No API? No problem!

API mocking with WireMock.Net

An open source workshop by …

Originally created by Bas Dijkstra - bas@ontestautomation.com - https://www.ontestautomation.com

What are we going to do?

_Stubbing, mocking and service virtualization

_WireMock.Net

_Exercises, examples, …

Preparation

_Install .NET 6

_Install Visual Studio 2022 (or any other IDE)

_Import project into your IDE
  _https://github.com/basdijkstra/restsharp-workshop

Section 0:

An introduction to service virtualization

Problems in test environments

_Systems are constructed out of of many different components

_Not all of these components are always available for testing

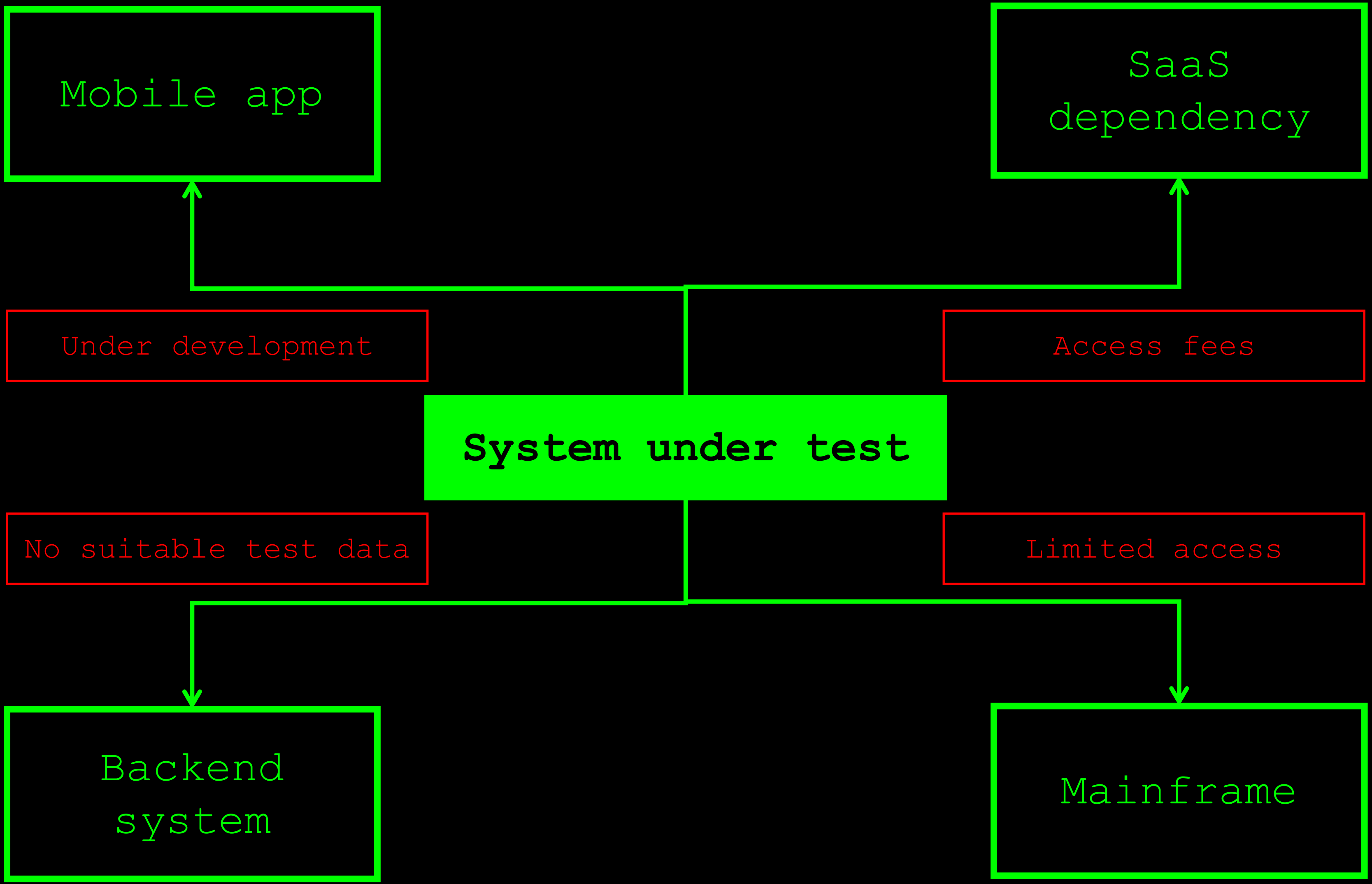
  _Parallel development

  _No control over testdata

  _Fees required for using third party component

  _…

Problems in test environments

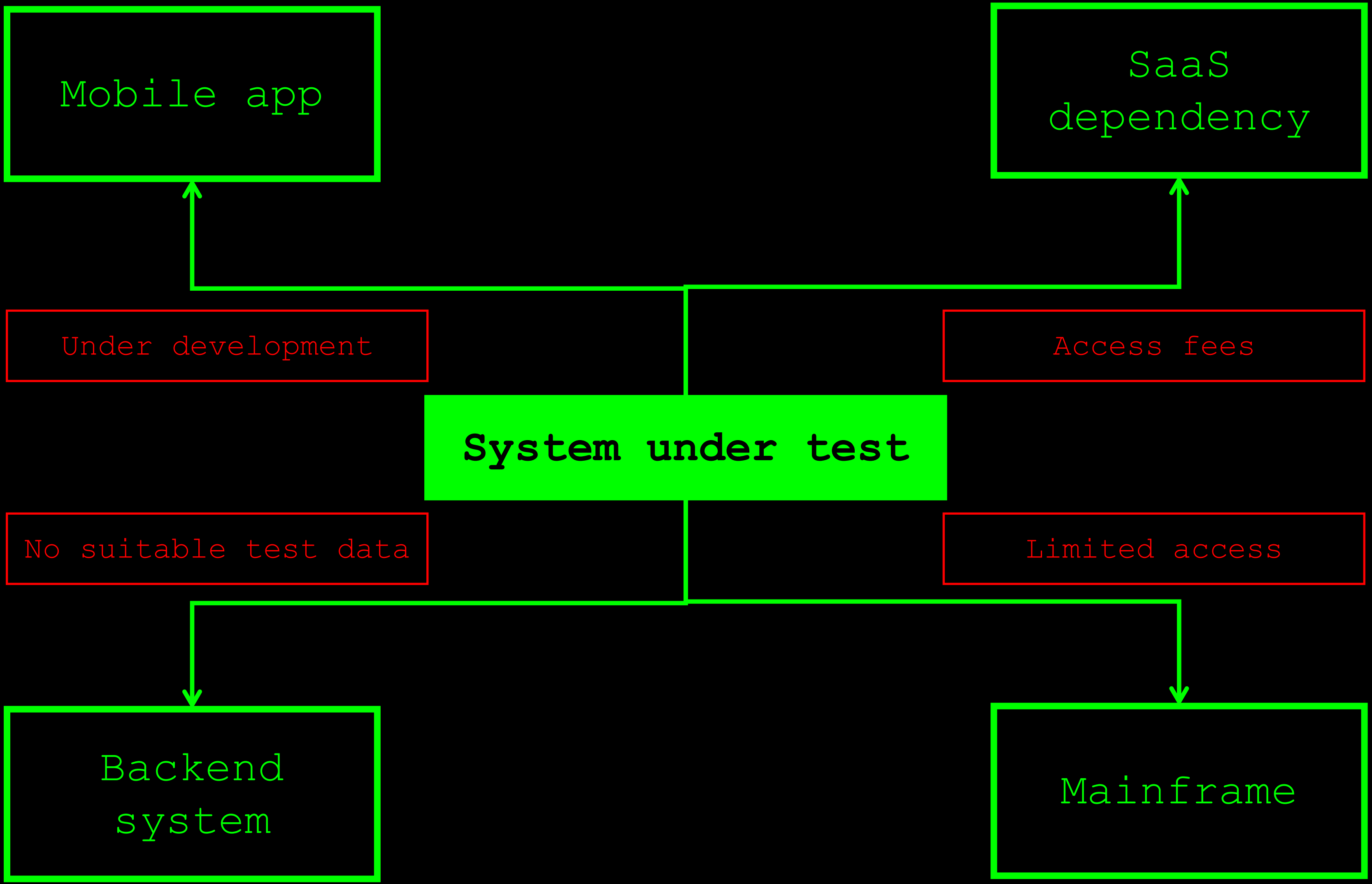


Simulation during test execution

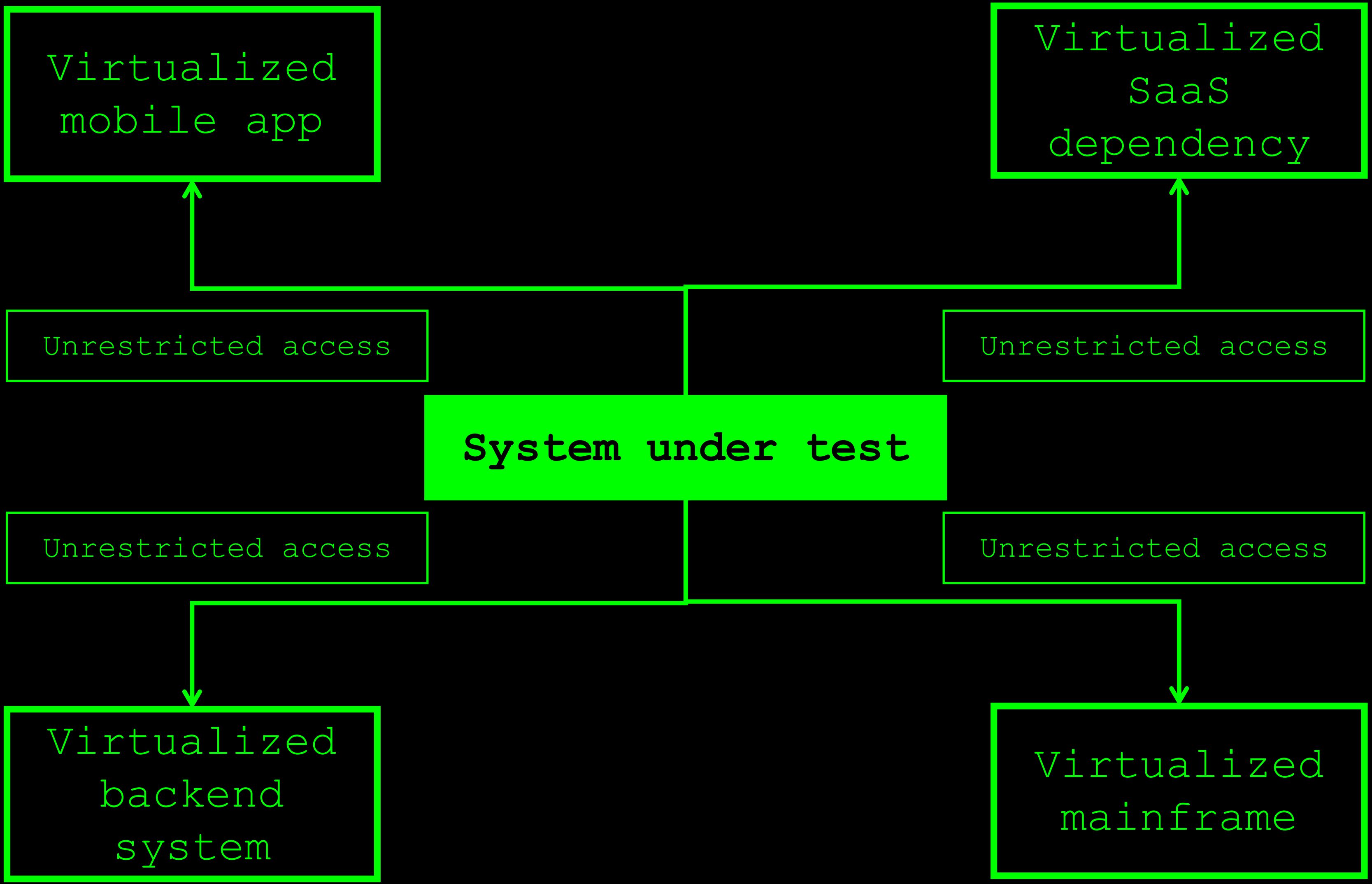
_Simulate dependency **behaviour**

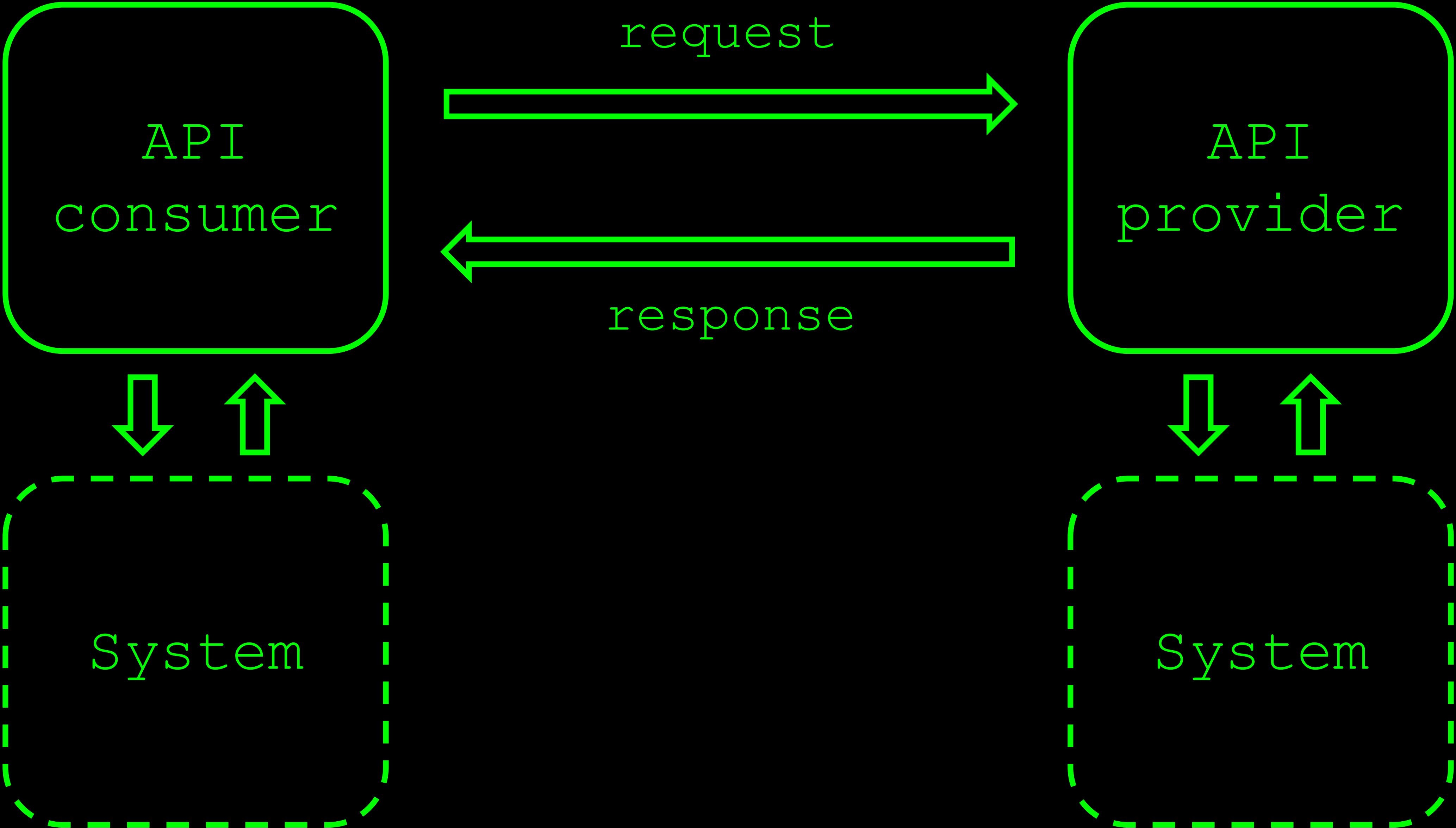
_Regain full control over test environment
  _Available on demand
  _Full control over test data (edge cases!)
  _No third party component usage fees
  _…

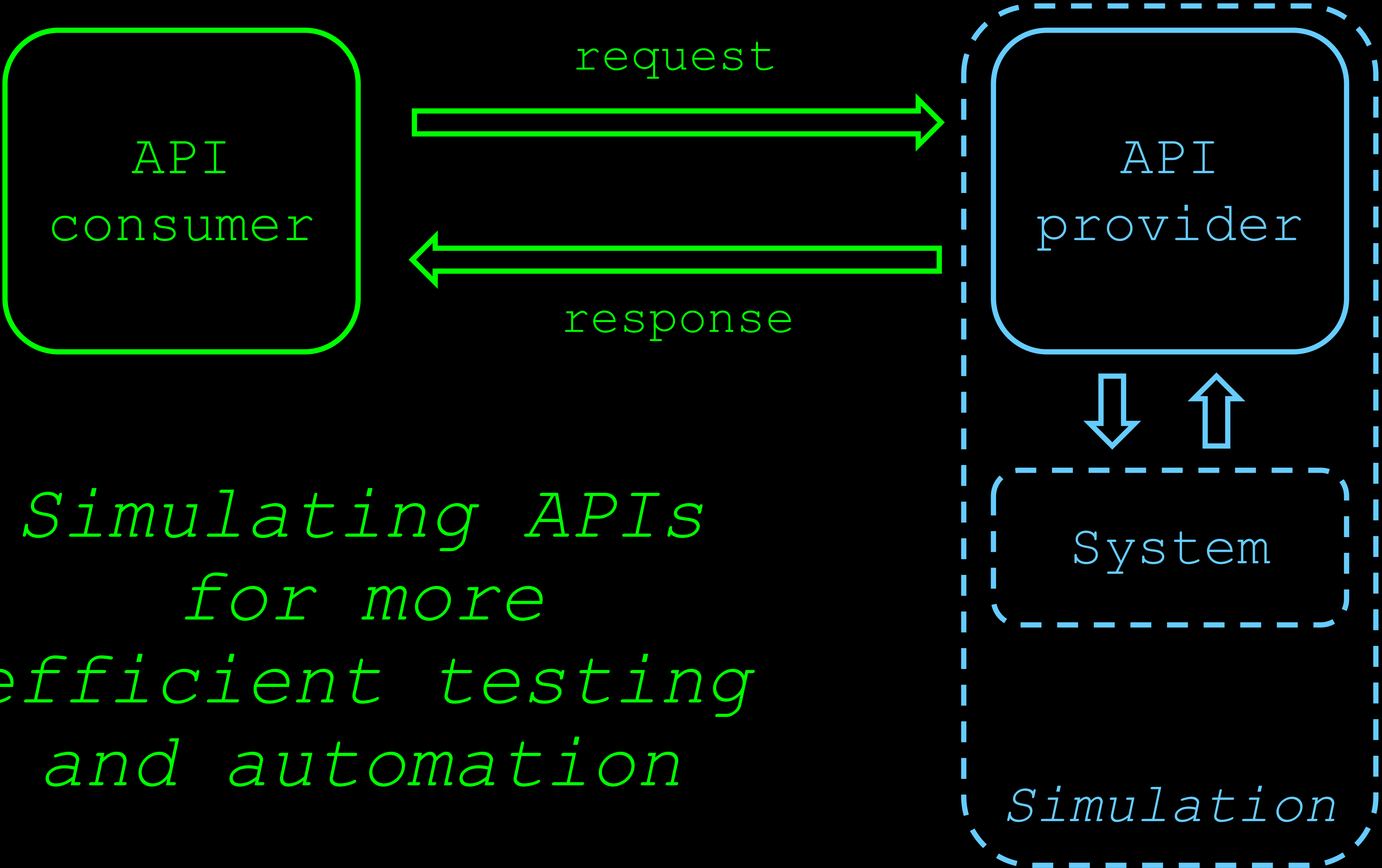
Problems in test environments



Simulation in test environments




request
API consumer
API provider
response
System
System



Simulating APIs for more efficient testing and automation

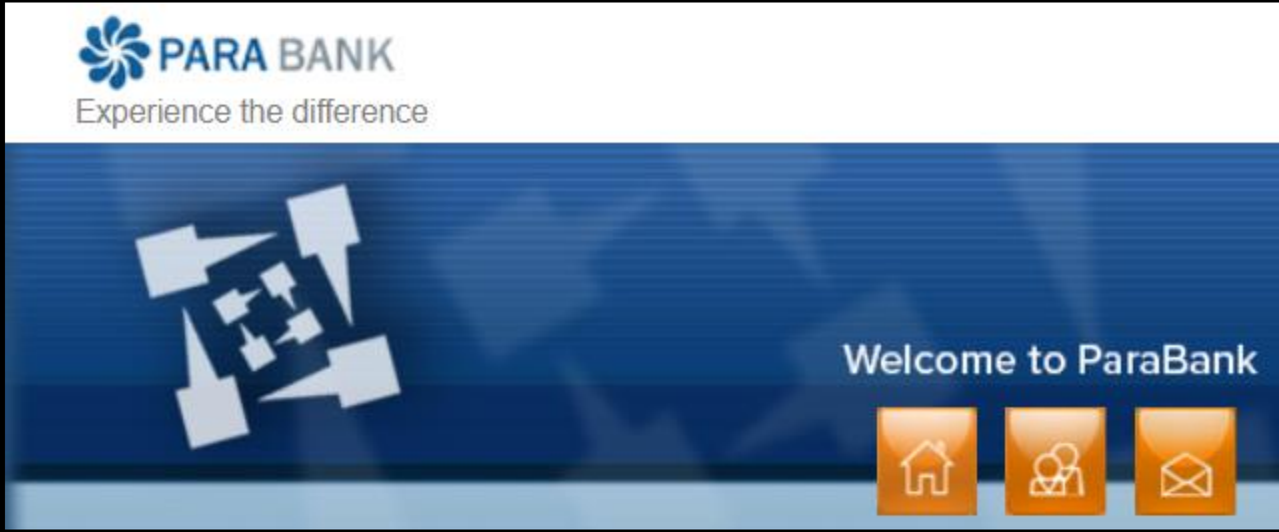
Our system under test

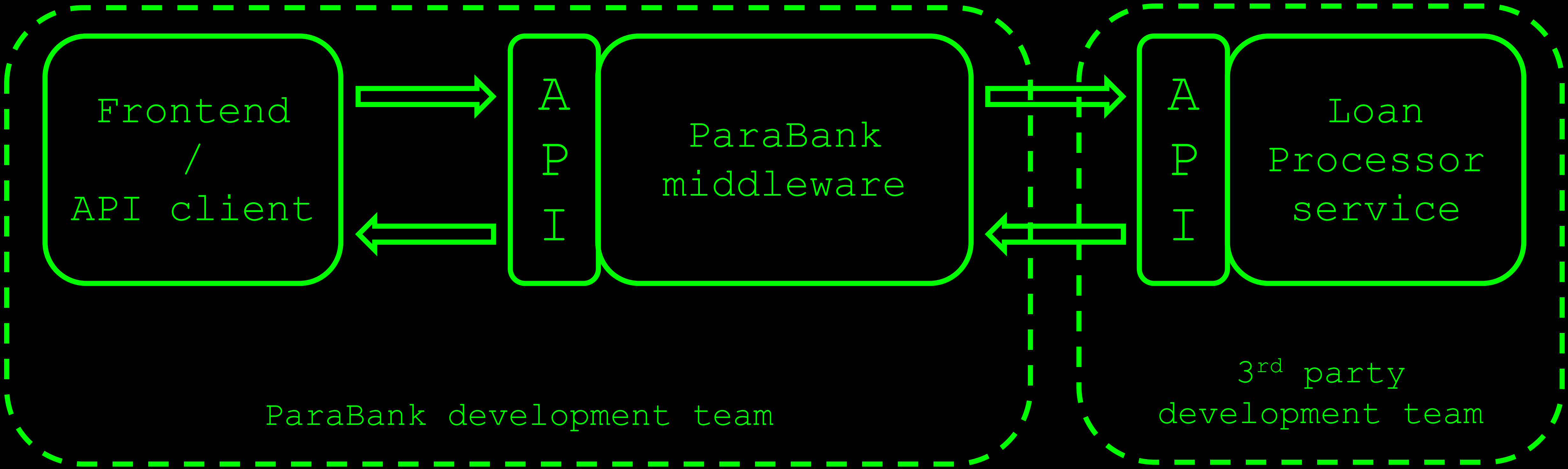
_ParaBank

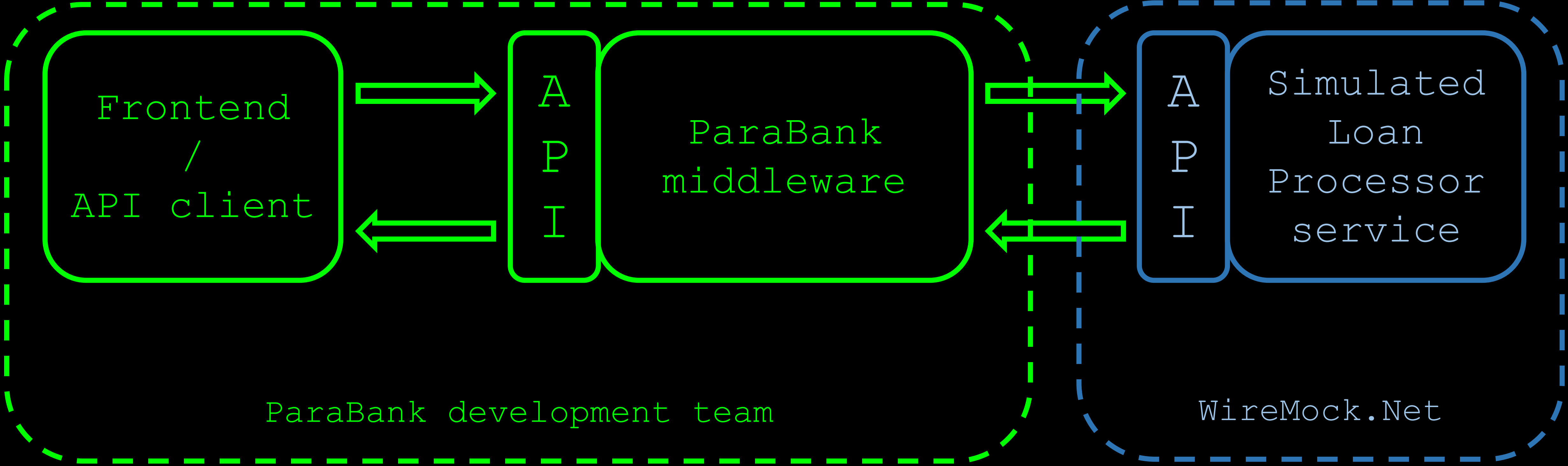
_The world's least safe online bank

_Request Loan process

_Loan application is processed by 3rd party loan provider component




Frontend / API client
API
ParaBank middleware
ParaBank development team
API
Loan Processor service
3rd party development team


Frontend
/
API client
A
P
I
ParaBank
middleware
A
P
I
Simulated
Loan
Processor
service
ParaBank development team
WireMock.Net

Start testing against features under development

Easy setup of state for edge cases

What might we want to simulate?

Delays, fault status codes, malformatted responses, …

…

Section 1:

Getting started with WireMock.Net

WireMock

_https://github.com/WireMock-Net/WireMock.Net/

_C#

_HTTP mock server
    _only supports HTTP(S)

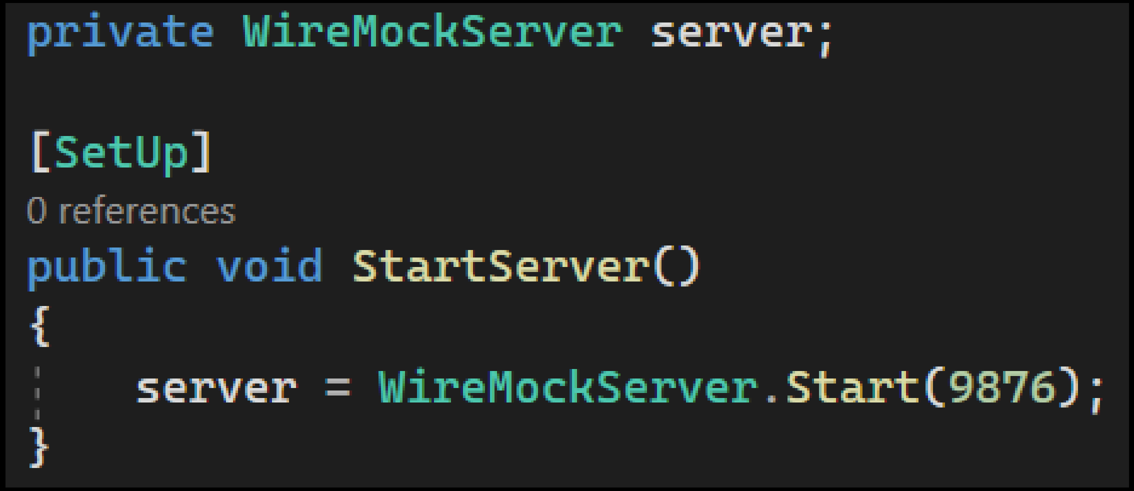
_open source
    _developed and maintained by Stef Heyenrath

Configuring WireMock.Net

_Install as a NuGet package

Starting and stopping the WireMock.Net server

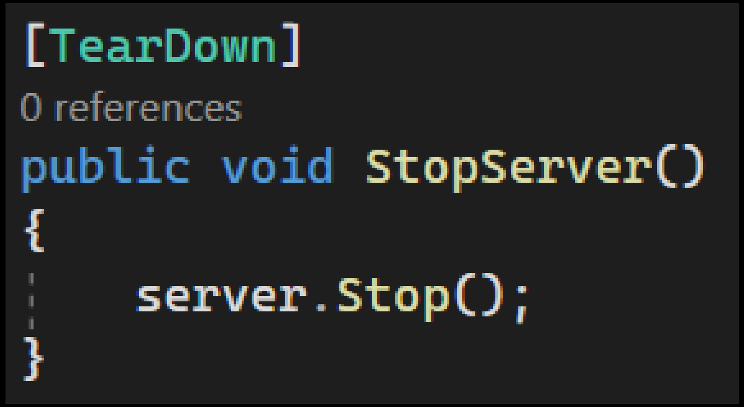
_Starting the server



```
private WireMockServer server;

[SetUp]
0 references
public void StartServer()
{
    server = WireMockServer.Start(9876);
}
```

_Stopping the server



```
[TearDown]
0 references
public void StopServer()
{
    server.Stop();
}
```

Starting WireMock.Net (standalone)

_Useful for exploratory testing purposes

_Allows you to share WireMock.Net instances between teams

_Long-running instances

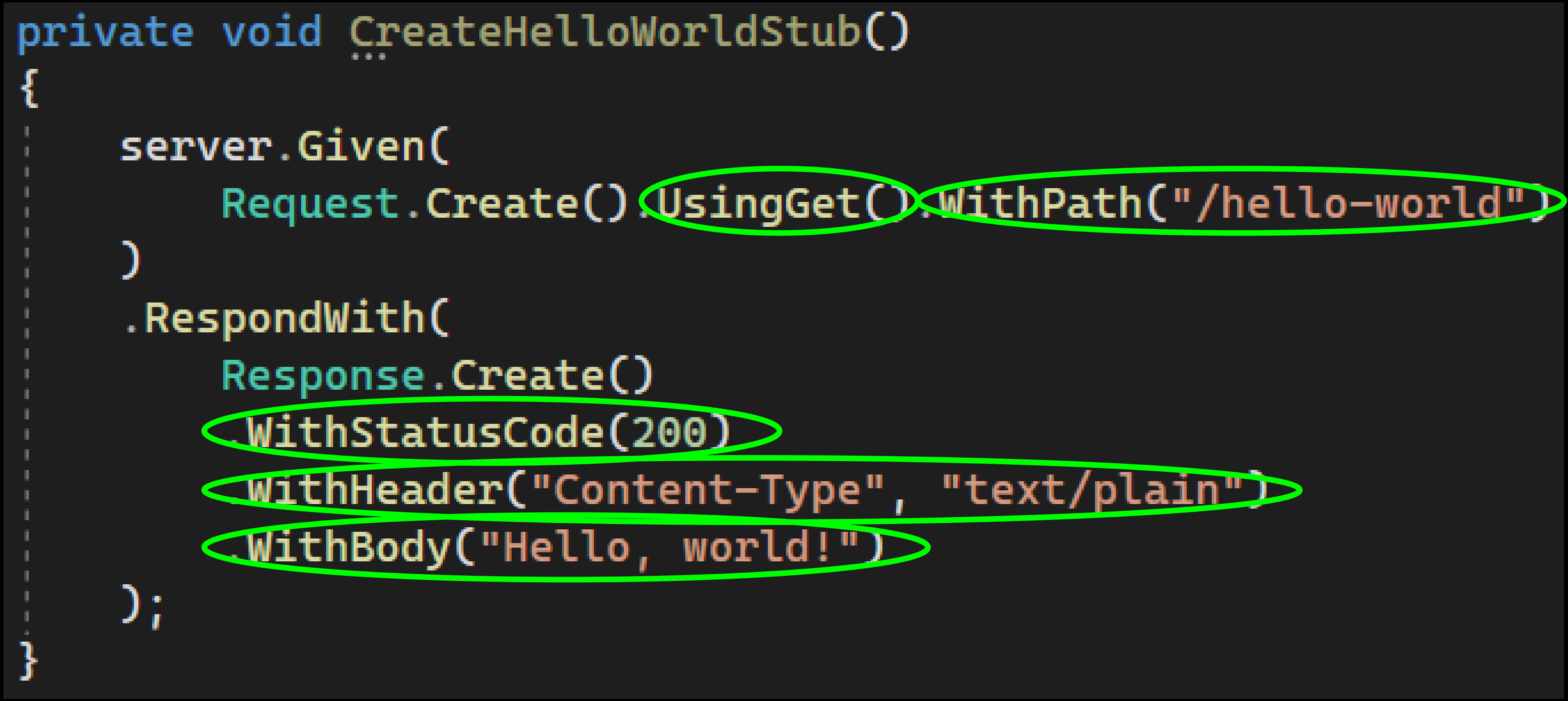
_Use WireMock.Net.StandAlone library

_We're not going to run WireMock in that mode in this workshop

Configure responses

- In C# code
  - This is what we'll do in the workshop
- Using JSON mapping files
  - See https://github.com/WireMock-Net/WireMock.Net/wiki/Stubbing#json-mapping-example for an example how to get started with JSON mapping files

An example mock defined in C#



```
private void CreateHelloWorldStub()
{
    server.Given(
        Request.Create().UsingGet().WithPath("/hello-world")
    )
    .RespondWith(
        Response.Create()
        .WithStatusCode(200)
        .WithHeader("Content-Type", "text/plain")
        .WithBody("Hello, world!")
    );
}
```

Now it's your turn!

_Exercises > Exercises01.cs

_Create a couple of basic mocks

  _Implement the responses as described in the comments

_Verify your solution by running the tests in the same class

_Answers are in Answers > Answers01.cs

_Examples are in Examples > Examples01.cs

Section 2:

Request matching strategies and fault simulation

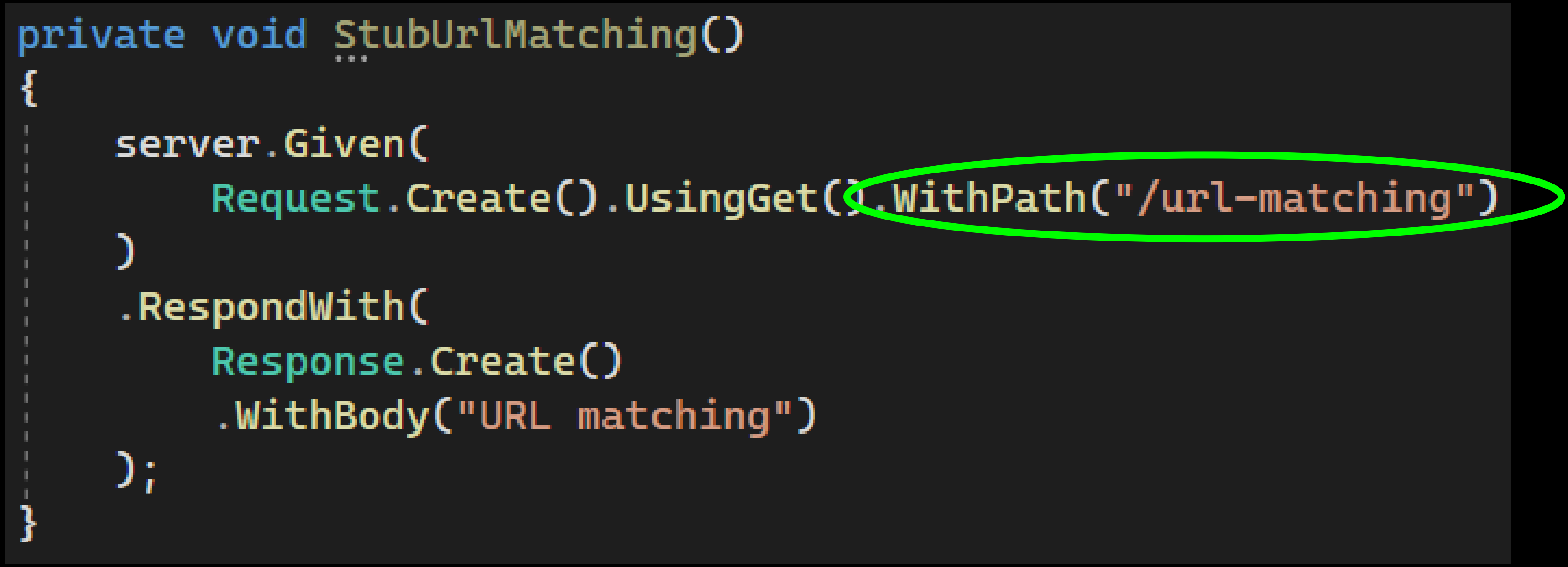
Request matching

_Send a response only when certain properties in the request are matched

_Options for request matching:

- _URL
- _HTTP method
- _Query parameters
- _Headers
- _Request body elements
- _…

Example: URL matching

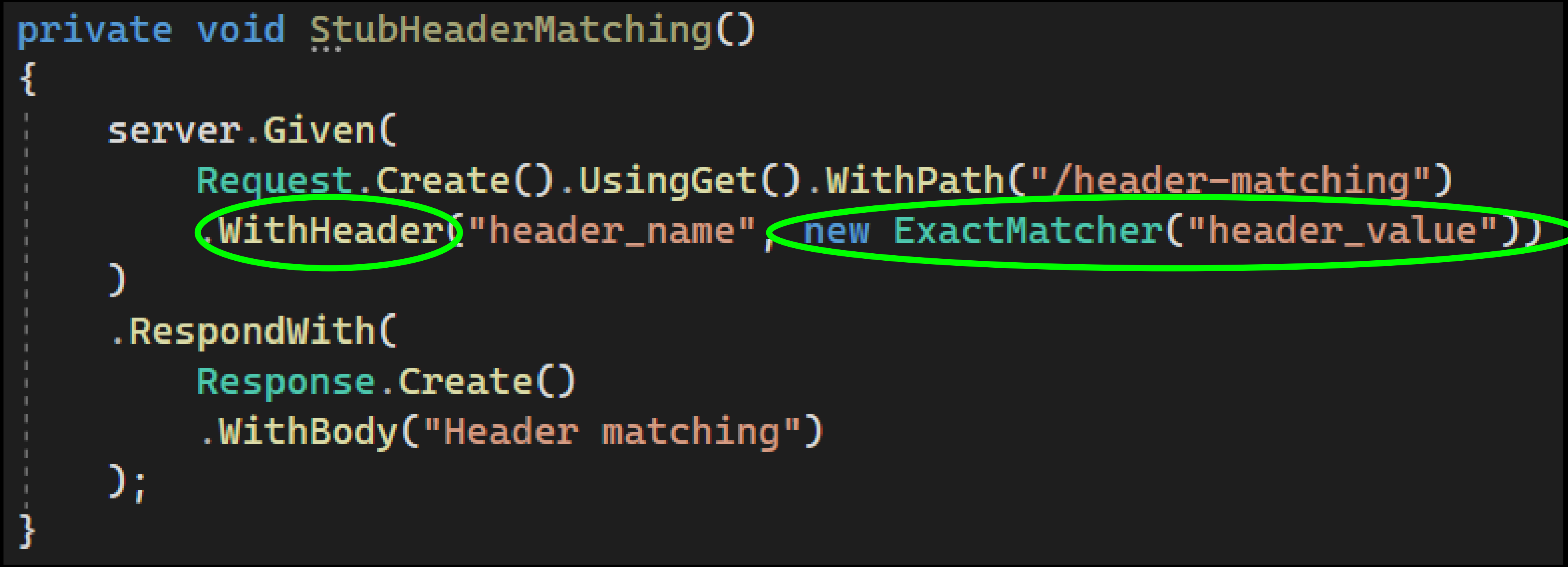


```
private void StubUrlMatching()
{
    server.Given(
        Request.Create().UsingGet().WithPath("/url-matching")
    )
    .RespondWith(
        Response.Create()
        .WithBody("URL matching")
    );
}
```

_Matchers can be used for more flexible matching

  _https://github.com/WireMock-Net/WireMock.Net/wiki/Request-Matching#two-matchers

Example: header matching

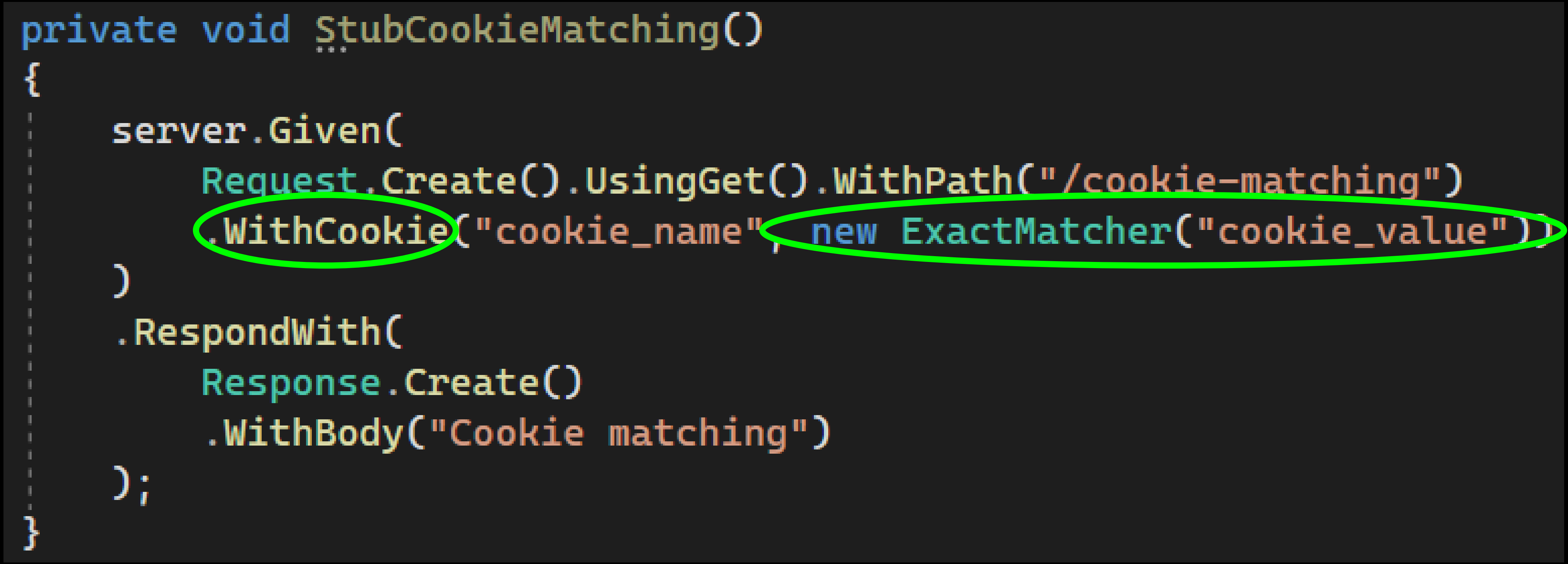


```
private void StubHeaderMatching()
{
    server.Given(
        Request.Create().UsingGet().WithPath("/header-matching")
        .WithHeader("header_name", new ExactMatcher("header_value"))
    )
    .RespondWith(
        Response.Create()
        .WithBody("Header matching")
    );
}
```

_Matchers can be used for flexible matching

  _https://github.com/WireMock-Net/WireMock.Net/wiki/Request-Matching#two-matchers

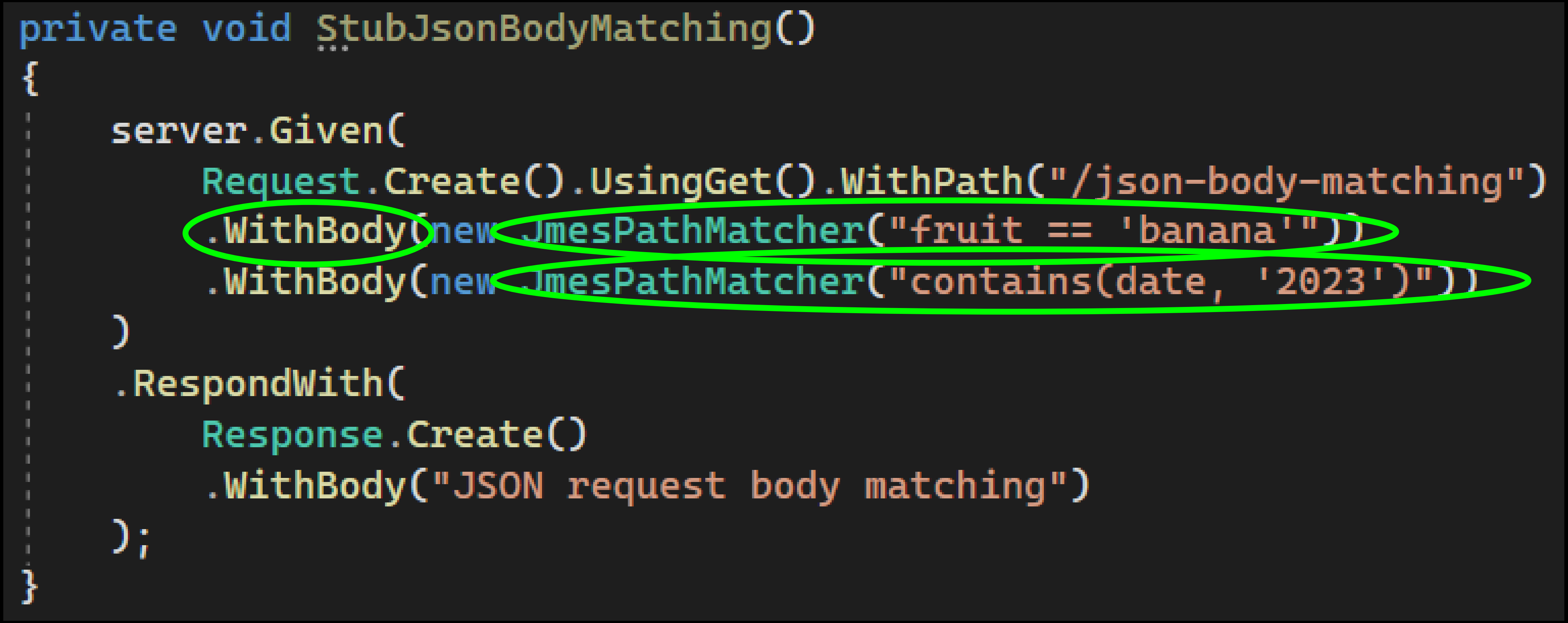
Example: cookie matching



```
private void StubCookieMatching()
{
    server.Given(
        Request.Create().UsingGet().WithPath("/cookie-matching")
        .WithCookie("cookie_name", new ExactMatcher("cookie_value"))
    )
    .RespondWith(
        Response.Create()
        .WithBody("Cookie matching")
    );
}
```

- Matchers can be used for flexible matching
  - https://github.com/WireMock-Net/WireMock.Net/wiki/Request-Matching#two-matchers

Example: JSON body matching



```
private void StubJsonBodyMatching()
{
    server.Given(
        Request.Create().UsingGet().WithPath("/json-body-matching")
        .WithBody(new JmesPathMatcher("fruit == 'banana'"))
        .WithBody(new JmesPathMatcher("contains(date, '2023')"))
    )
    .RespondWith(
        Response.Create()
        .WithBody("JSON request body matching")
    );
}
```

- Matchers can be used for flexible matching
  - https://github.com/WireMock-Net/WireMock.Net/wiki/Request-Matching#two-matchers

Fault simulation

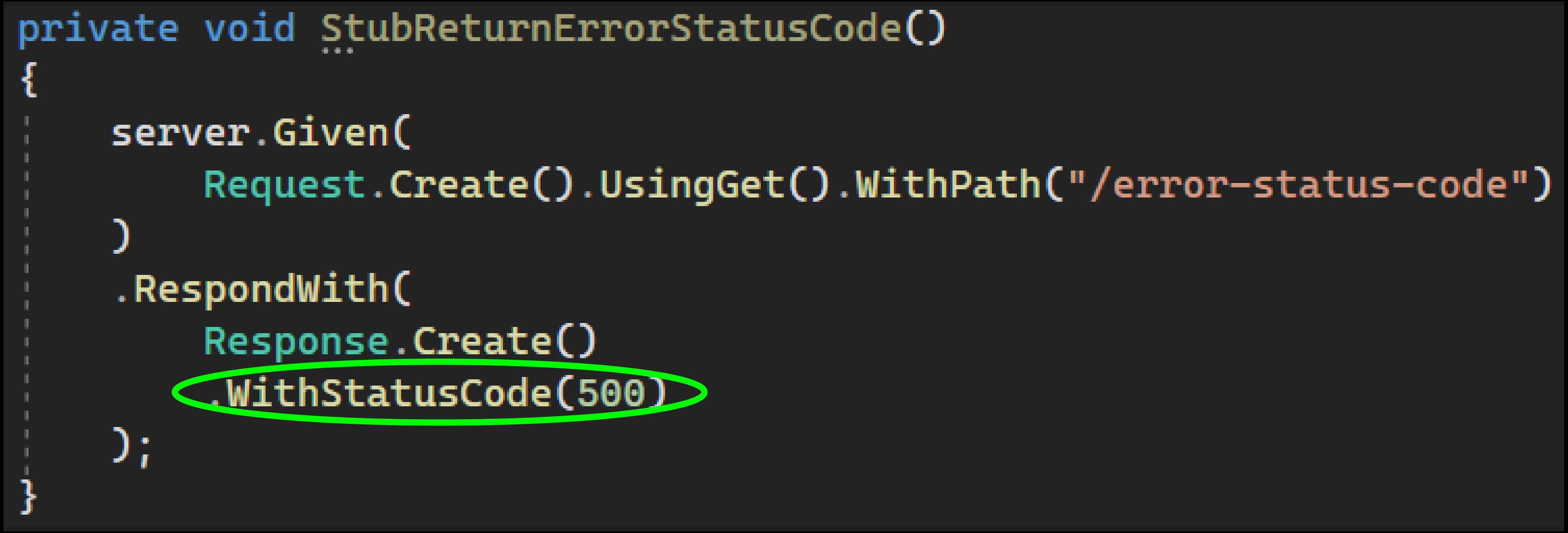
_Extend test coverage by simulating faults

_Often hard to do in real systems

_Easy to do using stubs or mocks

_Used to test the exception handling of your application under test

Example: HTTP status code

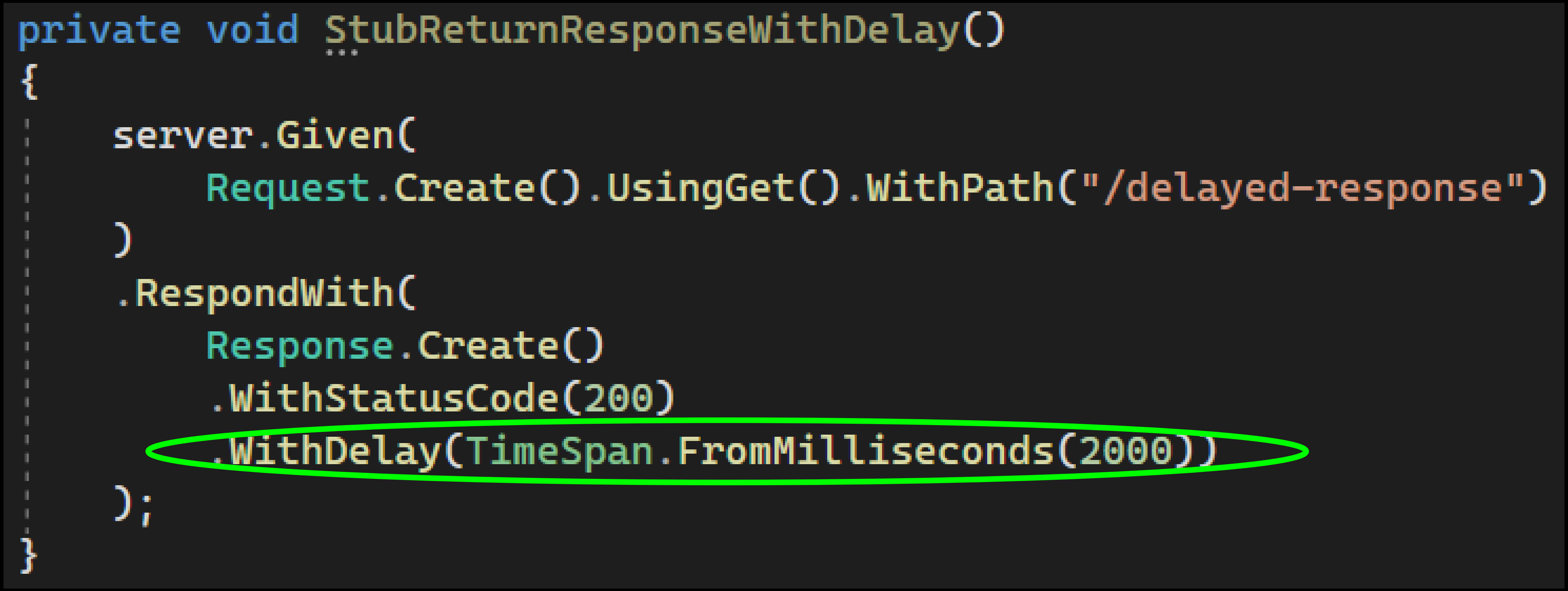


```
private void StubReturnErrorStatusCode()
{
    server.Given(
        Request.Create().UsingGet().WithPath("/error-status-code")
    )
    .RespondWith(
        Response.Create()
        .WithStatusCode(500)
    );
}
```

_Some often used HTTP status codes:

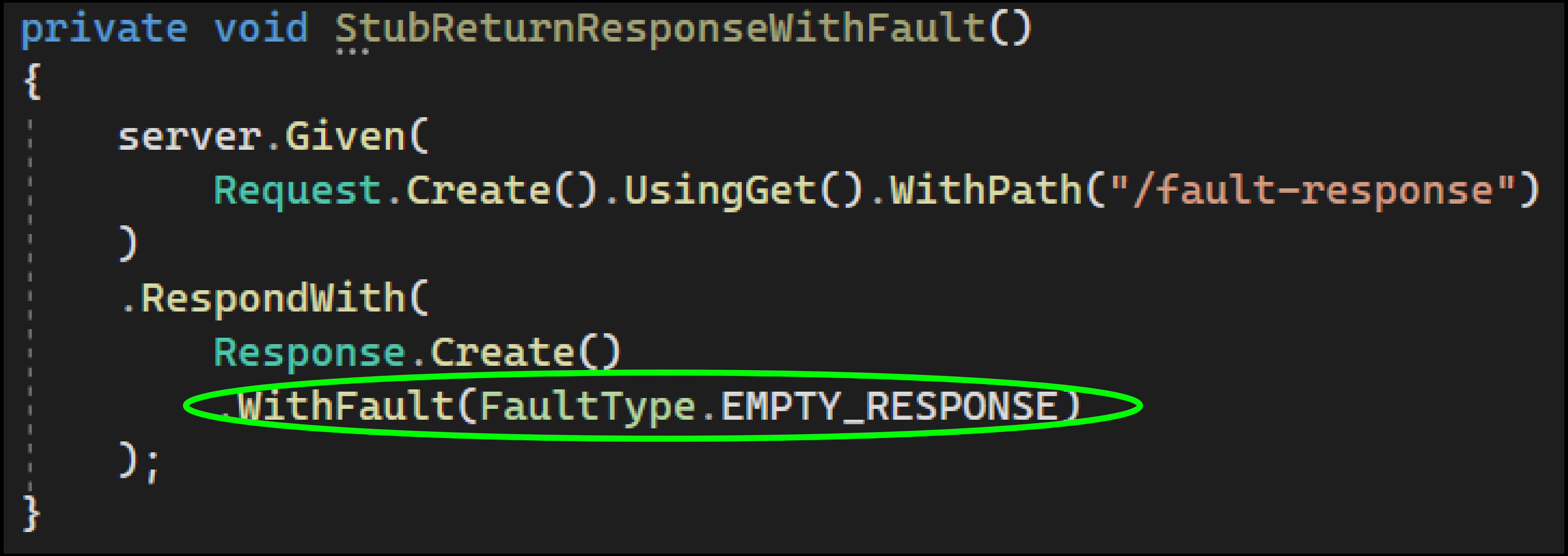
Consumer error	**Provider error**
403 (Forbidden)	500 (Internal server error)
404 (Not found)	503 (Service unavailable)

Example: delays



```
private void StubReturnResponseWithDelay()
{
    server.Given(
        Request.Create().UsingGet().WithPath("/delayed-response")
    )
    .RespondWith(
        Response.Create()
        .WithStatusCode(200)
        .WithDelay(TimeSpan.FromMilliseconds(2000))
    );
}
```

Example: faults

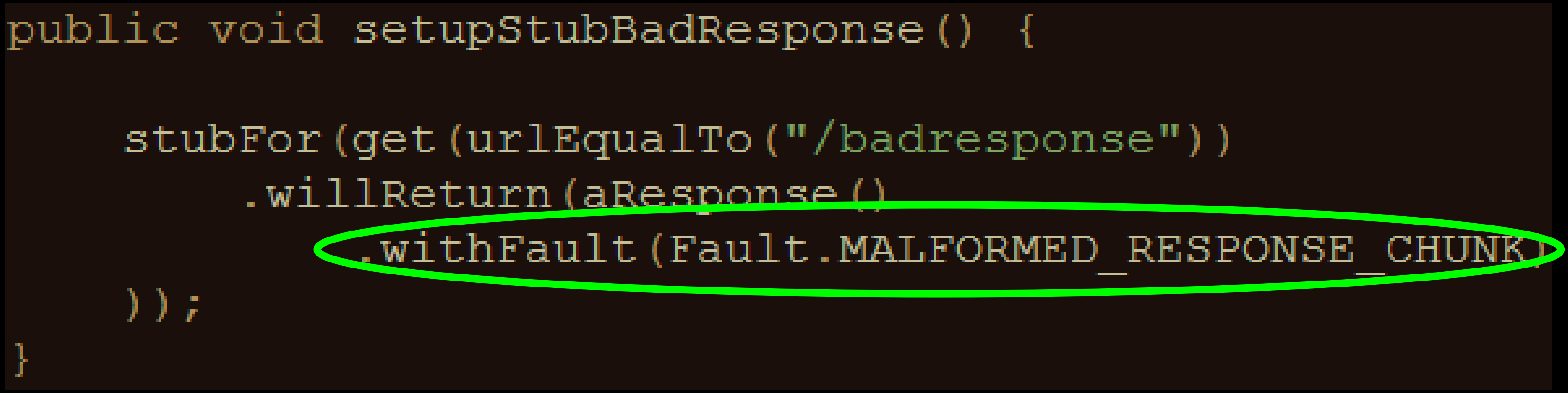


```
private void StubReturnResponseWithFault()
{
    server.Given(
        Request.Create().UsingGet().WithPath("/fault-response")
    )
    .RespondWith(
        Response.Create()
        .WithFault(FaultType.EMPTY_RESPONSE)
    );
}
```

_Options:

- _NONE (no fault)
- _EMPTY_RESPONSE (does what it says on the tin)
- _MALFORMED_RESPONSE_CHUNK (HTTP 200, garbage in body)

Example: bad responses (Java)



```
public void setupStubBadResponse() {

    stubFor(get(urlEqualTo("/badresponse"))
        .willReturn(aResponse()
            .withFault(Fault.MALFORMED_RESPONSE_CHUNK)
    ));
}
```

_HTTP status code 200, but garbage in response body

_Other options:
   _RANDOM_DATA_THEN_CLOSE (as above, without HTTP 200)
   _EMPTY_RESPONSE (does what it says on the tin)
   _CONNECTION_RESET_BY_PEER (close connection, no response)

Now it's your turn!

- _Exercises > Exercises02.cs
- _Practice fault simulation and different request matching strategies
  - _Implement the responses as described in the comments
- _Verify your solution by running the tests in the same class
- _Answers are in Answers > Answers02.cs
- _Examples are in Examples > Examples02.cs

Section 3:

Creating stateful mocks

Statefulness

_Sometimes, you want to simulate stateful behaviour

_Shopping cart (empty / containing items)

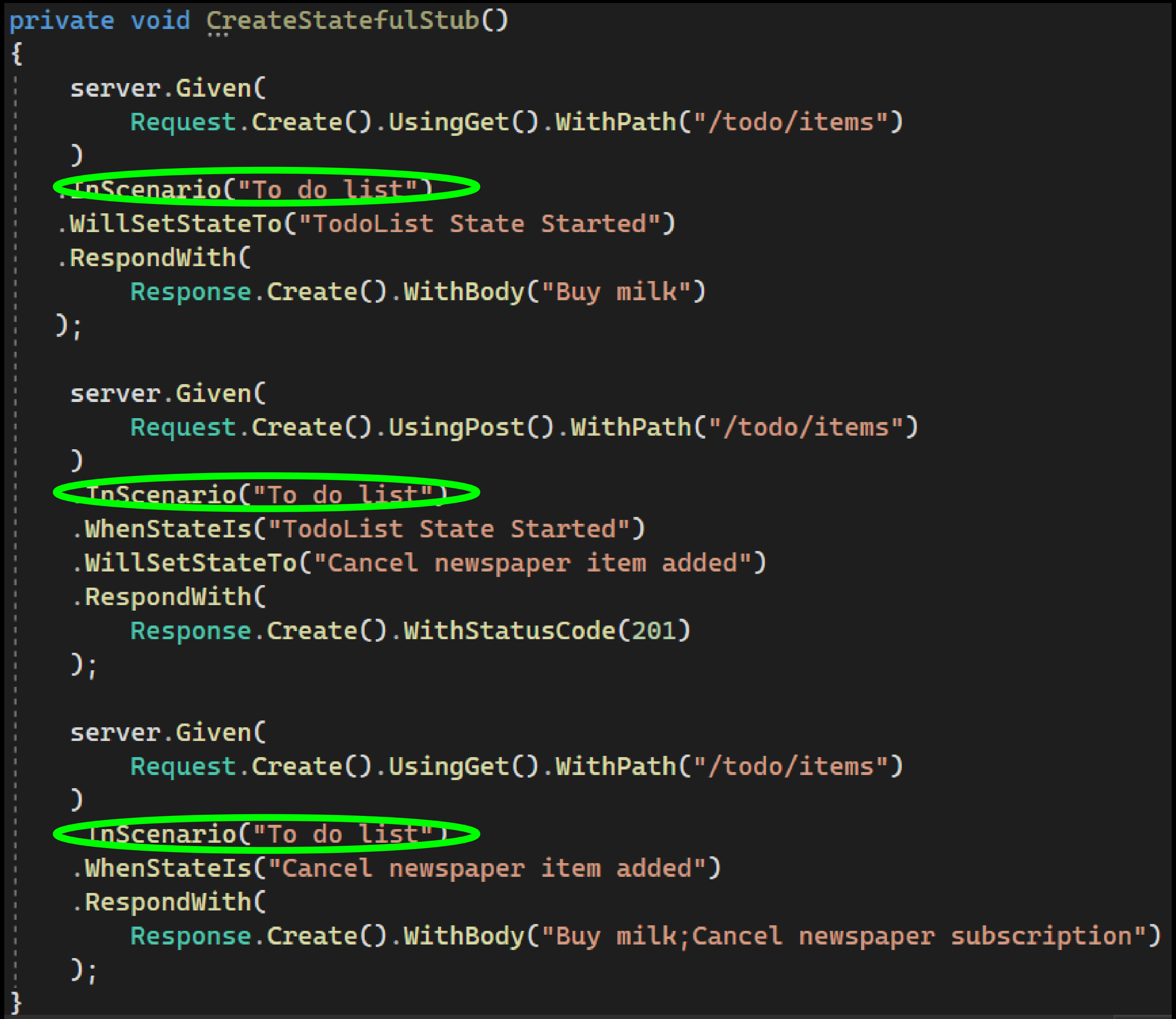
_Database (data present / not present)

_Order in which requests arrive is significant

Stateful mocks in WireMock.Net

- _Supported through the concept of a Scenario
- _Essentially a finite state machine (FSM)
  - _States and state transitions
- _Combination of current state and incoming request determines the response being sent
  - _Before now, it was only the incoming request

Stateful mocks: an example



```
private void CreateStatefulStub()
{
    server.Given(
        Request.Create().UsingGet().WithPath("/todo/items")
    )
   .InScenario("To do list")
   .WillSetStateTo("TodoList State Started")
   .RespondWith(
        Response.Create().WithBody("Buy milk")
   );

    server.Given(
        Request.Create().UsingPost().WithPath("/todo/items")
    )
    .InScenario("To do list")
    .WhenStateIs("TodoList State Started")
    .WillSetStateTo("Cancel newspaper item added")
    .RespondWith(
        Response.Create().WithStatusCode(201)
    );

    server.Given(
        Request.Create().UsingGet().WithPath("/todo/items")
    )
    .InScenario("To do list")
    .WhenStateIs("Cancel newspaper item added")
    .RespondWith(
        Response.Create().WithBody("Buy milk;Cancel newspaper subscription")
    );
}
```

Responses are grouped by scenario name

Response depends on both the incoming request as well as the current state

The first mock should define the initial state

Incoming requests can trigger state transitions

Now it’s your turn!

- _Exercises > Exercises03.cs
- _Create a stateful mock that exerts the described behaviour
  - _Implement the responses as described in the comments
- _Verify your solution by running the tests in the same class
- _Answers are in Answers > Answers03.cs
- _Examples are in Examples > Examples03.cs

Section 4:

Response templating

Response templating

_Often, you want to reuse elements from the request in the response

  _Request ID header

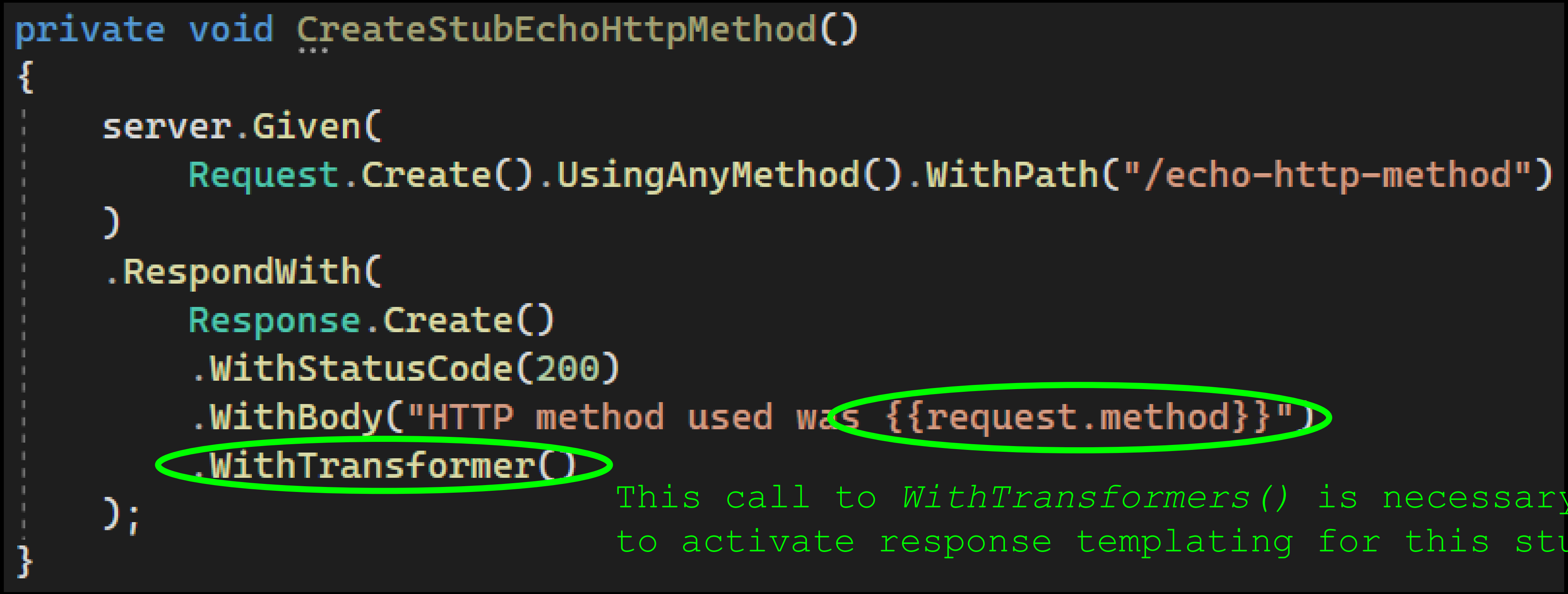
  _Unique body elements (client ID, etc.)

  _Cookie values

_WireMock.Net supports this through response templating

Enable/apply response templating

- This template reads the HTTP request method (GET/POST/PUT/…) using *{{request.method}}* and returns it as the response body



```
private void CreateStubEchoHttpMethod()
{
    server.Given(
        Request.Create().UsingAnyMethod().WithPath("/echo-http-method")
    )
    .RespondWith(
        Response.Create()
        .WithStatusCode(200)
        .WithBody("HTTP method used was {{request.method}}")
        .WithTransformer()
    );
}
```

This call to *WithTransformers()* is necessary to activate response templating for this stub

Request attributes

_Many different request attributes available for use

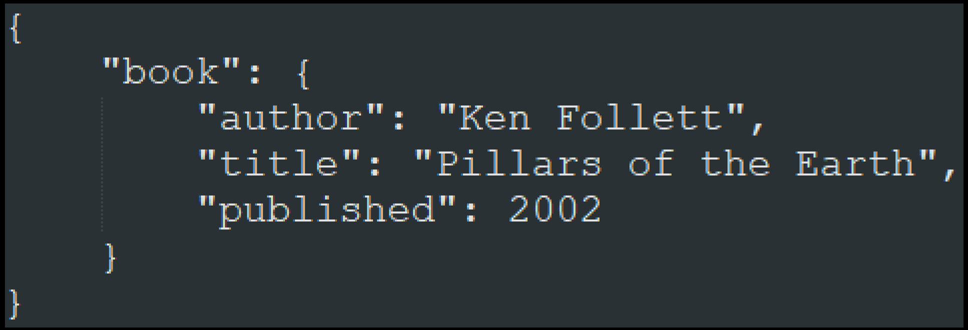
_request.method	: HTTP method (example)
_request.PathSegments.[<n>]	: nth path segment
_request.query.<key>	: query parameter value
_…	

_All available attributes listed at

https://github.com/WireMock-Net/WireMock.Net/wiki/Response-Templating#the-request-model

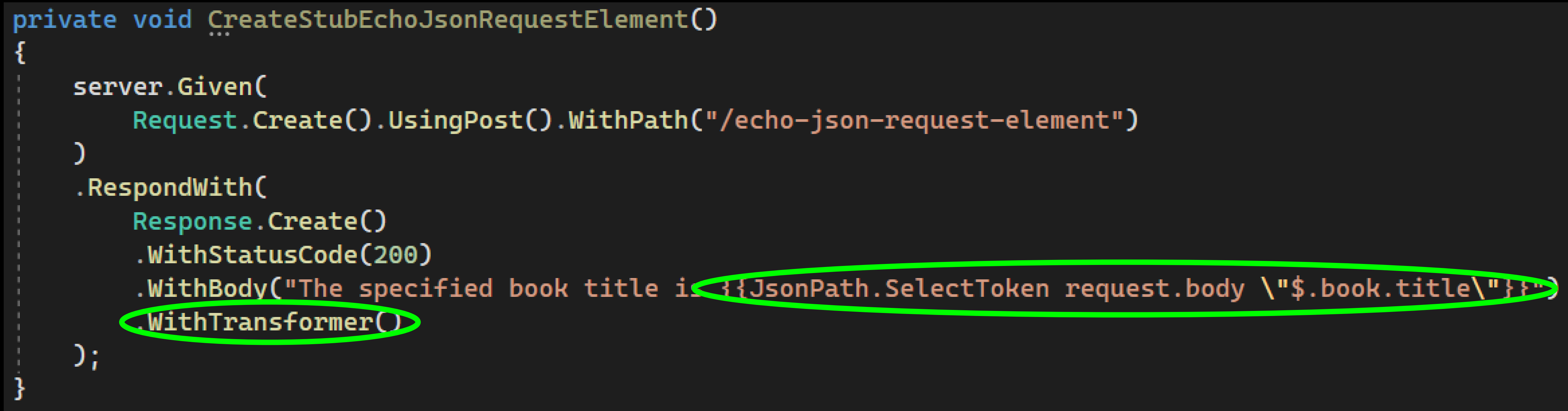
JSON extraction example

_When sent this JSON request body:



```
{
    "book": {
        "author": "Ken Follett",
        "title": "Pillars of the Earth",
        "published": 2002
    }
}
```

_This stub returns a response with body “The specified book title is Pillars of the Earth”:



```
private void CreateStubEchoJsonRequestElement()
{
    server.Given(
        Request.Create().UsingPost().WithPath("/echo-json-request-element")
    )
    .RespondWith(
        Response.Create()
        .WithStatusCode(200)
        .WithBody("The specified book title is {{JsonPath.SelectToken request.body \"$.book.title\"}}")
        .WithTransformer()
    );
}
```

Now it's your turn!

_Exercises > Exercises04.cs

_Create dynamic mock by using response templating

  _Implement the responses as described in the comments

_Verify your solution by running the tests in the same class

_Answers are in Answers > Answers04.cs

_Examples are in Examples > Examples04.cs

?