xUnit Testing

.NET

xUnit.net is a free, open source, community-focused unit testing tool for the .NET Framework. xUnit.net is the latest technology for unit testing C# and other .NET languages. xUnit.net is part of the .NET Foundation.

xUnit Testing – Step By Step

https://docs.microsoft.com/en-us/dotnet/core/testing/unit-testing-with-dotnet-test#create-a-test
https://docs.microsoft.com/en-us/dotnet/core/testing/unit-testing-with-dotnet-test

1. Open a Solution in Visual Studio.
2. Right-Click the Solution.
3. Add >> new project…
4. Type “xunit” in the template box.
5. Select ***xUnit Test Project(.NET Core)***.
6. Name the project whatever you want (VS inserts ‘_’ for spaces).
7. Right-Click ‘Dependencies’ in the test project.
8. Click ‘Add Reference’
9. In the left pane, click ‘Projects’
10. In the center pane, click to place a check next to the Projects containing methods you want to test.
11. Click ‘OK’
12. Add Tests to the Test project.

Arrange, Act, Assert

The three steps to create a test.

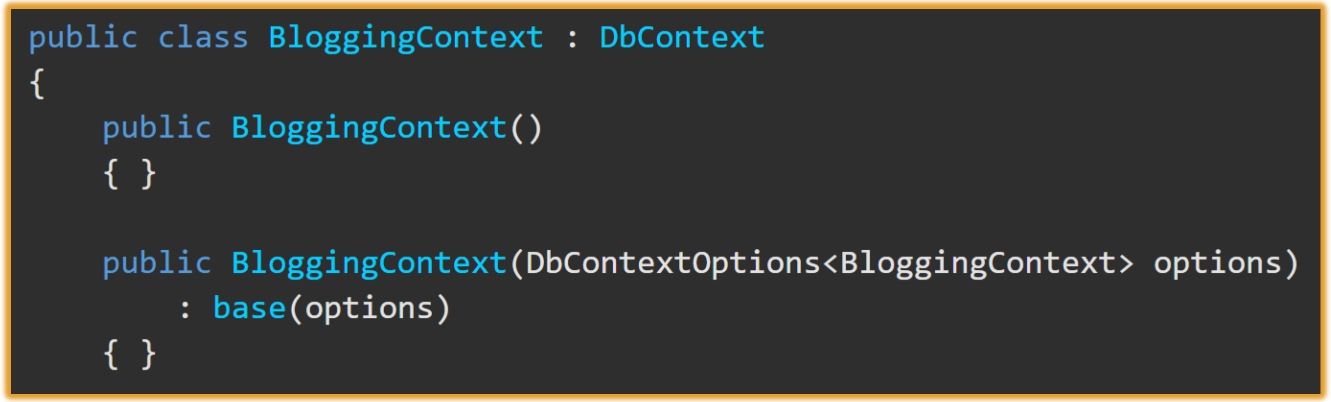
InMemory DB Step-by-Step

https://docs.microsoft.com/en-us/ef/core/miscellaneous/testing/in-memory

EF Core database providers do not have to be relational databases. *InMemory* is designed to be a general purpose database for testing, and is not designed to mimic a relational database.

There are a few steps to setting up a *InMemory* DB.

1. Set up the constructor in your DB Context class to accept a DB configuration parameter called *DbContextOptions*.



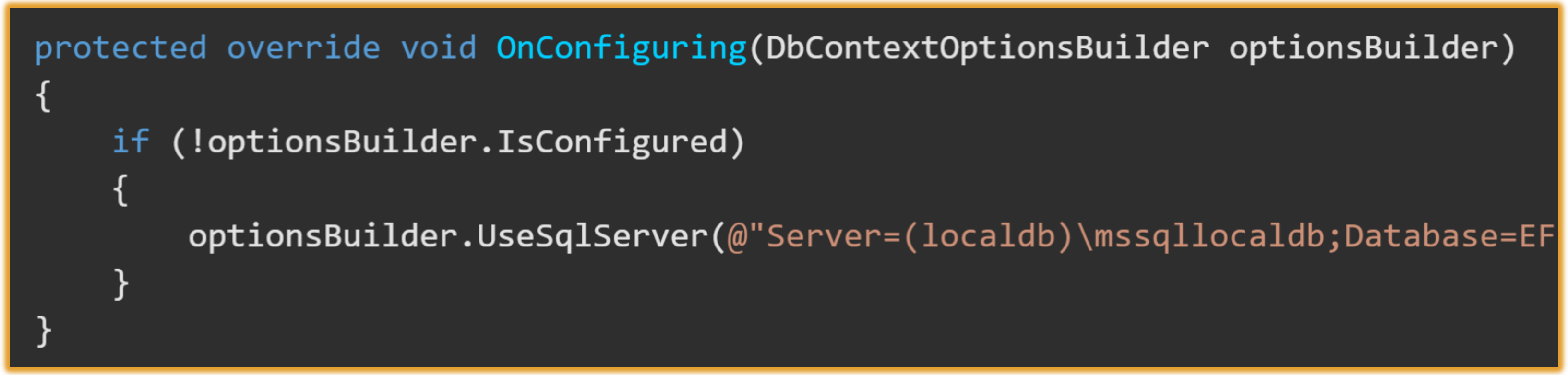
```
public class BloggingContext : DbContext
{
    public BloggingContext()
    { }

    public BloggingContext(DbContextOptions<BloggingContext> options)
        : base(options)
    { }
```

InMemory DB Step-by-Step

https://docs.microsoft.com/en-us/ef/core/miscellaneous/testing/in-memory

2. Alter your *DbContex.OnConfiguring()* to check for an already configured DB and not use your production DB if there's already a DB configured.



```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    if (!optionsBuilder.IsConfigured)
    {
        optionsBuilder.UseSqlServer(@"Server=(localdb)\mssqllocaldb;Database=EF
    }
}
```

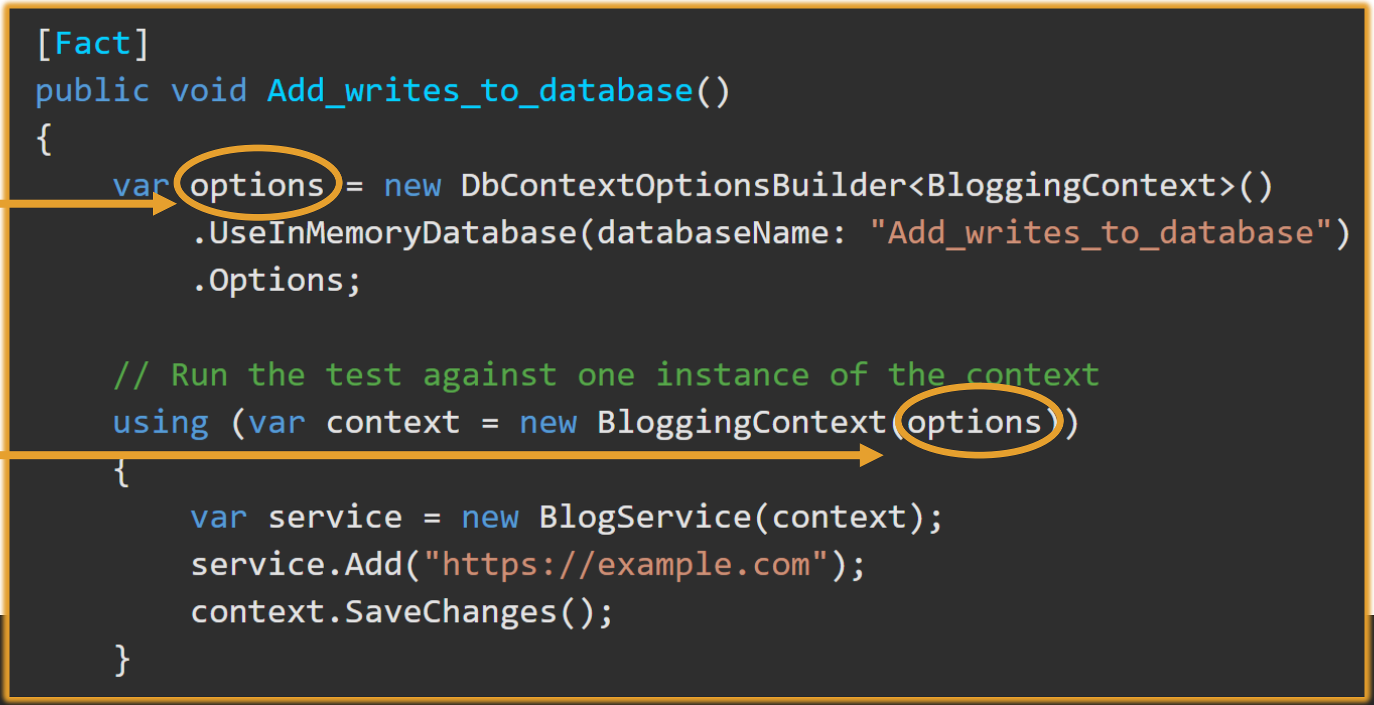
InMemory DB Step-by-Step

https://docs.microsoft.com/en-us/ef/core/miscellaneous/testing/in-memory

3. Right-click your test project to download the NuGet Package ***Microsoft.EntityFrameworkCore.InMemory.***

4. In the test project, You will have to configure a new, clean Db context for every test.

5. You have to send the configurations for this *InMemory* DB to the DbContext Constructor on instantiation.



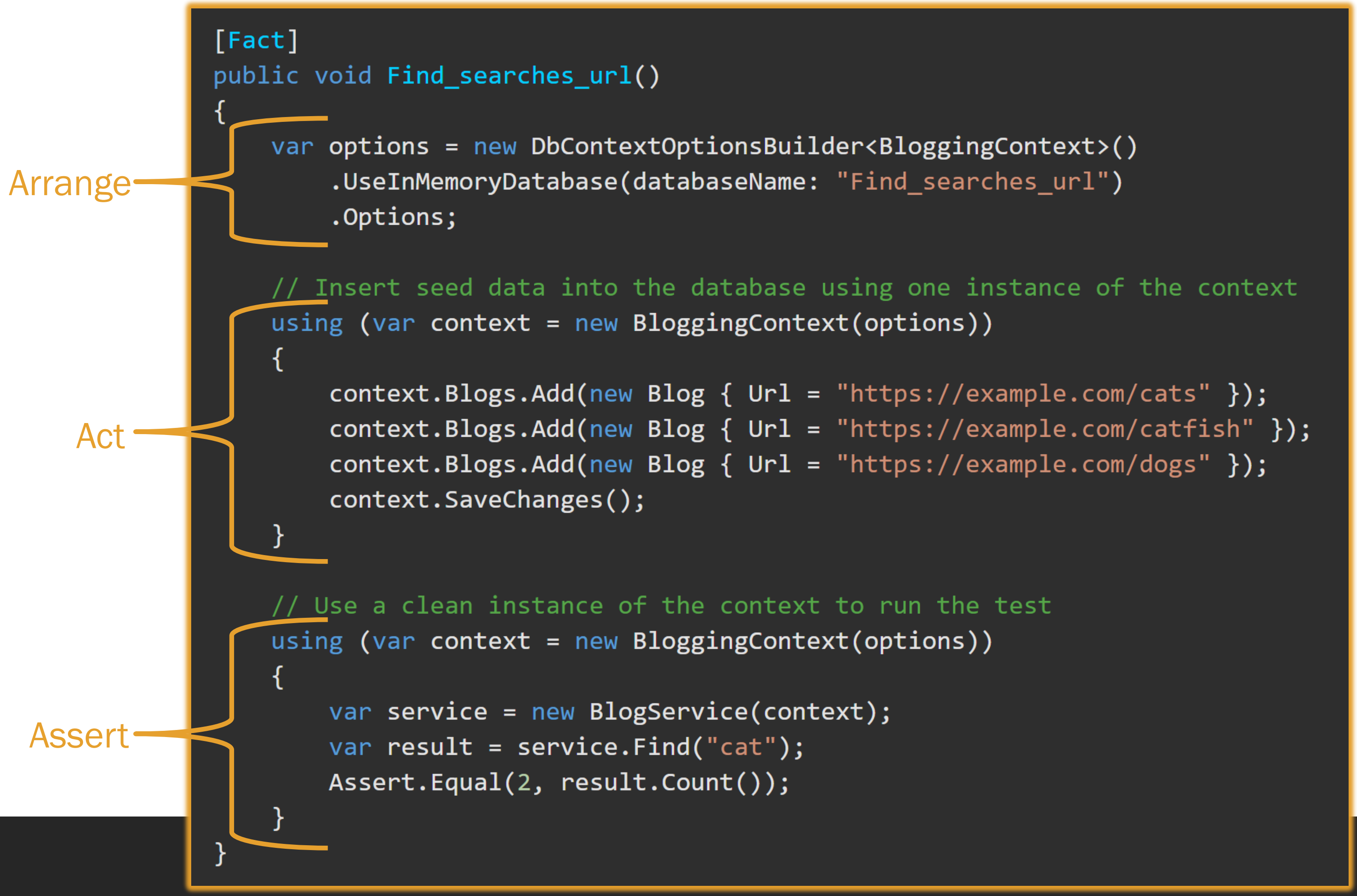
```
[Fact]
public void Add_writes_to_database()
{
    var options = new DbContextOptionsBuilder<BloggingContext>()
        .UseInMemoryDatabase(databaseName: "Add_writes_to_database")
        .Options;

    // Run the test against one instance of the context
    using (var context = new BloggingContext(options))
    {
        var service = new BlogService(context);
        service.Add("https://example.com");
        context.SaveChanges();
    }
```

InMemory DB Step-by-Step

https://docs.microsoft.com/en-us/ef/core/miscellaneous/testing/in-memory

Here is a sample test for comparison.



```
[Fact]
public void Find_searches_url()
{
    var options = new DbContextOptionsBuilder<BloggingContext>()
        .UseInMemoryDatabase(databaseName: "Find_searches_url")
        .Options;

    // Insert seed data into the database using one instance of the context
    using (var context = new BloggingContext(options))
    {
        context.Blogs.Add(new Blog { Url = "https://example.com/cats" });
        context.Blogs.Add(new Blog { Url = "https://example.com/catfish" });
        context.Blogs.Add(new Blog { Url = "https://example.com/dogs" });
        context.SaveChanges();
    }

    // Use a clean instance of the context to run the test
    using (var context = new BloggingContext(options))
    {
        var service = new BlogService(context);
        var result = service.Find("cat");
        Assert.Equal(2, result.Count());
    }
}
```