Difference between RestController and Controller?

Great question! Both @RestController and @Controller are used in Spring MVC, but they serve **different purposes**, especially in how they return data.

**✅ Quick Summary**

| **Feature** | **@RestController** | **@Controller** |
| --- | --- | --- |
| Purpose | REST APIs (JSON/XML responses) | Web pages (HTML + Thymeleaf/JSP) |
| Returns | JSON, XML, or any serialized object | View name (like a JSP or Thymeleaf page) |
| Combines | @Controller + @ResponseBody | Just @Controller |
| Common use case | APIs for mobile, frontend apps, microservices | MVC apps with UI rendering |

## 1. Change Maven Local Repository via settings.xml (Recommended)

1. Open your Maven settings.xml file:
   * For most systems:  
     ~/.m2/settings.xml (Linux/macOS)  
     C:\Users\<your-user>\.m2\settings.xml (Windows)
2. Add or modify the <localRepository> tag:

xml

CopyEdit

<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0

http://maven.apache.org/xsd/settings-1.0.0.xsd">

<!-- Set custom local repo location -->

<localRepository>D:/custom-m2-repo</localRepository>

</settings>

🔄 After this, Maven will start downloading and storing dependencies in D:/custom-m2-repo instead of the default .m2/repository.

## All CascadeType Options

| **CascadeType** | **Description** |
| --- | --- |
| PERSIST | When the parent is persisted, the child is also persisted |
| MERGE | When the parent is merged (updated), the child is also merged |
| REMOVE | When the parent is deleted, the child is also deleted |
| REFRESH | Refreshes child entities when the parent is refreshed from the database |
| DETACH | Detaches child entities when the parent is detached from the persistence context |
| ALL | Shortcut for applying all above types |

………………………………………………………………………………………………………………………………………………………..

## 🧱 1. One-to-Many and Many-to-One

### 🔹 School.java (One-to-Many)

@Entity

public class School {

@Id @GeneratedValue

private Long id;

private String name;

@OneToMany(mappedBy = "school", cascade = CascadeType.ALL)

private List<Student> students = new ArrayList<>();

// Getters, setters

}

### 🔹 Student.java (Many-to-One)

@Entity

public class Student {

@Id @GeneratedValue

private Long id;

private String name;

@ManyToOne

@JoinColumn(name = "school\_id") // ← creates school\_id FK in student table

private School school;

// Getters, setters

}

| **Concept** | **Explanation** |
| --- | --- |
| @JoinColumn | Tells JPA where the **foreign key column** is stored |
| mappedBy | Tells JPA the **owning side is the other entity** |
| school\_id column | Created in the student table to link to school.id |
| Not in School.java | Because OneToMany doesn't hold the foreign key itself |

## 🧱 3. Many-to-Many

### 🔹 Student.java

java

CopyEdit

@ManyToMany

@JoinTable(

name = "student\_course",

joinColumns = @JoinColumn(name = "student\_id"),

inverseJoinColumns = @JoinColumn(name = "course\_id")

)

private List<Course> courses = new ArrayList<>();

### 🔹 Course.java

java

CopyEdit

@Entity

public class Course {

@Id @GeneratedValue

private Long id;

private String title;

@ManyToMany(mappedBy = "courses")

private List<Student> students = new ArrayList<>();

}

## 2. One-to-One

### 🔹 Student.java (add address)

java

CopyEdit

@OneToOne(cascade = CascadeType.ALL)

@JoinColumn(name = "address\_id")

private Address address;

### 🔹 Address.java

java

CopyEdit

@Entity

public class Address {

@Id @GeneratedValue

private Long id;

private String city;

private String postalCode;

// Optional reverse mapping:

@OneToOne(mappedBy = "address")

private Student student;

}

…………………………………………………………………………..

@Query("SELECT s FROM Student s WHERE s.department = ?1")

List<Student> findByDepartment(String department);