Testing with EF Core

Julie Lerman

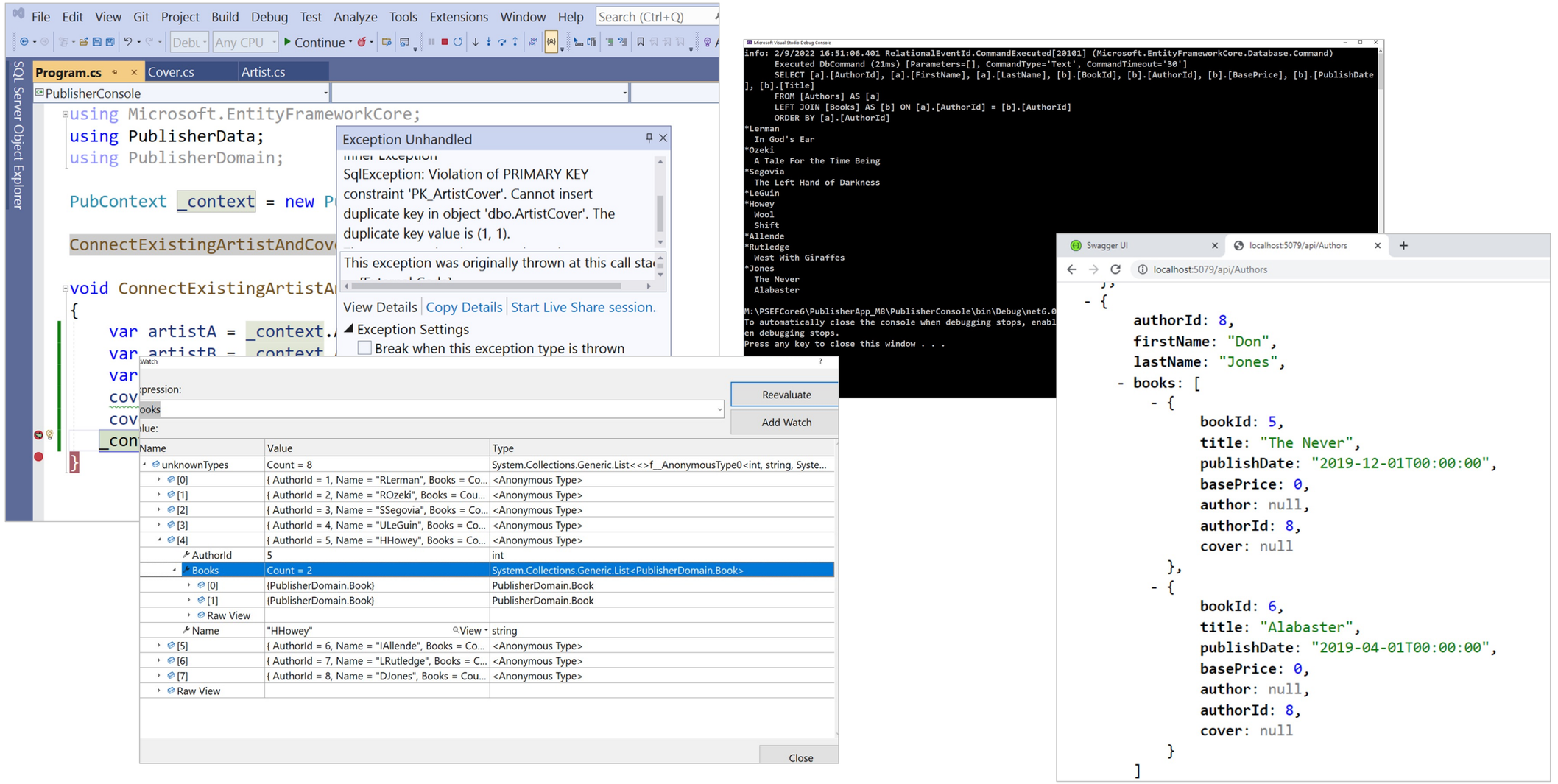
Most Trusted Authority on Entity Framework Core

@JulieLerman www.thedatafarm.com

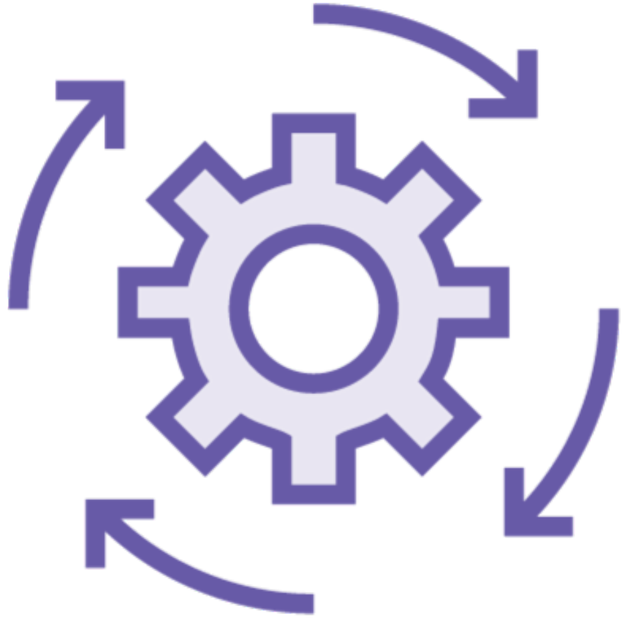
Module Overview

- Super quick testing intro
- What does it mean to test EF Core?
- Writing tests that use a database
- Writing tests to use an in-memory database
- Testing EF Core used in your apps
- Testing EF Core when used in a web app
- Refactor demos for testability
- Learn about structure of testable apps

Manual Testing



Automated Testing



Module Overview

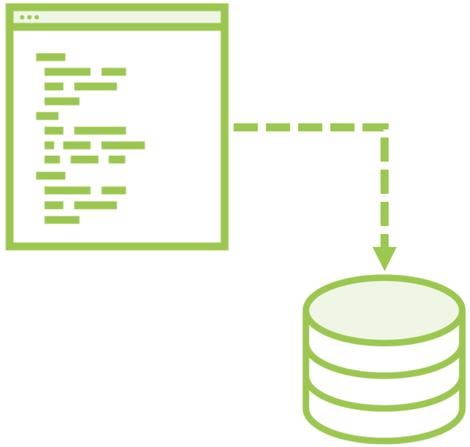
- Super quick testing intro
- What does it mean to test EF Core?
- Writing tests that use a database
- Writing tests to use an in-memory database
- Testing EF Core used in your apps
- Testing EF Core when used in a web app
- Refactor demos for testability
- Learn about structure of testable apps

A Very Quick Testing Overview

Common Types of Automated Tests

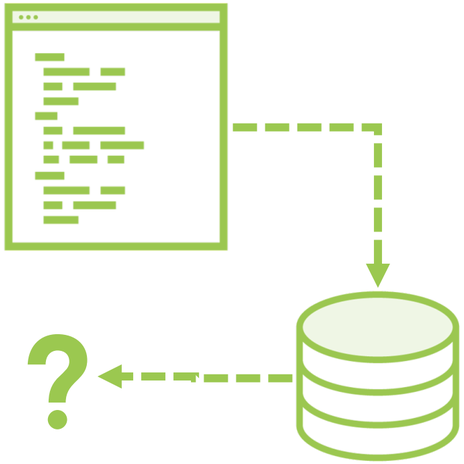
Unit Test

Test small units of your own code



Integration Test

Test that your logic interacts with other services or modules



Functional Test

Verify results of interaction

Other

Many other types of automated testing

Unit Tests: For Small Units of Your Code

```
public class Person
{
  public Person(string first,
                string last)
  {
    FirstName = first;
    LastName = last;
  }
  public string FirstName { get; }
  public string LastName { get; }
}
```

- Your code with no dependencies

```
var p = new Person("Maria“,"Adigon");

Assert.AreEqual(p.FirstName, “Maria”);
Assert.AreEqual(p.FirstName, “Adigon”);
```

- Testing out the constructor
- Test code assertions

Understanding What We Mean by “Testing EF Core”

Testing EF Core Directly or Indirectly

Validate your DbContext against the database

Validate your business logic that uses the DbContext and database

Validate your business logic against the DbContext

Preparing the DbContext

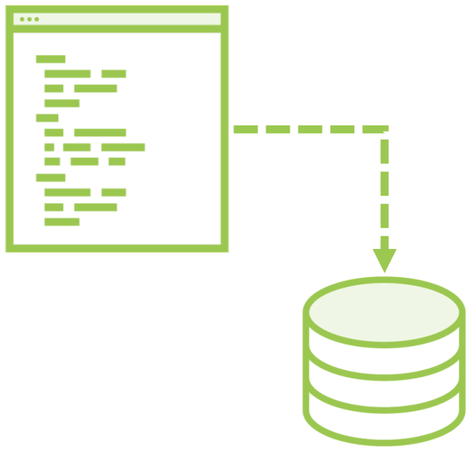
Favor the constructor that takes in the options and don't hard code the connection into the DbContext.

Creating Your First Test and Using It Against the Database

Common Types of Automated Tests

Unit Test

Test small units of your own code



Integration Test

Test that your logic interacts with other services or modules

Your logic
How you configured PubContext

Other services
Entity Framework Core
SQL Server Database

Exploring Test Results and Performance Considerations

If you’re testing with a real database, resetting it is important!

When Your Database Server Is a Resource Hog

Use SQL Server

Need to test specific behaviors of the target database

SQLite or SQL CE

Need to test generic SQL database behaviors

EF Core's InMemory DB

Need to test EF Core behavior or biz logic that uses EF Core

Using the InMemory Provider in Place of a Database Provider

Microsoft.EntityFramework.InMemory

Emulates RDBMS via in-memory lists

Handles generic RDBMS scenarios

Great alternative for test mocks

Requires mods to our existing solution

InMemoryDatabase Connection String

```
public

void Test1()
{

  builder
   .UseInMemoryDatabase(“Test1”);
```

```
public void TestX()
{
   builder
   .UseInMemoryDatabase(“TestXY”);

public void TestY()
{
   builder
   .UseInMemoryDatabase(“TestXY”);
```

“Test1” is a fresh, empty, unique in memory database

TestXY is a fresh database in the first method, and will get reused in the second

Truly Unique Strings

```
public

void Test1()
{

  builder
   .UseInMemoryDatabase(
       Guid.NewGuid().ToString());
```

```
var name=Guid.NewGuid().ToString();
public void TestX()
{
   builder
   .UseInMemoryDatabase(name);

public void TestY()
{
   builder
   .UseInMemoryDatabase(name);
```

Using a GUID for the name

Sharing a GUID across multiple instances

Refactoring and Testing Some Console App Logic

Hey, we have to import authors again!

LOL! Did they buy another publisher *again*?

Let’s create a reusable method!

Time to Create Properly Separated Logic & Testable!

So far, all logic has been in program.cs to demo EF Core features, syntax and behavior

New method should be accessible throughout the solution.

New Test Features

Does not need the database

In-memory provider will suffice

Verify that InsertAuthors returns the correct result

Separation of Concerns Makes Code

Readable

Maintainable

Testable

Testing EF Core in an ASP.NET Core App

I'll walk you through the key changes to the ASP.NET Core app to make it more testable.

Seeding the InMemory Provider

For queries, InMemory db will need some data!

EnsureCreated() will run HasData() methods against InMemory

But...let's see how to re-use an InMemory db instance

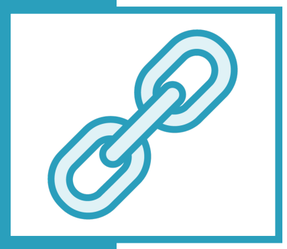
So ...we'll use a local method to seed the InMemory db

Review

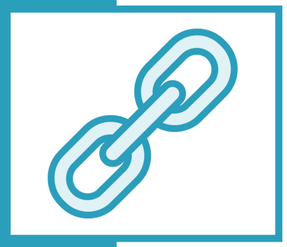
- Testing is an important skill and should be a primary part of your software design
- Verify that EF Core comprehends your DbContext mappings
- Testing against the app's database is not always necessary or efficient
- Substitute with lighter DB or InMemory provider
- InMemory does not emulate a true DB
- Refactor your apps for more testability
- *Testing full app integration in download*

Up Next: Adding Some More Practical Mappings to Your Data Model

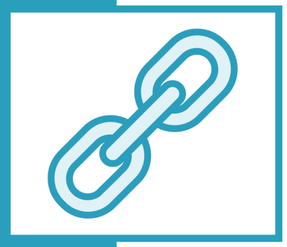
Resources



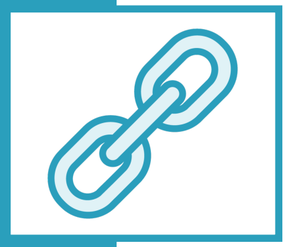
Entity Framework Core on GitHub github.com/dotnet/efcore



EF Core Documentation docs.microsoft.com/ef



Microsoft Docs on ASP.NET Core Testing with Minimal APIs
docs.microsoft.com/en-us/aspnet/core/test/integration-tests



EF Community Standup (Jan 2022): Testing EF Core Applications
youtube.com/watch?v=KO2aFuLqGkc