

# The basics of Integration Tests with net 5 and xUnit (compatible with net core x)

Author: Guilherme Salvi

Created: 29/12/2020

## Introduction

Integration tests ensure the application works at a higher level than unit tests, including the application infrastructure, often from a Controller (in the case of applications that follow the MVC architectural pattern) with HTTP requests, passing through all components up to the database (but not limited to the database, depending on the application's proposal).

Often we do not want integration tests to directly access the database due to slowness or unavailability in test scenarios, in which case we can implement an in-memory version of the database.

## First configuration

We must first create a test project separate from our main application using the following command:

```
PS> dotnet new xunit -f net5 -n IntegrPoc.IntegrationTests -o ./tests/IntegrPoc.UnitTests/
```

Where the *-n* parameter corresponds to the project name and the *-o* parameter corresponds to the output directory.

Now let's create a new directory called *Configuration* and inside it add a new class called *IntegrPocAppFactory* like this:



```

1  using Microsoft.AspNetCore.Hosting;
2  using Microsoft.AspNetCore.Mvc.Testing;
3
4  namespace IntegrPoc.IntegrationTests.Configuration
5  {
6      public class IntegrPocAppFactory<TStartup> : WebApplicationFactory<TStartup>
7          where TStartup : class
8      {
9          protected override void ConfigureWebHost(IWebHostBuilder builder)
10         {
11             builder.UseStartup<TStartup>();
12             builder.UseEnvironment("Testing");
13         }
14     }
15 }

```

Note that at this moment we have a generic parameter *TStartup* that will later be replaced by our concrete class *Startup*.

Another class that we must add within the *Configuration* directory is a class called *IntegrationTestsApiFixtureCollection* so that it is shared by all tests.



```

1  using Microsoft.AspNetCore.Hosting;
2  using Microsoft.AspNetCore.Mvc.Testing;
3
4  namespace IntegrPoc.IntegrationTests.Configuration
5  {
6      public class IntegrPocAppFactory<TStartup> : WebApplicationFactory<TStartup>
7          where TStartup : class
8      {
9          protected override void ConfigureWebHost(IWebHostBuilder builder)
10         {
11             builder.UseStartup<TStartup>();
12             builder.UseEnvironment("Testing");
13         }
14     }
15 }
16

```

Note that in line 12 we are defining a "Testing" environment, but this can be replaced by an environment of your choice such as "Production", "Development", etc.

## Writing the integration tests

Now we can write our integration tests.

For a Controller and an Action like this:

```
1 namespace IntegrPoc.Api.Controllers
2 {
3     [ApiController]
4     [Route("api/customer")]
5     public class CustomerController : ControllerBase
6     {
7         private readonly ICustomerService _service;
8
9         public CustomerController(ICustomerService service)
10        {
11            _service = service ?? throw new ArgumentNullException(nameof(service));
12        }
13
14        [HttpGet("{id:Guid}")]
15        public IActionResult Get(Guid id)
16        {
17            var customer = _service.Get(id);
18
19            if (customer != null)
20            {
21                return Ok(customer);
22            }
23
24            return StatusCode(422);
25        }
26    }
27 }
```

We can write an integration test like this, using the *GetAsync* method from *Client* of *TestFixture* to call the specific endpoint and ensuring that the return is a valid customer and the Http Status Code is 200:


```

1  using IntegrPoc.Api;
2  using IntegrPoc.Api.Models;
3  using IntegrPoc.IntegrationTests.Configuration;
4  using Newtonsoft.Json;
5  using System;
6  using System.Threading.Tasks;
7  using Xunit;
8
9  namespace IntegrPoc.IntegrationTests.Controllers
10 {
11     [Collection(nameof(IntegrationTestsApiFixtureCollection))]
12     public class CustomerControllerTests
13     {
14         private readonly IntegrationTestsFixture<Startup> _testsFixture;
15
16         public CustomerControllerTests(IntegrationTestsFixture<Startup> testsFixture)
17         {
18             _testsFixture = testsFixture;
19         }
20
21         [Fact(DisplayName = "Get_ShouldFindOneCustomerById")]
22         public async Task Get_ShouldFindOneCustomerById()
23         {
24             // Arrange
25             var id = new Guid("3fff9268-b022-4e4d-b1b5-f462c8240b25");
26
27             // Act
28             var response = await _testsFixture.Client.GetAsync($"api/customer/{id}");
29             var responseString = await response.Content.ReadAsStringAsync();
30             var customer = JsonConvert.DeserializeObject<CustomerViewModel>(responseString);
31
32             // Assert
33             response.EnsureSuccessStatusCode();
34             Assert.NotNull(customer);
35             Assert.Equal("First Customer", customer.Name);
36         }
37     }
38 }
39

```

Note the Annotation Collection just above our class referencing the *IntegrationTestsApiFixtureCollection* class and the private field referencing our concrete *Startup* class.

Another example when the customer ID is not valid:



```
1 [Fact(DisplayName = "Get_ShouldReturn422StatusCode_WhenCustomerIdIsValid")]
2 public async Task Get_ShouldReturn422StatusCode_WhenCustomerIdIsValid()
3 {
4     // Arrange
5     var id = Guid.Empty;
6
7     // Act
8     var response = await _testsFixture.Client.GetAsync($"api/customer/{id}");
9
10    // Assert
11    Assert.Equal(422, (int)response.StatusCode);
12 }
```

---

The source code and more examples can be found at my GitHub

<https://github.com/guilhermesalvi/integration-tests-poc>

#### References:

<https://docs.microsoft.com/en-us/aspnet/core/test/integration-tests?view=aspnetcore-5.0>

<https://desenvolvedor.io/curso-online-dominando-os-testes-de-software>