1. **Properly testing Spring MVC controllers**

**December 11, 2012**Written by [**Tom van Zummeren**](https://blog.trifork.com/author/tom/)

[**12 Comments**](https://blog.trifork.com/2012/12/11/properly-testing-spring-mvc-controllers/#comments)

In this post I want to introduce you to the Spring MVC testing framework, a way to properly test your Spring MVC controllers.

1. The way I used to do it

I have always found unit testing Spring MVC controllers nearly useless because I felt they don’t test anything useful for the most part. If something would break in the implementation, those unit tests would rarely fail. To explain what I mean by this, let me give you an example. Say you want to unit test the following controller method:

|  |  |
| --- | --- |
| 1 | @RequestMapping(value = "/people/{groep}", method = RequestMethod.GET) |
| 2 | public String listPeopleInGroup(@PathVariable String group, ModelMap modelMap) { | |

|  |  |  |
| --- | --- | --- |
| 3 | List people = peopleService.listPeople(group); | |
| 4 | modelMap.put("people", people); |

|  |  |  |
| --- | --- | --- |
| 5 | return "peopleList"; | |
| 6 | } |

All this controller method does is take the “group” parameter from the URL and list all people who are in that group. But **it has a bug in it** I didn’t notice. Usually you would unit test this controller by just mocking the peopleService using EasyMock or Mockito, pass in a fake Model and off you go! This is what the unit test in that case typically would look like (using JUnit, Mockito and Hamcrest):

|  |  |  |
| --- | --- | --- |
| 1 | public class PeopleControllerTest { | |
| 2 |  |

|  |  |
| --- | --- |
| 3 | @InjectMocks |
| 4 | PeopleController controller; | |

|  |  |
| --- | --- |
| 5 |  |
| 6 | @Mock | |

|  |  |  |
| --- | --- | --- |
| 7 | PeopleService mockPeopleService; | |
| 8 |  |

|  |  |  |
| --- | --- | --- |
| 9 | @Before | |
| 10 | | public void setUp() throws Exception { | |

|  |  |  |
| --- | --- | --- |
| 11 | MockitoAnnotations.initMocks(this); | |
| 12 | } |

|  |  |
| --- | --- |
| 13 |  |
| 14 | @Test | |

|  |  |
| --- | --- |
| 15 | public void testListPeopleInGroup() { |
| 16 | List expectedPeople = asList(new Person()); | |

|  |  |  |  |
| --- | --- | --- | --- |
| 17 | when(mockPeopleService.listPeople("group")).thenReturn(expectedPeople); | | |
| 18 | |  |

|  |  |
| --- | --- |
| 19 | ModelMap modelMap = new ModelMap(); |
| 20 | String viewName = controller.listPeopleInGroup("group", modelMap); | |

|  |  |
| --- | --- |
| 21 |  |
| 22 | assertEquals("peopleList", viewName); | |

|  |  |  |
| --- | --- | --- |
| 23 | assertThat(modelMap, hasEntry("people", (Object) expectedPeople)); | |
| 24 | } |

|  |  |
| --- | --- |
| 25 | } |

Let’s run this unit test …

Afbeelding met schermafbeelding

Automatisch gegenereerde beschrijving  
Green light! Wow that’s amazing, this controller works like a charm! But oh wait, let me try to run this in a browser now.

Afbeelding met schermafbeelding

Automatisch gegenereerde beschrijving

1. What did we test?

Hmm, ok something obviously went wrong here. Why did the unit test not cover this? Well, as I mentioned before: it’s useless! Because what did it test exactly?

* A call to the peopleService is made
* The return value from the peopleService is put inside the ModelMap
* The correct view name is returned

1. What didn’t we test?

Some examples of things the unit test didn’t cover:

* That it responds to the correct url
* That the PathVariable correctly takes the value from the “{group}” part in the url
* That the method accepts GET requests
* That any class-level annotations are also working as you expect
* That any handler interceptors are called before and/or after executing this method
* That any other methods in the controller are called beforehand, like those  
  annotated with @ModelAttribute or @InitBinder.
* That binding from parameters to an object works the way you expect
* That any validation errors were registered by validators

1. Introducing the Spring MVC test framework

Recently a colleague of mine pointed me towards a little framework created as part of the Spring Framework called [**spring-test-mvc**](https://github.com/SpringSource/spring-test-mvc). This framework makes unit testing controllers a lot more meaningful! It actually has the ability to test all the aspects of a controller method I couldn’t test before. Let’s see if I can reproduce the above bug using Spring Test MVC.

|  |  |  |
| --- | --- | --- |
| 1 | public class PeopleControllerTest { | |
| 2 |  |

|  |  |
| --- | --- |
| 3 | @InjectMocks |
| 4 | PeopleController controller; | |

|  |  |
| --- | --- |
| 5 |  |
| 6 | @Mock | |

|  |  |  |
| --- | --- | --- |
| 7 | PeopleService mockPeopleService; | |
| 8 |  |

|  |  |  |
| --- | --- | --- |
| 9 | @Mock | |
| 10 | | View mockView; | |

|  |  |
| --- | --- |
| 11 |  |
| 12 | MockMvc mockMvc; | |

|  |  |
| --- | --- |
| 13 |  |
| 14 | @Before | |

|  |  |
| --- | --- |
| 15 | public void setUp() throws Exception { |
| 16 | MockitoAnnotations.initMocks(this); | |

|  |  |  |
| --- | --- | --- |
| 17 | mockMvc = standaloneSetup(controller) | |
| 18 | .setSingleView(mockView) |

|  |  |  |
| --- | --- | --- |
| 19 | .build(); | |
| 20 | } |

|  |  |
| --- | --- |
| 21 |  |
| 22 | @Test | |

|  |  |  |
| --- | --- | --- |
| 23 | public void testListPeopleInGroup() throws Exception { | |
| 24 | List expectedPeople = asList(new Person()); |

|  |  |  |  |
| --- | --- | --- | --- |
| 25 | when(mockPeopleService.listPeople("someGroup")).thenReturn(expectedPeople); | | |
| 26 | |  |

|  |  |  |
| --- | --- | --- |
| 27 | mockMvc.perform(get("/people/someGroup")) | |
| 28 | .andExpect(status().isOk()) |

|  |  |  |
| --- | --- | --- |
| 29 | .andExpect(model().attribute("people", expectedPeople)) | |
| 30 | .andExpect(view().name("peopleList")); |

|  |  |  |
| --- | --- | --- |
| 31 | } | |
| 32 | } |

The above unit test almost needs no explanation. It reads like reading a book. We perform a get request, check if the status is OK (status 200), check if a model attribute exists named “people” and check if the view name is “peopleList”. Now let’s run this unit test …

Afbeelding met schermafbeelding

Automatisch gegenereerde beschrijving

Yes we did it! We are now essentially seeing the same error message as we saw in the browser. So what mistake could I have made in my controller method that a ‘normal’ unit test failed to cover? Let’s take another quick look at the code.

|  |  |
| --- | --- |
| 1 | @RequestMapping(value = "/people/{groep}", method = RequestMethod.GET) |
| 2 | public String listPeopleInGroup(@PathVariable String group, ModelMap modelMap) | |

Oh yes, of course! I made a typo in the path variable inside the url string. Of course “groep” doesn’t match the method argument named “group” so the request fails. Let’s correct my mistake.

|  |  |
| --- | --- |
| 1 | @RequestMapping(value = "/people/{group}", method = RequestMethod.GET) |
| 2 | public String listPeopleInGroup(@PathVariable String group, ModelMap modelMap) | |

Run it again …

Afbeelding met schermafbeelding

Automatisch gegenereerde beschrijving

This proves my point. A unit test written using Spring Test MVC closely resembles a real request made by a browser. That’s why this test is a lot more meaningful. The fact that this test is now passing makes it very likely that it will work in a browser as well. The fact that the other unit test passed, didn’t tell us anything.

1. Testing a REST interface

Spring Test MVC is especially useful when building a REST interface which for example returns JSON responses. The framework contains all sorts of stuff for easily building a test request and carefully examining the response.

Let’s say we have a controller that listens to the URL “/people” and adds a person to the database whenever a POST request was received on that URL. The **request body** would be a JSON representation of the person to add. The **response body** will hold JSON that tells us what the database identifier of the person is that was just added, and also gives us a list of all people that have been added so far. This is how you would test this:

|  |  |
| --- | --- |
| 1 | mockMvc.perform(post("/people") |
| 2 | .contentType(MediaType.APPLICATION\_JSON) | |

|  |  |  |
| --- | --- | --- |
| 3 | .body("{\"firstName\":\"Tom\", \"lastName\":\"van Zummeren\"}".getBytes())) | |
| 4 | .andExpect(status().isCreated()) |

|  |  |
| --- | --- |
| 5 | .andExpect(jsonPath("$.identifier", equalTo("123"))) |
| 6 | .andExpect(jsonPath("$.allPeople[\*].firstName", hasItem("Tom"))); | |

|  |  |
| --- | --- |
| 7 |  |
| 8 | verify(mockPeopleService).persistPerson(new Person("Tom", "van Zummeren")); | |

In this example we see a few new things. The content type is set on the request, a request body is given, and the response is checked using an expression language provided by the “json-path” framework.

This test tests so many things now. It’s not just calling the controller method, it’s also testing:

* The mapping of the request body to a Person object
* The status code that was set on the response
* That it supports the given content type
* That it listens to a POST on the /people URL
* The mapping of the response object is transformed correctly to JSON

1. Conclusion

Spring Test MVC is indispensable if you want to test your Spring MVC controllers. Simply testing the controller methods without including the Spring MVC framework itself, is useless. Spring Test MVC will be included in the Spring 3.2 release (so I’m told) but for now it can be found on Github: <https://github.com/SpringSource/spring-test-mvc>

[Java](https://blog.trifork.com/category/development/java/), [Software Development](https://blog.trifork.com/category/development/), [Spring](https://blog.trifork.com/category/development/spring/)[Spring](https://blog.trifork.com/tag/spring/), [Spring MVC](https://blog.trifork.com/tag/spring-mvc/), [Testing](https://blog.trifork.com/tag/testing/)