Libery
LAB

ESIEE - 5I-IN9 Développement web

Cours 3 – Services web

Libery
LAB

Il était une fois … l'interopérabilité

Cours 3 – Services web

Au commencement était …

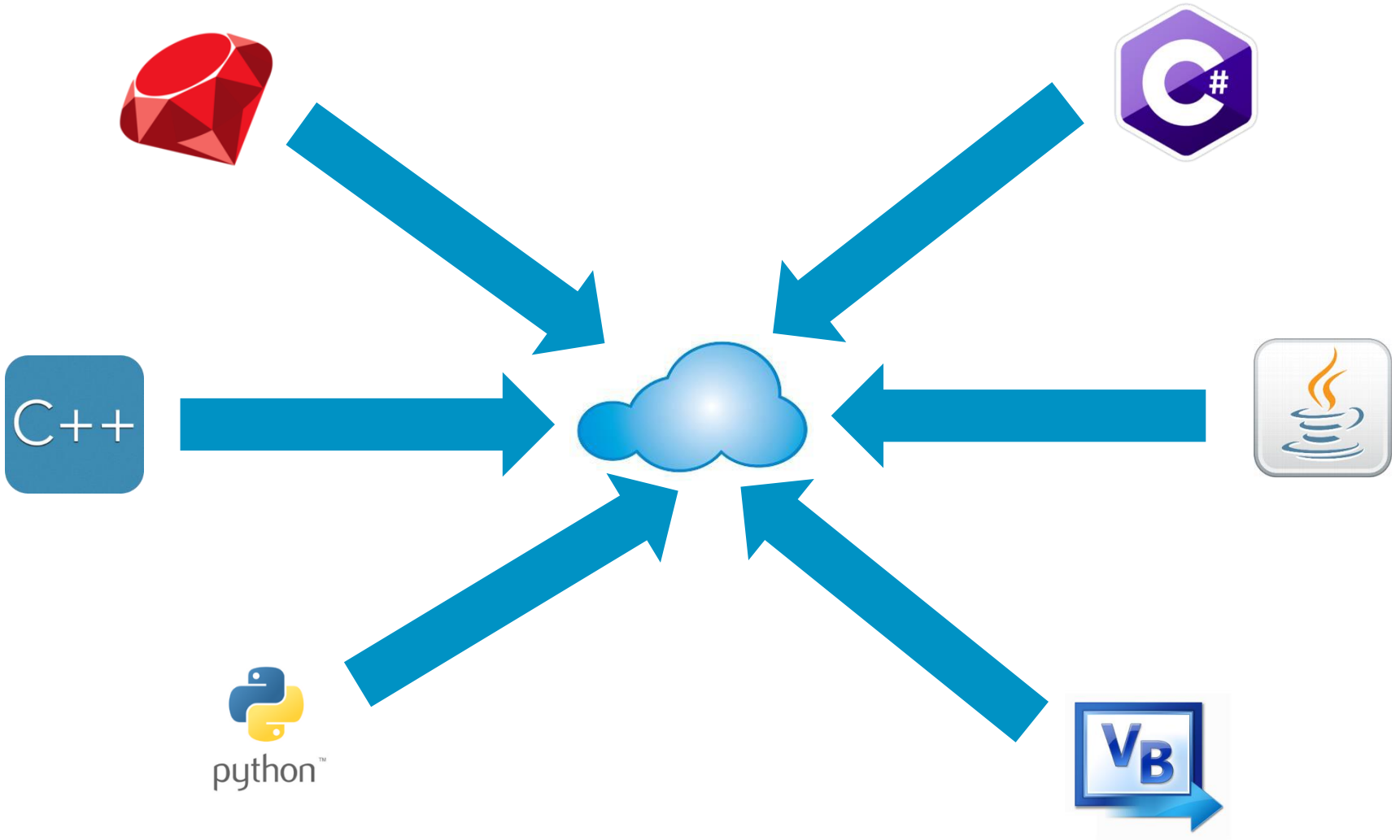
- Des systèmes informatiques indépendants
  - Une unité de calcule
  - Un système d'informations
  - Des fonctions élémentaires
  - Des procédures
  - Pas de communication entre deux applications

Libery LAB

Et puis les réseaux sont arrivés

- Architecture mainframe
  - Un serveur actif et des terminaux passifs
- Architecture 2-tier
  - Un Serveur et un client actif
- Architecture 3-tier
  - Un client, un serveur et une base de données
- Architecture multi-tier
  - Un client, plusieurs serveurs de ressources
- Architecture peer-to-peer
  - Plusieurs nœuds faisant chacun rôle de client et de serveur

Et l'interopérabilité est arrivé …

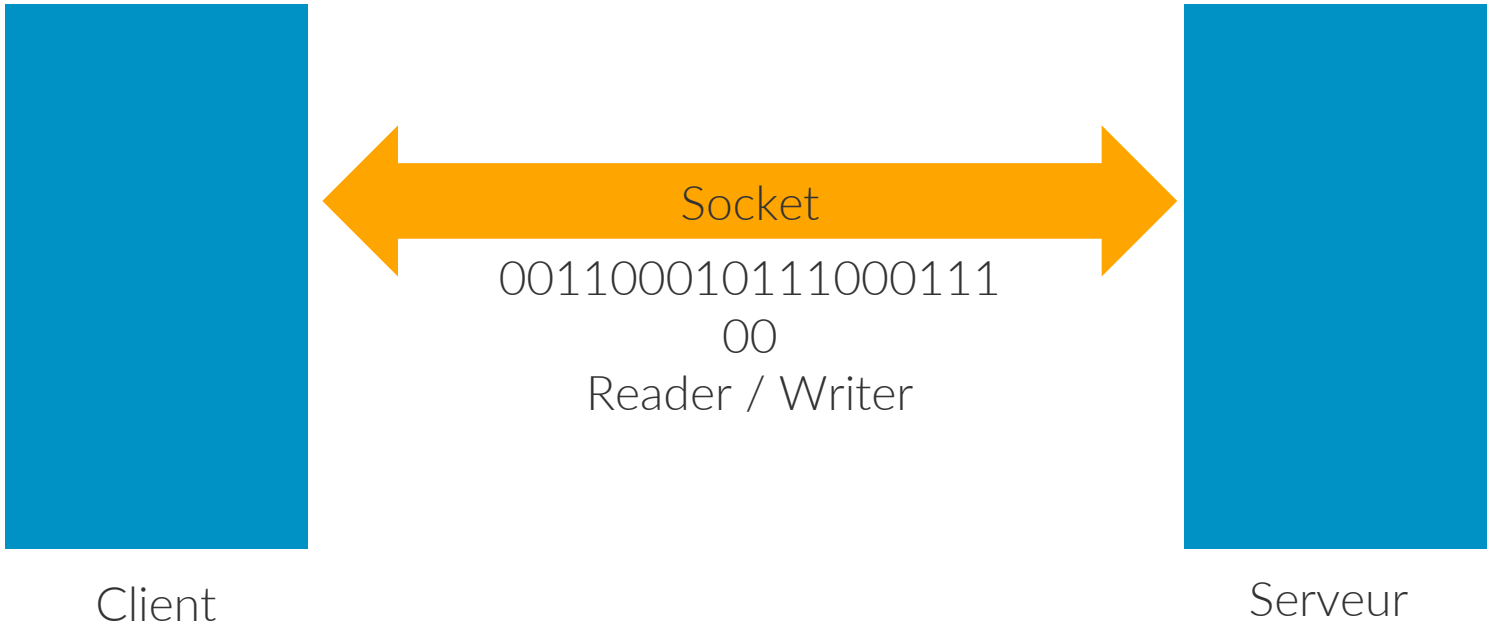


Et l'interopérabilité est arrivé …

Comment faire communiquer ces systèmes hétérogènes ?

Les sockets

- Socket bidirectionnelle



Les sockets

- Socket unidirectionnelle

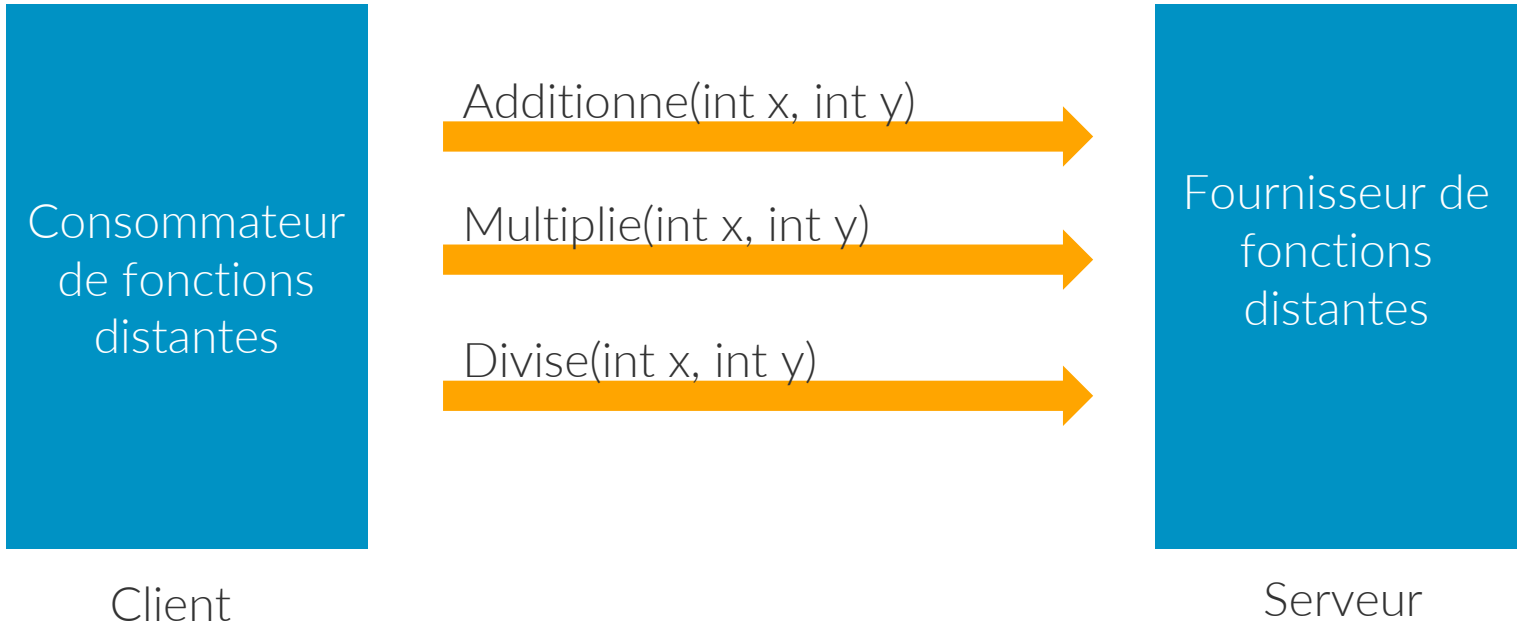
Les sockets

- Problèmes
  - Nécessite une synchronisation des flux
  - Nécessite un ordonnancement pour les messages longs
  - Nécessite de transmettre les données sous forme binaires
  - Nécessite de comprendre les couches de bas niveaux
  - Nécessite des ouvertures de ports pour passer les routeurs
  - Protocole à définir et non normalisé

- Avantages
  - Permet une bonne maitrise des performances
  - Permet l'interopérabilité dans certains contextes d'utilisation

RPC, RMI et ORB

- Exemple : Calculatrice distante



RPC : Remote Procedure Call
RMI : Remove Method Invocation
ORB : Object request broker

RPC, RMI et ORB

- Quelques implémentations phares
  - CORBA : Common Object Request Broker Architecture
    - Une des premières implémentations
    - Permet de définir des applications avec traitements réparties
    - Permet l'interopérabilité des systèmes
    - Protocole de communication définit par des interfaces IDL : Interface description language

- Java RMI : Jave Remote Method Invocation
  - Permet de définir des applications avec traitements réparties
  - Protocole de communication définit par des interfaces Java
  - Compatible avec CORBA pour interopérabilité des systèmes
  - Définit dans le standard JEE
  - Utilise un registre de services

Et les EJB ?

- EJB : Enterprise Javaresources
  - Permet le partage d'instances d'objets sur le réseau
  - Se repose sur Java RMI
  - Interopérabilité des systèmes possible avec CORBA RMI/IDL

Et enfin …

- Architecture SOA
  - SOA : Service Oriented Architecture
  - Permet l'interopérabilité des systèmes par la mise en place d'un langage intermédiaire
  - Intégration sur le web : WOA : Web-oriented architecture

Libery LAB

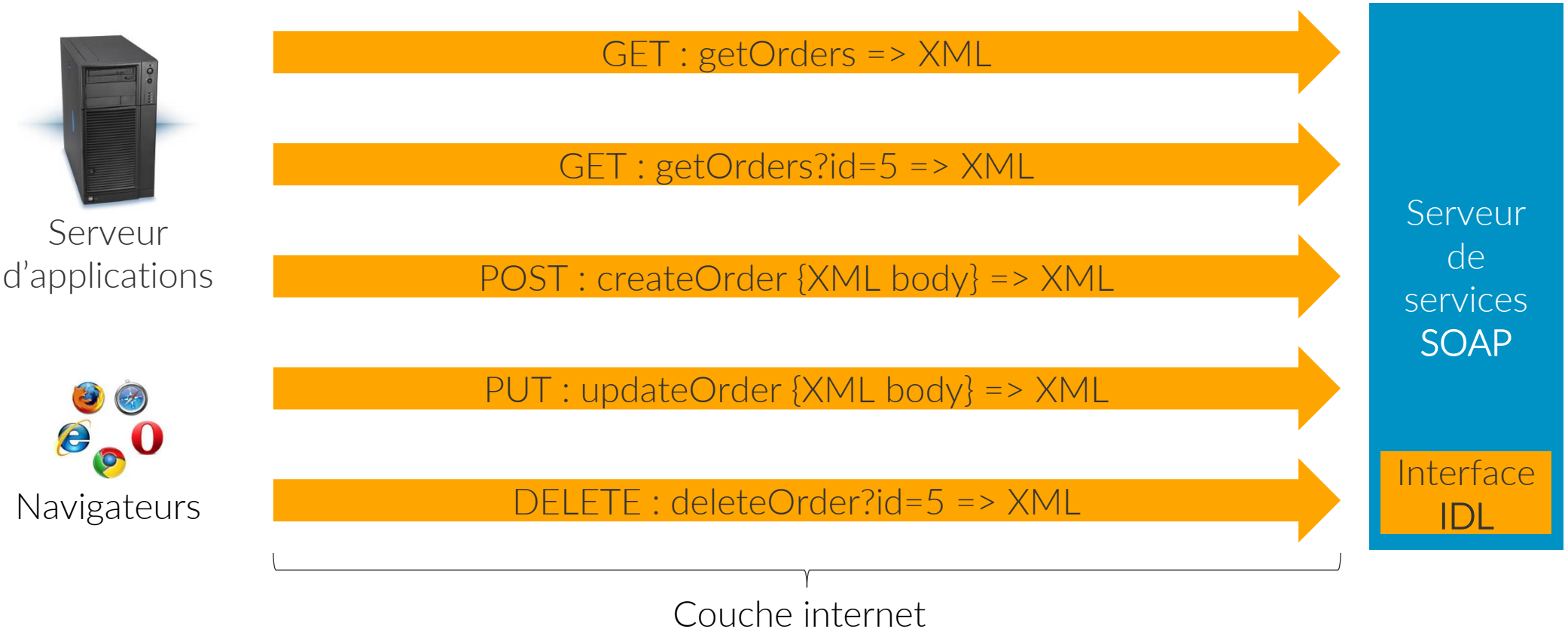
Architecture SOA/WOA

- Les deux grandes implémentations

SOAP: *Simple Object Access Protocol*
REST: *Representational State Transfer*

Architecture SOA/WOA

- Implémentation SOAP



Architecture SOA/WOA

- Implémentation REST

Serveur d'applications

Navigateurs

GET : orders => JSON

GET : orders/5 => JSON

POST : orders {JSON body} => JSON

PUT : orders {JSON body} => JSON

DELETE : orders/5 => JSON

Couche internet

Serveur de services REST

swagger

Architecture SOA/WOA

- Avantages
  - Répond aux problématiques d'interopérabilités
  - Faible couplage avec les consommateurs
  - Résous les problématiques d'ouvertures de ports par l'utilisation du protocole HTTP
  - Possibilité de découper les services par la mise en œuvre de plusieurs applications

- Inconvénients
  - Pose parfois des problématiques de performances pour respecter les normes
  - Plus ou moins souple selon l'implémentation choisie

Libery
LAB

Les services Web REST

Cours 3 – Services web

Services Web REST

- Les grands concepts
  - Orientation objet (un service = une ressource)
  - Manipulation des objets par des verbes (Opération)
  - Représentation des objets par du JSON (Serialization)
  - Pas de notion d'état (requêtes indépendantes)
  - Absence de langage protocolaire mais un pseudo standard (RESTfull)

Services Web REST

- Format d'échange des services REST, le JSON
  - Parfaitement adapté la sérialisation / désérialisation JavaScript
  - Format décrit dans la RFC 7159

```
{
        "firstname": "René",
        "lastname": "Dubos"
}
```

Libery LAB

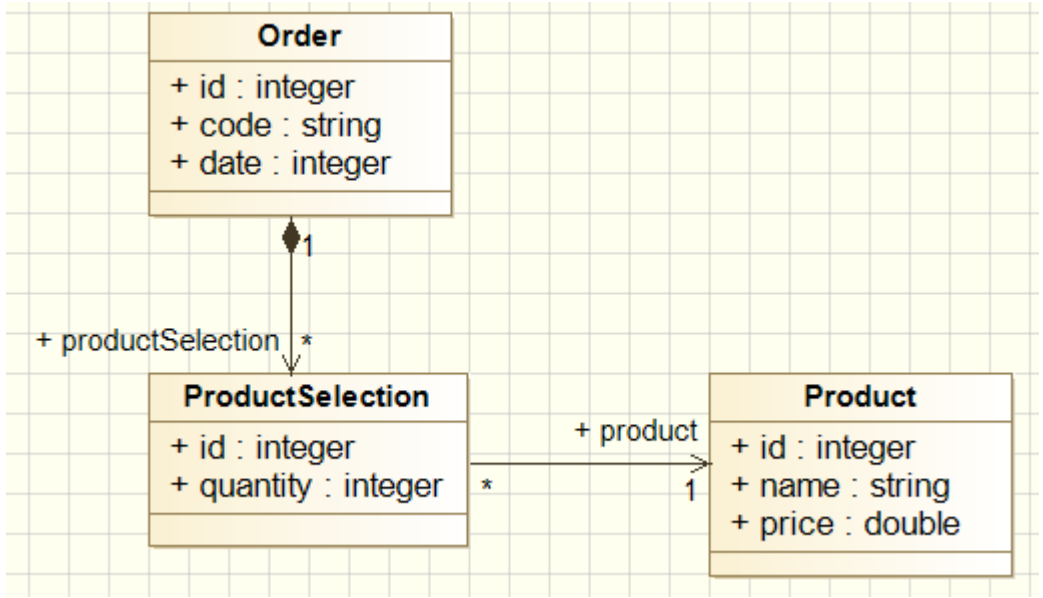
Services Web REST

- Les verbes HTTP
  - Les verbes représentent des opérations sur des ressources
    - GET : Permet la récupération d'objets
    - POST : Permet la création d'objets
    - PUT : Permet la modification d'objets
    - DELETE : Permet la suppression d'objets

$$9 - 3 \div \frac{1}{3} + 1 =$$

REST : Cas pratique

- Nous disposons du modèle de données suivant :



REST : Cas pratique

- Etape 1 : Définir notre API
  - On commence par définir l'url permettant d'accéder à notre API

http://<adresse du serveur>/api/

  - Notre API permettra l'accès au modèle de données mais également à des règles de gestions ou procédures, pour cela nous identifions les catégories par des URLs distincts comme suit :

http://<adresse du serveur>/api/resources/
http://<adresse du serveur>/api/businessProcess/

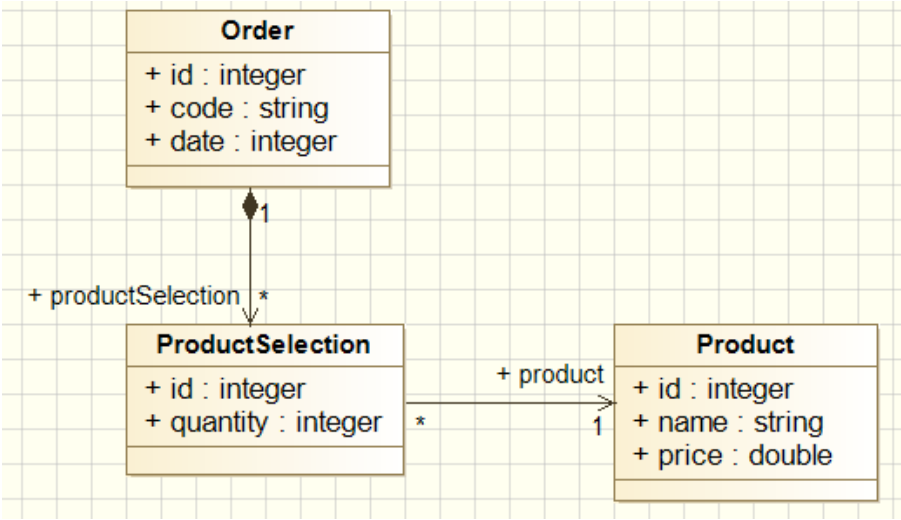
API: *Application Programming Interface*

REST : Cas pratique

- Etape 1 : Définir notre API
  - On détermine les règles comportementales de notre API afin d'obtenir des services homogènes
    - Un service par objet (objet = ressource)
    - Seule les relations de type « OneToOne » et « ManyToOne » sont étendues dans les résultats
    - Chaque service doit permettre la **création**, la **modification**, la **consultation** et la **suppression** d'un objet
    - La sauvegarde des relations se fait du côté où la cardinalité est la plus faible, en cas d'égalité entre les cardinalités, c'est le sens métier de la relation qui détermine le propriétaire du lien
    - Les sauvegardes d'objets retournent toujours l'objet mis à jour ou nouvellement créé

REST : Cas pratique

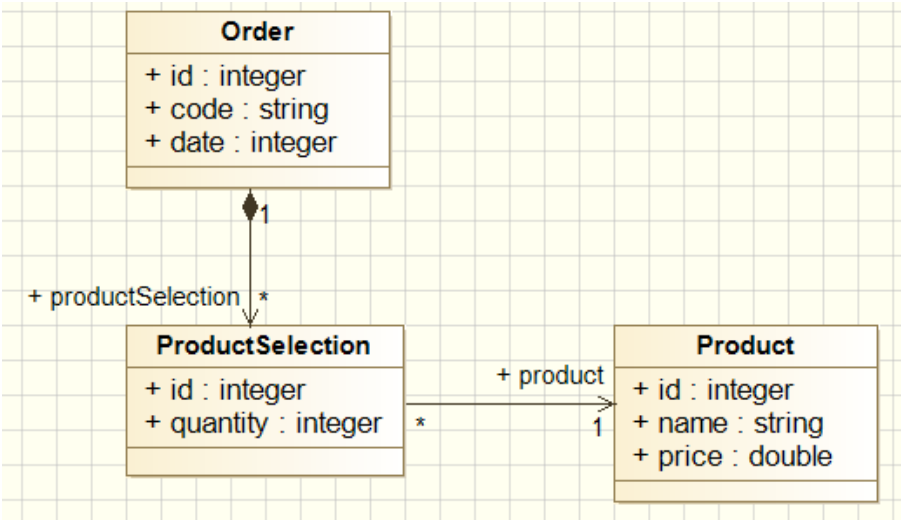
- Etape 1 : Définir notre API
  - On identifie ensuite les relations sur nos objets
    - Les commandes sont composées d'une sélection de produits (OneToMany)
    - Une sélection de produit est liée à un produit (ManyToOne)



REST : Cas pratique

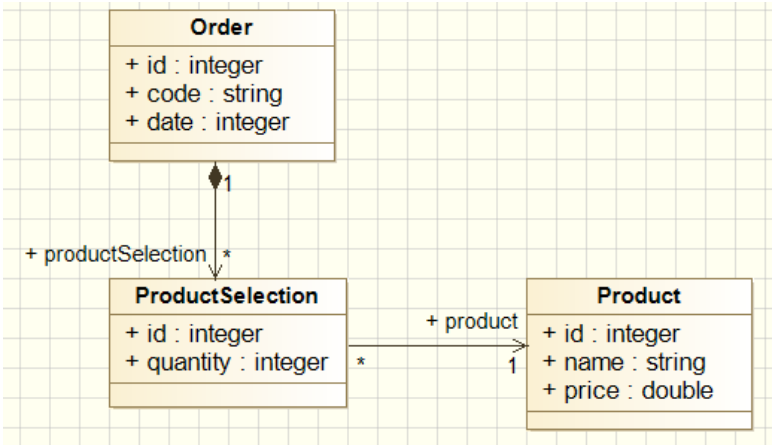
- Etape 1 : Définir notre API
  - On identifie ensuite nos objets pour en déduire des services

Order, ProductSelection, Product



REST : Cas pratique

- Etape 1 : Définir notre API
  - Nous obtenons les services suivants



/api/resources/orders
/api/resources/orders/{id}
/api/resources/orders/{id}/productSelections
/api/resources/orders/{id}/productSelections/{id}
/api/resources/productSelections
/api/resources/productSelections/{id}
/api/resources/products
/api/resources/products/{id}

REST : Cas pratique

- Etape 2 : Sécurisation de l'API
  - Mise en place d'un service d'authentification permettant de récupérer un **token** à partir du couple **login/password**
  - Mise en place d'un mécanisme d'expiration du token (*ex : 20 minutes*)
  - Chiffrement des appels REST (*HTTPS*)

/api/businessProcess/authentication

- **Get** : Récupération des informations d'authentification
- **Post** : Authentification (*body avec login/password*)
- **Delete** : Suppression des données de l'authentification

HTTPS = HTTP + SSL
SSL : Secure Socket Layer

Libery LAB

REST : Cas pratique

- Etape 3 : Implémentation de l'API
  - Sélection des technologies (dans le cadre du cours)
    - Modélisation des services par des servlets
    - Utilisation du parser JSON « Jackson »
    - Utilisation de la librairie Jquery pour les appels fronts
    - Utilisation du plugin Firefox « RESTClient » pour tester notre API

REST : Cas pratique

- Etape 3 : Implémentation de l'API
  - Ajouter « Jackson » au projet
    - Ajouter les lignes suivantes dans le fichier « pom,xml » entre les balise « dependencies »

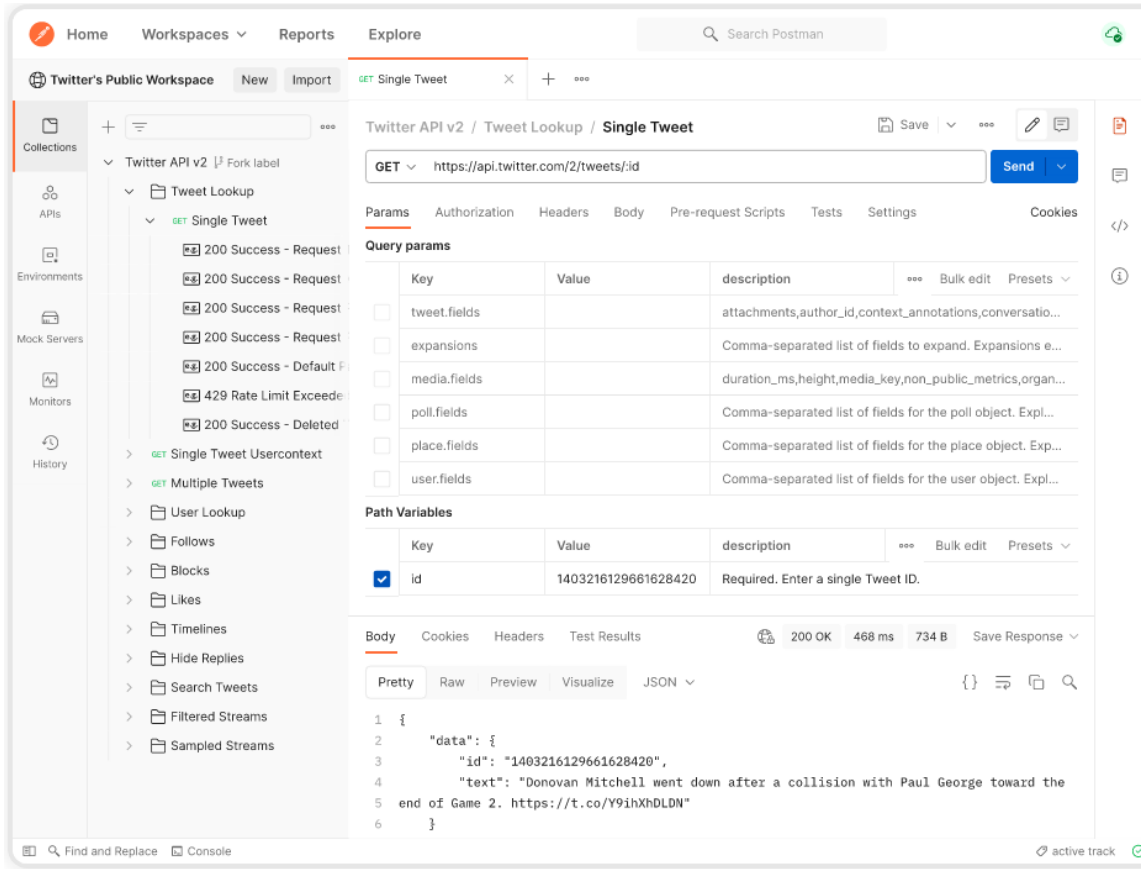
```
<dependency>
    <groupId>com.fasterxml.jackson.datatype</groupId>
    <artifactId>jackson-datatype-jsr310</artifactId>
    <version>2.6.1</version>
</dependency>
```

Faire un « maven update project » en cas de problème

Libery LAB

REST : Cas pratique

- Etape 3 : Implémentation de l'API
  - Télécharger l'outil « **Postman** »
    - https://www.postman.com/downloads/



REST : Cas pratique

- Etape 3 : Implémentation de l'API
  - Construction des POJOs serialisable avec Jackson

```java
public class Product implements Serializable {

    private static final long serialVersionUID = 1L;
    private Long id;
    private String name;
    private Double price;
```

*Jackson a besoin que l'objet implémente l'interface « **Serializable** » pour fonctionner*

Libery LAB

REST : Cas pratique

- Etape 3 : Implémentation de l'API
  - Construction des POJOs serialisable avec Jackson

```java
public class ProductSelection implements Serializable {

    private static final long serialVersionUID = 1L;
    private Long id;
    private Integer quantity;
    private Product product;

    @JsonIgnore
    private Order order;
```

*L'attribut « **order** » est obligatoire pour la sauvegarde des « **ProductSelection** » bien qu'il ne soit pas présent dans le model* (sens de la relation)

Libery LAB

REST : Cas pratique

- Etape 3 : Implémentation de l'API
  - Construction des POJOs serialisable avec Jackson

```java
public class Order implements Serializable {

    private static final long serialVersionUID = 1L;
    private Long id;
    private String code;
    private Long date;

    @JsonIgnore
    private List<ProductSelection> productSelections;

    public Order() {
        productSelections = new ArrayList<ProductSelection>();
    }
```

@JsonIgnore : Ignore l'attribut dans la construction du JSON

REST : Cas pratique

- Etape 3 : Implémentation de l'API
  - Mise en place du premier service de type « **ressource** »

```
@WebServlet("/api/resources/products/*")
public class ProductResource extends HttpServlet {
```

Remarque : *Une bonne nomenclature de code permet un code clair*

Libery LAB

REST : Cas pratique

- Etape 3 : Implémentation de l'API
  - Mise en place du premier service de type « ressource »
    - Mise en place d'un outil d'analyse d'URI pour simplifier les traitements « ResourceUriAnalyser »

```
public static boolean hasIdParameter(HttpServletRequest req)
public static Long getIdParameter(HttpServletRequest req)
public static boolean hasFirstRelationParameter(HttpServletRequest req)
public static String getFirstRelationParameter(HttpServletRequest req)
```

Pour la réalisation des traitements, les méthodes utilisent la méthode « getPathInfo » de la classe « HttpServletRequest »

Libery LAB

REST : Cas pratique

- Etape 3 : Implémentation de l'API
  - Mise en place du premier service de type « ressource »
    - Implémentation de la méthode GET (*Récupération d'objet(s)*)

```
@Override
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {

    String json = null;
    ObjectMapper mapper = new ObjectMapper();

    if (ResourceUriAnalyser.hasIdParameter(req)) {
        Long id = ResourceUriAnalyser.getIdParameter(req);
        json = mapper.writeValueAsString(findOne(id));
    } else {
        json = mapper.writeValueAsString(findAll());
    }

    resp.setContentType("application/json");
    resp.getWriter().write(json);
}
```

REST : Cas pratique

- Etape 3 : Implémentation de l'API
  - Mise en place du premier service de type « ressource »
    - Implémentation de la méthode POST (*Création d'un objet*)

```java
@Override
protected void doPost(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {

    String json = null;
    ObjectMapper mapper = new ObjectMapper();
    String body = req.getReader().lines().collect(Collectors.joining(System.lineSeparator()));
    Product product = mapper.readValue(body, Product.class);
    if(product.getId() != null) {
        throw new ServletException("id must be null !");
    }
    save(product);
    json = mapper.writeValueAsString(product);
    resp.setStatus(201);
    resp.setContentType("application/json");
    resp.getWriter().write(json);
}
```

REST : Cas pratique

- Etape 3 : Implémentation de l'API
  - Mise en place du premier service de type « ressource »
    - Implémentation de la méthode PUT (*modification d'un objet*)

```
@Override
protected void doPut(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {

    String json = null;
    ObjectMapper mapper = new ObjectMapper();
    String body = req.getReader().lines().collect(Collectors.joining(System.lineSeparator()));
    Product product = mapper.readValue(body, Product.class);
    if(product.getId() == null) {
        throw new ServletException("id is required !");
    }
    save(product);
    json = mapper.writeValueAsString(product);
    resp.setStatus(200);
    resp.setContentType("application/json");
    resp.getWriter().write(json);
}
```

REST : Cas pratique

- Etape 3 : Implémentation de l'API
  - Mise en place du premier service de type « ressource »
    - Implémentation de la méthode DELETE (*suppression d'un objet*)

```
@Override
protected void doDelete(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {

    if (ResourceUriAnalyser.hasIdParameter(req)) {
        Long id = ResourceUriAnalyser.getIdParameter(req);
        Product product = findOne(id);
        if (product == null) {
            throw new ServletException("Product not found for id \'"+id+"\' !");
        }
        delete(product);
        resp.setStatus(200);
    } else {
        throw new ServletException("id is required !");
    }
}
```

REST : Cas pratique

- Etape 3 : Implémentation de l'API
  - Mise en place du premier service de type « **ressource** »
    - Gestion des relations dans la méthode **GET** : Service de ressource « **Order** »

```
@Override
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {

    String json = null;
    ObjectMapper mapper = new ObjectMapper();
    if (ResourceUriAnalyser.hasIdParameter(req)) {
        Long id = ResourceUriAnalyser.getIdParameter(req);
        if(ResourceUriAnalyser.hasFirstRelationParameter(req)) {
            String relation = ResourceUriAnalyser.getFirstRelationParameter(req);
            if("productSelections".equalsIgnoreCase(relation)) {
                json = mapper.writeValueAsString(findOne(id).getProductSelections());
            }
        } else {
            json = mapper.writeValueAsString(findOne(id));
        }
    } else {
        json = mapper.writeValueAsString(findAll());
    }
    resp.setContentType("application/json");
    resp.getWriter().write(json);
}
```

REST : Cas pratique

- Etape 3 : Implémentation de l'API
  - Implémentation de la sécurisation
    - Ajout d'un filtre pour **vérifier** l'existence du **token** dans les headers de la requête et le valider

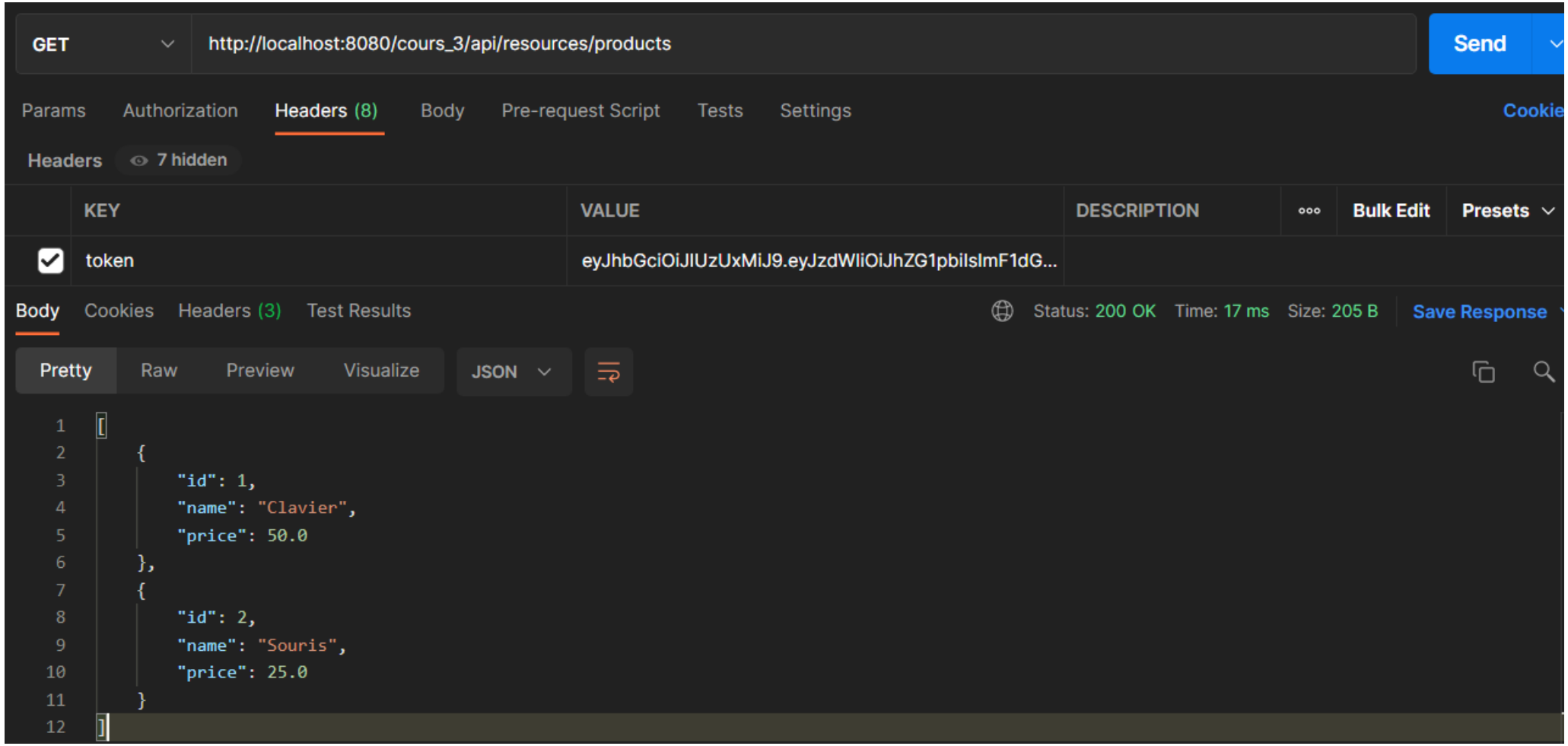
```
HttpServletRequest httpRequest = (HttpServletRequest) request;
String token = httpRequest.getHeader(TOKEN_VARIABLE_NAME);
if (token != null) {
    if (!validateToken(token)) {
        throw new ServletException("unauthorized access");
    }
} else {
    throw new ServletException("unauthorized access");
}
```

  - Modélisation du service d'authentification par la mise en place d'une servlet

Le code présenté est à titre d'exemple

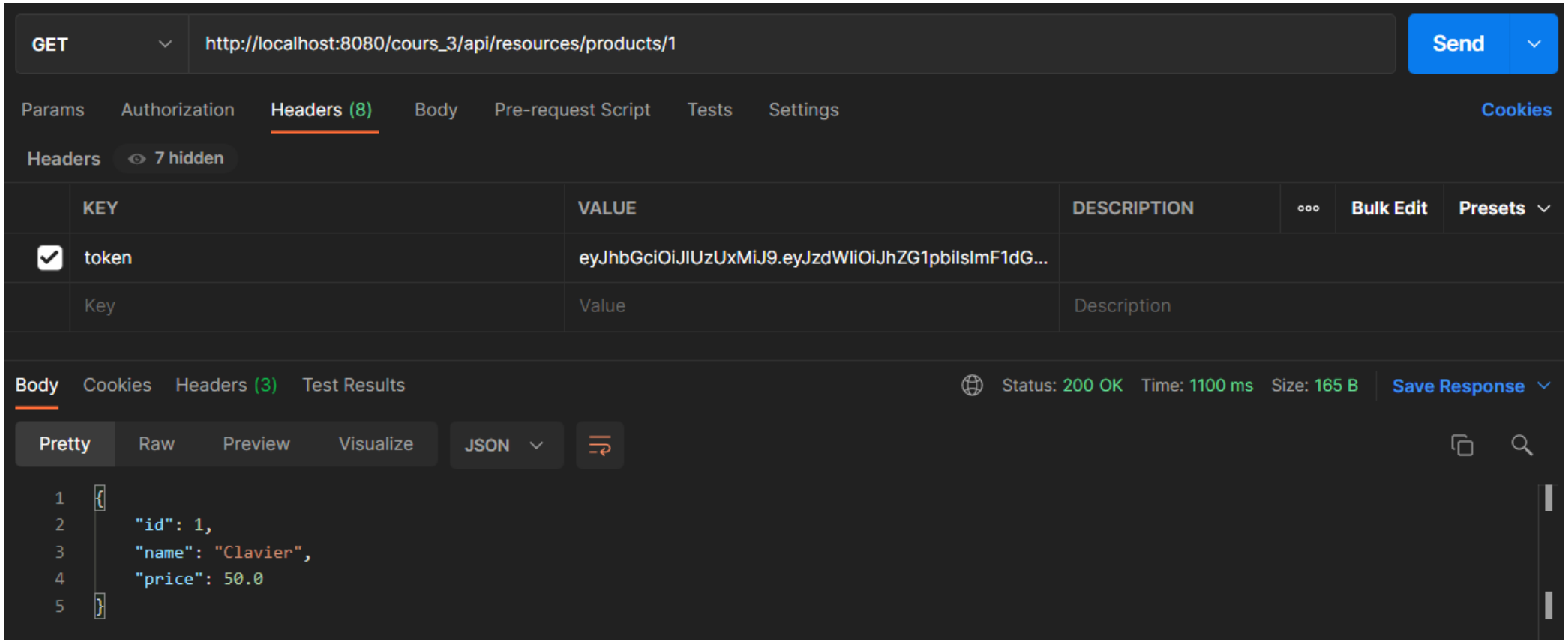
REST : Cas pratique

- Etape 4 : Utilisation de l'API
  - Avec l'outil « Postman » :
    - Je souhaite consulter les produits



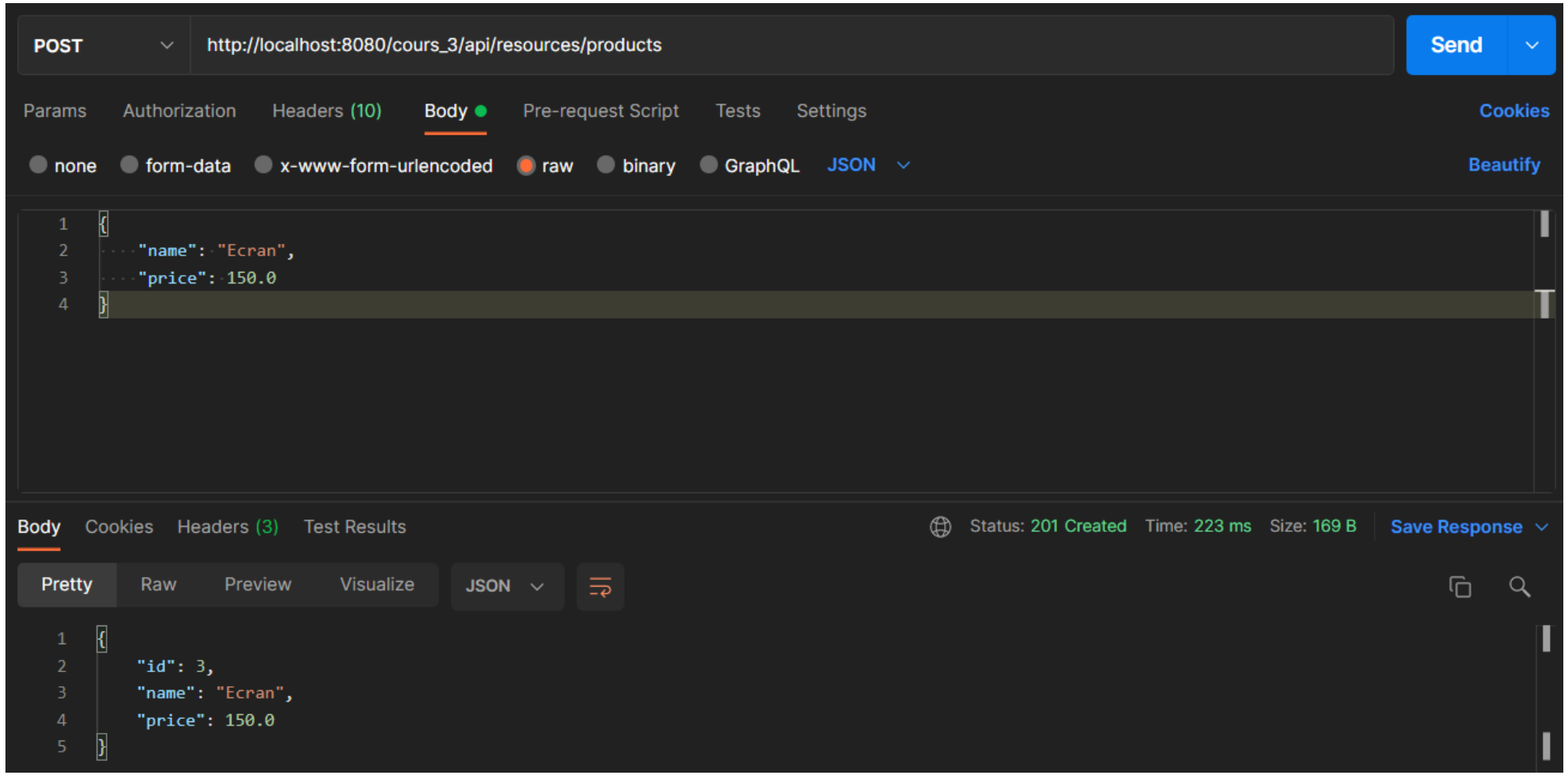
REST : Cas pratique

- Etape 4 : Utilisation de l'API
  - Avec l'outil « Postman » :
    - je souhaite consulter le produit d'id « 1 »



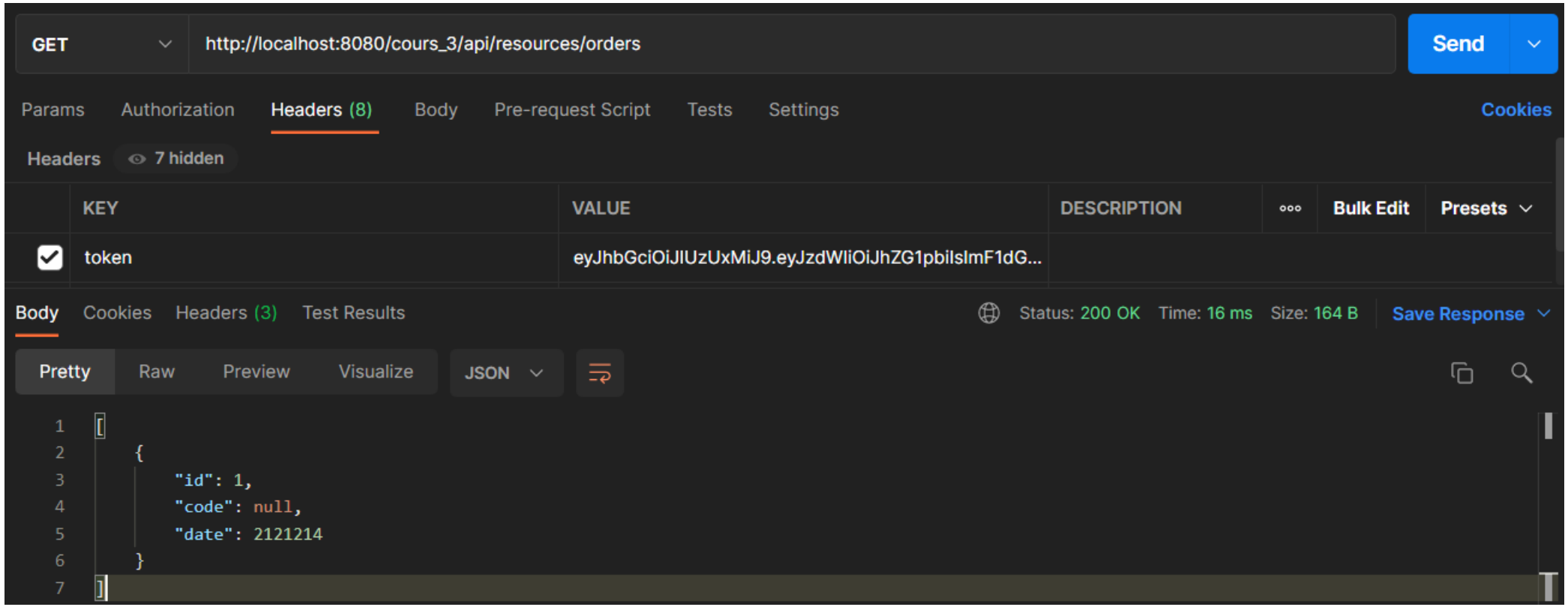
REST : Cas pratique

- Etape 4 : Utilisation de l'API
  - Avec l'outil « Postman » :
    - je souhaite ajouter un nouveau produit



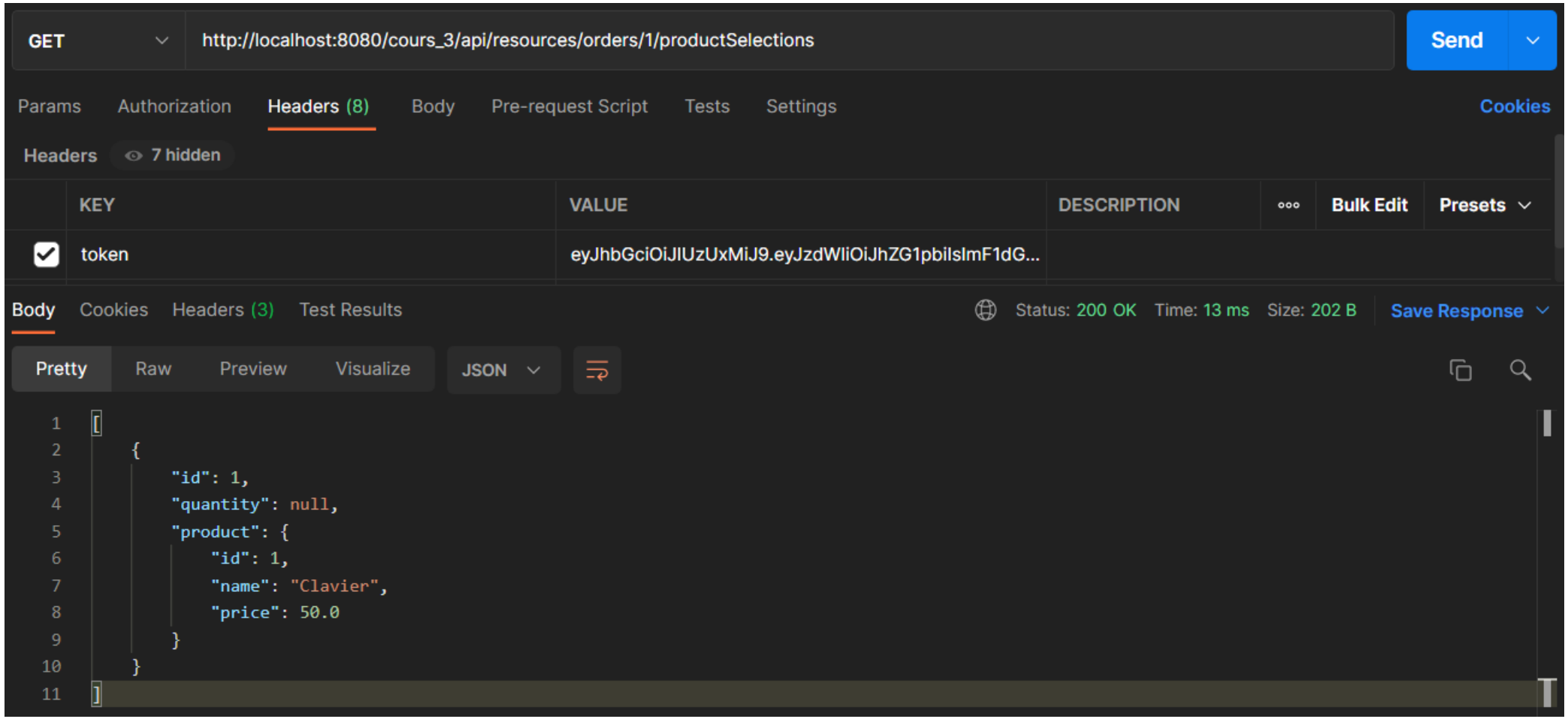
REST : Cas pratique

- Etape 4 : Utilisation de l'API
  - Avec l'outil « Postman » :
    - je souhaite consulter les commandes



REST : Cas pratique

- Etape 4 : Utilisation de l'API
  - Avec l'outil « Postman » :
    - je souhaite consulter les sélections de produits de la commande d'id « 1 »



Libery
LAB

TP3

Cours 3 – Services web

Libery Lab

Apprendre et partager - La passion du développement