

Développement des applications Web basées sur les microservices (Partie 1)

Responsable du cours: Dr. Mariam LAHAMI

Email: lahami.mariam@gmail.com



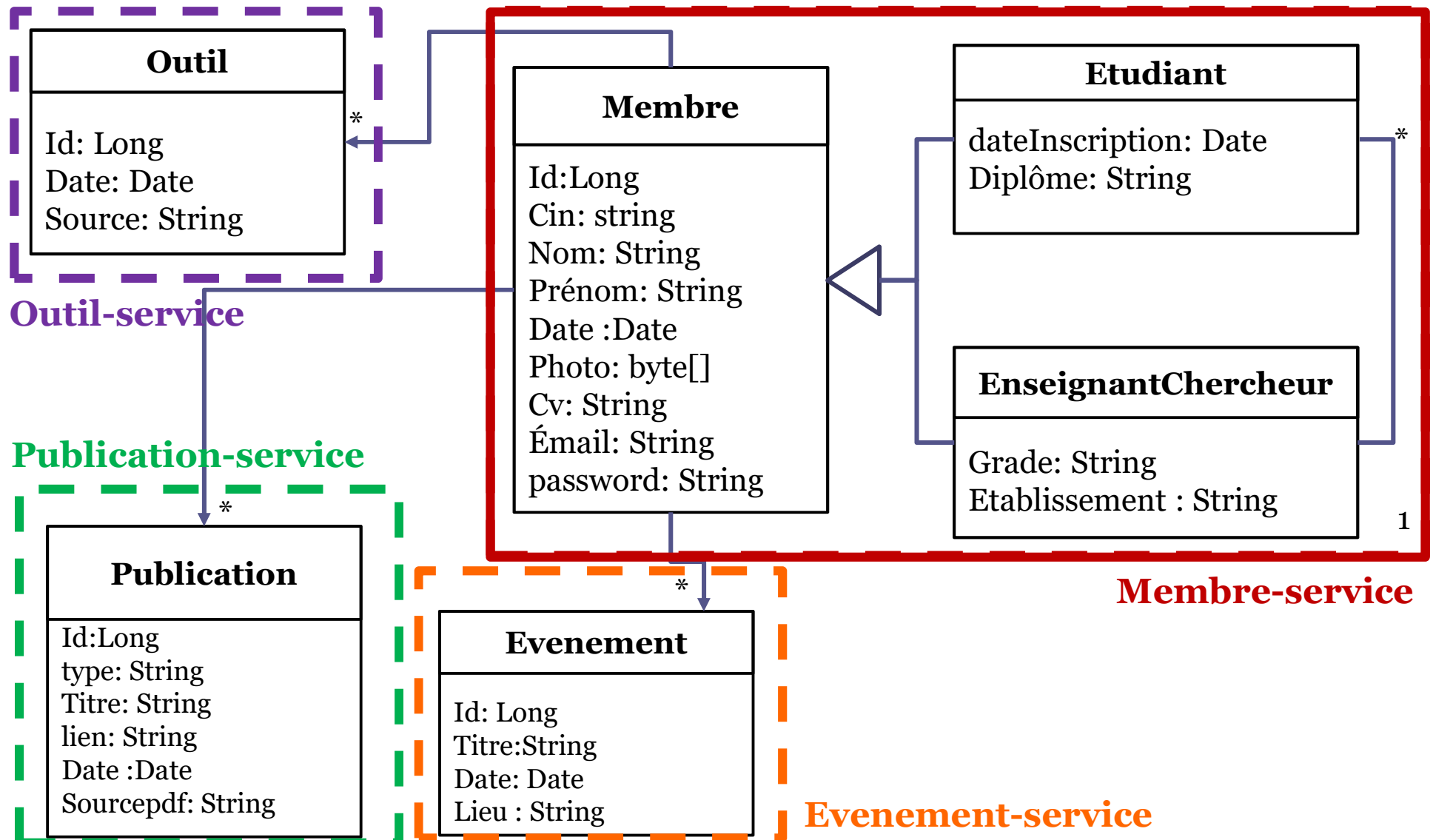
SPRING
Framework

Développement d'un micro-service avec Spring boot

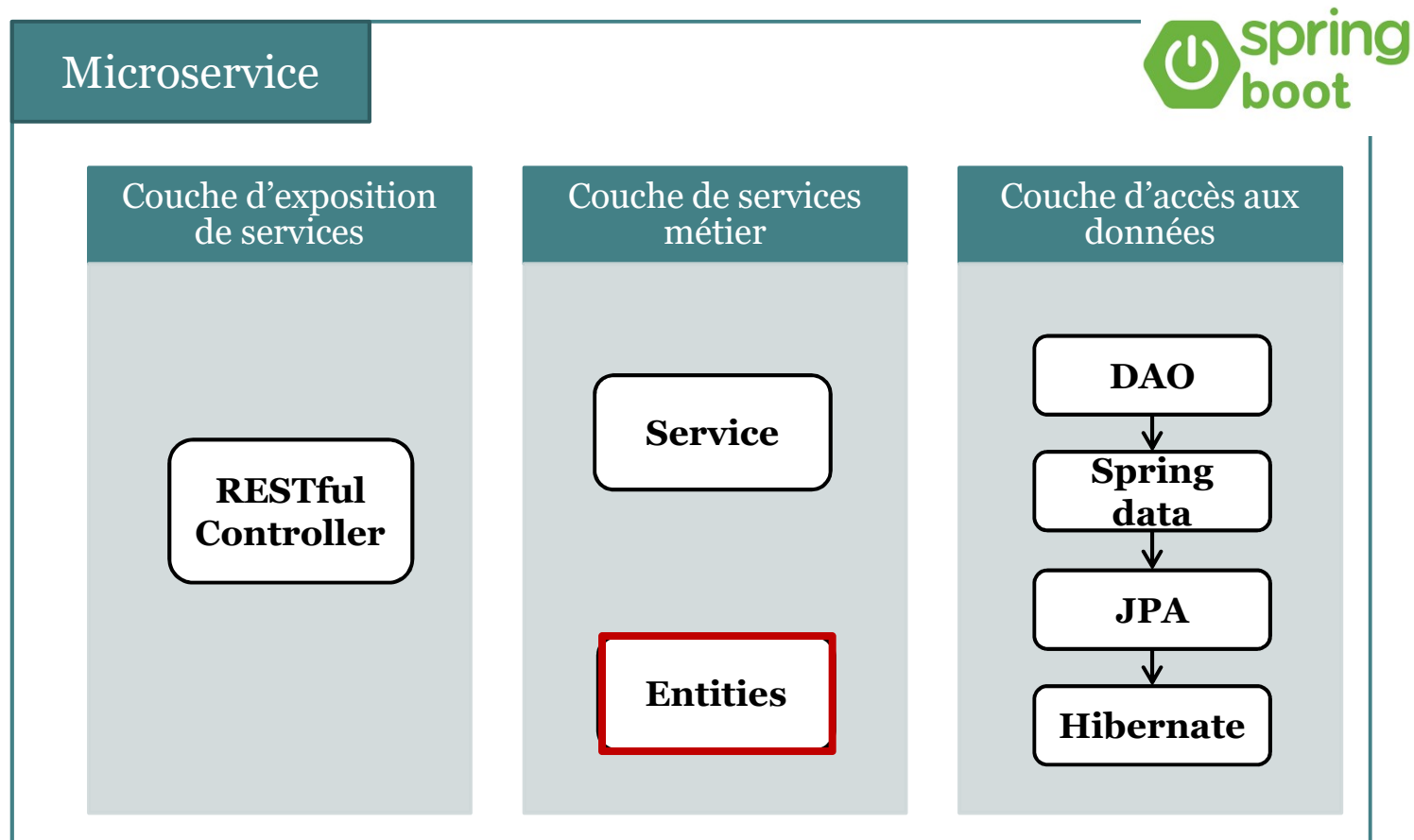
- Développement des entités avec JPA
- Développement de la couche DAO avec Spring Data
- Développement de la couche métier
- Développement de la couche controller

« Do one thing and do it well »

Étude de cas: 4 microservices



Structure d'un microservice



Création du projet Membre-service

- Créer un projet Spring Boot Starter
- Ajouter les dépendances suivantes

```
X Spring Boot DevTools  
X Lombok  
X Spring Data JPA  
X MySQL Driver  
X Spring Web
```

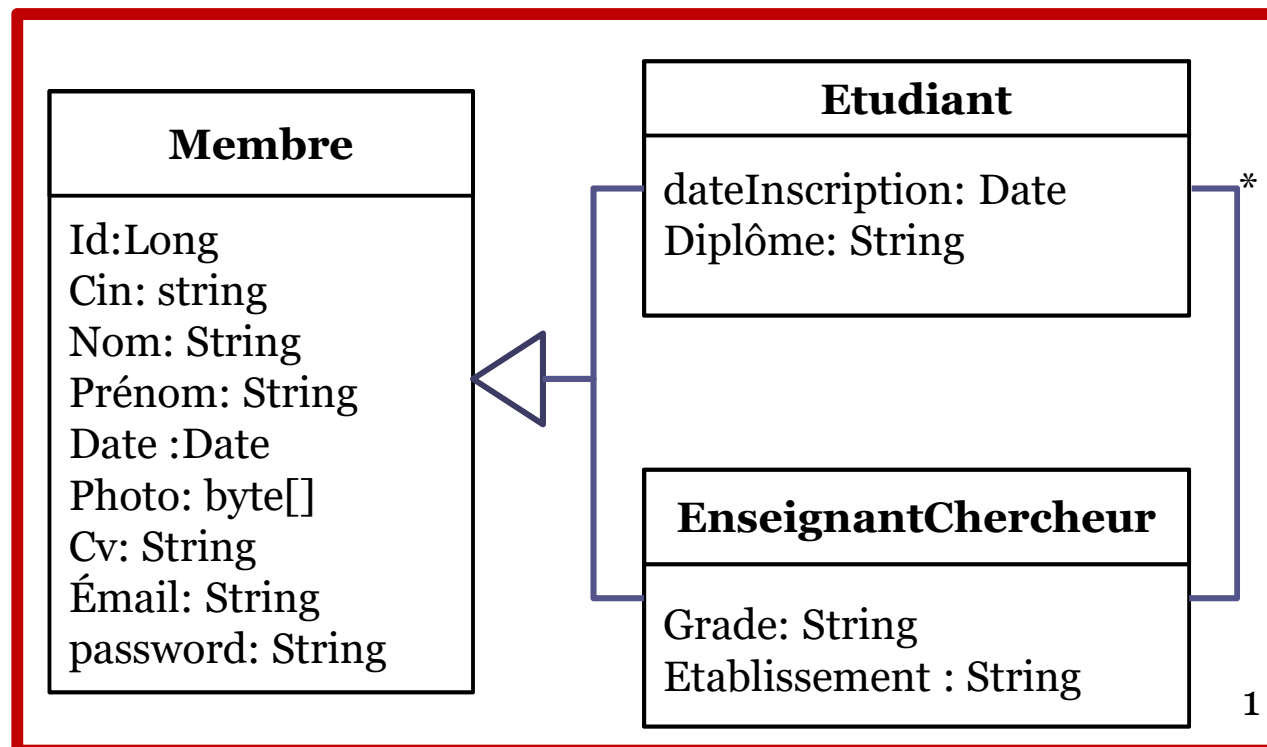
Création du projet Membre-service

- Configurer le fichier Application.properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/lab2023
spring.datasource.username=root
spring.datasource.password=
spring.datasource.driverClassName=com.mysql.cj.jdbc.Driver
spring.jpa.hibernate.ddl-auto=create
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQLDialect
spring.jpa.show-sql = true
```

Microservice Membre

Implémenter ces entités



Membre-service

Rappel sur la création des entités persistantes avec JPA (1/2)

- Entité persistance : JAVA bean avec **@Entity**
- Des getters et des setters
- Un constructeur sans paramètres
- Clé primaire : **@Id**
- Méthode de génération de l'identifiant :
@GeneratedValue(strategy=GenerationType.Identity)

Rappel sur la création des entités persistantes avec JPA (2/2)

- Afin d'avoir une seule table pour stocker tous les membres (étudiants et enseignants) :
 - @Inheritance au niveau de la classe mère

```
@Entity @Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name= "type_mbr", discriminatorType =
DiscriminatorType.STRING, length = 3)
public abstract class Membre{ ... }
```

- @DiscriminatorValue ("val") au niveau de la classe fille

```
@Entity @DiscriminatorValue("etd")
public class Etudiant{ ... }
```

- Relation entre Etudiant et EnseignantChercheur

```
@ManyToOne
private EnseignantChercheur encadrant;
```

Rappel sur la création des entités avec Lombok (1/4)

- **@ Getter** : générer les getters sur les champs
- **@ Setter** : générer les setters sur les champs
- **@NoArgsConstructor** : générer le constructeur sans paramètres
- **@AllArgsConstructor** : générer le constructeur avec paramètres
- **@RequiredArgsConstructor** : générer un constructeurs avec des paramètres spécifiés avec l'annotation **@nonNull**

Rappel sur la création des entités avec Lombok (2/4)

- `@FieldDefaults(level=AccessLevel.PRIVATE)` :
passe tous les champs en private
- `@EqualsAndHashCode(of=...)` :
génère equals et hashCode (et d'autres méthodes) sur les champs donnés ;
- `@ToString(of=...)` : génère toString sur les champs donnés.
- `@Data= @FieldDefaults(level=AccessLevel.PRIVATE) + @Setter + @Getter + @ToString`

Rappel sur la création des entités avec Lombok (3/4)

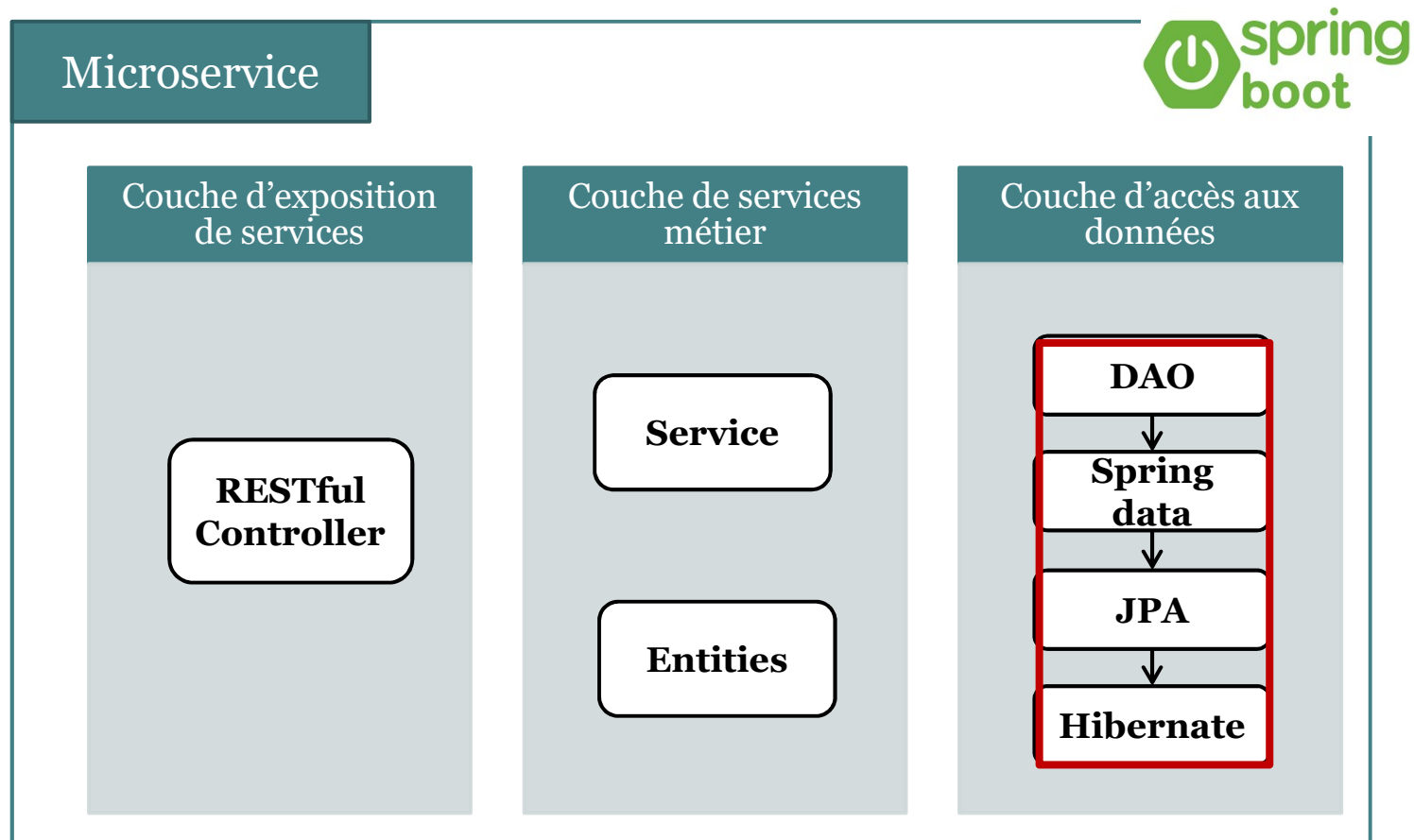
```
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name= "type_mbr", discriminatorType =
DiscriminatorType.STRING, length = 3)
@Getter @Setter
@NoArgsConstructor
@AllArgsConstructor
@RequiredArgsConstructor
public abstract class Membre implements Serializable {
@Id @GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;
@NotNull
private String cin;
@NotNull
private String nom;
@NotNull
private String prenom;
@NotNull @Temporal(TemporalType.DATE)
private Date dateNaissance;
```

Rappel sur la création des entités avec Lombok (4/4)

```
@Builder
public Etudiant( String cin, String nom, String prenom, Date dateNaissance,
String cv,
String email, String password, Date dateInscription, String sujet, String
diplome,
EnseignantChercheur encadrant) {
super( cin, nom, prenom, dateNaissance, cv, email, password);
this.dateInscription = dateInscription;
this.sujet = sujet;
this.diplome = diplome;
this.encadrant = encadrant;
}
```

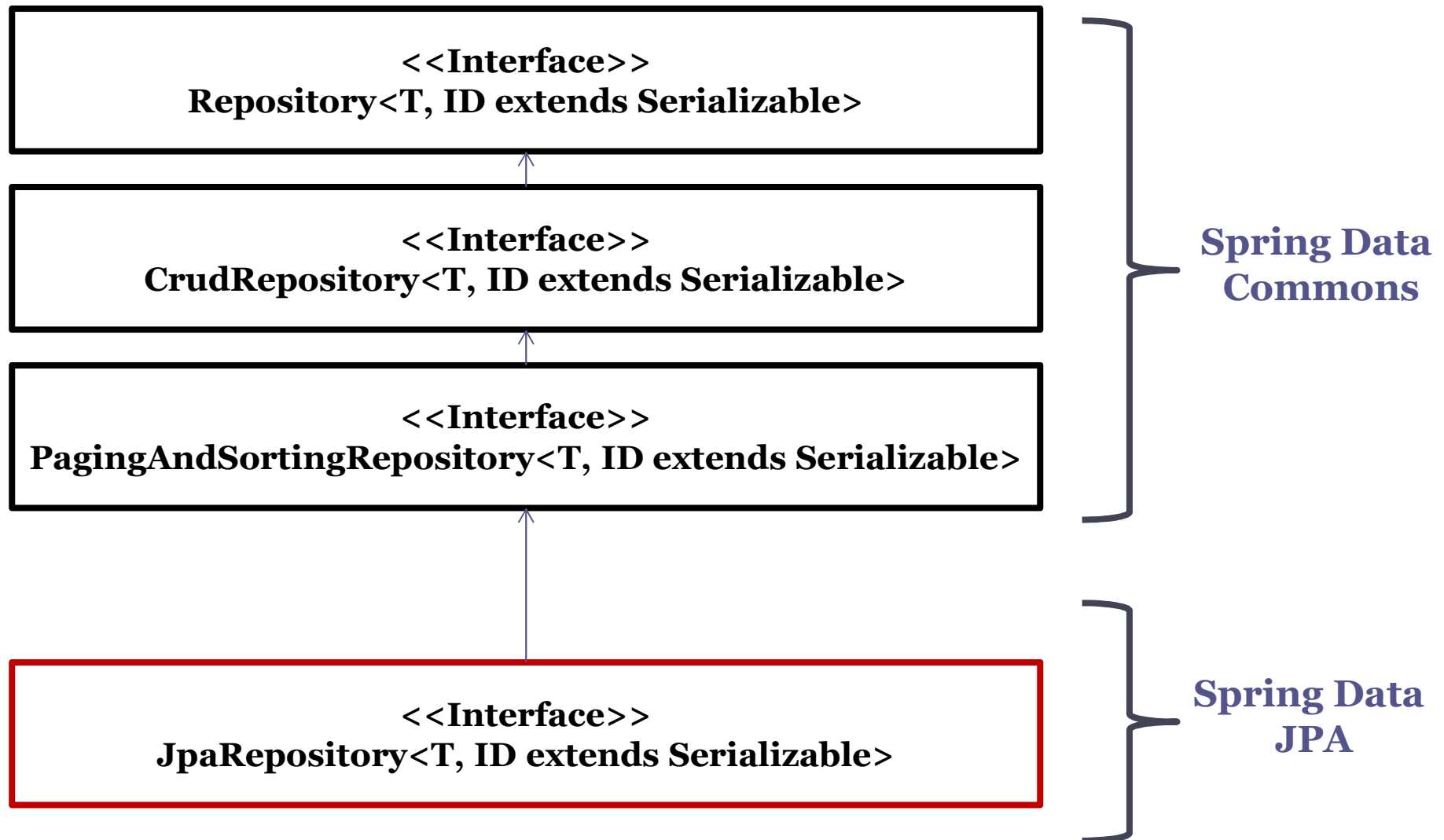
```
Etudiant etd1=Etudiant.builder()
.cin("123456")
.dateInscription(new Date())
.dateNaissance(new Date())
.diplome("mastère")
.email("etd1@gmail.com")
.password("pass1")
.encadrant(null)
.cv("cv1")
.nom("abid")
.prenom("youssef")
.sujet("blockchain")
.build();
```

Structure d'un microservice



Utilisation de Spring Data au niveau de la couche DAO

Spring Data

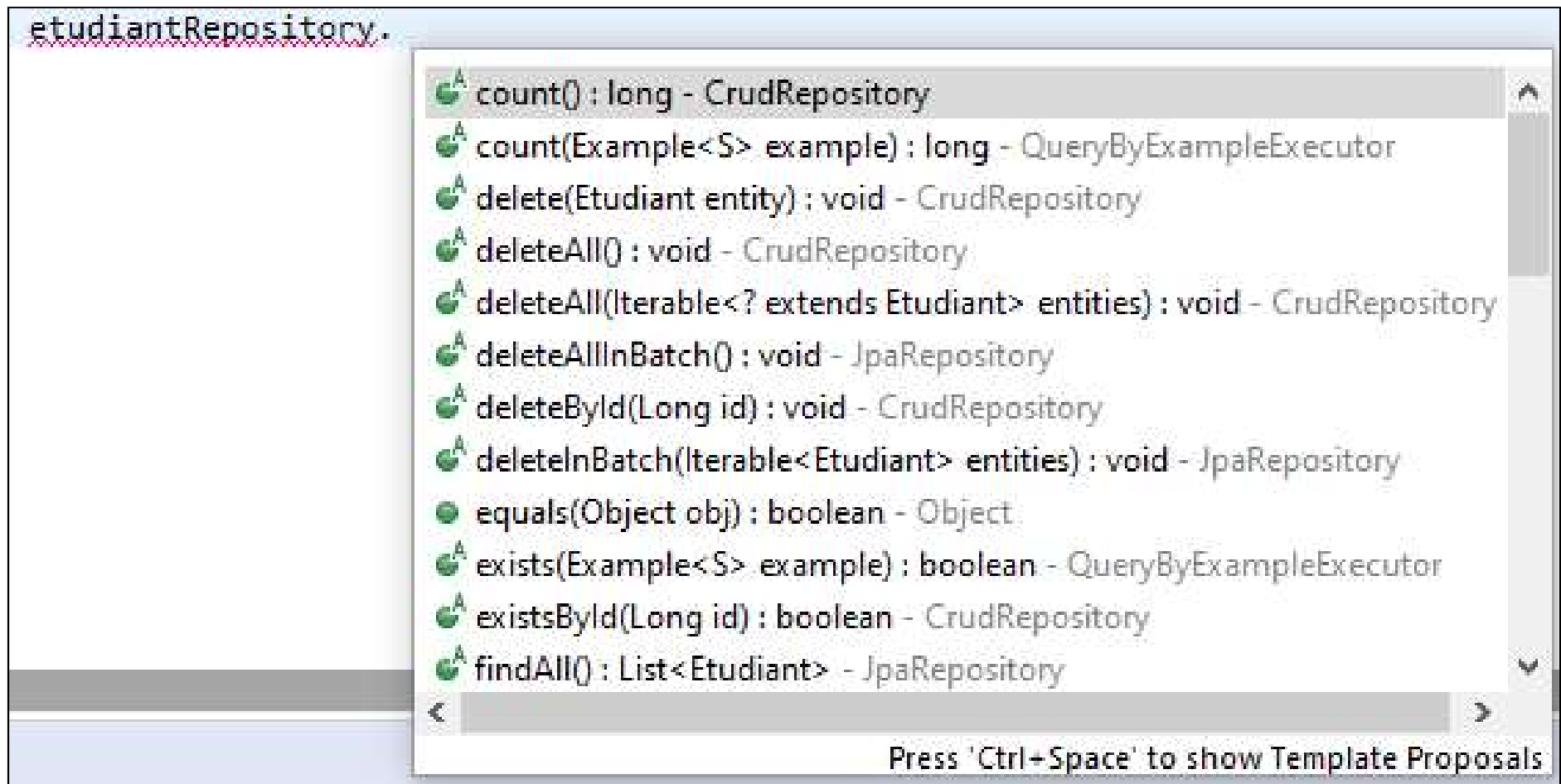


Spring Data JPA via l'exemple (1/3)

- Pour utiliser Spring Data, il faut déclarer spring-boot-starter-data-jpa comme dépendance dans le pom.xml
- Créer pour chaque entité persistante, une interface héritant de l'interface **JpaRepository**

```
public interface EtudiantRepository extends JpaRepository<Etudiant,  
Long> {  
  
}
```

Spring Data JPA via l'exemple (2/3)



Spring Data JPA via l'exemple (3/3)

- On peut spécifier ses propres Query JPQL directement dans le corps de l'interface, en redéfinissant l'implémentation standard par une annotation **@Query**(« select... »).

```
public interface EtudiantRepository extends JpaRepository<Etudiant,  
Long> {  
    @Query("select e from Etudiant e where e.nom like :x")  
    public List<Etudiant> chercher(@Param("x")String mc);  
}
```

Requête générée par le nom de méthode

- La magie de Spring Data va au-delà des requêtes CRUD. Le framework est capable de **générer la requête SQL automatiquement en partant du nom d'une méthode !**
- Spring Data JPA utilise un ensemble des mots clés pour générer automatiquement les requêtes :

<https://docs.spring.io/spring-data/data-jpa/docs/1.1.x/reference/html/#jpa.query-methods.query-creation>

- Récupérons par exemple la liste des étudiants en mastère, il suffit de définir cette méthode dans EtudiantRepository

findByDiplome (String diplôme)

Requête générée par le nom de méthode

Keyword	Sample	JPQL snippet
And	<code>findByLastnameAndFirstname</code>	<code>... where x.lastname = ?1 and x.firstname = ?2</code>
Or	<code>findByLastnameOrFirstname</code>	<code>... where x.lastname = ?1 or x.firstname = ?2</code>
Is , Equals	<code>findByFirstname</code> , <code>findByFirstnameIs</code> , <code>findByFirstnameEquals</code>	<code>... where x.firstname = ?1</code>
Between	<code>findByStartDateBetween</code>	<code>... where x.startDate between ?1 and ?2</code>
LessThan	<code>findByAgeLessThan</code>	<code>... where x.age < ?1</code>
LessThanEqual	<code>findByAgeLessThanEqual</code>	<code>... where x.age <= ?1</code>
GreaterThan	<code>findByAgeGreaterThan</code>	<code>... where x.age > ?1</code>
GreaterThanEqual	<code>findByAgeGreaterThanEqual</code>	<code>... where x.age >= ?1</code>
After	<code>findByStartDateAfter</code>	<code>... where x.startDate > ?1</code>
Before	<code>findByStartDateBefore</code>	<code>... where x.startDate < ?1</code>

Création de la couche DAO de Membre-service (1/2)

```
public interface MemberRepository extends JpaRepository<Membre, Long>
{
    Membre findByCin(String cin);
    List<Membre>findByNomStartingWith(String caractere);
    Membre findByEmail(String email);
}
```

```
public interface EnseignantChercheurRepository extends
JpaRepository<EnseignantChercheur, Long> {

List<EnseignantChercheur>findByGrade(String grade);
List<EnseignantChercheur>findByEtablissement(String etablissement);
}
```

Création de la couche DAO de Membre-service(2/2)

```
public interface EtudiantRepository extends JpaRepository<Etudiant,  
Long> {  
  
    List<Etudiant>findByDiplome(String diplome);  
    List<Etudiant>findByDiplomeOrderByDateInscriptionDesc(String  
    diplome);  
}
```

Tester la couche DAO de Membre-service (1 / 3)

- Créer et enregistrer deux étudiants
- Créer et enregistrer deux enseignants chercheurs
- Afficher le liste des membres dans le labo
- Chercher un membre par ID
- Modifier un membre
- Supprimer un membre

Tester la couche DAO de Membre-service (2/3)

```
@SpringBootApplication
public class MembreServiceApplication implements CommandLineRunner
{
    @Autowired
    MemberRepository memberRepository;

    public static void main(String[] args) {
        SpringApplication.run(MembreServiceApplication.class, args);
    }

    public void run(String... args) throws Exception {

        // to do
    }
}
```

Injection par champ



Tester la couche DAO de Membre-service (3/3)

Injection par constructeur

```
@SpringBootApplication
```

```
@AllArgsConstructor
```

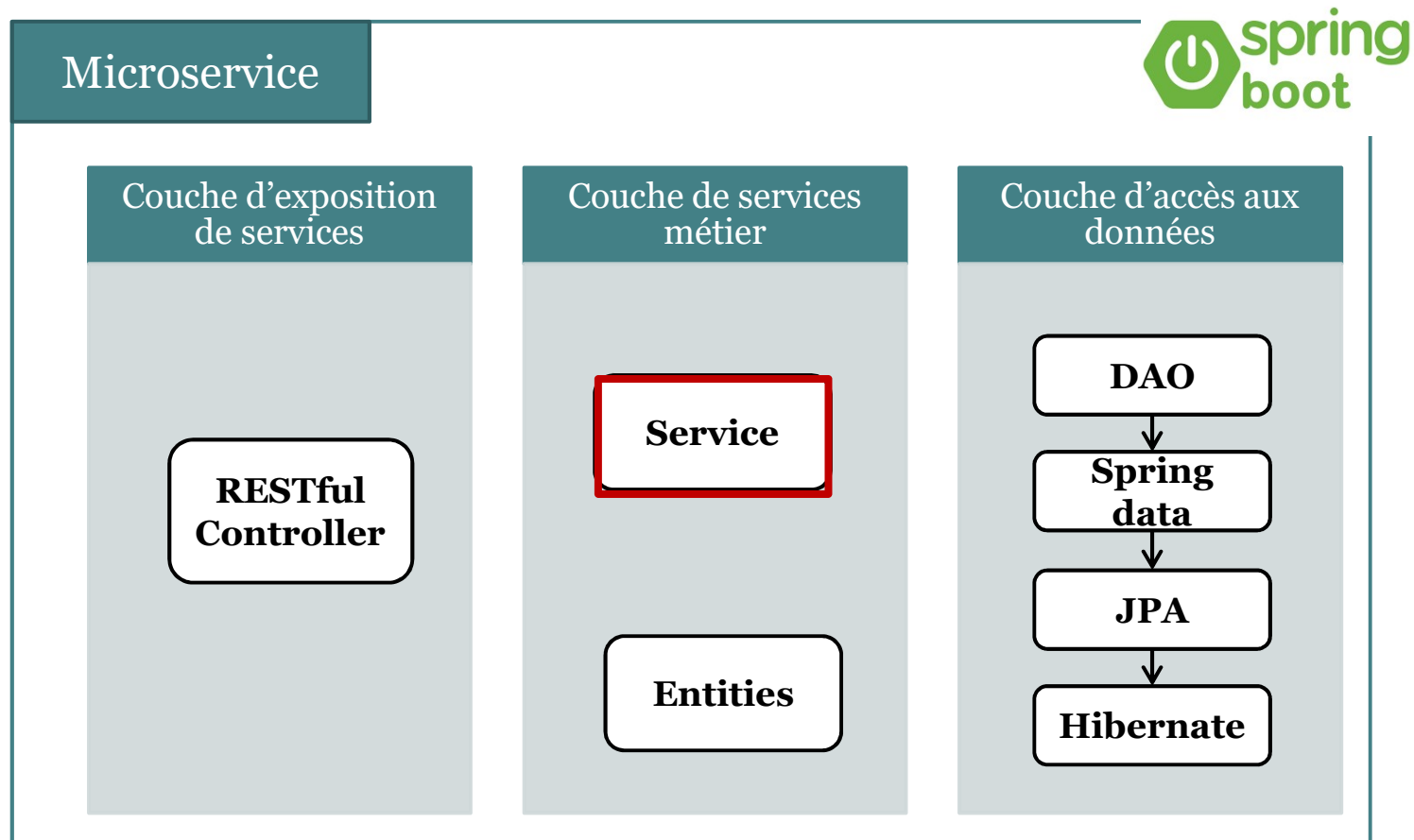
```
public class MembreService2023Application implements CommandLineRunner{  
    MembreRepository membreRepository;
```

```
    public static void main(String[] args) {  
        SpringApplication.run(MembreService2023Application.class, args);  
    }  
    public void run(String... args) throws Exception {
```

```
        // to do  
    }
```



Structure d'un microservice



Implémentation de la couche métier

(1/2)

- Définir des interfaces contenant la spécification de la partie métier d'une application
- Définir des classes implémentant ces interfaces et utilisant l'annotation **@Service**
- Cette annotation est une forme spécialisée de l'annotation @Component destinée à être utilisée dans la couche de service.

Implémentation de la couche métier (2/2)

- **Exemple:**

- Dans le microservice membre-service :
 - Définir l'interface **IMemberService** ainsi que son implémentation
- Dans le microservice Publication-service :
 - Définir l'interface **IPublicationService** ainsi que son implémentation
- Dans le microservices Evenement-service :
 - Définir l'interface **IEventService** ainsi que son implémentation
- Dans le microservices Outil-service :
 - Définir l'interface **IToolService** ainsi que son implémentation

Implémentation de la couche métier:

IMemberService (1/3)

```
public interface IMemberService {  
    //Crud sur les membres  
    public Membre addMember(Membre m);  
    public void deleteMember(Long id) ;  
    public Membre updateMember(Membre p) ;  
    public Membre findMember(Long id) ;  
    public List<Membre> findAll();  
    //Filtrage par propriété  
    public Membre findByCin(String cin);  
    public Membre findByEmail(String email);  
    public List<Membre> findByNom(String nom);  
    //recherche spécifique des étudiants  
    public List<Etudiant> findByDiplome(String diplome);  
    //recherche spécifique des enseignants  
    public List<EnseignantChercheur> findByGrade(String grade);  
    public List<EnseignantChercheur> findByEtablissement(String  
    etablisement);  
    //other ...  
}
```

Implémentation de la couche métier

IMemberService (2/3)

```
@Service
public class MemberImpl implements IMemberService {
    @Autowired
    MemberRepository memberRepository;
    public Membre addMember(Membre m) {
        memberRepository.save(m);
        return m;
    }
    public void deleteMember(Long id) {
        memberRepository.deleteById(id);
    }
    public Membre updateMember(Membre m) {
        return memberRepository.saveAndFlush(m);
    }
    public Membre findMember(Long id) {
        Membre m= (Membre)memberRepository.findById(id).get();
        return m;
    }
    //
```

Implémentation de la couche métier :

IMemberService (3/3)

```
public List<Membre> findAll() {  
    return memberRepository.findAll();  
}  
public Membre findByCin(String cin) {  
    return memberRepository.findByCin(cin);  
}  
public Membre findByEmail(String email) {  
    return memberRepository.findByEmail(email);  
}  
public List<Membre> findByNom(String nom) {  
    return memberRepository.findByNom(nom);  
}  
public List<Etudiant> findByDiplome(String diplome) {  
    return etudiantRepository.findByDiplome(diplome);  
}  
}  
public List<EnseignantChercheur> findByGrade(String grade) {  
    return enseignantChercheurRepository.findByGrade(grade);  
}  
public List<EnseignantChercheur> findByEtablissement(String  
    etablisement) {  
    return  
    enseignantChercheurRepository.findByEtablissement(etablisement);  
}}
```


Tester la couche métier

```
@Autowired
IMemberService memberService;

public static void main(String[] args) {
    SpringApplication.run(LaboProjectV1Application.class, args);
}

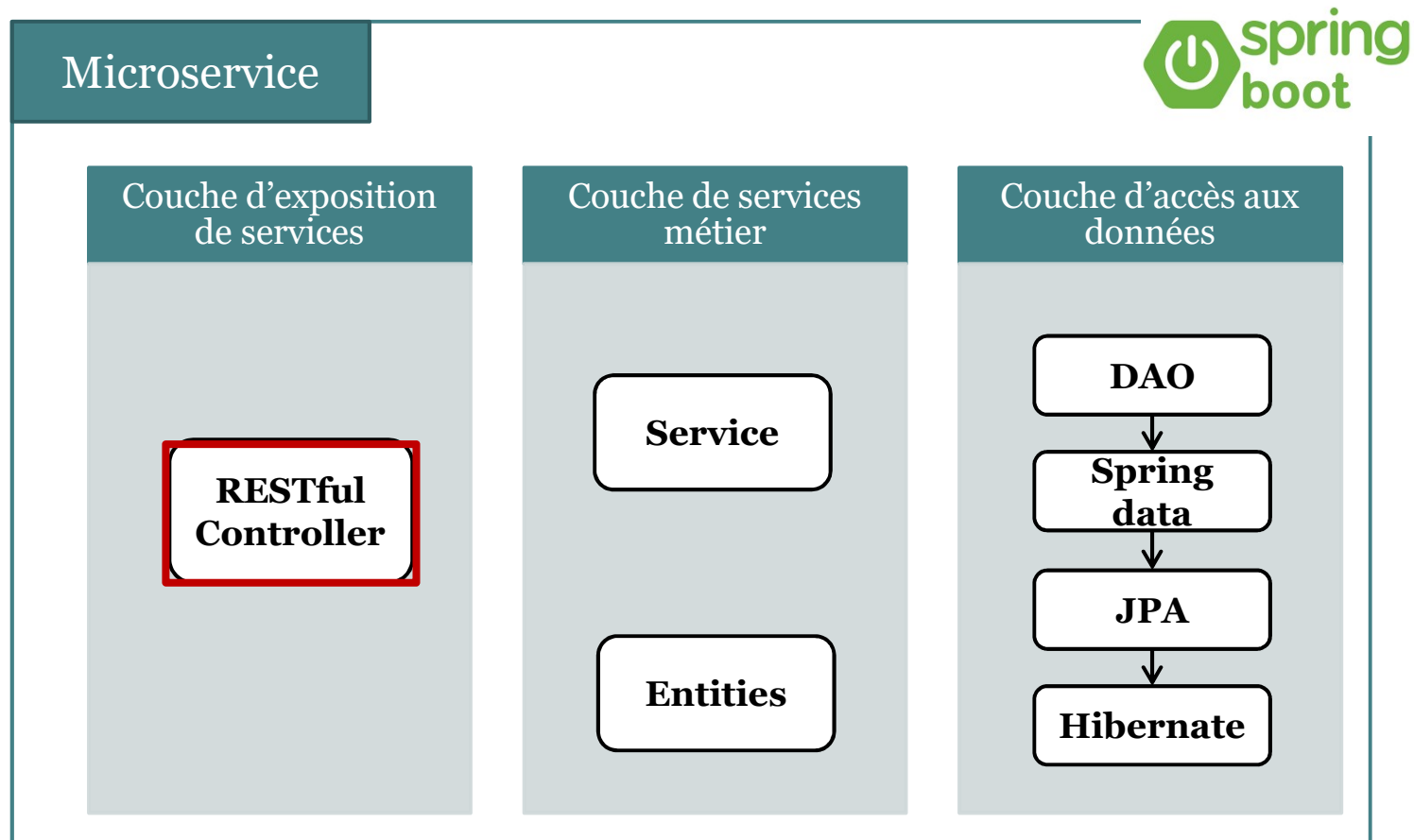
public void run(String... args) throws Exception {
    // Update a Member
    Membre m= memberService.findMember(1L);
    m.setCv("cv1.pdf");
    memberService.updateMember(m);
    // Delete a Member
    memberService.deleteMember(2L);
}
```

Application

- Affecter un étudiant à un enseignant
- Pour un enseignant donné, afficher les étudiants qu'il encadre



Structure d'un microservice



Implémentation de la couche contrôleur (1/9)

- Un service RESTful est une classe Java décorée par l'annotation **@RestController**

```
@RestController  
public class MemberRestController  
{  
  
}
```

- **@RestController** est simplement la combinaison des deux annotations:
 - **@Controller** de Spring qui permet de désigner une classe comme contrôleur, ayant la capacité de traiter les requêtes de type GET, POST, etc.
 - **@ResponseBody** sera ajouté au niveau des méthodes qui devront répondre directement sans passer par une vue.

Implémentation de la couche contrôleur (2/9)

- Nous cherchons à appeler par exemple le micro-service gérant les membres sur les URIs suivantes :
 - Requête **GET** à **/Membres** : affiche la liste de tous les membres;
 - Requête **GET** à **/Membres/{id}** : affiche un membre par son id;
 - Requête **PUT** à
 - **/Membres/etudiant/{id}** : met à jour un étudiant par son Id
 - **/Membres/enseignant/{id}** : met à jour un enseignant par son Id
 - Requête **POST** à
 - **/Membres/etudiant** : ajoute un étudiant
 - **/Membres/enseignant**: ajoute un enseignant
 - Requête **DELETE** à **/Membres/{id}** : supprime un membre par son Id.

Implémentation de la couche contrôleur (3/9)

```
@RestController
public class MemberRestController {
    @Autowired
    IMemberService memberService;
    @RequestMapping(value="/membres", method=RequestMethod.GET)
    public List<Membre> findMembres (){
        return memberService.findAll();
    }
    @GetMapping(value="/membres/{id}")
    public Membre findOneMemberById(@PathVariable Long id){
        return memberService.findMember(id);
    }
}
```

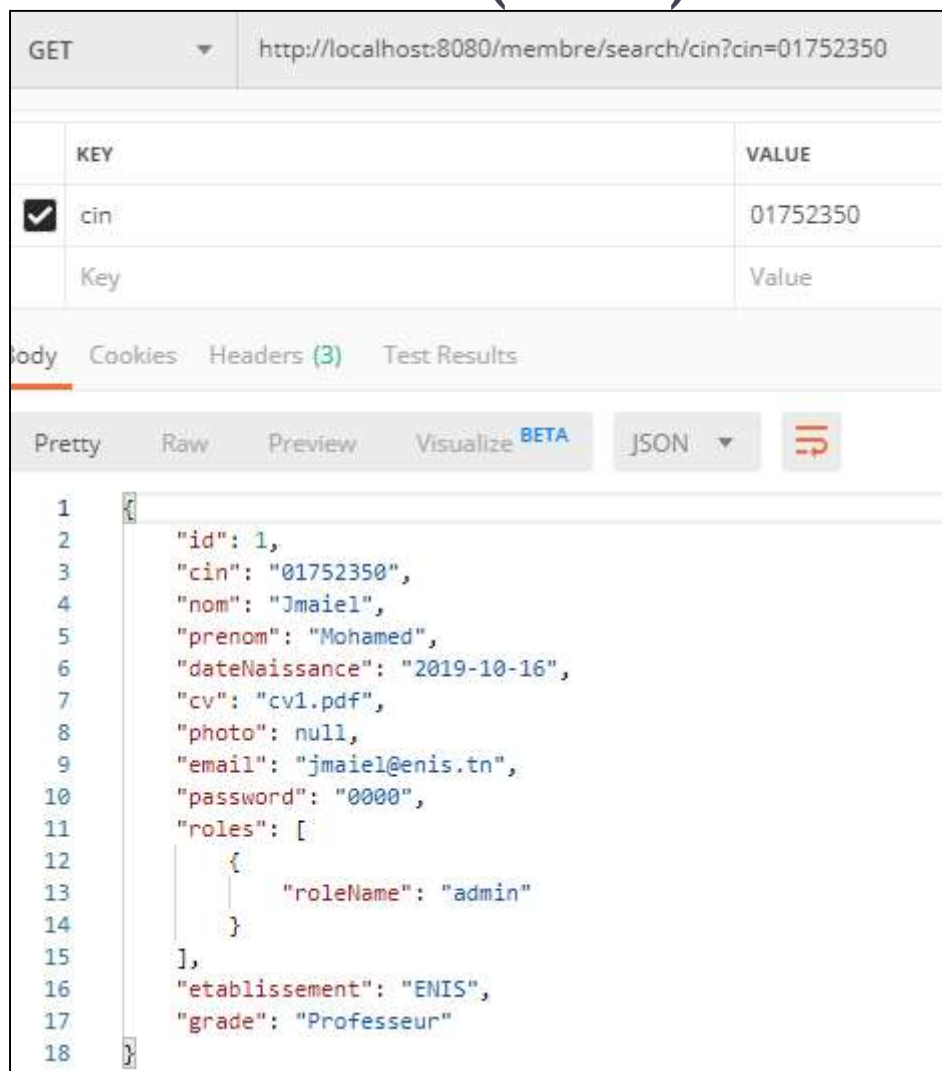
- **@RequestMapping** ou bien **@GetMapping** permet de faire le lien entre l'URI `/membres`, invoquée via GET, et la méthode `findMembres()`
- **@PathVariable** Cette annotation est utilisée pour annoter les arguments de la méthode du gestionnaire de requêtes.
 - Cette méthode doit répondre uniquement aux requêtes avec une URI de type `/Membre/2` par exemple.

Implémentation de la couche contrôleur (4/9)

```
@GetMapping(value="/membres/search/cin")
public Membre findOneMemberByCin(@RequestParam String cin)
{
    return memberService.findByCin(cin);
}
@GetMapping(value="/membres/search/email")
public Membre findOneMemberByEmail(@RequestParam String email)
{
    return memberService.findByEmail(email);
}
```

- **@RequestParam** est utilisée pour lier les paramètres de requête à un paramètre de méthode dans le contrôleur.
 - <http://localhost:8080/membres/search/cin?cin=01752350>
 - http://localhost:8080/membres/search/email?email=rim_a@enis.rn

Implémentation de la couche contrôleur (5/9)



GET http://localhost:8080/membre/search/cin?cin=01752350

KEY	VALUE
<input checked="" type="checkbox"/> cin	01752350
Key	Value

body Cookies Headers (3) Test Results

Pretty Raw Preview Visualize BETA JSON

```
1 {
2   "id": 1,
3   "cin": "01752350",
4   "nom": "Jmaiel",
5   "prenom": "Mohamed",
6   "dateNaissance": "2019-10-16",
7   "cv": "cv1.pdf",
8   "photo": null,
9   "email": "jmaiel@enis.tn",
10  "password": "0000",
11  "roles": [
12    {
13      "roleName": "admin"
14    }
15  ],
16  "etablissement": "ENIS",
17  "grade": "Professeur"
18 }
```

**Tester l'URL soit
via un simple
navigateur soit via
Postman**

Implémentation de la couche contrôleur (6/9)

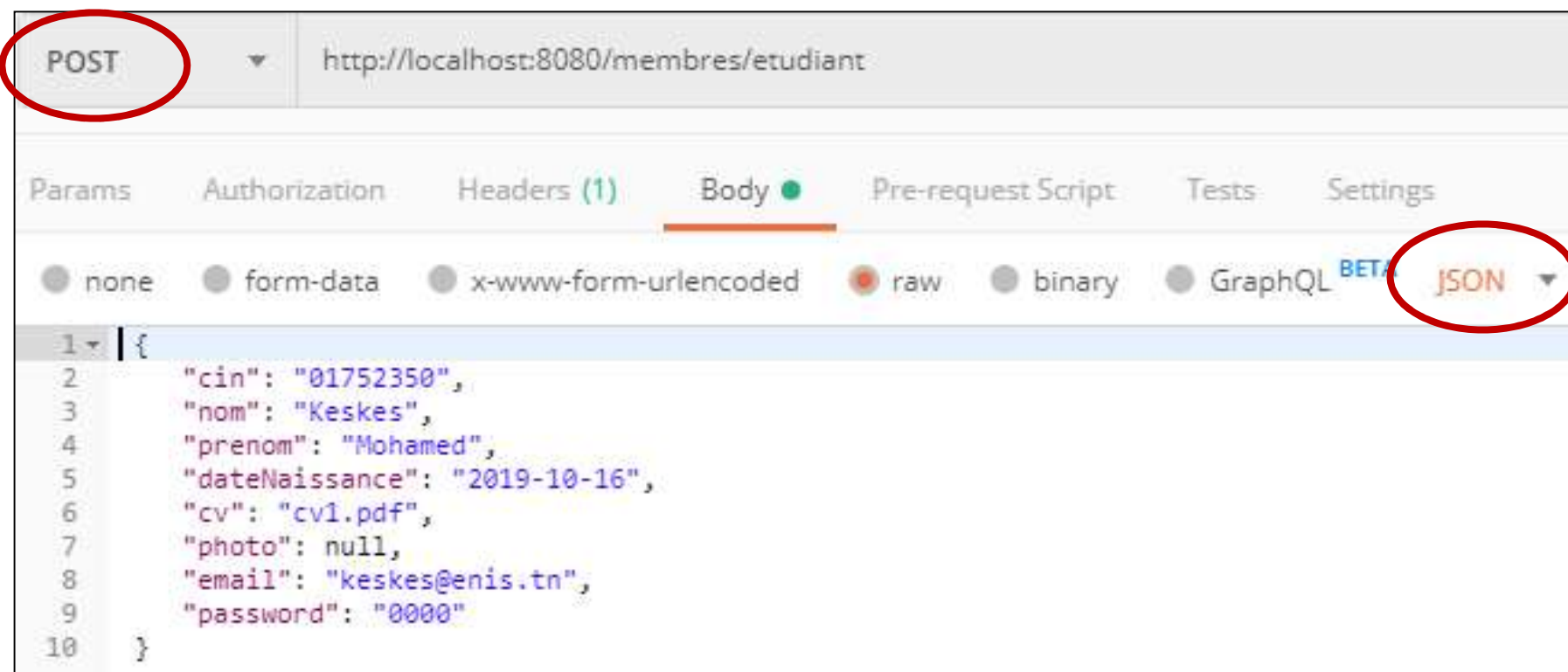
```
@PostMapping(value="/membres/enseignant")
public Membre addMembre(@RequestBody EnseignantChercheur m)
{
    return memberService.addMember(m);
}
@PostMapping(value="/membres/etudiant")
public Membre addMembre(@RequestBody Etudiant e)
{
    return memberService.addMember(e);
}
```



Ajouter

- **@RequestBody** : indique qu'un paramètre de méthode doit être lié à la valeur du corps de la requête HTTP.
- Cette annotation demande à Spring que le JSON contenu dans le corps de la requête sera transformé en Objet

Implémentation de la couche contrôleur (7/9)



Implémentation de la couche contrôleur (8/9)

```
@DeleteMapping(value="/membres/{id}")  
public void deleteMembre(@PathVariable Long id)  
{  
    memberService.deleteMember(id);  
}
```



Supprimer

DELETE ▼

http://localhost:8080/membres/19

Implémentation de la couche contrôleur (9/9)

```
@PutMapping(value="/membres/etudiant/{id}")
public Membre updatemembre(@PathVariable Long id, @RequestBody
Etudiant p)
{
    p.setId(id);
    return memberService.updateMember(p);
}

@PutMapping(value="/membres/enseignant/{id}")
public Membre updateMembre(@PathVariable Long id, @RequestBody
EnseignantChercheur p)
{
    p.setId(id);
    return memberService.updateMember(p);
}
```



Modifier

Utilisation de l'annotation

@CrossOrigin au niveau du contrôleur

- Cette annotation est utilisée à la fois au niveau de la classe et de la méthode pour permettre les demandes lorsque l'hôte qui sert JavaScript (front-end) sera différent de l'hôte qui sert les données (back-end).
- Par défaut, l'annotation **@CrossOrigin** autorise toutes les origines, tous les en-têtes, les méthodes HTTP spécifiées dans l'annotation @RequestMapping et une valeur maxAge de 30 min.

Spring Data Rest pour des projets simples (1 / 3)

- Pour que notre repository soit un repository REST, il faut ajouter:
 - La dépendance 
 - l'annotation **@RepositoryRestResource**

```
//exposer les méthodes comme API REST
@RepositoryRestController
public interface EtudiantRepository extends JpaRepository<Etudiant,
Long> {

    List<Etudiant>findByDiplome(String diplome);
    List<Etudiant>findByNomStartingWith(String carctere);
    List<Etudiant>findByDiplomeOrderByDateInscriptionDesc(String diplome);

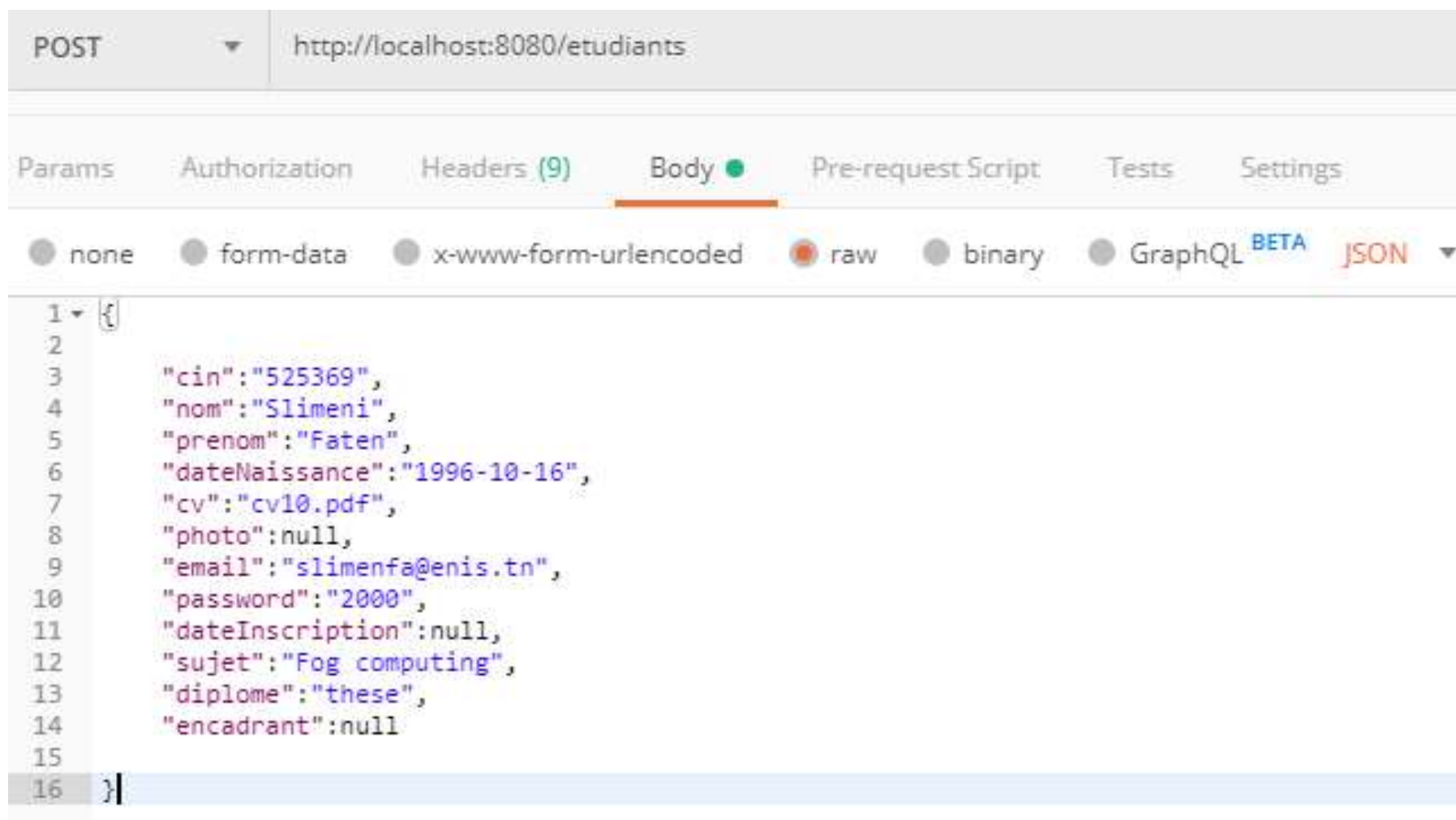
}
```

Spring Data Rest pour des projets simples (2/3)

Voici quelques exemples d'URLs de recherche

- <http://localhost:8080/etudiants/>
- <http://localhost:8080/etudiants/10>
- <http://localhost:8080/etudiants/search/findByDiplome?diplome=these>
- <http://localhost:8080/etudiants/search/findByDiplomeOrderByDateInscriptionDesc?diplome=these>

Spring Data Rest pour des projets simples (3/3)



Tutoriels à voir

- Spring Data Rest :
 - <https://www.codeflow.site/fr/article/spring-data-rest-intro>
 - <https://www.baeldung.com/spring-data-rest-intro>
 - <https://www.springboottutorial.com/spring-boot-introduction-to-spring-data-rest>

Travail à faire

- Terminer la création des microservices
 - Publication-service
 - Evenement-Service
 - Outil-service

