

# Documentazione backend

Progetto gestione competenze

The logo for Plansoft, featuring the word "plansoft" in a white, lowercase, sans-serif font, centered within a blue, rounded, trapezoidal shape. The shape has a soft drop shadow, giving it a 3D appearance as if it's a card or a block floating above the page.

plansoft

## Sommario

Introduzione .....	3
Schema ER (Entità-Relazione) .....	3
Inizializzazione del progetto .....	4
Applicazione .....	6
Controller .....	6
Service .....	7
Repository .....	7
Entità .....	7
Entità Employee .....	8
Entità Skill .....	13
Entità Level .....	18
Entità Area .....	23
Entità SkillEmployee .....	28

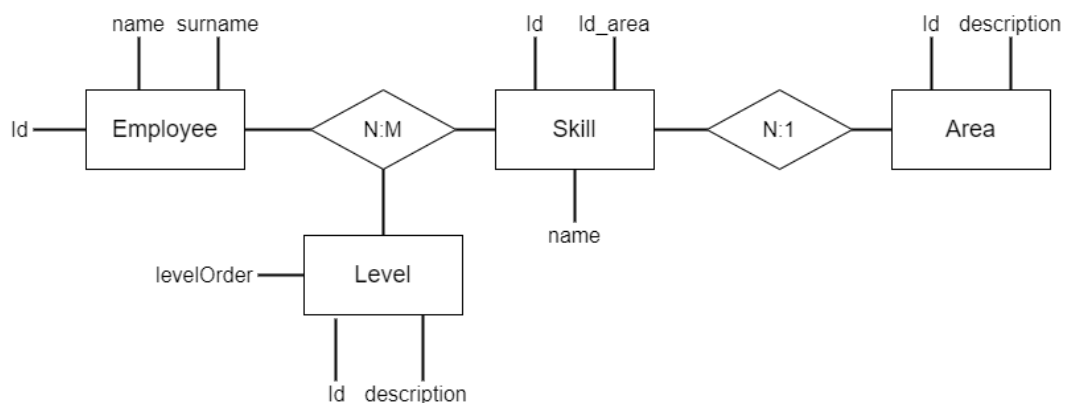
# Introduzione

In questa documentazione verranno descritti gli strumenti sfruttati per la realizzazione del lato backend del progetto “Gestione Competenze”, e, quindi, il codice finale ottenuto. In particolare, sono stati impiegati:

- Linguaggio di programmazione Java
- Spring Boot, un tool che consente di creare applicazioni basate su Spring
- STS (Spring Tool Suite), l’ambiente di sviluppo
- Maven, uno strumento che, tra le tante funzionalità, agevola il programmatore nella gestione delle dipendenze in un progetto Java
- Postman, utilizzato per testare le API

## Schema ER (Entità-Relazione)

Prima di iniziare con la programmazione vera e propria, è stata effettuata un’analisi dei requisiti con lo scopo di realizzare uno schema entità-relazione e, da questo, chiarire quali fossero le entità da realizzare all’interno dell’applicazione stessa (come verrà descritto in seguito). Nella figura sottostante si riporta il suddetto schema:



*Figura 1 Schema ER del progetto Gestione Competenze*

# Inizializzazione del progetto

Esistono vari modi per inizializzare un progetto Spring, ma, nella presente documentazione, verrà spiegato solo quello utilizzato per il progetto “Gestione Competenze”, per evitare di allontanarsi dallo scopo della stessa. Andando sul sito ufficiale di Spring tramite l’indirizzo <https://spring.io/>, si può utilizzare lo strumento Spring Initializr che consente di inizializzare in modo molto semplice un progetto. Come si può vedere in figura 2, infatti,

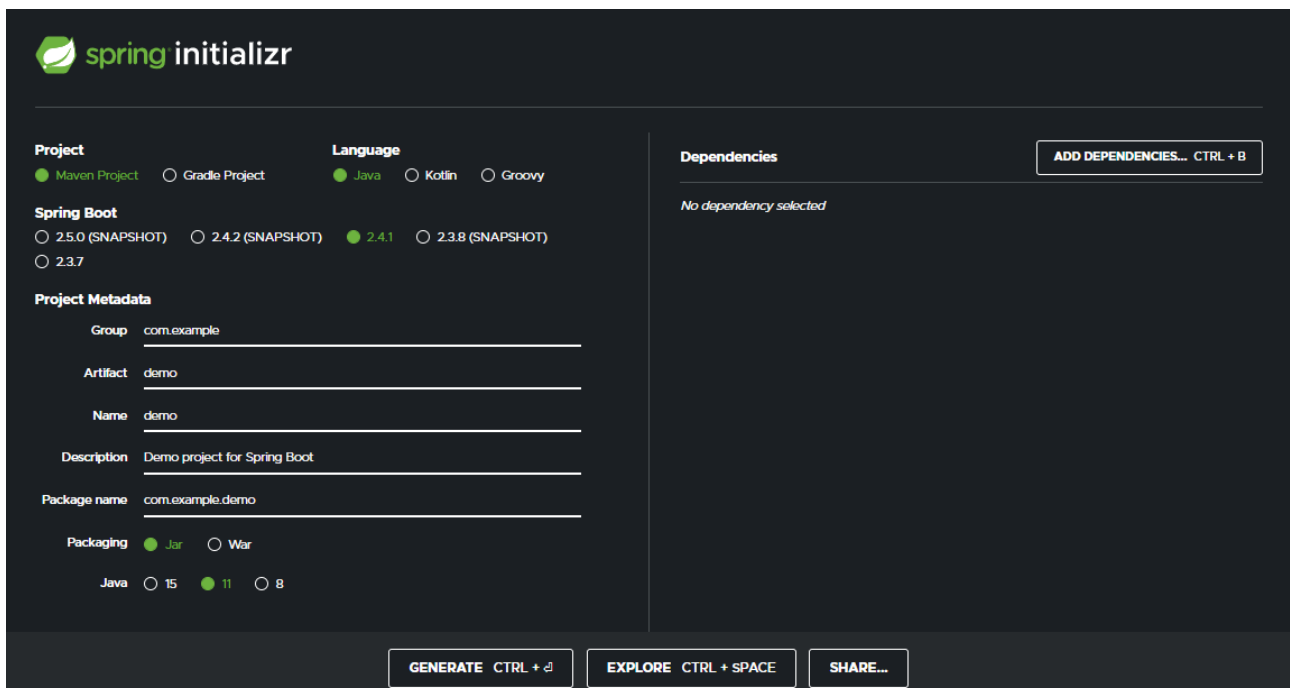
The screenshot shows the Spring Initializr web application interface. At the top left is the 'spring initializr' logo. The interface is divided into two main sections. The left section contains several configuration options: 'Project' with radio buttons for 'Maven Project' (selected) and 'Gradle Project'; 'Language' with radio buttons for 'Java' (selected), 'Kotlin', and 'Groovy'; 'Spring Boot' with radio buttons for versions '2.5.0 (SNAPSHOT)', '2.4.2 (SNAPSHOT)', '2.4.1' (selected), and '2.3.8 (SNAPSHOT)', plus '2.3.7'; 'Project Metadata' with input fields for 'Group' (com.example), 'Artifact' (demo), 'Name' (demo), 'Description' (Demo project for Spring Boot), and 'Package name' (com.example.demo); 'Packaging' with radio buttons for 'Jar' (selected) and 'War'; and 'Java' with radio buttons for versions '15', '11' (selected), and '8'. The right section is titled 'Dependencies' and contains a button 'ADD DEPENDENCIES... CTRL + B' and the text 'No dependency selected'. At the bottom, there are three buttons: 'GENERATE CTRL + G', 'EXPLORE CTRL + SPACE', and 'SHARE...'. The entire interface has a dark theme.

Figura 2 Spring Initialz

sulla sinistra si può decidere che tipo di progetto inizializzare (Maven o Gradel), in quale linguaggio programmare, quale versione di Spring Boot utilizzare, inserire le coordinate del progetto (“Group” e “Artifact”) e altre informazioni, come nome e descrizione. Inoltre, si può scegliere il tipo di packaging e la versione del linguaggio Java. Sulla destra, invece, è possibile aggiungere le dipendenze di cui si ha bisogno all’interno del progetto stesso.

Infine, cliccando sul pulsante “generate” viene scaricato una cartella in formato zip che, dopo l'estrazione, può essere importata come progetto esistente in STS. Tutto ciò che è stato inserito e scelto con Spring Initializr (coordinate del progetto, nome, descrizione, dipendenze) verrà riportato nel file pom.xml (figura 3), facente parte del progetto stesso.

```
<groupId>plansoft</groupId>
<artifactId>gestionecompetenze</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>gestionecompetenze</name>
<description>progetto per gestione competenze</description>
```

*Figura 3 Frammento di codice contenuto nel file pom.xml*

Le dipendenze scelte per questo progetto sono mostrate nella figura sottostante:

```
<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>

  <dependency>
    <groupId>org.apache.derby</groupId>
    <artifactId>derby</artifactId>
    <scope>runtime</scope>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>
```

*Figura 4 Dipendenze utilizzate nel progetto "Gestione Competenze"*

## Applicazione

Dopo aver importato il progetto in STS, possiamo notare che, tra i vari file, Spring Initializr ha creato anche quello della classe Java (SkillsManagmentApplication.java) in cui si trova il metodo main. Questo si occupa di creare il servlet container e lo inizializza per ospitare la nostra applicazione. Inoltre, la suddetta classe viene annotata con la dicitura “@SpringBootApplication”, la quale specifica che proprio SkillsManagmentApplication.java è il punto di partenza dell’applicazione stessa.

```
package plansoft.skillsmanagement;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
public class SkillsManagementApplication {

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

}
```

*Figura 5 Classe SkillsManagmentApplication*

A questo punto è possibile iniziare a realizzare le varie entità del progetto. Per far ciò, è stato creato un nuovo package, per ognuna di esse, che conterrà non solo la classe col nome dell’entità stessa, ma anche un controller, un service e una repository. Prima di passare alla descrizione delle entità, è, quindi, importante comprendere l’utilità di questi ultimi tre elementi.

## Controller

Data una determinata richiesta, il controller consente di far eseguire del codice sul servlet container. Si tratta, sostanzialmente, di una classe Java contrassegnata con l’annotazione @RestController, tramite la quale si

informa Spring su qual è l'URL su cui si sta mappando e cosa deve succedere quando la richiesta è su quell'URL.

## Service

Anche il service è una classe Java, contrassegnata con l'annotazione `@Service`. Generalmente, in Spring i service sono dei Singleton e servono a scrivere la logica di funzionamento dei metodi mappati nel controller.

## Repository

La repository rappresenta quell'elemento che consente al service di connettersi al database ed effettuare le funzioni base come salvare un dato, leggerlo, modificarlo e cancellarlo. Per implementarla, si crea un' interfaccia che estende un'altra interfaccia (da `springframework.data.repository`) chiamata `CrudRepository`, che contiene tutti i metodi base che sono gli stessi per ogni entità. Essa è, tuttavia, un tipo generico, quindi, dovrà essere specificato il tipo di entità sulla quale i metodi funzioneranno e il tipo della chiave primaria dell'entità stessa.

## Entità

Questo paragrafo verrà dedicato ad illustrare la struttura di ogni entità, commentando, di volta in volta, il codice all'interno delle varie classi, che servono a creare l'entità stessa ed a consentire l'interazione con essa.

## Entità Employee

L'entità Employee serve a rappresentare un dipendente. Per gestire tale entità sono state create tre classi e un'interfaccia.

**Employee.java** definisce l'entità stessa, che possiede un id, nome e cognome. La classe è contrassegnata con l'annotazione `@Entity` che specifica, appunto, che quella classe rappresenta un'entità in un database. Essa ha come chiave primaria la variabile id e, per specificare questo, viene usata l'annotazione `@Id`. L'annotazione `@GeneratedValue` serve a specificare, invece, che quell'id si dovrà auto-incrementare tutte le volte che viene creato un nuovo oggetto di tipo Employee.



```

package plansoft.skillsmanagement.employee;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Employee {

    @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;
    private String name;
    private String surname;

    public Employee() {

    }

    public Employee(int id, String name, String surname) {
        this.id = id;
        this.name = name;
        this.surname = surname;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getSurname() {
        return surname;
    }

    public void setSurname(String surname) {
        this.surname = surname;
    }

}

```

*Figura 6 Classe Employee.java*

In essa, come si può vedere dalla figura 6 ci sono:

- metodi setter e getter, per settare e prendere, rispettivamente, i valori di questa entità
- costruttore vuoto
- costruttore con i parametri in ingresso per inizializzare gli attributi di questa entità

**EmployeeController.java** è il controller dell'entità Employee, in cui è definita una sola variabile di tipo EmployeeService. I metodi sono:

- getAllEmployee(), che restituisce una lista di oggetti di tipo Employee
- getEmployee(@PathVariable int id), che restituisce un oggetto di tipo Employee dato un determinato id. L'annotazione @PathVariable serve ad indicare che l'id è quello dato nell'URL.
- addEmployee(@RequestBody Employee employee), che consente di aggiungere un nuovo oggetto di tipo Employee. L'annotazione @RequestBody serve ad indicare che l'oggetto deve essere inviato nel corpo della richiesta (in formato json)
- updateEmployee(@RequestBody Employee employee, @PathVariable int id), che consente di apportare delle modifiche ad un determinato oggetto di tipo Employee, che ha un determinato id. L'oggetto di tipo Employee modificato deve essere inviato nel corpo della richiesta
- deleteEmployee(@PathVariable int id), che cancella l'oggetto di tipo Employee che ha id uguale a quello passato al metodo stesso

```

package plansoft.skillsmanagement.employee;
import java.util.List;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class EmployeeController {

    @Autowired
    private EmployeeService employeeService;

    @RequestMapping("/employees")
    public List<Employee> getAllEmployees() {
        return employeeService.getAllEmployees();
    }

    @RequestMapping("/employees/{id}")
    public Optional<Employee> getEmployee(@PathVariable int id) {
        return employeeService.getEmployee(id);
    }

    @RequestMapping(method = RequestMethod.POST, value = "/employees")
    public void addEmployee(@RequestBody Employee employee) {
        employeeService.addEmployee(employee);
    }

    @RequestMapping(method = RequestMethod.PUT, value = "/employees/{id}")
    public void updateEmployee(@RequestBody Employee employee, @PathVariable int id) {
        employeeService.updateEmployee(employee, id);
    }

    @RequestMapping(method = RequestMethod.DELETE, value = "/employees/{id}")
    public void deleteEmployee(@PathVariable int id) {
        employeeService.deleteEmployee(id);
    }
}

```

*Figura 7 Controller dell'entità Employee*

**EmployeeService.java** è il service dell'entità Employee. La classe ha una sola variabile di tipo EmployeeRepository, sfruttata all'interno dei metodi per chiamare quelli della repository ed eseguire, quindi, le varie richieste. I metodi sono:

- getAllEmployee(), che restituisce una lista di oggetti di tipo Employee
- getEmployee(int id), che restituisce un oggetto di tipo Employee dato un determinato id.

- addEmployee(Employee employee), che consente di aggiungere un nuovo oggetto di tipo Employee.
- updateEmployee(Employee employee, int id), che consente di apportare delle modifiche ad un determinato oggetto di tipo Employee, che ha un determinato id.
- deleteEmployee(int id), che cancella l'oggetto di tipo Employee che ha id uguale a quello passato al metodo stesso

```
package plansoft.skillsmanagement.employee;

import java.util.ArrayList;

@Service
public class EmployeeService {

    @Autowired
    private EmployeeRepository employeeRepository;

    public List<Employee> getAllEmployees() {
        List<Employee> employees = new ArrayList<>();
        employeeRepository.findAll().forEach(employees::add);
        return employees;
    }

    public Optional<Employee> getEmployee(int id) {
        return employeeRepository.findById(id);
    }

    public void addEmployee(Employee employee) {
        employeeRepository.save(employee);
    }

    public void updateEmployee(Employee employee, int id) {
        if (employeeRepository.existsById(id)) {
            employeeRepository.deleteById(id);
            employeeRepository.save(employee);
        }
    }

    public void deleteEmployee(int id) {
        employeeRepository.deleteById(id);
    }
}
```

*Figura 8 Service dell'entità Employee*

**EmployeeRepository.java** è la repository dell'entità Employee. Si tratta di un'interfaccia che estende l'interfaccia CrudRepository da springframework.data.repository, come spiegato nel paragrafo precedente. I tipi specificati questa sono di tipo Employee e Integer.

```
package plansoft.skillsmanagement.employee;

import org.springframework.data.repository.CrudRepository;

public interface EmployeeRepository extends CrudRepository<Employee, Integer> {

}
```

*Figura 9 Repository dell'entità Employee*

Come si può vedere dalla figura 9, la repository dell'entità Employee non ha bisogno della definizione di metodi custom, in quanto i metodi che fornisce l'interfaccia CrudRepository sono sufficienti per soddisfare le varie richieste.

## Entità Skill

L'entità Skill serve a rappresentare una competenza che un dipendente può possedere. Per gestire tale entità sono state create tre classi e un'interfaccia.

**Skill.java** definisce l'entità stessa che possiede un id, nome e un oggetto di tipo Area per specificare a quale area quella competenza appartiene (es. Java è una competenza che appartiene all'area dei linguaggi di programmazione). La classe è contrassegnata con l'annotazione @Entity che specifica, appunto, che quella classe rappresenta un'entità in un database. Essa ha come chiave primaria la variabile id e, per specificare questo, viene usata l'annotazione @Id. L'annotazione @GeneratedValue serve a specificare, invece, che quell'id si dovrà auto-incrementare tutte le volte che viene creato un nuovo oggetto di tipo Skill.

```

package plansoft.skillsmanagement.skill;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToOne;

import plansoft.skillsmanagement.area.Area;

@Entity
public class Skill {

    @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;
    private String name;

    @ManyToOne
    private Area area;

    public Skill() {
    }

    public Skill(int id, String name) {
        this.id = id;
        this.name = name;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Area getArea() {
        return area;
    }

    public void setArea(Area area) {
        this.area = area;
    }
}

```

*Figura 10 Classe Skill.java*

In essa, come si può vedere dalla figura 10 ci sono:

- metodi setter e getter, per settare e prendere, rispettivamente, i valori di questa entità
- costruttore vuoto

- costruttore con i parametri in ingresso per inizializzare gli attributi di questa entità

**SkillController.java** è il controller dell'entità Skill, in cui è definita una sola variabile di tipo SkillService. I metodi sono:

- `getSkills(@PathVariable int id)`, che restituisce una lista di oggetti di tipo Skill appartenenti ad un'area con id uguale a quello passato al metodo stesso
- `getSkill(@PathVariable int id)`, che restituisce un oggetto di tipo Skill dato un determinato id. L'annotazione `@PathVariable` serve ad indicare che l'id è quello dato nell'URL.
- `addSkill(@RequestBody Skill skill)`, che consente di aggiungere un nuovo oggetto di tipo Skill. L'annotazione `@RequestBody` serve ad indicare che l'oggetto deve essere inviato nel corpo della richiesta (in formato json)
- `updateSkill(@RequestBody Skill skill, @PathVariable int areasId, @PathVariable int id)`, che consente di apportare delle modifiche ad un determinato oggetto di tipo Skill, che ha un determinato id ed appartiene ad una specifica area con id uguale a `areasId`. L'oggetto di tipo Skill modificato deve essere inviato nel corpo della richiesta
- `deleteSkill(@PathVariable int areasId, @PathVariable int id)`, che cancella l'oggetto di tipo Skill che ha id uguale a quello passato al metodo stesso ed appartiene ad una specifica area con id uguale a `areasId`

```

package plansoft.skillsmanagement.skill;

import java.util.List;

import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

import plansoft.skillsmanagement.area.Area;

@RestController
public class SkillController {

    @Autowired
    private SkillService skillService;

    @RequestMapping("/areas/{id}/skills")
    public List<Skill> getSkills(@PathVariable int id) {
        return skillService.getSkills(id);
    }

    @RequestMapping("/areas/{areasId}/skills/{id}")
    public Optional<Skill> getSkill(@PathVariable int id) {
        return skillService.getSkill(id);
    }

    @RequestMapping(method = RequestMethod.POST, value = "/areas/{areasId}/skills")
    public void addSkill(@RequestBody Skill skill, @PathVariable int areasId) {
        skill.setArea(new Area(areasId, ""));
        skillService.addSkill(skill);
    }

    @RequestMapping(method = RequestMethod.PUT, value = "/areas/{areasId}/skills/{id}")
    public void updateSkill(@RequestBody Skill skill, @PathVariable int areasId, @PathVariable int id) {
        skill.setArea(new Area(areasId, ""));
        skillService.updateSkill(skill, id);
    }

    @RequestMapping(method = RequestMethod.DELETE, value = "/areas/{areasId}/skills/{id}")
    public void deleteSkill(@PathVariable int areasId, @PathVariable int id) {
        skillService.deleteSkill(id);
    }
}

```

Figura 11 Controller dell'entità Skill

**SkillService.java** è il service dell'entità Skill. La classe ha una sola variabile di tipo SkillRepository, sfruttata all'interno dei metodi per chiamare quelli della repository ed eseguire, quindi, le varie richieste. I metodi sono:

- getSkills(int id), che restituisce una lista di oggetti di tipo Skill appartenenti ad un'area con id uguale a quello passato al metodo stesso
- getSkill(int id), che restituisce un oggetto di tipo Skill dato un determinato id.
- addSkill(Skill skill), che consente di aggiungere un nuovo oggetto di tipo Skill.



- `updateSkill(Skill skill, int id)`, che consente di apportare delle modifiche ad un determinato oggetto di tipo `Skill`, che ha un determinato `id`.
- `deleteSkill(int id)`, che cancella l'oggetto di tipo `Skill` che ha `id` uguale a quello passato al metodo stesso

```
package plansoft.skillsmanagement.skill;

import java.util.ArrayList;

@Service
public class SkillService {

    @Autowired
    private SkillRepository skillRepository;

    public List<Skill> getSkills(int id) {
        List<Skill> skills = new ArrayList<>();
        skillRepository.findByAreaId(id).forEach(skills::add);
        return skills;
    }

    public Optional<Skill> getSkill(int id) {
        return skillRepository.findById(id);
    }

    public void addSkill(Skill skill) {
        skillRepository.save(skill);
    }

    public void updateSkill(Skill skill, int id) {
        Skill currentSkill = skillRepository.findById(id).get();
        currentSkill.setName((skill.getName()));
        skillRepository.save(currentSkill);
    }

    public void deleteSkill(int id) {
        skillRepository.deleteById(id);
    }
}
```

*Figura 12 Service dell'entità Skill*

**SkillRepository.java** è la repository dell'entità `Skill`. I tipi specificati questa sono di tipo `Skill` e `Integer`.

```

package plansoft.skillsmanagement.skill;

import java.util.List;
import org.springframework.data.repository.CrudRepository;

public interface SkillRepository extends CrudRepository<Skill, Integer>{

    public List<Skill> findByAreaId(int id);
}

```

*Figura 13 Repository dell'entità Skill*

Come si può vedere dalla figura 13, nella repository dell'entità Skill è stato necessario scrivere l'implementazione di un metodo custom che consente di ottenere tutte le competenze (oggetti di tipo Skill) che appartengono ad una determinata area.

## Entità Level

L'entità Level serve a rappresentare il livello di una determinata competenza posseduta da un dipendente. Per gestire tale entità sono state create tre classi e un'interfaccia.

**Level.java** definisce l'entità stessa che possiede un id, una descrizione e un ordine (variabile levelOrder nella figura 13). La classe è contrassegnata con l'annotazione `@Entity` che specifica, appunto, che quella classe rappresenta un'entità in un database. Essa ha come chiave primaria la variabile id e, per specificare questo, viene usata l'annotazione `@Id`. `@GeneratedValue` serve a specificare, invece, che quell'id si dovrà auto-incrementare tutte le volte che viene creato un nuovo oggetto di tipo Level.

```

package plansoft.skillsmanagement.level;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Level {

    @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;
    private String description;
    private int levelOrder;

    public Level() {

    }

    public Level(int id, String description, int levelOrder) {
        this.id = id;
        this.description = description;
        this.levelOrder = levelOrder;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public int getLevelOrder() {
        return levelOrder;
    }

    public void setLevelOrder(int levelOrder) {
        this.levelOrder = levelOrder;
    }
}

```

*Figura 14 Classe Level.java*

In essa, come si può vedere dalla figura 14 ci sono:

- metodi setter e getter, per settare e prendere, rispettivamente, i valori di questa entità
- costruttore vuoto
- costruttore con i parametri in ingresso per inizializzare gli attributi di questa entità

**LevelController.java** è il controller dell'entità Level, in cui è definita una sola variabile di tipo LevelService. I metodi sono:

- getAllLevels(), che restituisce una lista di oggetti di tipo Level

- `getLevel(@PathVariable int id)`, che restituisce un oggetto di tipo `Level` dato un determinato id. L'annotazione `@PathVariable` serve ad indicare che l'id è quello dato nell'URL.
- `addLevel(@RequestBody Level level)`, che consente di aggiungere un nuovo oggetto di tipo `Level`. L'annotazione `@RequestBody` serve ad indicare che l'oggetto deve essere inviato nel corpo della richiesta (in formato json)
- `updateLevel(@RequestBody Level level, @PathVariable int id)`, che consente di apportare delle modifiche ad un determinato oggetto di tipo `Level`, che ha un determinato id. L'oggetto di tipo `Level` modificato deve essere inviato nel corpo della richiesta
- `deleteLevel(@PathVariable int id)`, che cancella l'oggetto di tipo `Level` che ha id uguale a quello passato al metodo stesso

```

package plansoft.skillsmanagement.level;

import java.util.List;

import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class LevelController {

    @Autowired
    private LevelService levelService;

    @RequestMapping("/levels")
    public List<Level> getAllLevels() {
        return levelService.getAllLevels();
    }

    @RequestMapping("/levels/{id}")
    public Optional<Level> getLevel(@PathVariable int id) {
        return levelService.getLevel(id);
    }

    @RequestMapping(method = RequestMethod.POST, value = "/levels")
    public void addLevel(@RequestBody Level level) {
        levelService.addLevel(level);
    }

    @RequestMapping(method = RequestMethod.PUT, value = "/levels/{id}")
    public void updateLevel(@RequestBody Level level, @PathVariable int id) {
        levelService.updateLevel(level, id);
    }

    @RequestMapping(method = RequestMethod.DELETE, value = "/levels/{id}")
    public void deleteLevel(@PathVariable int id) {
        levelService.deleteLevel(id);
    }
}

```

*Figura 15 Controller dell'entità Level*

**LevelService.java** è il service dell'entità Level. La classe ha una sola variabile di tipo LevelRepository, sfruttata all'interno dei metodi per chiamare quelli della repository ed eseguire, quindi, le varie richieste. I metodi sono:

- getAllLevels(), che restituisce una lista di oggetti di tipo Level
- getLevel(int id), che restituisce un oggetto di tipo Level dato un determinato id.
- addLevel(Level level), che consente di aggiungere un nuovo oggetto di tipo Level.

- `updateLevel(Level level, int id)`, che consente di apportare delle modifiche ad un determinato oggetto di tipo `Level`, che ha un determinato `id`.
- `deleteLevel(int id)`, che cancella l'oggetto di tipo `Level` che ha `id` uguale a quello passato al metodo stesso

```
package plansoft.skillsmanagement.level;

import java.util.ArrayList;

import java.util.List;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class LevelService {

    @Autowired
    private LevelRepository levelRepository;

    public List<Level> getAllLevels() {
        List<Level> levels = new ArrayList<>();
        levelRepository.findAll().forEach(levels::add);
        return levels;
    }

    public Optional<Level> getLevel(int id) {
        return levelRepository.findById(id);
    }

    public void addLevel(Level level) {
        List<Level> levels = getAllLevels();
        for (Level currentLevel : levels) {
            if (currentLevel.getLevelOrder() == level.getLevelOrder()) {
                int currentLevelOrder = level.getLevelOrder();
                for (Level levelToChange : levels) {
                    if (levelToChange.getLevelOrder() >= currentLevelOrder) {
                        levelToChange.setLevelOrder(levelToChange.getLevelOrder() + 1);
                    }
                }
                break;
            }
        }
        levelRepository.save(level);
    }

    public void updateLevel(Level level, int id) {
        Level currentLevel = levelRepository.findById(id).get();
        currentLevel.setDescription(level.getDescription());
        currentLevel.setLevelOrder((level.getLevelOrder()));
        levelRepository.save(currentLevel);
    }

    public void deleteLevel(int id) {
        levelRepository.deleteById(id);
    }
}
```

*Figura 16 Service dell'entità Level*

**LevelRepository.java** è la repository dell'entità `Level`. I tipi specificati questa sono di tipo `Level` e `Integer`.

```
package plansoft.skillsmanagement.level;

import org.springframework.data.repository.CrudRepository;

public interface LevelRepository extends CrudRepository<Level, Integer> {

}
```

*Figura 17 Repository dell'entità Level*

Come si può vedere dalla figura 17, in questo caso è stato sufficiente estendere l'interfaccia `CrudRepository` senza implementare dei metodi custom.

## Entità Area

L'entità `Area` serve a rappresentare un insieme di competenze tutte correlate ad uno specifico argomento (es. linguaggi di programmazione, sistemi operativi, ecc.). Per gestire tale entità sono state create tre classi e un'interfaccia.

**Area.java** definisce l'entità stessa che possiede un id ed una descrizione. La classe è contrassegnata con l'annotazione `@Entity` che specifica, appunto, che quella classe rappresenta un'entità in un database. Essa ha come chiave primaria la variabile `id` e, per specificare questo, viene usata l'annotazione `@Id`. `@GeneratedValue` serve a specificare, invece, che quell'id si dovrà auto-incrementare tutte le volte che viene creato un nuovo oggetto di tipo `Area`.

```

package plansoft.skillsmanagement.area;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Area {

    @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;
    private String description;

    public Area() {
    }

    public Area(int id, String description) {
        this.id = id;
        this.description = description;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }
}

```

*Figura 18 Classe Area.java*

In essa, come si può vedere dalla figura 18 ci sono:

- metodi setter e getter, per settare e prendere, rispettivamente, i valori di questa entità
- costruttore vuoto
- costruttore con i parametri in ingresso per inizializzare gli attributi di questa entità

**AreaController.java** è il controller dell'entità Area, in cui è definita una sola variabile di tipo AreaService. I metodi sono:

- `getAreas()`, che restituisce una lista di oggetti di tipo Area



- `getArea(@PathVariable int id)`, che restituisce un oggetto di tipo `Area` dato un determinato id. L'annotazione `@PathVariable` serve ad indicare che l'id è quello dato nell'URL.
- `addArea(@RequestBody Area area)`, che consente di aggiungere un nuovo oggetto di tipo `Area`. L'annotazione `@RequestBody` serve ad indicare che l'oggetto deve essere inviato nel corpo della richiesta (in formato json)
- `updateArea (@RequestBody Area area, @PathVariable int id)`, che consente di apportare delle modifiche ad un determinato oggetto di tipo `Area`, che ha un determinato id. L'oggetto di tipo `Area` modificato deve essere inviato nel corpo della richiesta
- `deleteArea(@PathVariable int id)`, che cancella l'oggetto di tipo `Area` che ha id uguale a quello passato al metodo stesso

```

package plansoft.skillsmanagement.area;

import java.util.List;

import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@RestController
public class AreaController {

    @Autowired
    private AreaService areaService;

    @RequestMapping("/areas")
    public List<Area> getAreas() {
        return areaService.getAreas();
    }

    @RequestMapping("/areas/{id}")
    public Optional<Area> getArea(@PathVariable int id) {
        return areaService.getArea(id);
    }

    @RequestMapping(method = RequestMethod.POST, value = "/areas")
    public void addArea(@RequestBody Area area) {
        areaService.addArea(area);
    }

    @RequestMapping(method = RequestMethod.PUT, value = "/areas/{id}")
    public void updateArea(@RequestBody Area area, @PathVariable int id) {
        areaService.updateArea(area, id);
    }

    @RequestMapping(method = RequestMethod.DELETE, value = "/areas/{id}")
    public void deleteArea(@PathVariable int id) {
        areaService.deleteArea(id);
    }
}

```

*Figura 19 Controller dell'entità Area*

**AreaService.java** è il service dell'entità Area. La classe ha una sola variabile di tipo AreaRepository, sfruttata all'interno dei metodi per chiamare quelli della repository ed eseguire, quindi, le varie richieste. I metodi sono:

- getAreas(), che restituisce una lista di oggetti di tipo Area
- getArea(int id), che restituisce un oggetto di tipo Area dato un determinato id.
- addArea(Area area), che consente di aggiungere un nuovo oggetto di tipo Area.

- `updateArea(Area area, int id)`, che consente di apportare delle modifiche ad un determinato oggetto di tipo `Area`, che ha un determinato `id`.
- `deleteArea(int id)`, che cancella l'oggetto di tipo `Area` che ha `id` uguale a quello passato al metodo stesso

```
package plansoft.skillsmanagement.area;

import java.util.ArrayList;

@Service
public class AreaService {

    @Autowired
    private AreaRepository areaRepository;

    public List<Area> getAreas() {
        List<Area> areas = new ArrayList<>();
        areaRepository.findAll().forEach(areas::add);
        return areas;
    }

    public Optional<Area> getArea(int id) {
        return areaRepository.findById(id);
    }

    public void addArea(Area area) {
        areaRepository.save(area);
    }

    public void updateArea(Area area, int id) {
        Area currentArea = areaRepository.findById(id).get();
        currentArea.setDescription(area.getDescription());
        areaRepository.save(currentArea);
    }

    public void deleteArea(int id) {
        areaRepository.deleteById(id);
    }
}
```

*Figura 20 Service dell'entità Area*

**AreaRepository.java** è la repository dell'entità `Area`. I tipi specificati questa sono di tipo `Area` e `Integer`.

```

package plansoft.skillsmanagement.area;

import org.springframework.data.repository.CrudRepository;

public interface AreaRepository extends CrudRepository<Area, Integer>{

}

```

*Figura 21 Repository dell'entità Area*

Come si può vedere dalla figura 21, anche in questo caso sono sufficienti i metodi base messi a disposizione dall'interfaccia CrudRepository

## Entità SkillEmployee

Come si è potuto vedere dalla figura 1 del paragrafo “Schema ER (Entità-Relazione)”, ci sono tre entità (Employee, Skill e Level) che convergono tra di loro in una relazione molti a molti (N:M); in Spring, ci sono alcuni metodi per gestire quest'ultima e, nel caso del presente progetto, è stato deciso di farla diventare un'entità a sé stante. Si ha, dunque, l'entità SkillEmployee, che ci consente di rappresentare la relazione esistente tra un dipendente, una skill e un livello (un dipendente possiede una o più competenze con un certo livello). Come per le altre, anche per la gestione di tale entità sono state create tre classi e un'interfaccia.

**SkillEmployee.java** definisce l'entità stessa che possiede un id, l'id di un oggetto Employee (idEmployee), l'id dell'oggetto Skill (idSkill) associato ad un Employee e l'id del livello (idLevel) associato alla Skill di quell'Employee. La classe è contrassegnata con l'annotazione @Entity che specifica, appunto, che quella classe rappresenta un'entità in un database. Essa ha come chiave primaria la variabile id e, per specificare questo, viene usata l'annotazione @Id. @GeneratedValue serve a specificare, invece, che quell'id si dovrà

auto-incrementare tutte le volte che viene creato un nuovo oggetto di tipo SkillEmployee.

```
package plansoft.skillsmanagement.skillemployee;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class SkillEmployee {

    @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;

    private int idEmployee;
    private int idSkill;
    private int idLevel;

    public SkillEmployee() {
        super();
    }

    public SkillEmployee(int id, int idEmployee, int idSkill, int idLevel) {
        super();
        this.id = id;
        this.idEmployee = idEmployee;
        this.idSkill = idSkill;
        this.idLevel = idLevel;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public int getIdEmployee() {
        return idEmployee;
    }

    public void setIdEmployee(int idEmployee) {
        this.idEmployee = idEmployee;
    }

    public int getIdSkill() {
        return idSkill;
    }

    public void setIdSkill(int idSkill) {
        this.idSkill = idSkill;
    }

    public int getIdLevel() {
        return idLevel;
    }

    public void setIdLevel(int idLevel) {
        this.idLevel = idLevel;
    }
}
```

*Figura 22 Classe SkillEmployee.java*

In essa, come si può vedere dalla figura 22 ci sono:

- metodi setter e getter, per settare e prendere, rispettivamente, i valori di questa entità
- costruttore vuoto
- costruttore con i parametri in ingresso per inizializzare gli attributi di questa entità

**SkillEmployeeController.java** è il controller dell'entità SkillEmployee, in cui è definita una sola variabile di tipo SkillEmployeeService. I metodi sono:

- getAllSkillEmployee(), che restituisce una lista di oggetti di tipo SkillEmployee
- getAllSkillEmployeeByIdSkill(@PathVariable("idskill") int idSkill), che restituisce uno o più oggetti di tipo SkillEmployee dato un determinato idSkill. La logica del metodo, quindi, è quella di restituire tutti i dipendenti che hanno una determinata competenza con id uguale a idSkill. L'annotazione @PathVariable serve ad indicare che l'id è quello dato nell'URL col nome di variabile "idskill"
- getAllSkillEmployeeByIdSkillAndIdLevel(@PathVariable("idskill") int idSkill, @PathVariable("idlevel") int idLevel), che restituisce uno o più oggetti di tipo SkillEmployee dato un determinato idSkill e idLevel. La logica del metodo, quindi, è quella di restituire tutti i dipendenti che hanno una determinata competenza con id uguale a idSkill e con livello pari a quello con id uguale a idLevel o maggiore. Le annotazioni @PathVariable servono ad indicare che gli id sono quelli dati nell'URL coi nomi di variabilei "idskill" e "idlevel", rispettivamente
- getAllSkillEmpByIdEmployee(@PathVariable("idemployee") int idEmployee), che restituisce uno o più oggetti di tipo SkillEmployee dato un determinato idEmployee. La logica del metodo, quindi, è quello

di restituire tutte le competenze (oggetti Skill), con relativi livelli, associate al dipendente identificato con l'id pari a idEmployee

- `updateSkillEmployeeLevel(@RequestBody SkillEmployee skillEmployee, @PathVariable("idemployee") int idEmployee, @PathVariable("idskill") int idSkill)`, che consente di modificare il livello di una competenza con id pari a idSkill associata ad un dipendente con id pari a idEmployee. L'annotazione `@RequestBody` serve ad indicare che l'oggetto SkillEmployee modificato deve essere inviato nel corpo della richiesta (in formato json)
- `addSkillEmployee(@RequestBody SkillEmployee skillEmployee)`, che consente di aggiungere un nuovo oggetto di tipo SkillEmployee. L'annotazione `@RequestBody` serve ad indicare che l'oggetto deve essere inviato nel corpo della richiesta (in formato json)
- `deleteSkillEmp(@PathVariable("idemployee") int idEmployee, @PathVariable("idskill") int idSkill)`, che cancella l'oggetto di tipo SkillEmployee, che ha un determinato idEmployee e idSkill. La logica del metodo, quindi, è quella di cancellare una competenza ad un determinato dipendente

```

package plansoft.skillsmanagement.skillemployee;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class SkillEmployeeController {

    @Autowired
    private SkillEmployeeService skillEmployeeService;

    @RequestMapping("/skillemployees")
    public List<SkillEmployee> getAllSkillEmployee() {
        return skillEmployeeService.getAllSkillEmployee();
    }

    @RequestMapping("/skillemployees/skills/{idskill}")
    public List<SkillEmployee> getAllSkillEmployeeByIdSkill(@PathVariable("idskill") int idSkill) {
        return skillEmployeeService.getAllSkillEmployeeByIdSkill(idSkill);
    }

    @RequestMapping("/skillemployees/skills/{idskill}/levels/{idlevel}")
    public List<SkillEmployee> getAllSkillEmployeeByIdSkillAndIdLevel(@PathVariable("idskill") int idSkill,
        @PathVariable("idlevel") int idLevel) {
        return skillEmployeeService.getAllSkillEmployeeByIdSkillAndIdLevel(idSkill, idLevel);
    }

    @RequestMapping("/skillemployees/employees/{idemployee}")
    public List<SkillEmployee> getAllSkillEmpByIdEmployee(@PathVariable("idemployee") int idEmployee) {
        return skillEmployeeService.getAllSkillEmployeeByIdEmployee(idEmployee);
    }

    @RequestMapping(method = RequestMethod.PUT, value = "/skillemployees/employees/{idemployee}/skills/{idskill}")
    public void updateSkillEmployeeIdLevel(@RequestBody SkillEmployee skillEmployee, @PathVariable("idemployee") int idEmployee,
        @PathVariable("idskill") int idSkill) {
        skillEmployeeService.updateSkillEmployeeIdLevel(skillEmployee.getIdLevel(), idEmployee, idSkill);
    }

    @RequestMapping(method = RequestMethod.POST, value = "/skillemployees")
    public void addSkillEmployee(@RequestBody SkillEmployee skillEmployee) {
        skillEmployeeService.addSkillEmployee(skillEmployee);
    }

    @RequestMapping(method = RequestMethod.DELETE, value = "/skillemps/employees/{idemployee}/skills/{idskill}")
    public void deleteSkillEmp(@PathVariable("idemployee") int idEmployee, @PathVariable("idskill") int idSkill) {
        skillEmployeeService.deleteSkillEmployee(idEmployee, idSkill);
    }
}

```

Figura 23 Controller dell'entità SkillEmployee

**SkillEmployeeService.java** è il service dell'entità SkillEmployee. La classe ha una sola variabile di tipo SkillEmployeeRepository, sfruttata all'interno dei metodi per chiamare quelli della repository ed eseguire, quindi, le varie richieste. I metodi sono:

- getAllSkillEmployee(), che restituisce una lista di oggetti di tipo SkillEmployee
- getAllSkillEmployeeByIdSkill(int idSkill), che restituisce uno o più oggetti di tipo SkillEmployee dato un determinato idSkill. La logica del metodo, quindi, è quella di restituire tutti i dipendenti che hanno una determinata competenza con id uguale a idSkill.



- `getAllSkillEmployeeByIdSkillAndIdLevel( int idSkill, int idLevel)`, che restituisce uno o più oggetti di tipo `SkillEmployee` dato un determinato `idSkill` e `idLevel`. La logica del metodo, quindi, è quella di restituire tutti i dipendenti che hanno una determinata competenza con id uguale a `idSkill` e con livello pari a quello con id uguale a `idLevel` o maggiore.
- `getAllSkillEmployeeByIdEmployee(int idEmployee)`, che restituisce uno o più oggetti di tipo `SkillEmployee` dato un determinato `idEmployee`. La logica del metodo, quindi, è quello di restituire tutte le competenze, con relativi livelli, associate al dipendente identificato con l'id pari a `idEmployee`
- `updateSkillEmployeeIdLevel(int idLevel, int idEmployee, int idSkill)`, che consente di modificare il livello di una competenza con id pari a `idSkill` associata ad un dipendente con id pari a `idEmployee`. Il nuovo livello assegnato è quello passato nel metodo stesso (`idLevel`).
- `addSkillEmployee(SkillEmployee skillEmployee)`, che consente di aggiungere un nuovo oggetto di tipo `SkillEmployee`.
- `deleteSkillEmp(int idEmployee, int idSkill)`, che cancella l'oggetto di tipo `SkillEmployee`, che ha un determinato `idEmployee` e `idSkill`. La logica del metodo, quindi, è quella di cancellare una competenza ad un determinato dipendente

```

package plansoft.skillsmanagement.skillemployee;

import java.util.ArrayList;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import plansoft.skillsmanagement.level.Level;
import plansoft.skillsmanagement.level.LevelRepository;

@Service
public class SkillEmployeeService {

    @Autowired
    private SkillEmployeeRepository skillEmployeeRepository;

    @Autowired
    private LevelRepository levelRepository;

    public List<SkillEmployee> getAllSkillEmployees() {
        List<SkillEmployee> skillEmployees = new ArrayList<>();
        skillEmployeeRepository.findAll().forEach(skillEmployees::add);
        return skillEmployees;
    }

    // return a list of SkillEmp in which every element represents an employee who
    // has
    // the skill with id equal to id skill
    public List<SkillEmployee> getAllSkillEmployeesByIdSkill(int idSkill) {
        List<SkillEmployee> skillEmployees = new ArrayList<>();
        skillEmployeeRepository.findAllByIdSkill(idSkill).forEach(skillEmployees::add);
        return skillEmployees;
    }

    // return a list of SkillEmp in which every element represents an employee who
    // has
    // the skill with id equal to id skill and the level equal or higher than id lev
    public List<SkillEmployee> getAllSkillEmployeesByIdSkillAndIdLevel(int idSkill, int idLevel) {
        List<SkillEmployee> skillEmployeesToReturn = new ArrayList<>();
        int inputLevelOrder = levelRepository.findById(idLevel).get().getLevelOrder();
        Iterable<SkillEmployee> skillEmployeesByIdSkill = skillEmployeeRepository.findAllByIdSkill(idSkill);
        for (SkillEmployee skillEmployee : skillEmployeesByIdSkill) {
            Level currentLevel = levelRepository.findById(skillEmployee.getIdLevel()).get();
            if (currentLevel.getLevelOrder() >= inputLevelOrder)
                skillEmployeesToReturn.add(skillEmployee);
        }
        return skillEmployeesToReturn;
    }

    // return a list of SkillEmp in which every element represents a skill related
    // to an employee, who owns the skill itself
    public List<SkillEmployee> getAllSkillEmployeesByIdEmployee(int idEmployee) {
        List<SkillEmployee> skillEmps = new ArrayList<>();
        skillEmployeeRepository.findAllByIdEmployee(idEmployee).forEach(skillEmps::add);
        return skillEmps;
    }

    // change level of a skill related to an employee
    public void updateSkillEmployeeIdLevel(int idLevel, int idEmployee, int idSkill) {
        List<SkillEmployee> skillEmployees = new ArrayList<>();
        skillEmployeeRepository.findAllByIdEmployee(idEmployee).forEach(skillEmployees::add); // all skills of employee
        // id_emp
        for (SkillEmployee skillEmployee : skillEmployees) {
            if (skillEmployee.getIdSkill() == idSkill) {
                skillEmployee.setIdLevel(idLevel);
                skillEmployeeRepository.save(skillEmployee);
            }
        }
    }

    public void addSkillEmployee(SkillEmployee skillEmployee) {
        skillEmployeeRepository.save(skillEmployee);
    }

    public void deleteSkillEmployee(int idEmployee, int idSkill) {
        List<SkillEmployee> skillEmployees = new ArrayList<>();
        skillEmployeeRepository.findAll().forEach(skillEmployees::add);
        for (SkillEmployee skillEmployee : skillEmployees) {
            if (skillEmployee.getIdEmployee() == idEmployee && skillEmployee.getIdSkill() == idSkill) {
                skillEmployeeRepository.delete(skillEmployee);
            }
        }
    }
}

```

Figura 24 Service dell'entità SkillEmployee

**SkillEmployeeRepository.java** è la repository dell'entità SkillEmployee. I tipi specificati questa sono di tipo SkillEmployee e Integer.

```

package plansoft.skillsmanagement.skillemployee;

import java.util.List;
import org.springframework.data.repository.CrudRepository;

public interface SkillEmployeeRepository extends CrudRepository<SkillEmployee, Integer> {

    List<SkillEmployee> findAllByIdSkill(int idSkill);

    List<SkillEmployee> findAllByIdEmployee(int idEmployee);

}

```

*Figura 25 Repository dell'entità SkillEmployee*

Come si può vedere dalla figura 25, nella repository dell'entità SkillEmployee è stato necessario scrivere l'implementazione di due metodi custom che consentono di ottenere tutti gli oggetti SkillEmployee ai quali è associato un determinato idSkill (findAllByIdSkill(int idSkill)) e tutti quelli ai quali è associato un determinato idEmployee (findAllByIdEmployee(int idEmployee)).

## Postman

Si tratta di un client API che viene usato per sviluppare, documentare, condividere, testare le API. In questo progetto è stato sfruttato per la sua funzionalità di testing. Come si può vedere dalla figura 26 la sua interfaccia è molto intuitiva.

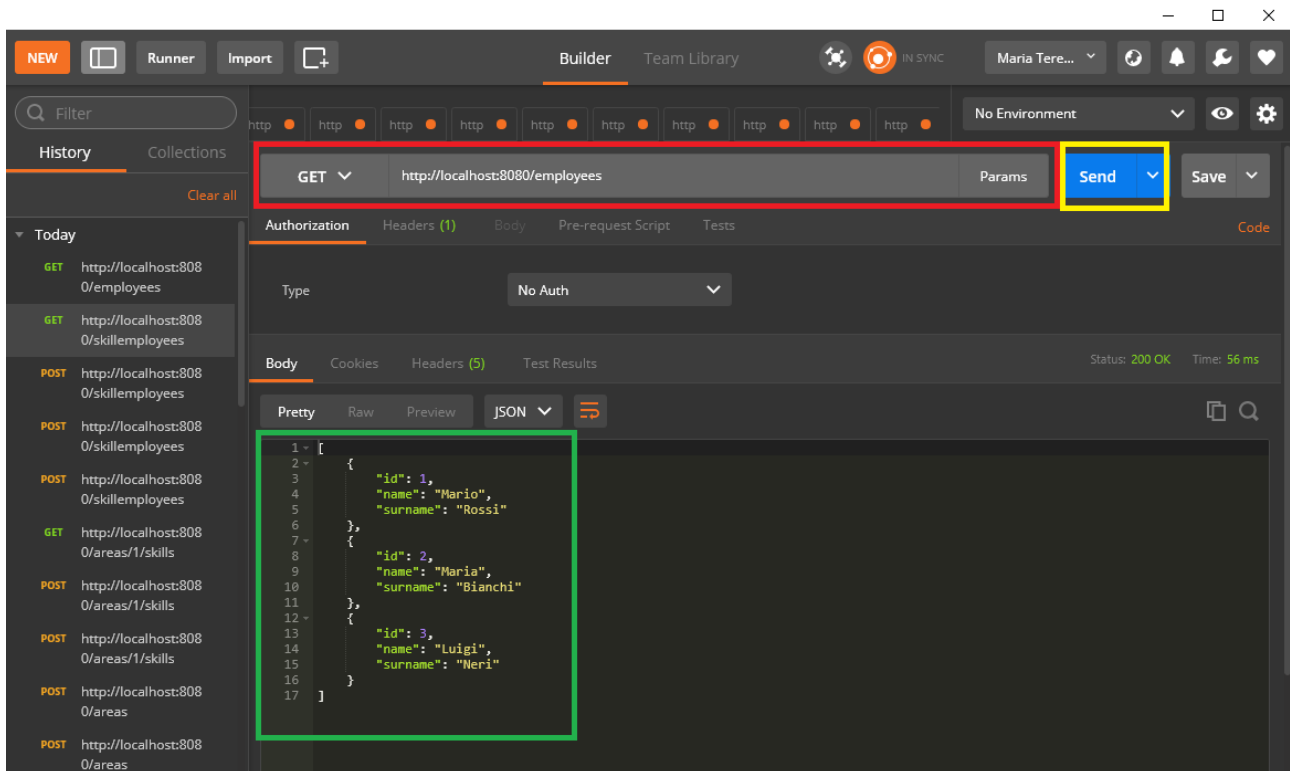


Figura 26 Postman: richiesta GET

Nella barra grigia (all'interno del riquadro rosso) si possono inserire le richieste http (le principali sono la GET, POST, PUT e DELETE), che possono essere inviate tramite il tasto "Send" (riquadro giallo); l'indirizzo URL inserito è proprio quello che si trova nel controller di una specifica entità. In basso, nel riquadro verde, ci sono i risultati della richiesta, in questo caso una GET. Per le richieste di tipo POST, viene usata la sezione "Body" per inserire l'oggetto in formato JSON (riquadri rossi nella figura 27), nel quale, come si può notare, manca l'id; si ricorda, infatti, che per tutti gli id è stata impostata la modalità di auto-generazione.

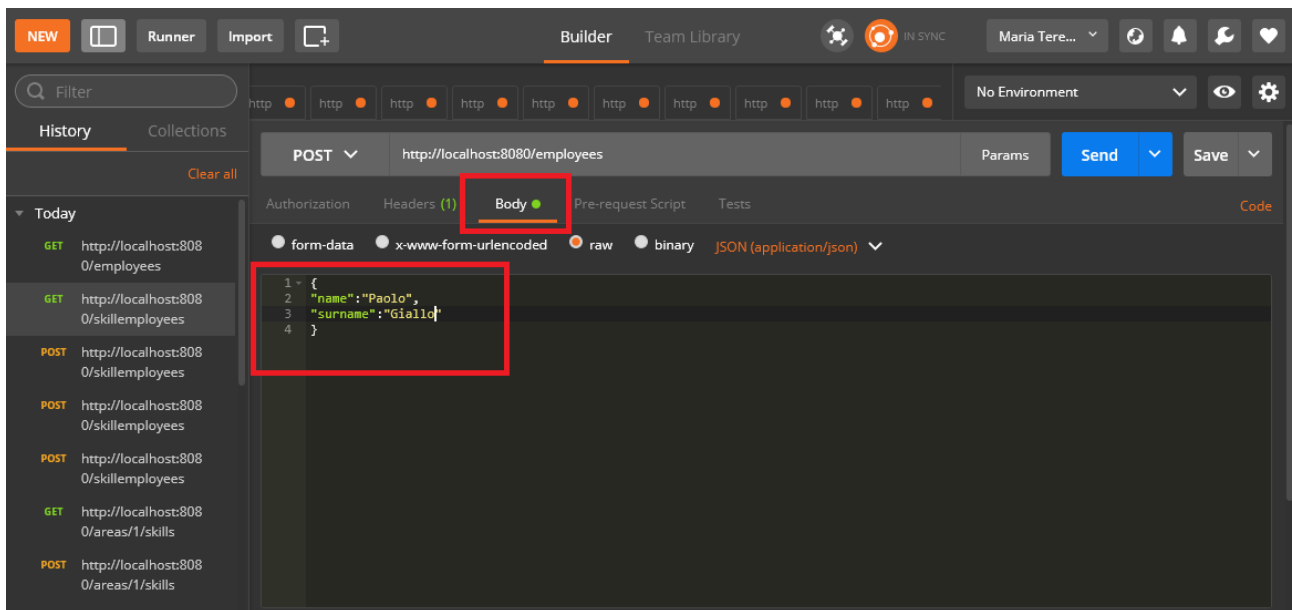


Figura 27 Postman: richiesta POST

Nella figura sottostante si può vedere il risultato della richiesta GET, specifica per un oggetto (riquadri verdi)

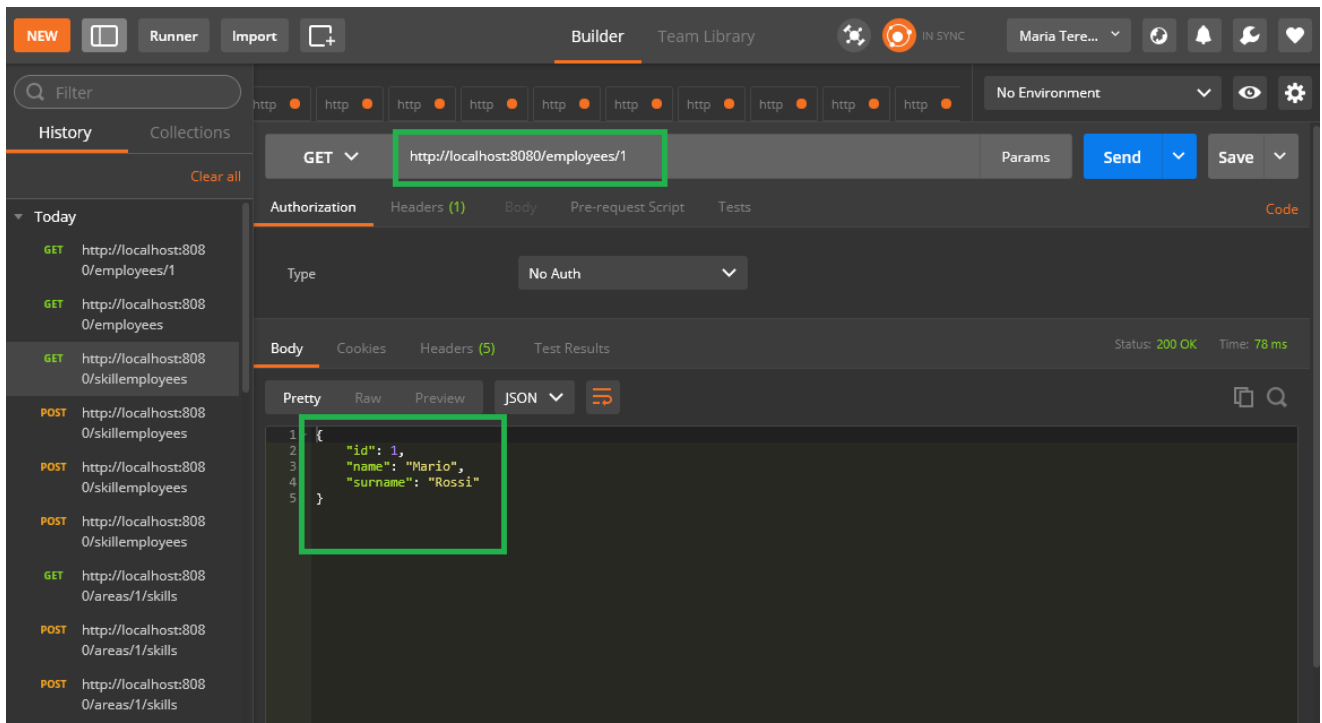


Figura 28 Postman: richiesta GET per un oggetto singolo

Nella figura 29, invece, viene eseguita una richiesta PUT: è, quindi, necessario, inserire l'URL per l'oggetto che si vuole modificare, andare nella sezione "Body" e inserire i parametri che si vogliono modificare, sempre in formato JSON.

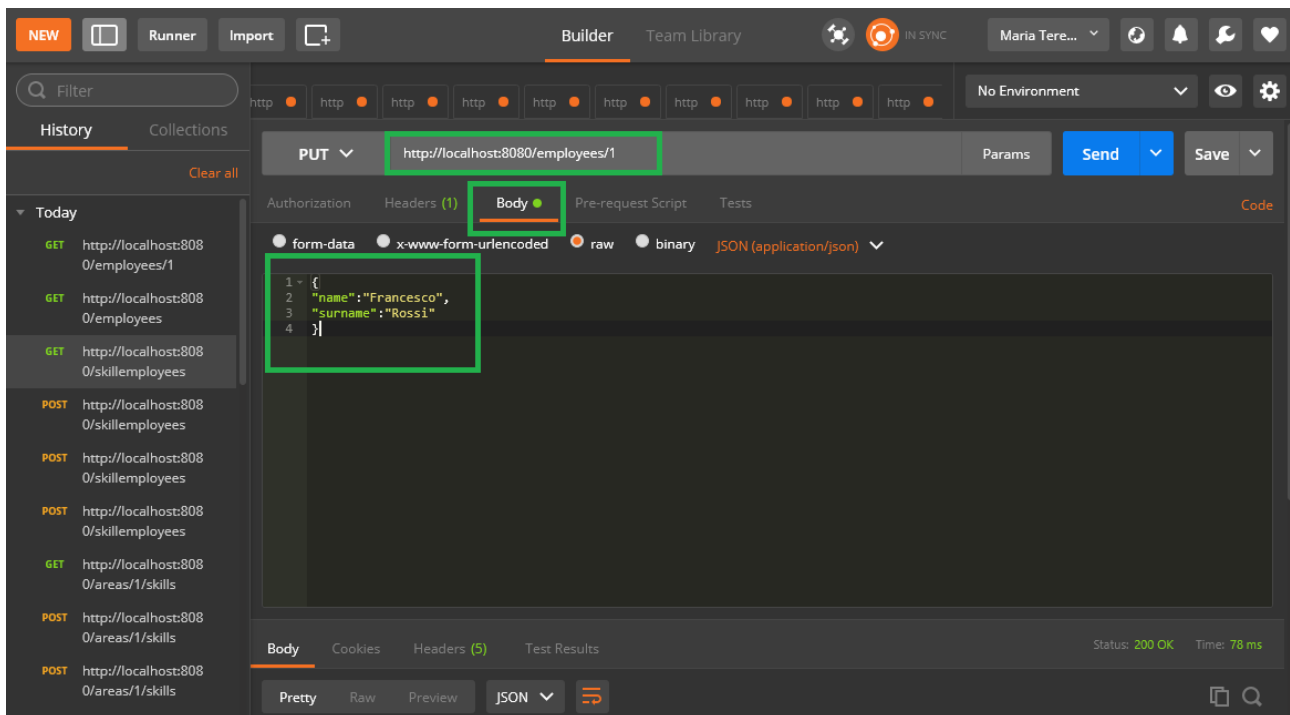


Figura 29 Postman: richiesta PUT

Infine, nella figura 30, si mostra la richiesta DELETE che, come suggerisce il nome stesso, consente di cancellare un oggetto

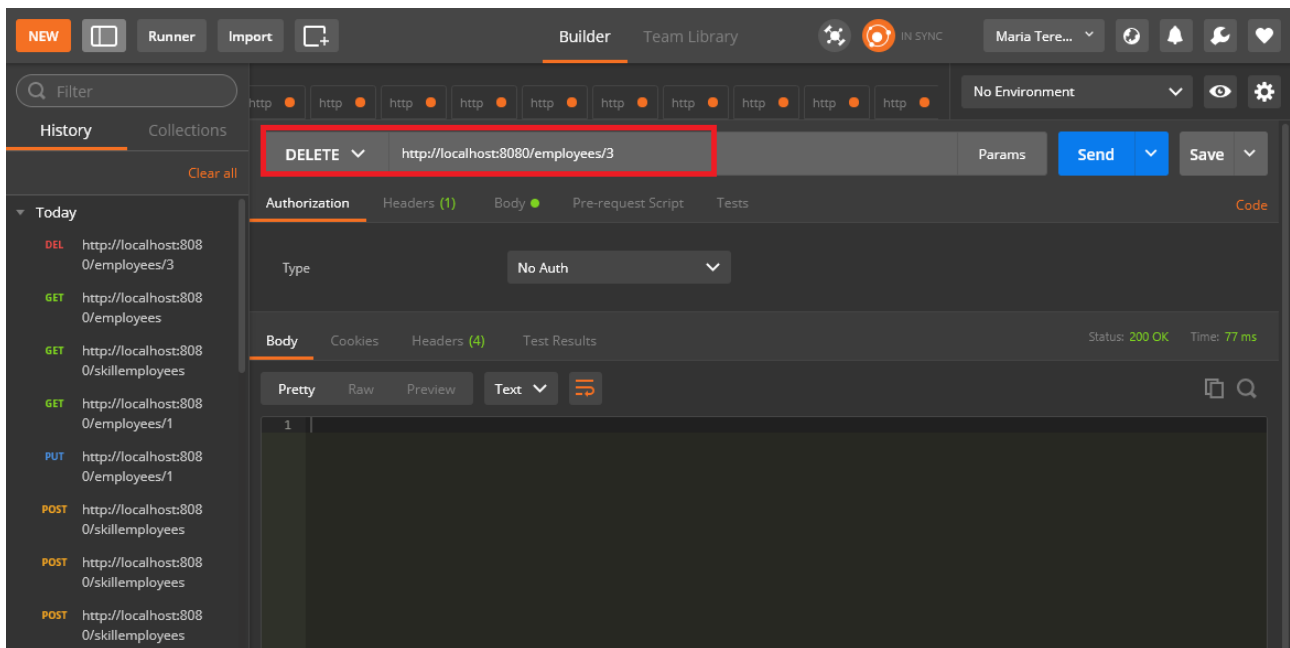


Figura 30 Postman: richiesta DELETE

Per controprova, se si esegue una richiesta GET per ottenere tutti gli Employee esistenti, mancherà l'ultimo Employee (Luigi Neri con id uguale a 3). Il risultato di questa controprova si mostra in figura 31.

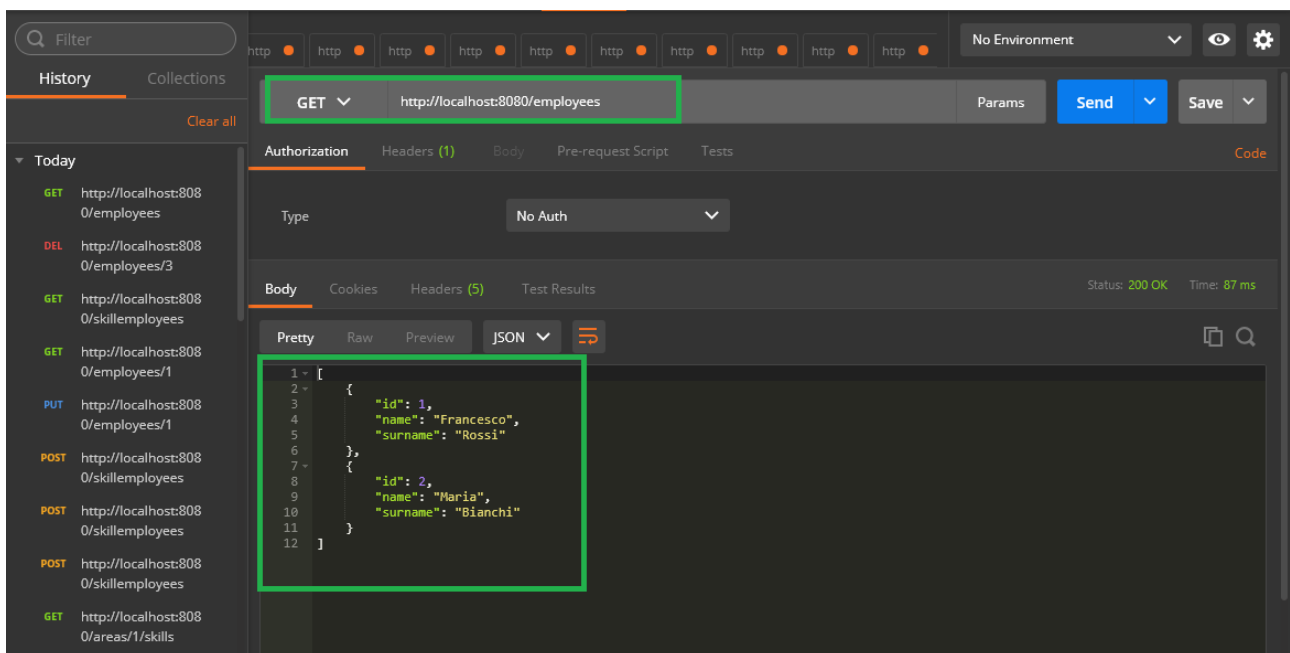


Figura 31 Postman: richiesta GET (controprova dopo la DELETE)

Nelle varie figure con la UI di Postman, si può notare, inoltre, che a sinistra c'è una sezione chiamata "History", nella quale vengono salvate tutte le richieste http, fatte per eseguire i vari test. Questa può essere molto utile in fase di debug.