# Développement back-end

API REST

# API REST

## Introduction

# API REST
## Introduction

REST (REpresentational State Transfer) est un style d'architecture pour les applications Web.

REST n'est pas un framework : il repose sur les concepts du protocole HTTP.

L'objectif est de fournir **une API auto-descriptive**, **simple** à utiliser pour le client.

Une API REST expose donc des **services**, utilisables par n'importe quel client HTTP, en restant agnostique au langage et aux librairies utilisés par le client.

# API REST
## Introduction

Les API REST offrent différents avantages :

- **Code plus simple**, car séparation claire entre le client et le serveur
- Maintenir une API “stateless” permet de **limiter la quantité de mémoire** nécessaire sur le serveur. Elle évite également de devoir maintenir une connexion ouverte entre le client et le serveur.
- La **scalabilité** de l’application est facilitée
- Une API auto-descriptive permet à un client d’**inférer le fonctionnement** de l’application

# API REST

## Ressource

# API REST
## Ressource

La ressource est l'élément fondamental d'une API REST.

Une ressource est un objet défini par :

- Un type
- Des données qui lui sont associées
- Des relations avec d'autres ressources
- Une série de méthodes applicables à cette ressource

On peut faire une analogie entre les ressources d'une API REST et une instance d'objet dans les langages orientés objet.

# API REST
## Ressource


```
{
 "id": 12,
 "gender": "female",
 "title": "ms",
 "first": "manuela",
 "last": "velasco",
 "location": {
   "street": "1969 calle de alberto aguilera",
   "city": "la coruña",
   "state": "asturias",
   "zip": "56298"
 },
 "emails": ["manuela.velasco50@example.com","another.email@gmail.com"]
}
```

# API REST
## Ressource

Les ressources peuvent être regroupées dans des collections. Une collection est une liste de ressources homogène (de même type).

Les collections peuvent être définies au niveau le plus haut de l'API (par exemple : *liste des utilisateurs*) ou bien être contenue dans une autre ressource (par exemple : *liste des adresses emails d'un utilisateur*).

# API REST
## Ressource

Les données qui composent une ressource peuvent être exprimées dans différents formats : XML, JSON, …

Le format JSON est aujourd'hui majoritairement utilisé, car il offre de nombreux avantage par rapport à XML :

- Typage (simple) des données : nombre, chaîne de caractère, booléen, tableau, objet
- Moins verbeux : plus léger à transporter
- Plus simple à intérpréter par le client

# API REST
## Ressource

Sur le modèle d'HTML, les ressources d'une API REST sont également composées de *metadata* qui définissent les relations entre les ressources.

Ces relations sont exprimées sous la forme d'un objet avec un attribut *href* (hyperlien). Ces hyperliens sont des URLs absolues qui peuvent être utilisées pour récupérer les données liées à la ressource actuelle.

# API REST
## Ressource

```
{
 "id": 12,

 (...)

 "links": [
   {
     "rel": "orders",
     "link": "http://api.example.com/api/v1/users/12/orders"
   },
   {
     "rel": "cards",
     "link": "http://api.example.com/api/v1/users/12/cards"
   },
 ]
}
```

# API REST

## HTTP

# API REST
## HTTP

L'**URI** (Uniform Resource Identifier) permet d'identifier de façon unique une ressource sur le réseau.

Elle est fondamentale dans les API REST : c'est le système global et universel pour identifier ce qui peut être récupéré par le client.

Elle offre de nombreux avantages : mise en cache, favoris, liens, ...

# API REST
## HTTP

Lors de la conception d'une API REST, on va faire attention à construire des URI indiquant clairement le type de ressources concernées.

Par exemple, pour récupérer la liste des clients :

**http://www.example.com/api/v1/clients**

Un client spécifique par son identifiant :

**http://www.example.com/api/v1/clients/{ID}**

Par convention, on va généralement utiliser la forme au pluriel des noms.

# API REST
## HTTP

HTTP est un protocole simple et sans état :

- La “conversation” se limite à une requête et une réponse
- Le serveur ne conserve rien des requêtes précédentes. Le client doit passer le contexte nécessaire à l’éxécution de chaque requête
- Les différentes opérations disponibles sont limitées : GET, POST, PUT, DELETE, HEAD, PATCH

Une API REST doit donc se baser uniquement sur ce qui est offert par HTTP.

# API REST
## HTTP

Les **verbes** d'HTTP ont une signification particulière :

- **GET** : récupérer une ressource
- **POST** : créer une ressource
- **PUT** : modifier une ressource
- **DELETE** : supprimer une ressource

HEAD et PATCH sont moins fréquemment utilisés : **HEAD** permet de récupérer les headers d'une ressource et **PATCH** permet de mettre à jour *certaines propriétés* d'une ressource.

# API REST

## Réponses

# API REST
## Réponses

Les réponses d'une API REST doivent respecter le standard HTTP.

Elles doivent faire référence aux codes HTTP standardisés :

- **1xx** : Réponses informatives
- **2xx** : Réponses de succès
- **3xx** : Réponses de redirections
- **4xx** : Réponses indiquant une erreur dans la requête du client
- **5xx** : Réponses indiquant une erreur dans le traitement côté serveur

# API REST
## Réponses

En plus du code standard HTTP, l'API REST doit répondre avec des données utiles pour le client.

En réponse à une requête GET, l'API retournera la ressource demandée.

En réponse à une requête POST ou PUT, l'API retournera la ressource telle qu'elle a été insérée ou mise à jour.

Pour une requête DELETE, il est fréquent de retourner simplement une confirmation. Dans ce cas, on utilisera le code HTTP 204 (No Content)

# API REST
## Réponses

En cas d'erreur, il est important de préciser au client des informations sur l'erreur survenue, en plus du code standard HTTP.

Par exemple, l'API REST pourra retourner la réponse suivante :

```
HTTP/1.1 404 Not Found
Content-Type: application/json



{
 "error": "Client not found",
 "error_description": "The client with ID 1234 was not found.",
 "link": "http://api.example.code/docs/errors/client#404"
}
```

# API REST

## Design d’une API

# API REST
## Design d’une API

L’objectif lors du design d’une API REST est donc :

- **De concevoir une API intuitive pour le client** : via l’utilisation de noms logiques dans les URIs, d’hyperliens permettant de naviguer entre les ressources, d’utiliser les verbes HTTP a bon escient
- **De conserver une API stateless**
- **D’assurer une cohérence dans le fonctionnement** des méthodes des différentes ressources et collections

# API REST
## Design d’une API

Un exemple : concevoir une API permettant de gérer une liste de client.

Fonctionnalités : lister les clients existants, récupérer un client spécifique, créer / modifier / supprimer un client, ajouter des commandes pour un client et lister les commandes d’un client.

On a donc 2 types de ressources : le **Client** et la **Commande**. L’API va être composée d’une collection de Clients, chacun composé d’une liste de commande.

# API REST
## Design d'une API

Création d'un client :

**Méthode :** POST (création)
**URI :** http://www.example.com/api/v1/clients
**Payload :** Un objet de type client, contenant l'ID du client créé
**Réponse :**

- **201 Created** si le client a bien été créé, en retournant les données insérées et un lien vers le détail d'un client
- **409 Conflict** si un client existe déjà avec l'adresse email fournie

# API REST
## Design d’une API

- Requête :

```
POST /api/v1/clients
{
  "gender": "female",
  "title": "ms",
  "first": "manuela",
  "last": "velasco",
  "location": {
    "street": "1969 calle de alberto aguilera",
    "city": "la coruña",
    "state": "asturias",
    "zip": "56298"
  },
  "emails": ["manuela.velasco50@example.com","another.email@gmail.com"]
}
```

# API REST
## Design d’une API

- Réponse :

```
HTTP/1.1 201 Created

{
  "id": 12,
  "gender": "female",
  "title": "ms",

        (...)

  "emails": ["manuela.velasco50@example.com","another.email@gmail.com"],
  "link": "http://api.example.com/api/v1/clients/12",
}
```

# API REST
## Design d’une API

Mise à jour d’un client :

**Méthode :** PUT (mise à jour)
**URI :** http://www.example.com/api/v1/clients/123
**Payload :** L’objet “Client” avec les toutes les données, dont celles modifiées
**Réponse :**

- **200 OK** si le client a bien été mise à jour, en retournant les données insérées

# API REST
## Design d’une API

- Requête :

```
PUT /api/v1/clients/12
{
  "gender": "female",
  "title": "ms",
  "first": "manuela",
  "last": "velasco",
  "location": {
    "street": "1969 calle de alberto aguilera",
    "city": "la coruña",
    "state": "asturias",
    "zip": "56298"
  },
  "emails": ["manuela.velasco50@example.com","another.email@gmail.com"]
}
```

# API REST
## Design d’une API

- Réponse :

```
HTTP/1.1 200 OK

{
  "id": 12,
  "gender": "female",
  "title": "ms",

        (...)

  "emails": ["manuela.velasco50@example.com","another.email@gmail.com"],
  "link": "http://api.example.com/api/v1/clients/12",
}
```

# API REST
## Design d’une API

Suppression d’un client :

**Méthode :** DELETE (suppression)
**URI :** http://www.example.com/api/v1/clients/123
**Payload :** Aucun payload n’est requis dans ce cas
**Réponse :**

- **204 No Content** si le client a bien été supprimé. La réponse ne contiendra par d’autres données.

# API REST
## Design d’une API

- Requête :

```
DELETE /api/v1/clients/12
```

- Réponse :

```
HTTP/1.1 204 No Content
```

# API REST
## Design d’une API

Lister les clients existants

**Méthode :** GET (récupération)
**URI :** http://www.example.com/api/v1/clients
**Payload :** Aucun payload n’est requis dans ce cas
**Réponse :**

- **200 OK**, avec une liste de ressources de type Client, chacune possédant un lien vers le détail de ce client

# API REST
## Design d’une API

- Requête :

```
GET /api/v1/clients
```

# API REST
## Design d’une API

- Réponse :

```
HTTP/1.1 200 OK
[
   "http://api.example.com/api/v1/clients/12",
   "http://api.example.com/api/v1/clients/13"
]
```

# API REST
## Design d’une API

Récupérer un client spécifique

**Méthode :** GET (récupération)
**URI :** http://www.example.com/api/v1/clients/123
**Payload :** Aucun payload n’est requis dans ce cas
**Réponse :**

- **200 OK**, avec la ressource Client, et notamment un lien vers la liste des commandes de ce client

# API REST
## Design d’une API

- Requête :

```
GET /api/v1/clients/12
```

# API REST
## Design d’une API

- Réponse :

```
HTTP/1.1 200 OK

{
  "id": 12,
  "gender": "female",
  "title": "ms",
       (...)
  "emails": ["manuela.velasco50@example.com","another.email@gmail.com"],
  "links": [
    {
      "rel": "orders", "link": "http://api.example.com/api/v1/clients/12"
    }
  ]
}
```

# API REST
## Design d’une API

Récupérer la liste des commandes d’un client spécifique

**Méthode :** GET (récupération)
**URI :** http://www.example.com/api/v1/clients/123/orders
**Payload :** Aucun payload n’est requis dans ce cas
**Réponse :**

- **200 OK**, avec la liste des ressources Orders de ce client, chacune possédant un lien vers le détail de cette commande

# API REST
## Design d’une API

- Requête :

```
GET /api/v1/clients/12/orders
```

# API REST
## Design d’une API

- Réponse :

```
HTTP/1.1 200 OK

[
  {
    "id": 100,
    "product": {
       "id": 19021,
       "label": "MacBook Pro",
       "href": "http://api.example.com/api/v1/products/19021"
    },
       (...)
    "href": "http://api.example.com/api/v1/clients/12/orders/100",
  },
   (...)
]
```

# API REST
## Design d’une API

Ajouter une commande à un client

**Méthode :** POST (création)
**URI :** http://www.example.com/api/v1/clients/123/orders
**Payload :** Un objet Order avec les données à insérer
**Réponse :**

- **201 Created**, avec l’objet Client tel qu’il a été inséré

# API REST

## Remarques

# API REST
## Remarques

**Versionning**

Il est important d'indiquer une référence à la version de l'API. Une façon courante est d'inclure le numéro de version dans les URI de l'API REST :

http://www.example.com/api/**v1**/clients/...

Cela permet d'offrir différentes versions de l'API en parallèle, pour assurer la compatibilité avec les anciens clients

# API REST
## Remarques

**Action URI**

Dans certains cas (d'exception), l'API devra exposer des méthodes qui ne sont pas compatibles avec les principes REST. Ce sont des actions qui vont modifier l'état d'une ressource par exemple, mais sans passer par la mise à jour complète de la ressource (par exemple : *désactiver un utilisateur*).

# API REST
## Remarques

**Action URI**

Une solution dans ce cas peut être d'utiliser une sous-collection d'actions. Par exemple :

**POST http://api.example.com/api/v1/clients/12/actions**
**{"type": "disable"}**

On peut considérer dans ce cas que "actions" est une sorte de queue d'action à effectuer sur la ressource. L'attribut "type" indique l'action à effectuer.

# API REST
## Remarques

**Pagination**

Dans les requêtes qui listent plusieurs ressources, il peut être intéressant de laisser le client indiquer des paramètres de pagination.

Plusieurs solutions sont utilisées :

- Utiliser les “query string parameters” : **?range=1-100**
- Utiliser le header HTTP spécifique “Range” : **Range: resources=1-100**

# API REST
## Remarques

**Pagination**

Les réponses avec pagination devraient être retournées avec le code HTTP **206 Partial Content**. Elles devraient également inclure un header HTTP :

**Content-Range offset – limit / count**

Avec **offset** l'index du premier élément dans la réponse ("1"), **limit** l'index du dernier élément dans la réponse ("100") et **count** le nombre d'éléments retournés ("100")

# API REST
## Remarques

**Filtres**

Les filtres sont fréquemment utilisés pour restreindre la liste des éléments retournées lors d'une requête vers une collection de ressource. On utilise généralement des paramètres dans l'URL, par exemple :

**http://www.example.com/api/v1/clients?status=enabled&minOrders=10**

# API REST
## Remarques

**Tri**

De la même façon, les tris sont généralement passés dans les paramètres de l'URI :

**http://www.example.com/api/v1/clients?sort=name,lastOrder&desc=lastOrder**

Par convention, l'ordre par défaut est l'ordre ascendant. Sinon, l'ordre est précisé dans un second paramètre de l'URI.