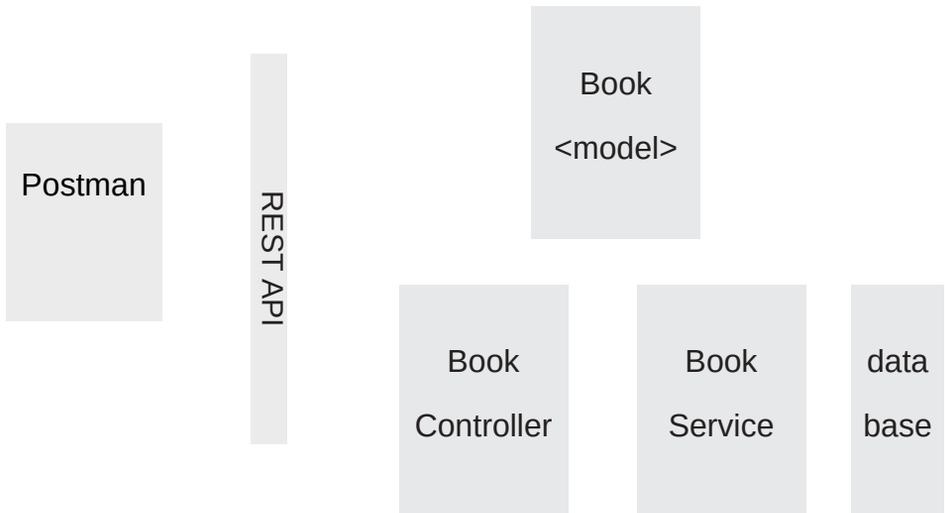
# ORM

Spring Data:  JPA/Hibernate




## Opdracht 1: Getting Started

Genereer een nieuw project op http://start.spring.io
Gebruik de volgende libraries: Web, JPA, H2
Open het project in IDEA.

- Maak de packages: controller, service en model
- Maak een Book, BookController en BookService met onderstaande code.
- Vul de application.properties. (zie de volgende pagina)
- Run de applicatie

```
@Entity
public class Book {
  @Id
  @GeneratedValue(strategy =
      GenerationType.IDENTITY)
  int id;

  String title;
  String isbn;

  // Code - Generate... Getters and Setters
}
```

```
public interface BookService
    extends CrudRepository
        <Book, Integer> {}
```

```
@RestController
public class BookController {
  @Autowired  private BookService
                bookService;

  @PostMapping("/book")
  public Book create(
        @RequestBody Book book) {
    return bookService.save(book);
  }

  @GetMapping("/book")
  public Iterable<Book> f ndAll() {
    return bookService.f ndAll();
  }
}
```

## application.properties

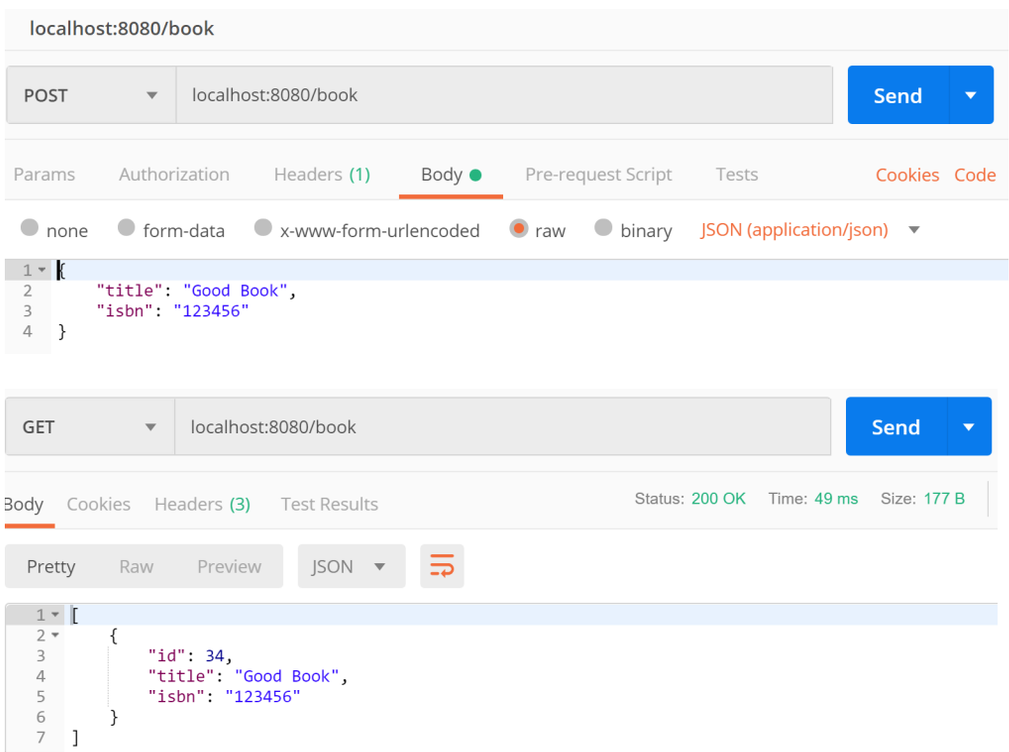
```
# H2
spring.h2.console.enabled=true
spring.h2.console.path=/h2

#sping.datasource.url=jdbc:h2:f le:~/test
spring.datasource.url=jdbc:h2:mem:test
spring.datasource.username=sa
spring.datasource.password=
spring.datasource.driver-class-name=org.h2.Driver
spring.jpa.hibernate.ddl-auto = update
```

## Run

command-line: mvn spring-boot:run
sidebar: Maven - demo - Plugins - spring-boot - spring-boot:run

## Postman




## Opdracht 2: Logging

Voeg de onderstaande regels toe aan **application.properties** en experimenteer met de **ddl-auto** waarde
Wat zijn de effecten? (Herstart de applicatie iedere keer)

**ddl-auto** = **update** of **none** of **create-drop**

```
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.use_sql_comments=true
spring.jpa.properties.hibernate.format_sql=true
```

## Opdracht 3: Author

Voeg een Author, AuthorController en AuthorService toe.

```
@Entity
public class Author {
  @Id
  @GeneratedValue(strategy =
      GenerationType.IDENTITY)
  int id;

  String name;

  // Code - Generate... Getters and Setters
}
```

---

```
public interface AuthorService
    extends CrudRepository
        <Author, Integer> {}
```

```
@RestController
public class AuthorController {
  @Autowired  private AuthorService
                authorService;

  @PostMapping("/author")
  public Author create(
        @RequestBody Author author) {
   return authorService.save(author);
  }

  @GetMapping("/author")
  public Iterable<Author> f ndAll() {
    return authorService.f ndAll();
  }
}
```

## Run

Herstart de applicatie.
Voeg met Postman een author toe (POST /author)
Vraag alle authors aan de database (GET /author)

## Opdracht 4: @ManyToOne

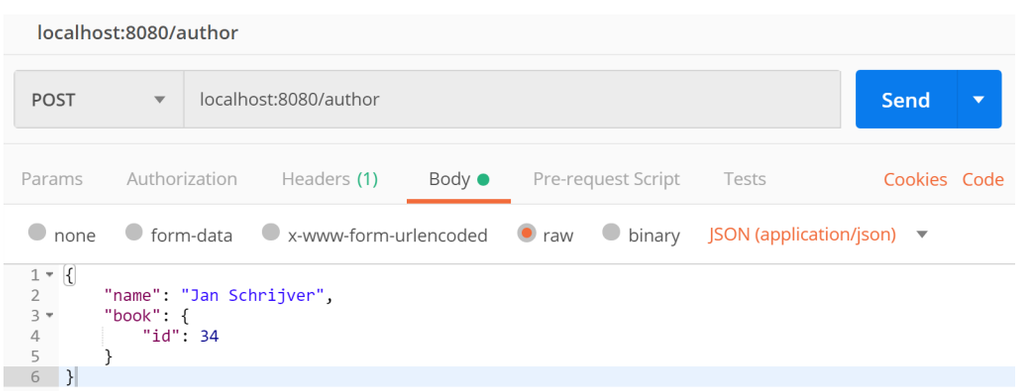
Voeg een @ManyToOne relatie toe aan de Author klasse.

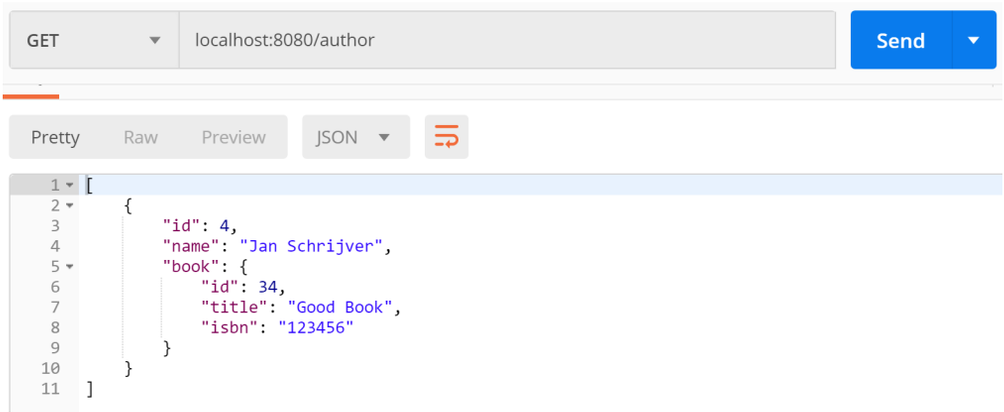
```
@ManyToOne
Book book;

 // Code - Generate... Getters and Setters
```


Herstart de applicatie en bekijk de tabellen in H2 database.

Voeg met Postman een author toe (POST /author)
Vraag alle authors aan de database (GET /author)






## Opdracht 5: @OneToMany en @ManyToOne

```
// In Book
@OneToMany(mappedBy = "book")
List<Author> authors;
```

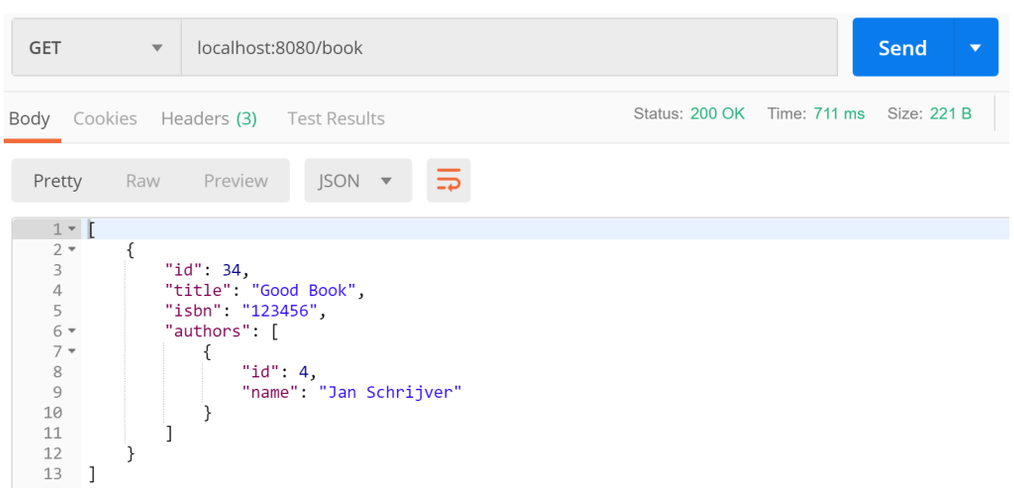

```
 // Code - Generate... Getters and Setters
```

Herstart de applicatie en bekijk de tabellen in H2 database.

Vraag alle boeken aan de database (GET /book)
[Postman komt in een oneindige lus]
Om de oneindige lus op te lossen:

Voeg **@JsonManagedReference** toe aan de List<Author> authors in de Book klasse
Voeg **@JsonBackReference** aan de Book book in de Author klasse

Herstart de applicatie en bekijk de tabellen in H2 database.
Vraag alle boeken uit de database (GET /book) (het gaat nu goed)



## Opdracht 6

In de bovenstaande opdracht heeft een boek meerdere auteurs.
Verander de applicatie waarbij een auteur meerdere boeken heeft.

## Opdracht 7: f ndBy

Voeg de volgende methodes toe aan de BookService

```
Iterable<Book> f ndByTitle(String title);
Iterable<Book> f ndByTitleAndIsbn(String title, String isbn);
```

Voeg de volgende methode toe aan de BookController

```
@PostMapping("/search")
public Iterable<Book> search(@RequestBody Book book) {
  return bookService.f ndByTitle(book.getTitle());
}
```

## Opdracht 8: JPAQL

Vervang de f ndBy methoden door queries in de BookService

```
@Query("select b from Book b where b.title = :title")
Iterable<Book> f ndByTitle(@Param("title") String title);
```

## Opdracht 9: JPAQL Join

Join kan alleen gebruikt worden als er een relatie tussen de tabellen bestaat!
Experimenteer met de onderstaande queries.

```
@Query("select b.authors from Book b where b.title = :title")
Iterable<Author> f ndByTitle(@Param("title") String title);

@Query("select b from Book b join b.authors c where c.name = :name")
Iterable<Book> f ndByAuthorName(@Param("name") String name);
```

## Opdracht 10: Rename

**JSON**

JSON mapping wordt gedaan door de Jackson library.
Jackson heeft een extra annotations (onderstaande en vele andere)

@JsonIgnore

@JsonIgnoreProperties({ "id" })

@JsonRootName(value = "authors")

@JsonProperty("author-name")

**JPA**

Ook in JPA kunnen tabellen en kolommen een andere naam krijgen.

@Entity(name = **"authors"**)

@Column(name = **"author_name"**)

Verander de naam van de tabellen in **books** en **authors**.

In JSON verwijder alle **id**'s uit het bericht.

## Opdracht 11: Hateoas

Voeg spring-data-rest toe aan de POM.xml

```
<dependency>
  <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-data-rest</artifactId>
</dependency>
```

Gooi de controllers weg en ga naar Postman.
En gebruik nu de urls: /books en /authors  b.v. GET /books;  POST /books
En kijk wat er gebeurt als je Postman.

## Opdracht 12: Extra (voor de liefhebbers)

In het bestand **http://www.javawerkplaats.nl/hibernate.pdf** staan nog andere relaties @Embedded en @Inheritance.
Probeer deze eens.