Isolating Unit Tests with ASP.NET Core Techniques and Mocking

Kevin Dockx
Architect

@KevinDockx https://www.kevindockx.com

Coming Up

Investigating test isolation approaches

- Fakes, dummies, stubs, spies, mocks

Test isolation with Entity Framework Core

Test isolation with HttpClient

Coming Up

Test isolation with Moq

- Creating a mock object
- Configuring a mock object
- Mocking an interface
- Mocking async code

Deciding on the best test isolation approach for your use case

Investigating Test Isolation Approaches

Unit tests should be isolated from other components of the system

- Database, file system, network, …
- Other dependencies like factories, repositories, custom services, …

By isolating a test you can be sure that when it passes or fails, it's the cause of the code under test

Test isolation

Test double

A generic term for any case where you replace a production object for testing purpose

Test Doubles

Fake
A working implementation not suitable for production use

Dummy
A test double that's never accessed or used

Stub
A test double that provides fake data to the system under test

Spy
A test double capable of capturing indirect output and providing indirect input as needed

Mock
A test double that implements the expected behavior

Test Isolation Approaches

Manually creating test doubles

Using built-in framework or library functionality to create test doubles

Using a mocking framework to create test doubles

Different types of test doubles and different approaches are often combined. Focus on the fact that the test is isolated, no matter how.

Test isolation focus

Unit testing with Entity Framework Core

Entity Framework Core contains a set of built-in functionalities to easily enable testing & test isolation

- Avoid calling into a real database
- Use in-memory implementations instead

Unit Testing with Entity Framework Core

In-memory database provider
Simple scenarios
Not the best option

SQLite in-memory mode
Best compatibility with real database

Demo

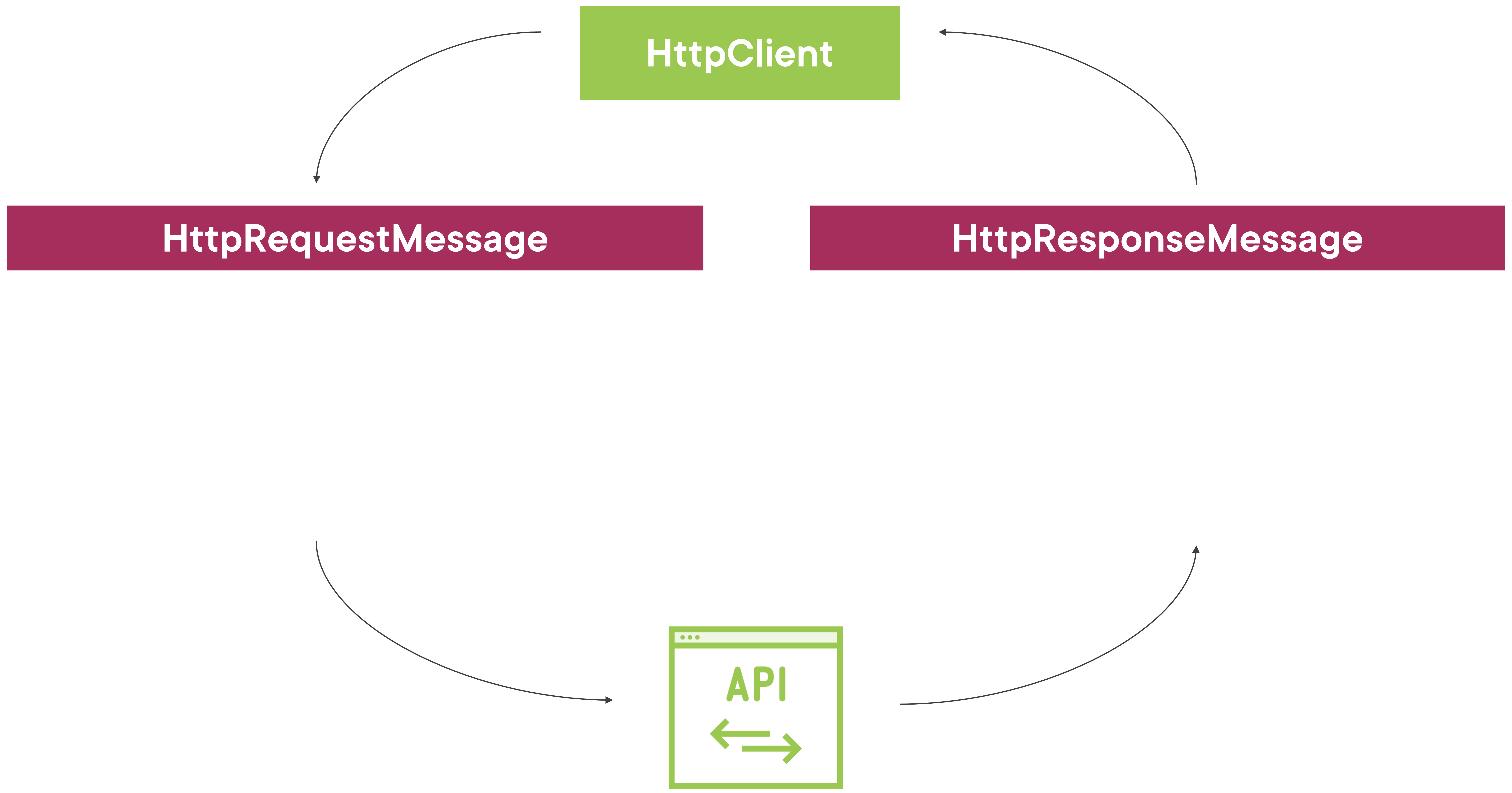
Using SQLite in-memory mode for unit testing

Unit Testing with HttpClient

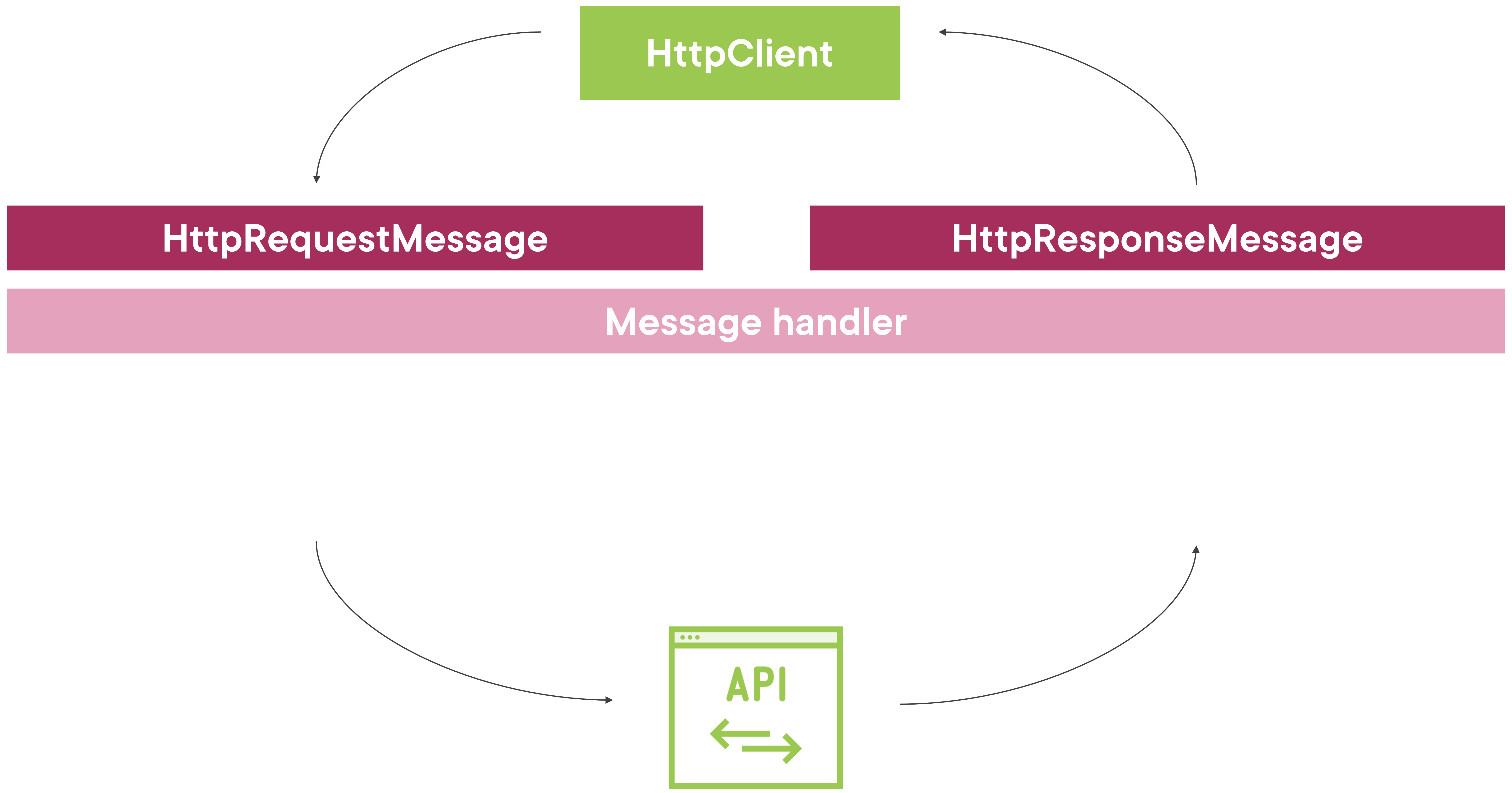
Tests must be isolated from network calls

- A custom message handler can short-circuit the actual call

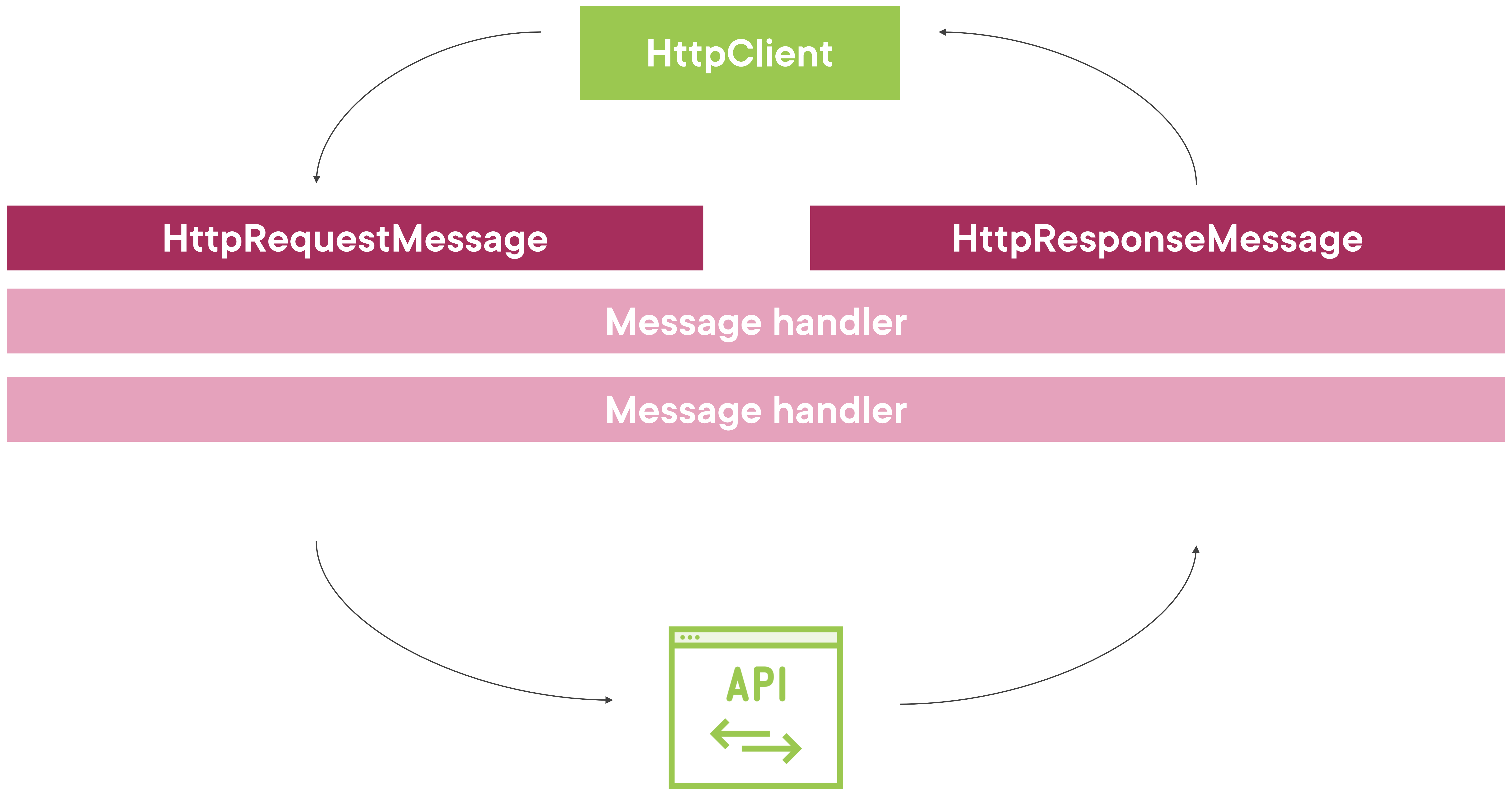
Tackling Integration with HttpClient



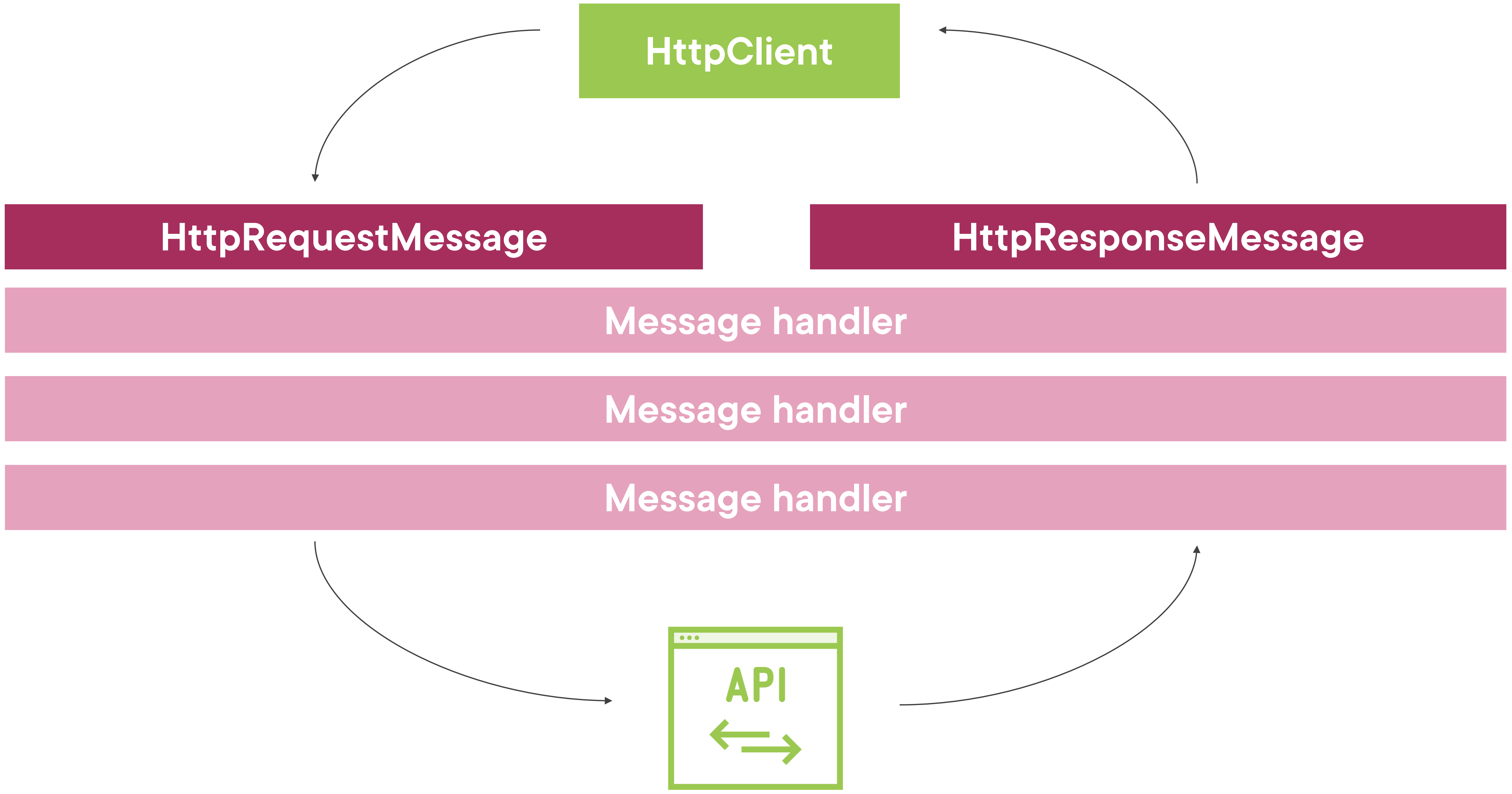
Tackling Integration with HttpClient



Tackling Integration with HttpClient



Tackling Integration with HttpClient



Demo

Unit testing with HttpClient

Demo

Creating and using a mock object

Demo

Configuring mock object return values

Demo

Mocking an interface

Demo

Mocking async code

Which Test Isolation Approach Should You Use?

Consider:

- Test reliability
- Effort required
- Available knowledge
- …

Summary

Test doubles

- Fakes, mocks, spies, dummies, stubs

Framework techniques

- In-memory SQLite databse
- Custom `HttpMessageHandler`

Mocking framework: Moq

Up Next:
Unit Testing ASP.NET Core MVC Controllers