REST API part 2

Working with Relationships in Spring Boot and JPA

Learning objectives

- Understand one-to-many and many-to-one relationships in JPA
- Learn how to connect entities using @OneToMany and @ManyToOne
- Use Spring Data JPA repositories with related entities
- Implement REST endpoints to handle relationships

The Demo Project – Smoothie Bar (Version 2)

- Goal: Extend the Smoothie Bar service to manage drink categories
- Entities:
  - Drink: Represents a specific beverage (name, size, price)
  - Category: Represents drink categories (e.g., Coffee, Smoothie)
- Relationship:
  - One Category → Many Drinks, each Drink belongs to one Category
- Endpoints:
  - /drinks – Manage individual drinks
  - /categories – Manage drink categories

Project Setup

- Continue developing the previous demo project
- Add a new Category entity to represent drink types
- Update Drink entity to include a reference to Category
- Add a CategoryRepository and CategoryController

Understanding Entity Relationships

- One-to-many: A single category can have many drinks
- Many-to-one: Each drink belongs to one category
- The database relationship is represented with a foreign key
- JPA annotations are used to define the relationship in code

Repository Design

- One repository interface for each entity:
  - DrinkRepository – handles CRUD operations for drinks
  - CategoryRepository – handles CRUD operations for categories
- JPA automatically manages relationships through the entity model
- The repositories simplify database access — no SQL required

REST Endpoints for Related Entities

- Expose endpoints for both drinks and categories
- Example endpoints:
  - GET /categories – list all categories (with their drinks)
  - GET /categories/{id} – get the category with this id (with their drinks)
  - POST /categories – add a new category
- Spring automatically serializes the relationships into JSON responses

Avoiding Circular References

- Circular references occur when two entities refer to each other in JSON output
- This leads to infinite recursion during serialization
- Ways to prevent this:
  - Ignore one side of the relationship in JSON
  - Use DTOs (Data Transfer Objects) to control output structure
- In this version we will use annotations to ignore one side of the relationship
- In a coming version we will introduce DTOs for better control of response data

Avoiding Circular References

- Circular references occur when two entities refer to each other in JSON output
- This leads to infinite recursion during serialization
- Ways to prevent this:
  - Ignore one side of the relationship in JSON
  - Use DTOs (Data Transfer Objects) to control output structure
- In this version we will use annotations to ignore one side of the relationship
- In a coming version we will introduce DTOs for better control of response data

Practical Ways to Avoid Circular References

- Option 1: Use @JsonManagedReference and @JsonBackReference
  - Marks one side of the relationship as the “parent” (managed) and the other as “child” (back).
  - The managed side is included in JSON, while the back side is omitted.
  - Best for bidirectional one-to-many relationships where both sides are modeled in entities.
- Option 2: Use @JsonIgnore
  - Simpler alternative: mark one reference to be completely ignored in JSON.
  - Useful when you don’t need to expose both sides of the relationship in responses.
  - Easier to use, but hides data in responses — not ideal if both directions are needed later.
- For full flexibility and control, consider DTOs (introduced in in the next demo project).

Preloading Relational Data

- Use data.sql to load initial categories and drinks automatically
- Example structure:
  - Categories: Smoothies, Coffee, Tea
  - Drinks: Mango Smoothie → Smoothies, Cappuccino → Coffee
- Helps demonstrate the relationship from the very first run
- Keeps demo data consistent across restarts
- Also, right now this is the only way to add drinks to categories (we will fix this later)

Lab Project: Beach Activity Service (Version 2)

- Extend the Beach Activity Service from the previous lab
- Introduce a new entity: Booking
- Each Activity can have multiple Bookings, each Booking belongs to one Activity
- Avoid circular reference (@JsonManagedReference and @JsonBackReference or @JsonIgnore on either side)
- Add repositories and REST endpoints for both entities
- Preload activities and bookings using data.sql
- Focus on:
  - Defining relationships and persisting related data
  - Retrieving parent-child data structures via REST

Summary of Technical Learnings

- Created entity relationships using JPA annotations
- Understood the difference between one-to-many and many-to-one
- Exposed related data via REST endpoints
- Preloaded relational data for testing with data.sql
- Addressed circular JSON references in bidirectional mappings