API testing with REST Assured

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 (REST) APIs

_REST Assured

_Hands-on exercises

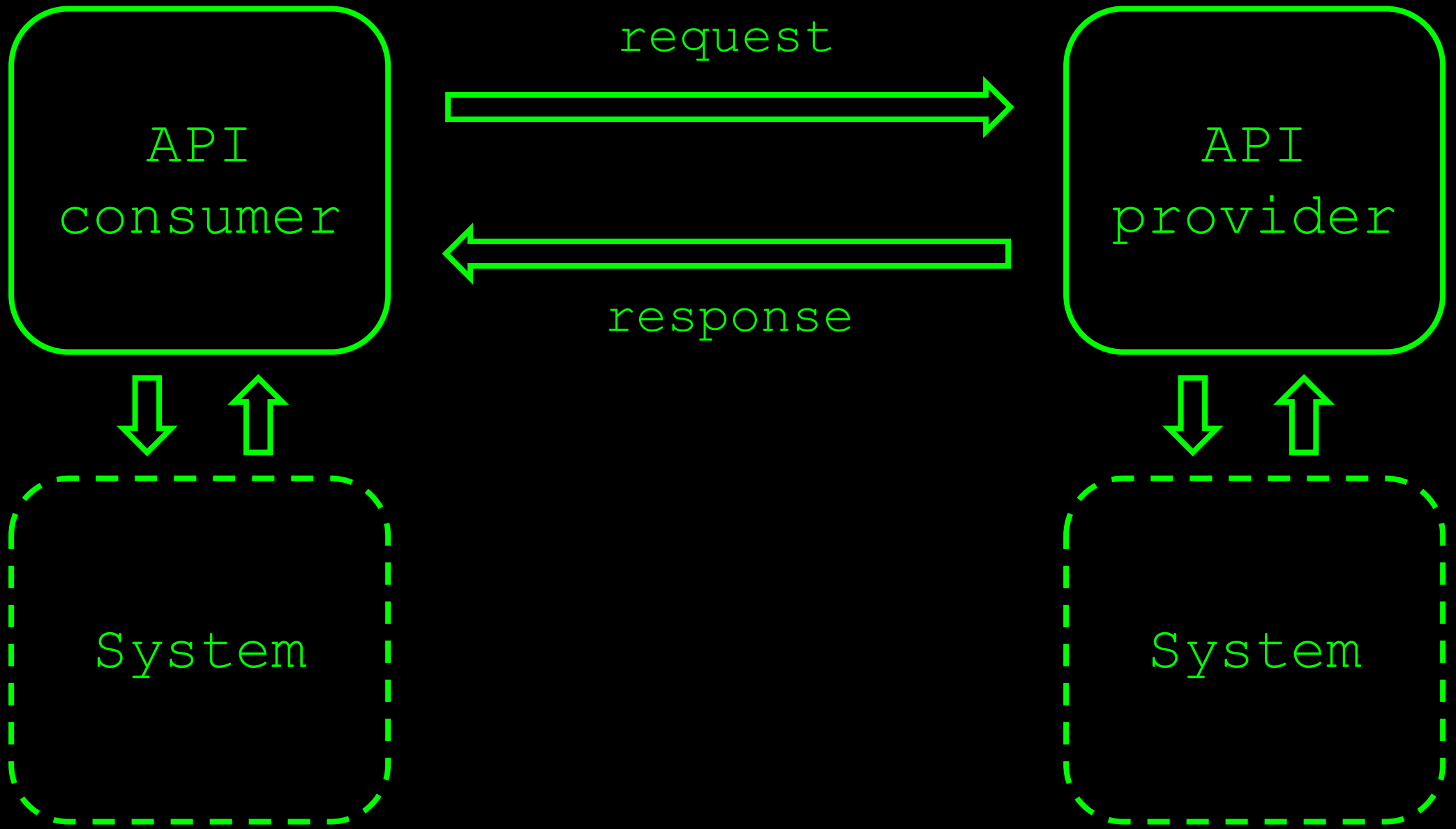
Preparation

_Install a recent JDK (17 or newer)

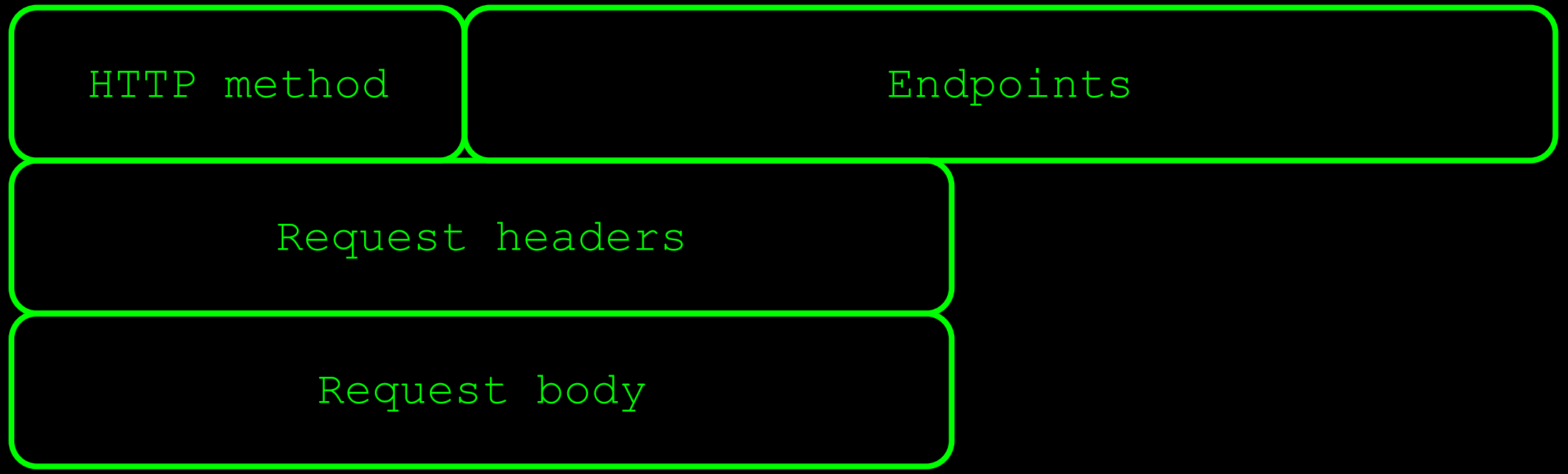
_Install IntelliJ (or any other IDE)

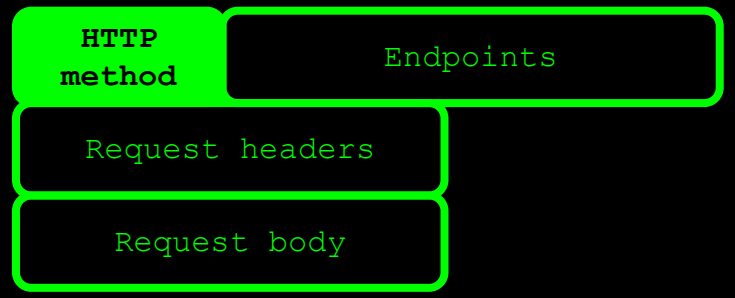
_Import Maven project into your IDE
  _https://github.com/basdijkstra/rest-assured-workshop

APIs are commonly used to exchange data between two parties


request
API
consumer
API
provider
response
System
System

A REST API request





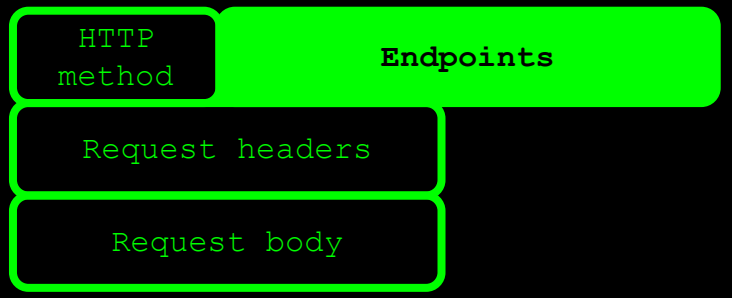
HTTP methods

_GET, POST, PUT, PATCH, DELETE, OPTIONS, …

_CRUD operations on data

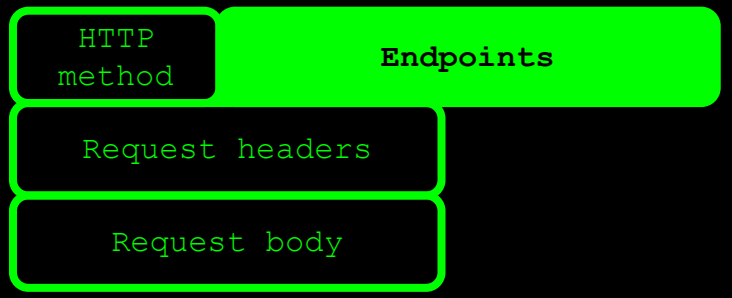
POST	Create
GET	Read
PUT / PATCH	Update
DELETE	Delete
…	…

_Conventions, not standards!



Endpoints

- Uniform Resource Identifier
- Identifies the resource to operate on
- Can contain parameters
  - Query parameters
  - Path parameters

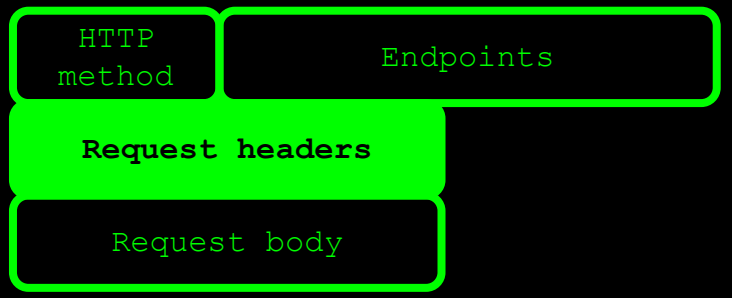


Parameters in endpoints

- 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!

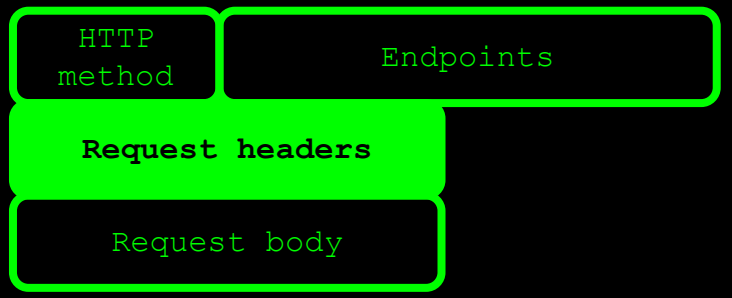


Request headers

- 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?)
  - …

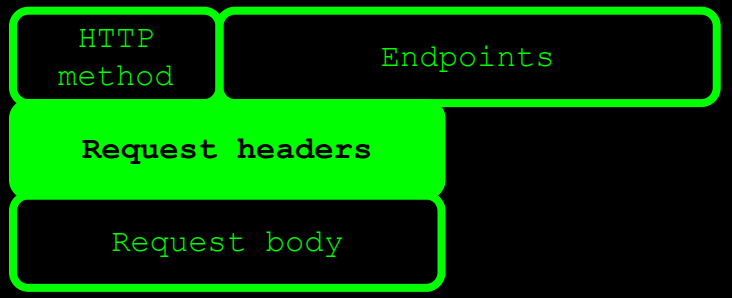
- Can contain session and authorization data
  - Cookies
  - Authorization tokens
  - …



Authorization: Basic

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

Authorization: Basic YWxhZGRpbjpvcGVuc2VzYW1l



Authorization: Bearer

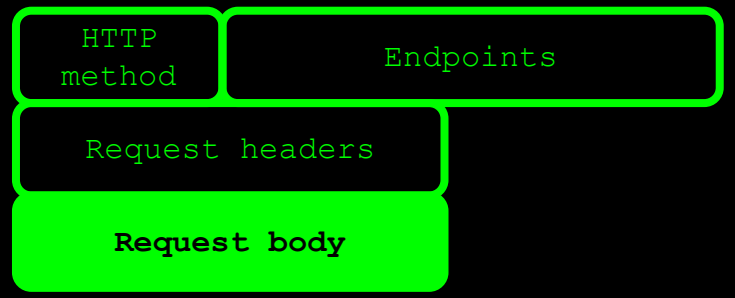
_Token with expiry date / time is obtained first

_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

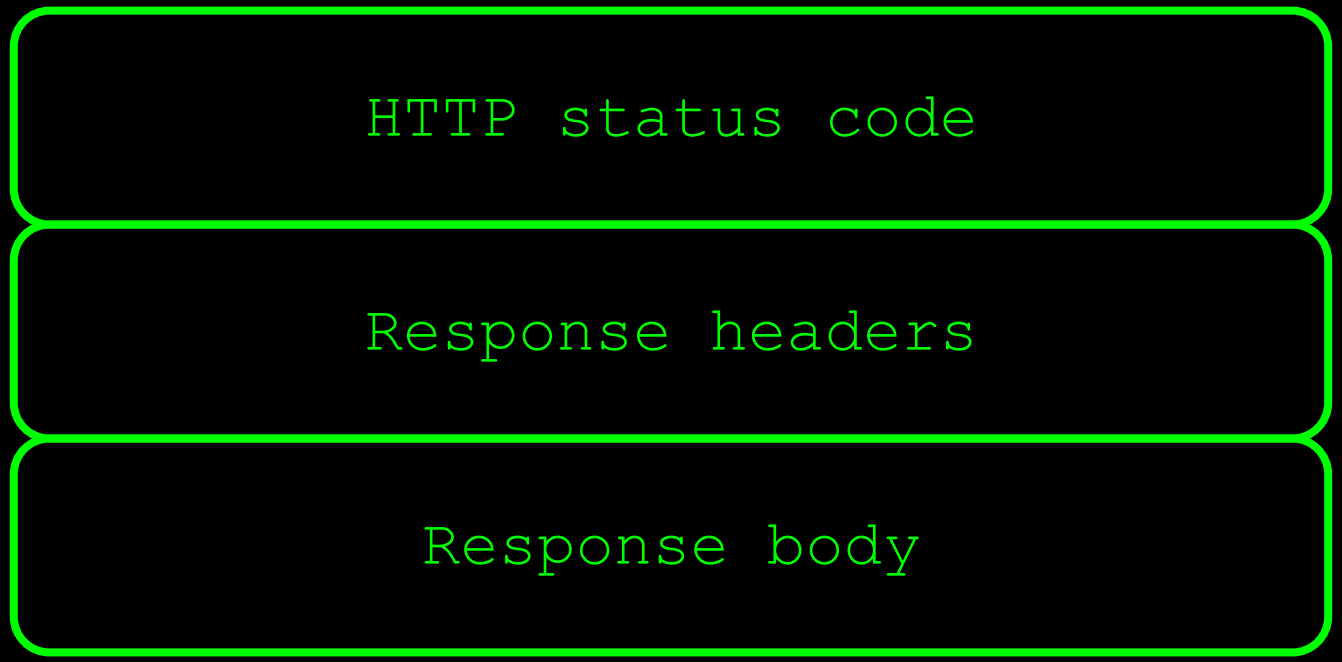
- Data to be sent to the provider

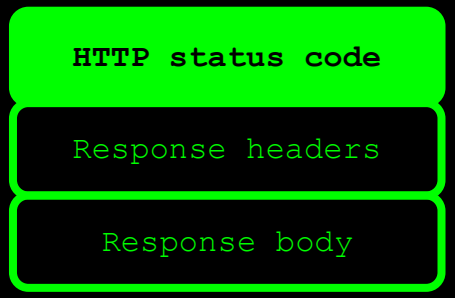
- REST does not prescribe a specific data format

- Most common:
  - JSON
  - XML
  - Plain text

- Other data formats can be sent using REST, too

A REST API response



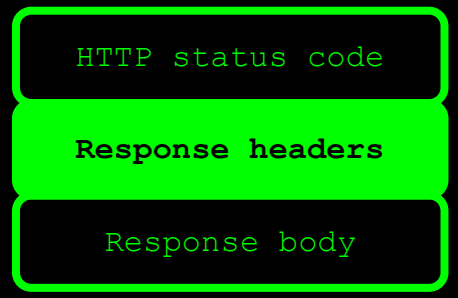


HTTP status code

_Indicates result of request processing by provider

_Five different categories

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

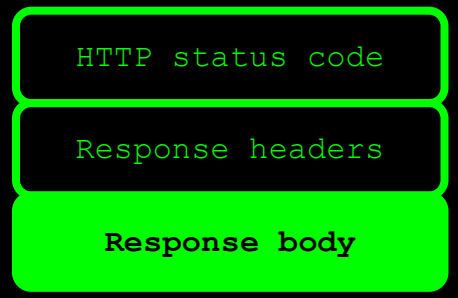


Response headers

- 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



Response body

_Data returned by the provider

_REST does not prescribe a specific data format

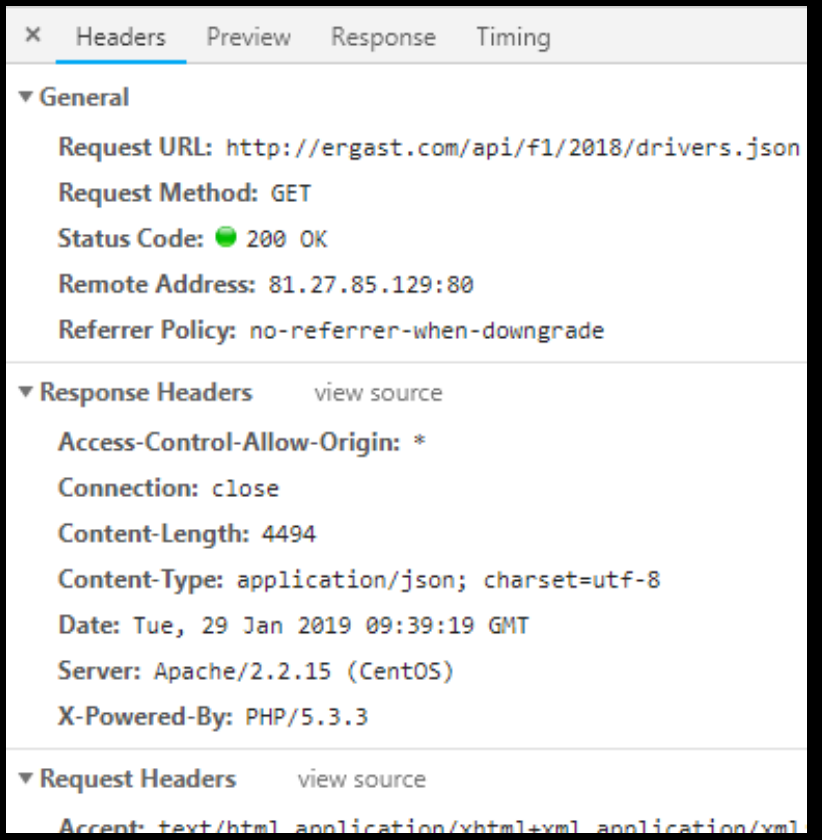
_Most common:
  _JSON
  _XML
  _Plain text

_Other data formats can be sent using REST, too

An example

_GET http://ergast.com/api/f1/2018/drivers.json

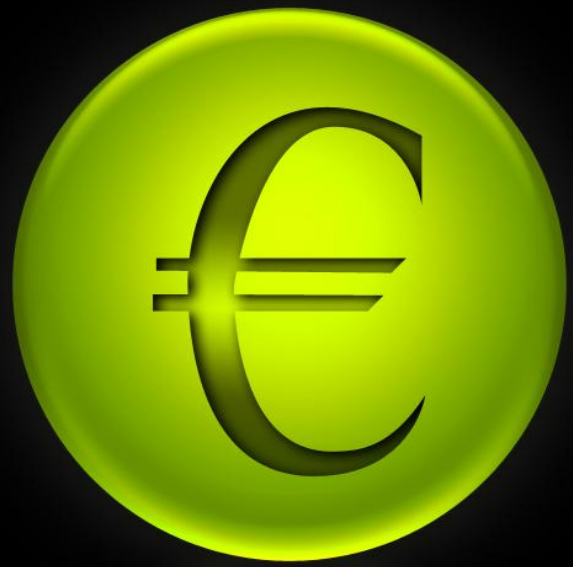

```
{
  - MRData: {
        xmlns: "http://ergast.com/mrd/1.4",
        series: "f1",
        url: "http://ergast.com/api/f1/2018/drivers.json",
        limit: "30",
        offset: "0",
        total: "20",
      - DriverTable: {
            season: "2018",
          - Drivers: [
              - {
                    driverId: "alonso",
                    permanentNumber: "14",
                    code: "ALO",
                    url: "http://en.wikipedia.org/wiki/Fernando_Alonso",
                    givenName: "Fernando",
                    familyName: "Alonso",
                    dateOfBirth: "1981-07-29",
                    nationality: "Spanish"
                },
              - {
                    driverId: "bottas",
                    permanentNumber: "77",
                    code: "BOT"
```

Where are APIs used?

Mobile

Internet of Things



API economy

Where are APIs used?

Web applications

Microservices architectures

Why I ♥ testing at the API level

_Writing and running tests is faster compared to UI-driven tests

_Tests are closer to end user experience than unit tests

_Business logic is often exposed at the API level

Tools to test HTTP (REST) APIs

- Free / open source
  - Postman
  - SoapUI
  - Bruno
  - Code libraries like REST Assured, RestAssured.Net, RestSharp, requests, …
  - …
- Commercial
  - Parasoft SOAtest
  - ReadyAPI
  - …
- Build your own (using HTTP libraries for your language of choice)

REST Assured

- Java DSL for writing tests for HTTP-based APIs
- Removes a lot of boilerplate code
- Runs on top of common unit testing frameworks
  - JUnit, TestNG
- Developed and maintained by Johan Haleby

Configuring REST Assured

_Download from http://rest-assured.io

_Add as a dependency to your project

- _Maven
- _Gradle

```
<dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>rest-assured</artifactId>
    <version>5.5.6</version>
    <scope>test</scope>
</dependency>
```

REST Assured documentation

_Official website: https://rest-assured.io

_Links to usage guide, JavaDoc, ...

A sample test

REST Assured uses JUnit (this could also be TestNG)

```
@Test
public void getUserData_verifyName_shouldBeLeanneGraham() {

    given().  Make an HTTP GET call to retrieve data from the provider
    when().
        get( s: "http://jsonplaceholder.typicode.com/users/1"). // Do a GET call to the specified resource
    then().
        assertThat()                                          // Assert that the value of the element 'name'
        body( s: "name", equalTo( operand: "Leanne Graham"));  // in the response body equals 'Leanne Graham'
}
```

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

REST Assured features

- Support for all HTTP methods (GET, POST, PUT, …)
- Gherkin-like (Given/When/Then) syntax
- Use of Hamcrest matchers for checks (*equalTo*)
- Use of GPath for selecting elements from JSON or XML responses

```
@Test
public void getUserData_verifyName_shouldBeLeanneGraham() {

    given().
    when().
        get( s: "http://jsonplaceholder.typicode.com/users/1").  // Do a GET call to the specified resource
    then().
        assertThat().                                           // Assert that the value of the element 'name'
        body( s: "name", equalTo( operand: "Leanne Graham"));    // in the response body equals 'Leanne Graham'
}
```

About Hamcrest matchers

_Express expectations in natural language

_Examples:

equalTo(X)	Does the object equal X?
hasItem(“Rome”)	Does the collection contain an item “Rome”?
hasSize(3)	Does the size of the collection equal 3?
not(equalTo(X))	Inverts matcher equalTo()

_ https://hamcrest.org/JavaHamcrest/javadoc/3.0/org/hamcrest/Matchers.html

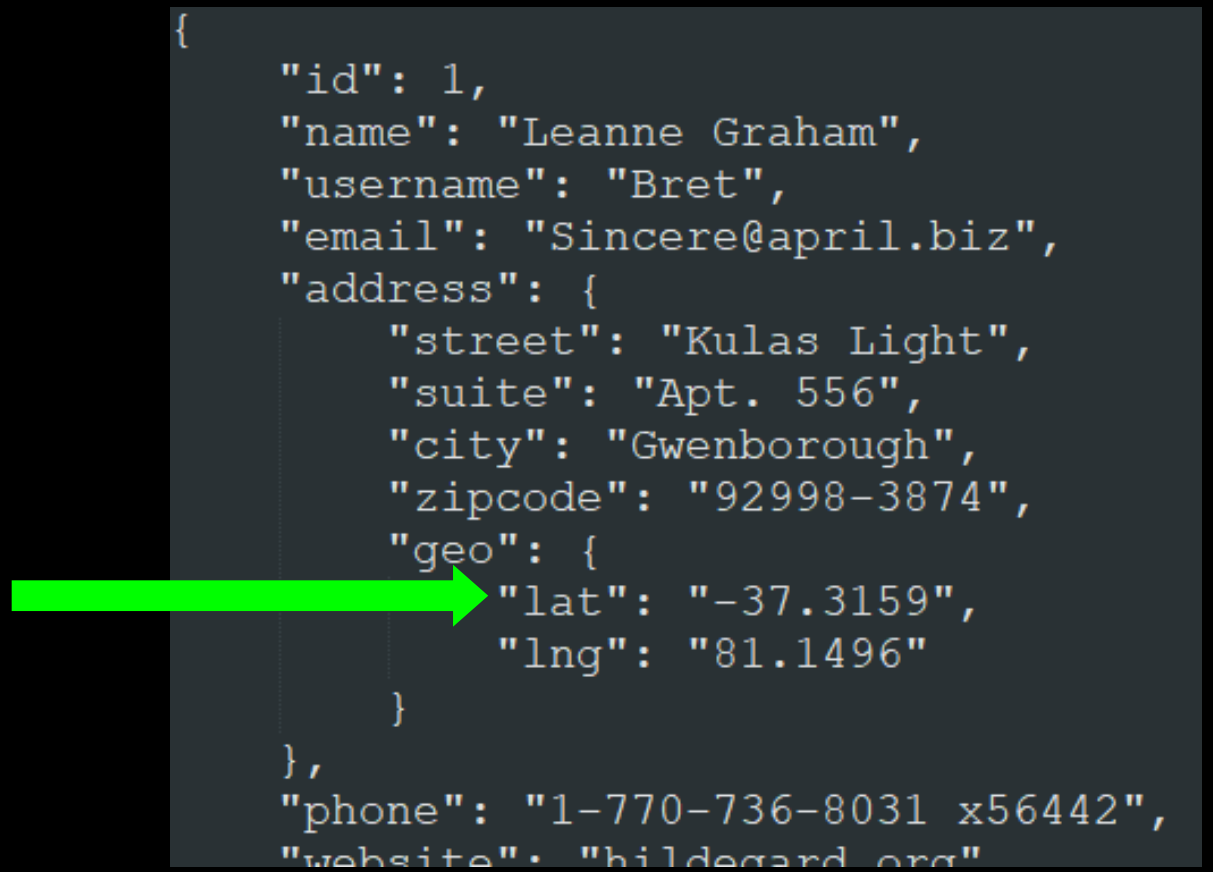
About GPath

- Syntax for selecting elements from JSON or XML documents

- Similar to JsonPath and XPath

- Documentation and examples:
  - http://groovy-lang.org/processing-xml.html#_gpath

GPath example




```
body("address.geo.lat", equalTo("-37.3159"));
```

Validating response properties

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

```
@Test
public void getUserData_verifyStatusCodeAndContentType() {

    given().
    when().
        get( s: "http://jsonplaceholder.typicode.com/users/1").
    then().
        assertThat().
        statusCode(200).
    and().
        contentType(ContentType.JSON);
}
```

Logging request data

```
@Test
public void logAllRequestData() {

    given().
        log().all().
    when().
        get( s: "http://jsonplaceholder.typicode.com/users/1").
    then().
        assertThat().
        body( s: "name", equalTo( operand: "Leanne Graham"));
}
```

log().all() after *given()* logs all request data to the console

You can also use *log().body()*, *log().headers()* as well as other options

Logging request data

```
@Test
public void logAllRequestData() {

    given().
        log().all().
    when().
        get( s: "http
    then().
        assertThat()
        body( s: "nam
}
```

```
Request method:  GET
Request URI:     http://jsonplaceholder.typicode.com/users/1
Proxy:           <none>
Request params:  <none>
Query params:    <none>
Form params:     <none>
Path params:     <none>
Headers:         Accept=*/*
Cookies:         <none>
Multiparts:      <none>
Body:            <none>
```

Logging response data

```
@Test
public void logAllResponseData() {

    given().
    when().
        get( s: "http://jsonplaceholder.typicode.com/users/1").
    then().
        log().all().
    and().
        assertThat().
        body( s: "name", equalTo( operand: "Leanne Graham"));
}
```

log().all() after *then()* logs all response data to the console

You can also use *log().body()*, *log().headers()* as well as other options

Logging response data

```java
@Test
public void logAllResponseData() {

    given().
    when().
        get( s: "http://jsonplaceholder.
    then().
        log().all().
    and().
        assertThat().
        body( s: "name", equalTo( operand:
}
```

```
HTTP/1.1 200 OK
Date: Wed, 28 Oct 2020 16:37:56 GMT
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
Set-Cookie: __cfduid=ddc99ce478f9e81d2e127ecfaf863768516039030
X-Powered-By: Express
X-Ratelimit-Limit: 1000
X-Ratelimit-Remaining: 993
X-Ratelimit-Reset: 1598842094
Vary: Origin, Accept-Encoding
Access-Control-Allow-Credentials: true
Cache-Control: max-age=43200
Pragma: no-cache
Expires: -1
X-Content-Type-Options: nosniff
Etag: W/"1fd-+2Y3G3w049iSZtw5t1mzSnunngE"
Via: 1.1 vegur
CF-Cache-Status: HIT
Age: 15396
cf-request-id: 0611abd0ce0000e668cd936000000001
Report-To: {"endpoints":[{"url":"https:\/\/a.nel.cloudflare.com\
NEL: {"report_to":"cf-nel","max_age":604800}
Server: cloudflare
CF-RAY: 5e9615947bb2e668-LHR
Content-Encoding: gzip

{
    "id": 1
```

Our API under test

_(Simulation of) an online banking API

_Customer data (GET, POST)

_Account data (POST, GET)

Demo

_How to use the test suite

_Executing your tests

_Reviewing test results

_Logging request and response data

Now it's your turn!

_src > test > java > exercises > RestAssuredExercises1Test.java

_Basic checks
  _Validating individual element values
  _Validating collections and items therein
  _Validating other response properties

_Stubs are predefined
  _Don't worry about the references to http://localhost
  _You only need to write the tests using REST Assured

_Answers are in answers > RestAssuredAnswers1Test.java

_Examples are in examples > RestAssuredExamples.java

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
public void useQueryParameter() {

                                Define a query parameter and its value
    given().
        queryParam( s: "text",  ...objects: "testcase").
    when().
        get( s: "http://md5.jsontest.com").
    then().
        assertThat().
        body( s: "md5", equalTo( operand: "7489a25fc99976f06fecb807991c61cf"));
}
```

Using path parameters

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

```
@Test
public void usePathParameter() {

    given().
        pathParam( s: "userId", o: 1).
    when().
        get( s: "http://jsonplaceholder.typicode.com/users/{userId}").
    then().
        assertThat().
        body( s: "name", equalTo( operand: "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

‘Glue’ between different systems or different components in a system

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

Most popular test runners support this

Parameterized testing

More efficient to do parameterized testing at the API level…

… as compared to doing this at the UI level

'Feeding' test data to your test

```
@ParameterizedTest
@CsvSource({
        "1, Leanne Graham",
        "2, Ervin Howell",
        "3, Clementine Bauch"
})
public void checkNameForUser
    (int userId, String expectedUserName) {

    given().
        pathParam( s: "userId", userId).
    when().
        get( s: "http://jsonplaceholder.typicode.com/users/{userId}").
    then().
        assertThat().
        body( s: "name",equalTo(expectedUserName));
}
```

Define test data in the @CsvSource annotation (one record for every iteration, parameters separated by commas)

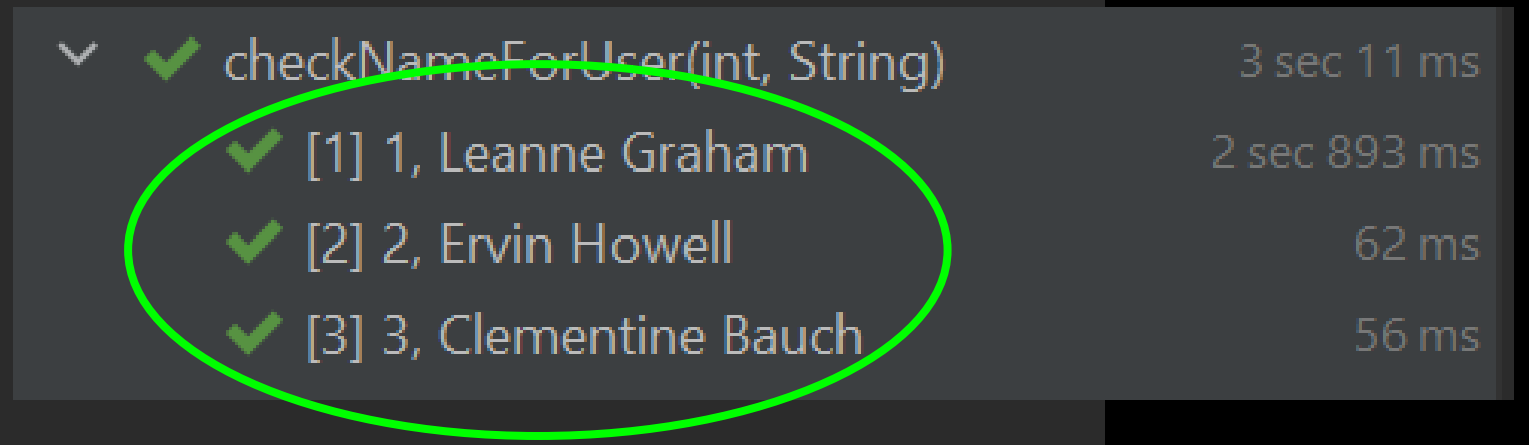
Use parameters to pass the test data values into the method

Use parameters in the test method where required

Running the data driven test

```
@ParameterizedTest
@CsvSource({
        "1, Leanne Graham",
        "2, Ervin Howell",
        "3, Clementine Bauch"
})
public void checkNameForUser
    (int userId, String expectedUserName) {

    given().
        pathParam( s: "userId", userId).
    when().
        get( s: "http://jsonplaceholder.typicode.com/users/{userId}").
    then().
        assertThat().
        body( s: "name",equalTo(expectedUserName));
}
```



The test method is run three times, once for each array ('test case') in the test data set

Now it's your turn!

_src > test > java > exercises > RestAssuredExercises2Test.java

_Data driven tests
  _Creating a test data object using @CsvSource
  _Using test data to call the right URI
  _Using test data in assertions

_Answers are in answers > RestAssuredAnswers2.java

_Examples are in examples > RestAssuredExamples.java

Authentication

_Securing the data exposed by APIs

_Common authentication schemes:

  _Basic authentication (username / password)

  _OAuth(2) (token-based)

Basic authentication

```
@Test
public void useBasicAuthentication() {

    given().
        auth().
        preemptive().
        basic( s: "username",  s1: "password").
    when().
        get( s: "https://my.secure/api").
    then().
        assertThat().
        statusCode(200);
}
```

Adding *preemptive()* makes REST Assured send the credentials directly, saving us from dealing with the provider challenging mechanism

OAuth(2)

```
@Test
public void useOAuthAuthentication() {

    given().
        auth().
        oauth2( s: "myAuthenticationToken").
    when().
        get( s: "https://my.very.secure/api").
    then().
        assertThat().
        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: uniquely generated IDs
- First call returns a unique ID (e.g. a new user ID)
- Second call needs to use this generated ID
- Since there's no way to predict the ID, we need to capture and reuse it

Sharing variables between tests

```
@Test
public void captureAndReuseUserId() {

    String userId =

        given().
        when().
            post( s: "http://my.user.api/user").
        then().
            extract().
            path( s: "id");

    given().
        pathParam( s: "userId", userId).
    when().
        get( s: "http://my.user.api/user/{userId}").
    then().
        assertThat().
        statusCode(200);
}
```

The return value can be stored in a variable…

path() takes a GPath expression to extract the required value

… and reused at a later point in time

RequestSpecifications

Reuse shared properties shared by different calls

_Base URI and base path

_Port number

_Authentication and other headers

_…

Defining and using RequestSpecifications

```
private RequestSpecification requestSpec;

@BeforeEach
public void createRequestSpec() {

    requestSpec =
        new RequestSpecBuilder().
            setBaseUri("http://api.zippopotam.us").
            setPort(9876).
            build();
}
```

Build your RequestSpecification using the Builder pattern…

```
@Test
public void useRequestSpec() {

    given().
        spec(requestSpec).
    when().
        get( s: "/us/90210.json").
    then().
        assertThat().
        statusCode(200);
}
```

… and use it by calling *spec()* in the *given()* section of your test

Sharing checks between tests

- Example: checking status code and MIME type for all responses

- Another maintenance burden if specified individually for each test

- What if we could specify this once and reuse throughout our tests?

Using a ResponseSpecification

```
@BeforeEach
public void createResponseSpec() {

    responseSpec =
        new ResponseSpecBuilder().
            expectStatusCode(200).
            expectContentType(ContentType.JSON).
            build();
}
```

Build your ResponseSpecification using the Builder pattern…

```
@Test
public void useResponseSpec() {

    given().
    when().
        get( s: "http://jsonplaceholder.typicode.com/users/1").
    then().
        spec(responseSpec).
    and().
        body( s: "name", equalTo( operand: "Leanne Graham"));
}
```

… and use it by calling *spec()* in the *then()* section of your test

Now it's your turn!

_src > test > java > exercises > RestAssuredExercises3Test.java

_Capture authentication token and reuse it in another API call

_Use basic and OAuth authentication schemes

_Answers are in answers > RestAssuredAnswers3Test.java

_Examples are in examples > RestAssuredExamples.java

XML support

- So far, we've only used REST Assured on APIs that return JSON
- It works just as well with XML-based APIs
- Identification of response elements uses GPath
- Let's also look at some other interesting things you can do with GPath

XmlPath - examples

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<cars>
    <car make="Alfa Romeo" model="Giulia">
        <country>Italy</country>
        <modelYear>2016</modelYear>
    </car>
    <car make="Aston Martin" model="DB11">
        <country>UK</country>
        <modelYear>1949</modelYear>
    </car>
    <car make="Toyota" model="Auris">
        <country>Japan</country>
        <modelYear>2012</modelYear>
    </car>
</cars>
```

Check country for the first car in the list

```java
@Test
public void checkCountryForFirstCar() {

    given().
    when().
        get( s: "http://path.to/cars/xml").
    then().
        assertThat().
        body( s: "cars.car[0].country", equalTo( operand: "Italy"));
}
```

XmlPath – examples

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<cars>
    <car make="Alfa Romeo" model="Giulia">
        <country>Italy</country>
        <modelYear>2016</modelYear>
    </car>
    <car make="Aston Martin" model="DB11">
        <country>UK</country>
        <modelYear>1949</modelYear>
    </car>
    <car make="Toyota" model="Auris">
        <country>Japan</country>
        <modelYear>2012</modelYear>
    </car>
</cars>
```

Check model year for the last car in the list

```java
@Test
public void checkYearForLastCar() {

    given().
    when().
        get( s: "http://path.to/cars/xml").
    then().
        assertThat().
        body( s: "cars.car[-1].modelYear", equalTo( operand: "2012"));
}
```

XmlPath - examples

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<cars>
    <car make="Alfa Romeo" model="Giulia">
        <country>Italy</country>
        <modelYear>2016</modelYear>
    </car>
    <car make="Aston Martin" model="DB11">
        <country>UK</country>
        <modelYear>1949</modelYear>
    </car>
    <car make="Toyota" model="Auris">
        <country>Japan</country>
        <modelYear>2012</modelYear>
    </car>
</cars>
```

Check model for the second car in the list

(use an @ to refer to an XML attribute)

```
@Test
public void checkModelForSecondCar() {

    given().
    when().
        get( s: "http://path.to/cars/xml").
    then().
        assertThat().
        body( s: "cars.car[1].@model", equalTo( operand: "DB11"));
}
```

XmlPath – examples

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<cars>
    <car make="Alfa Romeo" model="Giulia">
        <country>Italy</country>
        <modelYear>2016</modelYear>
    </car>
    <car make="Aston Martin" model="DB11">
        <country>UK</country>
        <modelYear>1949</modelYear>
    </car>
    <car make="Toyota" model="Auris">
        <country>Japan</country>
        <modelYear>2012</modelYear>
    </car>
</cars>
```

Check there's one car from Japan in the list

findAll is a filter operation in GPath

```
@Test
public void checkTheListContainsOneJapaneseCar() {

    given().
    when().
        get( s: "http://path.to/cars/xml").
    then().
        assertThat().
        body( s: "cars.car.findAll{it.country=='Japan'}", hasSize(1));
}
```

XmlPath – examples

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<cars>
    <car make="Alfa Romeo" model="Giulia">
        <country>Italy</country>
        <modelYear>2016</modelYear>
    </car>
    <car make="Aston Martin" model="DB11">
        <country>UK</country>
        <modelYear>1949</modelYear>
    </car>
    <car make="Toyota" model="Auris">
        <country>Japan</country>
        <modelYear>2012</modelYear>
    </car>
</cars>
```

Check that two cars have a make starting with 'A'

grep takes a regular expression to search in a list of values

```java
@Test
public void checkTheListContainsTwoCarsWhoseMakeStartsWithAnA() {

    given().
    when().
        get( s: "http://path.to/cars/xml").
    then().
        assertThat().
        body( s: "cars.car.@make.grep(~/A.*/)", hasSize(2));
}
```

Now it's your turn!

_src > test > java > exercises > RestAssuredExercises4Test.java

_Communicating with an API returning an XML document

_Use GPath to select the right nodes

_Use filters, in, grep() where needed

_Answers are in answers > RestAssuredAnswers4Test.java

_Examples are in examples > RestAssuredExamplesXml.java

(De-)serialization of objects

- REST Assured is able to convert object instances directly to XML or JSON (and back)

- Useful when dealing with test data objects
  - Creating request body payloads
  - Processing response body payloads

- Requires additional libraries on the classpath
  - Jackson or Gson for JSON
  - JAXB for XML

```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>${jackson.databind.version}</version>
    <scope>test</scope>
</dependency>
```

Example: serialization

_Object representing an address

```
public class Address {

    private String street;
    private int houseNumber;
    private int zipCode;
    private String city;

    public Address(String street, int houseNumber, int zipCode, String city) {

        this.street = street;
        this.houseNumber = houseNumber;
        this.zipCode = zipCode;
        this.city = city;
    }
```

Example: serialization

```
@Test
public void serializeAddressToJson() {

    Address myAddress = new Address( street: "My street",  houseNumber: 1,  zipCode: 1234,  city: "Amsterdam");

    given().
        body(myAddress).
    when().
        post( s: "http://localhost:9876/address").
    then().
        assertThat().
        statusCode(200);
}
```

Pass the object as a request body using *body()*…

… and REST Assured will serialize it to JSON using Jackson
(which means you can customize the field names if required)

```
Body:
{"street":"My street","houseNumber":1,"zipCode":1234,"city":"Amsterdam"}
```

Example: deserialization

```
@Test
public void deserializeJsonToAddress() {

    Address myAddress =

        given().
        when().
            get( s: "http://localhost:9876/address").
        then().
            statusCode(200).
        and().
            extract().
            body().
            as(Address.class);

    assertEquals( expected: "Amsterdam", myAddress.getCity());
}
```

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

Perform response verifications as usual…

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

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

Example: deserialization (without initial checks)

```
@Test
public void deserializeJsonToAddressWithoutInitialChecks() {

    Address myAddress =

        given().
        when().
            get( s: "http://localhost:9876/address").
            as(Address.class);

    assertEquals( expected: "Amsterdam", myAddress.getCity());
}
```

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

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

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

Now it's your turn!

_src > test > java > exercises > RestAssuredExercises5Test.java

_Practice (de-)serialization for yourself

_You don't need to create or adapt the data objects

_Answers are in answers > RestAssuredAnswers5Test.java

_Examples are in examples > RestAssuredExamples.java

One challenge with ‘traditional’ REST APIs

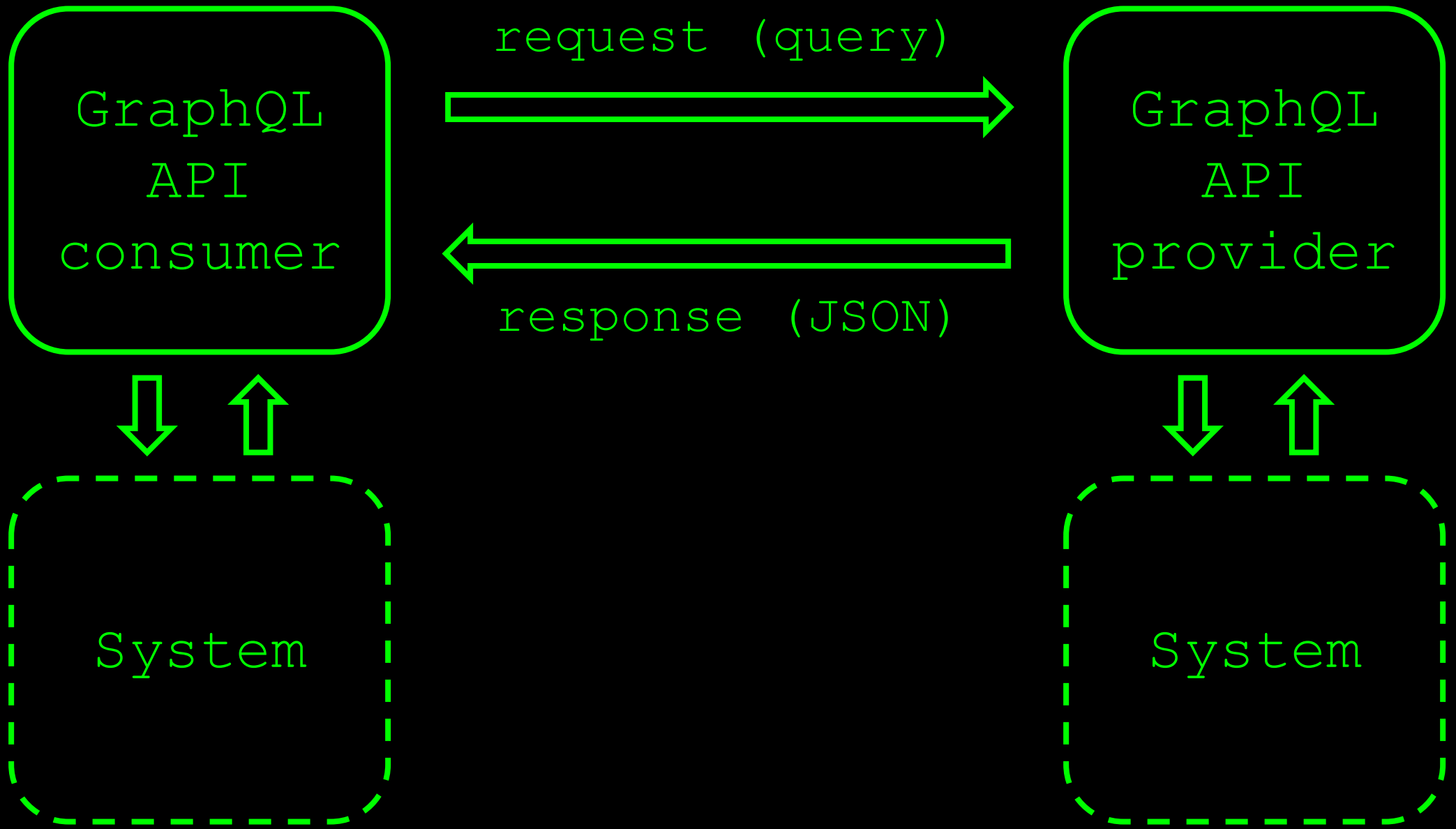
Query language for APIs…

… 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

A Java

HashMap<String, Object>

structure is a good fit for this situation

These are ‘regular’ REST responses, with…

… an HTTP status code, …

GraphQL API responses

… response headers…

… and a JSON response body
containing the requested data

Sending a basic GraphQL query

```
String queryString = """
        {
          getCityByName(name: "Amsterdam") {
            weather {
              summary {
                title
              }
            }
          }
        }
        """;
```

The query can be a simple (multiline) String

Initialize the GraphQL query object in a HashMap…

```
@Test
public void useHardCodedValuesInQuery_checkTheWeather() {

    HashMap<String, Object> graphQlQuery = new HashMap<>();
    graphQlQuery.put("query", queryString);

    given().
        contentType(ContentType.JSON).
        body(graphQlQuery).
    when().
        post( s: "https://graphql-weather-api.herokuapp.com/").
    then().
        assertThat().
        statusCode(200).
    and().
        body( s: "data.getCityByName.weather.summary.title", equalTo( operand: "Clear"));
}
```

… and send it as the request body

The response body is regular JSON, so we know how to handle that already

Parameterizing GraphQL queries

```
String queryString = """
        query getWeatherFor($name: String!)
        {
          getCityByName(name: $name) {
            weather {
              summary {
                title
              }
            }
          }
        }
        """;
```

GraphQL queries can be parameterized, too

Initialize the GraphQL query and set query variable values…

… and send the parameterized query to the API endpoint

```
@ParameterizedTest
@CsvSource({
        "Amsterdam, Clouds",
        "Berlin, Clear",
        "Rome, Clear"
})
public void useJSONObjectInQuery_checkTheWeather(String cityName, String expectedWeather) {

    HashMap<String, Object> variables = new HashMap<>();
    variables.put("name", cityName);

    HashMap<String, Object> graphQlQuery = new HashMap<>();
    graphQlQuery.put("query", parameterizedQueryString);
    graphQlQuery.put("variables", variables);

    given().
        contentType(ContentType.JSON).
        body(graphQlQuery).
    when().
```

Let's create a test that queries and verifies the weather for three different cities

```
useJSONObjectInQuery_checkTheWeather(String, String)  3 sec 442 ms
  [1] Amsterdam, Clouds                               2 sec 892 ms
  [2] Berlin, Clear                                   297 ms
  [3] Rome, Clear                                     253 ms
```

```
qualTo(expectedWeather));
```

Now it's your turn!

_src > test > java > exercises > RestAssuredExercises6Test.java

_Working with a GraphQL API

_Create a basic query, send it and verify the response

_Create a parameterized query and a data driven test, create and send queries and verify the responses

_Answers are in answers > RestAssuredAnswers6Test.java

_Examples are in examples > RestAssuredExamplesGraphQLTest.java

Now it's your turn!

_src > test > java > exercises > RestAssuredExercises7Test.java

_Capstone assignment

_Combines several concepts we have seen throughout this workshop
  _Extracting values from responses
  _Deserialization
  _Using filters
  _Parameterization, assertions, …

_Answers are in answers > RestAssuredAnswers7Test.java

Contact

_Email: bas@ontestautomation.com

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

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