Backend testing Asp.Net

By Thomas Pouw

Table of Contents

[2. Backend Testing (please change name) 2](#_Toc104901337)

[3. ASP.Net testing 2](#_Toc104901338)

[4. Conclusion 3](#_Toc104901339)

# 2. Why do we test?

The reason why testing is so important is because programmers do not want the customer/user to be inconvenienced by problems easily prevented by the programmer. In 1997 Kent Beck had the same problem. How could he know that his code was safe in the smallest time possible and so he made JUNIT, a unit testing framework for Java. Before JUNIT’s creation debugging and testing were the same thing. Just manual test it, but afterward testing is done by the program itself. By giving the program the parameters, it normally gets and predicting what the outcome should be.

# 3. ASP.Net testing

In ASP.NET we work with Moq and FluentAssertions. FluentAssertions is the framework while Moq is the library that handles mocking certain methods. You can see the mocking of methods as a kind of replacement. Where it has the same name as the original, but it just returns the thing you just ordered it to return. So, if normally the method “Say\_1” returns 1. You can say with Moq that it returns 2 now. Of course, you still need to remember what the normal return field is. As you can not just Moq return an object if the normal return is a string. Its still just pretending that it is that method it does not truly replace it. And so, you also must play with the rules of that method. It also means that if the method is an async one you have to say that the return has to be an async return. As if you do not do that Moq will not know what to do as it returns to quickly then and because async methods have a special return field called Task. And so, it would not even work if the problem was not the whole async/await thing.

To check if you get the right answer you use .Should() from the AssertionExtensions section of FluentAssertions. As it just says that something is being compared to something else and is the basis on what the rest is built on. Without .Should() you cannot do anything else. It can be that you need to check if an array is correct or you just want to check if an object has a certain condition like sorted then you use a method of the .[Have()](https://fluentassertions.com/collections/) line. If you want to check if a certain item is in an array, you should use [.Contain()](https://fluentassertions.com/collections/) . And if you just have one item to check and you just have one thing returning you use the [.Be()](https://fluentassertions.com/objectgraphs/) method.

For exception testing I recommend crashing that method manually. So, you do not mock the exception. But you just give the method wrong information you know is invalid and know that it errors if you do. So, for example if you have a method that updates by firstly getting the item out of the database you can give it a non-existing ID for it to look up. And when it tries to put the new items in that object it will give a null reference error. As that object will be counted as null. As it could not be found by the database.  
And if you return the message of the error, you can also check for that by checking for the value of the error. As in NonFoundObjectResult from IActionResult the return value is in value. But only if you put the message in the error method. But of course, there are other ObjectResult objects. As there is also OkObjectResult, ExceptionResult, ConflictObjectResult, BadRequestObjectResult and AcceptedResult. Of course, there are more and you will find it [here](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.objectresult?view=aspnetcore-6.0).

There is also Assert by NUnit. With Assert you can do everything that Moq does without the mocking. So, you just use the normal methods in there intended uses. It has it upsides and downsides. As its better if you do not know the precise return of a methods because when its randomized. But the downside is that this is very bad when you have any database things linked to that function. If it is just a get its fine. But any add, update, or delete function will be a nightmare. But only if you do not mock your database.

# 4. Conclusion

We learned of how Moq, FluentAssertions and NUnit work and why it is so important to test. As it is needed to make sure that our software is safe to use. As even if glitches and bugs can be fun, it is not fun when that glitch or bug happens while you are doing anything super important like filling in your taxes online. So, it is always a good idea to test and debug your code as good as possible. Of course, you cannot catch every single bug. But if you just catch 90% of it is already good enough.