SPRING MVC

esprit
Se former autrement

UP ASI
Bureau E204

Plan du Cours

- Spring MVC (Définition + Spring web)

- Spring MVC + Postman
  - Postman
  - Dépendance web
  - Cycle de Vie d'une requête HTTP (Spring Boot+Postman)
  - RestController
- TP Spring Boot + Spring Data JPA + Spring MVC (REST) + Postman
- Swagger

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.

Spring WEB

- 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 nos 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**.

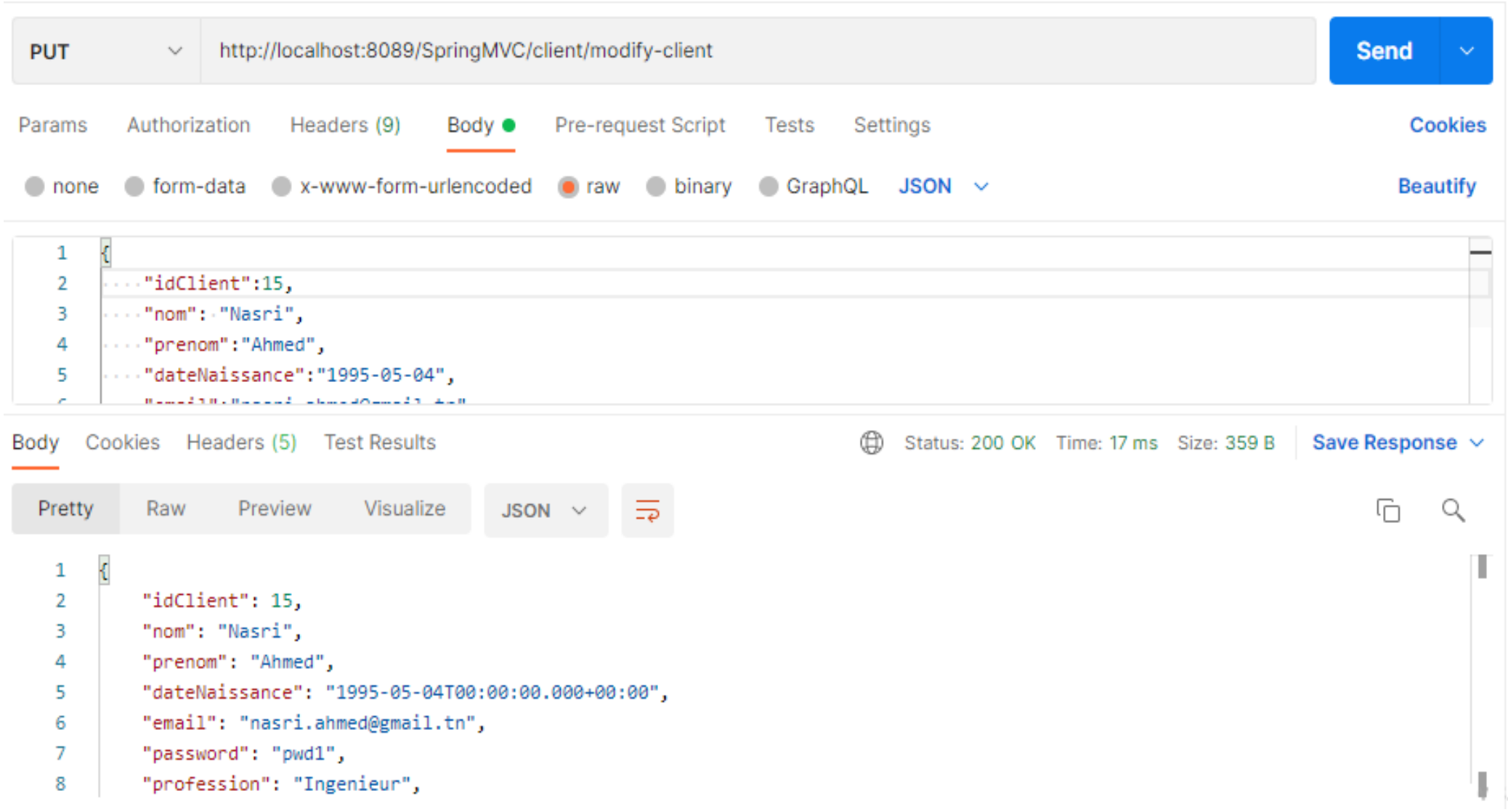
Spring MVC + Postman

Postman

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

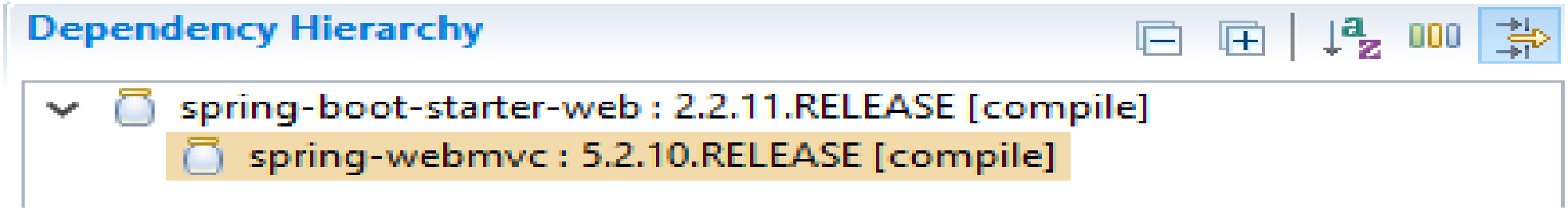
- Postman permet de construire et d'exécuter des requêtes HTTP, de les stocker dans un historique afin de pouvoir les rejouer.

Postman


PUT
http://localhost:8089/SpringMVC/client/modify-client
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"idClient":15,
"nom": "Nasri",
"prenom":"Ahmed",
"dateNaissance":"1995-05-04",
Body
Cookies
Headers (5)
Test Results
Status: 200 OK
Time: 17 ms
Size: 359 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"idClient": 15,
"nom": "Nasri",
"prenom": "Ahmed",
"dateNaissance": "1995-05-04T00:00:00.000+00:00",
"email": "nasri.ahmed@gmail.tn",
"password": "pwd1",
"profession": "Ingenieur",

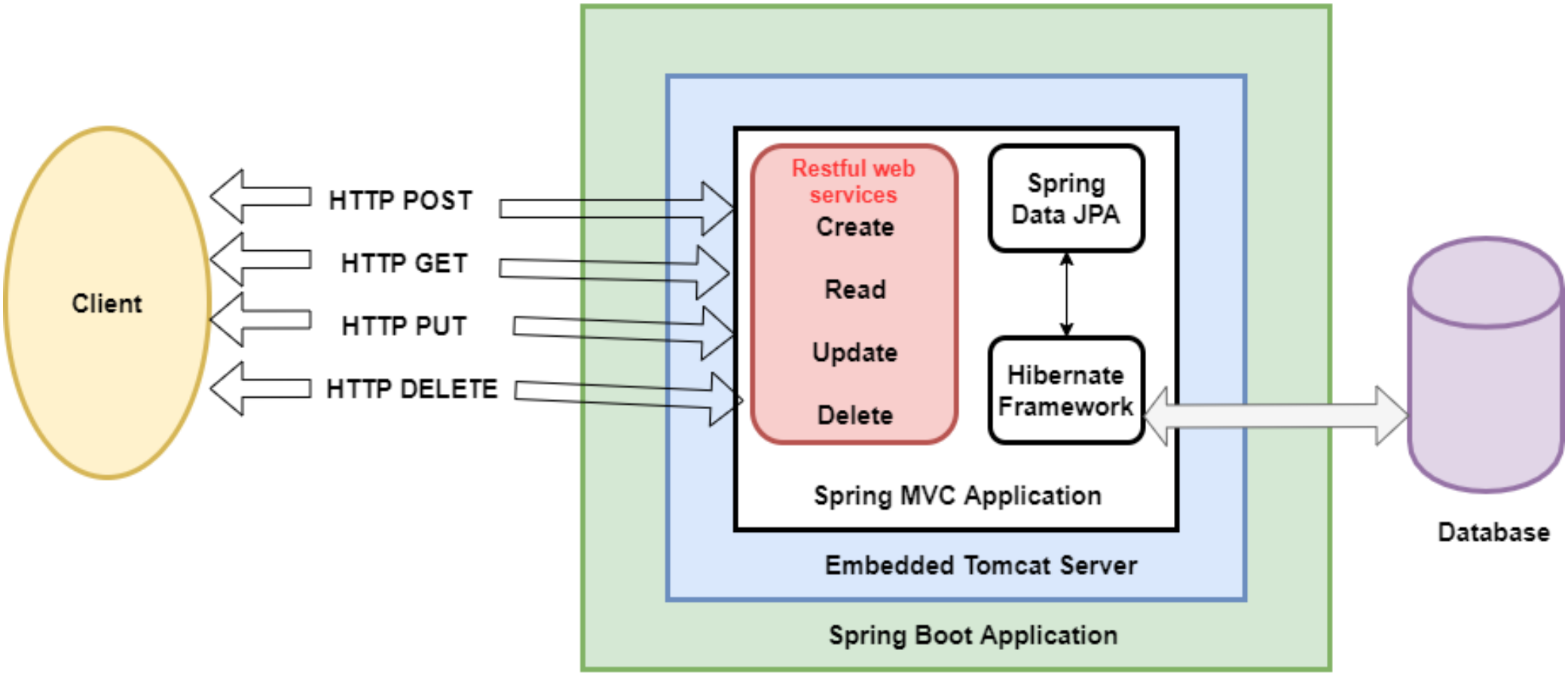
Dépendance web

```
<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)



Url de notre Application Web

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

```
#Server configuration
server.port=8089
server.servlet.context-path=/SpringMVC
```

RestController

```
@RestController
@RequestMapping("/client")
public class ClientRestController {

 @Autowired
 IClientService clientService;

 // http://localhost:8089/SpringMVC/client/retrieve-all-clients
 @GetMapping("/retrieve-all-clients")
 @ResponseBody
public List<Client> getClients() {
List<Client> listClients = clientService.retrieveAllClients();
return listClients;
   }
```

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 sur l'entité Client. Ce projet a été testé avec JUnit.
- Nous allons reprendre le même projet 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, …) :

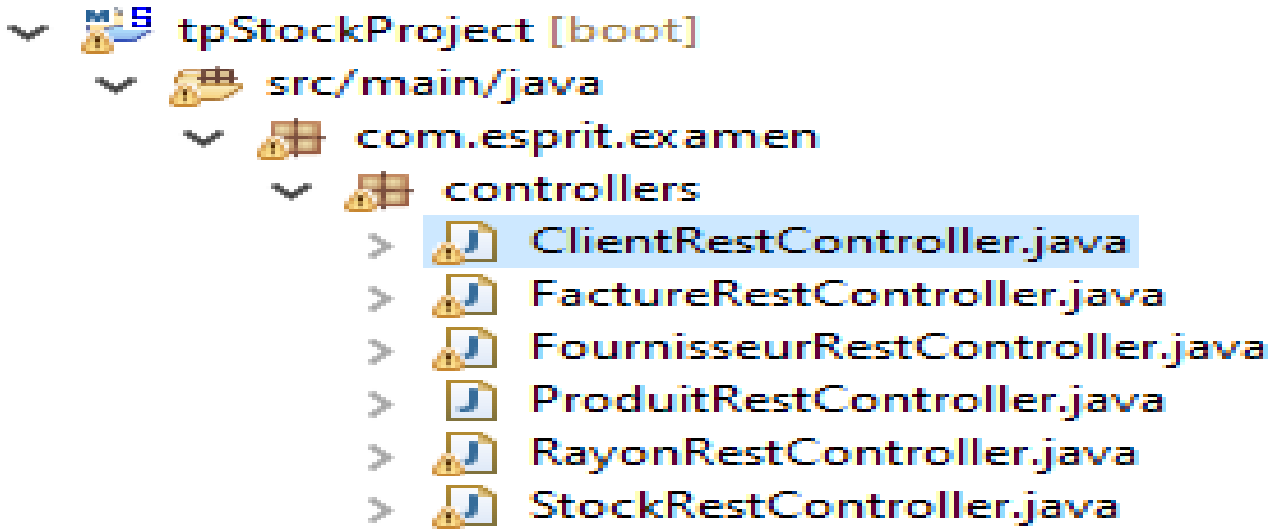
```
#Server configuration
server.servlet.context-path=/SpringMVC
server.port=8089

### 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
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect
```

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



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

```
@RestController
@RequestMapping("/client")
 public class ClientRestController {

@Autowired
IClientService clientService;

 // http://localhost:8089/SpringMVC/client/retrieve-all-clients
  @GetMapping("/retrieve-all-clients")
@ResponseBody
public List<Client> getClients() {
List<Client> listClients = clientService.retrieveAllClients();
 return listClients;
 }
```

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

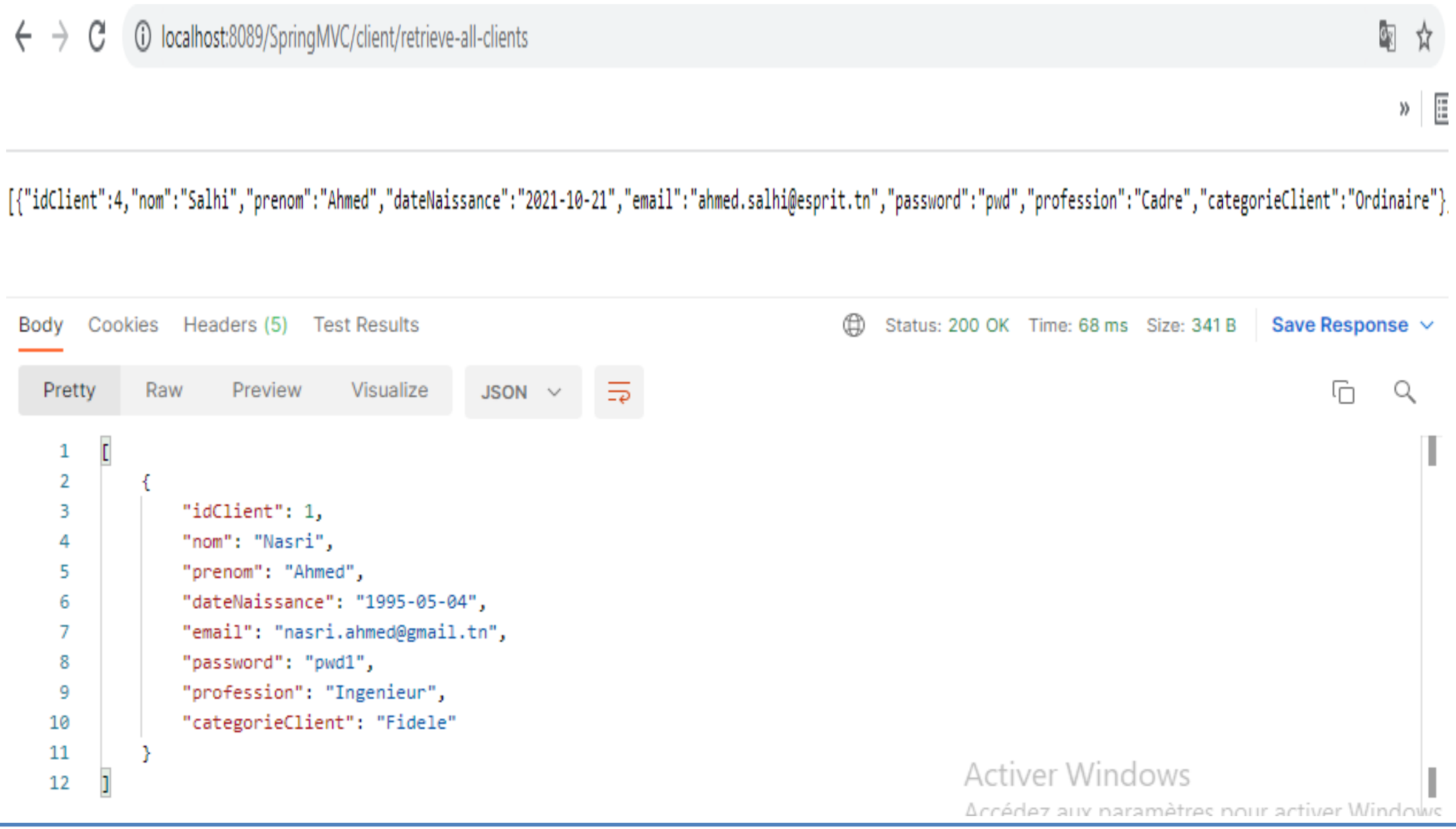
```
 // http://localhost:8089/SpringMVC/client/retrieve-client/8
 @GetMapping("/retrieve-client/{client-id}")
@ResponseBody
public Client retrieveClient(@PathVariable("client-id") Long clientId) {
return clientService.retrieveClient(clientId);
 }

 // http://localhost:8089/SpringMVC/client/add-client
 @PostMapping("/add-client")
 @ResponseBody
public Client addClient(@RequestBody Client c)
{
Client client = clientService.addClient(c);
return client;
 }
```

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

```
 // http://localhost:8089/SpringMVC/client/remove-client/{client-id}
@DeleteMapping("/remove-client/{client-id}")
@ResponseBody
public void removeClient(@PathVariable("client-id") Long clientId) {
clientService.deleteClient(clientId);
 }

// http://localhost:8089/SpringMVC/client/modify-client
@PutMapping("/modify-client")
@ResponseBody
public Client modifyClient(@RequestBody Client client) {
return clientService.updateClient(client);
 }
```

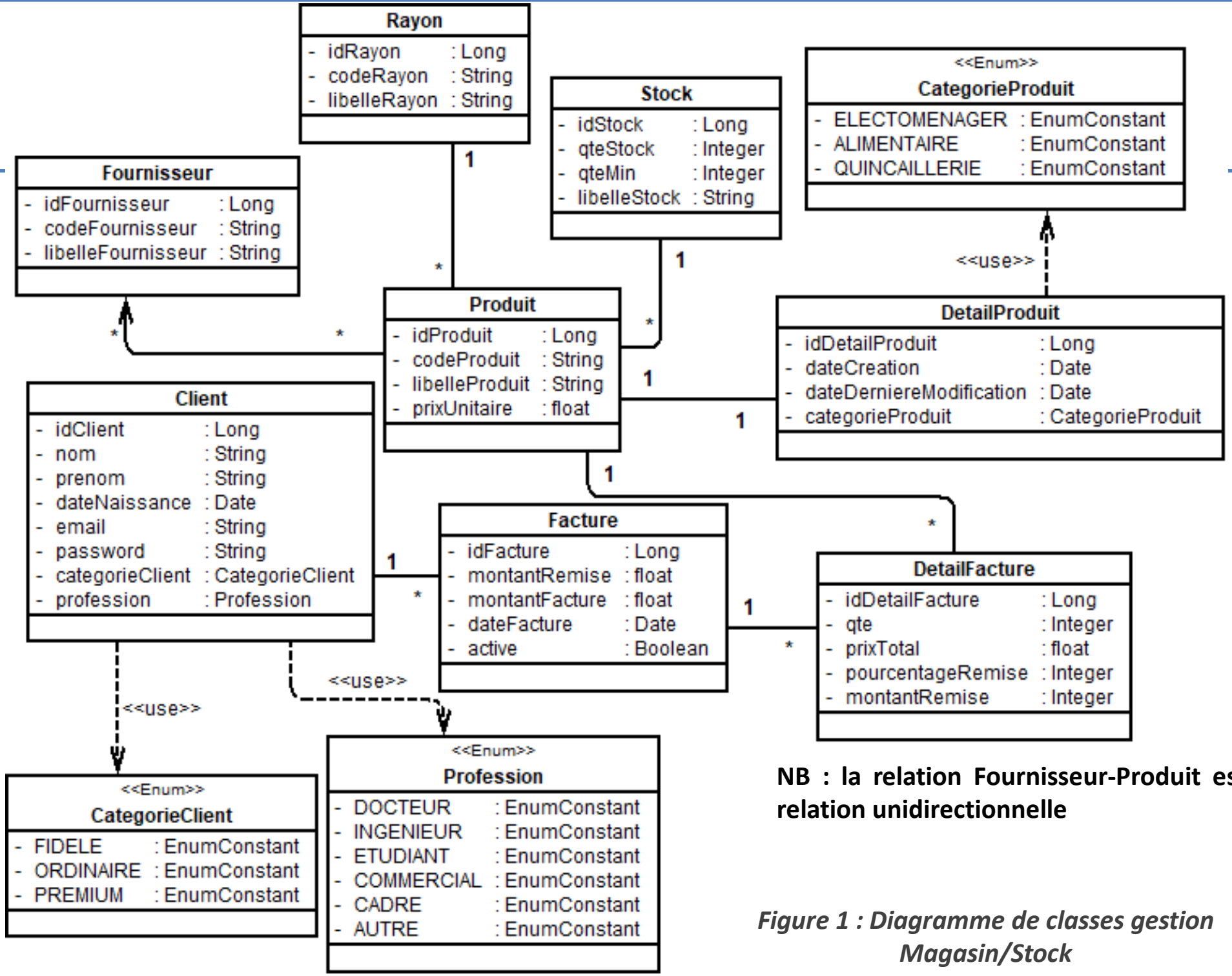
TP - Spring : Boot – Core – Data JPA – MVC (REST)

Liste des clients (navigateur + Postman)



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

- Tester l'ensemble des méthodes exposées avec Postman



NB : la relation Fournisseur-Produit est une relation unidirectionnelle

Figure 1 : Diagramme de classes gestion Magasin/Stock

Travail à faire

Spring MVC

Exposer les services implémentés dans la dernière séance avec Postman pour les tester.

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

Département Informatique
UP ASI
Bureau E204