# Web API testing end2end

## The speaker

Illia Levandovskyi

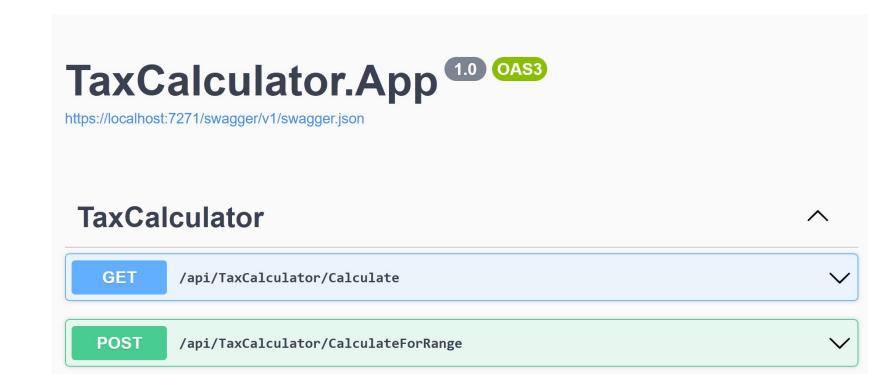
lead software developer @Luxoft

dotnet trainer

global dotnet chapter lead



## The web api



## Testing controller instance

```
public TaxCalculatorControllerTests(ITestOutputHelper output)
   _output = output;
   _scope = new ServiceCollection()
        .AddScoped<ICalculator, Calculator>()
        .AddScoped<IRangeCalculator>(_ ⇒ new Mock<IRangeCalculator>().Object)//is not
        .AddScoped<ITaxationRuleProvider, InMemoryTaxationRuleProvider>()
        .Decorate<ITaxationRuleProvider, TaxationRuleRepository>()
        .AddScoped<ITaxationRuleStorage>(_ ⇒ new Mock<ITaxationRuleStorage>().Object)//
        .AddScoped<TaxCalculatorController>()
        .BuildServiceProvider(validateScopes: true)
        .CreateScope();
   _controller = _scope.ServiceProvider.GetRequiredService<TaxCalculatorController>();
    _calculator = _scope.ServiceProvider.GetRequiredService<ICalculator>();
```

```
0 references | ™ Cyclomatic complexity: 1
public void Dispose() ⇒ _scope.Dispose();
```

#### What is not covered so far

Reading config from appsettings.json

Routing

Serialization/deserialization issues

Request/response caching issues

Middleware and Action/Exception filters

HostedServices

Higher viscosity due to absent Mother Object

#### How to solve all of these issues?

Reading config from appsettings.json

Routing

Serialization/deserialization issues

Request/response caching issues

Middleware and Action/Exception filters

HostedServices

Higher viscosity due to absent Mother Object

## In memory integration tests

Launches Kestrel and your web app in memory, as if you debug the app

- => reads all configuration
- => consumes your app's Composition Root
- => you can send http(s) requests (check routing, serialization, caching, filters & middleware)
- => your tests depend on the web api, not the classes composition (are more robust)

And a cherry on the top - you still can customize it.

## Setup steps

Rely on Microsoft.NET.Sdk.Web

Install NuGet Microsoft.AspNetCore.Mvc.Testing

For xunit use xunit.runner.json with {"shadowCopy": false}

For test class use IClassFixture<WebApplicationFactory<Program>>

requires: public partial class Program {}

## When you have Startup class

Inject into your test classes IClassFixture<WebApplicationFactory<Startup>>

requires:

public static IHostBuilder CreateHostBuilder(string[] args) => Host

.CreateDefaultBuilder(args)

.ConfigureWebHostDefaults(webBuilder => webBuilder.UseStartup<Startup>());

#### How to customize?

Inherit WebApplicationFactory<T>

Override ConfigureWebHost()

Use ConfigureTestServices() to Replace() or Remove() DI-registrations.

Provide environment with UseEnvironment()

# | IHostedService | | BackgroundService

Is run with in memory tests

Could be removed from the DI-container manually

# Microsoft.AspNetCore.Mvc.Testing

test web host

Copies .deps into the test project's bin folder

Sets the content root to the sut

Provides bootstrapping in-memory test server via WebApplicationFactory

## Shadow copy

Shadow copying causes the tests to execute in a different directory than the output directory. If your tests rely on loading files relative to Assembly. Location and you encounter issues, you might have to disable shadow copying.



# Issues: override config from tests

How to override the config from tests before WebApplicationBuilder.Build() is called.

```
private class MyWebApplicationFactory : WebApplicationFactory<Program>
    protected override IHost CreateHost(IHostBuilder builder)
       builder.ConfigureHostConfiguration(config =>
            config.AddInMemoryCollection(new Dictionary<string, string> { "Result", "Override" } });
       });
       return base.CreateHost(builder);
```

## Issues: cannot load appsettings.json

System.IO.InvalidDataException

Failed to load configuration from file '...\appsettings.json'.

Possible reason: malformed appsettings.json

## Summary

Using DI-container in test project is a sort of Object Mother pattern implementation

Creating end2end tests with WebApplicationFactory allows to test app part.

But requires deep knowledge/curiosity of AspNetCore framework.

Although it does not substitute end2end tests on UAT environment (e.g. via Postman)

### References

Microsoft's guide for Integration tests.

Pluralsight trainings by Steve Gordon and Steve Smith.

Source code is available via the QR-code or by the <u>link</u>.

