**Project Documentation: Many-to-Many Relationship in Spring Boot**

**Overview**

This project demonstrates implementing a many-to-many relationship in Spring Boot using JPA/Hibernate. The example models a real-world scenario where software projects involve multiple developers, and developers can participate in multiple projects.

**Features**

* Entity modeling with @ManyToMany annotation.
* JPA join table to map many-to-many relationship.
* Spring Data JPA repositories providing CRUD operations.
* Service layer encapsulating business logic.
* RESTful API controllers with full CRUD endpoints for Project and Developer.
* JSON serialization handling to avoid infinite recursion.
* Configured using MySQL with Spring Boot application.properties.

**Technologies Used**

* Java 17+ (or compatible)
* Spring Boot 3.x
* Spring Data JPA
* Hibernate ORM
* MySQL Database
* Lombok (for boilerplate code reduction)
* Jackson (for JSON serialization)

**Project Structure**

|  |  |
| --- | --- |
| Package | Description |
| entity | JPA entity classes for Project and Developer |
| repository | Spring Data JPA repository interfaces |
| service | Service interfaces and implementations for business logic |
| controller | REST API controllers for CRUD operations |
| resources | Configuration files including application.properties |

**Database Model**

* **Tables:** projects\_table, developers\_table, and join table developer\_project\_table.
* The join table stores associations between projects and developers.
* Foreign key constraints ensure referential integrity.

**Entity Design Highlights**

* Project entity is the owning side of the relationship, annotated with @JoinTable.
* Developer entity references the relationship with mappedBy.
* Collection type used: Set to avoid duplicates.
* Bidirectional relationship managed carefully with JSON annotations to prevent recursion.

**API Endpoints**

|  |  |  |
| --- | --- | --- |
| Method | URI | Description |
| GET | /api/projects | Fetch all projects |
| GET | /api/projects/{id} | Fetch project by ID |
| POST | /api/projects | Create new project |
| PUT | /api/projects/{id} | Update project details |
| DELETE | /api/projects/{id} | Delete a project |
| GET | /api/developers | Fetch all developers |
| GET | /api/developers/{id} | Fetch developer by ID |
| POST | /api/developers | Create new developer |
| PUT | /api/developers/{id} | Update developer details |
| DELETE | /api/developers/{id} | Delete a developer |

**How to Run**

1. Ensure MySQL is running and database is created as configured in application.properties.
2. Clone the repository.
3. Build the project using Maven or Gradle.
4. Run the Spring Boot application via IDE or command line:

mvn spring-boot:run

1. Use Postman or similar tools to test the REST endpoints.

**Sample JSON Payloads**

**Create Developer:**

{  
 "name": "Alice"  
}

**Create Project:**

{  
 "projectTitle": "AI Chatbot"  
}

**Update Project with Developers (example):**

{  
 "projectTitle": "AI Chatbot",  
 "developers": [  
 {"id": 1},  
 {"id": 2}  
 ]  
}

**Potential Improvements**

* Add dedicated endpoints to add/remove developers from projects without full updates.
* Implement validation and exception handling with custom error responses.
* Integrate Swagger/OpenAPI for API documentation.
* Include unit and integration test coverage.

**Summary**

This project is a practical demonstration of many-to-many relationships using Spring Boot and JPA, showcasing layered architecture, API design, and database management, suitable for production extension or educational purposes.