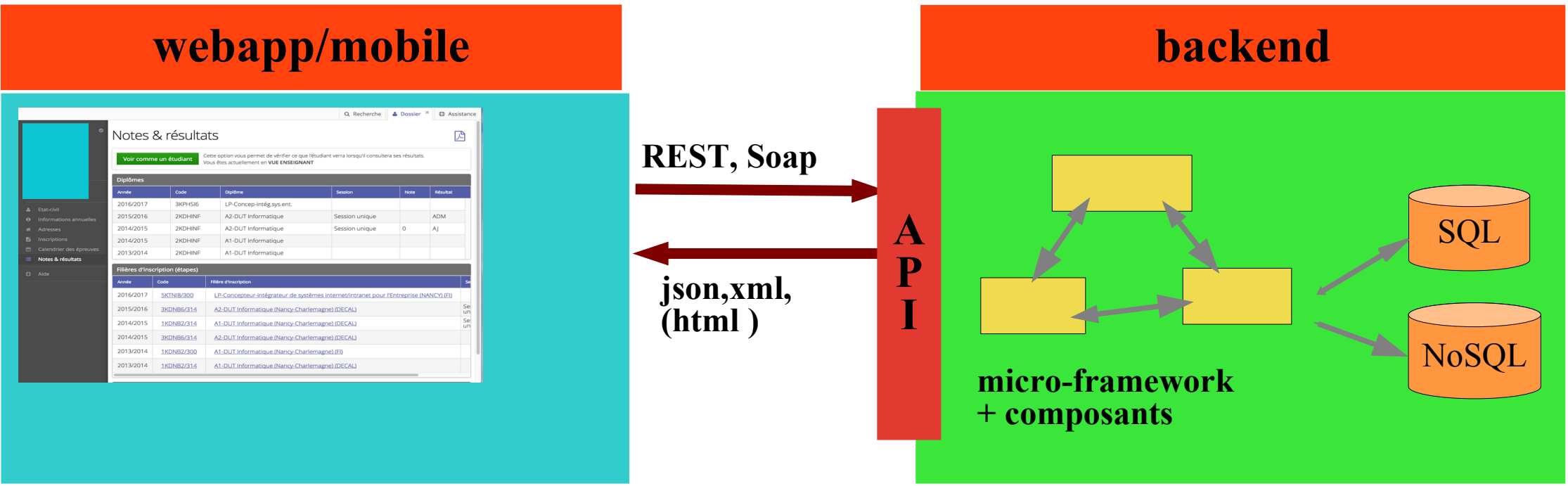
Construire des api REST

Protocole, urls, action

contexte

Applications dans lesquelles une **webapp** ou une **appli mobile** accède à des services proposés par un **backend** au travers d'une **API**

L'interface est *générée* et *exécutée* sur la webapp/mobile
le backend réalise des ***services*** et retourne des données en format neutre



REST : **RE**presentational **S**tate **T**ransfer

- Un **modèle de conception** des API exposées par un backend
  - **désignation des ressources** (données, services) exposées par les API,
  - **actions** sur les ressources disponibles pour le client,
  - **échange des données** entre le client et le backend
- Un **protocole d'interaction** et d'échange entre un client et un backend,

REST

- un **service web** expose des données ou **ressources** désignées par des **URI**
- un **client** interagit avec ces ressources au travers d'**actions élémentaires** :
  - création, → create
  - lecture, → retrieve,
  - remplacement, → update,
  - suppression → delete
- Ces actions et leurs résultats sont exprimées à l'aide de **requêtes/réponses HTTP**

les éléments d'une API REST

1) les ressources et les URL/URI

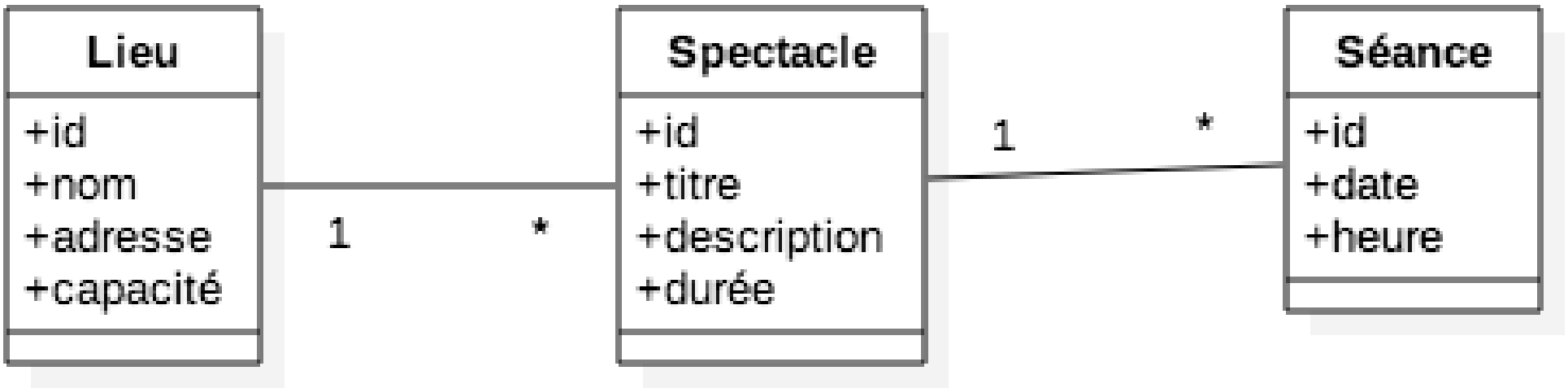
2) les opérations sur les ressources

3) les représentations des ressources : contenu des échanges entre clients et serveurs

Identifier et désigner les ressources

- une **ressource** = 1 entité métier d'intérêt pour les fonctionnalités exposées par l'API
  - un sandwich, une catégorie, une commande
- les ressources sont regroupées en **collections** de même type
  - les Sandwichs, les Catégories
- et peuvent être liées entre elles
  - les sandwichs d'une catégories
- On peut faire un modèle UML des ressources
  - description des ressources exposées par l'API
  - différent du modèle de la BD utilisée par le backend

exemple


Lieu
+id
+nom
+adresse
+capacité
1
*
Spectacle
+id
+titre
+description
+durée
1
*
Séance
+id
+date
+heure

les url/uri

- une **URI** Restful désigne :
  - une **collection** de ressources de même type
  - une **ressource** particulière au sein d'une collection, identifiée par un ID unique, ou
  - une **association** entre des ressources
- Forme générale
  - **collection** (endpoint) : /nom-type-entités
  - **ressource** : /nom-type-entités/{id-ressource}

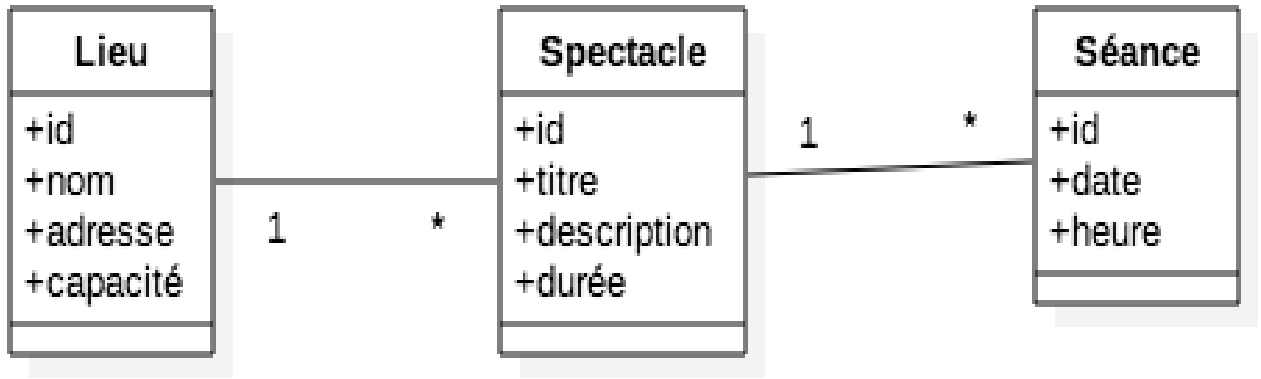
exemples

- **Collections** : (endpoints)

  `/spectacles`

  `/seances`

  `/lieux`



- **ressources**

  `/spectacles/2`

  `/spectacles/73`

  `/seances/1337`

- **associations**

  `/spectacles/2/lieu`

  `/spectacles/2/seances`

Les opérations

Le modèle REST permet de décrire des actions appliquées aux ressources, aux collections et aux associations

Les actions sont indépendantes des ressources et sont spécifiées par la **méthode HTTP** utilisée dans la requête

```
GET   /spectacles/2
POST /spectacles
GET   /spectacles
```

```
PUT   /spectacles/2
DELETE /spectacles/3
GET /spectacles/13/seances
```

Les opérations

GET	obtenir la représentation d'une ressource le serveur transmet une représentation des ressources au client
POST	créer une ressource dont l'identifiant sera calculé par le service, ou déclencher une exécution liée à une ressource – **actions non idempotentes**
PUT	remplacer une ressource existante, ou la créer si elle n'existe pas et que son identifiant est fourni – **actions idempotentes** – le client transmet la nouvelle représentation au serveur
DELETE	supprimer une ressource existante

Actions sur les ressources

METHOD HTTP	/spectacles	/spectacles/*{id}*
GET	Retourne la collection : liste des spectacles 200 OK	Retourne la ressource désignée le spectacle d'identifiant {id} 200 OK | 404 NOT FOUND
POST	Création d'une ressource dans la collection : nouveau spectacle ID créé par le serveur 201 CREATED	ERREUR 405 NOT ALLOWED
PUT	ERREUR 405 NOT ALLOWED	remplace la ressource (la crée si elle n'existe pas) 200 OK | 204 No Content 201 Created | 404
DELETE	Supprime la collection complète des spectacles 200 OK | 204 No Content	Supprime la ressource désignée 200 OK | 204 No Content 404 NOT FOUND

Actions sur les associations

METHOD	/lieu/73/spectacles
GET	Retourne la collection de ressources associées : liste des speactcles pour le lieu 73 200 OK | 404 NOT FOUND
POST	Création d'une ressource associée : création d'un spectacle pour le lieu 73, ID du spectacle créé par le serveur – 201 CREATED
PUT	Remplace la collection de ressources associées – seule l'association est remplacée 200 OK
DELETE	Supprime toutes les associations entre le lieu 73 et les spectacles 200 OK | 204 No Content

remarque

- **Il n'y a pas d'autres URI !**
- **Il n’y a pas d’autres opérations !**

- **en particulier ces URI ne sont pas conformes au principes REST :**
  - `/spectacle/32/getDuree`
  - `/seances/add`
  - `/spectacle/28/addSeance`
  - `/spectacle/updateDuree?id=73`

exemples

- Toutes les infos sur un spectacle :

```
GET   /spectacles/45443
GET   /spectacles/45443/lieu
GET   /spectacles/45443/seances
```

- Le programme d'un lieu :

```
GET   /lieux/73/spectacles
GET   /spectacles/<n>/seances
GET   /spectacles/<m>/seances
```

- Un nouveau spectacle :

```
POST   /spectacles/
POST   /spectacles/<n>/seances
PUT    /spectacles/<n>/lieu
```

- Changer l'horaire d'une séance d'un spectacle :

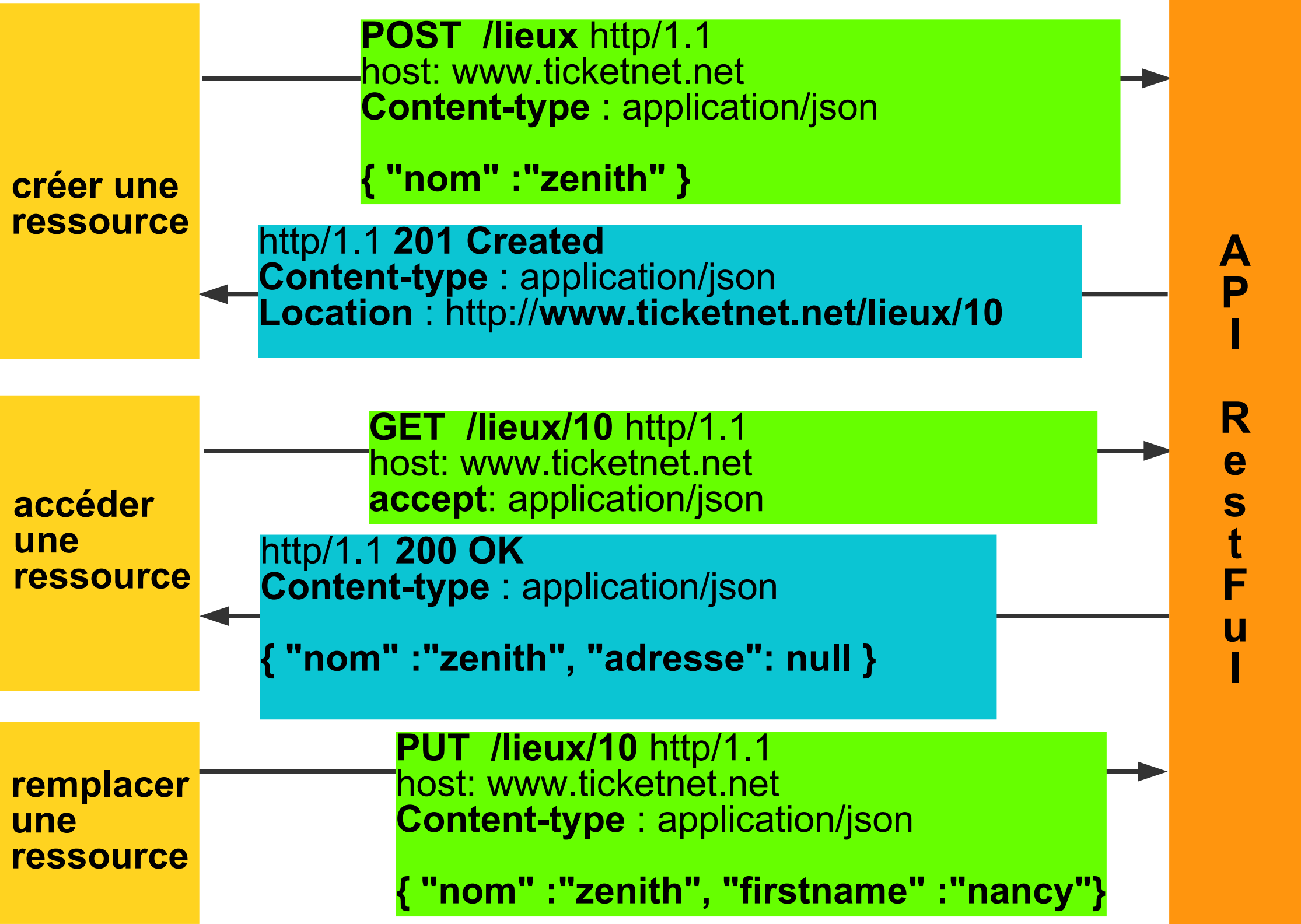
```
GET   /spectacles/45443/seances
PUT   /seances/<n>
```

bonnes pratiques sur les URI

- utiliser des noms d'entité, éviter les verbes
- pour les collections/endpoints : utiliser le pluriel
  - `/dogs/ ; /cats/`
- pour les ressources : pluriel + ID
  - une ressource est un élément dans une collection
  - /dogs/42
- pour les associations N : pluriel
  - /owner/73/cats
- pour les associations 1 : singulier
  - /cats/73/owner
- pour les associations : éviter les uri trops longues
  - ~~/cats/22/owner/dogs/42/brothers~~

le protocole REST

- Basé entièrement sur HTTP :
  - Un **message** REST = 1 **message** HTTP
  - **l'URI** désigne la **ressource**
  - la **méthode** HTTP désigne **l'action** à réaliser sur une ressource
    - ➔GET, POST, PUT, DELETE, (PATCH)
  - le **status** de la réponse HTTP est un **code de retour** indiquant les succès ou les erreurs d'exécution
  - le **body** du message HTTP contient les **données** transférées
  - les **headers** complètent les messages http, en particulier pour préciser le type des données attendues/envoyées


créer une ressource
POST /lieux http/1.1
host: www.ticketnet.net
Content-type : application/json
{ "nom" :"zenith" }
http/1.1 201 Created
Content-type : application/json
Location : http://www.ticketnet.net/lieux/10
accéder une ressource
GET /lieux/10 http/1.1
host: www.ticketnet.net
accept: application/json
http/1.1 200 OK
Content-type : application/json
{ "nom" :"zenith", "adresse": null }
remplacer une ressource
PUT /lieux/10 http/1.1
host: www.ticketnet.net
Content-type : application/json
{ "nom" :"zenith", "firstname" :"nancy"}
API RestFul

les requêtes

action : GET, POST, PUT, DELETE

uri de la ressource

```
POST  /lieux   HTTP/1.1
Host: www.tikenet.net
Accept: application/json, application/xml
Content-Type: application/json;charset=utf8

{ "nom":"le zainite", "capacite" : 150 }
```

Accept : type des données ***attendues*** par le client dans la réponse du serveur

```
application/json
application/xml
```

Content-Type : type des données ***envoyées*** par le client (POST ou PUT)

```
application/x-www-form-urlencoded
application/json
```

les requêtes

- la **méthode** http désigne l'opération
- l'**URI** désigne la ressource
- **Headers** :
  - **Accept :** désigne le formet des données **attendues** dans la réponse : application/json ; application/xml
  - **Content-Type :** format des données **transmises** : application/json ; application/xml ; application/x-www-form-urlencoded
- le **body** contient les données dans le cas d'un POST ou un PUT

les réponses

code de retour indiquant le succès ou 1 erreur

```
HTTP/1.1 201 CREATED
Location: /lieux/128
Content-Type: application/json

{"lieu" : {
        "nom" : "le zainite" ,
        "capacite" : 150 ,
        "id" : 128
 }}
```

uri de la nouvelle ressource (POST)

Content-Type : type des données ***retournées*** par le serveur

```
application/xml
application/json
```

body: la représentation de la ressource créée (POST) ou demandée (GET)

les réponses

- la **status line** contient un code de retour pour l'action

- **Headers :**
  - **Content-Type**: format de représentation de la ressource retournée : application/json , application/xml
  - **Location**: indique l'uri de la ressource créée dans le cas d'un POST ou d'un PUT

- **body** : représentation de la ressource demandée,supprimée, créée ou supprimée

réponses du serveur : status

- on utilise les status HTTP :

`200 OK`	GET, PUT, DELETE	succès de la requête, un contenu est retourné
`204 NO CONTENT`	PUT, DELETE	succès de la requête, pas de contenu retourné
`201 CREATED`	POST, PUT	succès avec création d'une ressource
`400 BAD REQUEST`	GET, PUT, POST, DELETE	erreur de requête : URI non valide, ou données manquantes
`401 UNAUTHORIZED`	GET, PUT, POST, DELETE	accès nécessitant une autorisation
`404 NOT FOUND`	GET, DELETE PUT	la ressource désignée n'a pas été trouvée (en général, id invalide ou inexistant)
`405 NOT ALLOWED`	GET, PUT, POST, DELETE	méthode non autorisée sur la ressource désignée
`500 Internal Server Error`	GET, POST, PUT DELETE	erreur interne du serveur : erreur d'accès BD, etc..,

les principes REST

- **Client-Serveur :** un ***client*** transmet des ***requêtes*** à un ***serveur*** qui lui retourne des ***réponses*** ;
- Basé sur **HTTP** : une requête REST est une requête HTTP, une réponse REST est une réponse HTTP
- **interface uniforme :** la forme d'une API REST est ***indépendante*** de l'application et des ressources :
  - les **ressources** sont toujours désignées de la même manière : **URI**
  - les **opérations** sont **indépendantes** des **ressources**, et limitées à 4 actions de base : GET, POST, PUT, DELETE

Les principes REST

- **sans état :** le serveur ne conserve aucune information liée au client : **pas de session sur le serveur !**
  - chaque requête doit contenir toutes les informations nécessaires à son exécution : ressource cible, action, types des données transmises et attendues, autorisations ...