Contents

Automating REST API Workshop

"Advanced" REST API Automating Tutorial/Workshop

Alan Richardson

- www.eviltester.com
- twitter: @eviltester

Workshop Overview

Slides and Handouts and Links

compendiumdev.co.uk/page/atautorest

- Also on the bottom of every slide.

Description

Workshop/Tutorial means that we will have discussions and do exercises to experiment and learn.

Not all the answers are found in the presenter or the slides. The answers are in the room and in the discussions.

You take control of your workshop.

- If exercises are too easy:
  - try the different serialisation exercises
  - try working through the abstraction exercises early

We aim to cover

- Overview of the basics to achieve common understanding:
  - basics of: API, HTTP, REST
  - tooling interaction: Postman, Insomnia, Proxies

– automating basics: REST Assured

We will then look at the 'advanced' content...

Advanced Content

- More Detailed look at REST Assured library for Automating a REST API in Java
- Automating through an HTTP Proxy
- parsing JSON and XML
- using JAVA HTTP libraries
- Abstraction layers: payload objects, API abstraction, HTTP library abstraction
- Performance/Stress testing of API
- Coverage of API Testing and approaches to support this
- Writing Bots for interleaved API functional testing

Anything else we need to discuss?

Hands on Experience

You will gain hands on experience with:

- Automating a REST API
- Creating Abstraction Layers for a REST API
- Refactoring code to abstraction layers
- Writing bots for an API

Requirements

Requirements to attend workshop:

- You will need a laptop to take part
- Wifi connection
- You will need the ability to install software on to your laptop
- During the workshop we will use REST Client and an HTTP Proxy

---

Requirements

- REST Client
  - Postman (https://www.getpostman.com/)
  - Insomnia (https://insomnia.rest/)
  - any other client
- An HTTP Proxy: Fiddler or Charles or ZAP Proxy or BurpSuite Community Edition
  - BurpSuite Community Edition (https://portswigger.net/burp/)
  - Owasp ZAP (https://www.zaproxy.org/)

Requirements

- Some sample applications are written in Java 1.8 these will only work if you have Java 1.8 (or higher) installed (run `java -version`)
- You need Java Development Kit installed
  - JDK >= 1.8 (`javac -version`)
  - maven (`mvn -version`)
  - IntelliJ IDEA (https://www.jetbrains.com/idea/)

Introduction

About Alan Richardson

www.compendiumdev.co.uk

Blogs: javafortesters.com, seleniumsimplified.com, eviltester.com, testerhq.com

- Linkedin - [@eviltester](https://uk.linkedin.com/in/eviltester)
- Twitter - [@eviltester](https://twitter.com/eviltester)
- Instagram - [@eviltester](https://www.instagram.com/eviltester)
- Facebook - [@eviltester](https://facebook.com/eviltester/)
- Youtube - EvilTesterVideos
- Pinterest - [@eviltester](https://uk.pinterest.com/eviltester/)
- Github - [@eviltester](https://github.com/eviltester/)
- Slideshare - [@eviltester](https://www.slideshare.net/eviltester)

Who are we and what do we do?

- name
- level of knowledge about automating APIs in Java?
- level of experience with automating APIs in Java?
- do we build APIs or test APIs?

What is advanced?

- Simple
- Controlled
- Abstractions for design and maintenance
- Abstractions which support ease of writing
- Not relying on a specific library - freedom to absorb and generate variety
- What else?

SECTION - BASICS

Overview - Browser - Web Application

Example - A Web Application

- Diagram showing browser making GET, POST requests to Server

---

What is HTTP?

- Verbs - GET, POST, DELETE, PUT, HEAD, OPTIONS, PATCH
- URL (URI)
- Headers - cookies, accept formats, user agent, content of message, authentication, etc.
- Data contained in message body - Form, JSON, XML

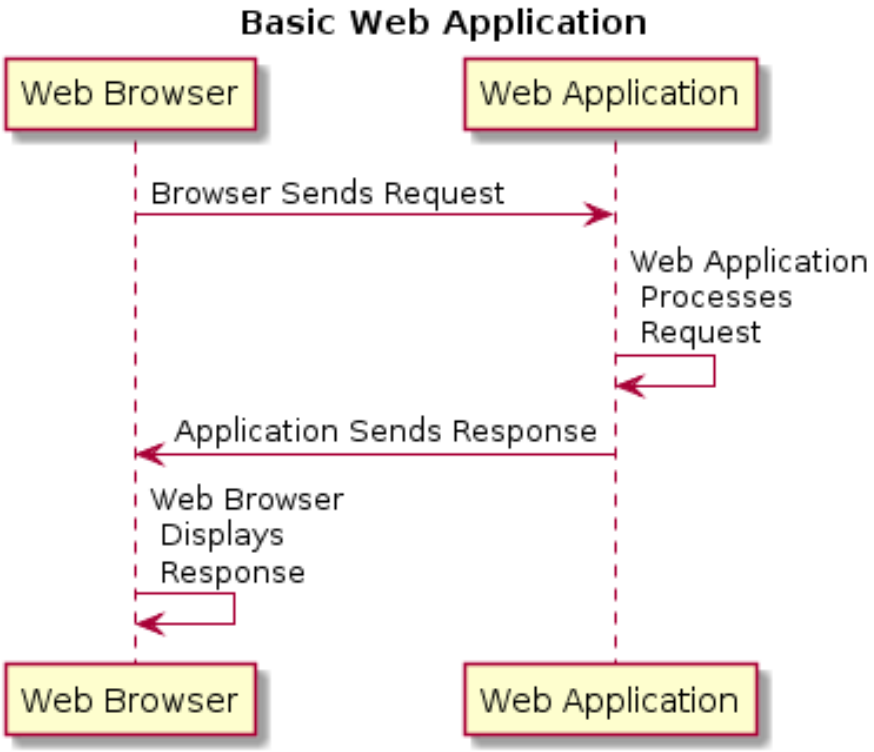

Figure 1: Browser to Web Application

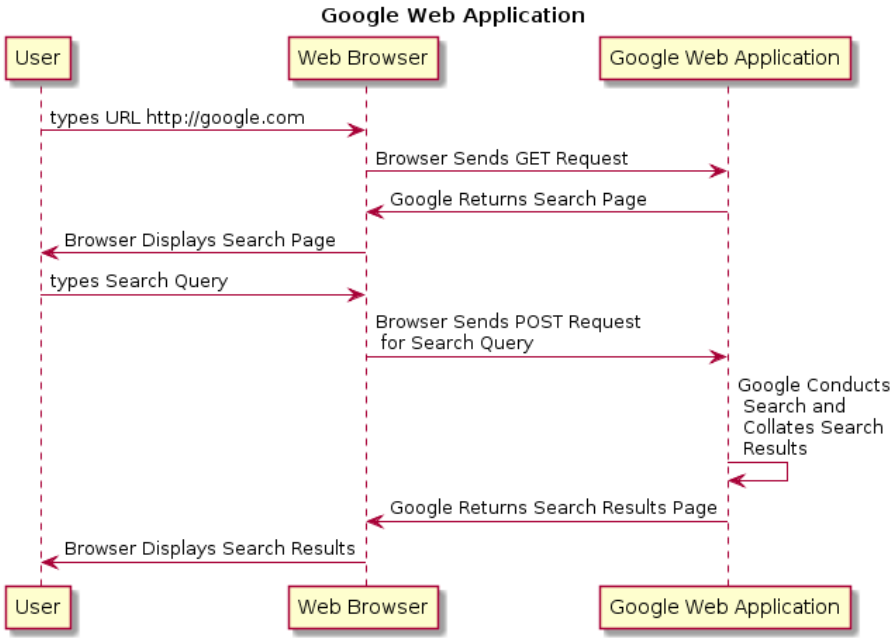

Figure 2: A Web Application

Example HTTP Request

use browser to GET http://compendiumdev.co.uk/apps/api/mock/reflect

Formatted for readability - headers are normally on one line.

```
GET http://compendiumdev.co.uk/apps/api/mock/reflect HTTP/1.1
Host: compendiumdev.co.uk
Connection: keep-alive
Cache-Control: max-age=0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
            AppleWebKit/537.36 (KHTML, like Gecko)
            Chrome/60.0.3112.90 Safari/537.36
Upgrade-Insecure-Requests: 1
Accept: text/html,application/xhtml+xml,
        application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.8
```

HTTP Requests - Human or System

- Human
  - user types URL into browser search bar (GET)
  - user submits form (POST)
- System -System automatically polls server for new content via JavaScript - AJAX (Asynchronous JavaScript and XML) - XHR (XML HTTP Request) - GET / POST - often returns JSON

View Browser Requests in Dev Tools Network Tab

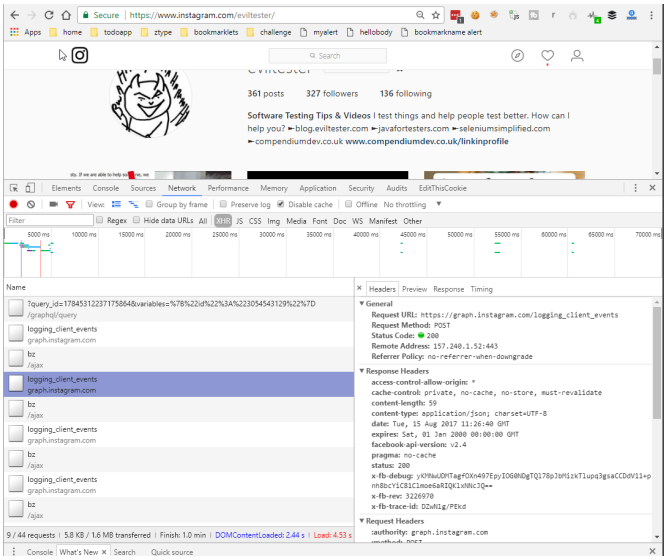

Figure 3: View Browser Requests in Dev Tools Network Tab

What is a Web API?

- Web Service
  - w3.org/TR/ws-gloss
- API
  - wikipedia.org/wiki/Application_programming_interface

Web Application with an interface designed for use by other software.

Why an API?

- Other systems to access
- Customisation
- Mobile Apps often use API
  - bag a SNES classic

JavaScript Using API Via AJAX/XHR

- AJAX/XHR requests have security protocols for same domain
- JSONP for cross domain access
- Very often API is used under covers, e.g. a serverside script/app on same domain uses an API on server side rather than client side

---

We Need Tools

Because Web Service designed for software, we need tools to access them.

Tools

- cURL
  - command line based
  - API examples often shown in cURL
  - recommended that you learn this eventually
  - download
- GUI Clients
  - Postman
  - Insomnia

What is a Web Service / Web Application?

- A web hosted HTTP accessed application without a GUI

What is an API?

- Application Programming Interface designed for use by software

Note: error messages need to be human readable - useful to be machine parsable when automating

---

Overview of Section - HTTP Requests and Responses

- HTTP Verbs - GET, POST, DELETE
- Headers
- Responses
  - Status Codes - e.g. 200, 404, 500
- This is the foundation for most web, HTTP, REST testing and automating.

HTTP GET Request sent from Postman

```
GET http://localhost:4567/heartbeat HTTP/1.1
cache-control: no-cache
Postman-Token: ddf30bfe-b7e2-4d3c-b478-1103a5a174e5
User-Agent: PostmanRuntime/6.2.5
Accept: */*
Host: localhost:4567
accept-encoding: gzip, deflate
Connection: keep-alive
```

- important stuff: Verb (GET), Http version (1.1), User-Agent, Accept, Host, endpoint

---

HTTP Response to Postman GET /heartbeat request

```
HTTP/1.1 200 OK
Date: Thu, 17 Aug 2017 10:34:32 GMT
Content-Type: application/json
Transfer-Encoding: chunked
Server: Jetty(9.4.4.v20170414)
```

- cURL response was same but content-type was `application/xml`
- important stuff: Status Code (200 OK), Http version (1.1), Date, Content-Type

---

Automating with REST Assured

- https://github.com/rest-assured/rest-assured
- Java/Groovy library
- HTTP Abstraction
- Assertions Abstraction
- Given When Then Abstraction
- JSON Path and XML Path for adhoc assertions and extract
- Marshalling - Serialization/Deserialization

Add REST Assured to pom.xml

```
<dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>rest-assured</artifactId>
    <version>3.2.0</version>
    <scope>test</scope>
</dependency>
```

GET request using REST Assured

```
    String rootUrl = "http://localhost:4567/listicator";

    @Test
    public void canCheckThatServerIsRunning(){

        RestAssured.
            get(rootUrl + "/heartbeat").
            then().assertThat().
                statusCode(200);
    }
```

see `RestAssuredBasicsTest.java`

Basic HTTP Verbs

- GET - retrieve data

- POST amend/create from partial information
- PUT - create or replace from full information
- DELETE - delete items
- OPTIONS - verbs available on this url

References

- W3c Standard
- IETF standard
- httpstatuses.com
- http://www.restapitutorial.com/lessons/httpmethods.html

User-Agent Header

- Often not sent when accessing an API
- Marks request as coming from a browser

```
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/60.0.3112.90 Safari/537.36
```

Sending User-Agent Header with REST Assured

```
@Test
public void canSetHeaders(){

    RestAssured.
        given().
            header("User-Agent",
                "Mozilla/5.0 (Windows NT 10.0; Win64; x64)").
        get(rootUrl + "/heartbeat").
        then().assertThat().
            statusCode(200);
}
```

see `RestAssuredBasicsTest.java`

Accept Header

- Defines the payload types that the receiver will accept
- If this was an API call it would likely return XML

```
Accept: text/html,application/xhtml+xml,application/xml;
q=0.9,image/webp,image/apng,*/*;q=0.8
```

Common values:

- `text/html`
- `application/json`
- `application/xml`

Sending Accept Header with REST Assured

```
final Response response =
          RestAssured.
              given().
                  auth().preemptive().
                  basic("admin", "password").
                  accept("application/json").
              when().
              get(rootUrl + "/users")
                  .andReturn();
```

see `RestAssuredBasicsTest.java` method `getUsersAsJSON`

HTTP Status Codes

- 1xx Informational
  - 100 Continue
- 2xx Success
  - e.g. 200 OK
- 3xx Redirection
  - e.g. 301 Moved Permanently
- 4xx Client Error
  - e.g. 404 Not Found
- 5xx Server Error
  - e.g. 500 Internal Server Error

Common HTTP Status Codes

Status Code	Status Code
200 OK	405 Method Not Allowed
201 Created	409 Conflict
301 Moved Permanently	500 Internal Server Error
307 Temporary Redirect	501 Not Implemented
400 Bad Request	502 Bad Gateway
401 Unauthorized	503 Service Unavailable
403 Forbidden	504 Gateway Timeout
404 Not Found	

HTTP Status code references

- https://httpstatuses.com/
- https://moz.com/blog/response-codes-explained-with-pictures
- https://http.cat/
- https://httpstatusdogs.com/

Common HTTP Status codes in response to a GET

- **200** - OK, found the url, returned contents
- **301, 307, 308** - content has moved, new url in `location` header
- **404** - url not found
- **401** - you need to give me authorisation details see `WWW-Authenticate` header
- **403** - url probably exists but you are not allowed to access it

---

Basic Auth Header

- This application uses Basic Auth Authentication
- `Authorization` Header

e.g. `Authorization: Basic dXNlcjpwYXNzd29yZA==`

`dXNlcjpwYXNzd29yZA==` is base64 encoded "user:password"

see base64decode.org

- cURL you need to add the header
- Postman & Insomnia use the Authorization and Auth tabs

Basic Auth with REST Assured

```
RestAssured.
given().
    auth().preemptive().
    basic("admin", "password").
    accept("application/json").
when().
get(rootUrl + "/users")
```

see `RestAssuredBasicsTest.java` method `getUsersAsJSON`

How else might we authenticate API calls?

Discuss

HTTP POST Verb

- POST amend/create from partial information
- send a 'body' format of content in the 'content-type' header
- usually used to create or amend data
- browser will usually send a POST request when submitting a form

---

HTTP POST Verb Request Example

```
POST http://localhost:4567/lists HTTP/1.1
User-Agent: curl/7.39.0
Host: localhost:4567
Connection: Keep-Alive
accept: application/json
content-type: application/json
Authorization: Basic dXNlcjpwYXNzd29yZA==
```

```
Content-Length: 22

{title:'a list title'}
```

HTTP POST Verb Response Example

```
HTTP/1.1 201 Created
Date: Thu, 17 Aug 2017 12:11:12 GMT
Content-Type: application/json
Location: /lists/f8134dd6-a573-4cf5-a6c6-9d556118ed0b
Server: Jetty(9.4.4.v20170414)
Content-Length: 171

{"lists":[{
"guid":"f8134dd6-a573-4cf5-a6c6-9d556118ed0b",
"title":"a list title",
"description":"",
"createdDate":"2017-08-17-13-11-12",
"amendedDate":"2017-08-17-13-11-12"}]}
```

HTTP POST Request with REST Assured

```
    RestAssured.
        given().
            auth().preemptive().
            basic("admin", "password").
            body("{title : 'my title'}").
        when().
            post(rootUrl + "/lists").
        then().assertThat().
            statusCode(201);
```

Common HTTP Status codes in response to a POST

- **200** - OK, did whatever I was supposed to
- **201** - OK created new items
- **202** - OK, I'll do that later
- **204** - OK, I have no more information to give you

- **400** - what? that request made no sense
- **404** - I can't post to that url it is not found
- **401** - need authorisation see `WWW-Authenticate` header
- **403** - url probably exists but you are not allowed to access it
- **409** - can't do that, already exists
- **500** - your request made me crash

HTTP Message Body Format - JSON

- JSON - JavaScript Object Notation
- an actual Object in JavaScript
- common data transfer and marshalling format for other languages
- https://en.wikipedia.org/wiki/JSON
- http://json.org
- http://countwordsfree.com/jsonviewer
- schema exists for JSON http://json-schema.org/

JSON Example Explained

```
{
    "users": [
        {
            "username": "superadmin"
        },
        {
            "username": "admin"
        },
        {
            "username": "user"
        }
    ]
}
```

- An object, which has an array called "users".
- the users array contains an object with field: `username`.

Basic JSON Parsing in REST Assured

```
    @Test
```

```
    public void getUsersWithJSON(){

        RestAssured.
            given().
                auth().preemptive().
                basic("admin", "password").
                accept("application/json").
            when().
                get(rootUrl + "/users").
            then().
                body("users[0].username",
                        equalTo("superadmin")).
                and().body("users[1].username",
                        equalTo("admin"));
    }
```

see `RestAssuredBasicsTest.java` method `getUsersWithJSON`

REST Assured can also parse response with JSON Path

```
final Response response =
RestAssured.
given().
    auth().preemptive().
    basic("admin", "password").
    accept("application/json").
when().get(rootUrl + "/users")
    .andReturn();

List<String> usernames =
    response.jsonPath().getList("users.username");
Assert.assertTrue(usernames.contains("superadmin"));
Assert.assertEquals("superadmin",
    response.jsonPath().getString("users[0].username"));
```

XML Example Explained

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<users>
  <users>
    <user>
      <username>superadmin</username>
```

```
    </user>
    <user>
      <username>admin</username>
    </user>
    <user>
      <username>user</username>
    </user>
  </users>
</users>
```

- elements, nested elements
- tags, values

HTTP Message Body Format - XML

- XML - eXtended Markup Language
- HTML is often XML
- another common marshalling format
- can be validated against XML schema
- http://countwordsfree.com/xmlviewer

Basic XML Parsing in REST Assured

```
@Test
public void getUsersWithXML(){

    RestAssured.
        given().
            auth().preemptive().
            basic("admin", "password").
            accept("application/xml").
        when().
            get(rootUrl + "/users").
        then().
            body("users.users.user[0].username",
                    equalTo("superadmin")).
            and().body("users.users.user[1].username",
            equalTo("admin"));
}
```

see `RestAssuredBasicsTest.java` method `getUsersWithXML`

REST Assured can also parse response with XML Path

```
final Response response =
RestAssured.
given().
    auth().preemptive().
    basic("admin", "password").
    accept("application/xml").
when().get(rootUrl + "/users")
    .andReturn();

List<String> usernames = response.xmlPath().
    getList("users.users.user.username");
Assert.assertTrue(usernames.contains("superadmin"));
Assert.assertEquals("superadmin",
    response.xmlPath().getString(
        "users.users.user[0].username"));
```

HTTP DELETE Verb

- DELETE - delete items

HTTP DELETE Request Example

```
DELETE http://localhost:4567/lists/{guid} HTTP/1.1
User-Agent: curl/7.39.0
Host: localhost:4567
Accept: */*
Connection: Keep-Alive
Authorization: Basic YWRtaW46cGFzc3dvcmQ=
```

HTTP DELETE Response Example

```
HTTP/1.1 204 No Content
Date: Thu, 17 Aug 2017 12:20:35 GMT
Content-Type: application/json
```

```
Server: Jetty(9.4.4.v20170414)
```

HTTP Delete using REST Assured

```
RestAssured.
    given().
        auth().preemptive().
        basic("admin", "password").
        body("{title : 'my new title'}").
    when().
        delete(rootUrl + "/lists" + "/" + aGuid).
    then().assertThat().
        statusCode(204);
```

Common HTTP Status codes in response to a DELETE

- **200** - OK, did whatever I was supposed to
- **202** - OK, I'll do that later
- **204** - OK, I have no more information to give you
- **404** - I can't post to that url it is not found
- **401** - you need to give me authorisation details see `WWW-Authenticate` header
- **403** - url probably exists but you are not allowed to access it
- **500** - your request made me crash

URI - Universal Resource Identifier

`scheme:[//[user[:password]@]host[:port]][/path][?query][#fragment]`

- `http://compendiumdev.co.uk/apps/api/mock/reflect`
  - scheme = `http`
  - host = `compendiumdev.co.uk`
  - path = `apps/api/mock/reflect`

wikipedia.org/wiki/Uniform_Resource_Identifier

A URL is a URI

URI vs URL vs URN

- URI - Universal Resource Identifier
  - 'generic' representation - might not include the 'scheme'
  - `http://compendiumdev.co.uk/apps/api/mock/reflect`
  - `compendiumdev.co.uk/apps/api/mock/reflect`
  - `/apps/api/mock/reflect`
- URL - Universal Resource Locator
  - `http://compendiumdev.co.uk/apps/api/mock/reflect`
  - defines how to locate the identified resource
- URN - Universal Resource Name
  - not often used - uses scheme `urn`

Scheme(s)

- http
- https
- ftp
- mailto
- file

Query Strings

```
GET /lists/{guid}?without=title,description
GET http://localhost:4567/lists/f13?without=title,description
```

Query String:

```
?without=title,description
```

- starts with `?`
- params separated with `&`

More About Query Strings

```
GET /lists/{guid}?without=title,description
```

- usually `name=value` pairs separate by '&'
  - convention since anything after the `?` is the Query string
  - app then parses as required

- can be used with any verb
- `GET` request - all params are send as query strings

https://en.wikipedia.org/wiki/Query_string

HTTP Standards?

- rfc7231 (HTTP/1.1): Semantics and Content
- rfc7230 (HTTP/1.1): Message Syntax and Routing

HTTP PUT Verb

- PUT - create or replace from full information

Full information means it should be idempotent - send it again and get exactly the same request

---

HTTP PUT Request Example

```
PUT http://localhost:4567/lists HTTP/1.1
User-Agent: curl/7.39.0
Host: localhost:4567
Accept: */*
Connection: Keep-Alive
Authorization: Basic dXNlcjpwYXNzd29yZA==
Content-Length: 180
Content-Type: application/json

{"title":"title added with put",
"description":"list description",
"guid": "guidcreatedwithput201708171440",
"createdDate": "2017-08-17-14-40-34",
"amendedDate": "2017-08-17-14-40-34"}
```

HTTP PUT Response Example

```
HTTP/1.1 201 Created
```

```
Date: Thu, 17 Aug 2017 13:41:46 GMT
Content-Type: application/json
Server: Jetty(9.4.4.v20170414)
Content-Length: 0
```

HTTP PUT REST Assured Example

```
String aGuid = UUID.randomUUID().toString();
String aTitle = "My Put Title";
String aDescription = "My mini description";
String createdDate="2018-12-08-13-50-40";
String amendedDate=createdDate;

String message = String.format(
    "{guid:'%s',title:'%s',description:'%s'," +
    " createdDate:'%s', amendedDate:'%s'}",
    aGuid, aTitle, aDescription, createdDate, amendedDate);

RestAssured.
    given().
        auth().preemptive().
        basic("admin", "password").
        body(message).
    when().
        put(rootUrl + "/lists").
    then().assertThat().
        statusCode(201);
```

HTTP OPTIONS Verb

- OPTIONS - verbs available on this url
- returns an `Allow` header describing the allowed HTTP Verbs

HTTP OPTIONS Request Example

```
OPTIONS http://localhost:4567/lists HTTP/1.1
User-Agent: curl/7.39.0
Host: localhost:4567
```

```
Accept: */*
Connection: Keep-Alive
```

HTTP OPTIONS Response Example

```
HTTP/1.1 200 OK
Date: Thu, 17 Aug 2017 12:24:39 GMT
Allow: GET, POST, PUT
Content-Type: text/html;charset=utf-8
Server: Jetty(9.4.4.v20170414)
Content-Length: 0
```

HTTP Options REST Assured Example

```
    @Test
    public void options(){
        RestAssured.
            given().
                auth().preemptive().
                basic("admin", "password").
            when().
                options(rootUrl + "/lists").
            then().
            assertThat().
                statusCode(200).
                header("Allow",
                "GET, POST, PUT");
    }
```

Common HTTP Status codes in response to a OPTIONS

- **200** - OK, did whatever I was supposed to
- **404** - I can't post to that url it is not found

HTTP OPTIONS Verb - Example swapi.co

e.g. swapi.co

OPTIONS - https://swapi.co/api/people/1/

```
{
    "name": "People Instance",
    "description": "",
    "renders": [
        "application/json",
        "text/html",
        "application/json"
    ],
    "parses": [
        "application/json",
        "application/x-www-form-urlencoded",
        "multipart/form-data"
    ]
}
```

Verb - Head

- HEAD
- same as GET but does not return a 'body'
- can be useful for checking 'existence' of an endpoint or entity

---

HEAD REST Assured Example

```
RestAssured.
    given().
        auth().preemptive().
        basic("admin", "password").
    when().
        head(rootUrl + "/lists").
    then().
        assertThat().
        statusCode(405);
```

Verb - Patch

- PATCH - An 'Update' method which provides a set of changes
- Contentious see
- Proposed standard for JSON Merge Patch format
- Promosed standard for XML Patch Using XPath

Most web services just use `POST` or `PUT`

SECTION - REST API BASICS

Overview of Section - REST API BASICS

- What is a REST API?
- CRUD and REST
- HTTP Verbs - HEAD, PATCH
- Authentication and Authorisation
- Postman collection runner

What is a REST API?

- HTTP API - generic, anything goes
- REST API
  - the HTTP Verbs mean something specific e.g. should not Delete with a POST request
  - URI are 'nouns' and describe entities

REST Standards?

Representational State Transfer

- Loose standards
- Lots of disagreement on teams and online

- DISSERTATION: “Architectural Styles and the Design of Network-based Software Architectures” by Roy Fielding
  - ics.uci.edu/~fielding/pubs/dissertation/top.htm

Guidance

- Idempotent - same request, same result (on server, not necessarily in response)
- Stateless - server does not need to maintain state of client requests between requests e.g.
  - request 1: select these files,
  - request 2: delete files selected in previous request
- Cacheable - on the server side e.g. GET can be cacheable until entities in GET are updated
- Does it comply with HTTP Standard Guidance?

CRUD

- Verbs are not as simple as Create, Read, Update, Delete

CRUD Action	Verb
Create	POST, PUT
Read	GET
Update	POST, PUT, PATCH
Delete	DELETE

Endpoints vs URL

Very often when discussing REST APIs we talk about ‘endpoints’.

Basically the ‘path’ part of the URL.

The following are the same Endpoint

- `/lists`
- `/lists?title="title"`

Payloads vs Body

A Payload is the content of the body of the HTTP request.

- XML and JSON
- Tends not to be Form encoded
- Request defined by `content-type` header
- Response requested in `accept` header
- usually unmarshalled into an object in the application

Requesting Formats

Header	Means
Accept: application/json	Please return JSON
Accept: application/xml	Please return XML
Content-Type: application/json	This payload is JSON
Content-Type: application/xml	This payload is XML

- XML might also be : `text/xml`
- The server might not support a particular format it might default to JSON or XML and ignore the header

---

Authentication

If you make a request to a server and receive a 401 then you are not authenticated.

`WWW-Authenticate` header should challenge you with the authentication required.

- Generally avoid header sending by known authenticating information in request.
- Common bug is `WWW-Authenticate` not sent back in response.

Common Authentication Approaches

- Basic Auth Header
  - `Authorization: Basic Ym9iOmRvYmJz`
  - base 64 encoded `username:password`
- Cookies
  - when 'login' server sends back a 'session cookie'
  - send 'session cookie' in future requests
- Custom Headers
  - API `secret codes`
  - e.g. `X-API-AUTH: thisismysecretapicode`

Common Authentication Approaches

- URL authentication
  - `https://username:password@www.example.com/`
  - deprecated
  - used to be very common when automating web GUIs

Recommended reading developer.mozilla.org/en-US/docs/Web/HTTP/Authentication

---

Authentication vs Authorization

Authentication

- Are you authenticated?
- Does the system know who you are?
- Are your auth details correct?

Authorization

- you are authenticated
- do you have permission to access this endpoint?

Real World vs Standards

Teams debate this all the time.

- Login? stackoverflow.com/questions/13916620

- Put vs Post stackoverflow.com/questions/630453
- see discussions on restcookbook.com

As a Tester:

- Refer to HTTP standards
  - headers, idempotency, response recommendations

Expect 'discussions' and 'debates' on a team.

How to test with this information

- Read the standards for the verbs and the status codes.
- Projects often argue about interpretations.
- Some of the standards are exact enough that it is possible to say "I observed X" it does not match the standard - include links and quotes to the standards.

SECTION - GUI Tools

GUI Clients

- Postman GetPostman.com
- Insomnia insomnia.rest

Benefits - Collections/Workspaces, Environment variables, Visual, Easy to send through proxy

---

Postman Collections

- "save as" requests to collections for re-use
- can share collections or export to file

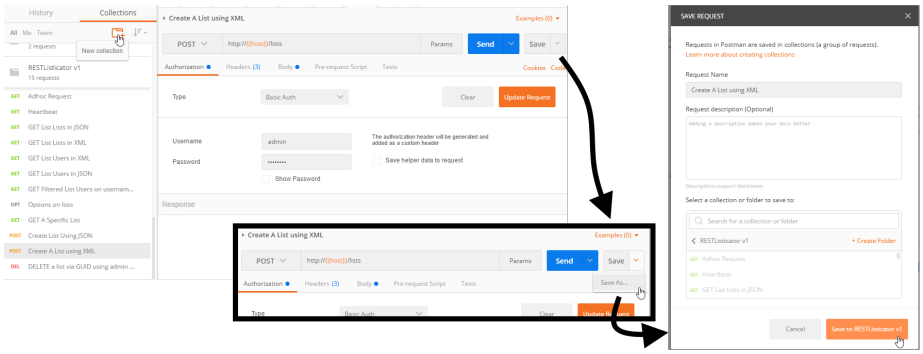

Figure 4: Postman Collections

Postman Collections

Insomnia Workspaces

- create new Workspace
- create new Request in Workspace
- changes automatically saved to workspace
- can export workspace to files

Insomnia Workspaces

Environment Variables

Postman:

- use environment variables e.g. `{{host}}` instead of `localhost:4567`
- `GET htttp://{{host}}/lists`

Insomnia:

- use environment variables e.g. `{'host':'localhost:4567'}`
- just type `host` for auto complete in URL editing

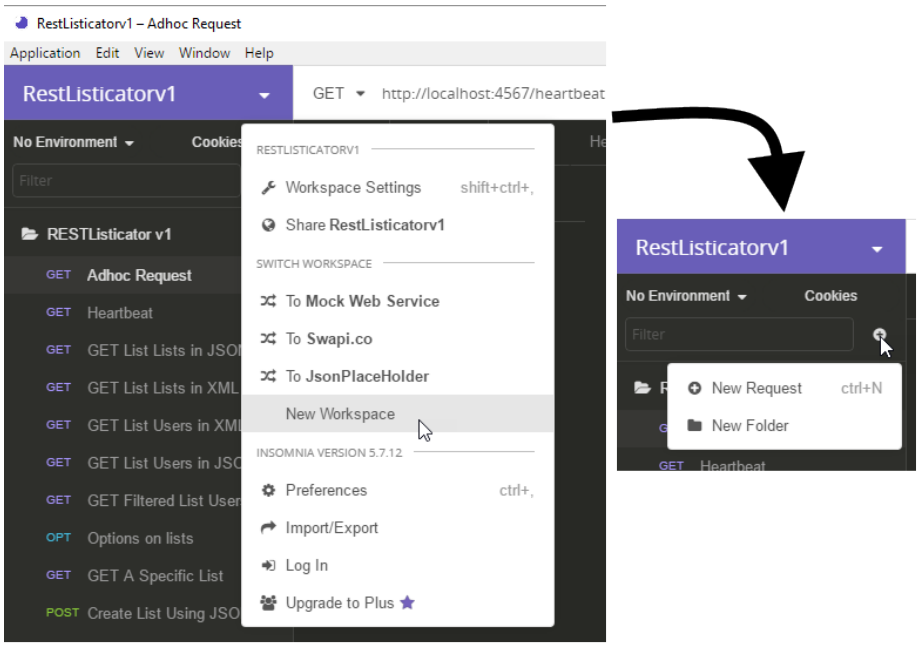

Figure 5: Insomnia Workspaces

Postman Create Environment

Postman Add Environment Variables

Insomnia Environment Management

SECTION - HTTP Proxies

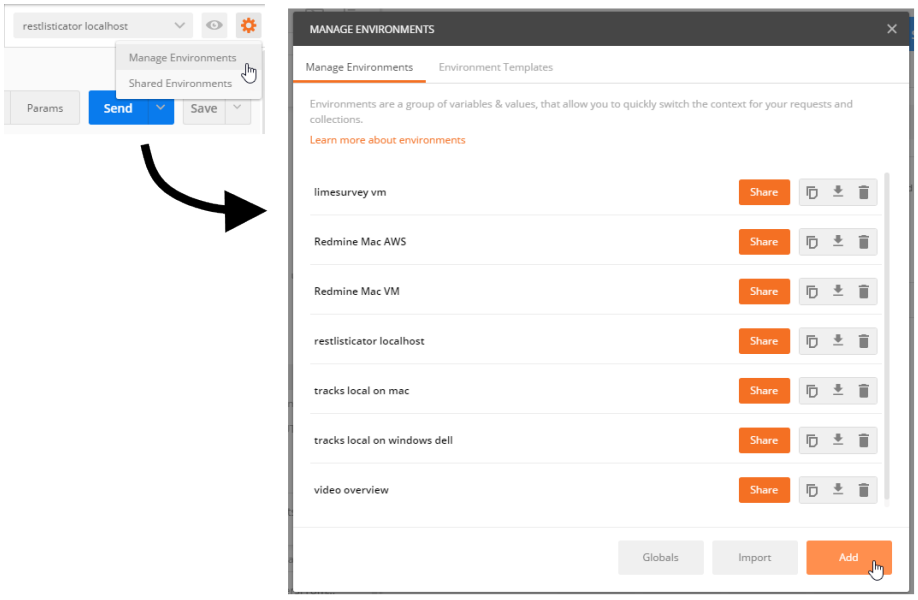

Figure 6: Postman Create Environment

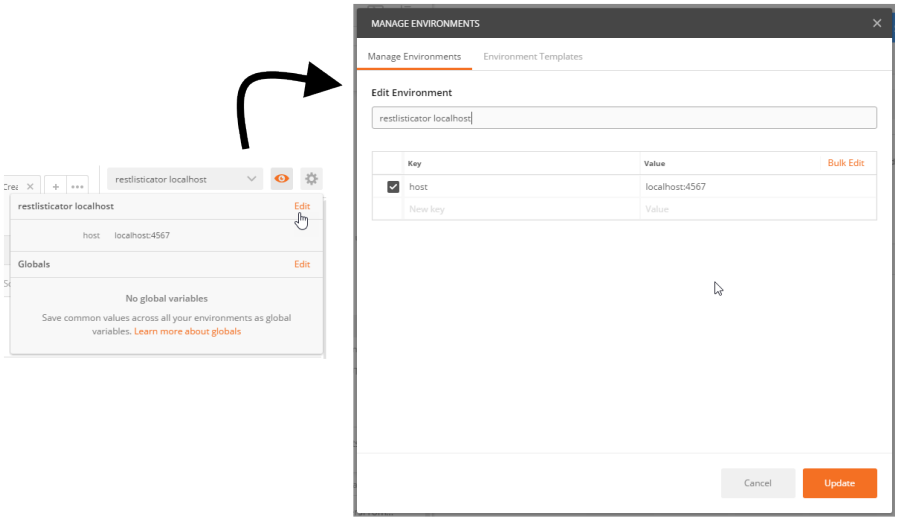

Figure 7: Postman Add Environment Variables

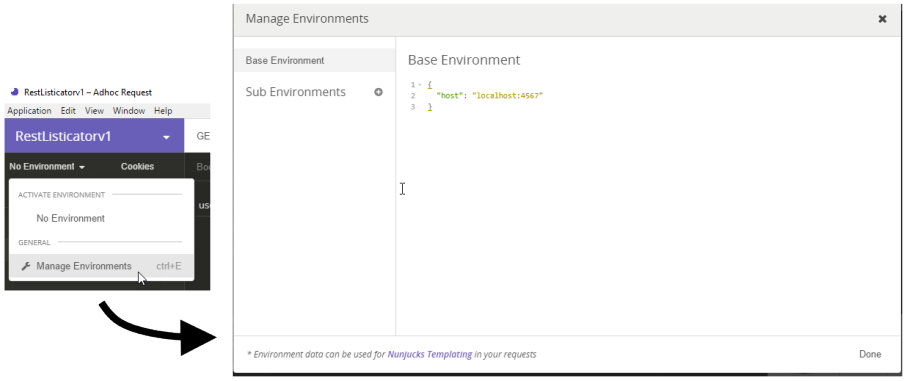

Figure 8: Insomnia Environment Management

Overview of Section - HTTP Proxies

- What is an HTTP Proxy?
- Example HTTP Proxies?
- Why use an HTTP Proxy?
- How to direct REST Client through Proxy?
  - Inspect Traffic
  - Filter Traffic (System Proxies)
  - Port Config
  - Replay Request
- Fuzzing

What is an HTTP Proxy?

- HTTP Proxy captures HTTP Traffic
- Allows replay of requests
- Allows manipulation of responses

Which proxies?

- Fiddler
  - Windows (Beta: Linux, Mac)
- Charles
  - Commercial but allows 30 mins in 'shareware' mode

- BurpSuite
  - Free edition good enough for API Testing
- Owasp ZAP
  - Open Source

Fiddler & Charles act as System Proxies making them easy to use with Postman.

Why use when Testing API?

- Record requests
- Create evidence of your testing
- Replay requests outside of client tool
- Fuzzing

When Automating?

- view requests sent to aid debugging and coverage

Using a Proxy When Automating with REST Assured

```
RestAssured.proxy("localhost", 8080);
```

- RestAssured is a singleton
- all requests will be made through proxy after this is run
- setting multiple times does not matter

see `RestAssuredProxyTest`

Exercises: Basic Concepts and Tooling

- assuming the Testing App Running at [http://localhost:4567/listicator/]
- run the example tests `RestAssuredBasicsTest`
- get used to the API
  - read the docs and experiment with Postman, Insomnia
  - GET, POST, DELETE, OPTIONS and POST
- run the example tests through a proxy `RestAssuredProxyTest`
- replay requests through a proxy
- get used to the app and explore the HTTP Client functionality and explore the API based on its documentation

---

Optional Exercises: Or Test REST Listicator in Buggy mode

```
java -jar compendium-of-test-apps-1-3-3.jar -bugfixes=false
```

- The system has been coded with some known bugs
- these are all fixed by default.
- start with `-bugfixes=false` to have known bugs
- See if you can find them

You can run the app twice on different ports to compare output, use the command line argument `-port` to start up the application on a different port e.g. `-port=1234` would start the app on port `1234`

SECTION - PAYLOAD OBJECTS

What are Payload Objects?

- Java Classes which represent payload sent or response received
- We can convert the received JSON or XML into Java Objects to Process them
  - Marshalling/Unmarshalling
  - Serialising/De-serialising
- This is a **fundamental Abstraction layer** used for automating API systems

---

Why Payload Objects?

But we already have the ability to perform queries on the responses in RESTAssured to get the details we need:

e.g.

```
body("shopping.category.find { it.@type == 'groceries' }.item"
    ,hasItems("Chocolate", "Coffee"));
```

from official REST Assured Documentation

```
then().
    body("users[0].username",
```

```
            equalTo("superadmin")).
    and().body("users[1].username",
            equalTo("admin"));
```

from ***RestAssuredBasicsTest.java***

Over-reliance on REST Assured and Groovy

- complicated query syntax
- need to learn Groovy GPath groovy-lang.org/processing-xml.html#_gpath
- hard to debug
- hard to maintain

NOTE: personal opinion

- I would rather convert to Java Objects and write simple Java code

---

REST Assured Marshalling with POJO

- REST Assured uses GSON and JAXB to marshall/unmarshall objects
- create normal POJO
- JSON requires no annotations
- XML requires JAXB annotations

```
@XmlRootElement(name="list")
private static class AListPayload{

    public String guid;
    public String title;
    public String description;
    public String owner;
    public String createdDate;
    public String amendedDate;
}
```

github.com/rest-assured/rest-assured/wiki/Usage#object-mapping

---

REST Assured Marshalling with POJO

```
AListPayload desiredList = new AListPayload();
desiredList.title = "my list " + System.currentTimeMillis();
```

```
final Response response = RestAssured.
        given().
            auth().preemptive().
            basic("admin", "password").
        body(desiredList).
        when().
            post(rootUrl + "/lists").andReturn();

Assert.assertEquals(201, response.getStatusCode());
```

see `RestAssuredMarshallingTest.java` method `simpleCreateListWithJSON`

REST Assured UnMarshalling with POJO

```
    @XmlRootElement(name = "lists")
    @XmlAccessorType(XmlAccessType.FIELD)
    private static class AListsCollectionResponse{
        @XmlElement(name="list")
        public List<AListPayload> lists;
    }

final AListsCollectionResponse lists =
        RestAssured.
            when().
        get(rootUrl + "/lists").
            as(AListsCollectionResponse.class);

int currentNumberOfLists = lists.lists.size();
```

see `RestAssuredMarshallingTest.java`

Exercises - Payload Object Abstractions

- Using `RestAssuredMarshallingTest`
- Run tests and make sure you understand them
- It might help to run them through a proxy
- see exercises at the bottom of the class file

SECTION - ALTERNATIVE MARSHALLING STRATEGIES

REST Assured is a wrapper

- REST Assured is a wrapper around Jaxb and GSON
- We could use alternative marshalling/unmarshalling if we wanted
- Might avoid reliance on REST Assured

REST Assured JsonPath and Jaxb

Converting to an object - uses Gson under the covers

```
AListsCollectionResponse lists =
    new JsonPath(getLists.body().asString()).
        getObject(".", AListsCollectionResponse.class);
```

Converting from an object uses Groovy `JsonBuilder`

```
final String payload =
    new JsonBuilder(payloadObject).toString();
```

- see `RestAssuredJsonPathMarshallingTest`

We could use Gson directly

Converting to an Object

```
AListsCollectionResponse lists =
        new Gson().fromJson(
            getLists.body().asString(),
            AListsCollectionResponse.class);
```

Coverting Object to Json

```
AListPayload payloadObject =
    new AListPayload("my list title");
String payload = new Gson().toJson(payloadObject);
```

Benefit - Gson is widely used and documented

Rest Assured using XmlPath and Jaxb

Convert XML to an Object

```
AListsCollectionResponse lists =
    new XmlPath(response.body().asString()).
        getObject(".", AListsCollectionResponse.class);
```

Convert Object to XML using Jaxb

```
AListPayload desiredList = new AListPayload("my list title");
StringWriter payloadWriter = new StringWriter();
JAXB.marshal(desiredList, payloadWriter);
String payload = payloadWriter.toString();
```

- Jaxb is widely known, but requires annotations on objects

Pure Jaxb implementation

I had to remove the BOM (Byte order mark) to allow Jaxb to process the message

```
String fixOutput = response.body().asString()
        .replace("\uFEFF", "");

AListsCollectionResponse lists =
    JAXB.unmarshal(new StringReader(fixOutput),
                   AListsCollectionResponse.class);
```

No change to converting object to XML

Xstream Implementation

- XStream is from thoughtworks
- requires no annotations to class - configured from within code
- http://x-stream.github.io/

```
xstream = new XStream();

xstream.allowTypesByWildcard(new String[] {
        "uk.co.compendiumdev.advancedrestautomating.**"
});

xstream.alias("lists", AListsCollectionResponse.class);
```

```
xstream.addImplicitCollection(AListsCollectionResponse.class,
            "lists");
xstream.alias("list", AListPayload.class);
```

Xstream Implmentation for Conversion

Convert from XML to Object

```
AListsCollectionResponse lists = (AListsCollectionResponse)
    xstream.fromXML(response.body().asString());
```

Convert from Object to XML

```
AListPayload desiredList = new AListPayload("my list title");

// xstream does not add the XML header by default
String header = "<?xml version=\"1.0\" encoding" +
                "=\"UTF-8\" standalone=\"yes\"?>\n";
String payload = header + xstream.toXML(desiredList);
```

- Xstream is lightweight if you need to deploy a jar
- configurable within code so POJOs without annotations

Why?

If we ever want to split our code into:

- HTTP Abstractions
- Domain Abstractions

Then we may be able to avoid RestAssured 'bleeding' into the other areas.

Demos using the marshalling package

- `RestAssuredMarshallingTest`
  - uses RestAssured to convert to/from objects
- `RestAssuredJsonPathMarshallingTest`
  - uses `JsonPath` and `JsonBuilder` directly
- `GsonMarshallingTest`
  - uses `Gson` directly
- `RestAssuredXmlPathMarshallingTest`

  - uses `XmlPath` and `Jaxb` directly
- `JaxbXmlMarshallingTest`
  - uses `Jaxb` directly
- `XstreamXmlMarshallingTest`
  - uses `Xstream` directly

SECTION - ALTERNATIVE HTTP STRATEGIES

REST Assured provides HTTP abstractions

```
final Response response =
    RestAssured.
        given().
            accept("application/xml").
        when().
            get(rootUrl + "/lists").andReturn();
```

- `Response` is a REST Assured HTTP response
- `accept` creates and HTTP header
- `get` issues an `HTTP GET` request

REST Assured wraps Apache httpclient

- REST Assured usess Apache HTTP Client
- we could use HTTP Client directly
- if we wanted to avoid external libraries we could use `java.net` directly
  - but we would have to write more code

Why?

- more control
- fewer dependencies
- avoid mixing HTTP with `@Test` code
- separation of Abstraction Layers

Apache HTTPClient

Create an HTTP Client

```
CredentialsProvider credentialsProvider =
                    new BasicCredentialsProvider();
UsernamePasswordCredentials credentials
     = new UsernamePasswordCredentials("admin", "password");
credentialsProvider.
        setCredentials(AuthScope.ANY, credentials);

CloseableHttpClient client = HttpClientBuilder.create().
        setDefaultCredentialsProvider(credentialsProvider).
        build();
```

see `GsonMarshallingHttpClientsExampleTest`

Use Client to issue requests

```
HttpResponse getLists =
        client.execute(new HttpGet(rootUrl + "/lists"));
Assert.assertEquals(200,
        getLists.getStatusLine().getStatusCode());
```

Get Body

```
EntityUtils.toString(getLists.getEntity()
```

see `GsonMarshallingHttpClientsExampleTest`

Converting Body Json to object using GSON

```
HttpResponse getLists =
        client.execute(new HttpGet(rootUrl + "/lists"));
Assert.assertEquals(200,
        getLists.getStatusLine().getStatusCode());

final AListsCollectionResponse lists =
        new Gson().fromJson(
            EntityUtils.toString(getLists.getEntity()),
                AListsCollectionResponse.class);
```

see `GsonMarshallingHttpClientsExampleTest`

Example Post Request with authentication

```
HttpPost postrequest = new HttpPost(rootUrl + "/lists");

/* this uses pre-emptive auth */
postrequest.addHeader(
    new BasicScheme().
        authenticate(credentials, postrequest, null));

postrequest.addHeader(HttpHeaders.ACCEPT,
                      "application/json");

// set the body of the request
postrequest.setEntity(new StringEntity(payload));

HttpResponse response = client.execute(postrequest);
```

see `GsonMarshallingHttpClientsExampleTest`

Example getting headers from response

```
Header[] headers = response.getHeaders(HttpHeaders.LOCATION);
payloadObject.guid  =
     headers[0].getValue().replace("/lists/","");
```

see `GsonMarshallingHttpClientsExampleTest`

Demo - Apache HTTPClient

see `GsonMarshallingHttpClientsExampleTest`

Custom HTTP Abstractions

- it can be more work to create a Custom HTTP Abstraction
- benefits:
  - no additional dependencies
  - can wrap other HTTP libraries

– no 'library' classes in tests, HTTP Domain objects instead
– allows swapping between HTTP or REST Libraries without impacting test code

---

Example

`package uk.co.compendiumdev.http`

- a crude HTTP library uses `Java.net` classes
- `HttpMessageSender` a wrapper which can send requests
- `CanSendHttpRequests` interface for `getLastRequest` and `send` and `getLastResponse`
- `HttpRequestSender` - actually sends requests (using `java.net`)
- `HttpRequestDetails` represents a logical HTTP Request
- `HttpResponseDetails` represents a logical HTTP Response

'logical' classes can be used in Test, then physical libraries can be replaced or switched between, provided a wrapper around the libary is created which implements `CanSendHttpRequests`

Example Usage

```
HttpMessageSender sender =
        new HttpMessageSender(
                ServerConfig.DEFAULT_SERVER_ROOT);

sender.setHeader(sender.HEADER_ACCEPT, "application/json");
sender.setBasicAuth("admin", "password");

// GET the lists
HttpResponseDetails getLists =
                sender.get(rootUrl + "/lists");
Assert.assertEquals(200, getLists.statusCode);

final AListsCollectionResponse lists =
         new Gson().fromJson(
                getLists.body,
                AListsCollectionResponse.class);

int currentNumberOfLists = lists.lists.size();
```

Example Usage

```
final AListPayload payloadObject =
                   new AListPayload("my list title");
final String payload = new Gson().toJson(payloadObject);
final HttpResponseDetails response =
               sender.post(rootUrl + "/lists", payload);
Assert.assertEquals(201, response.statusCode);

String header = response.getHeaders().get("Location");

payloadObject.guid  = header.replace("/lists/","");
```

Demo - Custom Http Client using java.net

see `GsonMarshallingCustomHttpExampleTest`

Discuss Pros and Cons of Http Level Abstractions

Discuss

Exercise

- Create a constructor for `HttpMessageSender` which takes a `CanSendHttpRequests` to allow switching between HTTP transport libraries
- Create an implementation of `CanSendHttpRequests` which uses RestAssured as the implementor rather than java.net
- Create an implementation of `CanSendHttpRequests` which uses Apache `HttpClient` as the implementor rather than java.net

---

SECTION - ABSTRACTIONS

Dijkstra - On Abstractions

> The purpose of abstraction is not to be vague, but to create a new semantic level in which one can be absolutely precise

- Edsger Dijkstra (as part of his ACM Turing Lecture on 1972: the Humble Programmer
- cs.utexas.edu/~EWD/transcriptions/EWD03xx/EWD340.html

Any Abstractions Here?

```
@Test
public void options(){
    RestAssured.
        given().
            auth().preemptive().
            basic("admin", "password").
        when().
            options("http:4567/listicator/lists").
        then().
        assertThat().
            statusCode(200).
            header("Allow",
            "GET, POST, PUT");
}
```

Abstractions Present Were

- wrapper around HTTP messaging
- Assertion abstractions
- Given/When/Then

No Abstractions for:

- our Domain
- our API
- our Environments

Writing at the REST Assured Semantic Level

We want to:

- write at the level of the test
- our domain
- our business language
- our physical model

To make it:

- easy to maintain
- code completion to help writing
- easy to read and understand

REST Assured

- is a very capable library
- has a lot of advanced functionality (see docs)
- has added features for 'reuse' e.g. reusing validation rules, static configuration
- is very easy to start using
- great for Automating Tactically
- can impact your ability to create effective abstractions if you use all its features at the `@Test` level

Strategically we might want a mix of libraries and abstractions.

What Abstractions might we want?

- What might we want?
- What have you used before?

Discuss

Possible Abstractions

- Request Payloads
- Response Payloads
- Environment/Server classes
- User - representation, actions, workflows

- Listicator Api
  - e.g. `getLists()`, `getList(aGUID)`,
- HTTP Abstractions - remove dependency on RESTAssured
- DSL for making testing easier?

Example Abstraction - Constants

- Constants for server urls
- Variable names

```
public class ServerConfig {

    public static final String DEFAULT_SERVER_ROOT =
            "http://localhost:4567/listicator";

    // public version on heroku - currently different version
    //public static final String DEFAULT_SERVER_ROOT
    //        = "http://rest-list-system.herokuapp.com";

}
```

- *This is as simple as you can get, but needs code changes to configure*
- *would probably want to wrap in a class with constructor to set by environment defaults etc.*

Example Abstraction - Request & Response Payloads

Class to serialise/deserialise responses and request payloads.

```
package uk.co.compendiumdev.restlisticator.payloads;

import javax.xml.bind.annotation.XmlRootElement;
@XmlRootElement(name="list")
public class ListPayload {
    private String title;
    private String guid;
    private String description;

    public String getGuid() { return guid;}
    public String getTitle() { return title; }
    public String getDescription() {return description; }
    public void setTitle(String title) {this.title = title;}
```

```
    public void setDescription(String description)
             {this.description = description;}
    public void setGuid(String guid) {this.guid = guid;}
}
```

Why Request and Response Payloads?

- easily create/parse responses
- support json/xml
- no more String concatenation etc.
- multiple payload objects for different config
- seperate locial domain from physical messages

Payload Builders

- normal 'builder' pattern to make payload creation easier

```
public class ListPayloadBuilder {
    private ListPayload listPayload;

    public static ListPayloadBuilder create(){
        return new ListPayloadBuilder();
    }
    public ListPayloadBuilder(){ listPayload =
                  new ListPayload(); }

    public ListPayload build() {return listPayload;}
    public ListPayloadBuilder with() { return this; }
    public ListPayloadBuilder and() { return this; }

    public ListPayloadBuilder title(String title) {
        listPayload.setTitle(title);
        return this;
    }
    // etc.
}
```

Example Abstraction - Environment/Server classes

- make it easy to configure environment to run tests against

```
public class RestListicatorServer {
    private final String host;
    private final int port;
    private String apiroot="";
    private scheme = "http";

    public RestListicatorServer(
                            final String httpHost,
                            final int port,
                            final String apipath) {
        this.host = host;
        this.port = port;
        apiroot = apipath;
    }

    public String getHTTPHost() {
        return String.format("%s://%s:%d/%s",
                    scheme, host, port, apiroot);
    }
}
```

Example Abstraction - User - representation, actions, workflows

- a very high level abstraction to model the user and actions a user can make
- would delegate off to an API abstraction to do the work

```
User user = new User("admin","password");
user.createList("my new list");
```

Example Abstraction - Listicator Api

- e.g. `getLists()`, `getList(aGUID)` make it easier to send to API

```
public class RestListicatorApi {
    // ...

    public RestListicatorApi(RestListicatorServer server){
```

```
        this.server = server;
        this.contentType = CONTENT_IS_JSON;
        this.accept = CONTENT_IS_JSON;
    }

    public Response getLists() {
        return RestAssured.
                given().
                    contentType(contentType).
                    accept(accept).
                when().
                    get(server.getHTTPHost() + "/lists").
                andReturn();
    }
}
```

Example Abstraction - HTTP Abstractions - remove dependency on RESTAssured

- seen previously in "Alternative HTTP" section

Example Abstraction - DSL for making testing easier

e.g. from RestMud Test Automating, make it easier to write test code

```
dsl.walkthroughStep("\n### The Bomb Puzzle\n");

successfully(walkthrough("", "score", ""));
successfullyVisitRoom("2", walkthrough(
            "I will solve the bomb puzzle", "go", "n"));
successfully(walkthrough("", "examine", "smallroundthing"));
successfully(walkthrough("", "take", "smallroundthing"));
successfully(walkthrough(
    "I could just wait, but I'll get points if I defuse it",
        "defuse", "smallroundthing"));
ResultOutput result =
        successfully(walkthrough("", "score", ""));
int score = result.users.get(0).score.intValue();
Assert.assertEquals(100,score);
```

Abstractions are there too. . .

- make it easy to write code, harness code completion
- make it easy to maintain, single responsibility
- write at the appropriate level of abstraction
  - easy to read and understand
  - easy to expand
- avoid clutter (too many methods on one class)
- etc.

  The purpose of abstraction is not to be vague, but to create a new semantic level in which one can be absolutely precise - Edsger Dijkstra

---

Example Abstractions - REST Listicator Automating

- Server abstractions
- Request Payload Abstractions
- Request Payload Factory Abstractions
- Response Payload Abstractions
- JSON & XML parsing with Payload Abstractions
- Static API wrappers around a dynamic API abstraction
- Basic RestAssured Usage

github.com/eviltester/rest-listicator-automating-examples

Exercises - Expand the Abstractions Test

remember

- keep existing code working
- refactor in small steps
- this is a 'real world exercise' - it is not 'easy'
- there are no 'right' answers
- there are no 'final' answers - keep refactoring
- if it works, its probably "good enough"

Exercises - Expand the Abstractions Test

- create a `RestListicatorServer` abstraction
  - to avoid hardcoding `"http://localhost:4567"`

- create an End Points abstraction
  - to avoid hardcoding `"\heartbeat"`, etc.
- separate API calls from Assertions
- perhaps create Http level abstractions as well as API abstractions
- use Payload Objects instead of hardcoded message strings
- perhaps create payload builder objects
- perhaps add random data generators

For 'an answer' see `package`

- **`uk.co.compendiumdev.advancedrestautomating.exercises.abstractions;` in the `code-answers` folder**

Exercises - Improve the custom HTTP Abstraction

- headers on response is current a map that is accessible by anything
  - create a 'getHeader' method on response which is not case sensitive e.g. `getHeader("location")`, `getHeader("Location")` should both return the "Location" header details if present

---

Exercises - Create an Abstraction for the RestListicator API

- Create a `RestListicatorApi` class
- it should have methods like:
  - `getLists()` which returns a `ListCollectionResponse`
  - `getList(guid)` which returns a `ListPayload` for a specific list
  - `amendList(aList)` which takes a `ListPayload` and returns a `ListPayload` representing the changed list
  - the `RestListicatorApi` should store the last response and allow the user to retrieve it e.g. `getLastResponse()` would return an `HttpResponse` object to allow checking status codes etc.

SECTION - COVERAGE

A discussion section.

What does coverage mean in terms of API testing?

- what does coverage mean?
- is coverage important?
- how would you measure it?
- could you automate tracking?
- discuss

Possible Points

- verbs - have all verbs been used?
- endpoints - has each end point been hit? with all verbs?
- query parameters - have all combinations of query params been tried?
  - query params from other end points?
- headers
- authentication approaches
- etc.

Automating Coverage Tracking

In the past I have:

- run all API tests through a proxy
- Browsermob proxy https://bmp.lightbody.net/
- exported a HAR file
- read HAR file and processed it to check:
  - what verbs used?
  - have all verbs been used on all endpoints?
  - etc.
- generate a report

This was all custom code for client projects, no public source code.

SECTION - PERFORMANCE TESTING

A discussion section

Performance Testing

- is performance testing an API important?
- what does it mean?
- what risks are there for APIs?
- what tools have you used?
- how would you performance test an API?

Possible Open Source Tools

- Artillery
  - https://artillery.io/docs/getting-started/
- JMeter
  - https://jmeter.apache.org/
  - https://github.com/flood-io/ruby-jmeter
  - http://gettaurus.org/
- Gatling
  - https://gatling.io/

Artillery Example

- `brew install node`
- `npm install -g artillery`
- To run through a proxy

```
HTTP_PROXY='http://localhost:8080' artillery run example.yml
```

- to generate output

```
artillery run example.yml -o output.json
```

- to generate a report

```
artillery report output.json
```

Artillery Example - example.yml

```
config:
  target: 'http://localhost:4567/listicator'
  phases:
    - duration: 20
```

```
      arrivalRate: 10
  processor: "./javascript-helpers.js"
scenarios:
  - flow:
    - get:
        url: "/lists"
    - function: "millisTitle"
    - post: # create a list
        headers:
          Authorization: "Basic YWRtaW46cGFzc3dvcmQ="
        url: "/lists"
        json:
          title: "{{ title }}"
        capture:
          - header: "location"
            as: "newlistguid"
    - get:
        url: "{{ newlistguid }}"
        ifTrue: "newlistguid"
```

JavaScript Helpers .js

```
module.exports = {
    millisTitle: millisTitle
  }

function millisTitle(context, events, done){
    context.vars['title'] =  'my title ' +
                             new Date().valueOf();
    return done();
}
```

Model Based Testing Approaches - Bots

- custom approach
- can potentially be performed in other tools when you understand how they work
- for RestMud I created player bots
  - implement different play strategies
  - maintain their own state
  - run in separate threads

- I use for exploratory multi user testing

see github.com/eviltester/restmudbots

A Simple Model Based Test Bot

```
@Test
public void createAMultiUserRestMudTestBot(){

    RestMudTestBot myFirstBot = new RestMudTestBot(
                "http://restmud.herokuapp.com")
            .setUserName("zzzamd64Windows811488121657822");
    myFirstBot.needsToRegisterOnSystem(true);
    myFirstBot.setRegistrationSecretCode("secretcode");
    myFirstBot.setPassword("password");
    myFirstBot.register();
    myFirstBot.login();

// ...
```

A Simple Model Based Test Bot Continued

```
    myFirstBot.addActionStrategy(
            new WalkerStrategy().canOpenDoors(true));
    myFirstBot.addActionStrategy(new MadHoarderStrategy());
    myFirstBot.addActionStrategy(new RandomTakerStrategy());
    myFirstBot.addActionStrategy(new RandomUseStrategy());
    myFirstBot.addActionStrategy(
            new AllDoorOpenerStrategy());

    myFirstBot.addWaitingStrategy(
            new RandomWaitStrategy().waitTimeBetween(0,100));

    for(int x=0;x<2000;x++) {
        myFirstBot.executeARandomActionStrategy();
        myFirstBot.waitAWhile();
    }
}
```

Code Structure Walkthrough

- `RestMudTestBot` - basic object that represents a player
- `ThreadedRestMudTestBot` - represents a player as a thread
- Bots use strategies which are either
  - Action strategies - `RestMudBotStrategy`
  - Waiting strategies - `RestMudBotWaitingStrategy`
- A Simple API abstraction `RestMudApi`
- A generic command abstraction `CommandExecutor`
- I used also for testing end point case testing with the `ChangeCaseifier`

see github.com/eviltester/restmudbots

Performance / Stress Testing Recap

- Can be a very 'static' process
- Strategic Performance Testing can be hard to maintain
- Code Based DSLs:
  - easy to version control and maintain
  - support tactical testing
- Performance Testing can be exploratory
- Continuous Testing should limit variation to allow comparison between runs
- Bot Modelling - custom, flexible

Suggested Exercises

- Experiment with Artillery
  - install node
  - install Artillery
  - try the script in the `performance/artillery` folder
  - run it through a proxy
  - expand the script, create a new script
    - make more calls
    - create new users
    - etc.
- Examine the bots in github.com/eviltester/restmudbots

SECTION - REST MUD Challenge

RestMud is a Browser Based Text Adventure Game

A normal text adventure game would look like:

```
> look
You Look: There is an exit to the east
and a torch on the ground.
> take torch
You take the torch.
> go e
You go east. It is too dark to see.
> illuminate
You switch on the torch and can
see exits to the west and east.
```

A RestMud session would look like:

```
> GET /player/bob/look
You look
> GET /player/bob/take/torch
You took: torch. You now have the A Torch
> GET /player/bob/go/e
You Go East
> GET /player/bob/illuminate
Good work. You illuminated the 'A Torch
```

/user/verb/noun

Verbs:	Nouns
- illuminate, darken	
- go, open, close	e, w, n, s
- inventory	
- hoard, polish, use	*itemid*
- say	*something*
- take, drop	*itemid*
- help	

There may be other verbs

Hoard

When you see your 'hoard' in the location you can hoard treasure to score points e.g.

`/player/bob/hoard/_treasureid_`

Some treasures are 'junk' and score -ve points

- `/score`
- `/scores`

Polish things for more points when hoarding.

- `/inspect/_itemid_` to find out what it scores - but this costs points

Tips

- use dev tools and inspect the code
- make a map (the wiz might turn off the lights)
- `hoard` to score points

Demo of GUI

- start game
  - http://http://restmud.herokuapp.com
- register or login
  - http://http://restmud.herokuapp.com/register
  - http://http://restmud.herokuapp.com/login
- use GUI to 'click' on verbs
- look at the URL
- inspect the DOM
  - look for `span` with IDs
- change URL directly when no link on screen

API

The 'Rest' in RestMud means it has a 'REST' interface.

- `GET`
- `POST`

API GET Requests

`GET`

- `http://_host_/api/player/_username_/_verb_/_noun_`

e.g.

```
GET http://localhost:4567/api/player/user/messages
GET http://localhost:4567/api/player/user/take/crown_1
```

API POST Requests

`POST`

- http://*host*/api/player/*username*
  - body
  - `{"verb":"inspect", "nounphrase":"cloth_2"}`
  - `{"verb":"look"}`

e.g.

```
POST http://localhost:4567/api/player/user
```

API Register

`POST Register`

- http://*host*/api/register
  - body
  - `{"username":"user", "displayname":"Mighty User", "password":"password"}`
  - **if game requires a 'secret code'**
    - add field `secret`

e.g. ~~~~~~~~ POST http://localhost:4567/api/register {"username":"user", "displayname":"Mighty User", "password":"password", "secret":"iwejwjwjw"} ~~~~~~~~

- session cookie automatically logged in for user after registering

---

API Register Response

```
{
  "status": "success",
  "data": {
    "X-RESTMUD-USER-AUTH":
         "be75b9a3-2b71-4b28-a92b-0295da34c814",
    "successMessage":
         "Registered user user as 'Mighty User'",
    "username": "user"
   }
}
```

- can add a custom header `"X-RESTMUD-USER-AUTH"` with GUID above as security to not require login

---

API Login

`POST Login`

- http://*host*/api/login
  - body
  - `{"username":"user", "password":"password"}`

e.g.

`POST http://localhost:4567/api/login`

API Login Response

```
{
  "status": "success",
  "data": {
    "resultoutput": {
      "lastactionstate": "success",
      "lastactionresult": "user bob2 is now logged in"
    },
    "userDetails": {
      "displayName": "Mighty User",
      "userName": "user",
```

```
      "authToken": "be75b9a3-2b71-4b28-a92b-0295da34c814"
    }
  }
}
```

- login response has a cookie that can be used for authentication

---

Message Authentication

- send the `JSESSIONID` cookie set on register or login or from GUI Session
- send the `X-RESTMUD-USER-AUTH` custom header in the HTTP request
- use basic auth with registerd username and password

Recommended API Usage Tool

I use postman for most of my interactive REST API usage.

- https://www.getpostman.com

API Demo

- login
- look
- take stuff
- etc.

see also github.com/eviltester/restmudbots

Overview of API for REST Mud API Challenge

- download github.com/eviltester/restmudbots
- open as project in IntelliJ
- `RestMudAPI` is a simple API abstraction for the game API
  - this could be used directly to automate the API
  - see `RestRoomsTest` as an example
- Challenge - if you choose to accept it
  - write a ChallengeBot that achieves a high score
  - `CreateAHighScoreBot`

Resources to learn from Alan Richardson

Github code samples using REST Assured

- https://github.com/eviltester/rest-listicator-automating-examples
- https://github.com/eviltester/tracksrestcasestudy
- https://github.com/eviltester/libraryexamples

Using JSoup

- https://github.com/eviltester/restmudbots

Automating and Testing a REST API

- support page (videos) - https://www.compendiumdev.co.uk/page/tracksrestsupport
- book - https://compendiumdev.co.uk/pag/tracksrestapibook

Resources to learn from Mark Winteringham

http://www.mwtestconsultancy.co.uk/

Mark Winteringham has some useful study material on REST and automating Web Services.

- https://github.com/mwinteringham/api-framework
  - code in different languages and frameworks demonstrating REST API automated execution
- https://github.com/mwinteringham/restful-booker
  - Test Web API
  - live at https://restful-booker.herokuapp.com/
- https://github.com/mwinteringham/presentations
  - Mark's REST Presentations

Bas Dijkstra & James Willett

Resources to learn from Bas Dijkstra

- https://github.com/basdijkstra/rest-assured-workshop/
  - API REST Assured Code and slides

- http://www.ontestautomation.com/open-source-workshops/
  - Info on Bas's open source workshops
- http://www.ontestautomation.com/category/api-testing/
  - Bas's Blog posts on API Testing

Resources to learn from James Willett

- https://james-willett.com/tag/rest-assured/
  - blog posts on REST Assured

Recommended Books

- "Automating and Testing a REST API" by Alan Richardson
- "Web Application Hacker's Handbook" by Dafydd Stuttard, Marcus Pinto

---

Recommended Libraries and Tools

- https://insomnia.rest/ Insomnia REST Client
- https://www.getpostman.com/ Postman REST Client
- https://portswigger.net/burp/ BurpSuite Proxy
- https://www.zaproxy.org/ OWasp ZAP Proxy
- http://rest-assured.io/ REST API Automating library
- https://jsoup.org/ Simple HTML parser and HTTP library
- https://artillery.io/ Lightweight performance testing
- https://github.com/google/gson JSON serialiser
- http://x-stream.github.io/ XML serialiser
- https://jmeter.apache.org/ JMeter
- https://gatling.io/ Gatling

About Alan Richardson

www.compendiumdev.co.uk

Blogs: javafortesters.com, seleniumsimplified.com, eviltester.com, testerhq.com

- Linkedin - [@eviltester](https://uk.linkedin.com/in/eviltester)
- Twitter - [@eviltester](https://twitter.com/eviltester)
- Instagram - [@eviltester](https://www.instagram.com/eviltester)

- Facebook - [@eviltester](https://facebook.com/eviltester/)
- Youtube - EvilTesterVideos
- Pinterest - [@eviltester](https://uk.pinterest.com/eviltester/)
- Github - [@eviltester](https://github.com/eviltester/)
- Slideshare - [@eviltester](https://www.slideshare.net/eviltester)

SECTION - HTTP Proxies Reference

Which proxies?

- Fiddler
  - Windows (Beta: Linux, Mac)
- Charles
  - Commercial but allows 30 mins in 'shareware' mode
- BurpSuite
  - Free edition good enough for API Testing
- Owasp ZAP
  - Open Source

Fiddler & Charles act as System Proxies making them easy to use with Postman.

Using a Proxy with Postman

- Change Proxy Settings with "File Settings" and then "Proxy" tab
  - on Mac use "Postman Preferences" and then "Proxy" tab
- Postman can use a Global Proxy by setting the IP address and Port
  - e.g. BurpSuite, OWasp Zap (or Fiddler and Charles)
- Postman can hook into System Proxy e.g
  - Charles, Fiddler
- Otherwise start postman with `--proxy-server`

Using a Proxy with Postman

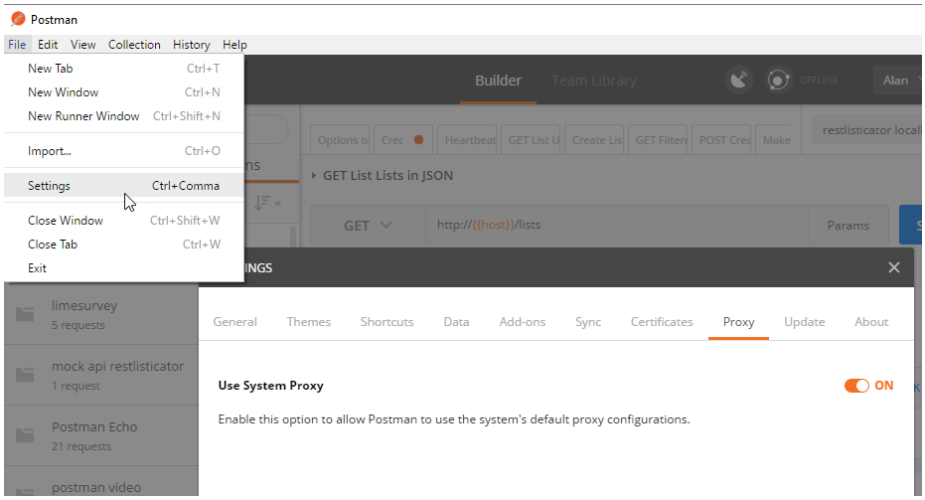

Figure 9: Using a Proxy with Postman

Setting Postman proxy from commandline

For full details see blog post

- Mac (type all on one line):

```
open /Applications/Postman.app --args
     --proxy-server=localhost:8888
```

- Windows:

```
cd C:\Users\Alan\AppData\Local\Postman\app-4.9.3\
postman.exe --proxy-server=localhost:8888
```

Using a Proxy with Insomnia

- Application Preferences
- on Mac "Insomnia preferences"

Use of Proxies

- Examples of Fiddler - with screenshots
- Examples of Charles with screenshots
- Examples of Owasp Zap with screenshots

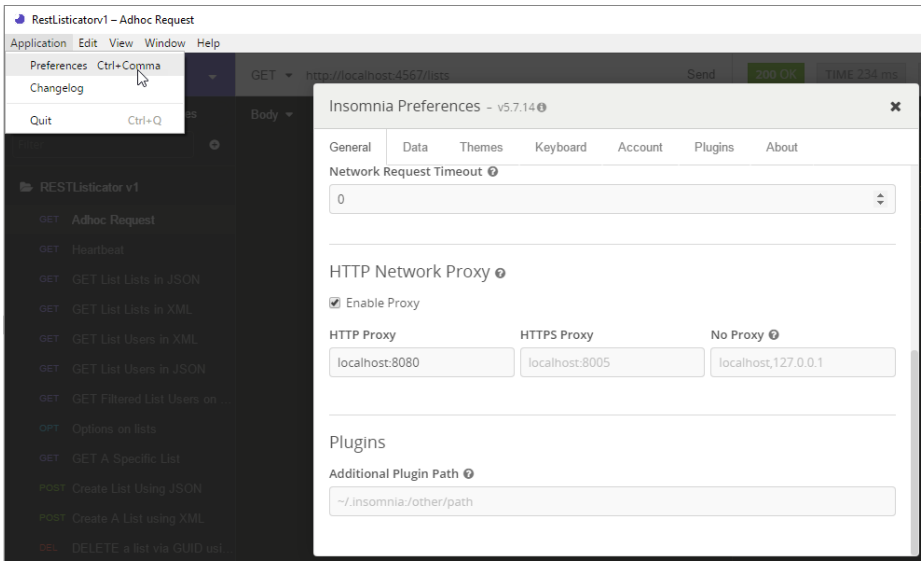

Figure 10: Using a Proxy with Insomnia

- Examples of BurpSuite with screenshots

Fiddler Filter Requests

- ctrl+X - to clear traffic history
- filter by process

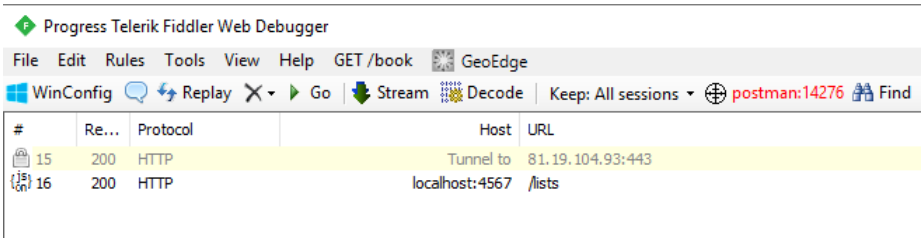

Figure 11: Fiddler Filter Requests

Fiddler Inspect Traffic

- Traffic shown in list - use inspectors to view request and response

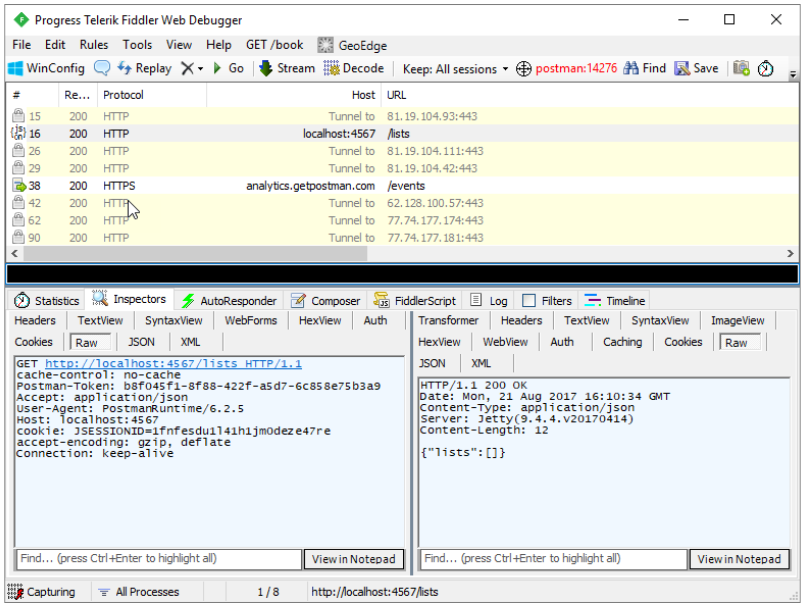

Figure 12: Fiddler Inspect Traffic

Fiddler listens on port 8888 by default

- find out/change port in `tools \ options`

Replay Request in Fiddler

- drag request from history to Composer to edit and replay

Charles Filter Requests

- Quick filter to `localhost`

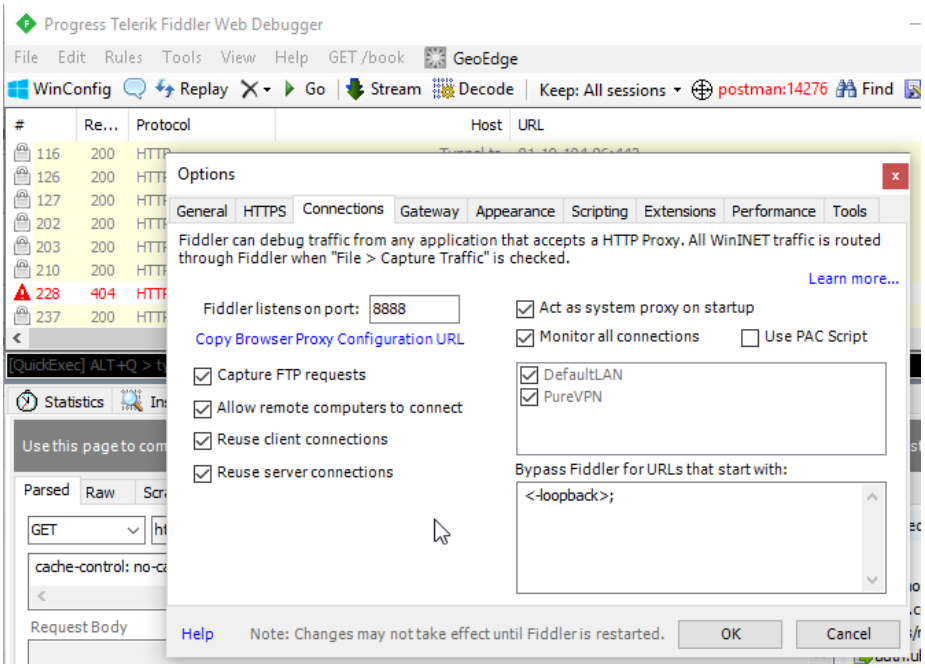

Figure 13: Fiddler listens on port 8888 by default

Charles Inspect Traffic

- inspect traffic

Charles Replay Traffic

- right click 'compose'

Charles port settings

- `tools \ proxy settings`

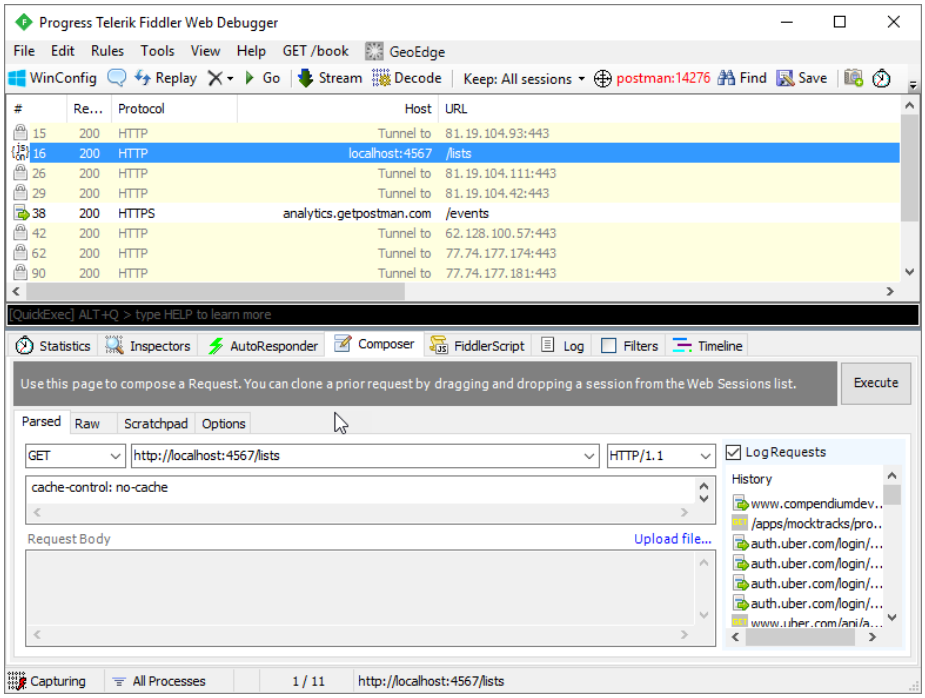

Figure 14: Replay Request in Fiddler

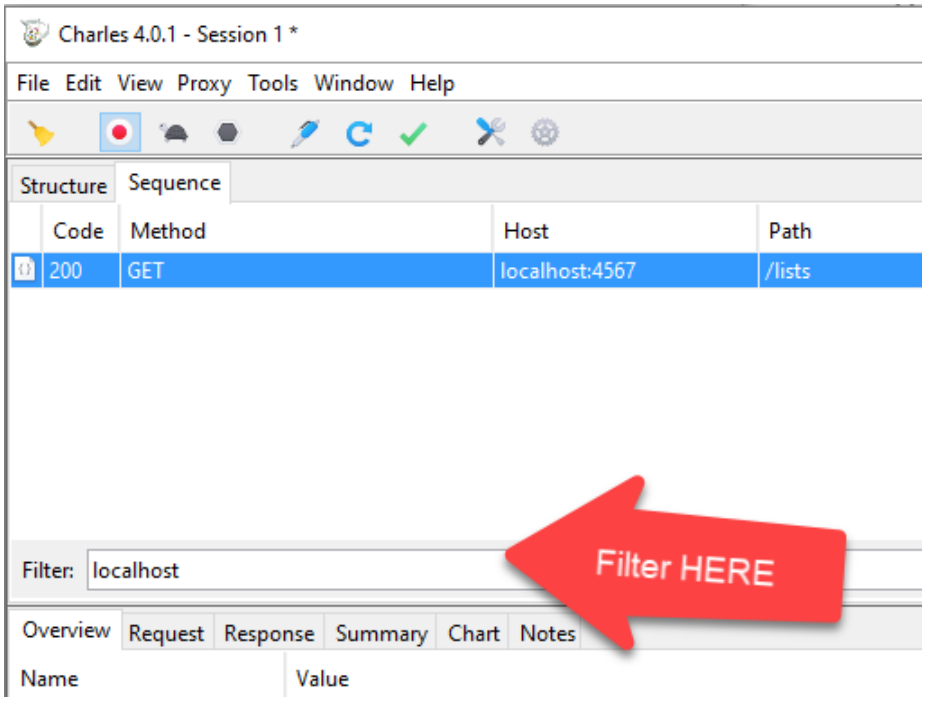

Figure 15: Charles Filter Requests

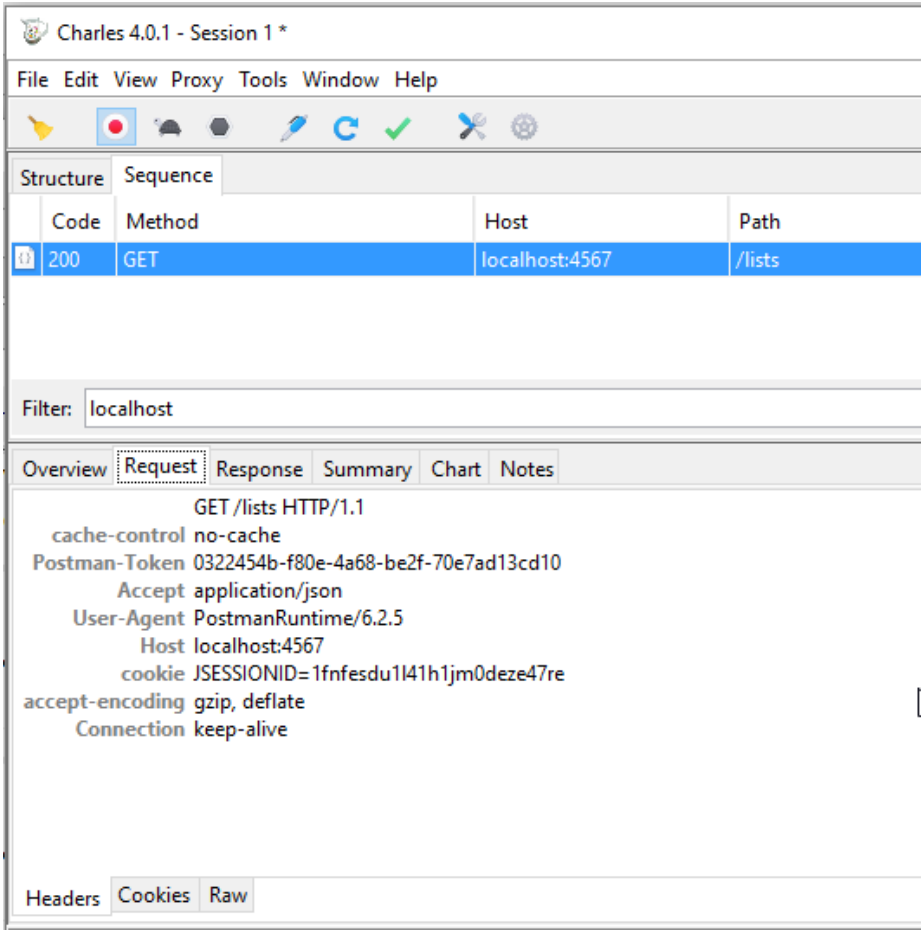

Figure 16: Charles Inspect Traffic

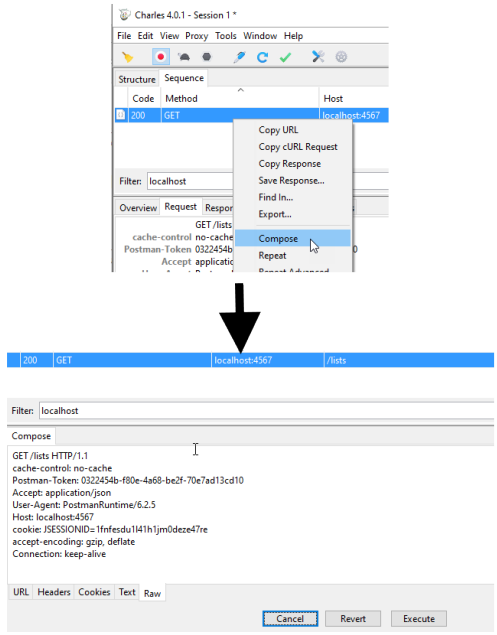

Figure 17: Charles Replay Traffic

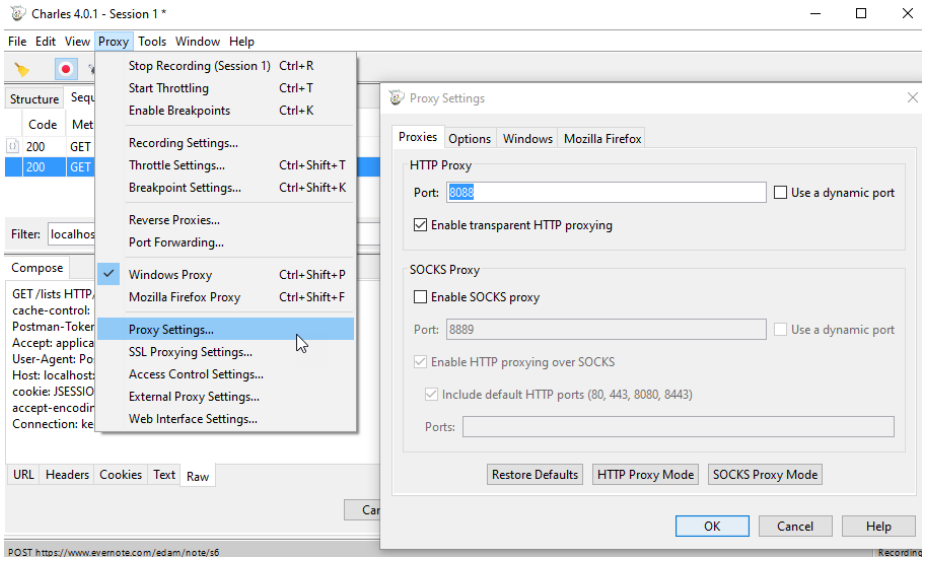

Figure 18: Charles port settings

BurpSuite Port Config

- tabs `proxy \ options`

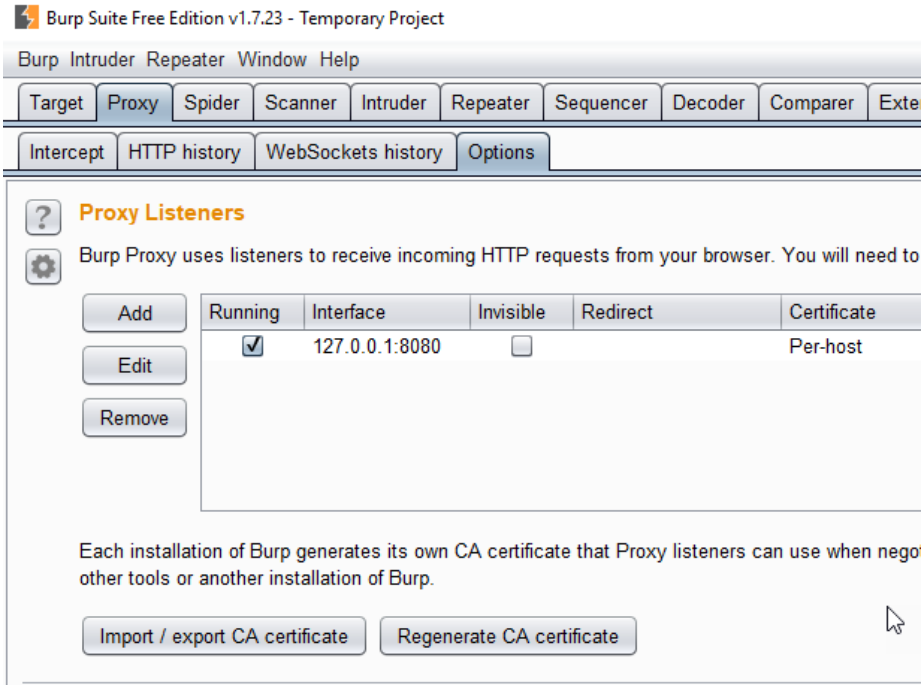

Figure 19: BurpSuite Port Config

BurpSuite Inspect Traffic

- Ensure intecept is off, view HTTP History

BurpSuite Replay Request

- right click and use repeater

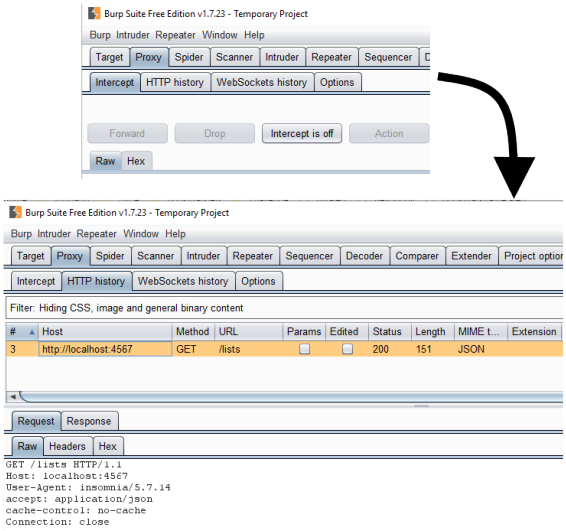

Figure 20: BurpSuite Inspect Traffic

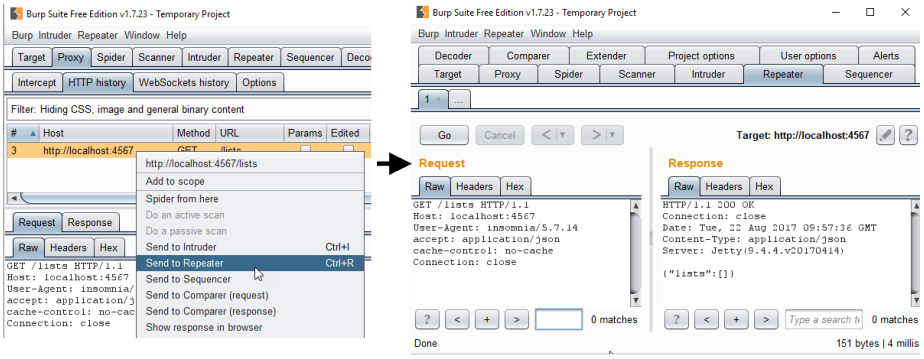

Figure 21: BurpSuite Replay Request

Owasp Zap Port Config

- `tools \ options \ local proxy`

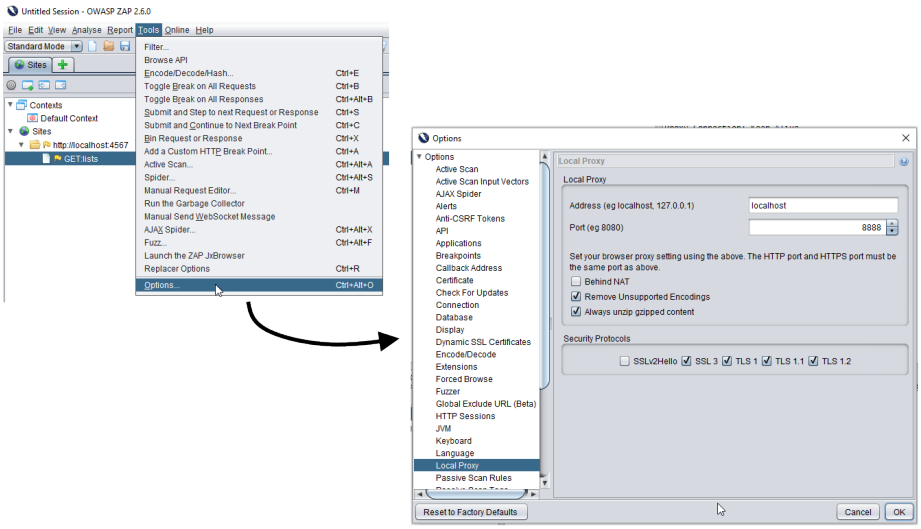

Figure 22: Owasp Zap Port Config

Owasp Zap Inspect Traffic

- history and 'sites', view in top right

Owasp Zap Replay Request

- right click 'Resend', edit and send

Fuzzing in BurpSuite

- right click, intruder, hightlight and add position, edit payload, start attack

---

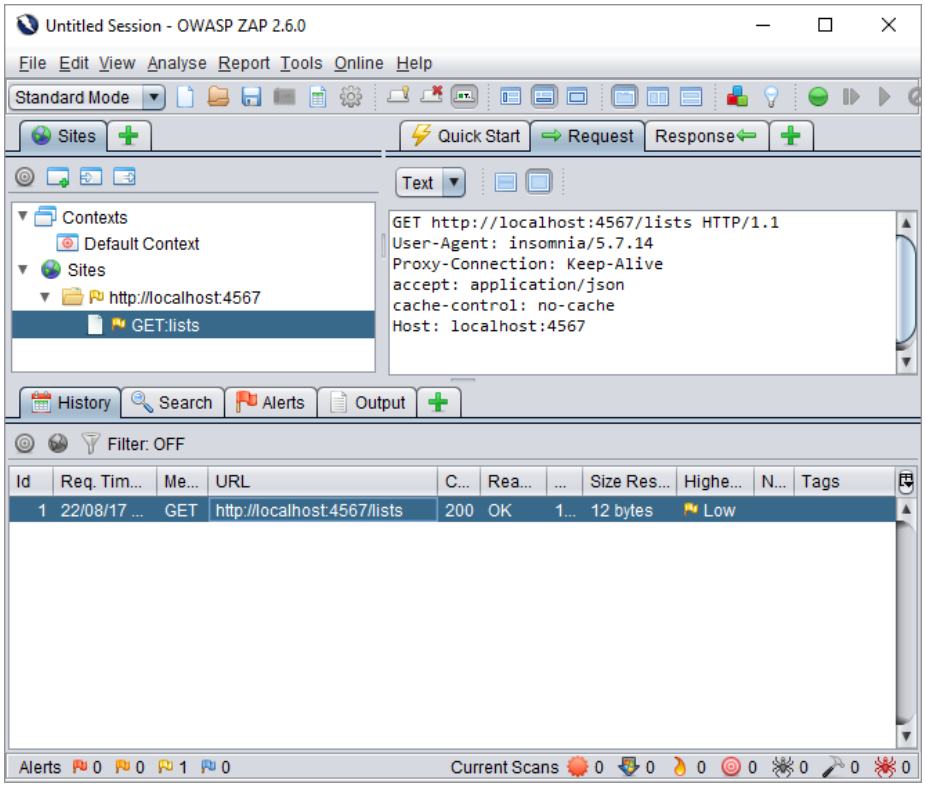

Figure 23: Owasp Zap Inspect Traffic

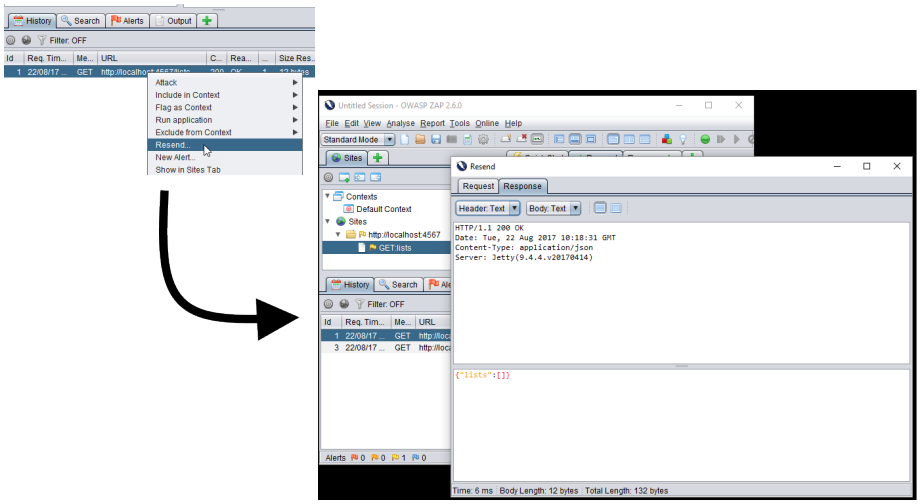

Figure 24: Owasp Zap Replay Request

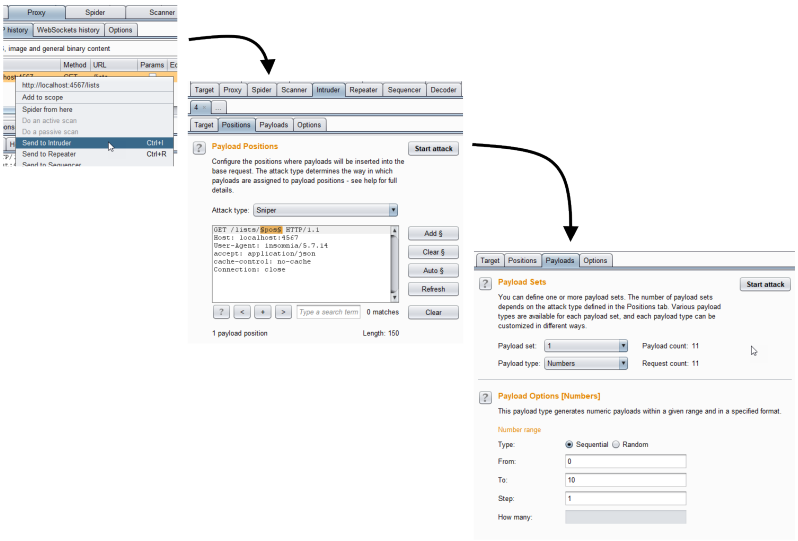

Figure 25: Fuzzing in BurpSuite

Fuzzing in Owasp ZAP

- right click, attack, fuzz, highlight and add location, add payload, start fuzzer

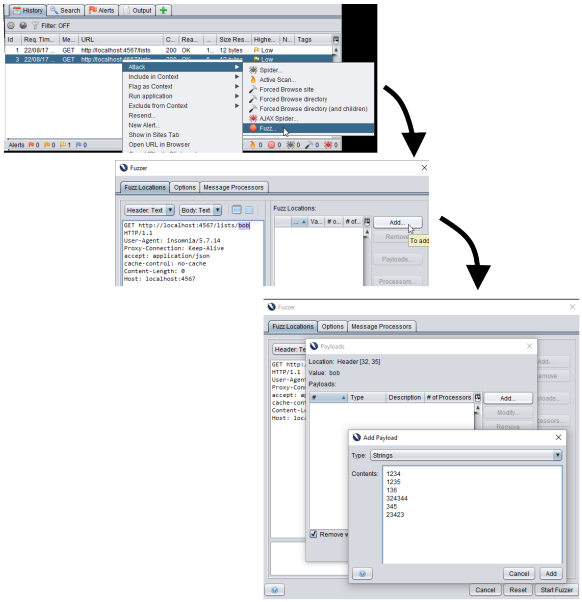

Figure 26: Fuzzing in Owasp ZAP