

Objectifs

Développer et déployer une application web d'entreprise en utilisant le Framework Spring.

Acquis d'apprentissage

- **AA1** : Identifier les différentes couches d'une architecture N-tiers
- **AA3** : Construire une application par l'intermédiaire d'un outil de gestion de projet
- **AA4** : Appliquer la notion de l'injection de dépendance
- **AA6** : Evaluer les différentes couches du projet Spring Data JPA
- **AA7** : Développer des services pour la manipulation des données
- **AA8**: Exposer des Web Services REST: Spring MVC REST

ETUDE DE CAS « KADDEM »

« KADDEM », **c'est quoi ?** : C'est un projet qui vise à encourager les jeunes étudiants à améliorer leurs compétences professionnelles notamment dans le cadre des nouvelles tendances du monde de l'informatique.

Objectif : Dans le but de préparer les étudiants aux nouvelles exigences du marché d'emploi, nous proposons de travailler sur le projet « KADDEM ».

Donc concrètement, **qu'est-ce qu'on veut faire ?**

- On désire créer une application de gestion des contrats d'étudiants dans le cadre du projet « KADDEM ».
- Ce projet définit dans chaque département des universités adhérentes un programme de répartition des étudiants par équipes.
- Chaque équipe aura l'un des niveaux (junior, senior ou expert) dans l'une des spécialités suivantes : IA, réseaux, sécurité, cloud.
- Ce projet propose donc aux étudiants des contrats selon la spécialité et le niveau.
- Chaque étudiant obtient un contrat s'il adhère à une équipe, Dans le cas où il sera affecté à plusieurs équipes, il aura un contrat pour chaque activité avec une équipe.
- Le contrat constitue simplement un engagement moral entre l'étudiant et son université ; il doit donc honorer son engagement en participant activement à l'évolution de son équipe.

La figure suivante (*Figure 1*), illustre le diagramme de classes de notre étude de cas.

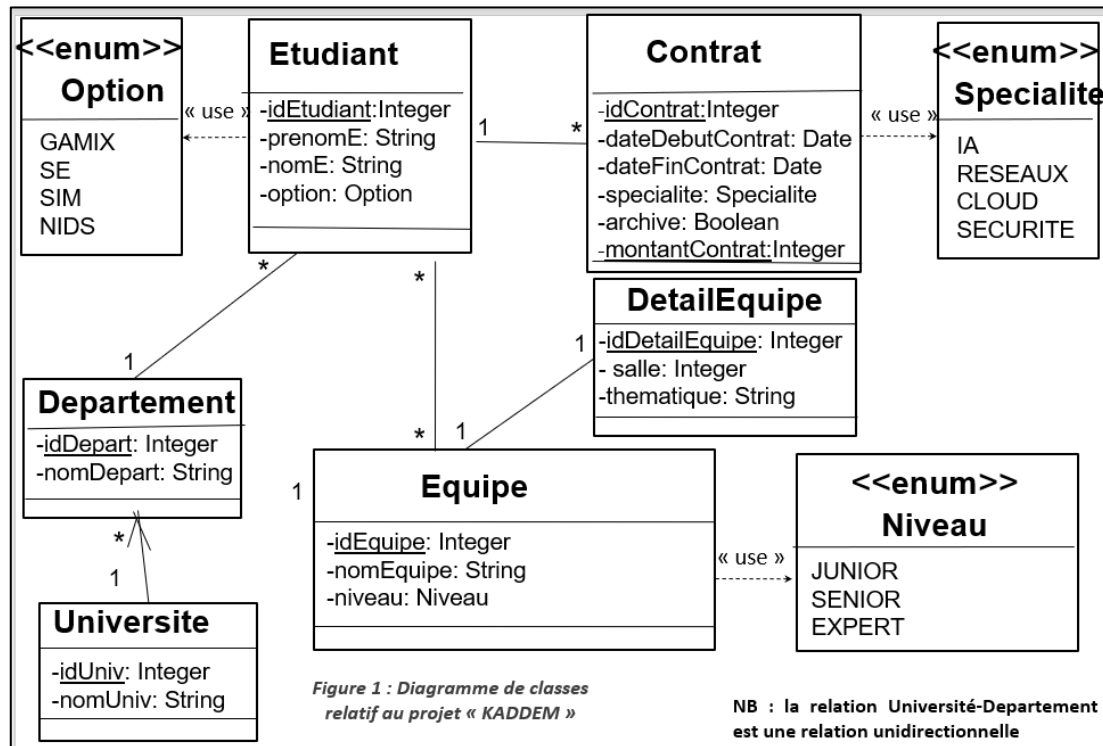


Figure 1. Diagramme de classes Etude de cas "KADDEM"

Partie 1 Spring Data JPA – Les entités et les associations

- Créer les classes et les énumérations suivantes dans le package « **entities** » de votre projet

- ▼ entities
 - Contrat
 - Departement
 - DetailEquipe
 - Equipe
 - Niveau
 - Option
 - Specialite
 - Student
 - Universite

```

public enum Niveau {
    JUNIOR, SENIOR, EXPERT
}

public enum Specialite {
    IA, RESEAUX, CLOUD, SECURITE
}

public enum Option {
    GAMIX, SE, SIM, NIDS
}
  
```

2. Ajouter les annotations suivantes dans toutes les classes créées

```
package tn.esprit.kaddemproject.entities;

import com.fasterxml.jackson.annotation.JsonIgnore;
import lombok.*;
import lombok.experimental.FieldDefaults;
import javax.persistence.*;
import java.io.Serializable;
import java.util.Set;

@Getter
@Setter
@ToString
@AllArgsConstructor
@NoArgsConstructor
@FieldDefaults(level = AccessLevel.PRIVATE)
@Entity
```

3. Implémenter les associations correspondantes au diagramme de classes

<pre>@Entity public class Contrat implements Serializable { @Id @GeneratedValue(strategy = GenerationType.IDENTITY) Integer idContrat; @Temporal(TemporalType.DATE) Date dateDebut; @Temporal(TemporalType.DATE) Date dateFin; @Enumerated(EnumType.STRING) Specialite specialite; Boolean archive; Integer montant; 1 usage @ManyToOne Student etudiant; }</pre>	<pre>@Entity public class Student implements Serializable { @Id @GeneratedValue(strategy = GenerationType.IDENTITY) Integer idEtudiant; String prenomEtudiant; String nomEtudiant; @Enumerated(EnumType.STRING) Option option; @JsonIgnore @OneToMany(mappedBy = "etudiant") Set<Contrat> contrats; 1 usage @ManyToOne(cascade = CascadeType.PERSIST) Departement departement; 1 usage @ManyToMany(cascade = CascadeType.PERSIST) @JsonIgnore Set<Equipe> equipes; }</pre>
<pre>@Entity public class Departement implements Serializable { @Id @GeneratedValue(strategy = GenerationType.IDENTITY) Integer idDepart; String nomDepart; @JsonIgnore @OneToMany(mappedBy = "departement", cascade = CascadeType.REMOVE) Set<Student> etudiants; }</pre>	<pre>@Entity public class Equipe implements Serializable { @Id @GeneratedValue(strategy = GenerationType.IDENTITY) Integer idEquipe; String nomEquipe; @ManyToMany(mappedBy = "equipes", cascade = CascadeType.REMOVE) Set<Student> etudiants; @OneToOne(mappedBy = "equipe", cascade = CascadeType.REMOVE) DetailEquipe detailEquipe; }</pre>

@JsonIgnore: permet d'ignorer une propriété ou une liste de propriétés dans l'objet JSON.

<pre> @Entity public class DetailEquipe implements Serializable { @Id @GeneratedValue(strategy = GenerationType.IDENTITY) Integer idDetailEquipe; Integer salle; String thematique; 1 usage @JsonIgnore @OneToOne(cascade = CascadeType.ALL) Equipe equipe; } </pre>	<pre> @Entity public class Universite implements Serializable { @Id @GeneratedValue(strategy = GenerationType.IDENTITY) Integer idUniv; String nomUniv; @OneToMany(cascade = {CascadeType.PERSIST, CascadeType.REMOVE}) Set<Departement> departements; } </pre>
--	--

4. Configurer votre projet dans le fichier application.properties

```

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

### DATABASE ###
spring.datasource.url=jdbc:mysql://localhost:3306/kaddemdb?createDatabaseIf
NotExist=true&useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyD
atetimeCode=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

### WEB ###
spring.mvc.format.date= yyyy-MM-dd

## LOGGING ##
logging.level.root= info
# Logging pattern for the console
logging.pattern.console= %d{yyyy-MM-dd HH:mm:ss} - %-5level- %logger{45}
- %msg %n

```

`server.servlet.context-path` : est un préfixe du chemin URL qui est utilisé pour identifier et différencier différents contextes. (C'est optionnel si on utilise un projet spring boot).

`server.port` : permet de changer le port du serveur Apache Tomcat Embarqué du projet Spring boot (par défaut, l'application sera déployé sur le port 8080)

`spring.datasource.url` : permet de configurer l'accès la base de données (type de serveur « mysql » pour notre cas, son adresse, le port de communication « par défaut : port 3306 » et le nom de la base).

`spring.datasource.username` et `spring.datasource.password` : permet d'indiquer les informations d'identification pour établir la connexion à la base de données.

`spring.jpa.show-sql` : permet d'activer la journalisation des requêtes générées par Spring Data JPA.

`spring.jpa.hibernate.ddl-auto` : prend une enum qui contrôle la génération du schéma d'une manière plus contrôlée.

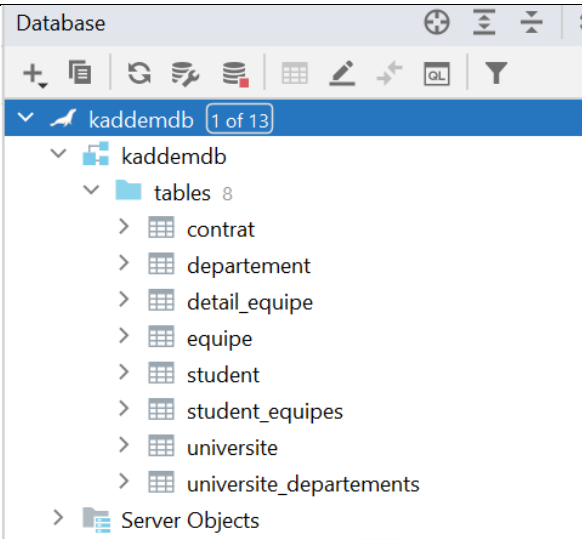
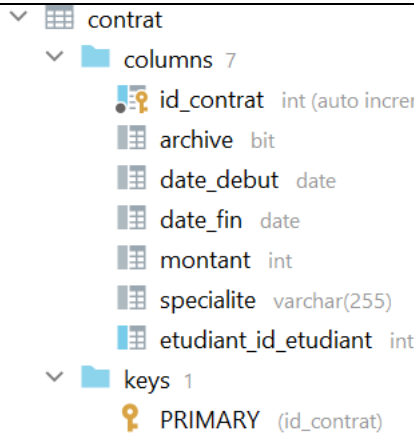
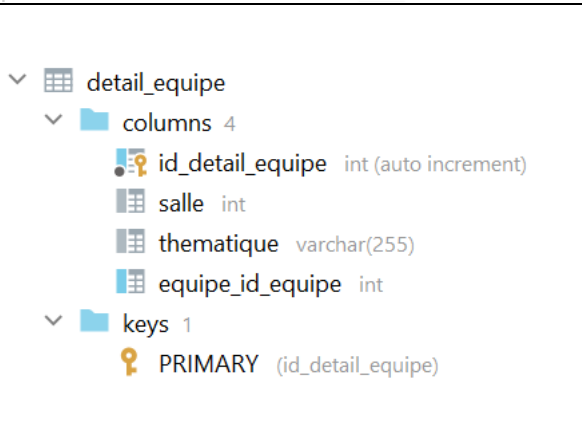
`spring.jpa.properties.hibernate.dialect` : permet à Hibernate de générer du SQL optimisé pour une base de données relationnelle particulière. (*Optionnelle*)

`spring.mvc.format.date` : permet de formater une variable de type String à un type `java.util.Date` dans les requêtes Spring MVC. Sinon vous devez utiliser `@DateTimeFormat(pattern = "yyyy-MM-dd")` Date date avec `@RequestParam` ou `@PathVariable` dans le RestController.

`logging.level.root` : permet de définir le niveau de journalisation dans Spring Boot.

`logging.pattern.console` : permet de personnaliser la mise en forme de journalisation de la console.

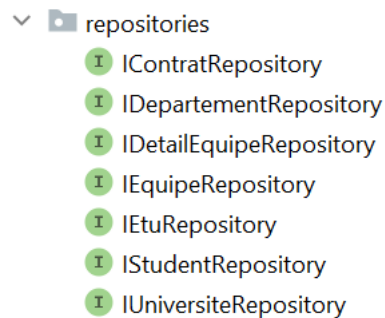
5. Exécuter votre projet pour générer le schéma relationnel dans la base de données (**N.B** : vous devez ouvrir le serveur de base de données MySQL avant)
6. Vérifier les tables générées et les migrations des clés correspondantes :

<p>Le nombre des tables doit être 8 :</p> <ul style="list-style-type: none"> ▪ 6 tables pour chaque entité ▪ 1 table associative générée par relation @OneToMany unidirectionnelle entre les entités Université et Département ▪ 1 table associative générée par relation bidirectionnelle @ManyToMany entre les entités Etudiant et Equipe 	
	

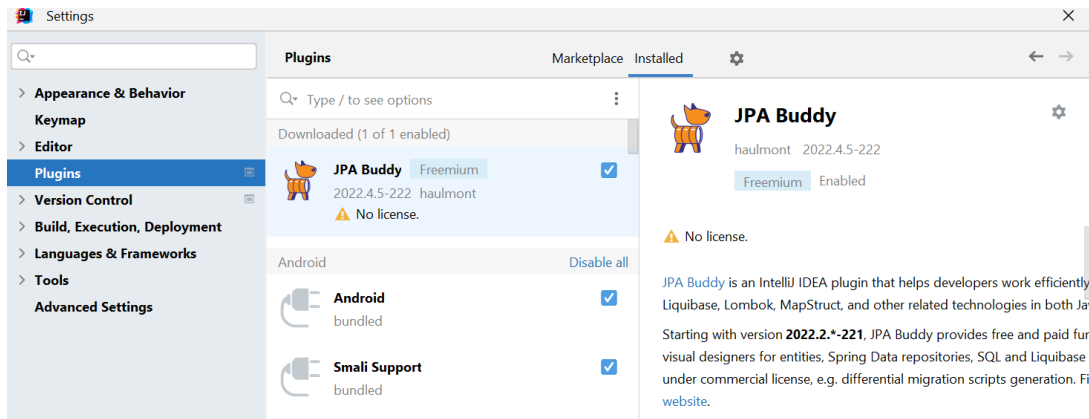
<ul style="list-style-type: none"> ▼ departement <ul style="list-style-type: none"> ▼ columns 2 <ul style="list-style-type: none"> id_depart int (auto increment) nom_depart varchar(255) ▼ keys 1 <ul style="list-style-type: none"> PRIMARY (id_depart) 	<ul style="list-style-type: none"> ▼ student <ul style="list-style-type: none"> ▼ columns 5 <ul style="list-style-type: none"> id_etudiant int (auto increment) nom_etudiant varchar(255) option varchar(255) prenom_etudiant varchar(255) departement_id_depart int ▼ keys 1 <ul style="list-style-type: none"> PRIMARY (id_etudiant)
<ul style="list-style-type: none"> ▼ equipe <ul style="list-style-type: none"> ▼ columns 2 <ul style="list-style-type: none"> id_equipe int (auto increment) nom_equipe varchar(255) ▼ keys 1 <ul style="list-style-type: none"> PRIMARY (id_equipe) 	<ul style="list-style-type: none"> ▼ student_equipes <ul style="list-style-type: none"> ▼ columns 2 <ul style="list-style-type: none"> etudiants_id_etudiant int equipes_id_equipe int ▼ keys 1 <ul style="list-style-type: none"> PRIMARY (etudiants_id_etudiant, equipes_id_equipe)
<ul style="list-style-type: none"> ▼ universite <ul style="list-style-type: none"> ▼ columns 2 <ul style="list-style-type: none"> id_univ int (auto increment) nom_univ varchar(255) ▼ keys 1 <ul style="list-style-type: none"> PRIMARY (id_univ) 	<ul style="list-style-type: none"> ▼ universite_departements <ul style="list-style-type: none"> ▼ columns 2 <ul style="list-style-type: none"> universite_id_univ int departements_id_depart int ▼ keys 2 <ul style="list-style-type: none"> PRIMARY (universite_id_univ, departements_id_depart) UK_aro365ycjablek7s04pu80sa (departements_id_depart)

Partie 2 Spring Data JPA - CRUD Repository (Data Access Layer)

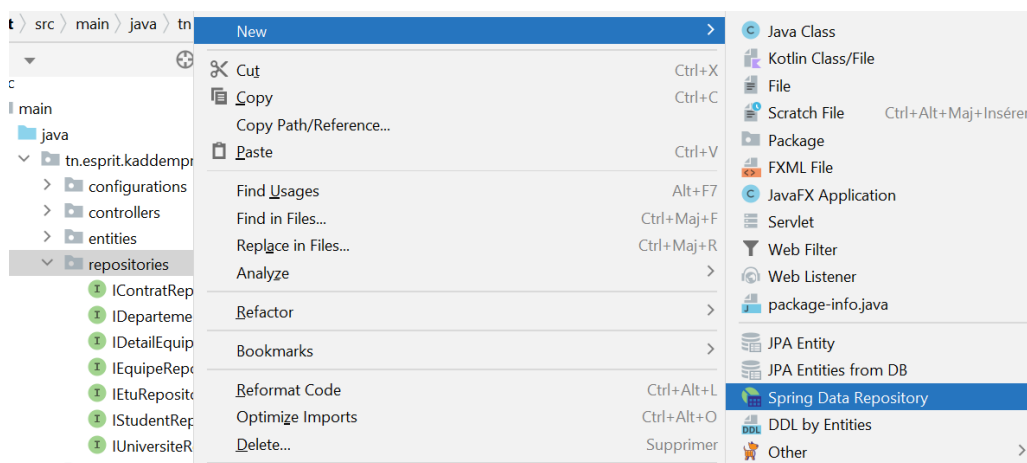
1. Créer les différentes interfaces de la couche d'accès aux données dans le package « **repositories** » de votre projet comme illustré ci-dessous :



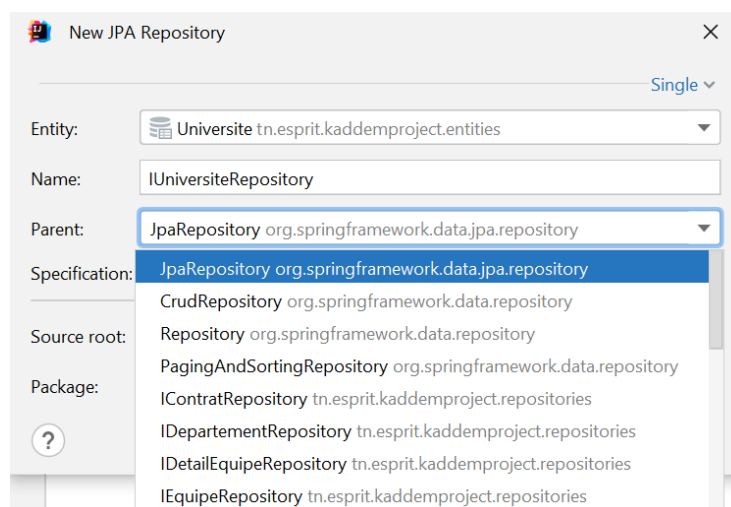
Indication : Vous pouvez installer le plugin **JPA Buddy** pour créer automatiquement les différentes interfaces de Spring Data repositories



- Cliquez droit sur le package repositories > New > Spring Data Repository



- Vous pouvez choisir l'une des interfaces de Spring Data Repositories : CrudRepository, PagingAndSortingRepository ou JpaRepository



- L'interface **IUniversiteRepository** sera créée automatiquement :

```
public interface IUniversiteRepository extends JpaRepository<Universite, Integer> {
}
```

Partie 3 Les services (Business Layer)

1. Créer les différentes interfaces de la couche traitement dans le package « **services** » de votre projet.
2. Définir les **CRUD** des différentes **entités** dans les interfaces associées.
3. Implémenter les différentes classes services de chaque interface créée.

▼ services

- ContratServicesImpl
- DepartementServiceImpl
- EquipeServicesImpl
- IContratServices
- IDepartementServices
- IEquipeServices
- IStudentServices
- IUniversiteServices
- StudentServicesImpl
- UniversiteServicesImpl

- Soit l'interface IUniversiteServices suivante :

```
public interface IUniversiteServices {
    1 usage 1 implementation
    List<Universite> retrieveAllUniversites();
    2 usages 1 implementation
    Universite addOrUpdateUniversite(Universite u);
    1 usage 1 implementation
    Universite retrieveUniversite (Integer idUniversite);
    1 usage 1 implementation
    void removeUniversite(Integer idUniversite);
}
```

- Soit la classe **UniversiteServicesImpl** suivante est l'implémentation de l'interface **IUniversiteServices** :

```
@Service
@RequiredArgsConstructor
public class UniversiteServicesImpl implements IUniversiteServices{
    4 usages
    private final IUniversiteRepository universiteRepository;
    1 usage
    @Override
    public List<Universite> retrieveAllUniversites() { return universiteRepository.findAll(); }
    2 usages
    @Override
    public Universite addOrUpdateUniversite(Universite u) { return universiteRepository.save(u); }
    1 usage
    @Override
    public Universite retrieveUniversite(Integer idUniversite) {
        return universiteRepository.findById(idUniversite).orElse( other: null);
    }
    1 usage
    @Override
    public void removeUniversite(Integer idUniversite) { universiteRepository.deleteById(idUniversite); }
}
```

N.B : On a utilisé dans cet exemple l'**injection de dépendances par constructeur** avec Lombok via l'annotation **@RequiredArgsConstructor**



Attention : Dans le cas où l'attribut *universityRepository* n'est pas *final* (constante), vous devez utiliser l'annotation **@AllArgsConstructor**.

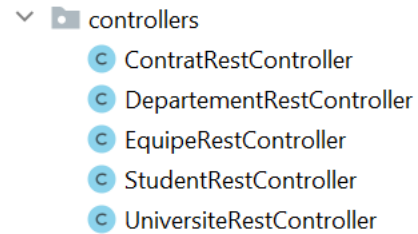


Rappel : Il existe 3 façons pour effectuer l'**injection de dépendances** (autowiring) :

- ✓ Une injection par constructeur
- ✓ Une injection par mutateur (setter)
- ✓ Une injection par attribut / propriété

Partie 3 Spring MVC REST (Presentation Layer)

1. Créer les différentes classes de la couche présentation dans le package « controllers » de votre projet.



2. Dans chaque classe, exposer les différents services en web Services REST. Indiquer la méthode http correspondante (Get, Post, Put, Delete, ...) pour chaque service exposé.

```
@RestController
@RequiredArgsConstructor
@RequestMapping("/university")
public class UniversiteRestController {
    5 usages
    private final IUniversiteServices universiteServices;

    @PostMapping("/add")
    Universite addUniversite(@RequestBody Universite universite){
        return universiteServices.addOrUpdateUniversite(universite);
    }

    @PutMapping("/update")
    Universite updateUniversite(@RequestBody Universite universite){
        return universiteServices.addOrUpdateUniversite(universite);
    }

    @GetMapping("/get/{id}")
    Universite getUniversite(@PathVariable("id") Integer idUniversite){
        return universiteServices.retrieveUniversite(idUniversite);
    }

    @GetMapping("/all")
    List<Universite> getAllUniversites() { return universiteServices.retrieveAllUniversites(); }

    @DeleteMapping("/delete/{id}")
    void deleteUniversite(@PathVariable("id") Integer idUniversite){
        universiteServices.removeUniversite(idUniversite);
    }
}
```

Les URIs ci-dessous correspondent respectivement à l'exposition des services exposés dans la classe UniversiteRestController ci-dessus.

```
###
POST http://localhost:8089/kaddem/university/add

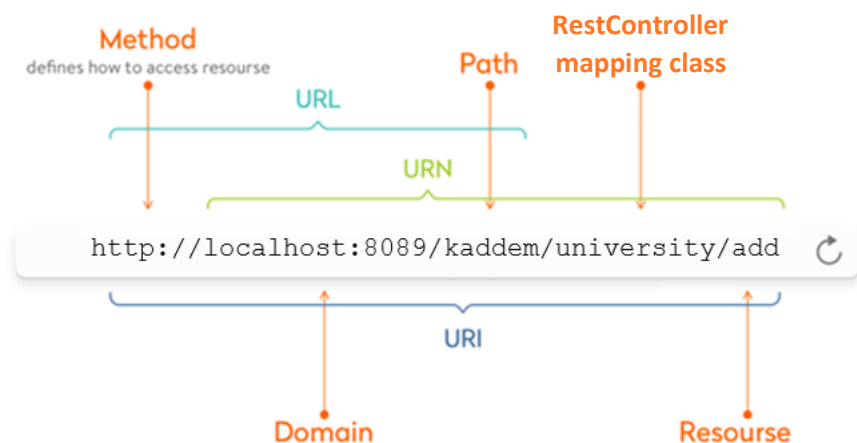
###
PUT http://localhost:8089/kaddem/university/update

###
GET http://localhost:8089/kaddem/university/get/{id}

###
GET http://localhost:8089/kaddem/university/all

###
DELETE http://localhost:8089/kaddem/university/delete/{id}
```

Comprendre le format des requêtes

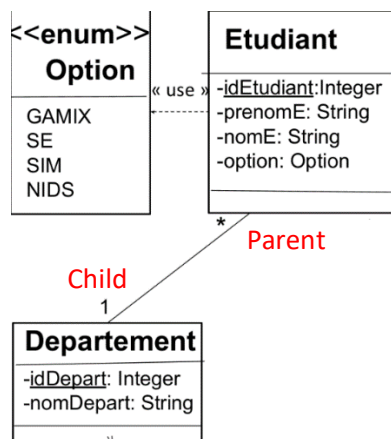


URL : Uniform Resource Locator | **URN** : Uniform Resource Name | **URI** : Uniform Resource Identifier

Affectation d'une relation entre deux entités

L'affectation d'une relation est la mise à jour de l'association dans la base de données relationnelle, soit dans la colonne associée à la clé étrangère, soit dans la table associative selon le type de l'association en question.

Exemple 1 : Le service **assignStudentToDep** ci-dessous, permet d'affecter un étudiant a un département.



L'association entre Etudiant – Departement est bidirectionnelle, alors on doit définir la relation Parent & Child : le bout Parent de cette relation N : 1 est Etudiant, donc il faut rajouter le département a étudiant : *C'est l'objet Etudiant qui va mettre à jour l'association.*

```

@Service
@AllArgsConstructor
public class StudentServicesImp implements IStudentServices {
    12 usages
    private final IStudentRepository studentRepository;
    1 usage
    private final IDepartementRepository departementRepository;

    1 usage
    @Override
    public Student assignStudentToDep(Integer idStudent, Integer idDep) {
        Student student = studentRepository.findById(idStudent).orElse( other: null);
        Departement departement = departementRepository.findById(idDep).orElse( other: null);
        student.setDepartement(departement); //Affectation
        return studentRepository.save(student);
    }
}
  
```

Figure 2. StudentServicesImpl.assignStudentToDep

Exposition du service : assignStudentToDep

```
@Tag(name = "Student Management")
@RestController
@RequestMapping("/student")
@RequiredArgsConstructor
public class StudentRestController {

    14 usages
    private final IStudentServices studentServices;

    @Operation(description = "Assign Student to Departement")
    @PutMapping("/assignStudentDep/{idStudent}/{idDept}")
    public Student assignStudentToDept(@PathVariable("idStudent") Integer idStuent,
                                      @PathVariable("idDept") Integer idDept){
        return studentServices.assignStudentToDep(idStuent, idDept);
    }
}
```

Figure 3. StudentRestController.assignStudentToDept

Test du service : assignStudentToDep

On a ajouté déjà dans la base de données des étudiants et des départements sans affectation.

id_etudiant	nom_etudiant	option	prenom_etudiant	departement_id_depart	id_depart	nom_depart
1	KALLEL	SE	Khaled	NULL	1	TELECOM
					2	INFORMATIQUE

Maintenant, on va affecter l'étudiant « Khaled KALLEL » au département « INFORMATIQUE » via la consommation de l'API REST du service **assignStudentToDep** en utilisant l'URI suivant :

PUT <http://localhost:8089/kaddem/student/assignStudentDep/1/2>

PUT <http://localhost:8089/kaddem/student/assignStudentDep/1/2>

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (5) Test Results 200 OK 342 ms 299 B

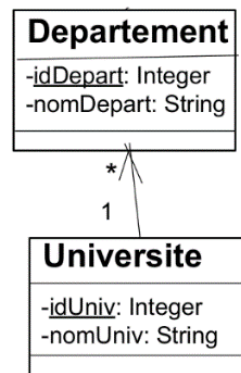
Pretty Raw Preview Visualize JSON

```
1 {
2   "idEtudiant": 1,
3   "prenomEtudiant": "Khaled",
4   "nomEtudiant": "KALLEL",
5   "option": "SE",
6   "departement": {
7     "idDepart": 2,
8     "nomDepart": "INFORMATIQUE"
9   }
10 }
```

La colonne « departement_id_depart » associé à la clé étrangère de la relation Etudiant - Departement est affecté.

id_etudiant	nom_etudiant	option	prenom_etudiant	departement_id_depart
1	KALLEL	SE	Khaled	2

Exemple 2 : Le service **assignUniversiteToDepartement** ci-dessous, permet d'affecter une université a un département. Il s'agit d'une relation unidirectionnelle, alors on a une seule visibilité d'université au département. *Donc, l'affectation sera assurée par l'objet Université.*



N.B : Il n'y a pas de notion Parent & Child dans une relation unidirectionnelle !



Rappel : L'association unidirectionnelle (1 :N) @OneToMany permet de générer une table associative dans la base de données.

```

@Service
@RequiredArgsConstructor
public class UniversiteServicesImpl implements IUniversiteServices{
    6 usages
    private final IUniversiteRepository universiteRepository;
    1 usage
    private final IDepartementRepository departementRepository;

    @Override
    public void assignUniversiteToDepartement(Integer idUniversite, Integer idDepartement) {
        Universite universite = universiteRepository.findById(idUniversite).orElse( other: null);
        Departement departement = departementRepository.findById(idDepartement).orElse( other: null);

        if (universite.getDepartements() == null){
            Set<Departement> departmentSet = new HashSet<>();
            departmentSet.add(departement);
            universite.setDepartements(departmentSet);
        }
        else {
            universite.getDepartements().add(departement);
        }
        universiteRepository.save(universite);
    }
}
  
```

Figure 4. UniversiteServicesImpl.assignUniversiteToDepartement

Exposition du service : assignUniversiteToDepartement

```
@RestController
@RequiredArgsConstructor
@RequestMapping("/university")
public class UniversiteRestController {
    6 usages
    private final IUniversiteServices universiteServices;

    @PutMapping("/assignUnivToDep/{idUniv}/{idDep}")
    void assignUniversiteToDep(@PathVariable("idUniv") Integer idUniv,
                              @PathVariable("idDep") Integer idDep){
        universiteServices.assignUniversiteToDepartement(idUniv,idDep);
    }
}
```

Figure 5. UniversiteRestController.assignUniversiteToDep

Test du service : assignUniversiteToDepartement

On a ajouté déjà dans la base de données des universités et des départements sans affectation.

id_univ	nom_univ	id_depart	nom_depart	SELECT * FROM `universite_departements`
1	ESPRIT	1	TELECOM	universite_id_univ departements_id_depart
2	ENSI	2	INFORMATIQUE	

Maintenant, on va affecter les départements « TELECOM » et « INFORMATIQUE » à l'université « ESPRIT » via la consommation de l'API REST du service **assignUniversiteToDepartement** en utilisant l'URI suivant :

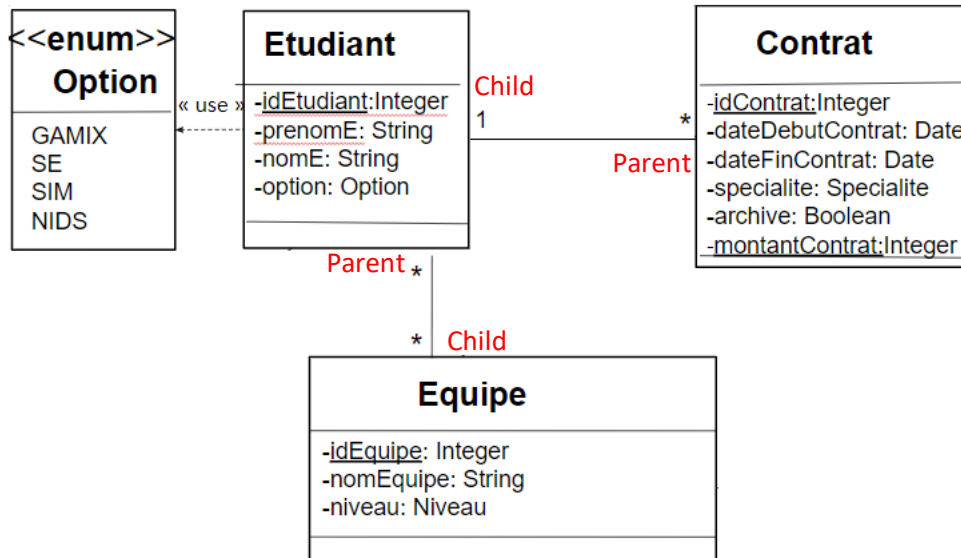
PUT http://localhost:8089/kaddem/university/assignUnivToDep/1/1

PUT	http://localhost:8089/kaddem/university/assignUnivToDep/1/1
PUT	http://localhost:8089/kaddem/university/assignUnivToDep/1/2

L'affectation sera sauvegardée dans la table associative « universite_departements » générée par la relation unidirectionnelle (1 :N) Université - Departement.

universite_id_univ	departements_id_depart
1	1
1	2

Exemple 3 : Le service **addAndAssignEtudiantToContractAndEquipe** ci-dessous, permet d'ajouter un étudiant et l'affecter à la fois à un contrat existant et à une équipe existante.



On doit définir la relation Parent & Child de l'association bidirectionnelle entre Etudiant et Contrat : le bout Parent de cette relation est associé à la cardinalité la plus forte qui est Contrat. *C'est l'objet Contrat qui va mettre à jour la relation.*

De même, on doit définir la relation Parent & Child de l'association bidirectionnelle entre Etudiant et Equipe : On a ici une égalité de cardinalités. Dans ce cas, on a le choix de préciser le bout Parent de cette relation. Ici, *c'est l'objet Etudiant qui va mettre à jour l'association.*

```

@Service
@AllArgsConstructor
public class StudentServicesImp implements IStudentServices {
    12 usages
    private final IStudentRepository studentRepository;
    2 usages
    private final IContratRepository contratRepository;
    1 usage
    private final IEquipeRepository equipeRepository;
    1 usage
    @Override
    @Transactional
    public Student addAndAssignEtudiantToContratAndEquipe(Student e, Integer idContrat, Integer idEquipe) {
        studentRepository.save(e);
        Contrat contrat = contratRepository.findById(idContrat).orElse( other: null);
        Equipe equipe = equipeRepository.findById(idEquipe).orElse( other: null);
        //Affectation avec contrat
        contrat.setEtudiant(e);
        contratRepository.save(contrat);
        //Affectation avec equipe
        Set<Equipe> equipes = new HashSet<>();
        equipes.add(equipe);
        e.setEquipes(equipes);

        return e;
    }
}
    
```

Figure 6. StudentServicesImpl.addAndAssignEtudiantToContratAndEquipe

Exposition du service : addAndAssignEtudiantToEquipeAndContract

```
@Tag(name = "Student Management")
@RestController
@RequestMapping("/student")
@RequiredArgsConstructor
public class StudentRestController {

    14 usages
    private final IStudentServices studentServices;

    @Operation(description = "Add and assign Student To contract and team")
    @PostMapping("/addAndAssignStudent/{idContrat}/{idEquipe}")
    public Student addAndAssignStudentToContratAndEquipe(@RequestBody Student e,
                                                         @PathVariable("idContrat") Integer idContrat,
                                                         @PathVariable("idEquipe") Integer idEquipe){
        return studentServices.addAndAssignEtudiantToContratAndEquipe(e, idContrat, idEquipe);
    }
}
```

Figure 7. StudentRestController.addAndAssignEtudiantToContratAndEquipe

Test du service : addAndAssignEtudiantToEquipeAndContract

On a ajouté déjà dans la base de données des équipes et des contrats sans affectation.

id_contrat	archive	date_debut	date_fin	montant	specialite	etudiant_id_etudiant
1	0	2022-11-15	2023-01-01	200	IA	NULL

id_equipe	niveau	nom_equipe
1	JUNIOR	Wireless com
2	EXPERT	SSD

SELECT * FROM `student_equipes`

etudiants_id_etudiant equipes_id_equipe

Maintenant, on va ajouter l'étudiant « Mohamed ZITOUNI » et on va l'affecter au contrat « 1 » et à l'équipe « SSD » via la consommation de l'API REST du service **addAndAssignEtudiantToEquipeAndContract** en utilisant un client Postman via l'URI suivant :

PUT http://localhost:8089/kaddem/student/addAndAssignStudent/1/2

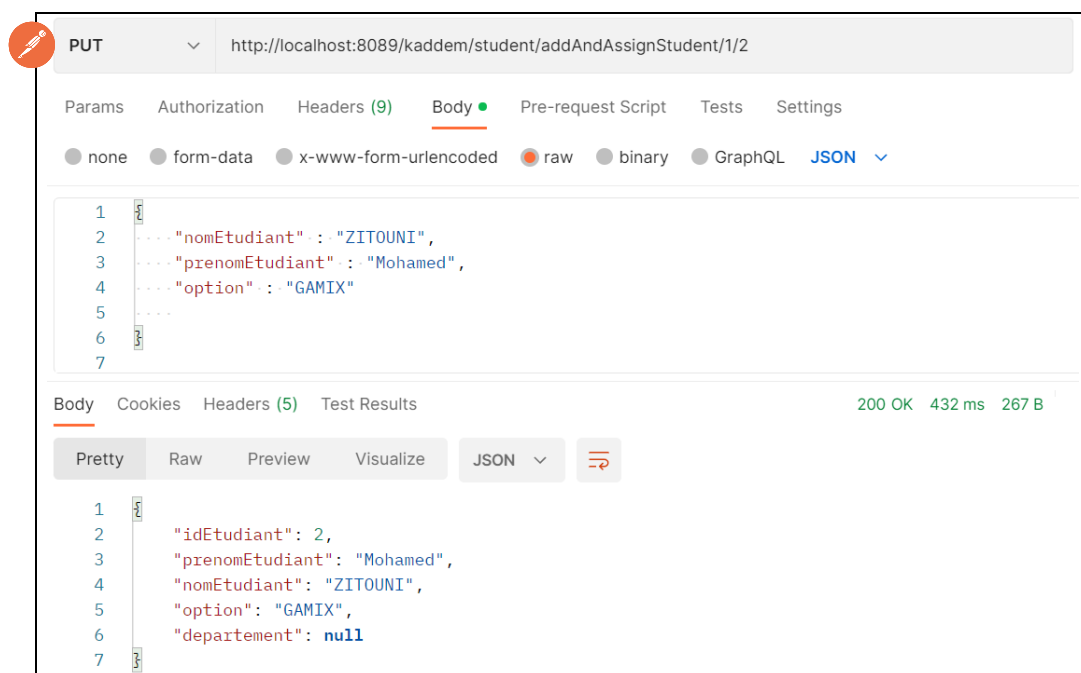


Figure 8. Test service addAndAssignEtudiantToEquipeAndContrat

L'ajout de l'objet Etudiant sauvegardé dans la table « student ».

id_etudiant	nom_etudiant	option	prenom_etudiant	departement_id_depart
1	KALLEL	SE	Khaled	2
2	ZITOUNI	GAMIX	Mohamed	NULL

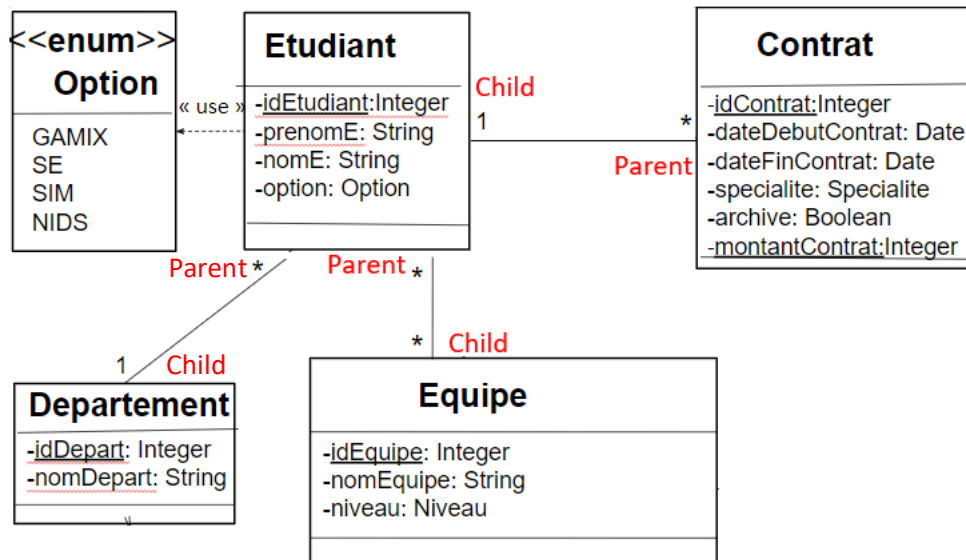
L'affectation de la relation bidirectionnelle Etudiant – Contrat sera sauvegardée dans la table « contrat » au niveau de la colonne « etudiant_id_etudiant » associé à la clé étrangère de la relation.

id_contrat	archive	date_debut	date_fin	montant	specialite	etudiant_id_etudiant
1	0	2022-11-15	2023-01-01	200	IA	2

L'affectation de la relation bidirectionnelle Etudiant – Equipe sera sauvegardée dans table associative « student_equipes » générée par la relation (N:N).

etudiants_id_etudiant	equipes_id_equipe
2	2

Exemple 4 : On va mettre à jour le service **addAndAssignEtudiantToContractAndEquipe** afin d'ajouter un étudiant et l'affecter à la fois à un département existant, à un contrat existant et à une équipe existante.



L'association bidirectionnelle entre Etudiant et Departement : le bout Parent de cette relation est associé à la cardinalité la plus forte qui est Etudiant. *C'est l'objet Etudiant qui va mettre à jour la relation.*

L'association bidirectionnelle entre Etudiant et Contrat : le bout Parent de cette relation est associé à la cardinalité la plus forte qui est Contrat. *C'est l'objet Contrat qui va mettre à jour la relation.*

L'association bidirectionnelle entre Etudiant et Equipe : On a ici une égalité de cardinalités. Dans ce cas, on a choisi le bout Parent de cette relation est Etudiant. *C'est l'objet Etudiant qui va mettre à jour la relation.*

```
@Service
@AllArgsConstructor
public class StudentServicesImp implements IStudentServices {
    12 usages
    private final IStudentRepository studentRepository;
    2 usages
    private final IDepartementRepository departementRepository;
    2 usages
    private final IContratRepository contratRepository;
    1 usage
    private final IEquipeRepository equipeRepository;
    1 usage
}
@Override
@Transactional
public Student addAndAssignEtudiantToContratAndEquipe(Student e, Integer idContrat, Integer idEquipe) {
    //Recupérer le département à partir de l'idDepart envoyé avec l'objet Student
    Departement departement = departementRepository.findById(e.getDepartement().getIdDepart()).orElse( other: null);
    //Affectation avec departement
    e.setDepartement(departement);
    studentRepository.save(e); //Persist l'etudiant avec l'affectation au departement

    Contrat contrat = contratRepository.findById(idContrat).orElse( other: null);
    Equipe equipe = equipeRepository.findById(idEquipe).orElse( other: null);
    //Affectation avec contrat
    contrat.setEtudiant(e);

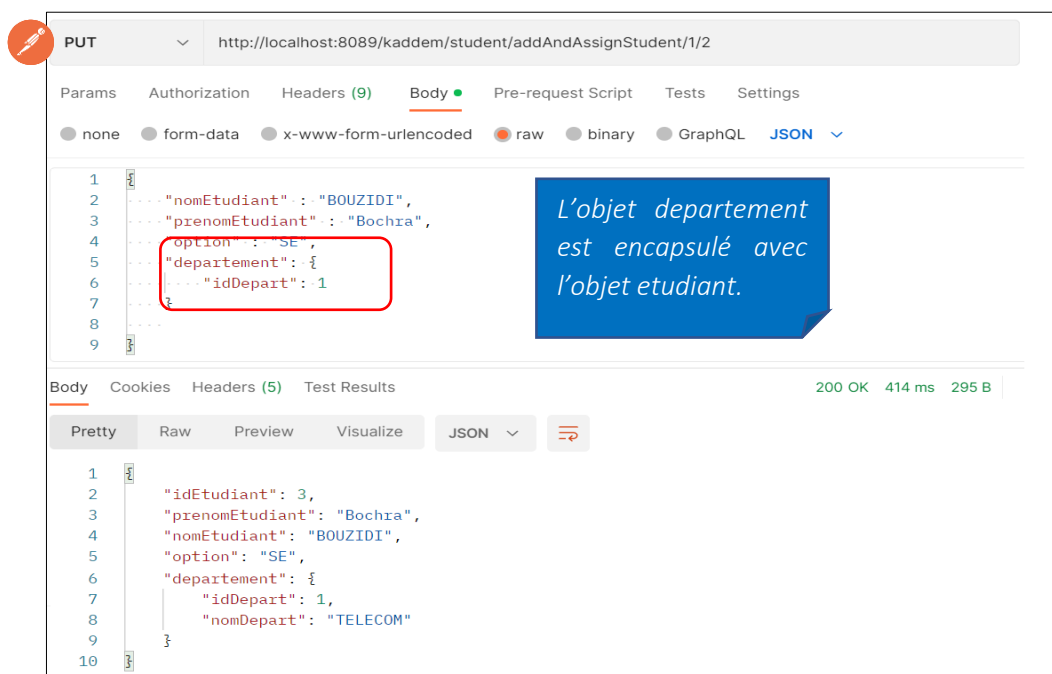
    //Affectation avec equipe
    Set<Equipe> equipes = new HashSet<>();
    equipes.add(equipe);
    e.setEquipes(equipes);

    return e;
}
```

```
@Entity
public class Student implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    Integer idEtudiant;

    ...
    @ManyToOne(cascade = CascadeType.PERSIST)
    Departement departement;
    ...
}
```

Maintenant, on va ajouter l'étudiante « Bochra BOUZIDI » et on va l'affecter au département « TELECOM », au contrat « 1 » et à l'équipe « SSD » via l'API REST du service **addAndAssignEtudiantToEquipeAndContract** :



PUT http://localhost:8089/kaddem/student/addAndAssignStudent/1/2

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "nomEtudiant": "BOUZIDI",
3   "prenomEtudiant": "Bochra",
4   "option": "SE",
5   "departement": {
6     "idDepart": 1
7   }
8 }
9
```

L'objet departement est encapsulé avec l'objet etudiant.

Body Cookies Headers (5) Test Results 200 OK 414 ms 295 B

Pretty Raw Preview Visualize JSON

```
1 {
2   "idEtudiant": 3,
3   "prenomEtudiant": "Bochra",
4   "nomEtudiant": "BOUZIDI",
5   "option": "SE",
6   "departement": {
7     "idDepart": 1,
8     "nomDepart": "TELECOM"
9   }
10 }
```

L'ajout de l'objet Etudiant et l'affectation de la relation bidirectionnelle (N:1) Etudiant – Departement sera sauvegardée dans la table « student ».

id_etudiant	nom_etudiant	option	prenom_etudiant	departement_id_depart
1	KALLEL	SE	Khaled	2
2	ZITOUNI	GAMIX	Mohamed	NULL
3	BOUZIDI	SE	Bochra	1

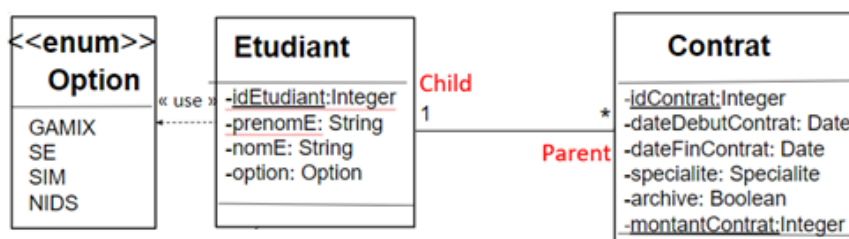
L'affectation de la relation bidirectionnelle (1:N) Etudiant – Contrat sera sauvegardée dans la table « contrat » au niveau de la colonne «etudiant_id_etudiantt » associée à la clé étrangère de la relation.

id_contrat	archive	date_debut	date_fin	montant	specialite	etudiant_id_etudiant
1	0	2022-11-15	2023-01-01	200	IA	3

L'affectation de la relation bidirectionnelle Etudiant – Equipe sera sauvegardée dans la table associative « student_equipes » générée par la relation (N:N).

etudiants_id_etudiant	equipes_id_equipe
2	2
3	2

Exemple 5 : On désire affecter un contrat à un étudiant en vérifiant que l'étudiant n'a pas dépassé la limite autorisée de 5 contrats actifs (archive=false).



Le bout Parent de la relation bidirectionnelle N : 1 Contrat – Etudiant est Contrat. *C'est l'objet Contrat qui va mettre à jour l'association.*

5.1. Définir la méthode qui permet de calculer le nombre des contrats non archivés d'un étudiant dans l'interface *IContratRepository* de la couche d'accès aux données.

```

public interface IContratRepository extends JpaRepository<Contrat, Integer> {
    2 usages
    Integer countByArchiveIsFalseAndEtudiant(Student s);
  
```

5.2. Définir la méthode qui permet de rechercher un étudiant selon son nom et son prénom dans l'interface *IStudentRepository* de la couche d'accès aux données.

```
public interface IStudentRepository extends JpaRepository<Student, Integer> {

    //Solution 1 : Recherche avec les mots clés (keywords) de Spring Data JPA
    1 usage
    Student findByNomEtudiantAndPrenomEtudiant(String nom, String prenom);

    //Solution 2 : Recherche avec une requête JPQL paramétrée
    @Query("select e from Student e where e.nomEtudiant = :nom and e.prenomEtudiant = :prenom")
    Student getStudentByNomAndPrenom(@Param("nom")String nomE, @Param("prenom")String prenomE);
}
```

5.3. Implémenter le service **assignContratToEtudiant** qui d'ajouter un contrat et l'affecter à un étudiant existant selon son nom et son prénom dans la classe *ContratServicesImp*.

```
@Service
@RequiredArgsConstructor
public class ContratServicesImpl implements IContratServices {
    10 usages
    private final IContratRepository contratRepository;
    2 usages
    private final IStudentRepository studentRepository;

    @Override
    public Contrat assignContratToEtudiant(Contrat ce, String nomE, String prenomE) {
        Student student = studentRepository.findByNomEtudiantAndPrenomEtudiant(nomE, prenomE);
        if (contratRepository.countByArchiveIsFalseAndEtudiant(student) < 5) {
            ce.setEtudiant(student); //Affectation contrat à etudiant
            contratRepository.save(ce);
        }
        return ce;
    }
}
```

Exposition du service : assignContratToEtudiant

```
@RestController
@RequestMapping("/contrat")
@RequiredArgsConstructor
public class ContratRestController {
    7 usages
    private final IContratServices contratServices;

    @PutMapping("/assignContratToEtudiant/{nom}/{prenom}")
    public Contrat assignContratToEtudiant(@RequestBody Contrat ce,
                                           @PathVariable("nom") String nomE,
                                           @PathVariable("prenom") String prenomE){
        return contratServices.assignContratToEtudiant(ce, nomE, prenomE);
    }
}
```

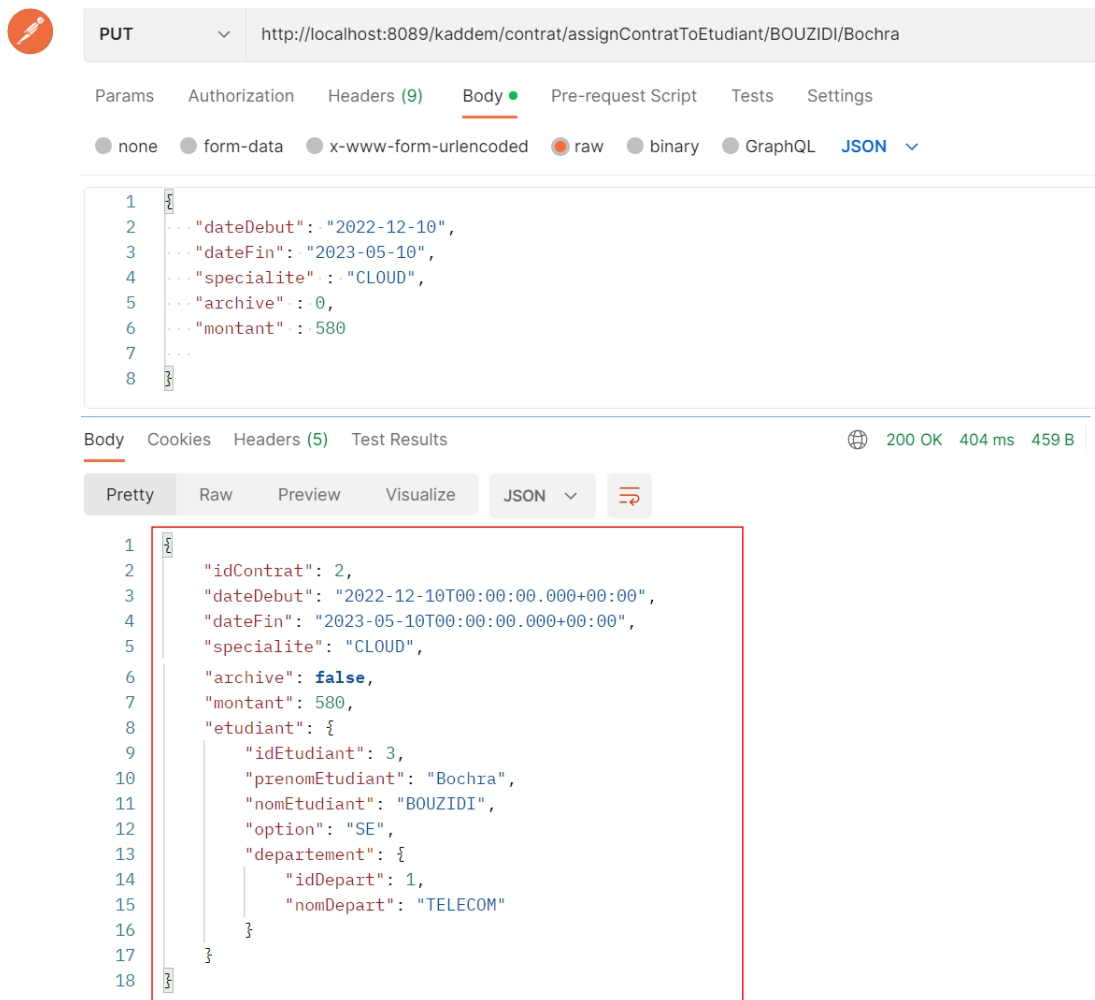
Maintenant, on va ajouter un nouveau contrat et l'affecter à l'étudiante « Bochra BOUZIDI » via l'API REST du service **assignContratToEtudiant** via l'URI suivant :

PUT

http://localhost:8089/kaddem/contrat/assignContratToEtudiant/{nom} / {{prenom}}

Test du service : assignContratToEtudiant

Le contrat sera ajouté et affecté à l'étudiant si ce dernier n'a pas 5 contrats actifs (archive=false).



The screenshot shows a REST client interface with a PUT request to the URL: `http://localhost:8089/kaddem/contrat/assignContratToEtudiant/BOUZIDI/Bochra`. The request body is a JSON object:

```

{
  "dateDebut": "2022-12-10",
  "dateFin": "2023-05-10",
  "specialite": "CLOUD",
  "archive": 0,
  "montant": 580
}

```

The response body is a JSON object:

```

{
  "idContrat": 2,
  "dateDebut": "2022-12-10T00:00:00.000+00:00",
  "dateFin": "2023-05-10T00:00:00.000+00:00",
  "specialite": "CLOUD",
  "archive": false,
  "montant": 580,
  "etudiant": {
    "idEtudiant": 3,
    "prenomEtudiant": "Bochra",
    "nomEtudiant": "BOUZIDI",
    "option": "SE",
    "departement": {
      "idDepart": 1,
      "nomDepart": "TELECOM"
    }
  }
}

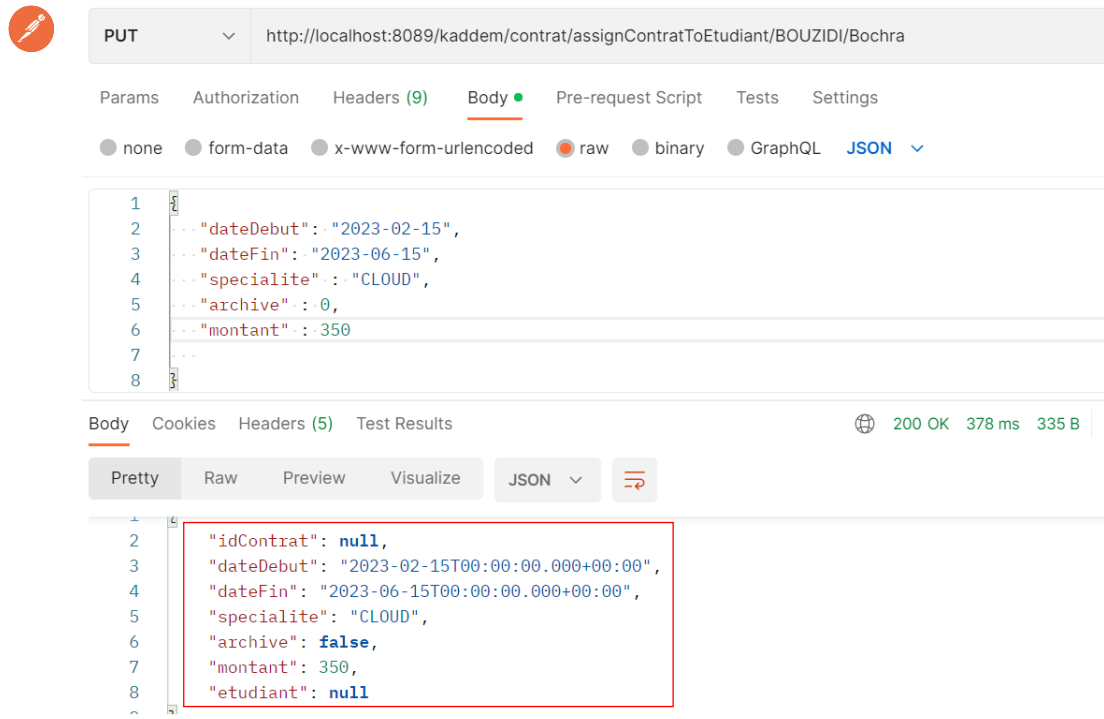
```

Cas 1. L'ajout de l'objet Contrat sera sauvegardé dans la table « contrat ».

id_contrat	archive	date_debut	date_fin	montant	specialite	etudiant_id_etudiant
1	0	2022-11-15	2023-01-01	200	IA	3
2	0	2022-12-10	2023-05-10	580	CLOUD	3

id_etudiant	nom_etudiant	option	prenom_etudiant	departement_id_depart
1	KALLEL	SE	Khaled	2
2	ZITOUNI	GAMIX	Mohamed	NULL
3	BOUZIDI	SE	Bochra	1

Cas 2. L'ajout de l'objet Contrat ne sera pas sauvegardé car l'étudiante « Bochra BOUZIDI » a 5 contrats actifs.



PUT ▼ http://localhost:8089/kaddem/contrat/assignContratToEtudiant/BOUZIDI/Bochra

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON** ▼

```

1  {
2    "dateDebut": "2023-02-15",
3    "dateFin": "2023-06-15",
4    "specialite": "CLOUD",
5    "archive": 0,
6    "montant": 350
7  }
8  }

```

Body Cookies Headers (5) Test Results 200 OK 378 ms 335 B

Pretty Raw Preview Visualize **JSON** ▼

```

1  {
2    "idContrat": null,
3    "dateDebut": "2023-02-15T00:00:00.000+00:00",
4    "dateFin": "2023-06-15T00:00:00.000+00:00",
5    "specialite": "CLOUD",
6    "archive": false,
7    "montant": 350,
8    "etudiant": null
9  }

```

id_contrat	archive	date_debut	date_fin	montant	specialite	etudiant_id_etudiant
1	0	2022-11-15	2023-01-01	200	IA	3
2	0	2022-12-10	2023-05-10	580	CLOUD	3
3	0	2022-12-01	2023-03-01	450	RESEAUX	3
4	0	2022-10-24	2023-10-24	880	SECURITE	3
5	0	2022-11-10	2023-11-10	700	CLOUD	1
6	0	2023-01-01	2023-06-01	630	IA	3

5 contrats
non archivés

Services avancés

Exemple 1 : Le service **getEtudiantsByDepartement** ci-dessous, permet de récupérer les étudiants d'un département donné.

```
@Service
@AllArgsConstructor
public class StudentServicesImp implements IStudentServices {
    12 usages
    private final IStudentRepository studentRepository;

    1 usage
    @Override
    public List<Student> getEtudiantsByDepartement(Integer idDepartement) {
        return studentRepository.getStudentsByDepartement_IdDepart(idDepartement);
    }
}
```

```
public interface IStudentRepository extends JpaRepository<Student, Integer> {

    //Solution 1 : Recherche avec les mots clés (keywords) de Spring Data JPA
    1 usage
    List<Student> getStudentsByDepartement_IdDepart(Integer idDep);

    //Solution 2 : Recherche avec une requête JPQL paramétrée
    1 usage
    @Query("Select e FROM Student e where e.departement.idDepart = :id")
    List<Student> getEtudiantsByDepartement(@Param("id") Integer idDepartement);
}
```

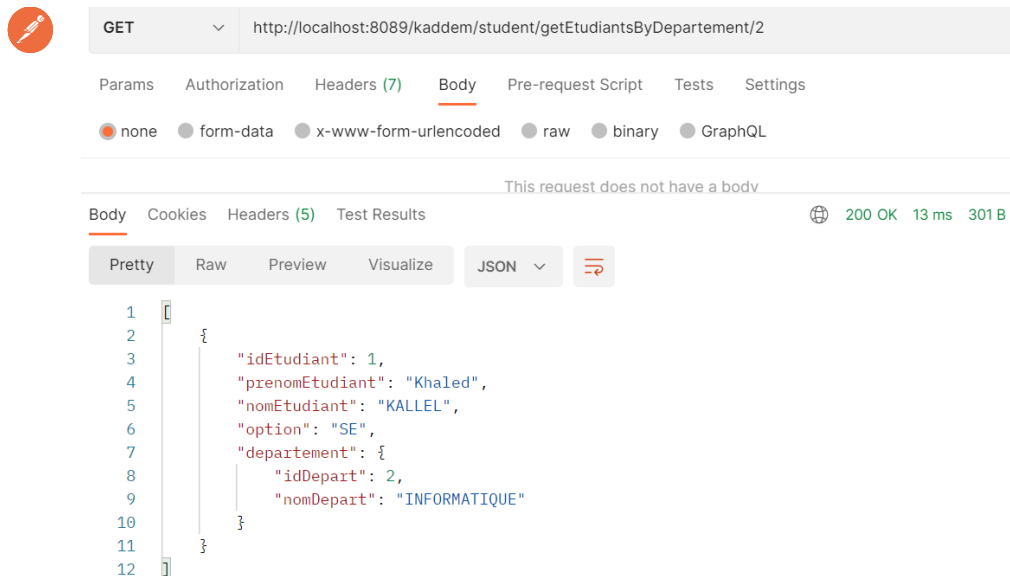
Exposition du service : getEtudiantsByDepartement

```
@Tag(name = "Student Management")
@RestController
@RequestMapping("/student")
@RequiredArgsConstructor
public class StudentRestController {

    14 usages
    private final IStudentServices studentServices;

    @Operation(description = "Retrieve Student by departementId with Keywords")
    @GetMapping("/getEtudiantsByDepartement/{idDep}")
    public List<Student> getEtudiantsByDepartement(@PathVariable("idDep") Integer idDepartement) {
        return studentServices.getEtudiantsByDepartement(idDepartement);
    }
}
```


Test du service : `getEtudiantsByDepartement`



```

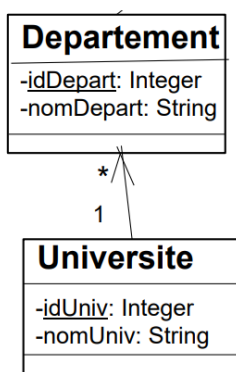
1  [
2    {
3      "idEtudiant": 1,
4      "prenomEtudiant": "Khaled",
5      "nomEtudiant": "KALLEL",
6      "option": "SE",
7      "departement": {
8        "idDepart": 2,
9        "nomDepart": "INFORMATIQUE"
10     }
11   }
12 ]
  
```

Exemple 2 : Le service `addOrUpdateUniversite` ci-dessous, permet d'ajouter à la fois une université et ses départements.

```

@Service
@RequiredArgsConstructor
public class UniversiteServicesImpl implements IUniversiteServices{
    6 usages
    private final IUniversiteRepository universiteRepository;
    2 usages
    @Override
    public Universite addOrUpdateUniversite(Universite u) { return universiteRepository.save(u); }
  
```

Dans ce cas, l'affectation sera assurée par l'objet `Universite` car la relation 1:N `Universite – Departement` est unidirectionnelle.



```

@Entity
public class Universite implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    Integer idUniv;
    String nomUniv;
    @OneToMany(cascade = {CascadeType.PERSIST, CascadeType.REMOVE})
    Set<Departement> departements;
  }
  
```

`CascadeType.PERSIST` permet de persister les départements encapsulés avec l'objet universite au moment de l'ajout. De même, il permet de persister l'affectation automatiquement avec universite.

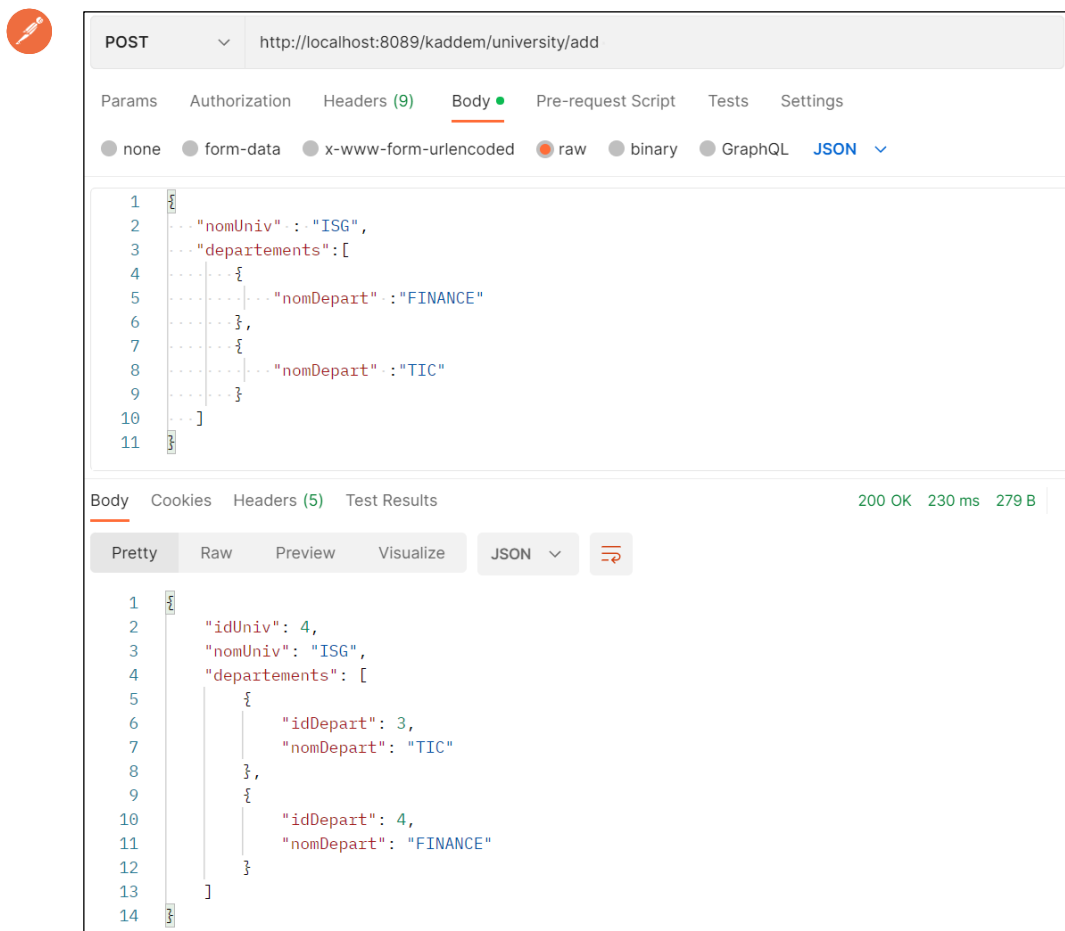
`CascadeType.REMOVE` permet de supprimer en cascade les départements avec l'objet universite au moment de suppression.

Exposition du service : addUniversite

```
@RestController
@RequiredArgsConstructor
@RequestMapping("/university")
public class UniversiteRestController {
    6 usages
    private final IUniversiteServices universiteServices;

    @PostMapping("/add")
    Universite addUniversite(@RequestBody Universite universite){
        return universiteServices.addOrUpdateUniversite(universite);
    }
}
```

Test du service : addUniversite



The screenshot shows a REST client interface with a POST request to `http://localhost:8089/kaddem/university/add`. The request body is a JSON object:

```
{
  "nomUniv": "ISG",
  "departements": [
    {
      "nomDepart": "FINANCE"
    },
    {
      "nomDepart": "TIC"
    }
  ]
}
```

The response is a 200 OK status with a response time of 230 ms and a body size of 279 B. The response body is shown in the 'Body' tab, formatted as JSON:

```
{
  "idUniv": 4,
  "nomUniv": "ISG",
  "departements": [
    {
      "idDepart": 3,
      "nomDepart": "TIC"
    },
    {
      "idDepart": 4,
      "nomDepart": "FINANCE"
    }
  ]
}
```

Exemple 3 : Nous souhaitons calculer le nombre de contrats encore valides entre deux dates. Créer un service permettant de faire le calcul en respectant la signature suivante :

Integer nbContratsValides(Date startDate, Date endDate)

```
@Service
@RequiredArgsConstructor
public class ContratServicesImpl implements IContratServices {
    10 usages
    private final IContratRepository contratRepository;
    1 usage
    @Override
    public Integer nbContratsValides(Date startDate, Date endDate) {
        return contratRepository.countByArchiveIsFalseAndDateDebutBetween(startDate, endDate);
    }
}
```

```
public interface IContratRepository extends JpaRepository<Contrat, Integer> {
    1 usage
    Integer countByArchiveIsFalseAndDateDebutBetween(Date startDate, Date endDate);
}
```

Exposition du service : nbContratsValides

```
@RestController
@RequestMapping("/contrat")
@RequiredArgsConstructor
public class ContratRestController {
    7 usages
    private final IContratServices contratServices;

    @GetMapping("/countByArchive/{d1}/{d2}")
    Integer countContratsValides(@PathVariable("d1") @DateTimeFormat(pattern = "yyyy-MM-dd") Date startDate,
        @PathVariable("d2") @DateTimeFormat(pattern = "yyyy-MM-dd") Date endDate){
        return contratServices.nbContratsValides(startDate, endDate);
    }
}
```

`@DateTimeFormat(pattern = "yyyy-MM-dd")` : permet de convertir la variable d2 de l'URI de type String vers le type Date.
`@DateTimeFormat` n'est pas obligatoire si vous avez configuré la conversion dans le fichier **application.properties** comme suit :

```
spring.mvc.format.date= yyyy-MM-dd
```

Test du service : nbContratsValides

GET http://localhost:8089/kaddem/contrat/countByArchive/2022-01-01/2022-12-31

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (5) Test Results 200 OK 451 ms 165 B

Pretty Raw Preview Visualize JSON

1 5