**IUT Nancy-Charlemagne – BUT Informatique S5 – DWM**

**Développement Web Serveur avancé**

# TD1.5 : API Restful : conception et programmation

## Partie 1: principes REST

On souhaite développer une API Restful pour interagir avec une plateforme de commerce en ligne. Pour cette API, il a été choisi d'exposer les ressources décrites dans le diagramme UML suivant:

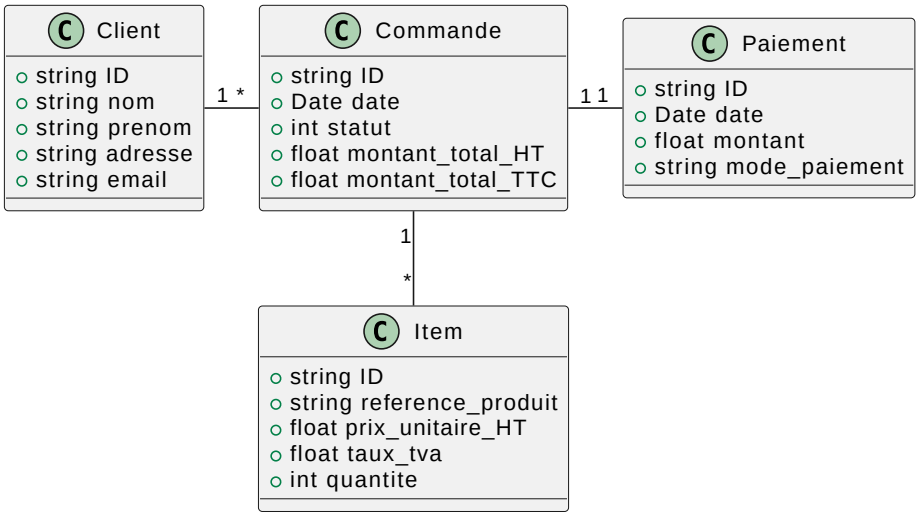


*Figure 1. Diagramme des ressources exposées par l'API Commande*

### Exercice 1: API v1

Un développeur (stagiaire) a commencé à concevoir une API pour gérer les commandes. Voici ce qu'il propose. Pour chaque cas, indiquez si vous pensez que c'est conforme, ou non, à REST et pourquoi.

- `GET /list-commandes` : retourne la liste des commandes, status `200`, ou `404` en cas d'erreur de connexion à la BD
- `GET /commandes/{ id }` : retourne la commande `id` , status `200`, ou `404` si la commande n'existe pas
- `POST /commandes/add` : crée une commande, status `201`, ou `404` si les données sont incorrectes
- `PUT /commande/set-paiement/{ id }` : enregistre le paiement de la commande `id`, status `200`, ou `404` si la commande n'existe pas
- `GET /commande/get_client/{ id }` : retourne les informations du client de la commande `id`, status `200`, ou `404` si le client n'existe pas

### Exercice 2: API v2

Le lead dev du projet (vous ;-) prend les choses en main et conçoit l'API en se conformant strictement aux principes REST.

Identifiez les entrypoints de votre API, puis, pour chacun des cas ci-dessous, indiquez la **route** à utiliser, la **méthode HTTP** à utiliser, le **statut de retour en cas de succès**, et le **statut de retour en cas d'erreur** (différents cas possibles):

1. créer une commande
2. ajouter un Item à une commande
3. définir le client d'une commande
4. lister les commandes d'un clients
5. lister les Items d'une commande
6. obtenir une commande en incluant le client et les Items
7. payer une commande

### Exercice 3: API et données échangées

Proposez une structure JSON pour les données échangées entre le client et l'API dans le cas 1 et le cas 6. Il faudra notamment prévoir les liens HATEOAS dans les données retournées par l'API.

# Partie 2: API RESTful pour le projet Toubilib

Cette partie est consacrée à l'application des principes RESTful dans le contexte du projet Toubilib. L'objectif est de vérifier que l'API construite jusqu'ici se conforme strictement aux principes RESTFul sur l'ensemble des aspects

- méthodes HTTP utilisées,
- URI et désignation des ressources,
- statuts de retour, dans les cas de succès et dans les cas d'erreur,
- données échangées au format JSON, et ajout des liens HATEOAS,

Pour l'instant, on ne se préoccupe pas d'authentification et de contrôle d'accès.

Tous les tests et essais sont à réaliser avec une application type Bruno (*usebruno.com*), Postman (*www.postman.com*), HTTPie (*httpie.io*) ou Insomnia (*insomnia.rest*).

# Exercice 1: API Restful pour la gestion les rendez-vous

Appliquer les principes RestFul pour déterminer les routes (méthode HTTP et URI) à utiliser pour les cas suivants, ainsi que les statuts de retour en cas de succès ou d'erreur. . lister les rendez-vous d'un praticien (uniquement les créneaux occupés) . obtenir l'agenda d'un praticien pour une période donnée . obtenir le détail d'un rendez-vous . créer un rendez-vous . annuler un rendez)vous . honorer un rendez-vous . ne pas honorer un rendez-vous

## Exercice 2: accéder à un rendez-vous

Mettez en forme RestFul la fonctionnalité d'accès à un RDV :

- On utilise la méthode `GET`
- l'URL à utiliser doit avoir la forme: `⋯/rdvs/{ ID-RDV }`
- Les données en réponse sont formatées en JSON et accompagnées de quelques informations comme indiqué dans l'exemple donné en annexe en incluant des liens HATEOAS.

En cas de succès, la réponse doit retourner un statut `200` et les données du rendez-vous au format JSON, avec les liens HATEOAS. En cas d'erreur, un message JSON devra être retourné, dans une réponse avec un statut adéquat (correspondant à l'erreur détectée). En particulier, si l'identifiant fourni ne correspond à aucun rendez-vous existant, la réponse doit retourner un statut `404`.

## Exercice 3: gérer des rendez-vous

Faites de même pour la fonctionnalité permettant d'obtenir l'agenda d'un praticien, puis pour la fonctionnalité d'annulation d'un rendez-vous.

## Exercice 4: CORS

Ajoutez la gestion des headers CORS à votre API.

Accéder à un rendez-vous :

`GET /rdvs/2e1a7275-2593-3c04-9a4c-4e7cbada9541`

```
HTTP/1.1 200 OK

{
    "type": "ressource",
    "rendez_vous": {
        "id": "2e1a7275-2593-3c04-9a4c-4e7cbada9541",
        "praticien_id": "4305f5e9-be5a-4ccf-8792-7e07d7017363",
        "patient_id": "5abcdbc4-90c9-3b86-82a3-c4cf1f7377d0",
        "date_debut": "04/12/2025 18:30",
        "date_creation": "25/11/2025",
        "duree": 30,
        "motif_visite": "IRM",
        "etat": 0
    },
    "links": {
      "self": {
        "href": "/rdvs/2e1a7275-2593-3c04-9a4c-4e7cbada9541/"
      },
      "praticien": {
        "href": "/praticiens/4305f5e9-be5a-4ccf-8792-7e07d7017363/"
      },
      "patient": {
        "href": "/patients/5abcdbc4-90c9-3b86-82a3-c4cf1f7377d0/"
      },
      "annuler": {
        "href": "/rdvs/2e1a7275-2593-3c04-9a4c-4e7cbada9541/annuler"
      },
      "honorer": {
        "href": "/rdvs/2e1a7275-2593-3c04-9a4c-4e7cbada9541/honorer"
      },
      "ne_pas_honorer": {
        "href": "/rdvs/2e1a7275-2593-3c04-9a4c-4e7cbada9541/ne-pas-honorer"
      }
    }
}
```

Réponse en cas d'erreur : rendez-vous inexistant

Les données JSON sont mise en forme par le traitement d'erreur par défaut de Slim.

```
HTTP/1.1 404 NOT FOUND
{
  "message": "404 Not Found",
  "exception": [
    {
    "type": "Slim\\Exception\\HttpNotFoundException",
```

```
    "code": 404,
    "message": "Rendez-vous b2dd56c2-700b-3b49-915d-d3be96f7f6a not found",
    "file": "...",
    "line": ...
     } ]
}
```