Testing HTTP-based APIs using RestAssured.Net

An open source workshop by …

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

What are we going to do?

_HTTP-based APIs

_RestAssured.Net

_Hands-on exercises

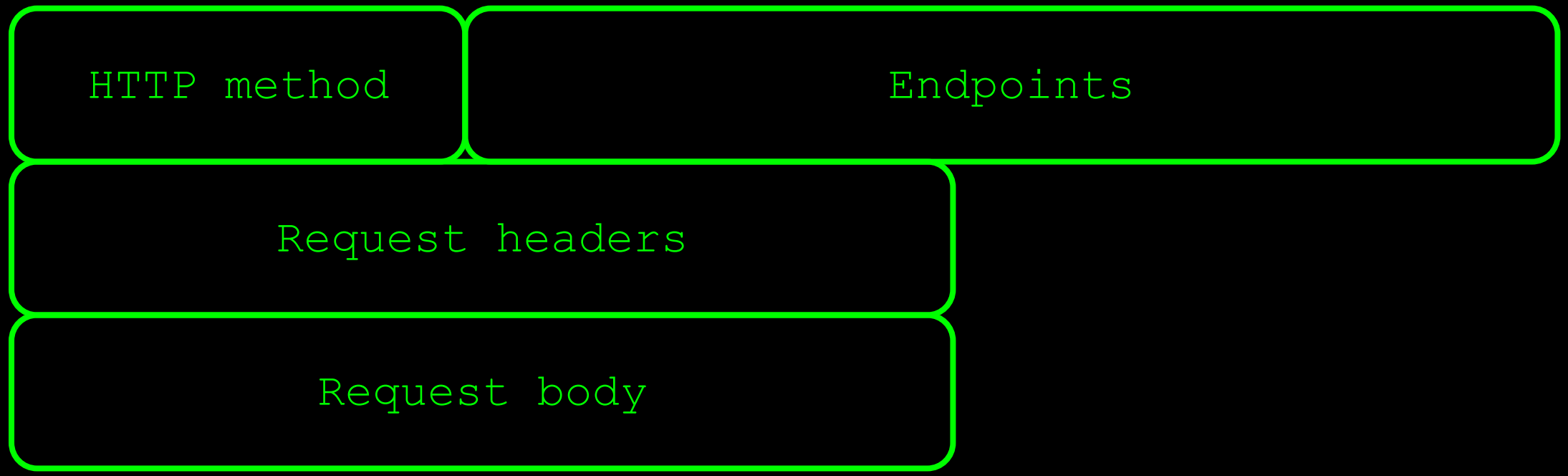
Preparation

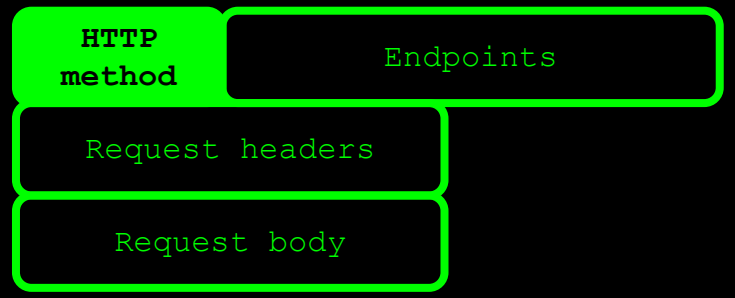
_Install a recent .NET SDK (.NET 6 or newer)

_Install Visual Studio (or any other IDE)

_Import project into your IDE
  _https://github.com/basdijkstra/rest-assured-net-workshop

A REST API request





HTTP methods

- GET, POST, PUT, PATCH, DELETE, OPTIONS, …

- CRUD operations on data

POST	Create
GET	Read
PUT / PATCH	Update
DELETE	Delete
…	…

- Conventions, not standards!

Endpoints

- Uniquely identifies the operation to perform and / or the resource to operate on

- Can contain parameters
  - Query parameters
  - Path parameters

Endpoint parameters

_Path parameters

_http://api.zippopotam.us/us/90210

_http://api.zippopotam.us/ca/B2A

_Query parameters

_http://md5.jsontest.com/?text=testcaseOne

_http://md5.jsontest.com/?text=testcaseTwo

_There is no official standard!

Request headers

- Key-value pairs

- Can contain metadata about the request body
  - Content-Type (what data format is the request body in?)
  - Accept (what data format would I like the response body to be in?)
  - …

- Can contain session and authorization data
  - Cookies
  - Authorization tokens
  - …

HTTP method

Endpoints

Request headers

Request body

Authorization: Basic

- Username and password sent with every request
- Base64 encoded (not at all secure!)
- Ex: username = aladdin and password = opensesame

Authorization: Basic YWxhZGRpbjpvcGVuc2VzYW1l

HTTP method

Endpoints

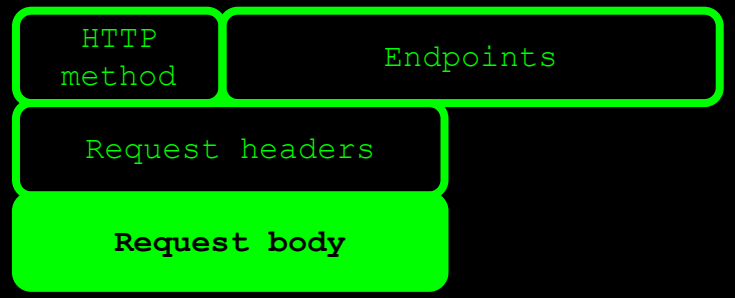
Request headers

Request body

Authorization: Bearer

- Token with expiry date is obtained first
- Token is then sent with all subsequent requests
- Most common mechanism is OAuth(2)
- JWT is a common token format

Authorization: Bearer RsT5OjbzRn430zqMLgV3Ia



Request body

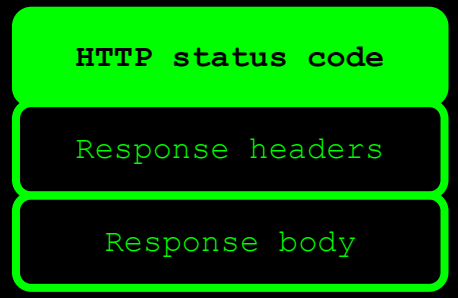
- Data to be sent to the provider
- REST does not prescribe a specific data format
- Most common:
  - JSON
  - XML
  - Plain text
- Other data formats can be sent using REST, too

A REST API response

HTTP status code

Response headers

Response body



HTTP status code

_Indicates result of request processing by provider

_Five different categories

_1XX	Informational	100 Continue
_2XX	Success	200 OK
_3XX	Redirection	301 Moved Permanently
_4XX	Consumer errors	400 Bad Request
_5XX	Provider errors	500 Internal Server Error

HTTP status code

Response headers

Response body

Response headers

- Key-value pairs

- Can contain metadata about the response body
  - Content-Type (what data format is the response body in?)
  - Content-Length (how many bytes in the response body?)

- Can contain provider-specific data
  - Caching-related headers
  - Information about the server type

HTTP status code

Response headers

Response body

Response body

- Data returned by the provider

- REST does not prescribe a specific data format

- Most common:
  - JSON
  - XML
  - Plain text

- Other data formats can be sent using REST, too

API specification / documentation

_SOAP: WSDL (Web Services Description Language)

_REST: OpenAPI / Swagger

_REST: RAML (RESTful API Modelling Language)

_REST: WADL (Web Application Description Language)

Tools for testing RESTful APIs

- Free / open source
  - Postman
  - SoapUI
  - Code libraries (REST Assured, RestAssured.Net, RestSharp, requests, …)

- Commercial
  - Parasoft SOAtest
  - SmartBear ReadyAPI
  - …

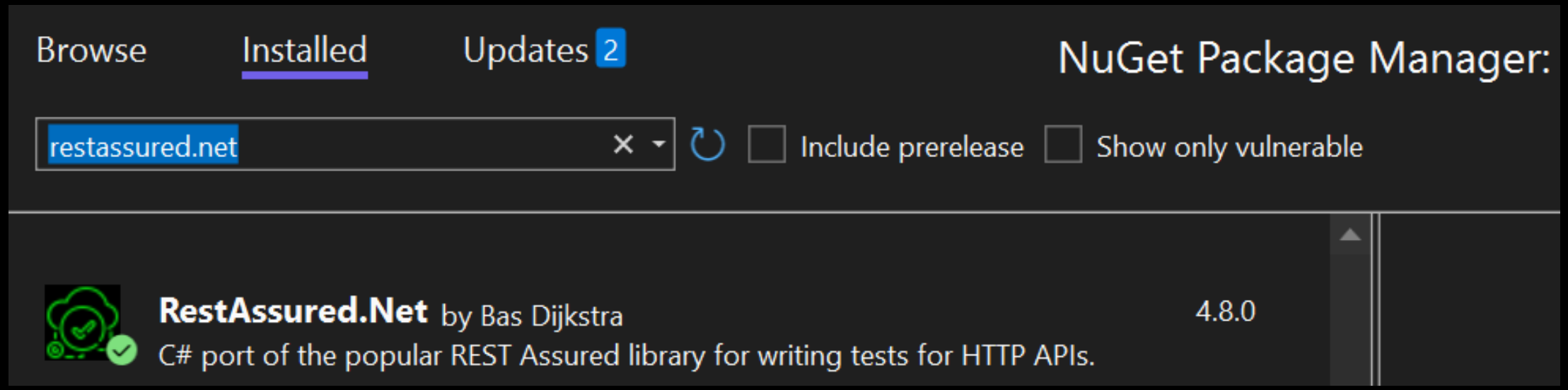
- Build your own (using HTTP libraries for your language of choice)

RestAssured.Net

- C# DSL for writing tests for RESTful APIs
- Port of REST Assured (Java)
- Removes a lot of boilerplate code
- Runs on top of common unit testing frameworks
  - NUnit, xUnit, MSTest
- Developed and maintained by Bas Dijkstra

Adding RestAssured.Net to your project

_Available as a NuGet package



RestAssured.Net documentation

_Usage Guide

https://github.com/basdijkstra/rest-assured-net/wiki/Usage-Guide

_Examples

https://github.com/basdijkstra/rest-assured-net/tree/main/RestAssured.Net.Tests

(these double as acceptance tests for the project)

A sample test

RestAssured.Net uses NUnit (this could also be xUnit or MSTest)

```
[Test]
0 references
public void GetUserData_VerifyName_ShouldBeLeanneGraham()
{
    Given()
        .When()
        .Get("https://jsonplaceholder.typicode.com/users/1")
        .Then()
        .AssertThat()
        .Body("$.name", NHamcrest.Is.EqualTo("Leanne Graham"));
}
```

Make an HTTP GET call to retrieve data from the provider

Perform an assertion on the returned response (here: on the JSON response payload)

RestAssured.Net features

- Support for all HTTP methods (GET, POST, PUT, …)
- Support for Gherkin (Given/When/Then)
- Use of NHamcrest matchers for checks (*Is.EqualTo*)
- Use of JsonPath to select elements from a JSON response

```
[Test]
| 0 references
public void GetUserData_VerifyName_ShouldBeLeanneGraham()
{
    Given()
        .When()
        .Get("https://jsonplaceholder.typicode.com/users/1")
        .Then()
        .AssertThat()
        .Body("$.name", NHamcrest.Is.EqualTo("Leanne Graham"));
}
```

About NHamcrest matchers

_Express expectations in natural language

_Examples:

`Is.EqualTo(X)`	Does the object equal X?
`Has.Item(Is.EqualTo(X))`	Does the collection contain an item equal to X?

About JsonPath

_JsonPath is a query language for JSON documents

_Similar aims and scope as XPath for XML

_Documentation and examples:
  _https://support.smartbear.com/alertsite/docs/monitors/api/endpoint/jsonpath.html

_All required JsonPath expressions are given in the exercise descriptions

Checking technical response data

- HTTP status code
- Response Content-Type header
- Other headers and their value
- Cookies and their value
- …

- Most methods accept both fixed values and NHamcrest matchers

```
[Test]
| 0 references
public void GetUserData_VerifyStatusCodeAndContentType()
{
    Given()
        .When()
        .Get("https://jsonplaceholder.typicode.com/users/1")
        .Then()
        .AssertThat()
        .StatusCode(200)
        .And()
        .ContentType("application/json; charset=utf-8");
}
```

Logging request data

```
[Test]
0 references
public void LogAllRequestAndResponseData()
{
    var logConfiguration = new LogConfiguration
    {
        RequestLogLevel = RequestLogLevel.All,
        ResponseLogLevel = ResponseLogLevel.All,
    };

    Given()
        .Log(logConfiguration)
        .When()
        .Get("https://jsonplaceholder.typicode.com/users/1")
        .Then()
        .AssertThat()
        .Body("$.name", NHamcrest.Is.EqualTo("Leanne Graham"));
}
```

You can also use *RequestLogLevel.Body*, *RequestLogLevel.Headers* as well as other options

Logging request data

```
[Test]
0 references
public void LogAllRequestAndResponseData()
{
    var logConfiguration = new LogConfiguration
    {
        RequestLogLevel = RequestLogLevel.All,
        ResponseLogLevel = ResponseLogLevel.All,
    };

    Given()
        .Log(logConfiguratio
        .When()
        .Get("https://jsonpl
        .Then()
        .AssertThat()
        .Body("$.name", NHam
}
```

```
LogAllRequestData
Source: Examples01.cs line 22
Duration: 30 ms

Standard Output:
  GET https://jsonplaceholder.typicode.com/users/1
  Content-Type: application/json; charset=utf-8
  Content-Length: 0
```

Logging response data

```
[Test]
0 references
public void LogAllRequestAndResponseData()
{
    var logConfiguration = new LogConfigur
    {
        RequestLogLevel = RequestLogLevel.
        ResponseLogLevel = ResponseLogLeve
    };

    Given()
        .Log(logConfiguration)
        .When()
        .Get("https://jsonplaceholder.typi
        .Then()
        .AssertThat()
        .Body("$.name", NHamcrest.Is.Equal
}
```

LogAllResponseData

Source: Examples01.cs line 34

Duration: 41 ms

Standard Output:

```
HTTP 200 (OK)
Content-Type: application/json; charset=utf-8
Content-Length: 509
Date: Thu, 25 Jan 2024 11:31:41 GMT
Connection: keep-alive
Report-To: {"group":"heroku-nel","max_age":3600,"endpoints":[{"
Reporting-Endpoints: heroku-nel=https://nel.heroku.com/reports?
Nel: {"report_to":"heroku-nel","max_age":3600,"success_fraction
X-Powered-By: Express
X-Ratelimit-Limit: 1000
X-Ratelimit-Remaining: 999
X-Ratelimit-Reset: 1698817725
Vary: Origin, Accept-Encoding
Access-Control-Allow-Credentials: true
Cache-Control: max-age=43200
Pragma: no-cache
X-Content-Type-Options: nosniff
ETag: W/"1fd-+2Y3G3w049iSZtw5t1mzSnunngE"
Via: 1.1 vegur
```

Our API under test

_(Simulation of) an online banking API

_Customer data (GET, POST)

_Account data (POST, GET)

_RESTful API

Demo

_How to use the test suite

_Executing your tests

_Reviewing test results

Now it’s your turn!

_Exercises > Exercises01.cs

_Simple checks
  _Validating individual elements
  _Validating collections and items therein
  _Validating technical response properties

_Stubs are predefined
  _Don’t worry about the references to http://localhost
  _You only need to write the tests using RestAssured.Net

_Answers are in Answers > Answers01.cs

_Examples are in Examples > Examples01.cs

Parameters in RESTful web services

_Path parameters

- _http://api.zippopotam.us/us/90210
- _http://api.zippopotam.us/ca/B2A

_Query parameters

- _http://md5.jsontest.com/?text=testcaseOne
- _http://md5.jsontest.com/?text=testcaseTwo

_There is no official standard!

Using query parameters

_GET http://md5.jsontest.com/?text=testcase

```
[Test]
0 references
public void UseQueryParameter()
{
    Given()
        .QueryParam("text", "testcase")
        .When()
        .Get("http://md5.jsontest.com")
        .Then()
        .Body("$.md5", NHamcrest.Is.EqualTo("7489a25fc99976f06fecb807991c61cf"));
}
```

Define a query parameter and its value

Using path parameters

_GET http://jsonplaceholder.typicode.com/users/1

```
[Test]
| 0 references
public void UsePathParameter()
{
    Given()
        .PathParam("userId", 1)
        .When()
        .Get("https://jsonplaceholder.typicode.com/users/[userId]")
        .Then()
        .Body("$.name", NHamcrest.Is.EqualTo("Leanne Graham"));
}
```

Define a (custom) path parameter name and the parameter value

Define the location of the path parameter using the chosen name between []

Exchange data between consumer and provider

GET to retrieve data from provider, POST to send data to provider, …

APIs are all about data

Business logic and calculations often exposed through APIs

Run the same test more than once…

… for different combinations of input and expected output values

Parameterized testing

More efficient to do this at the API level…

… as compared to doing this at the UI level

‘Feeding’ test data to your test

Define test cases using the [TestCase] attribute (one for every iteration with test data values separated by commas)

```
[TestCase(1, "Leanne Graham", TestName = "User with ID 1 is Leanne Graham")]
[TestCase(2, "Ervin Howell", TestName = "User with ID 2 is Ervin Howell")]
[TestCase(3, "Clementine Bauch", TestName = "User with ID 3 is Clementine Bauch")]
0 references
public void CheckNameForUser(int userId, string expectedUserName)
{
    Given()
        .PathParam("userId", userId)
        .When()
        .Get("https://jsonplaceholder.typicode.com/users/[userId]")
        .Then()
        .Body("$.name", NHamcrest.Is.EqualTo(expectedUserName));
}
```

Use parameters to pass the test data values into the method

Use parameters in the test method where appropriate

Running the parameterized test

```
[TestCase(1, "Leanne Graham", TestName = "User with ID 1 is Leanne Graham")]
[TestCase(2, "Ervin Howell", TestName = "User with ID 2 is Ervin Howell")]
[TestCase(3, "Clementine Bauch", TestName = "User with ID 3 is Clementine Bauch")]
0 references
public void CheckNameForUser(int userId, string expectedUserName)
{
    Given()
        .PathParam("userId", userId)
        .When()
        .Get("https://jsonplaceholder.typicode.com/users/[userId]")
        .Then()
                                                                    erName));
}
```

Examples02 (3)	1,6 sec
User with ID 1 is Leanne Graham	966 ms
User with ID 2 is Ervin Howell	342 ms
User with ID 3 is Clementine Bauch	337 ms

The test method is run three times, once for each iteration (or 'test case')

Now it's your turn!

_Exercises > Exercises02.cs

_Parameterized tests

- _Creating iterations using the [TestCase] annotation
- _Using parameterized data to call the right URI
- _Using parameterized data in assertions

_Answers are in Answers > Answers02.cs

_Examples are in Examples > Examples02.cs

Authentication

- Securing APIs
- Most common authentication schemes:
  - Basic authentication (username / password)
  - Token-based, often using OAuth(2)

Basic authentication

```
[Test]
0 references
public void UseBasicAuthentication()
{
    Given()
        .BasicAuth("username", "password")
        .When()
        .Get("https://my.secure/api")
        .Then()
        .StatusCode(200);
}
```

This will add the Authorization header to the request, with the appropriate value

OAuth(2)

```
[Test]
| 0 references
public void UseOAuth2Authentication()
{
    Given()
        .OAuth2("my_authentication_token")
        .When()
        .Get("https://my.very.secure/api")
        .Then()
        .StatusCode(200);
}
```

The authentication token is typically retrieved prior to running the tests to ensure that a valid token is used

Sharing variables between tests

- Example: tokens, uniquely generated IDs
- First call returns a unique value (e.g. a new user ID)
- Second call needs to use this generated value
- Since there’s no way to predict the value, we need to capture and reuse it

Sharing variables between tests

```
[Test]
0 references
public void CaptureAndReuseUniqueId()
{   The return value can be stored in a variable…
    string userId = (string)Given()
        .When()
        .Post("https://my.user.api/user")
        .Then()
        .Extract()
        .Body("$.id");

    Given()
        .PathParam("userId", userId)
        .When()
        .Get("https://my.user.api/user/[userId]")
        .Then()
        .StatusCode(200);
}
```

Body() takes a JsonPath expression to extract the required value

… and reused at a later point in time

RequestSpecifications

_Reuse shared properties shared by many calls

_Base URI

_Port

_Headers, authentication, cookies

_…

Defining and using a RequestSpecification

```
[Test]
0 references
public void UseRequestSpecification()
{
    Given()
        .Spec(requestSpec)
        .When()
        .Get("/users/1")
        .Then()
        .StatusCode(200);
}
```

… and use it by calling *Spec()* in the *Given()* section of your test

```
private RequestSpecification? requestSpec;

[SetUp]
0 references
public void CreateRequestSpecification()
{
    requestSpec = new RequestSpecBuilder()
        .WithBaseUri("https://jsonplaceholder.typicode.com")
        .WithContentType("application/json")
        .WithOAuth2("my_authentication_token")
        .Build();
}
```

Build your RequestSpecification using a fluent Builder pattern…

Now it's your turn!

_Exercises > Exercises03.cs

_Reuse shared values

- _Apply value reuse as shown in the slides
- _Use basic and OAUth authentication schemes
- _Extract common values to a RequestSpecification

_Answers are in Answers > Answers03.cs

_Examples are in Examples > Examples03.cs

Working with XML responses

- REST APIs can return XML responses, too
- RestAssured.Net can work with this data format, too
- Use XPath instead of JsonPath to select response body element(s)

```xml
<account>
    <id>12345</id>
    <customerId>12212</customerId>
    <type>CHECKING</type>
    <balance>-2300.00</balance>
</account>
```

```
[Test]
0 references
public void GetAccount12345_CheckType_ShouldBeChecking()
{
    Given()
        .When()
        .Get("https://parabank.parasoft.com/parabank/services/bank/accounts/12345")
        .Then()
        .StatusCode(200)
        .Body("//type", NHamcrest.Is.EqualTo("CHECKING"));
}
```

```
<accounts>
    <account>
        <id>12567</id>
        <customerId>12212</customerId>
        <type>CHECKING</type>
        <balance>100.00</balance>
    </account>
    <account>
        <id>12678</id>
        <customerId>12212</customerId>
        <type>SAVINGS</type>
        <balance>-100.00</balance>
    </account>
    <account>
        <id>12789</id>
```


```
[Test]
0 references
public void GetAccountsForCustomer12212_CheckSavingsAccounts_ShouldContain12678()
{
    Given()
        .When()
        .Get("https://parabank.parasoft.com/parabank/services/bank/customers/12212/accounts")
        .Then()
        .StatusCode(200)
        .Body(
            "//account/type[text()='SAVINGS']/parent::account/id",
            NHamcrest.Has.Item(NHamcrest.Is.EqualTo(12678))
        );
}
```

Now it's your turn!

_Exercises > Exercises04.cs

_Work with APIs returning XML response bodies
  _Verify XML response body element values
  _Create the required XPath expressions yourself

_Answers are in Answers > Answers04.cs

_Examples are in Examples > Examples04.cs

(De-)serialization of objects

- RestAssured.Net is able to convert objects directly to XML or JSON (and back)

- Useful when dealing with API payloads
  - Creating request body payloads
  - Processing response body payloads

- No need for additional configuration or libraries

Example: serialization

_Class / DTO / POCO / … representing a blog post

```
internal class Post
{
    [JsonProperty("userId")]
    1 reference | 0/1 passing
    public int UserId { get; set; }

    [JsonProperty("title")]
    3 references | 0/3 passing
    public string Title { get; set; } = string.Empty;

    [JsonProperty("body")]
    1 reference | 0/1 passing
    public string Body { get; set; } = string.Empty;
}
```

RestAssured.Net uses Json.NET for (de-)serialization, which means that all Json.NET attributes can be used

[JsonProperty] defines the name of the property as it appears in JSON

Example: serialization

```
[Test]
0 references
public void SerializePostObjectToJson()
{
    Post post = new Post
    {
        UserId = 1,
        Title = "My new blog post",
        Body = "This is an awesome piece of content"
    },

    Given()
        .Body(post)
        .When()
        .Post("https://jsonplacehold
        .Then()
        .StatusCode(201);
}
```

Create a new Post object …

… then pass it as a request body using *Body()*…

```
{
  "userId": 1,
  "title": "My new blog post",
  "body": "This is an awesome piece of content"
}
```

… and RestAssured.Net will serialize it to JSON automatically

Serializing anonymous objects

```
[Test]
0 references
public void SerializeAnonymousObjectToJson()
{
    var post = new
    {
        userId = 1,
        title = "My new blog post",
        body = "This is an awesome piece of content"
    };

    Given()
        .Body(post)
        .When()
        .Post("https://jsonplaceholde
        .Then()
        .StatusCode(201);
}
```

Create a new anonymous object …

This is useful for one-off request payloads, as it doesn't require you to create a separate class to serialize from

… then pass it as a request body using *Body()*…

```
{
  "userId": 1,
  "title": "My new blog post",
  "body": "This is an awesome piece of content"
}
```

… and RestAssured.Net will serialize it to JSON automatically

Example: deserialization

```
[Test]
0 references
public void DeserializeJsonToPostAfterVerification()
{
    Post post = (Post)Given()
        .When()
        .Get("https://jsonplaceholder.typicode.com/posts/1")
        .Then()
        .StatusCode(200)
        .DeserializeTo(typeof(Post));

    Assert.That(post.Title, Contains.Substring("sunt aut facere"));
}
```

… store the deserialized response payload in an object of that type…

Don't forget to cast it to the type explicitly!

Perform response verifications as usual…

Specify the type to deserialize to using *DeserializeTo()*…

… and then use it in the remainder of your test method as required

Example: deserialization (without initial checks)

```
[Test]
| 0 references
public void DeserializeJsonToPost()
{
    Post post = (Post)Given()
        .When()
        .Get("https://jsonplaceholder.typicode.com/posts/1")
        .DeserializeTo(typeof(Post));

    Assert.That(post.Title, Contains.Substring("sunt aut facere"));
}
```

… store the deserialized response payload in an object of that type…

Don’t forget to cast it to the type explicitly!

Specify the object type to deserialize to using *DeserializeTo()*…

… and then use it in the remainder of your test method as required

Now it's your turn!

_Exercises > Exercises05.cs

_Practice with (de-)serialization

  _You don't need to create or adapt the classes / DTOs yourself, that has been done for you already. By all means go ahead and inspect them, though

_Answers are in Answers > Answers05.cs

_Examples are in Examples > Examples05.cs

One challenge with
'traditional' REST APIs

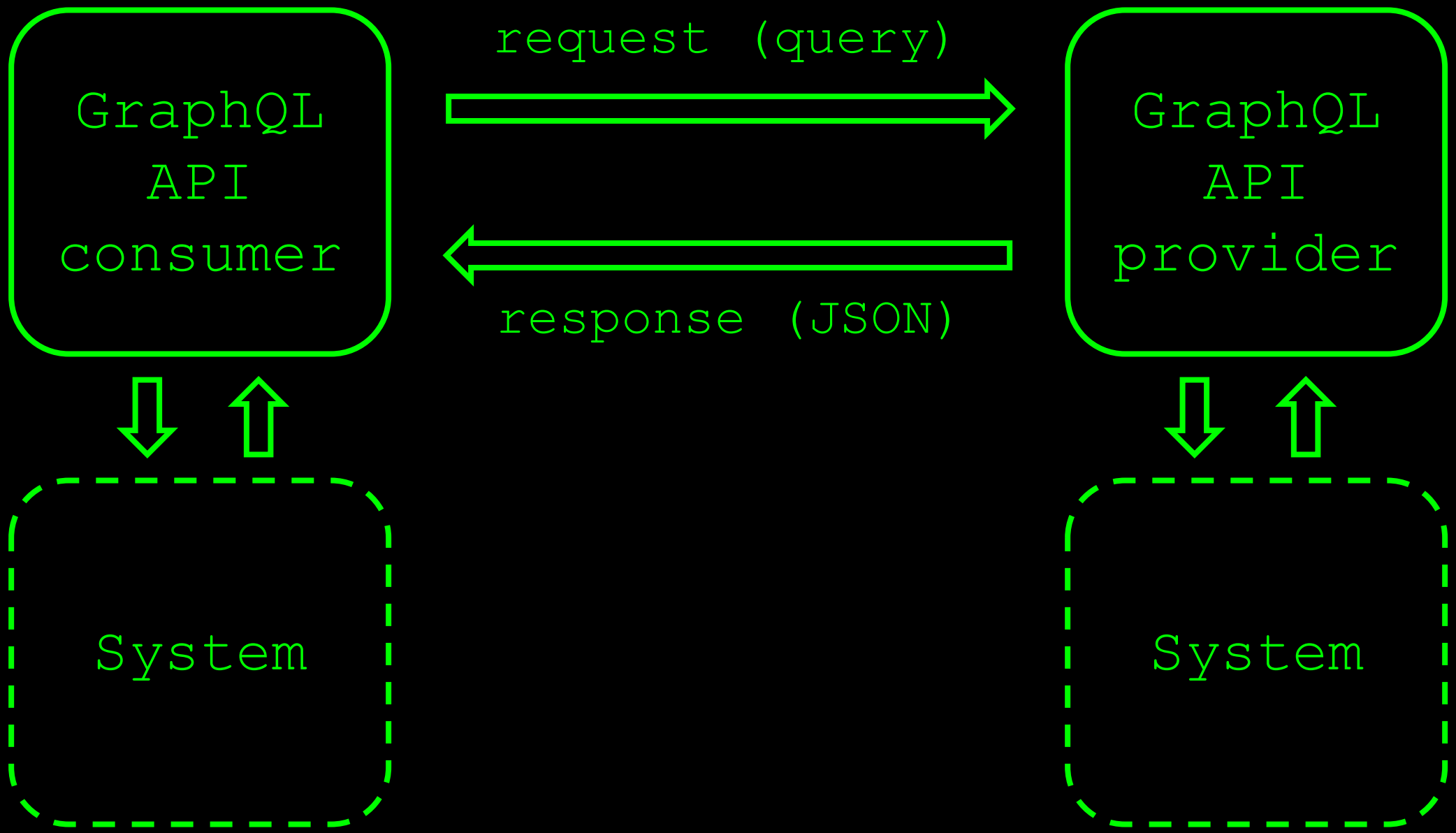
Query language for APIs…

… as well as a runtime to fulfill them

GraphQL

"Ask for what you need,
and get exactly that"

https://graphql.org


GraphQL
API
consumer
request (query)
response (JSON)
GraphQL
API
provider
System
System

Create a valid GraphQL query…

… and send it in the request body (*query*)

Sending a GraphQL query

"Ask for what you need, and get exactly that"

Request payload is still in JSON format

These are ‘regular’ REST responses, with…

… an HTTP status code, …

GraphQL API responses

… response headers…

… and a JSON response body containing the requested data

```
private readonly string hardcodedGraphQLQuery =
    @"query GetCountryData {
        country(code: ""NL"") {
            name
            capital
            currency
            lang
                (
                )
            }
        }
    }";
```

```
[Test]
0 references
public void UseHardCodedValuesInQuery_CheckTheCapital()
{
    GraphQLRequest request = new GraphQLRequestBuilder()
        .WithQuery(this.hardcodedGraphQLQuery)
        .WithOperationName("GetCountryData")
        .Build();

    Given()
        .GraphQL(request)
        .When()
        .Post("https://countries.trevorblades.com/graphql")
        .Then()
        .StatusCode(200)
        .Body("$.data.country.capital", NHamcrest.Is.EqualTo("Amsterdam"));
}
```

```
private readonly string parameterizedGraphQLQuery =
    @"query GetCountryData($country: ID!) {
        country(code: $
            name
            capital
            currency
            languages {
                code
                name
            }
        }
    }";
```

```
[TestCase("NL", "Amsterdam", TestName = "The capital of NL is Amsterdam")]
[TestCase("IT", "Rome", TestName = "The capital of IT is Rome")]
[TestCase("CA", "Ottawa", TestName = "The capital of CA is Ottawa")]
0 references
public void UseParametersInQuery_CheckTheCapital
    (string countryCode, string expectedCapital)
{
    Dictionary<string, object> variables = new Dictionary<string, object>
    {
        { "country", countryCode },
    };

    GraphQLRequest request = new GraphQLRequestBuilder()
        .WithQuery(this.parameterizedGraphQLQuery)
        .WithOperationName("GetCountryData")
        .WithVariables(variables)
        .Build();

    Given()
        .GraphQL(request)
        .When()
        .Post("https://countries.trevorblades.com/graphql")
        .Then()
        .StatusCode(200)
        .Body("$.data.country.capital", NHamcrest.Is.EqualTo(expectedCapital));
}
```

Now it's your turn!

_Exercises > Exercises06.cs

_Work with GraphQL APIs

- _Invoke an endpoint with a non-parameterized query
- _Invoke an endpoint with a parameterized query

_Answers are in Answers > Answers06.cs

_Examples are in Examples > Examples06.cs

Adding abstraction layers

- Once your test suite grows, you'll find yourself reusing certain requests over and over

- Writing them in full every time you need them means code duplication and decreased maintainability

- Solution: add an abstraction layer on top of (parts of) the RestAssured.Net code

Step 1: Create a ClientBase

This class should not be instantiated in tests

```
public abstract class ClientBase
{
    private string baseUri;

    1 reference
    protected ClientBase(string baseUri)
    {
        this.baseUri = baseUri;
    }

    2 references
    public RequestSpecification GetRequestSpec()
    {
        return new RequestSpecBuilder()
            .WithBaseUri(baseUri)
            .WithContentType("application/json")
            .Build();
    }
}
```

Parameters that might change for different environments go here

If these are always the same they can also be hardcoded

The RequestSpecification contains properties shared among all requests

Step 2: Create a Client class

```
public class PostClient : ClientBase
{
    private static readonly string BaseUri = "https://jsonplaceholder.typicode.com";

    1 reference
    public PostClient() : base(BaseUri)
    {
    }

    1 reference | 0/1 passing
    public VerifiableResponse GetPost(int postId)
    {
        return Given()
            .Spec(base.GetRequestSpec())
            .When()
            .Get($"/posts/{postId}");
    }

    1 reference | 0/1 passing
    public HttpResponseMessage CreatePost(Post post)
    {
        return Given()
            .Spec(base.GetRequestSpec())
            .Body(post)
            .When()
            .Post("/posts")
            .Then()
            .StatusCode(201)
            .Extract().Response();
    }
```

Inherit shared properties from the base class

Initialize the class and the base class

Create a method that performs an action that is repeated across tests

Here, it's retrieving a post based on the post ID

It returns a VerifiableResponse (this is a RestAssured.Net class)

Alternatively, you can create methods that return the 'raw' HttpResponseMessage (this is a System.Net.Http class)

This way, you can also perform basic checks before returning the response object, if you want

Step 3: Use the Client class in your tests

```
public class Examples05 : TestBase
{
    private readonly PostClient postClient = new PostClient();

    [Test]
    | 0 references
    public void ApplyClientTestModel_ReturnVerifiableResponse_CheckStatusCodeAndResponseHeader()
    {
        postClient.GetPost(1)
            .Then()
            .StatusCode(200)
            .ContentType(NHamcrest.Contains.String("application/json"));
    }

    [Test]
    | 0 references
    public void ApplyClientTestModel_ReturnHttpResponseMessage_CheckStatusCodeAndResponseHeader()
    {
        Post post = new Post
        {
            UserId = 1,
            Title = "My new blog post",
            Body = "This is the body of my brand new blog post."
        };

        var response = postClient.CreatePost(post);
        Assert.That(response.StatusCode, Is.EqualTo(HttpStatusCode.Created));

        response.Content.Headers.TryGetValues("Content-Type", out IEnumerable<string>? values);
        Assert.That(values!.First(), Does.Contain("application/json"));
```

Create a new client instance

Call client methods to perform repeatable actions

Returning a VerifiableResponse enables using RestAssured.Net verifications and a fluent syntax

Working with an HttpResponseMessage requires a bit more work to perform verifications on the response

You do have access to the raw response, though, which could be beneficial in certain cases

Now it's your turn!

_Exercises > Exercises07.cs

_Practice with adding abstraction layers to your tests

- _The ClientBase has been created for you
- _First, define the appropriate methods in the AccountClient
- _Then, complete the tests using the AccountClient methods

_Answers are in Answers > Answers07.cs

_Examples are in Examples > Examples07.cs

?

Contact

_Email: bas@ontestautomation.com

_Website: https://www.ontestautomation.com/training

_LinkedIn: https://www.linkedin.com/in/basdijkstra