Creating a REST API in Spring Boot involves structuring your application into different layers: the controller, service, and repository layers. Here's a simple example of a REST API for managing "books." I'll break down each part of the code and explain it line by line.

1. Setup

Make sure you have Spring Boot set up in your project. You can create a Spring Boot application using Spring Initializr (https://start.spring.io/) and select the following dependencies:

- **Spring Web**
- **Spring Data JPA**
- **H2 Database (for simplicity)**

2. Project Structure

Your project structure will look like this:

```
css
Copy code
src
└── main
    └── java
        └── com
            └── example
                ├── demo
                │   ├── controller
                │   │   └── BookController.java
                │   ├── service
                │   │   └── BookService.java
                │   ├── repository
                │   │   └── BookRepository.java
                │   ├── model
                │   │   └── Book.java
                │   └── DemoApplication.java
                └── resources
                    └── application.properties
```

3. Model Class

Book.java

```
java
Copy code
package com.example.demo.model;

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

@Entity
public class Book {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String title;
    private String author;

    // Getters and Setters
}
```

- **`@Entity`: Marks this class as a JPA entity.**
- **`@Id`: Specifies the primary key.**
- **`@GeneratedValue`: Indicates that the ID will be generated automatically.**

4. Repository Layer

BookRepository.java

```
java
Copy code
package com.example.demo.repository;

import com.example.demo.model.Book;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface BookRepository extends JpaRepository<Book,
Long> {
```

```
}
```

- `@Repository`: Marks this interface as a repository bean.
- `JpaRepository`: Provides CRUD operations for the `Book` entity.

5. Service Layer

BookService.java

```
java
Copy code
package com.example.demo.service;

import com.example.demo.model.Book;
import com.example.demo.repository.BookRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class BookService {
    @Autowired
    private BookRepository bookRepository;

    public List<Book> findAll() {
        return bookRepository.findAll();
    }

    public Book findById(Long id) {
        return bookRepository.findById(id).orElse(null);
    }

    public Book save(Book book) {
        return bookRepository.save(book);
    }

    public void delete(Long id) {
        bookRepository.deleteById(id);
    }
}
```

- **`@Service`: Marks this class as a service component.**
- **`@Autowired`: Injects the `BookRepository` bean.**
- **Methods:**
  - **`findAll()`: Retrieves all books.**
  - **`findById(Long id)`: Retrieves a book by ID, returns `null` if not found.**
  - **`save(Book book)`: Saves a book.**
  - **`delete(Long id)`: Deletes a book by ID.**

6. Controller Layer

`BookController.java`

```
java
Copy code
package com.example.demo.controller;

import com.example.demo.model.Book;
import com.example.demo.service.BookService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/books")
public class BookController {
    @Autowired
    private BookService bookService;

    @GetMapping
    public List<Book> getAllBooks() {
        return bookService.findAll();
    }

    @GetMapping("/{id}")
    public ResponseEntity<Book> getBookById(@PathVariable Long
id) {
        Book book = bookService.findById(id);
```

```
        return book != null ? ResponseEntity.ok(book) :
ResponseEntity.notFound().build();
    }

    @PostMapping
    public Book createBook(@RequestBody Book book) {
        return bookService.save(book);
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteBook(@PathVariable Long
id) {
        bookService.delete(id);
        return ResponseEntity.noContent().build();
    }
}
```

- `@RestController`: Indicates that this class is a RESTful controller.
- `@RequestMapping("/api/books")`: Sets the base URL for this controller.
- Methods:
  - `getAllBooks()`: Handles GET requests for all books.
  - `getBookById(@PathVariable Long id)`: Handles GET requests for a specific book by ID.
  - `createBook(@RequestBody Book book)`: Handles POST requests to create a new book.
  - `deleteBook(@PathVariable Long id)`: Handles DELETE requests to remove a book by ID.

7. Application Class

`DemoApplication.java`

```
java
Copy code
package com.example.demo;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
@SpringBootApplication
public class DemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}
```

- **`@SpringBootApplication`: Indicates that this is a Spring Boot application.**
- **`main()`: The entry point of the application.**

8. Application Properties

`application.properties`

```
properties
Copy code
spring.h2.console.enabled=true
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
spring.jpa.hibernate.ddl-auto=create-drop
```

- **This configures an in-memory H2 database for development purposes.**

Running the Application

1. **Run the application by executing the `DemoApplication` class.**
2. **Access the API endpoints:**
    - **`GET /api/books` to retrieve all books.**
    - **`GET /api/books/{id}` to retrieve a book by ID.**
    - **`POST /api/books` to create a new book.**
    - **`DELETE /api/books/{id}` to delete a book.**

Conclusion

This is a basic REST API structure in Spring Boot. You can expand upon it with additional features like exception handling, validation, and logging as needed. If you have any questions or need further clarification on any part, feel free to ask!