui permet de récupérer la liste des films.

Créer une classe FilmService qui permet de récupérer la liste des films.
Il aura comme attribut une liste de films :

- **Liste de films**

Requête get avec id

En plus des gets, nous pouvons utiliser des gets avec un id. De cette manière, nous pouvons récupérer une ressource en particulier.

Un exemple de get avec id :

```
@Get
@Path("/films/{id}")
@Produces(MediaType.APPLICATION_JSON)
public Film getFilm(@PathParam("id") int id) {
        return filmService.getFilm(id);
}
```

TP : créer une API REST

Créer une API REST qui se base sur une unité persistante.

Paramètres de requête

Les méthodes GET avec plusieurs arguments seront créées comme suit :

```
@Get
@Path("nom-{nom}-film-{film}")
@Produces(MediaType.APPLICATION_JSON)
public List<Acteur> getActeurs(@PathParam("nom") String nom, @PathParam("film") String film) {
        return acteurService.getActeurs(nom, film);
}
```

Paramètres de requête

L'annotation @QueryParam permet de récupérer les paramètres de requête.

```
@Get
@Path("acteurs")
@Produces(MediaType.APPLICATION_JSON)
public List<Acteur> getActeurs(@QueryParam("nom") String nom, @QueryParam("film") String film) {
        return acteurService.getActeurs(nom, film);
}
```

```
GET /acteurs?nom=nom&film=film
```

@DefaultValue

L'annotation @DefaultValue permet de définir une valeur par défaut pour un paramètre de requête.

```
@Get
@Path("acteurs")
@Produces(MediaType.APPLICATION_JSON)
public List<Acteur> getActeurs(@QueryParam("nom") @DefaultValue("nom") String nom, @QueryParam("film") @DefaultValue("film") String film) {
        return acteurService.getActeurs(nom, film);
}
```

Les requêtes post

Les requêtes post sont des méthodes HTTP qui permettent d'envoyer des données.
Nous voulons les utiliser pour envoyer des données.

Un exemple de post avec paramètres :

```
@Post
@Path("/films")
@produces(MediaType.APPLICATION_JSON)
public void addFilm(@FormParam("titre") String titre, @FormParam("annee") int annee) {
        filmService.addFilm(titre, annee);
}
```

TP : créer une API REST

Créer une API REST qui nous permet de lire et d'écrire des données.
Dans une base de données, nous aurons une table film qui contiendra des objets de type Film. avec les attributs suivants :

- **Id**
- **Titre**
- **Annee**

Les requêtes put

Les requêtes put sont des méthodes HTTP qui permettent de modifier des données.
Nous voulons les utiliser pour modifier des données.

Un exemple de put avec paramètres :

```
@Put
@Path("/films/{id}")
@produces(MediaType.APPLICATION_JSON)
public void updateFilm(@PathParam("id") int id, @FormParam("titre") String titre, @FormParam("annee") int annee) {
        filmService.updateFilm(id, titre, annee);
}
```

TP : créer une API REST

Créer une API REST qui nous permet de modifier des données. Dans une base de données, nous aurons une table film qui contiendra des objets de type Film. avec les attributs suivants :

- **Id**
- **Titre**
- **Annee**

Les requêtes delete

Les requêtes delete sont des méthodes HTTP qui permettent de supprimer des données.

Un exemple de delete avec paramètres :

```
@Delete
@Path("/films/{id}")
@produces(MediaType.APPLICATION_JSON)
public void deleteFilm(@PathParam("id") int id) {
        filmService.deleteFilm(id);
}
```

TP : créer une API REST

Créer une API REST qui nous permet de supprimer des données. Dans une base de données, nous aurons une table film qui contiendra des objets de type Film. avec les attributs suivants :

- **Id**
- **Titre**
- **Annee**

Conclusion

Nous avons vu comment créer une API REST avec JAX-RS. Les méthodes HTTP GET, POST, PUT et DELETE sont utilisées pour récupérer, envoyer, modifier et supprimer des données.

L'api REST est une interface de programmation qui permet de communiquer avec une application.

Les annotations @Path, @Get, @Post, @Put, @Delete, @Produces, @Consumes, @PathParam, @QueryParam, @FormParam, @DefaultValue sont utilisées pour créer une API REST.

TP : créer une API REST

Intégrer a l'API REST précédente la possibilité de créer un utilisateur.