

Rappel Séance 11

(week-end 29/30/2022)

Objectifs de la séance 12

(week-end 12/02/2022)

Les objectifs de la séance d'aujourd'hui:

Objectif 12.1 :

Utiliser le framework **Spring Data** pour implémenter la couche DAO

Objectif 12.2 :

Utiliser le framework **Spring Boot** pour configurer l'application

Complément Youtube de cette séance:

Dans ce TP on suppose que :

- ☒ ~~Vous avez réalisé totalement le TP8.~~
- ☒ ~~Vous avez réalisé totalement le TP9.~~
- ☒ ~~Vous avez réalisé totalement le TP10.~~
- ☒ ~~Vous avez réalisé totalement le TP11.~~

SI CE N'EST PAS LE CAS : FAIRE D'ABORD LE TP8, TP9, TP10 ET TP11 D'URGENCE

I. UTILISATION DE SPRING BOOT FRAMEWORK

1. AJOUTER LES DÉPENDANCES DU SPRING BOOT DANS LE FICHIER POM.XML

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>org.example</groupId>
  <artifactId>TP6_TP8_JPA</artifactId>
  <version>1.0-SNAPSHOT</version>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.5.1</version>
    <relativePath/>
  </parent>
```

```
<properties>
  <maven.compiler.source>11</maven.compiler.source>
  <maven.compiler.target>11</maven.compiler.target>
</properties>

<dependencies>
  <!-- Spring Data Starter -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>
  <!-- Spring Web MVC Starter -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <!-- MySQL Driver -->
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.6</version>
  </dependency>
  <!-- Lombok Annotations -->
  <dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.20</version>
    <scope>provided</scope>
  </dependency>

  <dependency>
    <groupId>javax.xml.bind</groupId>
    <artifactId>jaxb-api</artifactId>
    <version>2.3.1</version>
  </dependency>
</dependencies>

</project>
```

2. SUPPRIMER LE FICHIER DE CONFIGURATION SPRING.XML ET LE FICHIER META-INF/PERSISTENCE.XML
3. MODIFIER LA CLASSE DE DEMARRAGE APPLICATIONRUNNER

```
package ma.cigma;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import
```

4. Créer le fichier de configuration resources/application.yml

5. Exécuter la classe Application Runner pour avoir les traces suivantes dans la console:

Mise à jour 18 Nov 2021

```
.s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in
DEFAULT mode.
2022-02-10 23:28:25.946 INFO 4000 --- [          main]
.s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in
78 ms. Found 2 JPA repository interfaces.
2022-02-10 23:28:27.477 INFO 4000 --- [          main]
o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8083 (http)
2022-02-10 23:28:27.493 INFO 4000 --- [          main]
o.apache.catalina.core.StandardService : Starting service [Tomcat]
2022-02-10 23:28:27.493 INFO 4000 --- [          main]
org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache
Tomcat/9.0.46]
2022-02-10 23:28:27.618 INFO 4000 --- [          main]
o.a.c.c.C.[Tomcat].[localhost].[/api] : Initializing Spring embedded
WebApplicationContext
2022-02-10 23:28:27.618 INFO 4000 --- [          main]
w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization
completed in 3316 ms
2022-02-10 23:28:27.790 INFO 4000 --- [          main]
com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2022-02-10 23:28:28.009 INFO 4000 --- [          main] com.zaxxer.hikari.pool.PoolBase
: HikariPool-1 - Driver does not support get/set network timeout for connections.
(Receiver class com.mysql.jdbc.JDBC4Connection does not define or inherit an
implementation of the resolved method abstract getNetworkTimeout()I of interface
java.sql.Connection.)
2022-02-10 23:28:28.024 INFO 4000 --- [          main]
com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2022-02-10 23:28:28.243 INFO 4000 --- [          main]
o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnitInfo
[name: default]
2022-02-10 23:28:28.306 INFO 4000 --- [          main] org.hibernate.Version
: HHH000412: Hibernate ORM core version 5.4.32.Final
2022-02-10 23:28:28.462 INFO 4000 --- [          main]
o.hibernate.annotations.common.Version : HCANN000001: Hibernate Commons Annotations
{5.1.2.Final}
2022-02-10 23:28:28.681 INFO 4000 --- [          main] org.hibernate.dialect.Dialect
: HHH000400: Using dialect: org.hibernate.dialect.MySQL5Dialect
2022-02-10 23:28:29.134 INFO 4000 --- [          main]
org.hibernate.tuple.PojoInstantiator : HHH000182: No default (no-argument)
constructor for class: ma.cigma.models.Address (class must be instantiated by
Interceptor)
Hibernate: drop table if exists address
Hibernate: drop table if exists client
Hibernate: create table address (id bigint not null auto_increment, description
varchar(255), fk_client_id bigint, primary key (id)) engine=MyISAM
Hibernate: create table client (id bigint not null auto_increment, name varchar(255),
primary key (id)) engine=MyISAM
Hibernate: alter table address add constraint FKagg3d6rjs3qe9y1j1blwv06nm foreign key
(fk_client_id) references client (id)
2022-02-10 23:28:29.618 INFO 4000 --- [          main]
o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000490: Using JtaPlatform implementation:
[org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
2022-02-10 23:28:29.634 INFO 4000 --- [          main]
j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for
persistence unit 'default'
2022-02-10 23:28:30.180 WARN 4000 --- [          main]
JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by
default. Therefore, database queries may be performed during view rendering. Explicitly
configure spring.jpa.open-in-view to disable this warning
2022-02-10 23:28:30.759 INFO 4000 --- [          main]
o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8083 (http) with
```

```
context path '/api'
2022-02-10 23:28:30.774 INFO 4000 --- [main] ma.cigma.ApplicationRunner
: Started ApplicationRunner in 7.65 seconds (JVM running for 8.622)
```

II. UTILISATION DE L'API REST : @RestController

1. D'abord, ajouter des clients dans la base pour tester. Pour le faire créer la classe Mock Clients qui **implements** CommandLineRunner

```
package ma.cigma.dao;

import ma.cigma.models.Client;
import ma.cigma.service.IClientService;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.context.annotation.Bean;

import org.springframework.stereotype.Component;

@Component
public class MockClients implements CommandLineRunner {
    private static final Logger logger =
        LoggerFactory.getLogger(MockClients.class);

    @Autowired
    private IClientDao clientRepository;

    @Override
    public void run(String... args) throws Exception {
        Client c1 = new Client("Omar");
        Client c2 = new Client("Said");
        Client c3 = new Client("Ahmed");
        Client c4 = new Client("Farah");
        clientRepository.save(c1);
        clientRepository.save(c2);
        clientRepository.save(c3);
        clientRepository.save(c4);
        clientRepository.findAll().forEach(c ->
System.out.println
(c)
        );
    }
}
```

NB1: Vous devriez ajouter le constructeur par défaut au niveau de la classe model : Client

Dossier des travaux pratiques. Module 1 : Java de base .Années scolaire 2021/2022. Niveau : Licence FST Settat

Professeur : M. Boulchahoub Hassan hboulchahoub@gmail.com

Mise à jour 18 Nov 2021

NB2: Il faut redéfinir la méthode toString au niveau de la classe Client

2. Vous devriez avoir dans la console les traces suivantes:

```

Hibernate: insert into client (name) values (?)
Hibernate: insert into client (name) values (?)
Hibernate: insert into client (name) values (?)
Hibernate: insert into client (name) values (?)
Hibernate: select client0_.id as id1_1_, client0_.name as name2_1_ from
client client0_
Hibernate: select addresses0_.fk_client_id as fk_clien3_0_0_,
addresses0_.id as id1_0_0_, addresses0_.id as id1_0_1_,
addresses0_.fk_client_id as fk_clien3_0_1_, addresses0_.description as
descript2_0_1_ from address addresses0_ where addresses0_.fk_client_id=?
Hibernate: select addresses0_.fk_client_id as fk_clien3_0_0_,
addresses0_.id as id1_0_0_, addresses0_.id as id1_0_1_,
addresses0_.fk_client_id as fk_clien3_0_1_, addresses0_.description as
descript2_0_1_ from address addresses0_ where addresses0_.fk_client_id=?
Hibernate: select addresses0_.fk_client_id as fk_clien3_0_0_,
addresses0_.id as id1_0_0_, addresses0_.id as id1_0_1_,
addresses0_.fk_client_id as fk_clien3_0_1_, addresses0_.description as
descript2_0_1_ from address addresses0_ where addresses0_.fk_client_id=?
Hibernate: select addresses0_.fk_client_id as fk_clien3_0_0_,
addresses0_.id as id1_0_0_, addresses0_.id as id1_0_1_,
addresses0_.fk_client_id as fk_clien3_0_1_, addresses0_.description as
descript2_0_1_ from address addresses0_ where addresses0_.fk_client_id=?
Client{id=1, name='Omar'}
Client{id=2, name='Said'}
Client{id=3, name='Ahmed'}
Client{id=4, name='Farah'}

```

3. Maintenant, annoter la classe ClientController en utilisant l'annotation @RestController et @RequestMapping

```

package ma.cigma.presentation;

import ma.cigma.models.Client;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import ma.cigma.service.IClientService;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import
org.springframework.web.bind.annotation.RequestMapping;
import
org.springframework.web.bind.annotation.RestController;

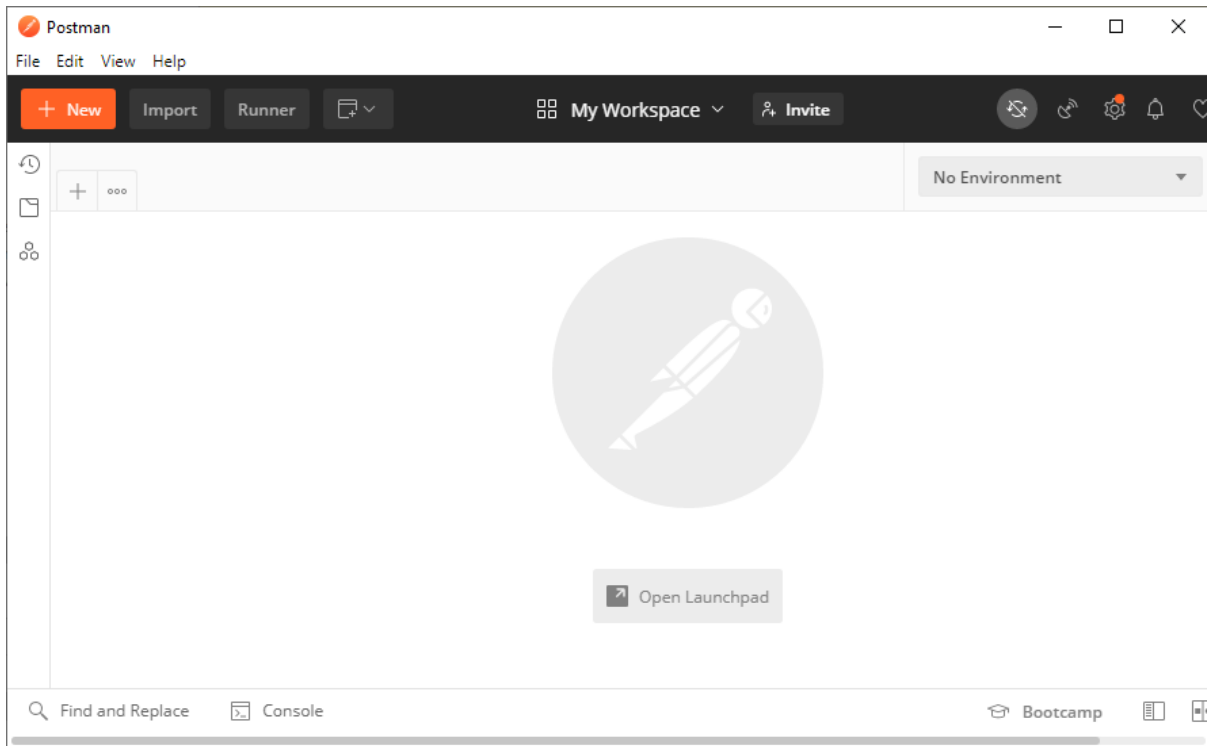
import java.util.List;

@RestController
@RequestMapping("/client")

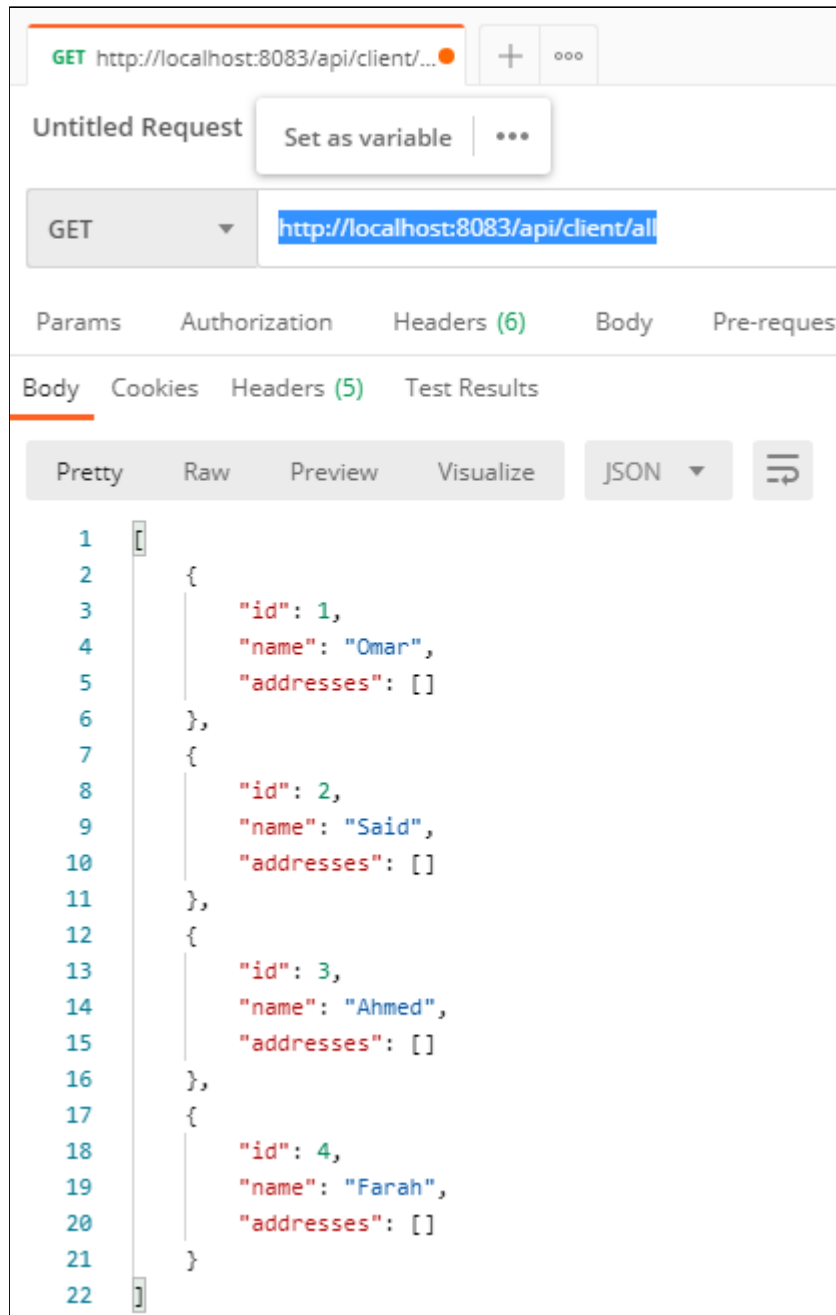
```

```
public class ClientController {  
  
    @Autowired  
    private IClientService service;  
  
    @GetMapping("/{id}")  
    public Client getOne(@PathVariable("id") long id) {  
        return service.getOne(id);  
    }  
  
    @GetMapping("/all")  
    public List<Client> getAll() {  
        return service.getAll();  
    }  
  
    public Client save(Client clt) {  
        return service.save(clt);  
    }  
  
    public Client modify(Client clt) {  
        return service.modify(clt);  
    }  
}
```

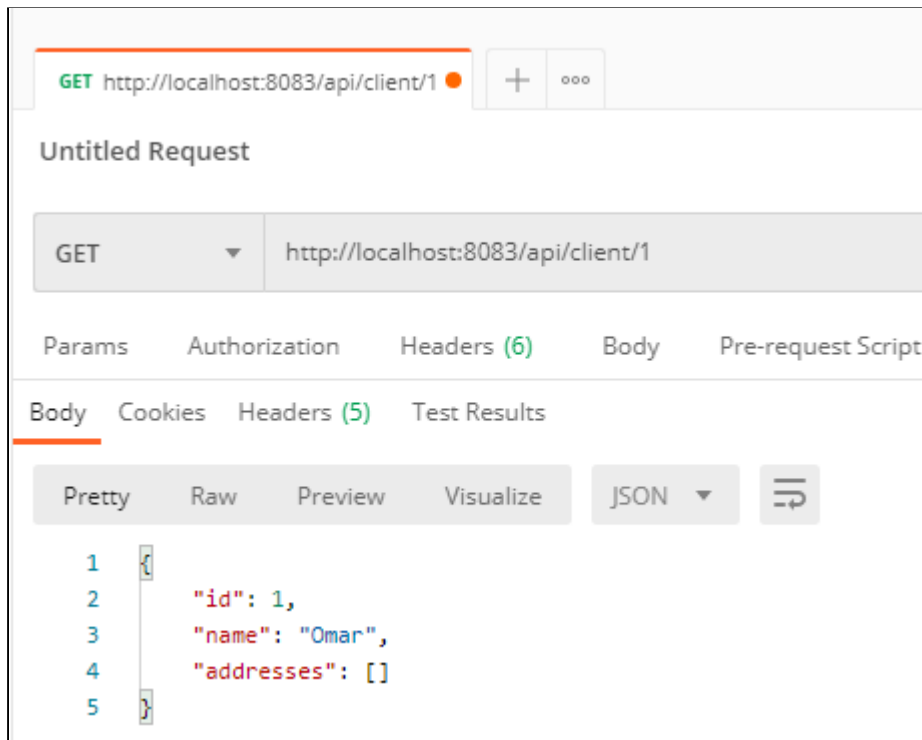
4. Tester l'appel des méthodes getAll() et getOne(id) par un client Rest: **PostMan** à titre exemple ou **Advanced REST client chrome extension**.
- a. Télécharger et installer PostMan
<https://www.postman.com/downloads/>



5. Démarrer l'application en exécutant la classe `ApplicationRunner`.
6. Dans la partie réservée à l'URL saisir l'url suivante:
<http://localhost:8083/api/client/all>



7. Dans la partie réservée à l'URL saisir l'url suivante:
<http://localhost:8083/api/client/1>



8. Créer les services rest suivantes :
 - a. Ajouter un client
 - b. Modifier un client
9. Le classe Client Controller devient :

```
package ma.cigma.presentation;

import ma.cigma.models.Client;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import ma.cigma.service.IClientService;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/client")
public class ClientController {

    @Autowired
    private IClientService service;

    @GetMapping("/{id}")
    public Client getOne(@PathVariable("id") long id) {
        return service.getOne(id);
    }
}
```

```

@GetMapping("/all")
public List<Client> getAll(){
    return service.getAll();
}

@PostMapping("/create")
public Client save(@RequestBody Client clt) {
    return service.save(clt);
}

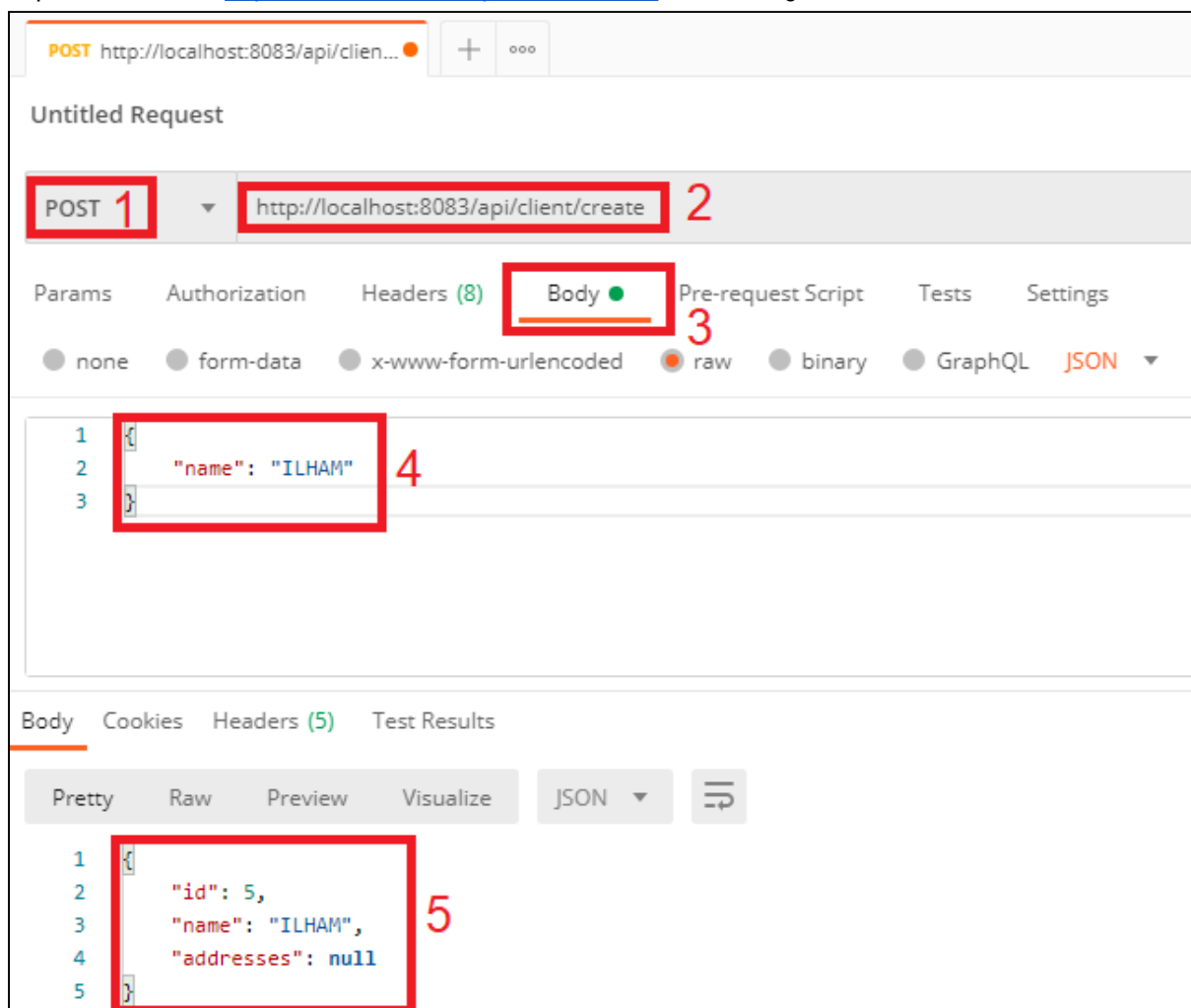
@PutMapping("/update")
public Client modify(@RequestBody Client clt){
    return service.modify(clt);
}
}

```

10. Redémarrer l'application en exécutant la classe ApplicationRunner.

11. Tester le Service REST create en utilisant PostMan:

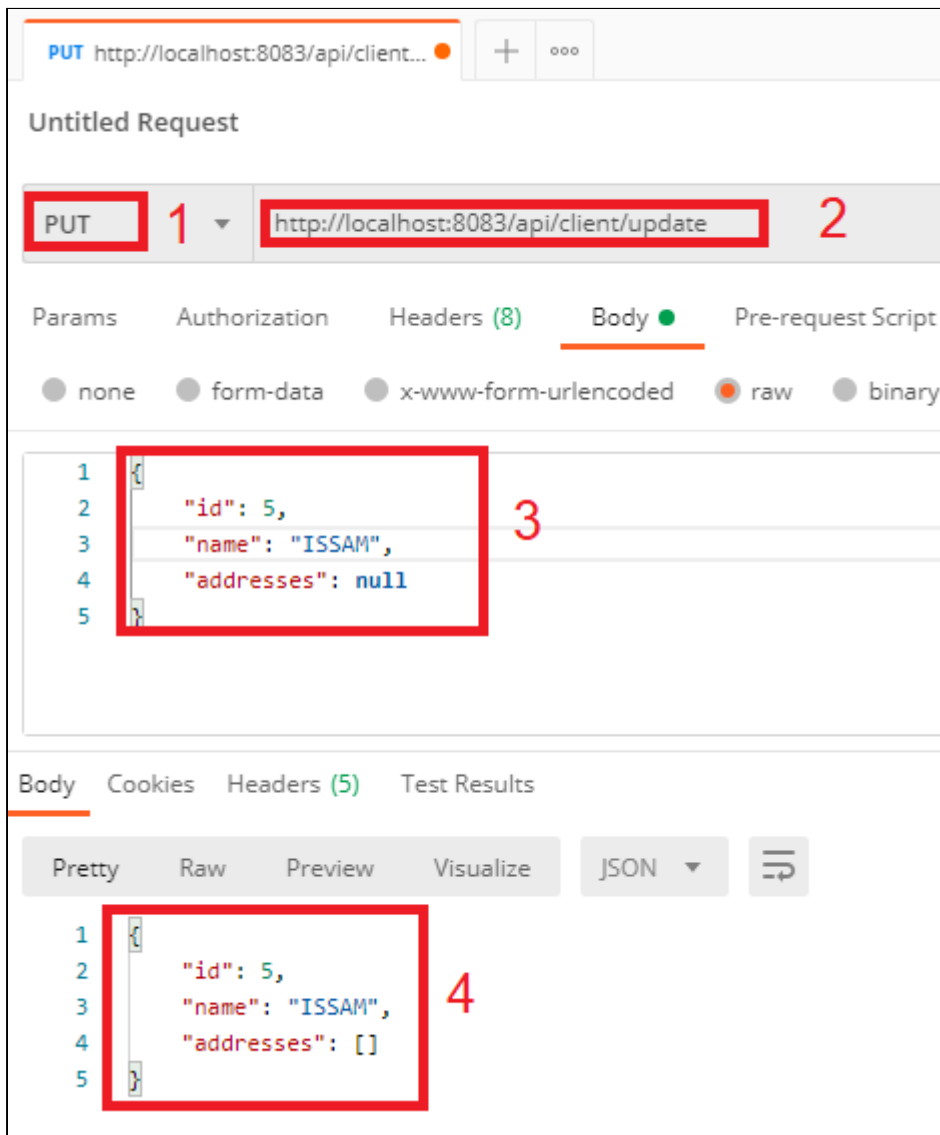
Tapez l'url : <http://localhost:8083/api/client/create> Pour l'ajout d'un client



The screenshot shows the Postman interface for a POST request. The URL is `http://localhost:8083/api/client/create`. The request body is a JSON object: `{ "name": "ILHAM" }`. The response body is a JSON object: `{ "id": 5, "name": "ILHAM", "addresses": null }`. Red boxes and numbers 1 through 5 highlight the following elements:

1. The **POST** method dropdown.
2. The **URL** input field.
3. The **Body** tab and the **raw** radio button.
4. The **JSON** body content: `{ "name": "ILHAM" }`.
5. The **JSON** response content: `{ "id": 5, "name": "ILHAM", "addresses": null }`.

12. Faire attention aux points mentionnés ci-dessus.
 - 1 - Rest Method (Post)
 - 2 - Rest URL
 - 3 - Rest Request Body
 - 4 - Rest Request Body Content
 - 5 - Rest Response Body Content
13. Tester le Service REST update en utilisant PostMan:
Tapez l'url : <http://localhost:8083/api/client/update> Pour l'ajout d'un client



14. Faire attention aux points mentionnés ci-dessus.
 - 1 - Rest Method (Post)
 - 2 - Rest URL
 - 3 - Rest Request Body
 - 4 - Rest Request Body Content
 - 5 - Rest Response Body Content