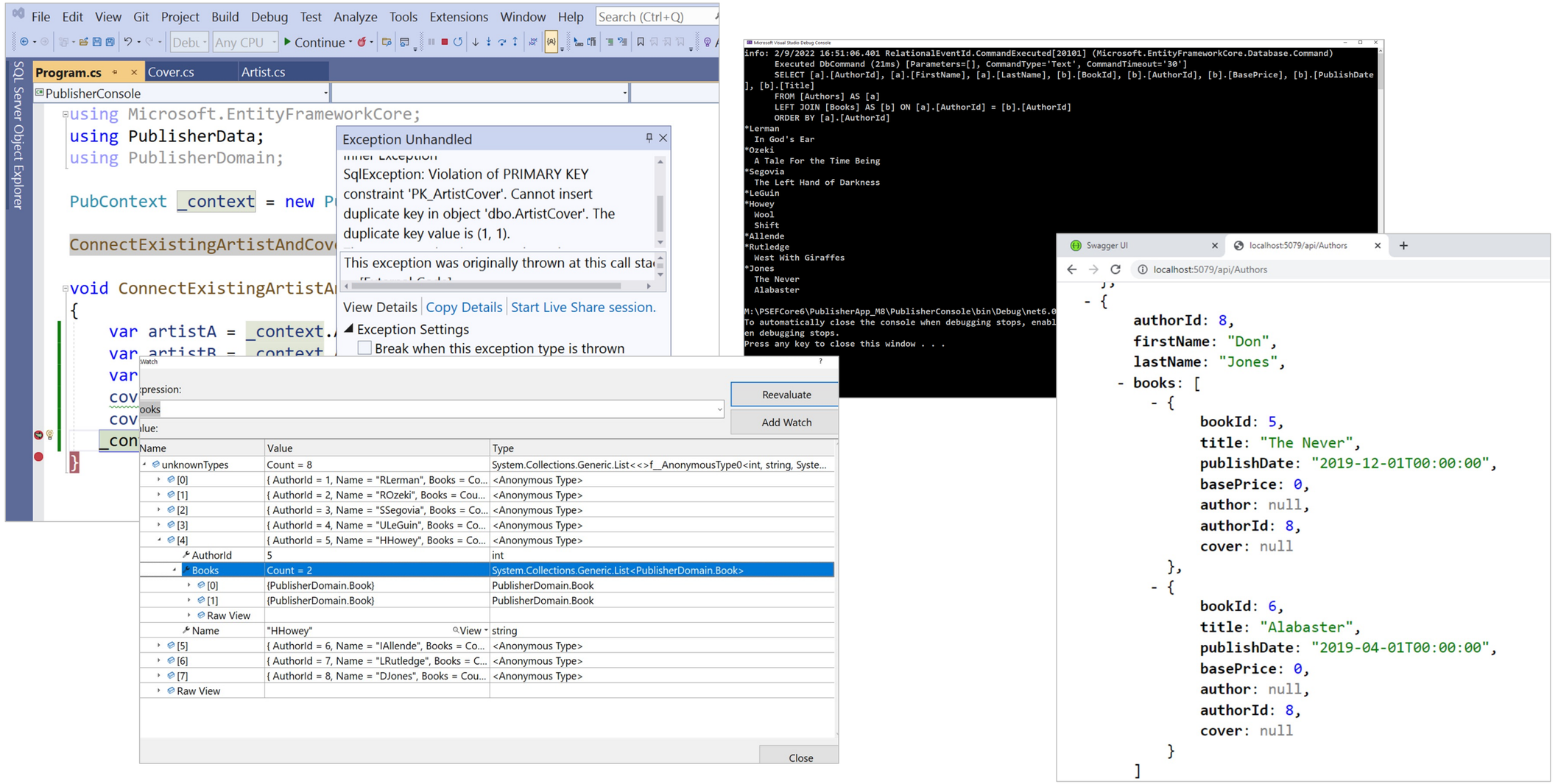
Testing with EF Core

Julie Lerman

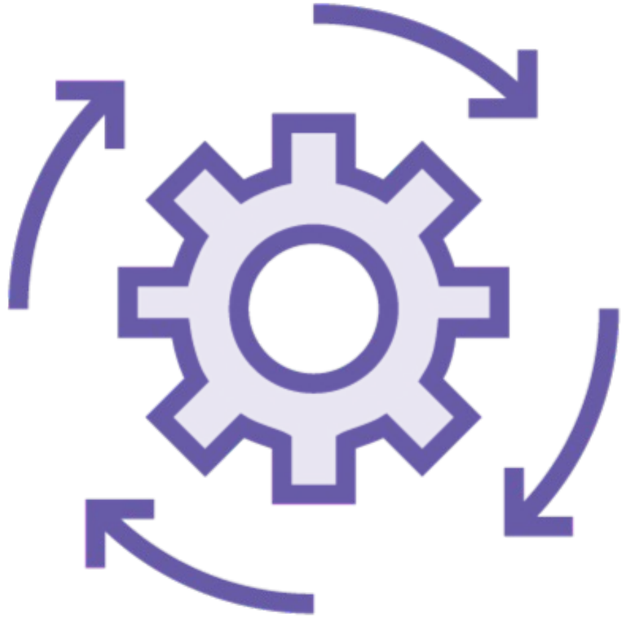
EF Core Expert and Software Coach

@julielerman | thedatafarm.com

Manual Testing



Automated Testing



EF Core InMemory provider is no longer recommended!

Module Overview

Super quick testing intro

What does it mean to test EF Core?

Writing tests that use a database

Testing with SQLite in-memory database

Testing EF Core used in your apps

Testing EF Core when used in a web app

Refactor demos for testability

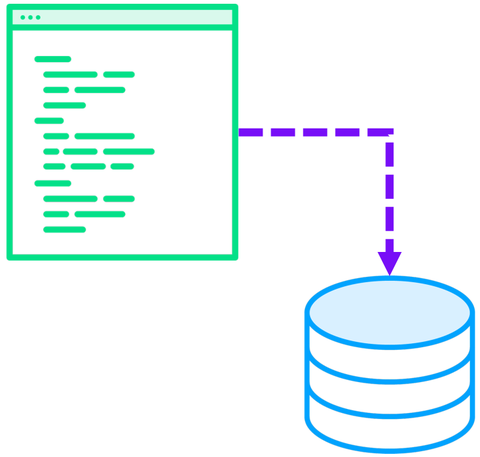
Learn about structure of testable apps

A Very Quick Testing Overview

Common Types of Automated Tests

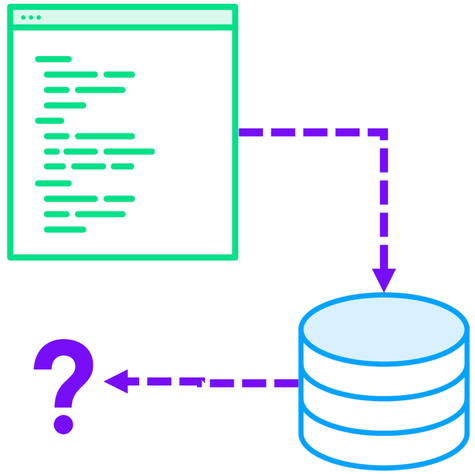
Unit Test

Test small units of your own code



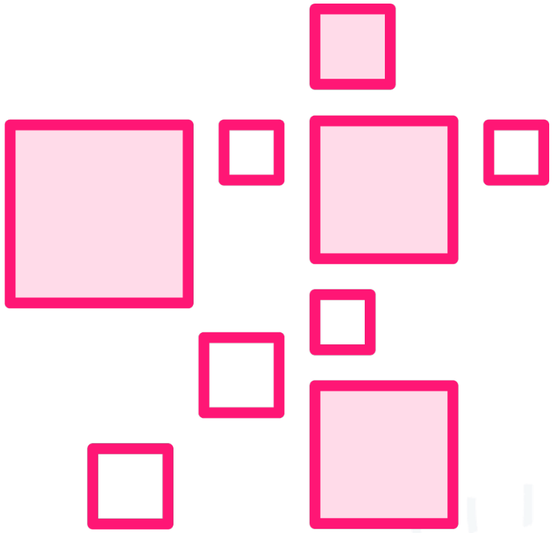
Integration Test

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; init }
  public string LastName { get; init }
}
```

- Your code with no dependencies

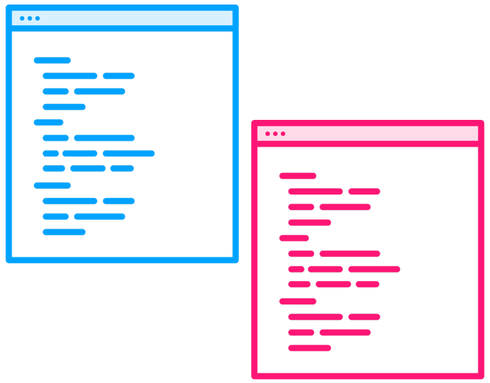
```
var p = new Person("Maria“,"Adigon");

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 against the DbContext

Validate your business logic that uses the DbContext and database

Preparing the DbContext

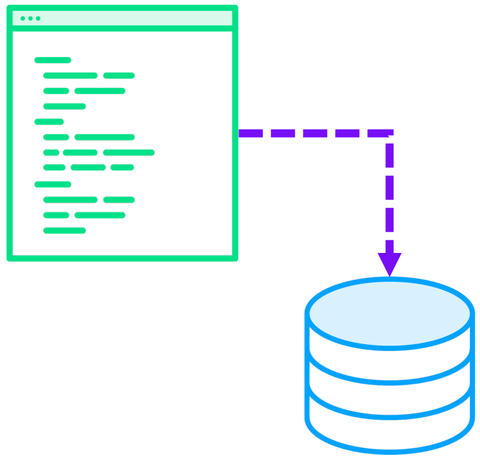
Favor the constructor that takes in the options and don’t hard code the connection into the DbContext.

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 behavior

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

What about EF Core InMemory provider?

What about EF Core InMemory provider?

Too many side-effects and no longer recommended

Testing with SQLite in Memory

SQLite's In-Memory Option

SQLite is already lightweight

Easy to force it to work 100% in memory

Recommended for lightweight EF Core testing

SQLite provider is maintained by EF Core team

Not for tests that depend on database activity

What’s the Goal of This Test?

Do we need to know that the database works...

..or just that the PubContext is working as expected?

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 the ASP.NET Core API

What Are We Testing?

API Endpoints?

Logic triggered by the endpoint?

Refactor ala DataLogic....

```
public async Task<AuthorDTO> GetAuthorDTOById(int authorid)
{
    return await _context.Authors.AsNoTracking()
      .Where(a => a.AuthorId == authorid)
      .Select(a => new AuthorDTO(a.AuthorId, a.FirstName, a.LastName))
      .FirstOrDefaultAsync();
}
```

```
group.MapGet("/{id}", async Task<Results<Ok<AuthorDTO>, NotFound>>(int id) =>
{    return await dl.GetAuthorDTOById(id)
              is AuthorDTO model
                  ? TypedResults.Ok(model)
                  : TypedResults.NotFound();
})
 .WithName("GetAuthorByIdDataLogic")
 .WithOpenApi();
```

What Are We Testing?

API Endpoints?

Logic triggered by the endpoint?

Testing Endpoints That Use EF Core

Remember the Database!

The API is currently using our PubData development database.

Switching the provider and database for the context is a bit trickier in the web API

We still need to create and seed the test database.

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 even SQLite's memory feature when appropriate

But these won't emulate a true DB

Refactor your apps for more testability

Up Next:

Adding Some More Practical Mappings to Your Data Model

Resources

EF Core Docs on Testing without Database
learn.microsoft.com/ef/core/testing/testing-without-the-database

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

How to test ASP.NET Core Minimal API, by Maarten Balliauw
twilio.com/blog/test-aspnetcore-minimal-apis

And Even More Resources …

ASP.NET Core Docs: Integration tests in ASP.NET Core
https://learn.microsoft.com/aspnet/core/test/integration-tests

C# 10 Dependency Injection, Henry Been
app.pluralsight.com/profile/author/henry-been