Contents

SECTION - WEB API BASICS

Overview of Section - WEB API Basics

- What is a Web API?
- Ajax/XHR and API
- Why Web HTTP API?
- Exercises using http://swapi.co

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

Exercise: A Web Application which uses API

- https://swapi.co
  - make a request from the GUI
  - Use network tab

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

Demo cURL

```
curl http://localhost:4567/heartbeat -i
curl -X GET http://localhost:4567/users
curl http://localhost:4567/lists -H "accept: application/xml"
```

Can be complicated but useful for emergencies, scripting, bug reporting.

Hint: can use Postman or Insomnia to generate cURL code but different continuation characters on different operating systems: `^` *Windows and* `\` *on Mac/Linux also* `"` *and* `'` *differences.*

Demo Postman

- Postman make GET request
- Postman console
- Postman set basic auth
- Postman add a header
- Postman Collections
- Postman Environment Variables

Demo Insomnia

- Insomnia make GET request
- Insomnia Timeline
- Insomnia set basic auth
- Insomnia add a header
- Insomnia Workspace
- Insomnia Environment Variables

Exercise: Install Tools for accessing HTTP API Web Services

Install either:

- Postman GetPostman.com
- Insomnia insomnia.rest

Exercise: Call a webservice using tools

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

'MOCK' Web Services

- GET http://compendiumdev.co.uk/apps/api/mock/heartbeat
- GET http://jsonplaceholder.typicode.com/users/1/todos

see exercises section for more

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

Why test interactively and not just automate?

- observe traffic
- create varied requests
- experiment fast
- setup data
- send 'invalid' requests
- exploratory testing of API
- test while API still 'flexible'
- Interactive CRUD testing - CREATE, READ, UPDATE, DELETE

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

CRUD Action	Verb
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.

Verb - Head

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

---

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`

Postman Collection Runner

- send multiple requests iterating over data files
- runs all requests in a collection
- create requests with params in body or query
- put data in a CSV file
- run collection, with environment, with data

Example Request With Params

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<lists>
    <list>
        <title>{{title}}</title>
        <description>{{description}}</description>
    </list>
</lists>
```

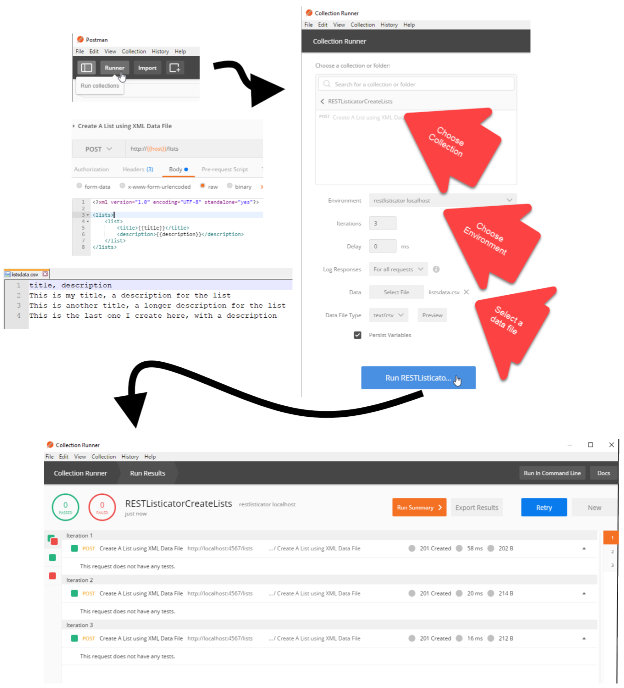
CSV file with params ~~~~~~~~ title, description This is my title, a description for the list This is another title, a longer description for the list This is the last one I create here, with a description ~~~~~~~~

Postman Collection Runner example

Exercises

Reading:

- read the REST Dissertation ics.uci.edu/~fielding/pubs/dissertation/top.htm
- Read the docs on authentication developer.mozilla.org/en-US/docs/Web/HTTP/Authentication



```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<lists>
    <list>
        <title>{{title}}</title>
        <description>{{description}}</description>
    </list>
</lists>
```

listsdata.csv

```
title, description
This is my title, a description for the list
This is another title, a longer description for the list
This is the last one I create here, with a description
```

Collection Runner

Choose a collection or folder: RESTListicatorCreateLists

POST Create A List using XML Data File

Environment: restlisticator localhost
Iterations: 3
Delay: 0 ms
Log Responses: For all requests
Data: listsdata.csv
Data File Type: text/csv
Persist Variables ✓

Collection Runner — Run Results

RESTListicatorCreateLists restlisticator localhost
just now
0 PASSED 0 FAILED

```
Iteration 1
POST  Create A List using XML Data File  http://localhost:4567/lists  .../ Create A List using XML Data File  201 Created  58 ms  202 B
This request does not have any tests.
Iteration 2
POST  Create A List using XML Data File  http://localhost:4567/lists  .../ Create A List using XML Data File  201 Created  20 ms  214 B
This request does not have any tests.
Iteration 3
POST  Create A List using XML Data File  http://localhost:4567/lists  .../ Create A List using XML Data File  201 Created  16 ms  212 B
This request does not have any tests.
```

Figure 1: Postman Collection Runner example

- for real world ‘discussions’ see restcookbook.com

Exercises

Doing:

- Experiment with `HEAD` and `PATCH`
- Continue to experiment with the other verbs and test the Web Service
- Create a Postman Collection to use in the runner which creates 10 new list entities
- for more exercises see the exercise section

SECTION - Testing a REST API

Overview of Section - Testing a REST API

- how to model an API
- testing ideas
- interactive discussion

Testing different from Technology and Tooling

- at this point we have discussed technology and tooling
- time to discuss testing

What would we test?

- ideas?

What are the architecture risks?

- Client -> Web Server -> App Server -> App
- Do we understand the architecture?

What are the capacity risks?

- Performance?
- Load Testing?

What are the security risks?

- Authentication
- Authorisation
- Injection

Data Risks

- minimum data in requests - missing fields, headers
- not enough data in requests
- wrong format data: json, xml, length, null, empty
- malformed data
- consistency? query params across requests?
- are defaults correct?
- duplicate data in payloads?
- headers: missing, malformed, too many, duplicate

Document your testing

- How can you document your testing?

Other Risks or Common Issues?

Exercise: Think through testing

- Read the requirements etc. for REST Listicator.
- Create some test ideas
- Look at the existing testing conducted
- Any ideas from that?
- Test REST Listicator
- Document and Track your Testing in a lightweight fashion

Exercise: Test REST Listicator in Buggy mode

```
java -jar rest-list-system.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 - AUTOMATING

Overview of Section -Automating

- Automating
- REST Listicator Example Automating code
- Abstraction Layers
- REST Assured
- Resources to learn from

Why Automate?

- repeatability
- speed
- data coverage
- deployment validation
- support exploratory testing

How?

- Postman?
- HTTP Libraries?
- REST Libraries?
- Which language?
- Other tools?

Examples Using Java and REST Assured

- https://github.com/rest-assured/rest-assured
- Java/Groovy library
- HTTP Abstraction
- Marshalling - Serialization/Deserialization
- Assertions

Basic GET Request

```
@Test
public void canCheckThatServerIsRunning(){

    RestListicatorServer server =
        new RestListicatorServer("localhost",4567);

    RestAssured.
```

```
            get(server.getHTTPHost() + "/heartbeat").
            then().assertThat().
            statusCode(200);
}
```

Payload Objects

```
@XmlRootElement(name="list")
public class ListPayload {

    private String title;
    private String guid;
    private String description;

    public String getGuid() {
        return guid;
    }
    public void setGuid(String guid) {
        this.guid = guid;
    }
  ...
}
```

REST Assured

- uses the `content-type` header to (de)serialize to JSON or XML

`contentType("application/xml")`

`contentType("application/json")`

Marshalling / Serializing

```
public Response createList(ApiUser user, ListPayload list) {
   return RestAssured.
          given().
             contentType(contentType).
```

```
              accept(accept).
              auth().preemptive().
              basic(user.getUsername(), user.getPassword()).
              body(list).
            when().
                post(server.getHTTPHost() + "/lists").
            andReturn();
}
```

Code Walkthrough of REST Listicator Automating Examples

- https://github.com/eviltester/rest-listicator-automating-examples
- code built to show refactoring steps e.g. ListCreationTest
- refactor to abstraction layers
- payload objects could be public fields but that is more vulnerable to app changes
  - xml & json annotations
- api method naming (`createList`) - would be better as `postList` - why?

Code Walkthrough of REST Listicator Automating Examples

- static api vs instantiated api e.g. `ListicatorAPI` singleton
  - readability vs flexibility
- Abstractions can restrict coverage as well as aid it
  - review abstractions to see what is not, and can not be tested with that abstraction code

---

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

- http://www.ontestautomation.com/open-source-workshops/
  - API REST Assured Code and slides
- 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

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

Automating and Testing a REST API

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

Exercises

- read the resources and learn more about automation and REST Assured
- if you have JDK and Java IDE then download the source for REST Listicator Automating Examples
- run the tests
- add more tests to cover the REST Listicator documentation e.g. users, api keys for authentication, post multiple lists, url parameters etc.
- refactor code as you go to build abstraction layers
- rename existing api methods to match verbs rather than logic
- see exercises section for more ideas

SECTION - SUMMARY with Q & A

Overview of Section - SUMMARY with Q & A

- Any Questions?
- Slide based summary of content
- Final Q&A
- Continue to Experiment

Technology

- Learn HTTP Standards
- You can base your 'bugs' on Standards
  - HTTP Message Syntax and Routing RFC 7230
- Learn the common VERBS: GET, POST, DELETE, PUT
- Read the REST Dissertation

Tools - Clients

- Different tools have different capabilites

- Experiment with multiple tools
- Postman: Collections for Data Creation, Console
- Insomnia: Import, Timeline, Proxies
- Import/Export between Tools

Tools - Proxies

- Often used for Security Testing
- Fuzzers create data
- Automatically keep a record of your testing
- View actual requests and responses
- Replay requests

Tools

- Clients
  - Postman
  - Insomnia
  - cURL
- Proxies
  - System
    - Fiddler
    - Charles
  - Other
    - BurpSuite
    - Owasp Zap

Automating

- HTTP libraries
- REST libraries
- Domain Abstractions
- Reuse for performance testing

Testing

- Requirements - domain, documentation, sdk
- Standards - HTTP, REST, Auth
- Security
- Capacity
- Interfacing Systems

Q & A
