# Test the REST

## Testing RESTful web services using 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?

_RESTful web services

_REST Assured

_Hands-on exercises

# Preparation

_Install JDK 1.8 (examples and exercises are not guaranteed to work on other JDK versions)

_Install IntelliJ (or any other IDE)

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

# What are RESTful web services?

_HTTP request methods (GET, POST, PUT, …)

_URI's

_CRUD operations on data

| | |
|---|---|
| POST | Create |
| GET | Read |
| PUT | Update |
| DELETE | Delete |

# An example

_GET http://api.zippopotam.us/us/90210

_Result:


```
{
    post code: "90210",
    country: "United States",
    country abbreviation: "US",
    places: [
        {
            place name: "Beverly Hills",
            longitude: "-118.4065",
            state: "California",
            state abbreviation: "CA",
            latitude: "34.0901"
        }
    ]
}
```

# Usage of RESTful web services

_Mobile applications

_Internet of Things

_API Economy

_Web applications

# Why I ♥ testing at the API level

_Tests run much faster than UI-driven tests

_Tests are much more stable than UI-driven tests

_Tests have a broader scope than unit tests

_Business logic is often exposed at the API level

# Tools for testing RESTful web services

_Browser (using plugins like Postman for Chrome)

_Open source (SoapUI, REST Assured)

_COTS (Parasoft SOAtest, SoapUI Pro)

_Build your own (using HTTP libraries for your language of choice)

# REST Assured

- Java DSL for writing tests for RESTful 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

```
<dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>rest-assured</artifactId>
    <version>3.3.0</version>
    <scope>test</scope>
</dependency>
```

# REST Assured documentation

_Usage guide

_https://github.com/rest-assured/rest-assured/wiki/Usage

_Links to other documentation (JavaDoc, getting started, release notes)

_http://rest-assured.io

# A sample test

```
@Test
public void validateCountryForZipCode() {

    given().
    when().
        get( s: "http://api.zippopotam.us/us/90210").          // Do a GET call to the specified resource
    then().
        assertThat().                                          // Assert that the value of the element 'country'
        body( s: "country", equalTo( operand: "United States"));  // in the response body equals 'United States'
}
```

# REST Assured features

- Support for HTTP methods (GET, POST, PUT, …)
- Support for BDD / Gherkin (Given/When/Then)
- Use of Hamcrest matchers for checks (equalTo)
- Use of Jsonpath/GPath for selecting elements from JSON response

```
@Test
public void validateCountryForZipCode() {

    given().
    when().
        get( s: "http://api.zippopotam.us/us/90210").
    then().
        assertThat().
        body( s: "country", equalTo( operand: "United States"));
}
```

# 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() |

_ http://hamcrest.org/JavaHamcrest/javadoc/1.3/org/hamcrest/Matchers.html

# About GPath

_JsonPath is a query language for JSON documents

  _REST Assured using the GPath implementation

_Similar aims and scope as XPath for XML

_Documentation and examples:

  _http://groovy-lang.org/processing-xml.html#_gpath

  _http://groovy.jmiguel.eu/groovy.codehaus.org/GPath.html

# GPath example

```
{
    "post code": "90210",
    "country": "United States",
    "country abbreviation": "US",
    "places": [
        {
            "place name": "Beverly Hills",
            "longitude": "-118.4065",
            "state": "California",
            "state abbreviation": "CA",
            "latitude": "34.0901"
        }
    ]
}
```

```
body("places[0].'place name'", equalTo("Beverly Hills"));
```

# Validating technical response data

_HTTP status code

_MIME-type of received responses

_Cookies and their value

_…

```
@Test
public void checkResponseHeaders() {

    given().
    when().
        get( s: "http://api.zippopotam.us/us/90210").
    then().
        assertThat().
        statusCode(200).
    and().
        contentType(ContentType.JSON);
}
```

# Our API under test

_Zippopotam.us

_Returns location data based on country and zip code

_http://api.zippopotam.us/

_RESTful API

# Demo

_API documentation

_Starting the stub server

_How to use the test suite

_Executing your tests

_Reviewing test results

# Now it's your turn!

- src > test > java > exercises > RestAssuredExercises1Test.java

- Simple checks
  - Validating individual elements
  - Validating collections and items therein
  - Validating technical response properties

- Stubs are predefined
  - You only need to write the tests using REST Assured

- RestAssuredExamples contains the examples shown so far

# 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() {

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

```
@Test
public void usePathParameter() {

    given().
        pathParam( s: "countryCode", o: "us").
        pathParam( s: "zipCode",  o: "90210").
    when().
        get( s: "http://api.zippopotam.us/{countryCode}/{zipCode}").
    then().
        assertThat().
        body( s: "country", equalTo( operand: "UnitedStates"));
}
```

# Using parameters in REST Assured

_Create test data

- _country code and zip code are input values
- _country name is an value expected in the response

```
@DataProvider
public static Object[][] zipCodeData() {
    return new Object[][] {
        { "us", "90210", "United States" },
        { "ca", "B2A", "Canada" }
    };
}
```

# Using parameters in REST Assured

_Use test data for input and output parameters:

```
@Test
@UseDataProvider("zipCodeData")
public void checkCountryForCountryCodeAndZipCode
    (String countryCode, String zipCode, String expectedCountry) {

    given().
        pathParam( s: "countryCode", countryCode).
        pathParam( s: "zipCode", zipCode).
    when().
        get( s: "http://api.zippopotam.us/{countryCode}/{zipCode}").
    then().
        assertThat().
        body( s: "country",equalTo(expectedCountry));
}
```

# Now it's your turn!

- src > test > java > exercises > RestAssuredExercises2Test.java

- Data driven tests
  - Creating a test data object
  - Using test data to call the right URI
  - Using test data in assertions

- RestAssuredExamples contains all examples from the presentation

# Authentication

- Securing web services
- Most common authentication schemes:
  - Basic authentication (username / password)
  - OAuth(2)

# Basic authentication

- Username/password sent in header for every request

- In many APIs, Basic auth. is typically only used to retrieve an (OAuth) authentication token

```
@Test
public void useBasicAuthentication() {

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

# OAuth(2)

- Request of authentication token based on username and password (Basic authentication)

- Include authentication token in header of all subsequent requests

```
@Test
public void useOAuthAuthentication() {

    given().
        auth().
        oauth2( s: "myAuthenticationToken").
    when().
        get( s: "https://my.very.secure/api").
    then().
        assertThat().
        statusCode(200);
}
```

# Sharing variables between tests

_Example: authentication tests

_Copy / paste required for OAuth2 token

_This results in added maintenance burden

_Preferably: store and retrieve for reuse!

# Sharing variables between tests

_REST Assured supports this with extract()

```
private static String myAuthenticationToken;

@BeforeClass
public static void retrieveToken() {

    myAuthenticationToken =

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

@Test
public void usePreviouslyStoredAuthToken() {

    given().
        auth().
        oauth2(myAuthenticationToken).
    when().
        get( s: "https://my.very.secure/api").
    then().
        assertThat().
        statusCode(200);
}
```

# 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?

# Sharing checks between tests

_Solution: ResponseSpecification

```
private static ResponseSpecification responseSpec;

@BeforeClass
public static void createResponseSpec() {

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

@Test
public void useResponseSpec() {

    given().
    when().
        get( s: "http://api.zippopotam.us/us/90210").
    then().
        spec(responseSpec).
    and().
        body( s: "country", equalTo( operand: "United States"));
}
```

# Reusing request properties

_The same can be done for request properties

_Example: set the base URI for the tests

```
private static RequestSpecification requestSpec;

@BeforeClass
public static void createRequestSpec() {

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

```
@Test
public void useRequestSpec() {

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

# Now it’s your turn!

- src > test > java > exercises > RestAssuredExercises3Test.java

- Try it for yourself

- Can you think of additional applications for reuse ?

- RestAssuredExamples contains all examples from the presentation

# 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 XmlPath instead of JsonPath
- No need for additional configuration
  - REST Assured uses response content type header value to determine how to process a response body

# XmlPath - examples

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

```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"));
}
```

Check country for the first car in the list

# XmlPath - examples

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

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

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

Check year for the last car in the list

# XmlPath – examples

```
<?xml version="1.0" encoding="UTF-8" ?>
<cars>
    <car make="Alfa Romeo" model="Giulia">
        <country>Italy</country>
        <year>2016</year>
    </car>
    <car make="A[illegible] model="DB11">
        <country>UK</country>
        <year>1949</year>
    </car>
    <car make="Toyota" model="Auris">
        <country>Japan</country>
        <year>2012</year>
    </car>
</cars>
```

```
@Test
public void checkModelForSecondCar() {

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

Check model for the second car in the list

# XmlPath – examples

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

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

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

Check there’s only one car from Japan in the list

# XmlPath - examples

```
<?xml version="1.0" encoding="UTF-8" ?>
<cars>
    <car make="Alfa Romeo" [redacted]ia">
        <country>Italy</country>
        <year>2016</year>
    </car>
    <car make="Aston Martin" [redacted]">
        <country>UK</country>
        <year>1949</year>
    </car>
    <car make="Toyota" model="Auris">
        <country>Japan</country>
        <year>2012</year>
    </car>
</cars>
```

```
@Test
public void checkTheListContainsTwoCarsWhoseMakeStartsWithAnA() {

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

Check there are two cars in the list whose make starts with 'A'

# Now it's your turn!

- src > test > java > exercises > RestAssuredExercises4Test.java
- Communicating with an API returning an XML document
- Use XmlPath to select the right nodes
- Use filters, in, grep() where needed
- All examples can be reviewed in RestAssuredExamplesXml.java

# (De-)serialization of POJOs

- REST Assured is able to convert POJO instances directly to XML or JSON (and back)
- Useful when dealing with test data objects
- 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>2.9.8</version>
    <scope>test</scope>
</dependency>
```

# Example: serialization

_POJO 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;
    }

    public String getStreet() { return this.street; }

    public int getHouseNumber() { return this.houseNumber; }
```

# Example: serialization

- Instantiating it in a test and sending it as a request body for a POST method:

```
@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);
}
```

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

# Example: deserialization

- We can also convert a JSON (or XML) body back to an instance of a POJO

- After that, we can do some verifications on it:

```
@Test
public void deserializeJsonToAddress() {

    Address myAddress =

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

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

# 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 Car POJO

_All examples can be reviewed in RestAssuredExamples.java

# Now it's your turn!

- src > test > java > exercises > RestAssuredExercises6Test.java

- Capstone assignment

- Combines several concepts we have seen throughout this workshop
  - Extracting values from responses
  - Deserialization
  - Using filters
  - Parameterization, assertions, …

?

https://testautomationu.applitools.com/automating-your-api-tests-with-rest-assured/

# Contact

_Email: bas@ontestautomation.com

_Blog: https://www.ontestautomation.com

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