**Unit Testing Entity Framework Database Update logic with NSubstitute**

In the previous post ([Unit Testing ASP.NET Web API by Mocking DbContext with NSubstitute),](https://www.mydatahack.com/unit-testing-asp-net-web-api-by-mocking-dbcontext-with-nsubstitute/) we explored how to create a unit test for getting data from database with Entity Framework by mocking DbContext and DbSet. We are going to build upon the API we created previously and add PUT logic where it updates the Actor record in MySQL by actor\_id.

If you need to brush up on ASP.NET Web API fundamental, check out this post, [simple Web API in 5 minutes](https://www.mydatahack.com/creating-web-apis-using-asp-net-and-mysql-in-5-minutes/). For the unit testing framework, we are using [xUnit](https://docs.microsoft.com/en-us/dotnet/core/testing/unit-testing-with-dotnet-test) and [NSubstitute](http://nsubstitute.github.io/).

Check the complete API and unit testing solutions in our repo [here](https://github.com/mydatahack/web-api-example-c-sharp-mysql-actor).

**Creating PUT logic**

To update database, we are going to use [Entity State](https://docs.microsoft.com/en-us/ef/ef6/saving/change-tracking/entity-state). To make this testable, we will create a MarkAsModified method in sakilaContext.cs.

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

public void MarkAsModified(Actor item)

{

Entry(item).State = EntityState.Modified;

}

[/cc]

Add it to the interface, IsakilaContext.cs. We also need to add SaveChange() method in the interface. This is to stub SaveChange() method.

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

int SaveChanges();

void MarkAsModified(Actor actor);

[/cc]

Then, add UpdateActor method to ActorsRepository. Check if the id exists in the database. If it does, update the record and return 1.

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

public int UpdateActorByIdEntityState(int id, Actor actor)

{

int updateSuccess = 0;

if (id != actor.ActorId)

{

return updateSuccess;

}

\_context.MarkAsModified(actor);

updateSuccess = \_context.SaveChanges();

return updateSuccess;

}

[/cc]

Controller use this method for PUT request.

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

[HttpPut("{id}")]

public IActionResult Put(int id, [FromBody]Actor actor)

{

if(!ModelState.IsValid)

{

return BadRequest();

}

int success = actors.UpdateActorByIdEntityState(id, actor);

if (success == 1)

{

return Ok();

}

return BadRequest();

}

[/cc]

**Creating Unit Test on Repository**

Mocking DbSet and DbContext has been described in the [previous post](https://www.mydatahack.com/unit-testing-asp-net-web-api-by-mocking-dbcontext-with-nsubstitute/). Now, we can [substitute the void method](http://nsubstitute.github.io/help/callbacks/), MarkAsModified and SaveChange() method in the IsakilaContext.

Rest is simple. There are four scenarios in the test. By substituting the void method to increment the counter every time it gets called, we can check how many times MarkAsModified method was called.

Here is the [Unit testing best practice guide](https://docs.microsoft.com/en-us/dotnet/core/testing/unit-testing-best-practices) for further reading.

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

[/cc]