# Api-Endpoint Test

Writter: Hong Le Nguyen last update : **11.2024**

Code to example on Github :
https://github.com/hong1234/spring-boot3-mvc-jdbc-restApi
https://github.com/hong1234/spring-boot3-mvc-jpa-restApi

test class ---

the Spring Initializr gives you a test class to get started.

```
@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
class MvcJdbcRestApiHApplicationTests {
```

injecting testRestTemplate ---

```
@Autowired
private TestRestTemplate restTemplate;
```

test method ---

```
@Test
@Order(1)
public void testAddBook() {
```

header object

```
HttpHeaders headers = new HttpHeaders();
headers.setBasicAuth("autor", "autor"); // username:password
headers.setAccept(Arrays.asList(MediaType.APPLICATION_JSON));
headers.setContentType(MediaType.APPLICATION_JSON);
return headers;
```

HttpEntity object

```
HttpEntity<Book> request = new HttpEntity<Book>(book, headers);
```

POST request to End Point: --------

```
Book book2 = restTemplate.postForObject(
   getRootUrl()+"/api/books", // URI
   request,                   // HttpEntity object
   Book.class                 // response type
);
```

or

```
ResponseEntity<Book> response = restTemplate.exchange(
   getRootUrl()+"/api/books",
   HttpMethod.POST,          // Http Method
   request,
   Book.class
);
Book book2 = response.getBody();
```

or

```
ResponseEntity<String> response = restTemplate.exchange(
   getRootUrl()+"/api/books",
   HttpMethod.POST,
   request,
   String.class
);
Book book2 = objectMapper.readValue(response.getBody(), Book.class);
```

making assertions against the content of stored book

```
assertThat(book2.getTitle()).isEqualTo("test book1");
```

GET request to End Point: --------

```
HttpHeaders headers = getHeaders();
HttpEntity<?> request = new HttpEntity<>(null, headers);

ResponseEntity<Iterable<Book>> response = restTemplate.exchange(
   getRootUrl()+"/api/books",
   HttpMethod.GET,
   request,
   new ParameterizedTypeReference<Iterable<Book>>(){}
);
Iterable<Book> list = response.getBody();

assertThat(list).hasSize(3);
```

tests ordering -------

```
@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
@TestMethodOrder(OrderAnnotation.class)
class MvcJdbcRestApiHApplicationTests {

   @Test
   @Order(2)
   public void testAddBook2() {
```

run tests --------

**./mvnw test**

# Releasing an Application with Spring Boot

Creating an uber JAR --------

**./mvnw clean package**

It grabs the JAR file originally generated by standard Maven packaging procedures (target/ch7-0.0.1-SNAPSHOT.jar in this case) and extracts all of its content.

With nothing but the JVM, we can launch our application, as follows:

**java -jar target/ch7-0.0.1-SNAPSHOT.jar**

Baking a Docker container --------

Assuming you have installed Docker (visit docker.com to get started) on your machine, this is all it takes!

**./mvnw spring-boot:build-image**

[INFO] Successfully built image 'docker.io/library/ch7:0.0.1-SNAPSHOT'
It's using Docker to build an image named *docker.io/library/ch7:0.0.1-SNAPSHOT*.
This includes the name of our module as well as the version, both found in our pom.xml file.

Using Docker, we can now run the container, as shown here:

**docker run -p 8080:8080 docker.io/library/ch7:0.0.1-SNAPSHOT**

**docker ps** ---> nervous_thompson // The human-friendly name Docker has given this container's instance.

is a command that shows any running Docker processes.

**docker stop nervous_thompson**