**Unit Testing ASP.NET Web API by Mocking DbContext with NSubstitute**

In the previous post, we created a [simple Web API in 5 minutes](https://www.mydatahack.com/creating-web-apis-using-asp-net-and-mysql-in-5-minutes/). Now, let’s refactor the code so that it is unit testable with [xUnit](https://docs.microsoft.com/en-us/dotnet/core/testing/unit-testing-with-dotnet-test).

The API we build previously was sourcing the data from MySQL database through DbContext. This is an external dependency that we need to replace in the unit test. There are a few good mocking libraries available for .NET. Here, we are going to use [NSubstitue](http://nsubstitute.github.io/) to mock the DbContext.

**Code Refactoring**

1. **Extract the Interface from sakilaDbContext**

The rule of thumb with NSubstitue is to mock interfaces, not classes. In the Actors API, we have the DbContext class called sakilaContext. We first need to extract the interface from the DbContext so that it can be mocked.

This can be done easily with Visual Studio by right clicking the class name in sakilaContext, choose Quick Actions and Refactoring, and then select Extract Interface.

1. **Create ActorsRepository.cs**

Let’s move the logic to get actors from the controller and handle it with ActorsRepository and extract the interface. In this way, our unit test runs against the repository not the controller. The controller simply uses the interface, IActorsRepository class to get all the actors.

First, create a folder called Repositories, add ActorsRepository.cs and extract the interface. In the repository, we inject the dependency on IsakilaContext. If you need more information on dependency injection, read [here](https://docs.microsoft.com/en-gb/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-2.1). Understanding [Inversion of Control](https://docs.microsoft.com/en-us/dotnet/standard/modern-web-apps-azure-architecture/architectural-principles#dependency-inversion) also helps you to make your code more robust and unit testable.

[cc lang="csharp" tab\_size="4" theme="mac-classic" lines="-1"]

public class ActorsRepository : IActorsRepository

{

private IsakilaContext \_dbContext;

public ActorsRepository(IsakilaContext dbContext)

{

\_dbContext = dbContext;

}

public Actor[] GetActors()

{

return \_dbContext.Actor.ToArray();

}

}

[/cc]

1. **Update ActorsController to use ActorsRepository.**

Let’s refactor the controller so that it uses the logic from ActorsRepository to get data. In this way, the controller is only responsible for receiving the request and returning the response and the core API logic sits in the repository class.

[cc lang="csharp" tab\_size="4" theme="mac-classic" lines="-1"]

[Route("api/[controller]")]

public class ActorsController : Controller

{

private IActorsRepository \_actors;

public ActorsController(sakilaContext context)

{

\_actors = new ActorsRepository(context);

}

public IActionResult Index()

{

return Ok(\_actors.GetActors());

}

}

[/cc]

**Creating Unit Test**

After refactoring your code, it is ready for unit testing. Let’s set up the test project and write test code.

1. **Add xUnit Test Project, Configure Project Dependencies and Install NSubstitute**

First of all, you need to create a test project by adding xUnit Test Project under the solution. The naming convention is the project name plus Test (e.g. ApiExampleTest if the project is called ApiExample).

Once the project is created, right click the project to add project dependencies. The unit test project needs the project dependency to the actual project you are testing on.

1. **Create a unit test**

Now you can mock the DbContest interface and mock the Actors data. Mocking DbSet is tricky because it is not a simple list object. You need to set up all IQueryable methods. For this bit, I referenced this excellent blog post called [Mock Entity Framework DbSet with NSubstitute](http://sinairv.github.io/blog/2015/10/04/mock-entity-framework-dbset-with-nsubstitute/) from Sina’s Blog.

[cc lang="csharp" tab\_size="4" theme="mac-classic" lines="-1"]

public class ActorsRepositoryTest

{

[Fact]

public void GetActors\_Should\_Return\_Actors()

{

var mockActors = new List<Actor>()

{

new Actor {ActorId = 1, FirstName = "Hello", LastName = "World", LastUpdate = DateTime.Now, FilmActor = new List<FilmActor>()},

new Actor {ActorId = 2, FirstName = "Second", LastName = "Record", LastUpdate = DateTime.Now, FilmActor = new List<FilmActor>()}

}.AsQueryable();

var mockSet = Substitute.For<DbSet<Actor>, IQueryable<Actor>>();

// setup all IQueryable methods using what you have from "data"

((IQueryable<Actor>)mockSet).Provider.Returns(mockActors.Provider);

((IQueryable<Actor>)mockSet).Expression.Returns(mockActors.Expression);

((IQueryable<Actor>)mockSet).ElementType.Returns(mockActors.ElementType);

((IQueryable<Actor>)mockSet).GetEnumerator().Returns(mockActors.GetEnumerator());

var dbContextMock = Substitute.For<IsakilaContext>();

dbContextMock.Actor.Returns(mockSet);

// Act

var actors = new ActorsRepository(dbContextMock);

var data = actors.GetActors();

// Assert

Assert.Equal(data[0].FirstName, "Hello");

}

}

[/cc]

**Refactoring Unit Test**

Now that the unit test runs, let’s refactor it a little better. First of all, we will create a class that contains mock data instead of hard coding it within the test code. In this way, we can use the same mock data for different logics.

Then, let’s take out the logic to mock Entity Framwork DbSet. For this, I am using the code example from Sina’s Blog [here](http://sinairv.github.io/blog/2015/10/04/mock-entity-framework-dbset-with-nsubstitute/).

1. **Create MockActors class**

Let’s create a folder called Repositories within the test project and create a class called MockActors. When the class is instantiated, the list of actors can be accessible through the public property, Actors.

[cc lang="csharp" tab\_size="4" theme="mac-classic" lines="-1"]

namespace ApiIn10MinutestTest.Repositories

{

public class MockActors

{

public List<Actor> Actors;

public MockActors()

{

Actors = new List<Actor>();

Actors.Add(new Actor { ActorId = 1, FirstName = "Hello", LastName = "World", LastUpdate = DateTime.Now, FilmActor = new List<FilmActor>() });

Actors.Add(new Actor { ActorId = 2, FirstName = "Second", LastName = "Record", LastUpdate = DateTime.Now, FilmActor = new List<FilmActor>() });

}

}

}

[/cc]

1. **Create a utility class to mock DbSet**

Let’s take out the logic to create mock DbSet. This is a non-async version of NSubstituteUtils from from Sina’s Blog example [here](http://sinairv.github.io/blog/2015/10/04/mock-entity-framework-dbset-with-nsubstitute/).

[cc lang="csharp" tab\_size="4" theme="mac-classic" lines="-1"]

namespace ApiIn10MinutestTest.Utilities

{

public static class NSubstituteUtils

{

public static DbSet<T> CreateMockDbSet<T>(IEnumerable<T> data = null)

where T : class

{

var mockSet = Substitute.For<DbSet<T>, IQueryable<T>>();

if (data != null)

{

var queryable = data.AsQueryable();

((IQueryable<T>)mockSet).Provider.Returns(queryable.Provider);

((IQueryable<T>)mockSet).Expression.Returns(queryable.Expression);

((IQueryable<T>)mockSet).ElementType.Returns(queryable.ElementType);

((IQueryable<T>)mockSet).GetEnumerator().Returns(queryable.GetEnumerator());

}

return mockSet;

}

}

}

[/cc]

1. **Refactor unit test to use the utility class and mock data class**

Putting them all together, the test code looks much neater. I am fond of [FluentAssertions](https://fluentassertions.com/documentation/) for assertions. Check it out. It is easy to use and gives you more options than the vanilla xUnit assertion options.

[cc lang="csharp" tab\_size="4" theme=" GeSHi" lines="-1"]

namespace ApiIn10MinutestTest

{

public class GetActorsTest

{

private MockActors \_Actors;

public GetActorsTest()

{

\_Actors = new MockActors();

}

[Fact]

public void GetActors\_Should\_Return\_All\_Actors()

{

// Arrange

var mockDbSet = NSubstituteUtils.CreateMockDbSet(\_Actors.Actors);

var mockSakilaContext = Substitute.For<IsakilaContext>();

mockSakilaContext.Actor.Returns(mockDbSet);

var actors = new ActorsRepository(mockSakilaContext);

// Act

var data = actors.GetActors();

// Assert

data[0].FirstName.Should().Be("Hello");

data.Should().HaveCount(2);

}

}

}

[/cc]

Good times!