Exercise 1: Online Bookstore - Setting Up RESTful Services

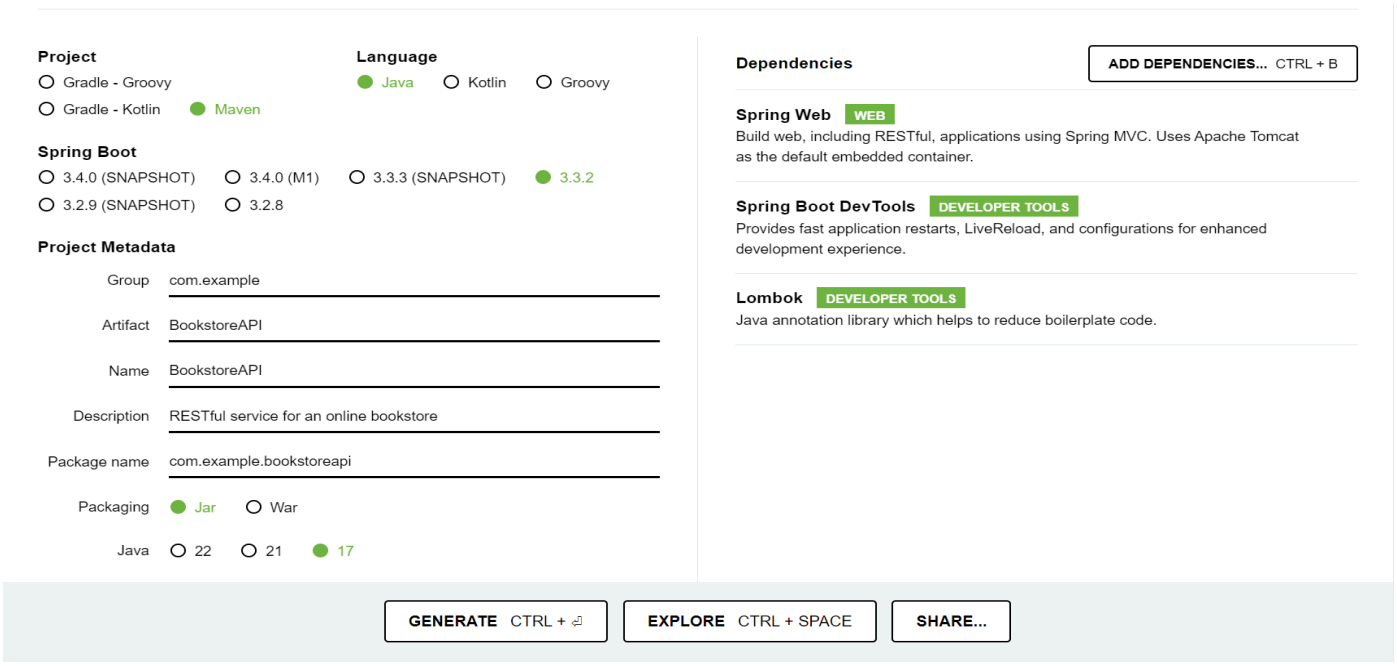
Business Scenario:

You are tasked with developing a RESTful service for an online bookstore. The service will manage books, authors, and customers.

Instructions:

1. **Setup Spring Boot Project:**
    - Initialize a new Spring Boot project named **BookstoreAPI**.
    - Add dependencies: **Spring Web, Spring Boot DevTools, Lombok**.

At first the Springboot project "BookstoreAPI" is setup in the local file using the Spring Initializr and extracted into the Eclipse IDE.



2. **Project Structure:**
    - Familiarize yourself with the generated project structure.

3. **What's New in Spring Boot 3:**
    - Explore and document the new features introduced in Spring Boot 3.

Spring Boot 3.x introduces several new features and improvements. Some of the key highlights include:

1. **Java 17 Support**: Spring Boot 3.x supports the latest LTS version of Java, which includes new language features and performance improvements.
2. **GraalVM Native Image Support**: Improved support for compiling Spring applications to native executables using GraalVM, reducing startup time and memory usage.
3. **Micrometer Observation API**: Introduction of a new observation API in Micrometer to provide a more flexible and powerful way to collect metrics and traces.
4. **Spring GraphQL**: Integration with Spring GraphQL, providing a new way to build GraphQL APIs with Spring Boot.
5. **WebFlux**: Enhancements to Spring WebFlux, including better support for RSocket and improvements in handling backpressure.
6. **Enhanced Security Configurations**: New security features and improvements, including support for multi-tenancy and enhanced OAuth2 configurations.
7. **Configuration Improvements**: Simplified configuration properties and improved support for externalized configuration.

8. **Deprecations and Removals**: Removal of deprecated features and libraries from previous versions, cleaning up the codebase and ensuring better performance and security.

Exercise 2: Online Bookstore - Creating Basic REST Controllers

Business Scenario:

Implement RESTful endpoints to manage books.

Instructions:

1. **Create Book Controller:**
    - Define a **BookController** class with request mappings for /books.

A BookController class is defined in the package com.example.bookstoreapi.controller .

2. **Handle HTTP Methods:**
    - Implement methods to handle **GET**, **POST**, **PUT**, and **DELETE** requests.

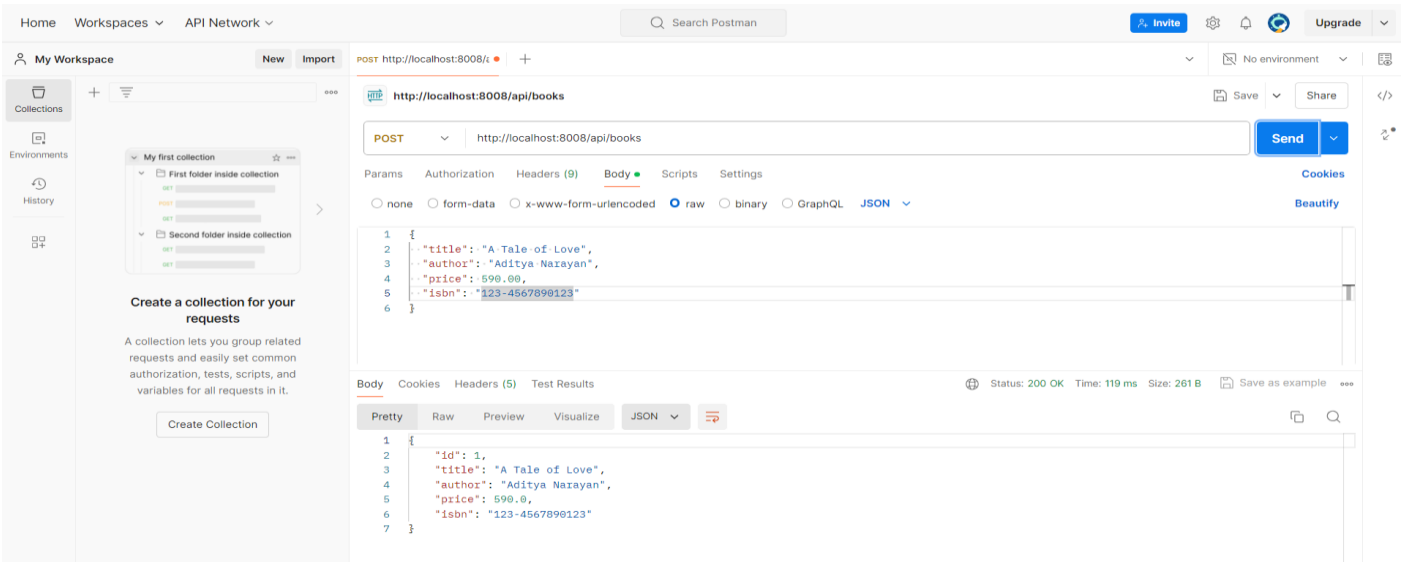
A BookController class is defined in the package com.example.bookstoreapi.controller with CRUD operations methods and REST endpoints. A BookService class in package com.example.bookstoreapi.service id defined to handle the methods.

3. **Return JSON Responses:**
    - Ensure the controller returns JSON responses.
    - Define the Book entity with attributes like **id, title, author, price**, and **isbn**.

A entity class is defined in the package com.example.bookstoreapi.model along with BookRepository interface extending JPA Repository in the package com.example.bookstoreapi.repository. Below all the CRUD oprations are validated using postman and the REST endpoints.

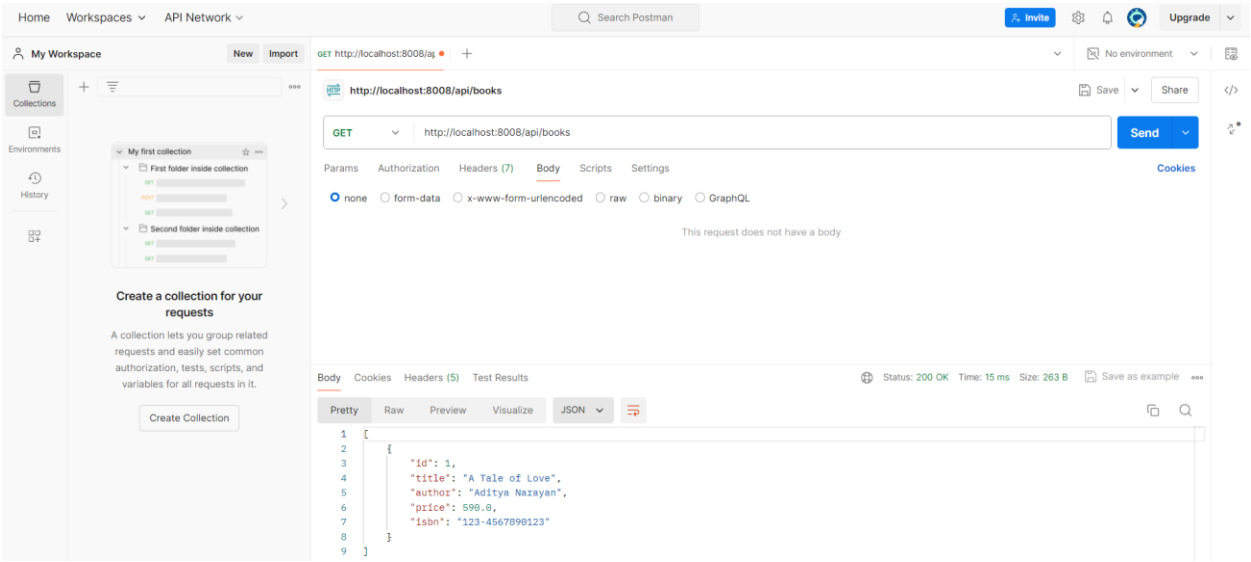
POST /api/books

- **Description: Create a new book.**
- **Method: POST**
- **URL: http://localhost:8008/api/books**



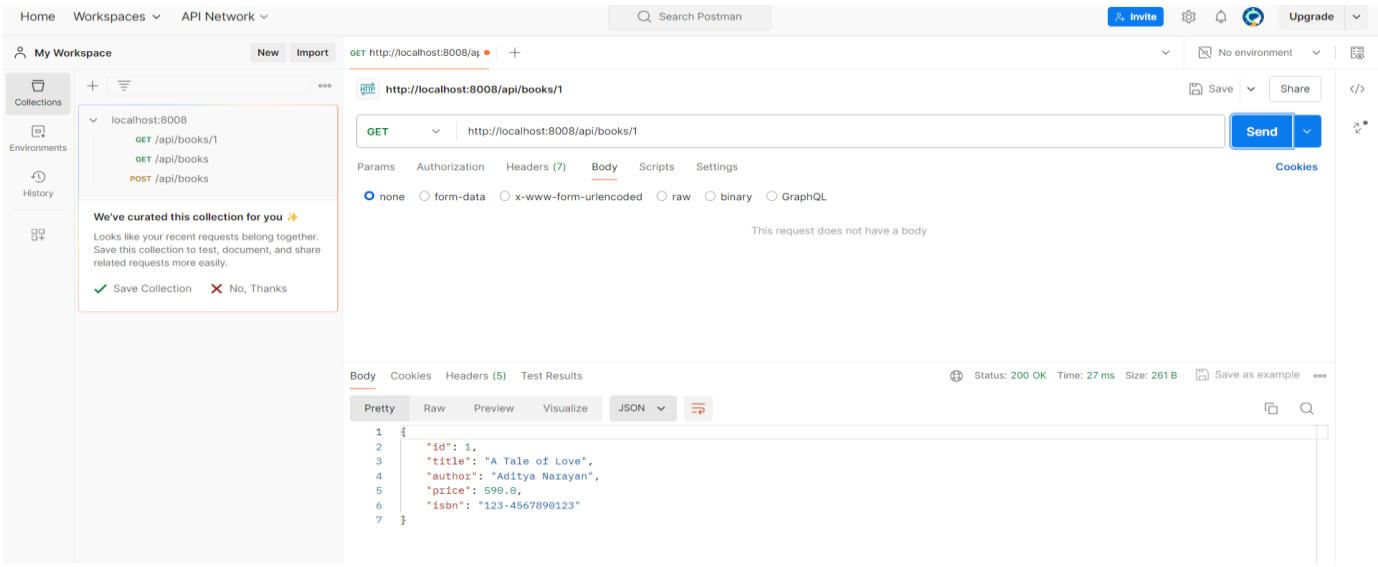
GET /api/books

- **Description: Retrieve all books.**
- **Method: GET**
- **URL: http://localhost:8008/api/books**



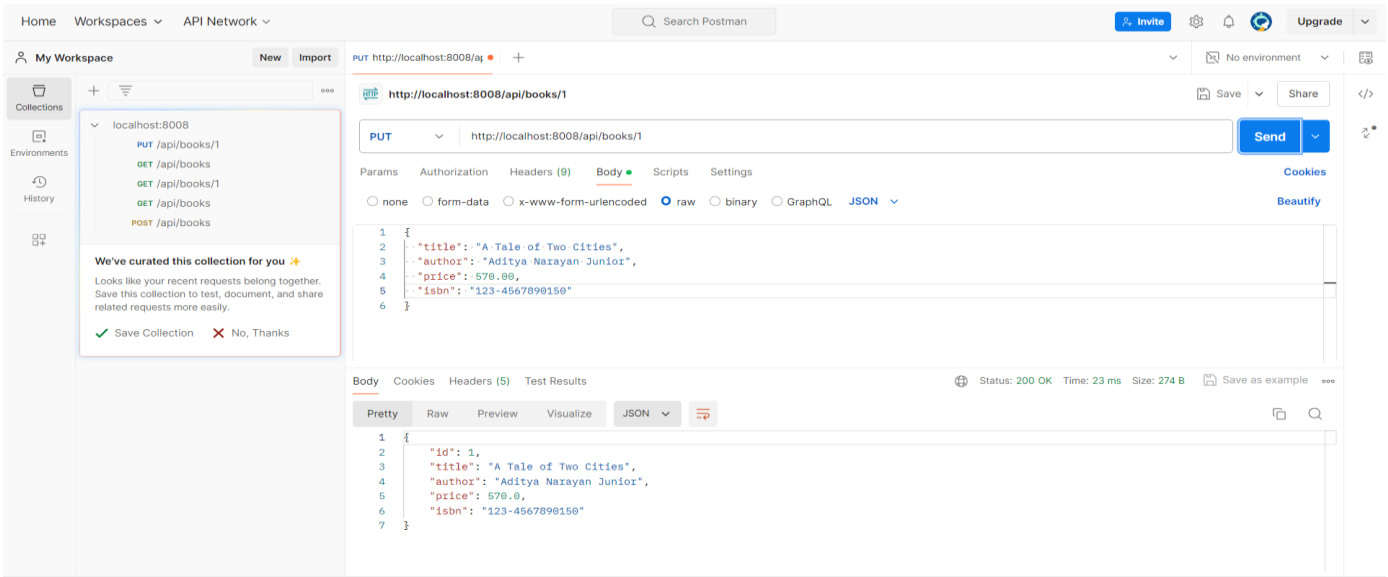
GET /api/books/{id}

- **Description: Retrieve a book by its ID.**
- **Method: GET**
- **URL: http://localhost:8008/api/books/1**



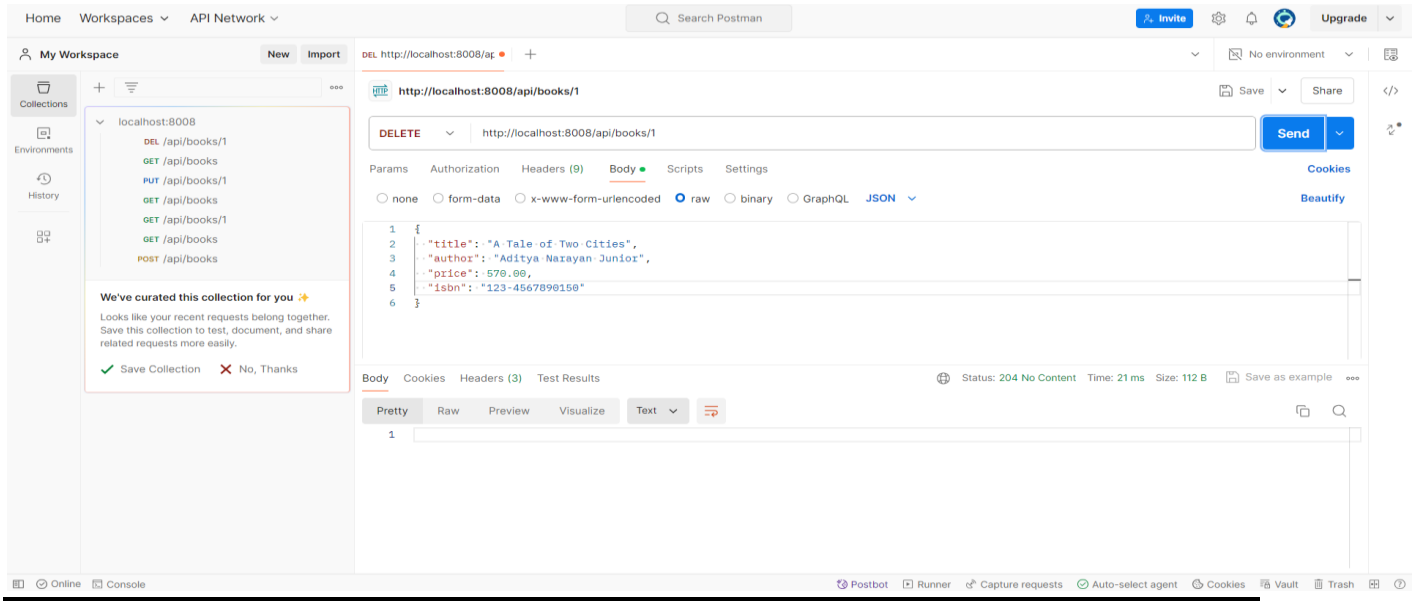
PUT /api/books/{id}

- **Description: Update an existing book.**
- **Method: PUT**
- **URL: http://localhost:8008/api/books/1**



DELETE /api/books/{id}

- **Description: Delete a book by its ID.**
- **Method: DELETE**
- **URL: http://localhost:8008/api/books/1**



Exercise 3: Online Bookstore - Handling Path Variables and Query Parameters

Business Scenario:

Enhance the book management endpoints to handle dynamic URLs and query parameters.

Instructions:

1. **Path Variables:**
    - Implement an endpoint to fetch a book by its ID using a path variable.

A GetMapping Endpoint using @PathVariable is implemented in the BookController class of the package com.example.bookstoreapi.controller. This endpoint fetches a book by its ID.

2. **Query Parameters:**
    - Implement an endpoint to filter books based on query parameters like title and author.

Custom query methods are added to the BookRepository in the package com.example.bookstoreapi.repository. To handle the query filtering logic methods are added in the BookService class of the package com.example.bookstoreapi.service. In the BookController class using a endpoint "/search" all the filter books logic are implemented.

Exercise 4: Online Bookstore - Processing Request Body and Form Data

Business Scenario:

Create endpoints to accept and process JSON request bodies and form data for customer registrations.

Instructions:

1. **Request Body:**

- Implement a POST endpoint to create a new customer by accepting a JSON request body.

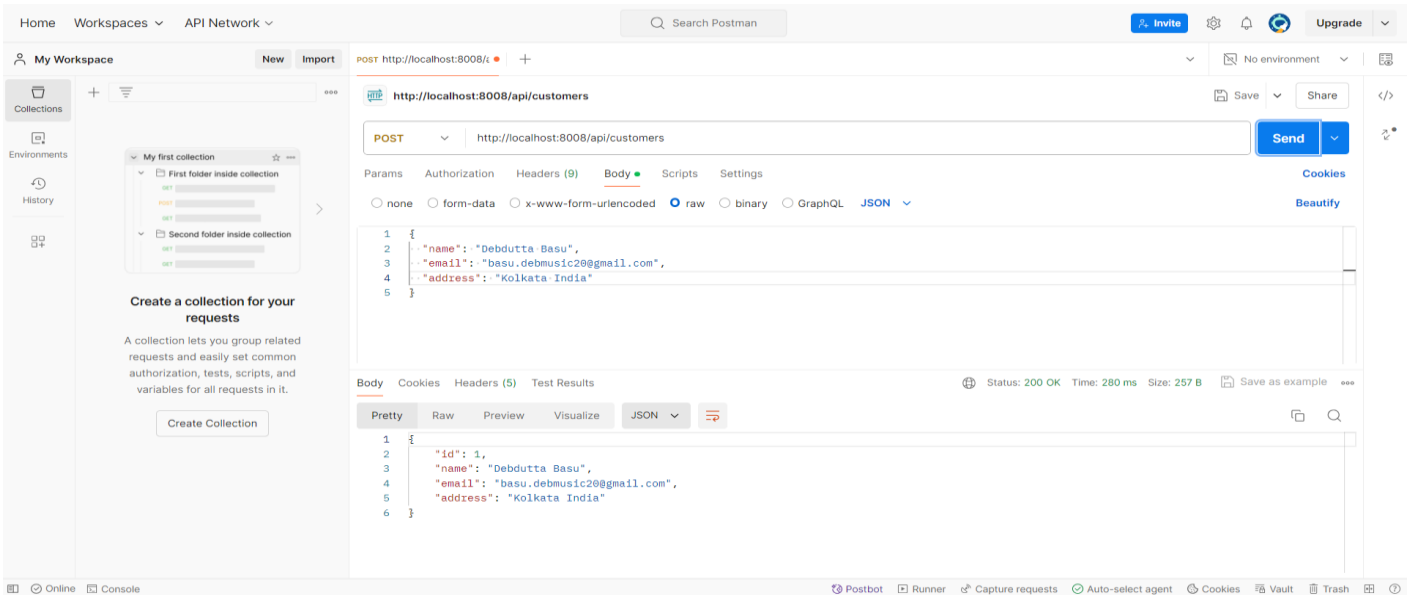
To create a new customer a entity class called Customer is defined in the package com.example.bookstoreapi.model with all the required fields. A JPA Respository with the name CustomerRepository is also defined in the package com.example.bookstoreapi.repository. Then a CustomerService class is also implemented in the package com.example.bookstoreapi.service to create methods for registering customers.

2. **Form Data:**
    - Implement an endpoint to process form data for customer registrations.

Now in the CustomerController class of the package com.example.bookstoreapi.controller the endpoints for creating and registering new user are implemented.To check the workings of the endpoints we will use Postman.

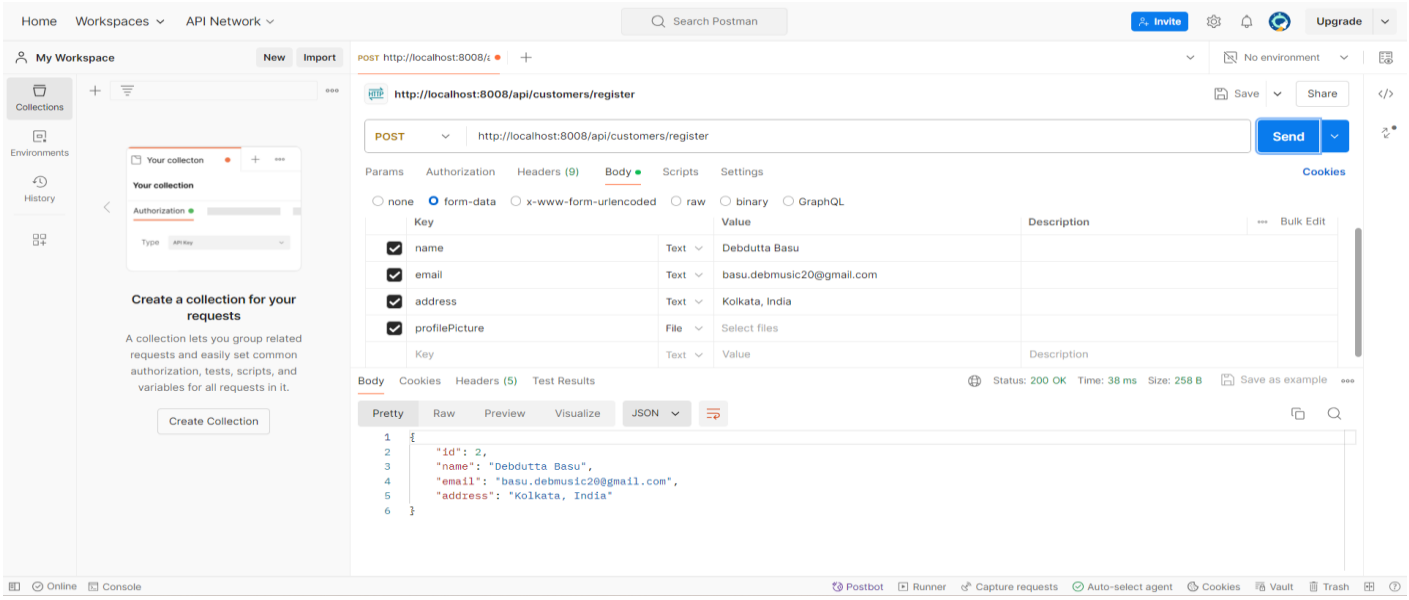
Create Customer with JSON Request Body

- **Method**: POST
- **URL**: http://localhost:8008/api/customers



Register Customer with Form Data

- **Method**: POST
- **URL**: http://localhost:8008/api/customers/register



Exercise 5: Online Bookstore - Customizing Response Status and Headers

Business Scenario:

Customize the HTTP response status and headers for the book management endpoints.

Instructions:

1. **Response Status:**
    - Use **@ResponseStatus** to customize HTTP status codes for your endpoints.

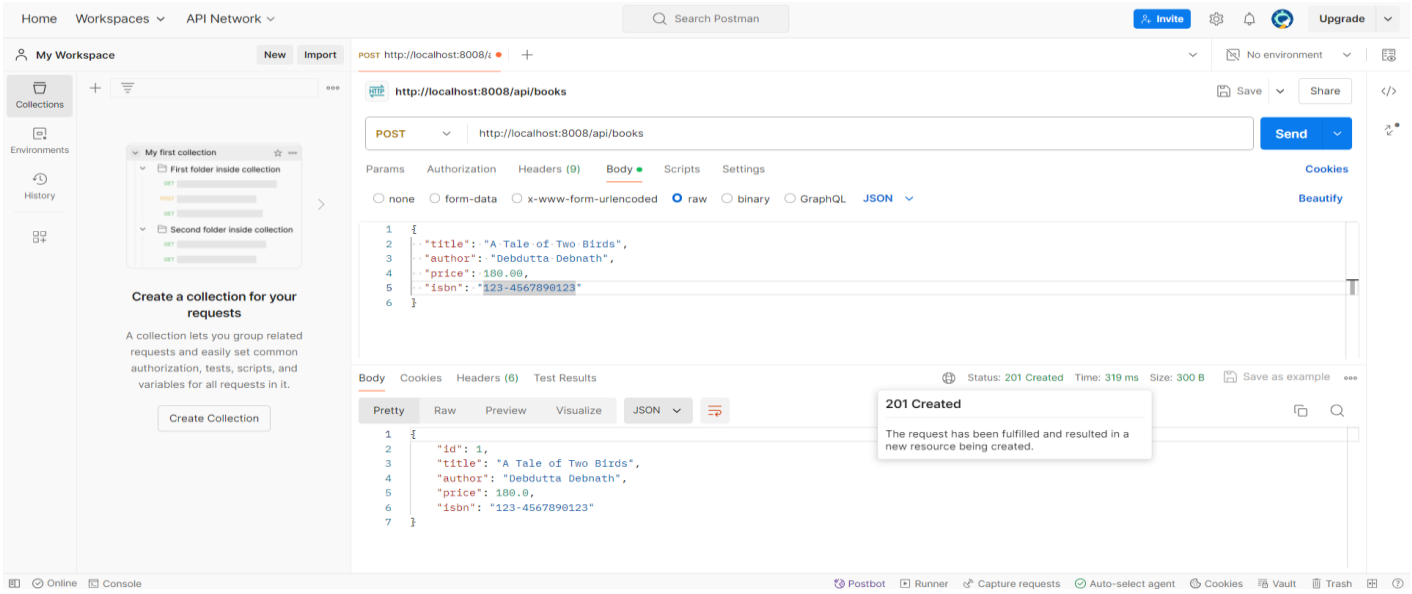
@ResponseStatus has been added to the BookController class of the package com.example.bookstoreapi.controller.

2. **Custom Headers:**
    - Add custom headers to the response using **ResponseEntity**.

Custom header has also been added with the CRUD methods in the BookController class. Now we will check the response status of the endpoints using Postman.

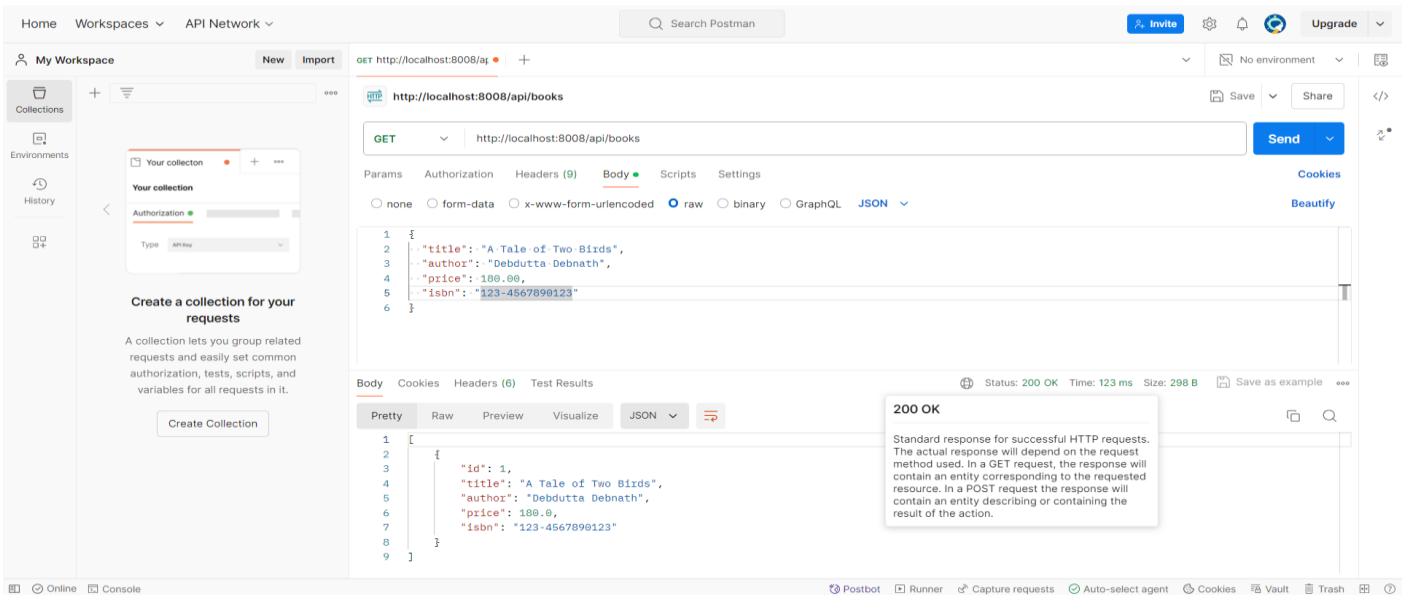
Create Book with Custom Header

- **Method**: POST
- **URL**: http://localhost:8080/api/books



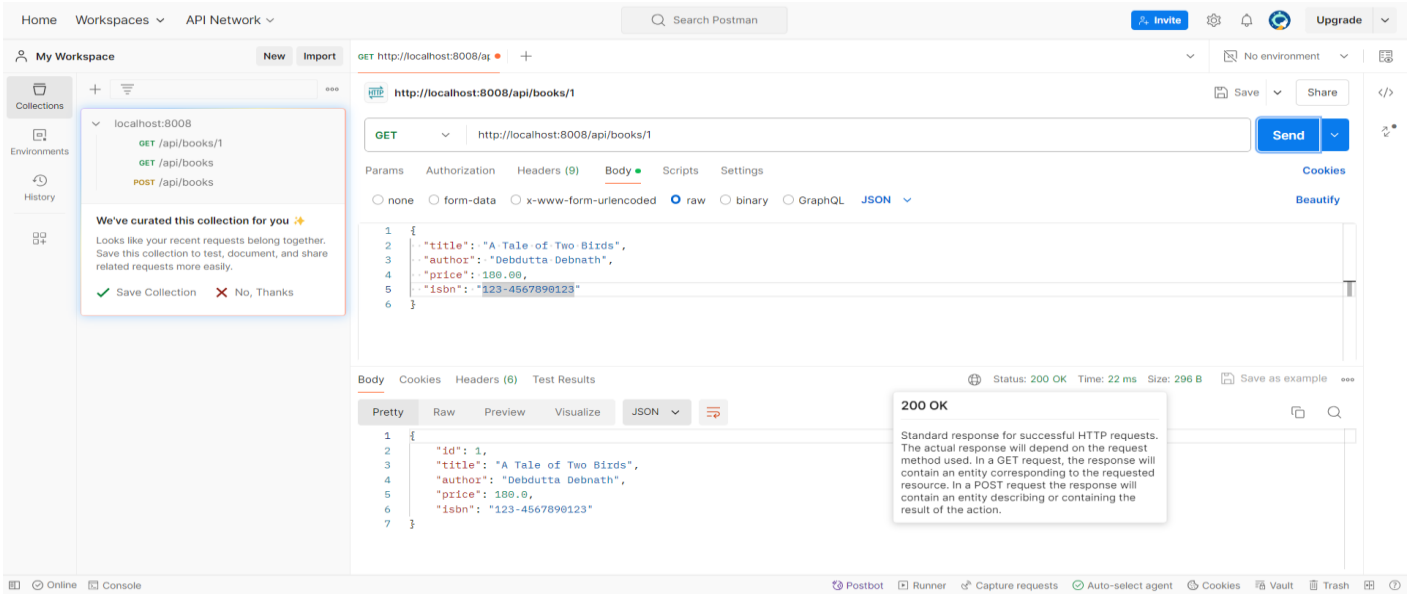
Get All Books with Custom Header

- **Method**: GET
- **URL**: http://localhost:8080/api/books



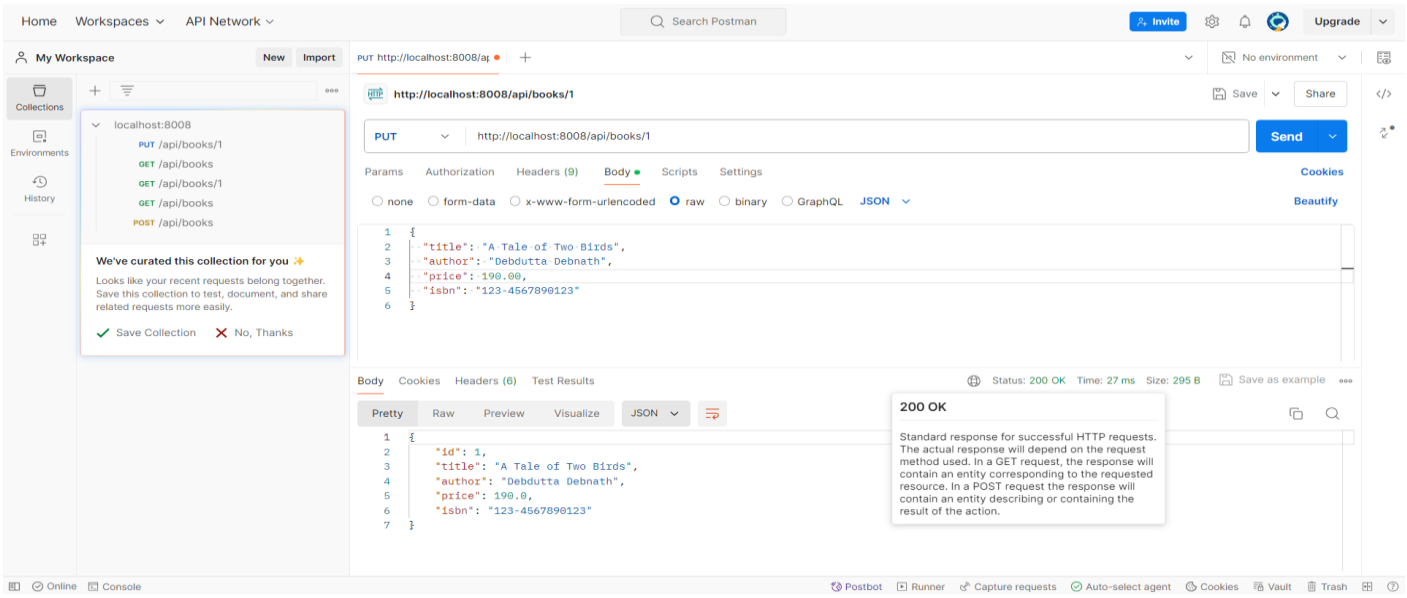
Get Book by ID with Custom Header

- **Method**: GET
- **URL**: http://localhost:8080/api/books/1



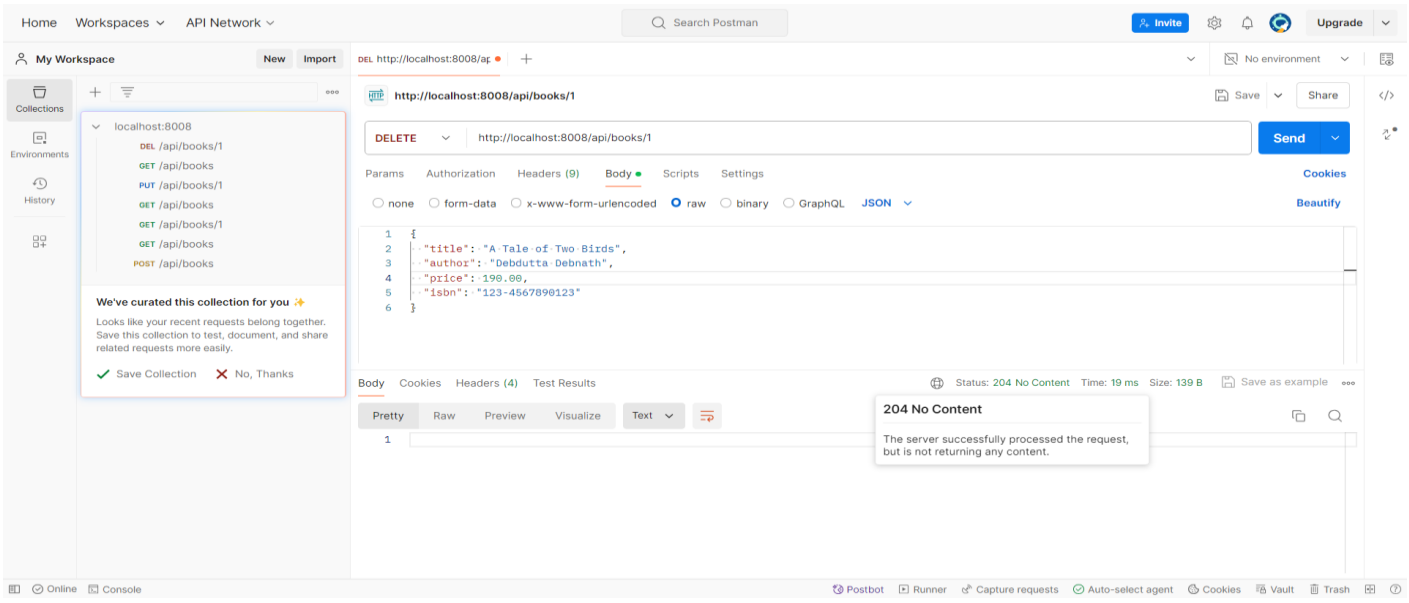
Update Book with Custom Header

- **Method**: PUT
- **URL**: http://localhost:8080/api/books/1



Delete Book with Custom Header

- **Method**: DELETE
- **URL**: http://localhost:8080/api/books/1



Exercise 6: Online Bookstore - Exception Handling in REST Controllers

Business Scenario:

Implement a global exception handling mechanism for the bookstore RESTful services.

Instructions:

1. **Global Exception Handler:**
    - Create a **GlobalExceptionHandler** class using **@ControllerAdvice**.

Created a GlobalExceptionHandler class in the package com.example.bookstoreapi.handler using @ControllerAdvice to handle exceptions globally.

    - Define methods to handle various exceptions and return appropriate HTTP status codes.

Defined custom exception classes (ResourceNotFoundException and InvalidInputException) in the package com.example.bookstoreapi.handler to represent specific error scenarios. Also updated the BookService class to represent those senarios.

Exercise 7: Online Bookstore - Introduction to Data Transfer Objects (DTOs)

Business Scenario:

Use DTOs to transfer data between the client and server for books and customers.

Instructions:

1. **Create DTOs:**
    - Define BookDTO and CustomerDTO classes.

DTOs to Transfer Data Between the Client and Server are formed in the package com.example.bookstoreapi.dto with the class names BookDTO and CustomerDTO to map entities to DTO.

2. **Mapping Entities to DTOs:**
    - Use a library like **MapStruct** or **ModelMapper** to map entities to DTOs and vice versa.

We have defined dependencies in the pom.xml file to use mapping classes. In the package com.example.bookstoreapi.mapper we have defined the MapStruct and ModelMapper class to map DTO entities.

3. **Custom Serialization/Deserialization:**
    - Customize JSON serialization and deserialization using Jackson annotations.

CustomerDTO and BookDTO are defined with JSON serialization.

Exercise 8: Online Bookstore - Implementing CRUD Operations

Business Scenario:

Implement Create, Read, Update, and Delete operations for the Book and Customer entities.

Instructions:

1. **CRUD Endpoints:**
    - Implement endpoints for creating, reading, updating, and deleting books and customers.

REST endpoints for creating, reading, updating, and deleting books and customers are sequentially defined in the package com.example.bookstoreapi.controller in the classes CustomerController and BookController.

2. **Validating Input Data:**
    - Use validation annotations like **@NotNull, @Size**, and **@Min** to validate input data.

Validation annotations are added in the entity class for Book and Customer in the package com.example.bookstoreapi.model.

3. **Optimistic Locking:**
    - Implement optimistic locking for concurrent updates using JPA versioning.

Optimistic locking using the @Version annotation is added in the entity class for Book and Customer in the package com.example.bookstoreapi.model.

Exercise 9: Online Bookstore - Understanding HATEOAS

Business Scenario:

Enhance your REST API to follow HATEOAS principles for navigation through resources.

Instructions:

1. **Add Links to Resources:**
    - Use **Spring HATEOAS** to add links to resources in your API responses.
2. **Hypermedia-Driven APIs:**
    - Build and consume hypermedia-driven APIs.

Exercise 10: Online Bookstore - Configuring Content Negotiation

Business Scenario:

Support different media types (JSON, XML) for your bookstore's RESTful services.

Instructions:

1. **Content Negotiation:**
    - Configure Spring Boot to support content negotiation.
2. **Accept Header:**
    - Implement logic to produce and consume different media types based on the Accept header.

Exercise 11: Online Bookstore - Integrating Spring Boot Actuator

Business Scenario:

Monitor and manage your bookstore's RESTful services using Spring Boot Actuator.

Instructions:

1. **Add Actuator Dependency:**
    - Include the Spring Boot Actuator dependency in your project.
2. **Expose Actuator Endpoints:**
    - Enable and customize Actuator endpoints.
3. **Custom Metrics:**

- Expose custom metrics for monitoring your application.

Exercise 12: Online Bookstore - Securing RESTful Endpoints with Spring Security

Business Scenario:

Secure your bookstore's RESTful endpoints using Spring Security with JWT-based authentication.

Instructions:

1. **Add Spring Security:**
   - Integrate Spring Security into your project.
2. **JWT Authentication:**
   - Implement JWT-based authentication and authorization.
3. **CORS Handling:**
   - Configure CORS to handle cross-origin requests.

Exercise 13: Online Bookstore - Unit Testing REST Controllers

Business Scenario:

Write unit tests for your bookstore's REST controllers using JUnit and Mockito.

Instructions:

1. **JUnit Setup:**
   - Set up JUnit and Mockito in your project.
2. **MockMvc:**
   - Use MockMvc to write unit tests for your REST controllers.
3. **Test Coverage:**
   - Ensure comprehensive test coverage and follow best practices for testing.

Exercise 14: Online Bookstore - Integration Testing for REST Services

Business Scenario:

Write integration tests for your bookstore's RESTful services.

Instructions:

1. **Spring Test:**
    - Set up Spring Test for integration testing.
2. **MockMvc Integration:**
    - Use MockMvc for end-to-end testing of your REST endpoints.
3. **Database Integration:**
    - Include database integration in your tests using an in-memory database like **H2**.

Scenario 15: Online Bookstore - API Documentation with Swagger

Business Scenario:

Document your bookstore's REST APIs using Swagger and Springdoc.

Instructions:

1. **Add Swagger Dependency:**
    - Include Swagger or Springdoc dependencies in your project.
2. **Document Endpoints:**
    - Annotate your REST controllers and methods to generate API documentation.
3. **API Documentation:**
    - Generate and review the API documentation using **Swagger UI** or **Springdoc UI**.