Spring Boot Practical Task for Java Developer

📌 Task Title:

Book Management System with Logical Validation and Testing

📝 Objective:

Build a RESTful web application using **Spring Boot** to manage a bookstore system. The task assesses your ability to:

- Design and develop REST APIs using Spring Boot.
- Apply logical thinking in validation and utility development.
- Implement authentication and authorization.
- Write robust unit and integration tests.
- Handle errors and edge cases gracefully.

🔧 Technical Requirements:

1. Core Features – CRUD Operations

Build the following endpoints for the `Book` entity:

Endpoint	Method	Access	Description
`/books`	GET	user/admin	Retrieve all books
`/books/{id}`	GET	user/admin	Retrieve a single book

`/books`	POST	admin	Create a new book
`/books/{id}`	PUT	admin	Update an existing book
`/books/{id}`	DELETE	admin	Delete a book by ID

---

2. Book Entity Requirements

Each book must have the following fields:

Field	Type	Constraints
id	Long	Auto-generated
title	String	Required, 1–100 characters
author	String	Required, 1–50 characters
publishedDate	LocalDate	Required
genre	String	Optional
price	BigDecimal	Required, must be positive
isbn	String	Required, valid ISBN-10 or ISBN-13

---

3. Logical Validator: ISBN Validation

Implement a utility class that validates whether a given ISBN is valid.

- Valid formats:
    - 10-digit numeric (`1234567890`)
    - 13-digit numeric (`9781234567890`)
- Must not contain letters or special characters.

Method Signature Example:

```
public class BookValidator {
    public static boolean isValidIsbn(String isbn);
```

```
}
```

Write unit tests to verify this logic with valid and invalid examples.

---

4. Security: Role-Based Access Control

Implement **Spring Security** with the following roles and permissions:

Role	Username	Password	Permissions
Admin	admin	admin123	Full access (CRUD)
User	user	user123	Read-only access

- Use **in-memory authentication**.
- Apply **method-level security** using `@PreAuthorize`.

---

5. Database

- Use **Spring Data JPA**.
- Use **H2 in-memory database** for simplicity.
- Seed initial book data (optional).

---

6. Exception Handling

Implement global exception handling using `@ControllerAdvice`.

Handle the following:

- `BookNotFoundException`
- `InvalidBookException`

- Validation errors (JSR-303 annotations)

---

7. Testing

Write **unit and integration tests** using **JUnit 5** and **MockMvc**.

Minimum Tests Required:

- ISBN validation logic
- Book creation with valid/invalid data
- GET (list and single book)
- Access control test (admin vs. user)
- Error cases (book not found, invalid ISBN, unauthorized access)

🧠 Bonus (Optional):

- Add **pagination** to the `GET /books` endpoint using `page` and `size` parameters.
- Add **Swagger** (SpringDoc/OpenAPI) for API documentation.
- Sort books by published date or price.

📂 Project Structure (Suggested):

```
src/
├── main/
│   └── java/com/example/bookstore/
│       ├── controller/
│       ├── service/
│       ├── model/
│       ├── repository/
│       ├── exception/
│       ├── security/
```

```
|       └── util/
├── test/
|   └── java/com/example/bookstore/
|       ├── controller/
|       ├── service/
|       └── validator/
```

🚀 Submission Instructions:

- Submit your project via GitHub link or ZIP file.
- Include a `README.md` with:
    - Setup and run instructions.
    - Sample curl/Postman requests.
    - Swagger URL (if implemented).
- All test cases should be included and pass.

✅ Evaluation Criteria:

Category	Weight
Functional completeness	30%
Code quality & structure	20%
Logical validator accuracy	15%
Test coverage	20%
Security implementation	15%

🛠️ Tools & Technologies

- Java 17+

- Spring Boot 3.x
- Spring Data JPA
- Spring Security
- H2 Database
- JUnit 5
- MockMvc
- Maven or Gradle
- (Optional) Swagger/OpenAPI