A Testable Approach To Entity Framework

The problem (Demo)

* Entity Framework code written inside of our business logic has to have a real database implementation to test.
* Tests that use a real database are slow
* DbContext can’t be mocked

The solution

* Inversion of Control
* The Repository Pattern
  + Pros
    - Gives us a seam for mocking and testing
    - Separates the data source from the
  + Cons
    - People have done bad things with the repository pattern
* The Query Object Pattern
  + Allows us to encapsulate our query logic in a class
  + A new query is a new class. We don’t have to modify an existing class
  + Allows us to use a single repository for all data access
* Highway Data
  + Does the heavy lifting
  + Gives us a great repository implementation
  + Implements interfaces for all its objects so it is easily mockable
  + Gives us some tools for easier testing

Highway

* IDataContext
  + Context
  + ConnectionString
  + IMappingConfiguration
* Repository
  + Query
    - ContextQuery
    - FindAll<T>
  + Scalar
    - IIdentifiable
    - GetById<Tid, TEntity>
  + Command
    - No return
  + AdvancedCommand
    - Access to raw sql
    - Takes a dependency on EF
  + Projections
    - Selector
    - Projector
* ~~Domains~~ 
  + ~~Domain Driven Design Bounded Context~~
  + ~~IDomain~~
  + ~~DomainContext~~
  + ~~DomainRepository~~
* Inject IRepository With Castle
  + LifecycleTransient

Testing With the IRepository and IDataContext

* Mock IRepository
  + Moq
  + Return List of Entities
* Repository with Mock IDataContext
  + Moq IDataContext
    - return Queryable<T>
    - spy on methods such as commit
  + InMemoryDataContext
* Domain Object that depends on IRepository
  + TalkModel

Unit Testing Queries

* Moq IDataContext
  + AsQueryable

Performance Testing Queries

* RunPerformanceTest
  + Only works on queries
  + By default does not include the time to spin up ef

Output SQL

* OutputSQLStatements

Resources

Highwya.Data Website

<http://hwyfwk.com/projects/data/>

Drivers Ed By Devlin Liles

<http://www.youtube.com/watch?v=S1irYAW9azA&app=desktop>

demoProblem

public class TalkModel

{

readonly IRepository \_repository;

public TalkModel(IRepository repository)

{

\_repository = repository;

}

public TalkViewModel GetTalks(int speakerId)

{

var talks = \_repository.Find(new TalksBySpeakerId(speakerId)).ToList();

return new TalkViewModel

{

AcceptedTalks = talks.Where(x=> x.Accepted),

SubmittedTalks = talks.Where(x=>!x.Accepted)

};

}

}

public class TalksBySpeakerId : Query<Talk>

{

public TalksBySpeakerId(int speakerId)

{

ContextQuery = c =>

c.AsQueryable<Talk>()

.Where(x => x.Speaker.Id == speakerId);

}

}

[TestMethod]

public void returns\_accepted\_and\_non\_accepted\_talks()

{

//Arrange

var repo = new Mock<IRepository>();

repo

.Setup(x => x.Find(It.IsAny<TalksBySpeakerId>()))

.Returns(new List<Talk>

{

new Talk{Accepted = true},

new Talk{Accepted = false}

});

var model = new TalkModel(repo.Object);

//Act

var talks = model.GetTalks(1);

//Assert

Assert.AreEqual(1, talks.AcceptedTalks.Count());

Assert.AreEqual(1, talks.SubmittedTalks.Count());

}

demoHighway (1)

var connectionString = ConfigurationManager.AppSettings["connectionString"];

var dataContext =

new DataContext(connectionString, new ConferenceDemoMappingConfiguration());

\_repository = new Repository(dataContext);

public class TalksController : ApiController

{

readonly IRepository \_repository;

public TalksController(IRepository repository)

{

\_repository = repository;

}

[HttpGet]

[Route("api/talks")]

public IHttpActionResult GetAll()

{

var talks = \_repository.Find(new AllTalks()).ToList();

return Ok(talks);

}

[HttpGet]

[Route("api/v2/talks")]

public IHttpActionResult GetAllDefault()

{

var talks = \_repository.Find(new FindAll<Talk>()).ToList();

return Ok(talks);

}

[HttpGet]

[Route("api/talks/{id}")]

public IHttpActionResult GetById(int id)

{

var talks = \_repository.Find(new TalkById(id));

return Ok(talks);

}

[HttpGet]

[Route("api/v2/talks/{id}")]

public IHttpActionResult GetByIdDefault(int id)

{

var talks = \_repository.Find(new GetById<int, Talk>(id));

return Ok(talks);

}

demoHighway (2)

[HttpDelete]

[Route("api/talks/{id}")]

public IHttpActionResult DeleteTalk(int id)

{

\_repository.Execute(new DeleteTalk(id));

return Ok();

}

[HttpDelete]

[Route("api/v2/talks/{id}")]

public IHttpActionResult DeleteTalkAdvanced(int id)

{

\_repository.Execute(new DeleteTalkAdvanced(id));

return Ok();

}

[HttpGet]

[Route("api/talks/titles")]

public IHttpActionResult GetTalkTitles()

{

var titles = \_repository.Find(new TalkTitles());

return Ok(titles);

}

}

demoHighway (3)

public class AllTalks : Query<Talk>

{

public AllTalks()

{

ContextQuery = c =>

c.AsQueryable<Talk>();

}

}

public class TalksBySpeakerId : Query<Talk>

{

public TalksBySpeakerId(int speakerId)

{

ContextQuery = c =>

c.AsQueryable<Talk>()

.Where(x => x.Speaker.Id == speakerId);

}

}

public class TalkById : Scalar<Talk>

{

public TalkById(int id)

{

ContextQuery = c =>

c.AsQueryable<Talk>()

.FirstOrDefault(x => x.Id == id);

}

}

public class TalkTitles : Query<Talk, string>

{

public TalkTitles()

{

Selector = c => c.AsQueryable<Talk>();

Projector = talks => talks.Select(x => x.Title);

}

}

demoHighway (4)

public class DeleteTalk : Command

{

public DeleteTalk(int id)

{

ContextQuery = c =>

{

var talk = c.AsQueryable<Talk>().FirstOrDefault(x => x.Id == id);

c.Remove(talk);

c.Commit();

};

}

}

public class DeleteTalkAdvanced : AdvancedCommand

{

public DeleteTalkAdvanced(int id)

{

ContextQuery = c =>

c.ExecuteSqlCommand(

"delete from talks where id = @id",

new SqlParameter("@id", id));

}

}

demoHighway (5)

public void Install(IWindsorContainer container, IConfigurationStore store)

{

var connectionString =

ConfigurationManager.AppSettings["connectionString"];

container.Register(

Component

.For<IMappingConfiguration>()

.ImplementedBy<ConferenceDemoMappingConfiguration>()

.LifestyleSingleton(),

Component

.For<IDataContext>()

.ImplementedBy<DataContext>()

.DependsOn(new {ConnectionString = connectionString})

.LifestyleTransient(),

Component

.For<IRepository>()

.ImplementedBy<Repository>()

.LifestyleTransient()

);

}

demoTesting (1)

[TestMethod]

public void can\_moq\_the\_repository()

{

//Arrange

var repository = new Mock<IRepository>();

repository

.Setup(x => x.Find(It.IsAny<TalksBySpeakerId>()))

.Returns(new List<Talk>

{

new Talk()

});

//Act

var talks = repository.Object.Find(new TalksBySpeakerId(1));

//Assert

Assert.AreEqual(talks.Count(), 1);

}

[TestMethod]

public void can\_mock\_datacontext()

{

//Arrange

var dataContext = new Mock<IDataContext>();

dataContext.Setup(x => x.AsQueryable<Talk>())

.Returns(new List<Talk>

{

new Talk(),

new Talk()

}.AsQueryable());

var repository = new Repository(dataContext.Object);

//Act

var talks = repository.Find(new AllTalks());

//Assert

Assert.AreEqual(talks.Count(), 2);

}

demoTesting (2)

[TestMethod]

public void can\_spy\_on\_dataContext()

{

//Arrange

var dataContext = new Mock<IDataContext>();

dataContext.Setup(x => x.AsQueryable<Talk>())

.Returns(new List<Talk>

{

new Talk(),

new Talk()

}.AsQueryable());

dataContext

.Setup(x => x.Commit());

var repository = new Repository(dataContext.Object);

//Act

repository.Context.Commit();

//Assert

dataContext.Verify(x=>x.Commit(), Times.Once);

}

[TestMethod]

public void can\_use\_in\_memory\_datacontext()

{

//Arrange

var dataContext = new InMemoryDataContext();

dataContext.Add(new Talk());

dataContext.Add(new Talk());

var repository = new Repository(dataContext);

//Act

var talks = repository.Find(new AllTalks());

//Assert

Assert.AreEqual(talks.Count(), 2);

}

demoTesting (3)

[TestMethod]

public void talkModel\_gets\_data\_from\_IRepository()

{

//Arrange

var dataContext = new InMemoryDataContext();

var speaker = new Speaker {Id = 7};

dataContext.Add(new Talk {Speaker = speaker, Accepted = false});

dataContext.Add(new Talk {Speaker = speaker, Accepted = false});

dataContext.Add(new Talk {Speaker = speaker, Accepted = true});

var repository = new Repository(dataContext);

var model = new TalksModel(repository);

//Act

var vm = model.GetTalks(7);

//Assert

Assert.AreEqual(vm.AcceptedTalks.Count(), 1);

Assert.AreEqual(vm.SubmittedTalks.Count(), 2);

}

demoTesting (4)

Testing Queries

[TestClass]

public class QueryTests

{

[TestMethod]

public void can\_unit\_test\_queries()

{

//Arrange

var dataContext = new InMemoryDataContext();

var speaker = new Speaker {Id = 7};

dataContext.Add(new Talk {Speaker = speaker, Accepted = true});

dataContext.Add(new Talk {Speaker = speaker, Accepted = false});

var query = new AcceptedTalksBySpeaker(7);

//Act

var acceptedTalks = query.Execute(dataContext);

//Assert

Assert.AreEqual(1, acceptedTalks.Count());

}

}

public class AcceptedTalksBySpeaker: Query<Talk>

{

public AcceptedTalksBySpeaker(int speakerId)

{

ContextQuery = c => c.AsQueryable<Talk>()

.Where(x =>

x.Speaker.Id == speakerId &&

x.Accepted);

}

}

demoTesting (5)

Performance Testing

[TestMethod]

public void can\_performance\_test\_queries()

{

//Arrange

var dataContext = Helper.RealDataContext();

var query = new AllTalks();

//Act

query.RunPerformanceTest(dataContext, false, 50);

//Assert

}

demoTesting (6)

Outputting Sql

[TestMethod]

public void can\_output\_sql\_for\_a\_given\_query()

{

//Arrange

var dataContext = Helper.RealDataContext();

var query = new TalksBySpeakerId(7);

//Act

var sql = query.OutputSQLStatement(dataContext);

Console.WriteLine(sql);

//Assert

Assert.IsFalse(string.IsNullOrEmpty(sql));

}

demoTestingMspec (1)

public class when\_mocking\_the\_repository

{

Establish context = () =>

{

\_repository = new Mock<IRepository>();

\_repository

.Setup(x => x.Find(Moq.It.IsAny<TalksBySpeakerId>()))

.Returns(new List<Talk>

{

new Talk()

});

};

Because of = () =>

{

\_talks = \_repository.Object.Find(new TalksBySpeakerId(1));

};

It returns\_repository\_data = () => \_talks.Count().ShouldEqual(1);

static Mock<IRepository> \_repository;

static IEnumerable<Talk> \_talks;

}

public class when\_mocking\_the\_datacontext

{

Establish context = () =>

{

var dataContext = new Mock<IDataContext>();

dataContext.Setup(x => x.AsQueryable<Talk>())

.Returns(new List<Talk>

{

new Talk(),

new Talk()

}.AsQueryable());

\_repository = new Repository(dataContext.Object);

};

Because of = () =>

{

\_talks = \_repository.Find(new AllTalks());

};

It returns\_datacontext\_data = () => \_talks.Count().ShouldEqual(2);

static Repository \_repository;

static IEnumerable<Talk> \_talks;

}

demoTestingMspec (2)

public class when\_spying\_on\_the\_dataContext

{

Establish context = () =>

{

\_dataContext = new Mock<IDataContext>();

\_dataContext.Setup(x => x.AsQueryable<Talk>())

.Returns(new List<Talk>

{

new Talk(),

new Talk()

}.AsQueryable());

\_dataContext

.Setup(x => x.Commit());

\_repository = new Repository(\_dataContext.Object);

};

Because of = () => \_repository.Context.Commit();

It should\_be\_called\_once = () => \_dataContext.Verify(x=>x.Commit(), Times.Once);

static Mock<IDataContext> \_dataContext;

static Repository \_repository;

}

public class when\_using\_in\_memory\_datacontext

{

Establish context = () =>

{

var dataContext = new InMemoryDataContext();

dataContext.Add(new Talk());

dataContext.Add(new Talk());

\_repository = new Repository(dataContext);

};

Because of = () =>

{

\_talks = \_repository.Find(new AllTalks());

};

It should\_return\_inmemory\_data = () => \_talks.Count().ShouldEqual(2);

static Repository \_repository;

static IEnumerable<Talk> \_talks;

}

demoTestingMspec (3)

public class when\_creating\_vm

{

Establish context = () =>

{

var dataContext = new InMemoryDataContext();

var speaker = new Speaker {Id = 7};

dataContext.Add(new Talk {Speaker = speaker, Accepted = false});

dataContext.Add(new Talk {Speaker = speaker, Accepted = false});

dataContext.Add(new Talk {Speaker = speaker, Accepted = true});

var repository = new Repository(dataContext);

\_model = new TalksModel(repository);

};

Because of = () =>

{

\_vm = \_model.GetTalks(7);

};

It should\_have\_accepted\_talks = () => \_vm.AcceptedTalks.Count().ShouldEqual(1);

It should\_have\_submitted\_talks = () => \_vm.SubmittedTalks.Count().ShouldEqual(2);

static TalksModel \_model;

static TalkViewModel \_vm;

}

demoTestingMspec (4)

Testing Queries

public class when\_querying\_acceptedTalksBySpeaker

{

Establish context = () =>

{

\_dataContext = new InMemoryDataContext();

var speaker = new Speaker {Id = 7};

\_dataContext.Add(new Talk {Speaker = speaker, Accepted = true});

\_dataContext.Add(new Talk {Speaker = speaker, Accepted = false});

\_query = new AcceptedTalksBySpeaker(7);

};

Because of = () =>

{

\_acceptedTalks = \_query.Execute(\_dataContext);

};

It should\_only\_return\_accepted\_talks = () => \_acceptedTalks.Count().ShouldEqual(1);

static AcceptedTalksBySpeaker \_query;

static InMemoryDataContext \_dataContext;

static IEnumerable<Talk> \_acceptedTalks;

}

public class AcceptedTalksBySpeaker : Query<Talk>

{

public AcceptedTalksBySpeaker(int speakerId)

{

ContextQuery = c => c.AsQueryable<Talk>()

.Where(x =>

x.Speaker.Id == speakerId &&

x.Accepted);

}

}

demoTestingMspec (5)

Performance Testing

public class when\_getting\_all\_talks

{

Establish context = () =>

{

\_dataContext = Helper.RealDataContext();

\_query = new AllTalks();

};

It should\_be\_performant = () =>

{

\_query.RunPerformanceTest(\_dataContext, false, 50);

};

static AllTalks \_query;

static IDataContext \_dataContext;

}

demoTestingMspec (6)

Outputting Sql

public class when\_you\_need\_the\_sql\_query

{

Establish context = () =>

{

\_dataContext = Helper.RealDataContext();

\_query = new TalksBySpeakerId(7);

};

Because of = () =>

{

\_sql = \_query.OutputSQLStatement(\_dataContext);

Console.WriteLine(\_sql);

};

It can\_be\_output = () => \_sql.IsNullOrEmpty().ShouldBeFalse();

static IDataContext \_dataContext;

static TalksBySpeakerId \_query;

static string \_sql;

}