SPRING MVC

UP ASI
Bureaux E204 | E304

Plan du Cours

- Spring MVC (Définition + Spring web)
- Les architectures physiques et logiques
- Serveur web vs. Serveur d'application
- Postman
- Dépendance web
- Cycle de Vie d'une requête HTTP (Spring Boot + Postman)
- RestController

– TP Spring Boot + Spring Data JPA + Spring MVC (REST) + Postman

Introduction

- Un **Conteneur de Servlets** (Servlet container en anglais) ou **Conteneur Web** (web container en anglais) est un logiciel qui exécute des servlets.

- Un ou une **Servlet** est une classe Java qui permet de créer dynamiquement des données au sein d'un serveur HTTP.

- Il existe plusieurs conteneurs de servlets, dont **Apache Tomcat** ou encore Jetty. Le serveur d'application JBoss Application Server(Wildfly) utilise Apache Tomcat.

- Nous allons nous intéresser au développent de la couche **Web** (Web Services REST + Contrôleur + Service + Repository) dans ce cours.
- Nous allons aussi pratiquer la consommation des services par Postman.

Introduction

- Plusieurs Projets Spring permettent d'implémenter des applications Web :

- Framework Spring (qui contient Spring MVC)
- Spring Web Flow (Implémenter les navigations Stateful).
- Spring mobile (Détecter le type de l'appareil connecté).
- Spring Social (Facebook, Twitter, LinkedIn).
- …

- Nous allons nous intéresser à **Spring MVC**.

SPRING MVC

- **Spring MVC** est un Framework Web basé sur le design pattern **MVC** (Model / View / Controller).
- Spring MVC fait partie du projet "Spring Framework".
- Spring MVC s'intègre avec les différentes technologies de vue tel que JSF, JSP, Velocity, Thymeleaf...
- Spring MVC n'offre pas une technologie de vue mais permet en revanche de communiquer avec toutes les technologies web les plus performantes tels que Angular, React, etc...
- Spring MVC est construit en se basant sur la spécification JavaEE : **Java Servlet**.

Architecture Physique

- **Tier** est un mot anglais qui signifie étage ou niveau.
- Une application peut être **1-Tier**, **2-Tiers**, **3-Tiers** ou **N-Tiers**.

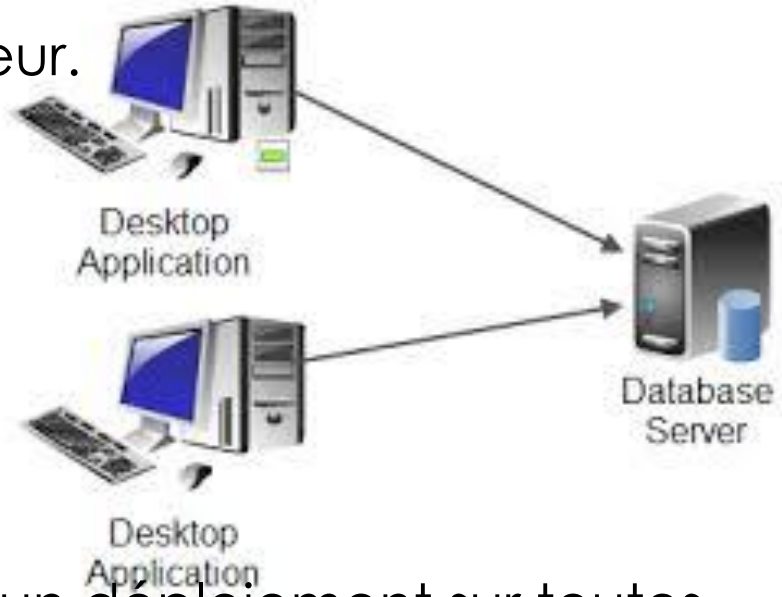
Architecture Physique - 1-Tiers

- Une application **1-Tier** est, par exemple, la Modification d'un document Word sur un ordinateur Local.
- Tout est sur la même machine et les couches sont fortement liées.
- Inconvénients : Risque de perte des données (non sauvegardées à distance), Impossible d'accéder à une même ressource par deux utilisateurs en même temps.

Architecture Physique - 2-Tiers

- Une application **2-Tiers** est typiquement une application **client lourd**.

- Le niveau **Présentation (IHM)** et le niveau **Traitement** sont sur la machine de l'utilisateur.
- Le niveau **Base de Données** est sur un autre serveur.



- C'est une architecture **Client / Serveur**.
- Client = demandeur de ressource
- Serveur = fournisseur de ressource

Inconvénients

- Toute mise à jour des fonctionnalités nécessitent un déploiement sur toutes les machines des utilisateurs.
- Le serveur ne fait pas appel à une autre application pour fournir le service.

Architecture Physique - 3-Tiers

- Une application **3-Tiers** introduit un niveau intermédiaire (middleware) entre le client et le serveur.
- Le niveau intermédiaire est chargé de fournir la ressource en faisant appel à un autre serveur.

Avantages

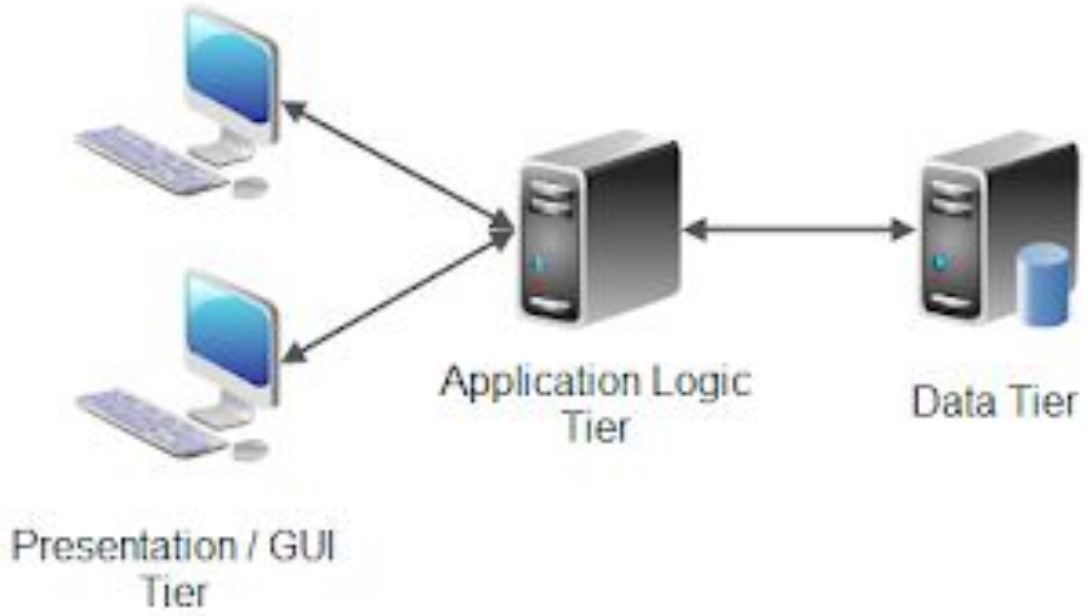
- Centraliser la logique application sur un serveur HTTP

Inconvénients

- Le serveur HTTP (élément principale de l'architecture)est fortement sollicité d'où une charge de demandes provenant à la fois du client et du serveur.
- Bien que cette architecture résout le problème du client lourd de l'architecture deux tiers, le soulagement du client est remplacé par un serveur fortement sollicité.

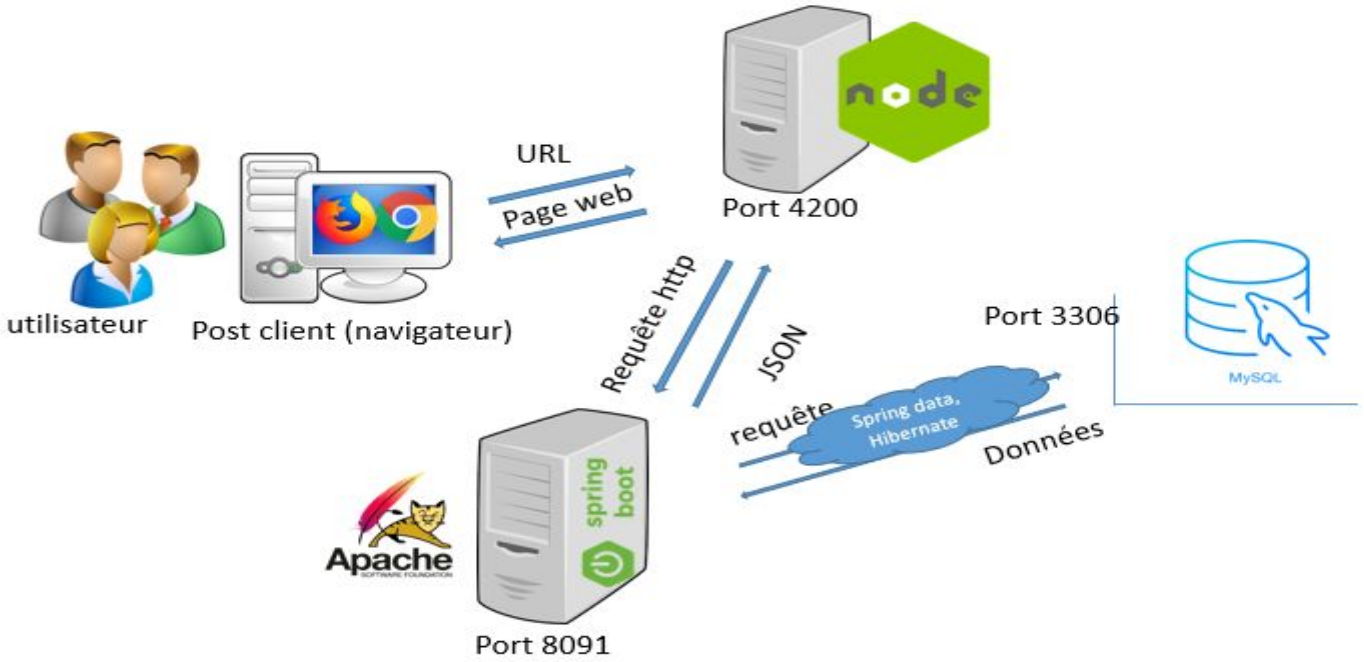
Architecture Physique - 3-Tiers

- Une application **3-Tiers** est typiquement une application Web :
  - Niveau **Présentation** : IHM (Navigateur sur la machine de l'utilisateur)
  - Niveau **Traitement** : Un serveur web (Tomcat, …) qui contient le WAR de notre application.
  - Niveau **Base de données** : Un serveur de BD qui stocke les données de notre application.

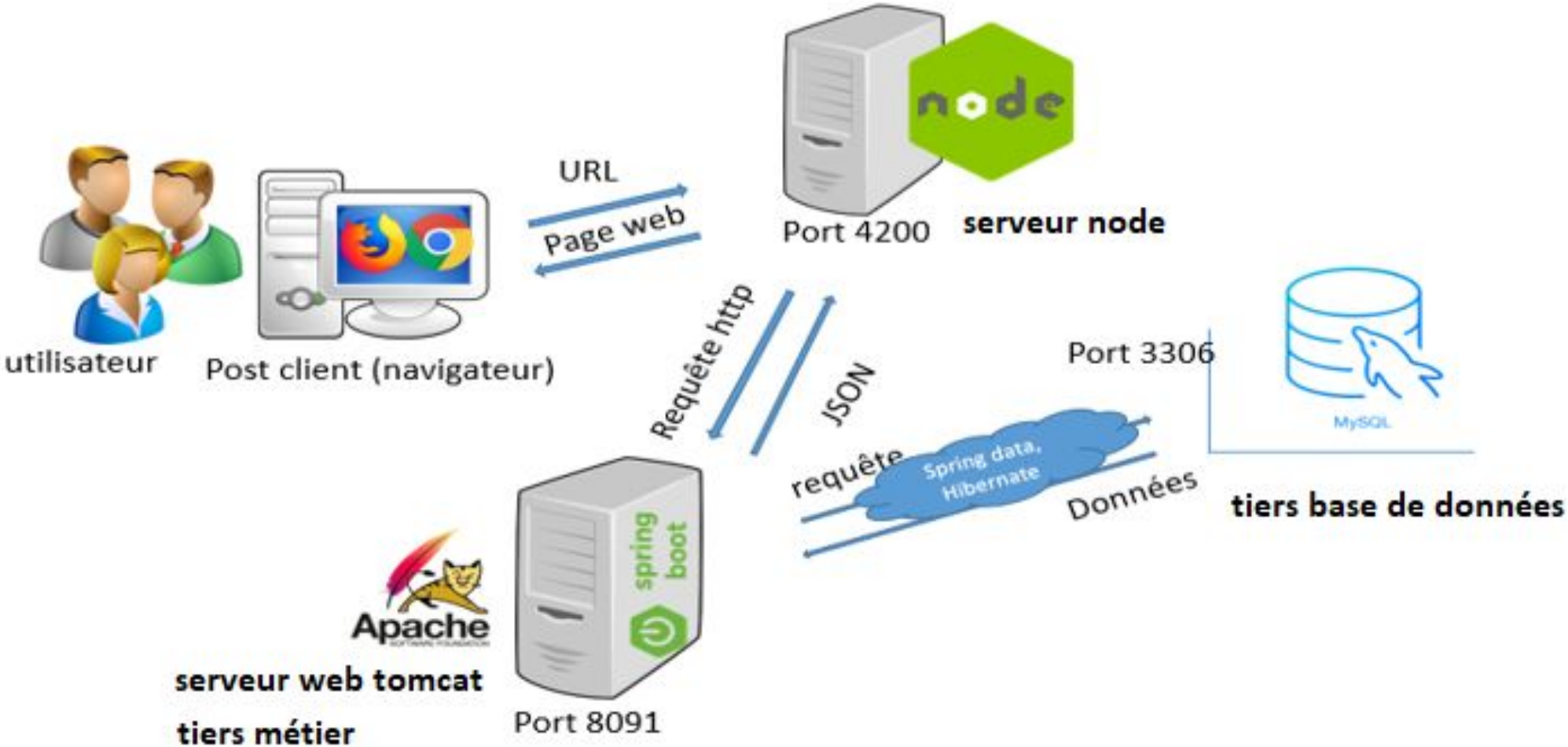


Architecture Physique - N-Tiers

- L'architecture N tiers assure un équilibre de charge entre le client et le serveur par l'introduction de nouvelles couches.
- Voici une architecture 4-Tiers d'une application web développée par un étudiant Esprit pendant son projet de fin d'étude (GUI – Angular sur le Serveur NodeJS – Spring Boot (Serveur Web Tomcat embarqué) – Serveur de base de données MySQL) :

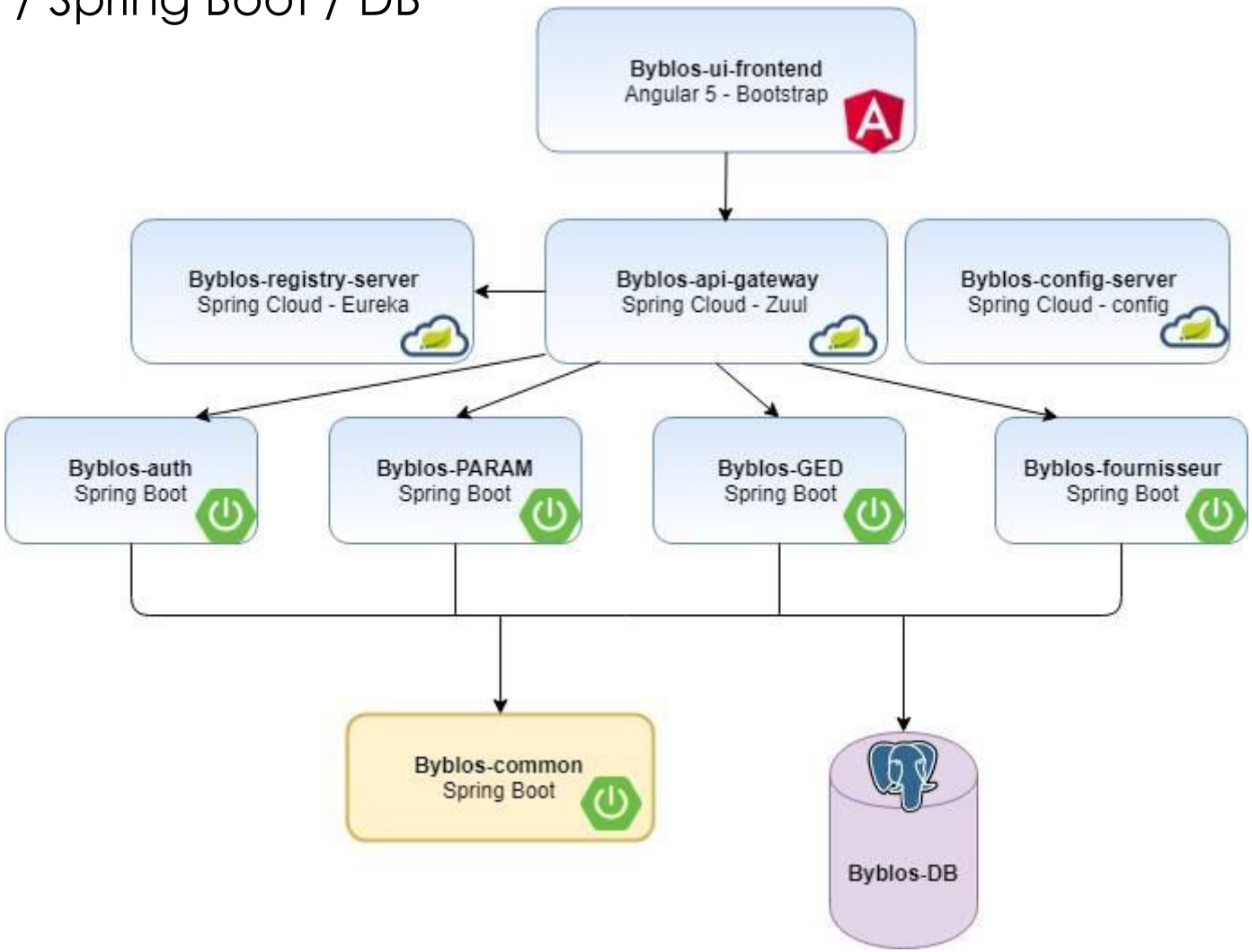


Architecture Physique - N-Tiers



Architecture Physique - N-Tiers

- Voici une architecture n-Tiers, **en Micro-Servcies**, d'une application web développée par un étudiant Esprit pendant son projet de fin d'étude : GUI / NodeJS / Spring Boot / DB



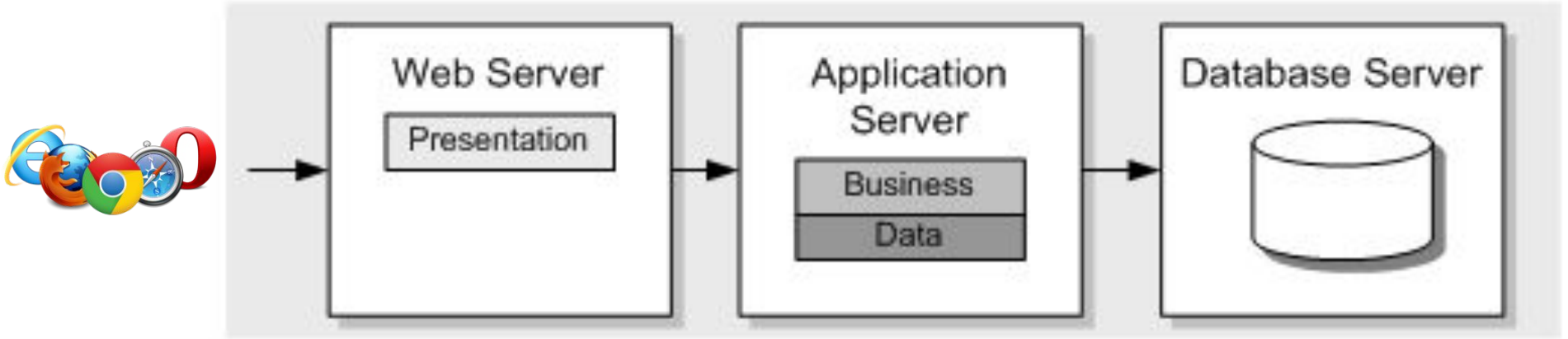
Architecture logique

- Une application typique utilisant Spring est généralement structurée en trois couches :
  - Couche **Présentation** : (Web + Contrôleur)
  - Couche **Service** : interface métier avec mise en œuvre de certaines fonctionnalités.
  - Couche **Accès aux Données** : recherche et persistance des objets.

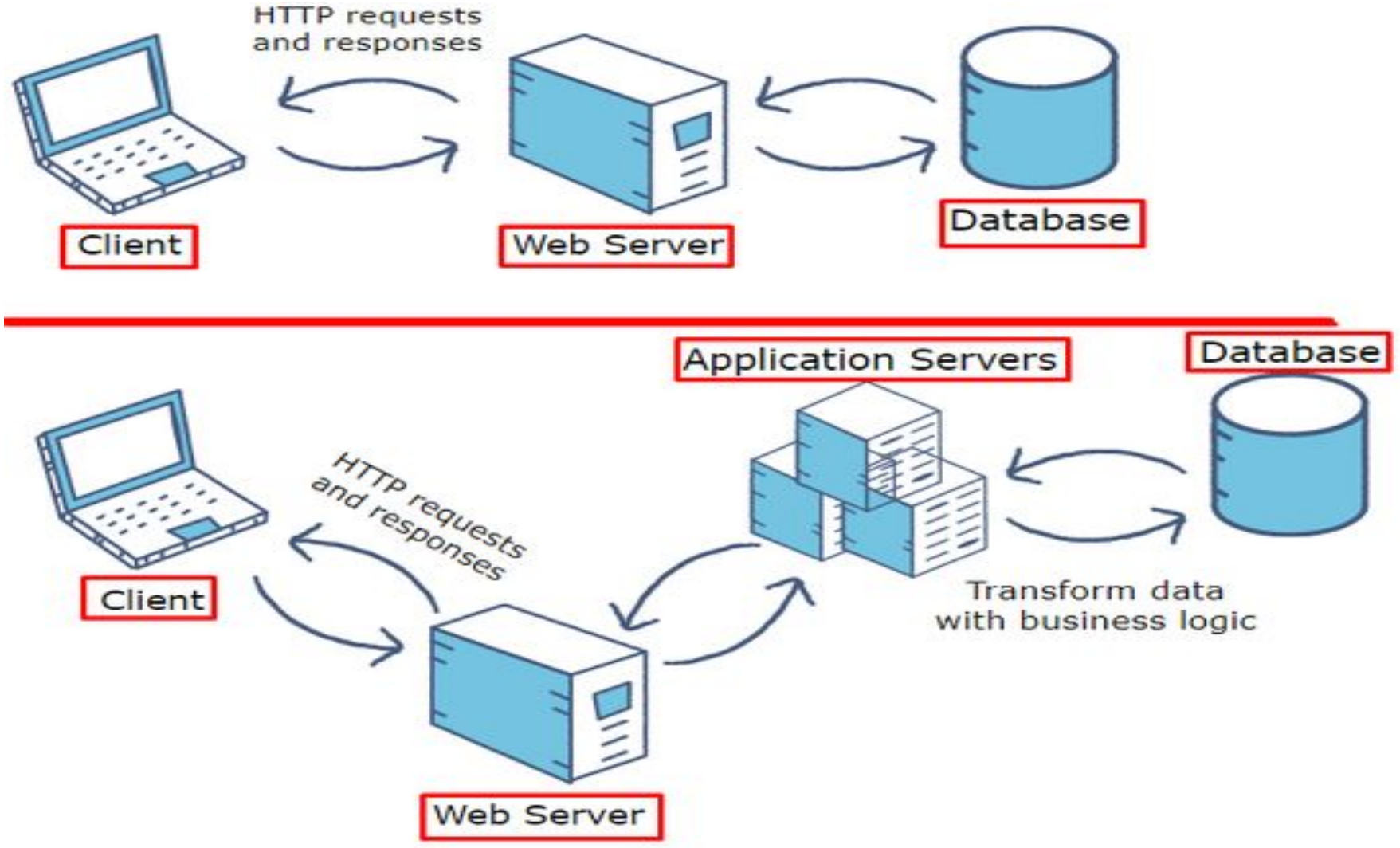
- **Spring est un Framework utilisé pour créer et injecter les objets requis pour communiquer entre les différentes couches.**

Serveur Web vs Serveur d'Application

Serveur Web	Serveur d'application JavaEE Serveur web + container
Héberge que la couche présentation et l'expose qu'à travers le protocole HTTP(S)	Héberge la logique métier et peut aussi héberger la couche présentation (supporte différents protocoles : HTTP, JNDI, ...).
Ne peut pas inclure un EJB Container.	Doit inclure un EJB Container.
lightweight	Relativement gourmand en ressources (CPU, RAM et Disk).
Exp : Apache HTTP Server, Tomcat, Jetty	Exp : Wildfly, WebSphere ...

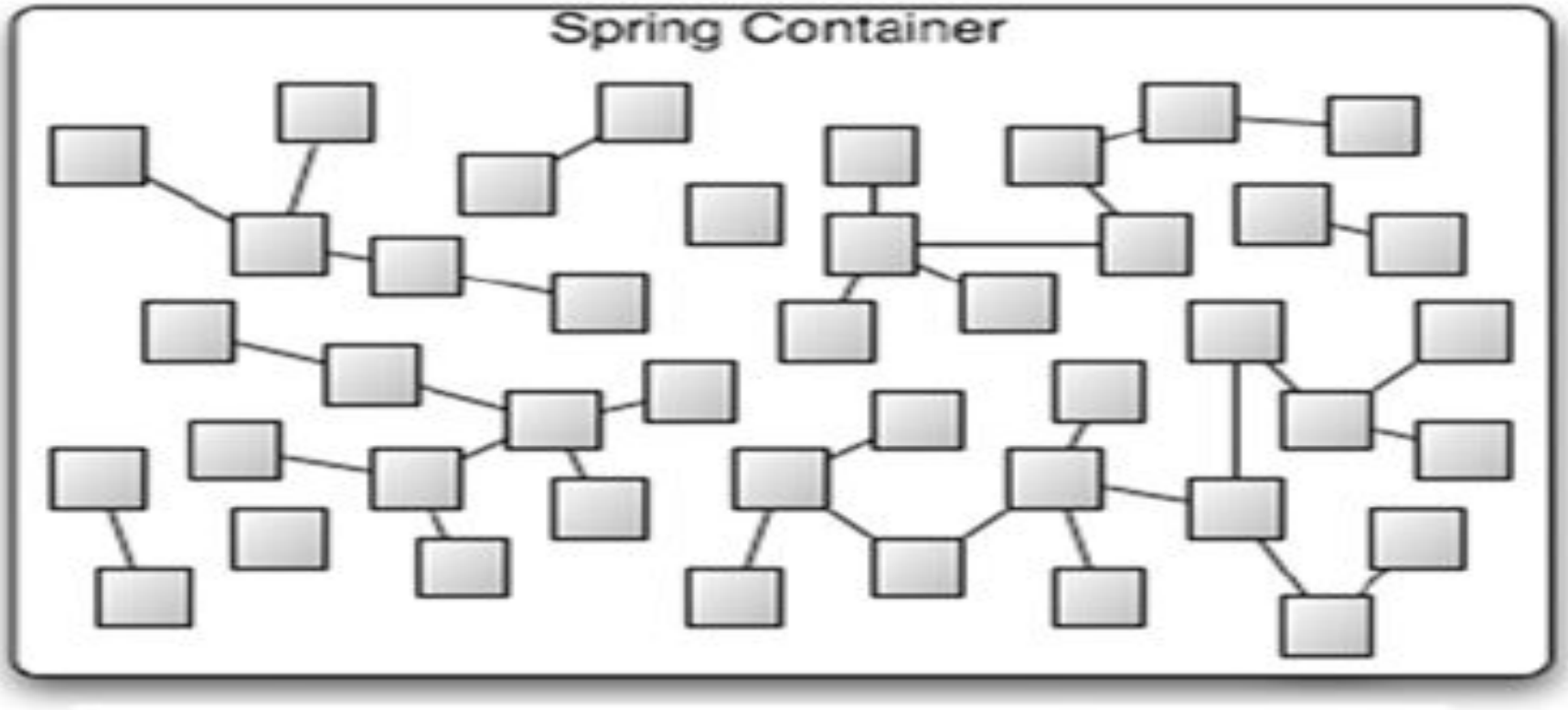


Serveur Web vs Serveur d'Application



Serveur Web vs Serveur d'Application

Spring IOC Container



Dans une application Spring, les objets sont crées, sont liés ensembles et communiquent dans le Spring IOC Container.

Postman

- Parmi les nombreuses solutions pour interroger ou tester les web services et les API, Postman propose de nombreuses fonctionnalités, une prise en main rapide et une interface graphique agréable.

- Postman permet de construire et d'exécuter des requêtes HTTP, de les stocker dans un historique afin de pouvoir les rejouer.

POST http://localhost:8082/tpFoyer17/api/etudiants/addEtudiant Send

Params Auth Headers (8) Body Pre-req. Tests Settings Cook

raw JSON Beauti

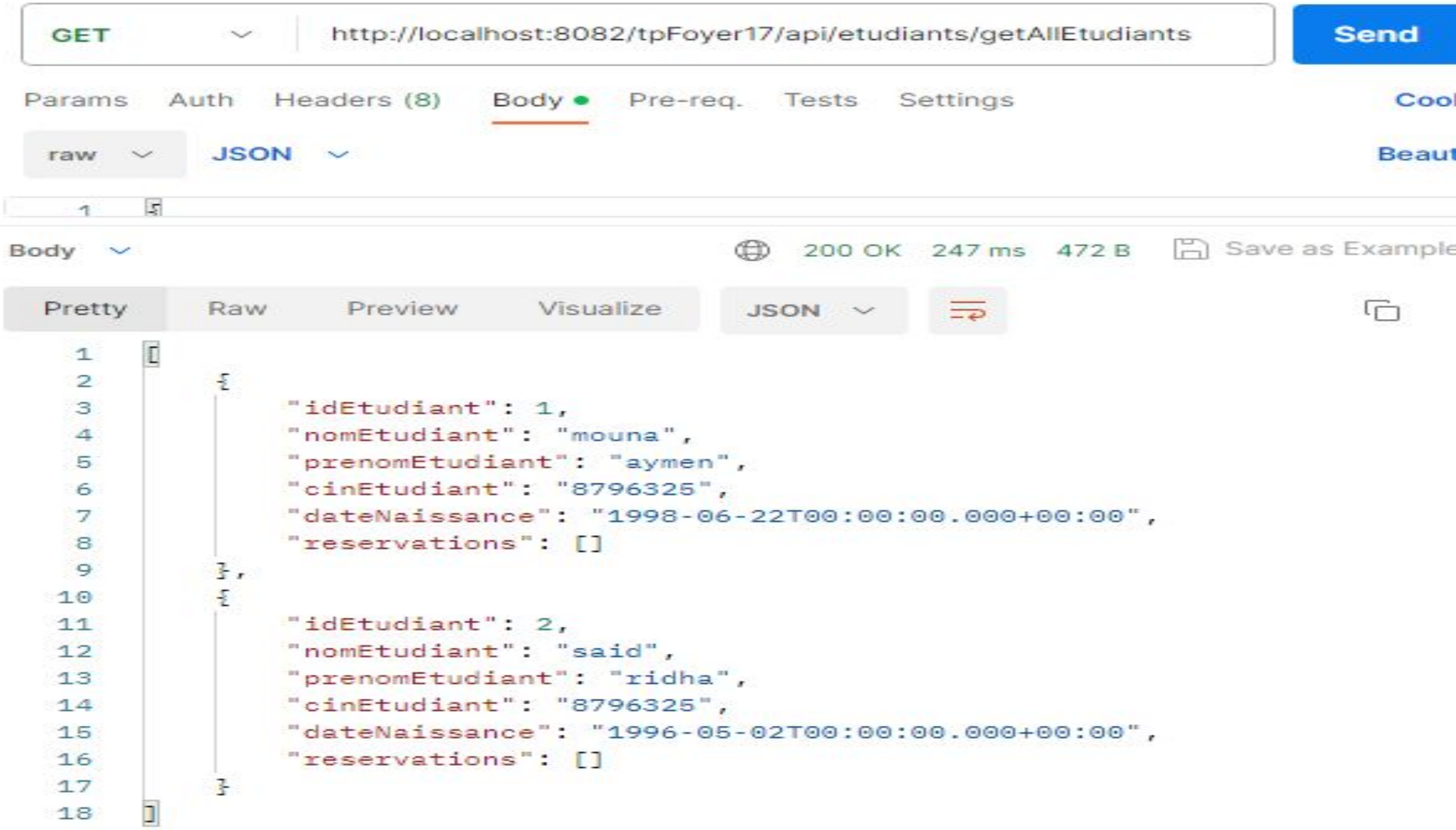
```
{
"nomEtudiant" : "mouna",
"prenomEtudiant" : "aymen",
"cinEtudiant" : 8796325,
"dateNaissance" :"1998-06-22"

}
```

Body 200 OK 567 ms 319 B Save as Example

Pretty Raw Preview Visualize JSON

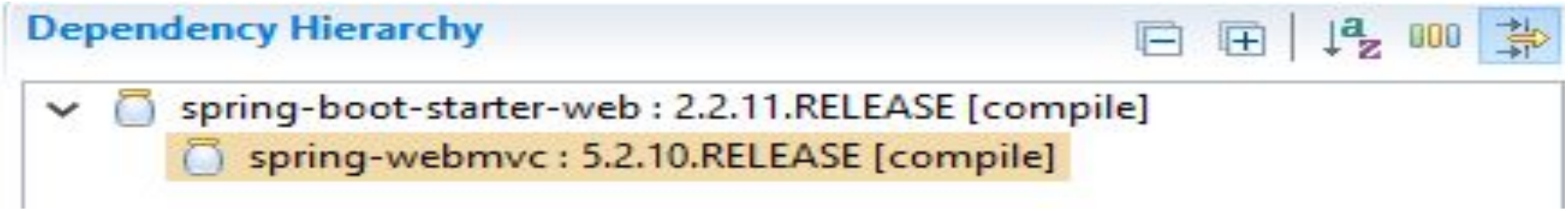
```
{
    "idEtudiant": 1,
    "nomEtudiant": "mouna",
    "prenomEtudiant": "aymen",
    "cinEtudiant": "8796325",
    "dateNaissance": "1998-06-22T00:00:00.000+00:00",
    "reservations": null
}
```

Postman


GET
http://localhost:8082/tpFoyer17/api/etudiants/getAllEtudiants
Send
Params
Auth
Headers (8)
Body
Pre-req.
Tests
Settings
raw
JSON
Body
200 OK
247 ms
472 B
Pretty
Raw
Preview
Visualize
JSON
[
{
"idEtudiant": 1,
"nomEtudiant": "mouna",
"prenomEtudiant": "aymen",
"cinEtudiant": "8796325",
"dateNaissance": "1998-06-22T00:00:00.000+00:00",
"reservations": []
},
{
"idEtudiant": 2,
"nomEtudiant": "said",
"prenomEtudiant": "ridha",
"cinEtudiant": "8796325",
"dateNaissance": "1996-05-02T00:00:00.000+00:00",
"reservations": []
}
]

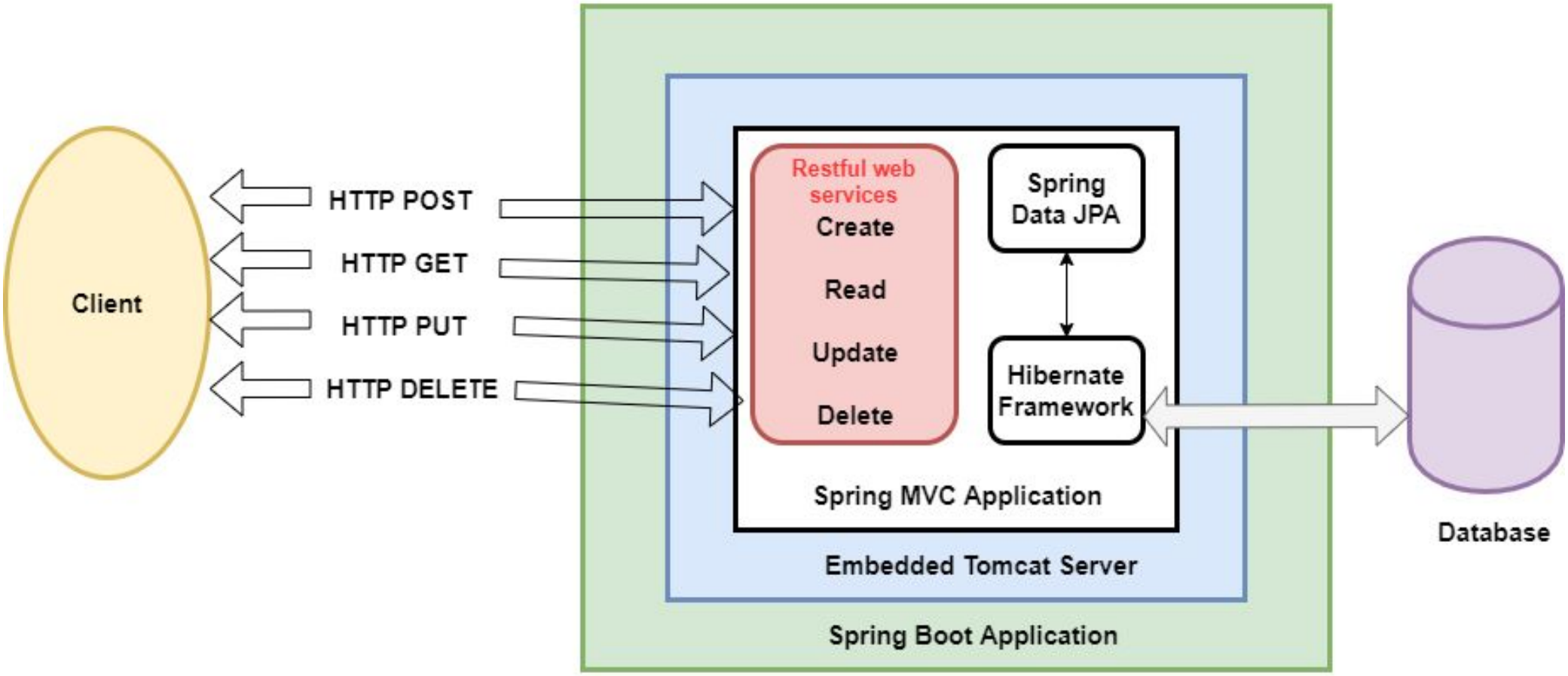
Dépendance web

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
```



- Le starter web permet d'ajouter toutes les dépendances liées à la partie web notamment ceux liées à Spring MVC et l'exposition des web services.

Cycle de Vie d'une requête HTTP (Spring Boot + Postman)



RestController

- Dans ce fichier de properties ajouter les lignes suivantes, pour définir l'url de notre application :

```
#Server configuration
server.port=8082
server.servlet.context-path=/tpFoyer17
```

- Cela permet de créer une partie de l'url que nous allons utiliser sur postman : **http://localhost:8082/tpFoyer17/**

- Le path complet sera crée au niveau de notre couche controller comme présentée dans le slide suivant :

RestController

```java
@RestController
@AllArgsConstructor
@Slf4j
@FieldDefaults(level = AccessLevel.PRIVATE)
@RequestMapping("api/etudiants")
public class EtudiantController {
    IEtudiantService etudiantService;
@GetMapping("/getAllEtudiants")
    public Iterable<Etudiant> getAllEtudiants(){
        return etudiantService.getAllEtudiants();
    }
```

TP - Spring : Boot – Core – Data JPA – MVC (REST)

- Nous allons commencer par exposer des Web Service REST :
  Spring Boot – Core – Data JPA – MVC (**REST**) -Postman

- Vous avez déjà créé un projet : Spring (Boot – Core – Data JPA) avec un CRUD.

- Nous allons reprendre le même projet (étude de cas **gestion_foyer**) et exposer ces méthodes (CRUD) avec des Web Servie REST.

- Ces Web Services seront testé avec **Postman**.

TP - Spring : Boot – Core – Data JPA – MVC (REST)

- Installation de Postman :

- L'exécutable est sur le **Drive** du cours Spring (dossier **Outils**), à télécharger et à installer.

TP - Spring : Boot – Core – Data JPA – MVC (REST)

- Vérifier que le fichier de properties contient les propriétés nécessaires (web, base de données, log4j, …) :

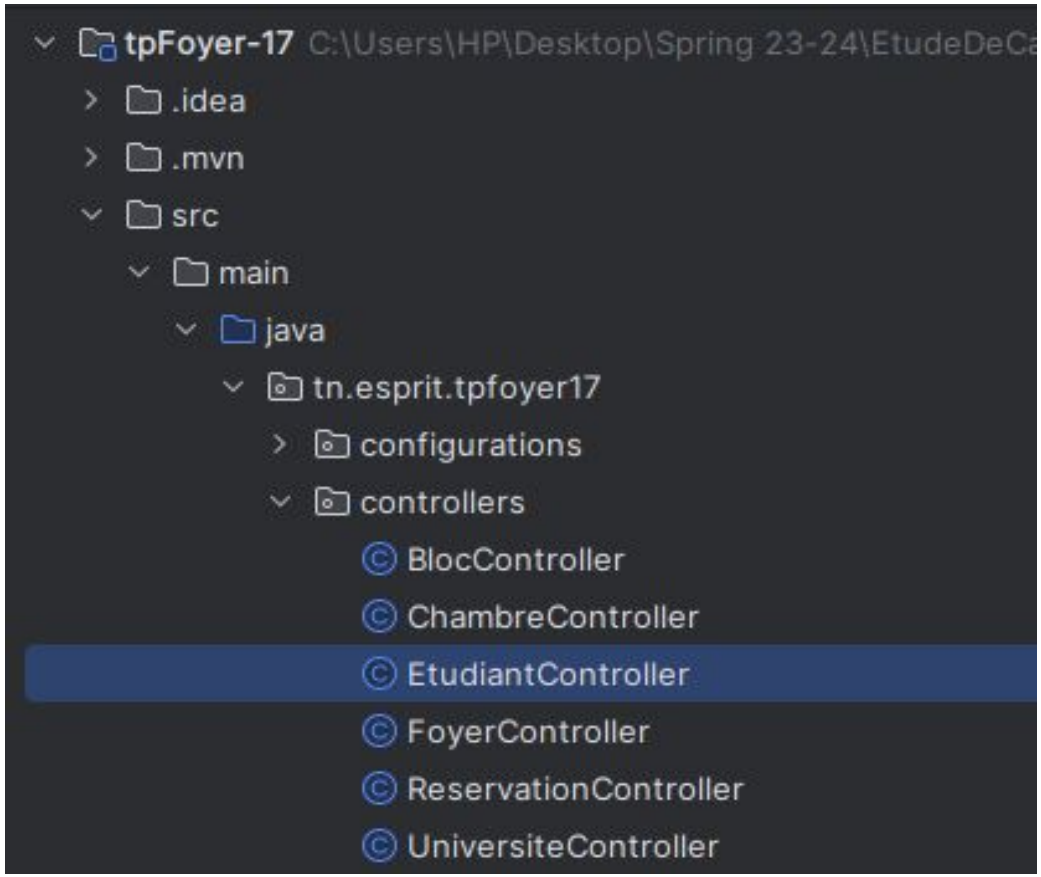
```
#Server configuration
server.servlet.context-path=/tpFoyer17
server.port=8082

### DATABASE ###

spring.datasource.url=jdbc:mysql://localhost:3306/springdb?useUnicode=true
&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=
### JPA / HIBERNATE ###
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
```

TP - Spring : Boot – Core – Data JPA – MVC (REST)

```
#logging configuration
# Spécifier le fichier externe ou les messages sont stockés
 logging.file=D:/spring_log_file.log
# Spécifier la taille maximale du fichier de journalisation
 logging.file.max-size= 100KB
# spécifier le niveau de Log
 logging.level.root=INFO
# Spécifier la forme du message
 logging.pattern.console=%d{yyyy-MM-dd HH:mm:ss} - %-5level - %logger{36} - %msg%n
```

TP - Spring : Boot – Core – Data JPA – MVC (REST)

- Créer le package **tn.esprit.spring.control**
- Créer le bean Spring **EtudiantRestController** annoté **@RestController**
- Créer les méthodes nécessaires pour exposer le CRUD (voir pages suivantes) :



TP - Spring : Boot – Core – Data JPA – MVC (REST)

```java
@RestController
@AllArgsConstructor
@Slf4j
@FieldDefaults(level = AccessLevel.PRIVATE)
@RequestMapping("api/etudiants")
public class EtudiantController {
    IEtudiantService etudiantService;
@PostMapping("/addEtudiant")
    public Etudiant addEtudiant(@RequestBody Etudiant e){
        return  etudiantService.addEtudiant(e);
    }
```

TP - Spring : Boot – Core – Data JPA – MVC (REST)

```
@GetMapping("/getAllEtudiants")
    public Iterable<Etudiant> getAllEtudiants(){
        return etudiantService.getAllEtudiants();
    }
@GetMapping("/getEtudiantById/{id}")
    public Etudiant getEtudById(@PathVariable long id){
        return etudiantService.getEtudById(id);
    }
```

TP - Spring : Boot – Core – Data JPA – MVC (REST)

```
@DeleteMapping("/deleteEtudiant/{id}")
    public void deleteEtudiant(@PathVariable long id){
        etudiantService.deleteEtud(id);
    }
}
```

```
@PutMapping("/updateEtudiant")
public Etudiant updateEtudiant(@RequestBody Etudiant e){
    return  etudiantService.updateEtudiant(e);
}
```

POST http://localhost:8082/tpFoyer17/api/etudiants/addEtudiant Send

Params Auth Headers (8) Body Pre-req. Tests Settings Cook

raw JSON Beauti

```
{
"nomEtudiant" : "mouna",
"prenomEtudiant" : "aymen",
"cinEtudiant" : 8796325,
"dateNaissance" :"1998-06-22"

}
```

Body 200 OK 567 ms 319 B Save as Example

Pretty Raw Preview Visualize JSON

```
{
    "idEtudiant": 1,
    "nomEtudiant": "mouna",
    "prenomEtudiant": "aymen",
    "cinEtudiant": "8796325",
    "dateNaissance": "1998-06-22T00:00:00.000+00:00",
    "reservations": null
}
```

PUT http://localhost:8082/tpFoyer17/api/etudiants/updateEtudiant Send

Params Auth Headers (8) Body Pre-req. Tests Settings Cook

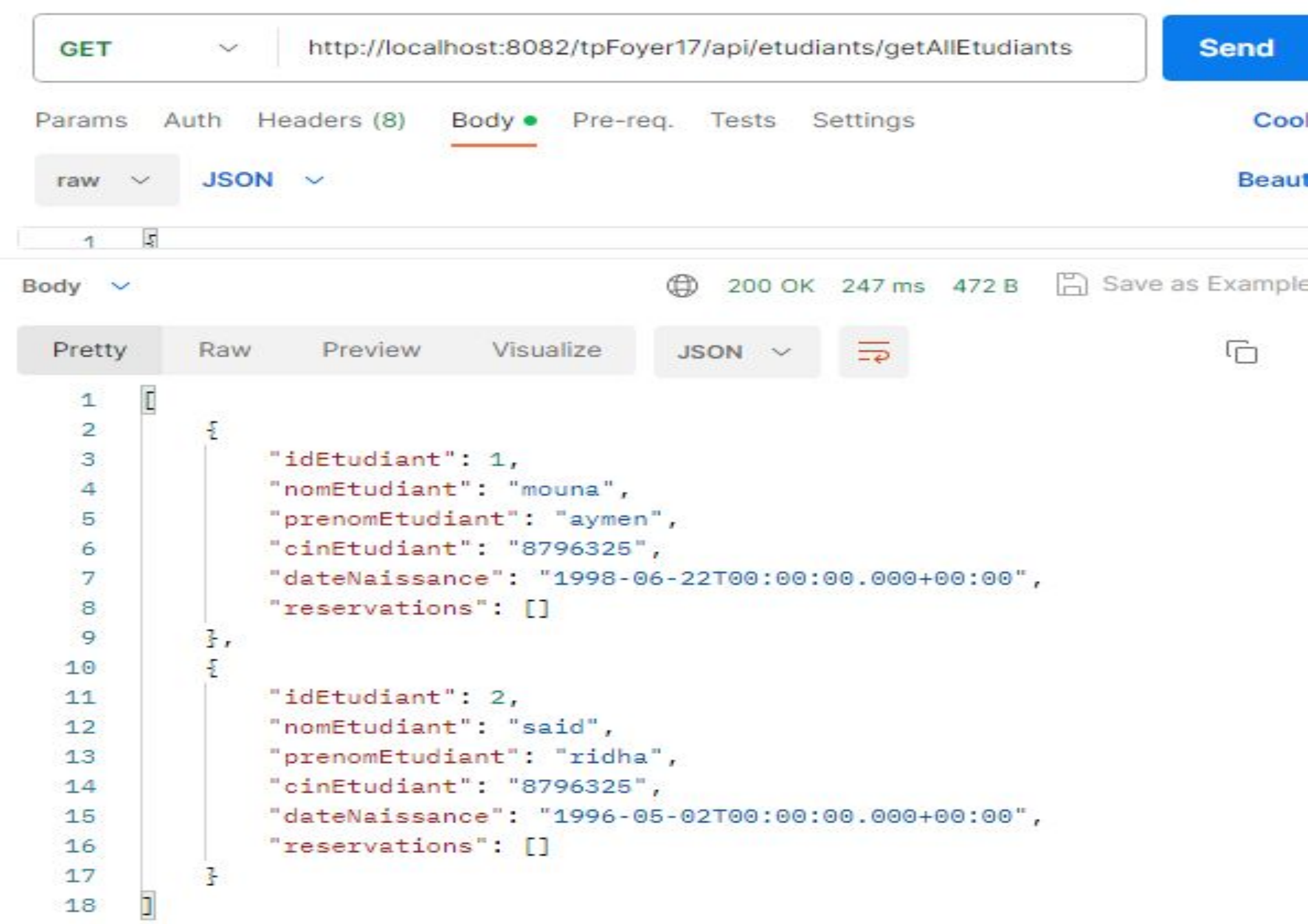
raw JSON Beaut

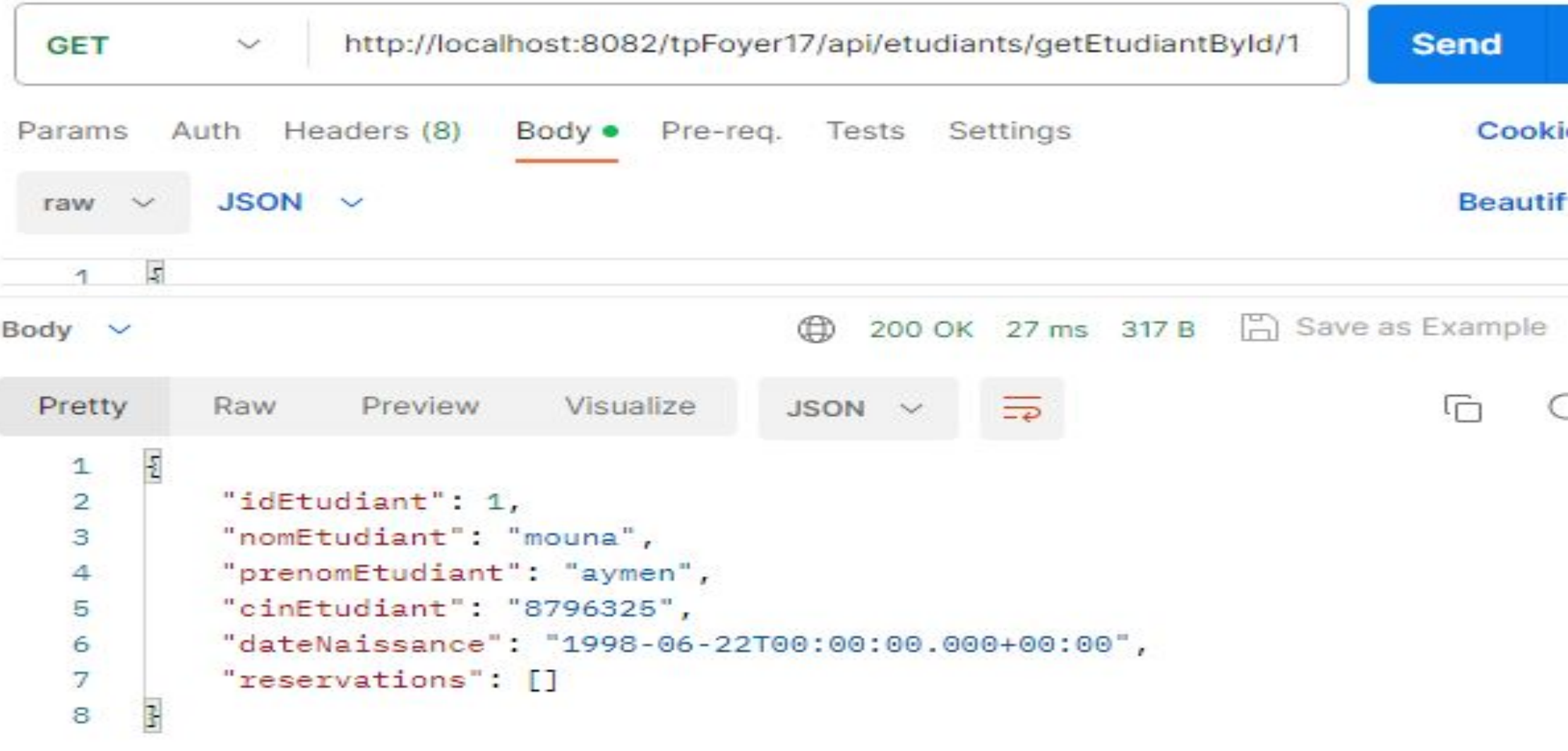
```
{
"idEtudiant": 2,
"nomEtudiant" : "said",
"prenomEtudiant" : "ridha",
"cinEtudiant" : 8796325,
"dateNaissance" :"1996-05-02"

}
```

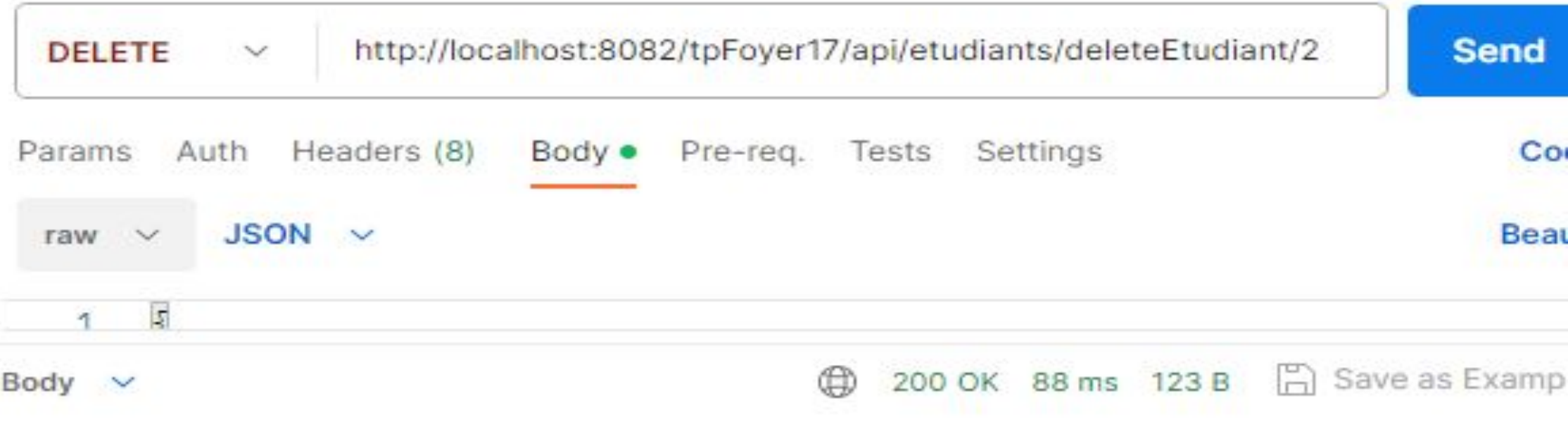
Body 200 OK 242 ms 318 B Save as Example

Pretty Raw Preview Visualize JSON

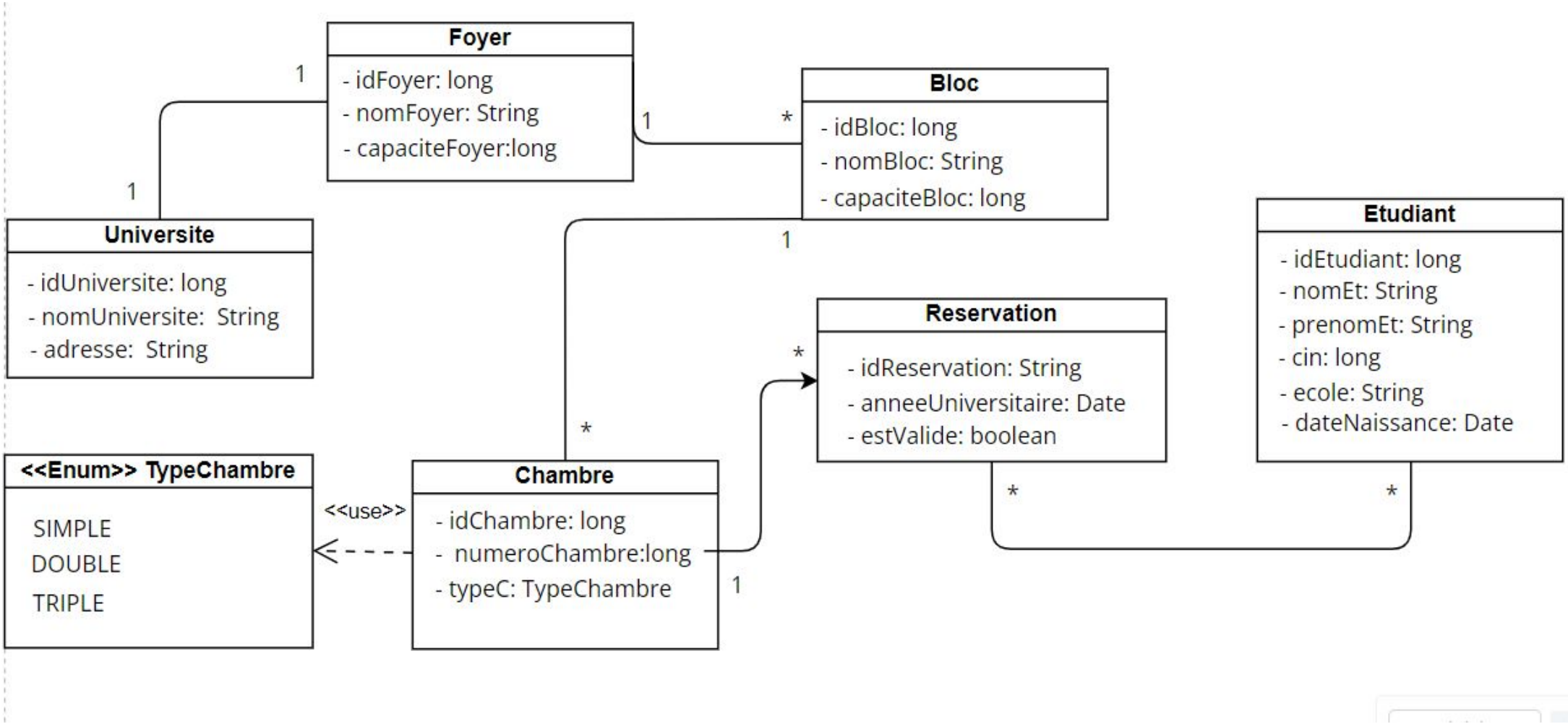
```
{
    "idEtudiant": 2,
    "nomEtudiant": "said",
    "prenomEtudiant": "ridha",
    "cinEtudiant": "8796325",
    "dateNaissance": "1996-05-02T00:00:00.000+00:00",
    "reservations": null
}
```


GET
http://localhost:8082/tpFoyer17/api/etudiants/getAllEtudiants
Send
Params
Auth
Headers (8)
Body
Pre-req.
Tests
Settings
raw
JSON
Body
200 OK
247 ms
472 B
Save as Example
Pretty
Raw
Preview
Visualize
JSON
[
{
"idEtudiant": 1,
"nomEtudiant": "mouna",
"prenomEtudiant": "aymen",
"cinEtudiant": "8796325",
"dateNaissance": "1998-06-22T00:00:00.000+00:00",
"reservations": []
},
{
"idEtudiant": 2,
"nomEtudiant": "said",
"prenomEtudiant": "ridha",
"cinEtudiant": "8796325",
"dateNaissance": "1996-05-02T00:00:00.000+00:00",
"reservations": []
}
]


GET
http://localhost:8082/tpFoyer17/api/etudiants/getEtudiantById/1
Send
Params
Auth
Headers (8)
Body
Pre-req.
Tests
Settings
raw
JSON
Body
200 OK
27 ms
317 B
Save as Example
Pretty
Raw
Preview
Visualize
JSON
{
"idEtudiant": 1,
"nomEtudiant": "mouna",
"prenomEtudiant": "aymen",
"cinEtudiant": "8796325",
"dateNaissance": "1998-06-22T00:00:00.000+00:00",
"reservations": []
}



	id_etudiant	cin_etudiant	date_naissance	nom_etudiant	prenom_etudiant
Éditer Copier Supprimer	1	8796325	1998-06-22 00:00:00.000000	mouna	aymen


Foyer
- idFoyer: long
- nomFoyer: String
- capaciteFoyer:long
Bloc
- idBloc: long
- nomBloc: String
- capaciteBloc: long
Universite
- idUniversite: long
- nomUniversite: String
- adresse: String
Etudiant
- idEtudiant: long
- nomEt: String
- prenomEt: String
- cin: long
- ecole: String
- dateNaissance: Date
Reservation
- idReservation: String
- anneeUniversitaire: Date
- estValide: boolean
<<Enum>> TypeChambre
SIMPLE
DOUBLE
TRIPLE
<<use>>
Chambre
- idChambre: long
- numeroChambre:long
- typeC: TypeChambre
1
*

Travail à faire

Spring MVC

Exposer les services implémentés dans l'étude de cas **Gestion Foyer** avec Postman pour les tester.

Si vous avez des questions, n'hésitez pas à nous contacter :

Département Informatique
UP ASI
Bureaux E204 | E304