Now, let's go ahead and test the interaction of this controller, with this new storage object. So, back in our unit testing project, inside the Mocking folder, I'm going to add a new class EmployeeControllerTests.

Let's add a TestFixture attribute. And create a test for DeleteEmployee method.

So when this is called, it should delete the employee from database.

Now we need to create a Mock object, so var storage as a new Mock of IEmployeeStorage, with this we can create a controller object, so controller, is a new EmployeeController, and as an argument the pass storage.object. Now, we need to act, so controller.DeleteEmployee. We use the simple value here like 1. In this case, we don't care about the result, we just want to test the interaction of this controller with a storage object. So, we call storage .Verify.

Give it a lambda expression you want to make sure that the DeleteEmployee method of storage is called, DeleteEmployee, and argument to this method should be 1. Let's run this test.

So here's our test, it passed successfully. Beautiful. So this is one test, for our DeleteEmployee method. But this method actually requires another test that I'm not going to write in this lecture, I'm just want to highlight it, you can do it as an exercise. So, back to our controller, we should also make sure that this method returns the right value, or the right object for the client. So in this case we're calling a private method called RedirectToAction. And inside this method we are returning a new RedirectResult object. So we should write another test and ensure that what we get from this method is an object of type redirect result.

Okay, also, just to refresh your memory, note that here we're calling a private method, and as I told you before you should not test private or protected methods, because they're purely responsible for implementation. So, we don't want to write a test and Assert that this method is called. That's implementation detail that can change in the future. Because in the future, instead of calling this private method, you may decide to directly return a new RedirectResult object.

Okay? So, let's revert this back, and one last thing before we finish this lecture, let's take a look at our EmployeeStorage object. So we move some of our code here, inside this method, but we don't have any unit tests for this method. Do we need unit tests here? No because here we are working directly with an external resource. So the proper way to test this method, is using an integration test, and that's beyond the scope of this course.

That's a separate course that I'm going to create in the future, with an integration test, we can call this method and ensure that the given employee from the database is deleted. We don't care how this method achieves that result. It may use any framework it may use plain a-dio (?) .net, it may use a different (?) these are all implementation details. So, we are done with this exercise, I hope you enjoyed, and thank you for watching.