



## Spring Boot Test

Spring Boot – Testing Guide  
Spring Boot – @SpringBootTest  
Spring Boot – @WebFluxTest  
Spring Boot – @RestClientTest  
Spring Boot – @DataJpaTest  
Spring Boot –  
@TestConfiguration  
Spring Boot – @MockBean  
[Spring Boot – MockMvc](#)  
Spring Boot – MockMvc Async  
Spring Boot – TestRestTemplate  
Spring Boot – JUnit  
Spring Boot – JUnit 5  
Spring Boot – Test Controller  
Spring Boot – Test Service Layer  
Spring Boot – Integration  
Testing

## Spring Boot 2 Tutorial

Spring Boot – Introduction  
Spring Boot – Starter parent  
Spring Boot – Starter templates  
Spring Boot – Multi-module  
project  
Spring Boot – Annotations  
Spring Boot – Auto  
configuration  
Spring Boot – AOP

# Spring boot MockMVC example

By Lokesh Gupta | Filed Under: [Spring Boot Test](#)

Learn to use **Spring MockMVC** to perform **integration testing** of Spring webmvc controllers. **MockMVC** class is part of [Spring MVC](#) test framework which helps in testing the controllers explicitly starting a Servlet container.

In this MockMVC tutorial, we will use it along with Spring boot's **WebMvcTest** class to execute [JUnit](#) testcases which tests [REST](#) controller methods written for [Spring boot 2 hateoas example](#).

## 1. Maven Dependencies

The `spring-boot-starter-test` dependency includes all required dependencies to create and execute tests.

pom.xml

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
</dependency>
```

If not using spring boot then include following dependencies.

pom.xml

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
```

## Search Tutorials

Type and Press ENTER...

## Compare API Management Vendors

mulesoft.com

### Gartner Magic Quadrant Report

Download Gartner's Magic Quadrant For Full API Lifecycle Management Free!

OPEN

Spring Boot –  
@EnableScheduling  
Spring Boot – Jersey  
Spring Boot – SOAP Webservice  
Spring Boot – SOAP Client  
Spring Boot – JMS Template  
Spring Boot – REST APIs  
Spring Boot – JSP View  
Spring Boot – Actuator  
endpoints  
Spring Boot – Role Based  
Security  
Spring Boot – RSS / ATOM Feed  
Spring Boot – Ehcache 2.x

## Popular Tutorials

Java 14 Tutorial  
Java 8 Tutorial  
Core Java Tutorial  
Collections in Java  
Java Concurrency  
Java Date and Time  
Spring Boot Tutorial  
Spring AOP Tutorial  
Spring MVC Tutorial  
Spring Security Tutorial  
Hibernate Tutorial  
Python Tutorial  
Jersey Tutorial  
Maven Tutorial  
Log4j Tutorial  
Regex Tutorial

- Using `EmployeeRestController.class` as parameter, we are asking to initialize only one web controller and you need to provide remaining dependencies required using `Mock` objects.

## 3. Spring boot MockMvc tests example

Let's create some JUnit tests which will test different HTTP methods in controller class.

### 3.1. HTTP GET

The HTTP APIs are defined in controller are given below. In given tests, we are testing two GET apis – one without path parameter and another with path parameter.

EmployeeRestController.java

```
@GetMapping(value = "/employees")
public EmployeeListVO getAllEmployees()
{
    //code
}

@GetMapping(value = "/employees/{id}")
public ResponseEntity<EmployeeVO> getEmployeeById (@PathVariable("id") int id)
{
    //code
}
```

And corresponding tests for the methods are given below. These tests hit the APIs, pass the path parameters using **MockMvcRequestBuilders** and verify the status response codes and response content using **MockMvcResultMatchers** and **MockMvcResultHandlers**.

TestEmployeeRestController.java

```
@Autowired
private MockMvc mvc;
```

[Spring Boot – Logging](#)  
[Spring Boot – DevTools](#)  
[Spring Boot – WAR Packaging](#)  
[Spring Boot – REST API](#)  
[Spring Boot – CRUD](#)  
[Spring Boot – OAuth2](#)  
[Spring Boot – Testing](#)  
[Spring Boot – RestTemplate](#)  
[Spring Boot – Thymeleaf](#)  
[Spring Boot – Hibernate](#)  
[Spring Boot – DataSource](#)  
[Spring Boot – Error Handling](#)  
[Spring Boot – Caching](#)  
[Spring Boot – Retry](#)  
[Spring Boot – BasicAuth](#)  
[Spring Boot – H2 Database](#)  
[Spring Boot – Ehcache 3.x](#)  
[Spring Boot – Gson](#)  
[Spring Boot – RMI](#)  
[Spring Boot – Send Email](#)  
[Spring Boot – Interview Questions](#)

## Spring Boot Tutorial

[Spring Boot – CommandLineRunner](#)  
[Spring Boot – Configure Jetty](#)  
[Spring Boot – Tomcat Default Port](#)  
[Spring Boot – Context Root](#)  
[Spring Boot – SSL \[https\]](#)  
[Spring Boot – Get all loaded beans](#)  
[Spring Boot – PropertyEditor](#)

```

<version>{version}</version>
<scope>test</scope>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-test</artifactId>
  <version>{version}</version>
  <scope>test</scope>
</dependency>

```

## 2. Test Class Configuration

A JUnit test class to test the Spring MVC controller request and responses, we can use below given configuration.

### Controller Test

```

@RunWith(SpringRunner.class)
@WebMvcTest(EmployeeRestController.class)
public class TestEmployeeRestController {

    @Autowired
    private MockMvc mvc;

}

```

- `SpringRunner` is an alias for the `SpringJUnit4ClassRunner`. It is a custom extension of JUnit's `BlockJUnit4ClassRunner` which provides functionality of the Spring `TestContext` Framework to standard JUnit tests by means of the `TestContextManager` and associated support classes and annotations.
- `@WebMvcTest` annotation is used for Spring MVC tests. It disables full auto-configuration and instead apply only configuration relevant to MVC tests.
- The `WebMvcTest` annotation auto-configure `MockMvc` instance as well.



# Compare API Management Vendors

mulesoft.com

## Gartner Magic Quadrant Report

Download Gartner's Magic Quadrant For Full API Lifecycle Management Free!

OPEN

```
@Test
public void getAllEmployeesAPI() throws Exception
{
    mvc.perform( MockMvcRequestBuilders
        .get("/employees")
        .accept(MediaType.APPLICATION_JSON))
        .andDo(print())
        .andExpect(status().isOk())
        .andExpect(MockMvcResultMatchers.jsonPath("$.employees").exists())
        .andExpect(MockMvcResultMatchers.jsonPath("$.employees[*].employeeId").isNot

}

@Test
public void getEmployeeByIdAPI() throws Exception
{
    mvc.perform( MockMvcRequestBuilders
        .get("/employees/{id}", 1)
        .accept(MediaType.APPLICATION_JSON))
        .andDo(print())
        .andExpect(status().isOk())
        .andExpect(MockMvcResultMatchers.jsonPath("$.employeeId").value(1));
}
```

### 3.2. HTTP POST

The HTTP POST API is defined in controller is:

EmployeeRestController.java

```
@PostMapping(value = "/employees")
public ResponseEntity<EmployeeVO> addEmployee (@Valid @RequestBody EmployeeVO emp]
{
    //code
    return new ResponseEntity<EmployeeVO>(employee, HttpStatus.CREATED);
}
```

And corresponding spring mockmvc test for post json request is as follows:

## TestEmployeeRestController.java

```
@Autowired
private MockMvc mvc;

@Test
public void createEmployeeAPI() throws Exception
{
    mvc.perform( MockMvcRequestBuilders
        .post("/employees")
        .content(asJsonString(new EmployeeVO(null, "firstName4", "lastName4", "email")
        .contentType(MediaType.APPLICATION_JSON)
        .accept(MediaType.APPLICATION_JSON))
        .andExpect(status().isCreated())
        .andExpect(MockMvcResultMatchers.jsonPath("$.employeeId").exists());
    }

    public static String asJsonString(final Object obj) {
        try {
            return new ObjectMapper().writeValueAsString(obj);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
}
```

### 3.3. HTTP PUT

The HTTP API is defined in controller is:

## EmployeeRestController.java

```
@PutMapping(value = "/employees/{id}")
public ResponseEntity<EmployeeVO> updateEmployee (@PathVariable("id") int id, @Val
{
    //code
    return new ResponseEntity<EmployeeVO>(emp, HttpStatus.OK);
}
```

And corresponding tests for the methods are:

TestEmployeeRestController.java

```
@Test
public void updateEmployeeAPI() throws Exception
{
    mvc.perform( MockMvcRequestBuilders
        .put("/employees/{id}", 2)
        .content(asJsonString(new EmployeeVO(2, "firstName2", "lastName2", "email2@n
        .contentType(MediaType.APPLICATION_JSON)
        .accept(MediaType.APPLICATION_JSON))
        .andExpect(status().isOk())
        .andExpect(MockMvcResultMatchers.jsonPath("$.firstName").value("firstName2"))
        .andExpect(MockMvcResultMatchers.jsonPath("$.lastName").value("lastName2"))
        .andExpect(MockMvcResultMatchers.jsonPath("$.email").value("email2@mail.com"
    )
}
```

### 3.4. HTTP DELETE

The HTTP API is defined in controller is:

EmployeeRestController.java

```
@DeleteMapping(value = "/employees/{id}")
public ResponseEntity<HttpStatus> removeEmployee (@PathVariable("id") int id)
{
    //code
    return new ResponseEntity<HttpStatus>(HttpStatus.ACCEPTED);
}
```

And corresponding tests for the methods are:

TestEmployeeRestController.java

```
@Test
```

```
public void deleteEmployeeAPI() throws Exception
{
    mvc.perform( MockMvcRequestBuilders.delete("/employees/{id}", 1) )
        .andExpect(status().isAccepted());
}
```

### 3.5. Test Results

When we run above tests, we get the test results as below.

Finished after 7.148 seconds

Runs: 5/5

Errors: 0

Failures: 0

com.howtodoinjava.demo.TestEmployeeRestController [Runner: JUnit 4] (1.351 s)

Failure Trace

- updateEmployeeAPI (1.119 s)
- getEmployeeByIdAPI (0.137 s)
- deleteEmployeeAPI (0.024 s)
- createEmployeeAPI (0.019 s)
- getAllEmployeesAPI (0.052 s)

#### Test Results

## 4. Summary

In this **spring boot integration test example**, we learned to write *Spring MVC integration tests* using MockMvc class. He learned to write JUnit tests for HTTP GET, POST, PUT and DELETE APIs.

We also looked to verify API response status and response body, pass HTTP headers, and request path parameters as well.

Happy Learning !!

Read More:

[SO Thread](#)

### About Lokesh Gupta

A family guy with fun loving nature. Love computers, programming and solving everyday problems. Find me on [Facebook](#) and [Twitter](#).

## Feedback, Discussion and Comments

Priynka

October 14, 2019

asJsonString is giving error it will give MIME does not contain '/'

[Reply](#)

Parshuram Patil

September 16, 2019

For one of my POST API I am getting such response, what's the possible cause??

MockHttpServletResponse:

Status = 200



Error message = null

Headers = []

Content type = null

Body =

Forwarded URL = null

Redirected URL = null

Cookies = []

[Reply](#)

Raj

July 12, 2019

org.springframework.web.util.NestedServletException: Request processing failed;  
nested exception is java.lang.ClassCastException:  
org.springframework.web.servlet.resource.ResourceHttpRequestHandler cannot be  
cast to org.springframework.web.method.HandlerMethod

I am getting this error, any idea what is wrong?

[Reply](#)

chandrajeet

July 18, 2019

Do you have any solution? I am getting the same exception

[Reply](#)

Baiju damodar

May 6, 2019

Hey , can we mock methods using MockMVC , if no How can we mock a method with the similar approach ?? Pleas do a session on that as well . A typical example of a method call "addValues(int a, int b)" whichj the adds , subtracts , multiplies, and divides a with b , and combines the entire result as one string and returns it.

[Reply](#)

adnsbd

April 10, 2019

Do you need to running service first to do Spring boot MockMvc tests ?

I got assertion error on line "status().isOk()"

[Reply](#)

sobhan

January 30, 2019

asJsonString is giving error

[Reply](#)

Lokesh Gupta

January 30, 2019

What error you are facing. I hope you have defined the method as given in post :

```
public static String asJsonString(final Object obj) {  
    try {  
        return new ObjectMapper().writeValueAsString(obj);  
    } catch (Exception e) {  
        throw new RuntimeException(e);  
    }  
}
```

[Reply](#)

## Ask Questions & Share Feedback

Please do not submit a comment only to say "Thank you".

Comment

\*Want to Post Code Snippets or XML content? Please use [java] ... [/java] tags otherwise code may not appear partially or even fully. e.g.

```
[java]  
public static void main (String[] args) {  
    ...
```

```
}  
[/java]
```

Name \*

Email \*

Website

POST COMMENT

### Meta Links

[Advertise](#)  
[Contact Us](#)  
[Privacy policy](#)  
[About Me](#)

### Recommended Reading

[10 Life Lessons](#)  
[Secure Hash Algorithms](#)  
[How Web Servers work?](#)  
[How Java I/O Works Internally?](#)  
[Best Way to Learn Java](#)  
[Java Best Practices Guide](#)  
[Microservices Tutorial](#)

Copyright © 2016 · HowToDoInJava.com · All Rights Reserved. | [Sitemap](#)