🏛️ Exercises to Master RESTful Spring Boot Development

1. Hello World REST API

   - Create a Spring Boot app.
   - Add a `HelloController` with `@RestController`.
   - Expose a GET endpoint:

     ```
     GET /hello → "Hello, World!"
     ```

✅ Practice: project setup, REST basics, annotations.

---

2. Path Variables and Query Parameters

   - Add endpoint:

     ```
     GET /greet/{name}?age=25
     → "Hello Alice, you are 25 years old"
     ```

✅ Practice: `@PathVariable`, `@RequestParam`.

---

3. Basic CRUD with In-Memory List

   - Build a simple `Book` model (`id`, `title`, `author`).
   - Endpoints:

     ```
     GET /books       → list all
     GET /books/{id}  → get by id
     POST /books      → add new
     PUT /books/{id}  → update
     DELETE /books/{id} → remove
     ```

   - Store data in an in-memory `List<Book>`. ✅ Practice: CRUD patterns, `@RequestBody`, HTTP methods.

---

4. Use Spring Data JPA with H2 Database

   - Replace the list with an H2 in-memory DB.
   - Create `BookRepository extends JpaRepository<Book, Long>`.
   - Same endpoints as above, but backed by DB. ✅ Practice: Spring Data JPA, database integration.

---

5. Validation and Error Handling

- Add validation with `@NotBlank`, `@Size`, etc. in `Book`.
- Return 400 Bad Request if validation fails.
- Add a `@ControllerAdvice` to handle exceptions (e.g., `BookNotFoundException` → 404). ✅ Practice: input validation, exception handling.

---

6. Pagination & Sorting

- Extend `GET /books` to support:

  ```
  GET /books?page=0&size=5&sort=title,asc
  ```

✅ Practice: pageable queries with Spring Data JPA.

---

7. DTOs and ModelMapper

- Create `BookDTO` to avoid exposing entity directly.
- Use `ModelMapper` or manual conversion. ✅ Practice: separating API layer from persistence.

---

8. Add Search Endpoint

- Example:

  ```
  GET /books/search?author=Rowling
  ```

- Use Spring Data JPA query methods like `findByAuthorContainingIgnoreCase`. ✅ Practice: custom queries.

---

9. Secure with Spring Security (Basic Auth / JWT)

- Add Spring Security.
- Restrict `POST/PUT/DELETE` to authenticated users.
- Keep `GET` open. ✅ Practice: authentication and authorization.

---

10. Document API with Swagger/OpenAPI

- Add `springdoc-openapi-ui`.
- Visit `/swagger-ui.html` to test endpoints. ✅ Practice: self-documenting REST APIs.

11. Write Unit & Integration Tests

- Use `MockMvc` for controller tests.
- Use H2 + Spring Boot Test for integration. ✅ Practice: testing REST APIs.

—

✅ Solutions for RESTful Spring Boot Exercises

---

1. Hello World REST API

```
@RestController
public class HelloController {
    @GetMapping("/hello")
    public String sayHello() {
        return "Hello, World!";
    }
}
```

---

2. Path Variables and Query Parameters

```
@RestController
public class GreetController {
    @GetMapping("/greet/{name}")
    public String greet(
            @PathVariable String name,
            @RequestParam int age) {
        return "Hello " + name + ", you are " + age + " years old";
    }
}
```

---

3. Basic CRUD with In-Memory List

```
@Data // from Lombok
@AllArgsConstructor
@NoArgsConstructor
class Book {
    private Long id;
    private String title;
    private String author;
```

```
}

@RestController
@RequestMapping("/books")
class BookController {
    private List<Book> books = new ArrayList<>();
    private AtomicLong counter = new AtomicLong();

    @GetMapping
    public List<Book> all() {
        return books;
    }

    @GetMapping("/{id}")
    public Book get(@PathVariable Long id) {
        return books.stream()
                .filter(b -> b.getId().equals(id))
                .findFirst()
                .orElseThrow(() -> new RuntimeException("Book not found"));
    }

    @PostMapping
    public Book add(@RequestBody Book book) {
        book.setId(counter.incrementAndGet());
        books.add(book);
        return book;
    }

    @PutMapping("/{id}")
    public Book update(@PathVariable Long id, @RequestBody Book updated) {
        Book book = get(id);
        book.setTitle(updated.getTitle());
        book.setAuthor(updated.getAuthor());
        return book;
    }

    @DeleteMapping("/{id}")
    public void delete(@PathVariable Long id) {
        books.removeIf(b -> b.getId().equals(id));
    }
}
```

---

4. Spring Data JPA with H2

application.properties

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.h2.console.enabled=true
```

Book.java

```
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Book {
    @Id @GeneratedValue
    private Long id;
    private String title;
    private String author;
}
```

BookRepository.java

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

BookController.java

```
@RestController
@RequestMapping("/books")
public class BookController {
    private final BookRepository repo;

    public BookController(BookRepository repo) {
        this.repo = repo;
    }

    @GetMapping
    public List<Book> all() { return repo.findAll(); }

    @GetMapping("/{id}")
    public Book get(@PathVariable Long id) {
        return repo.findById(id).orElseThrow();
    }

    @PostMapping
    public Book add(@RequestBody Book book) { return repo.save(book); }

    @PutMapping("/{id}")
```

```
    public Book update(@PathVariable Long id, @RequestBody Book updated) {
        Book book = repo.findById(id).orElseThrow();
        book.setTitle(updated.getTitle());
        book.setAuthor(updated.getAuthor());
        return repo.save(book);
    }

    @DeleteMapping("/{id}")
    public void delete(@PathVariable Long id) { repo.deleteById(id); }
}
```

---

5. Validation + Exception Handling

```
@Entity
@Data
public class Book {
    @Id @GeneratedValue
    private Long id;

    @NotBlank(message = "Title is required")
    private String title;

    @NotBlank(message = "Author is required")
    private String author;
}
```

GlobalExceptionHandler.java

```
@RestControllerAdvice
public class GlobalExceptionHandler {
    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<?> handleValidationErrors(MethodArgumentNotValidException ex) {
        Map<String, String> errors = new HashMap<>();
        ex.getBindingResult().getFieldErrors()
                .forEach(e -> errors.put(e.getField(), e.getDefaultMessage()));
        return ResponseEntity.badRequest().body(errors);
    }

    @ExceptionHandler(NoSuchElementException.class)
    public ResponseEntity<?> handleNotFound(NoSuchElementException ex) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body("Book not found");
    }
}
```

---

6. Pagination & Sorting

```
@GetMapping
public Page<Book> all(Pageable pageable) {
    return repo.findAll(pageable);
}
```

Example request:

```
GET /books?page=0&size=5&sort=title,asc
```

---

7. DTO + ModelMapper

BookDTO.java

```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class BookDTO {
    private String title;
    private String author;
}
```

BookController.java

```
private final ModelMapper mapper = new ModelMapper();

@GetMapping("/{id}")
public BookDTO get(@PathVariable Long id) {
    Book book = repo.findById(id).orElseThrow();
    return mapper.map(book, BookDTO.class);
}
```

---

8. Search Endpoint

```
public interface BookRepository extends JpaRepository<Book, Long> {
    List<Book> findByAuthorContainingIgnoreCase(String author);
}

@GetMapping("/search")
public List<Book> search(@RequestParam String author) {
    return repo.findByAuthorContainingIgnoreCase(author);
}
```

---

9. Spring Security (Basic Auth)

`application.properties`

```
spring.security.user.name=admin
spring.security.user.password=secret
```

This protects all endpoints. To allow read-only public access, add config:

```java
@EnableWebSecurity
public class SecurityConfig {
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http.csrf().disable()
            .authorizeHttpRequests()
            .requestMatchers(HttpMethod.GET, "/books/**").permitAll()
            .anyRequest().authenticated()
            .and()
            .httpBasic();
        return http.build();
    }
}
```

---

10. Swagger/OpenAPI

Add dependency (Maven):

```xml
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
    <version>2.0.4</version>
</dependency>
```

Run app → visit:

```
http://localhost:8080/swagger-ui.html
```

---

11. Unit & Integration Tests

`BookControllerTest.java`

```java
@SpringBootTest
@AutoConfigureMockMvc
public class BookControllerTest {
    @Autowired
    private MockMvc mockMvc;
```

```
    @Test
    void testHello() throws Exception {
        mockMvc.perform(get("/hello"))
                .andExpect(status().isOk())
                .andExpect(content().string("Hello, World!"));
    }
}
```

⚡By the end of these exercises, you'll have:

- A fully working REST API
- With persistence, validation, security, documentation, and tests