Automating & Testing a REST API

A case-study using : Java, REST Assured, Postman, Tracks, cURL and HTTP Proxies

Alan Richardson

Automating and Testing a REST API

A Case Study in API testing using: Java, REST Assured, Postman, Tracks, cURL and HTTP Proxies

Alan Richardson

This book is for sale at http://leanpub.com/testrestapi

This version was published on 2017-08-21

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Contents

Thanks for Reading This Sample

Thanks for reading this sample of my "Automating and Testing REST APIs" book. This is an excerpt from the main book.

There should be enough in the sample to provide general introductory information to help you get started with testing REST APIs, specifically the Tracks API.

What you basically have here are chapters 1, 2, 3, 4, 5, 8, 9 and 10. So you get to see:

- Introduction
- How to install tracks
- A GUI tour of tracks
- The tracks API Documentation
- Using a Proxy to view GUI traffic
- Using cURL through a Proxy
- cURL Reference Guide
- Starting to Automate Decisions

What you don't see in here is the actual case study, about 120 pages (or more) of extra information:

- How I used cURL to test the API
- How I used a proxy fuzzer to create users automatically via HTTP
- How I automated creating users with REST Assured even though the REST API doesn't support it
- Creating abstraction layers for REST API testing in Java
- How to use the REST Assured Java/Groovy library for HTTP/REST Testing
- Random Data Creation
- Refactoring and Next Steps

Absolutely masses of valuable and practical content.

But you do have links to all the tools, and libraries and you can view the source code on GitHub so you can see exactly what I will explain in detail in the full book.

I hope you enjoy the sample, and when you are ready to take the next step you can find out more about the full book on my web site.

- compendiumdev.co.uk/page/tracksrestapibook[1]

[1] http://compendiumdev.co.uk/page/tracksrestapibook

Introduction

We can read on-line about the "Test Automation Pyramid" and we can also learn that "GUI Automation is brittle" and "we should test under the GUI". Fine. But how many in-depth examples can you find? Examples that show you how to automate quickly, and how to improve on that initial 'quick fix'?

That's what this case study is all about - an example of automating an application without using GUI based libraries.

This case study will show how you can add value to a process quickly with fairly crude and "hacky" code, and then how you can change that code to make it better.

Throughout the case study, I'm not just going to tell you how I did it. I'm going to explain why, and what I could have done differently. Why I made the decisions I made, because then you can try different approaches and build on this case study.

Since this is a case study, and not a 'step by step' course. I assume some basic knowledge:

- You know how to install Java and an IDE,
  - if you don't then the Starter Page[2] on Java For Testers will help.
- You have some basic Web experience or HTTP knowledge,
  - if not then my Technical Web Testing 101[3] course might help or my YouTube channel[4].

I'll cover some of the above topics, although not in depth. If you get stuck you can use the resources above or contact me[5].

The background behind this case study is that I've used Tracks as an Application Under Test in a few workshops, for practising my own testing, and to improve my ability to automate.

For the workshops I built code to create users and populate the environment with test data. I found that people like to learn how to do that, and I realised during the workshops that I also approach this differently to other people.

[2] http://javafortesters.com/page/install/

[3] http://compendiumdev.co.uk/page.php?title=techweb101course

[4] http://eviltester.com/youtube

[5] http://compendiumdev.co.uk/contact

I didn't automate under the GUI because I follow a "Test Automation Pyramid". I automated beneath the GUI:

- because it is fast,
- because we can do things we can't do through the GUI.

By 'Under the GUI' I mean:

- Using the API (Application Programming Interface).
- Using the 'APP as API',
  - sending through the HTTP that the GUI would have sent, but not using the GUI.

I explain 'App as API' in the case study later, and show examples of it in practice. I realised, during the teaching of this stuff, that most people don't automate in this way.

For most people testing 'under the GUI' means API. To me it means working at the different system communication points anywhere 'under' the GUI. I explain this in the case study as well.

Working under the GUI isn't always easier. In this case study you'll see that working through the GUI would have been 'easier'. I wouldn't have had to manage cookie sessions and scrape data off pages.

But working beneath the GUI is faster, once it is working, and arguably is more robust - but we'll consider that in more detail in the case study and you'll see when it isn't.

You'll see initial code that I used for Tracks 2.2 and then updated for 2.3, I'll walk you through the reasons for the changes and show you the investigation process that I used and changes I made.

If you haven't automated an HTTP application below the GUI before then I think this case study will help you learn a lot, and you'll finish with a stack of ideas to take forward.

If you have automated the GUI before. I think you'll enjoy learning why I made the decisions I made, and you'll be able to compare them with the decisions you've made in the past. I think, after finishing the case study, you might expand the range of decisions you have open to you in the future.

Introduction to APIs

This chapter will provide an introduction to the concept of an API (Application Programming Interface) and concentrates on Web or Internet accessible APIs. I will explain most of the concepts that you need to understand this book and I will start from the very basics. Most of this chapter will be written colloquially for common sense understanding rather than to provide formal computer science definitions.

You can probably skip this chapter if you are familiar with Web Services, URI and URL, HTTP, HTTP Status Codes, JSON and XML.

Also, because this chapter covers a lot of definition, feel free to skip it if you want to get stuck in to the practical aspects. You can always come back to this chapter later if you don't understand some basic terminology.

First I'll start by saying that we are going to learn how to test Web Applications. i.e. Applications that you can access over a network or the internet without installing them on your machine.

The Web Application we will test has an API, which is a way of accessing the application without using a Web Browser or the application's Graphical User Interface (GUI).

What Is a Web Application?

A Web Application is a software application that is deployed to a Web Server. This means the application is installed onto a computer and users access the application via a Web Browser.

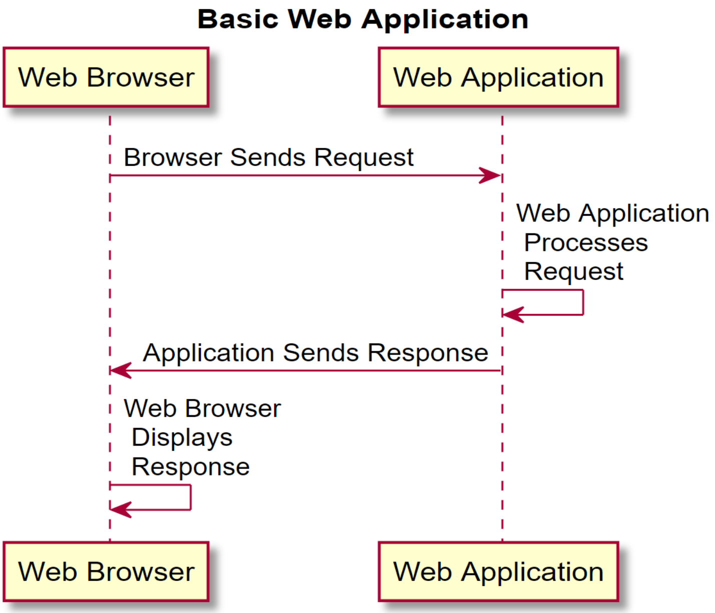
Google Is an Example of a Web Application

`google.com` is an example of a Web Application. A user visits `google.com` in a Browser and the application's Graphical User Interface (GUI) is displayed in the Browser. The GUI consists of a search input field which the user fills in and then clicks a button to search the Web.

When the user clicks the search button, the Browser makes a request to the Web Application on the Web Server to have the Google Search Application make the search and return the results to the user in the form of a web page with clickable links.

Basically,

- Web Browser -> Sends a Request to -> Web Application
- Web Application -> Processes Request and Sends a Web Page as Response to -> Web Browser



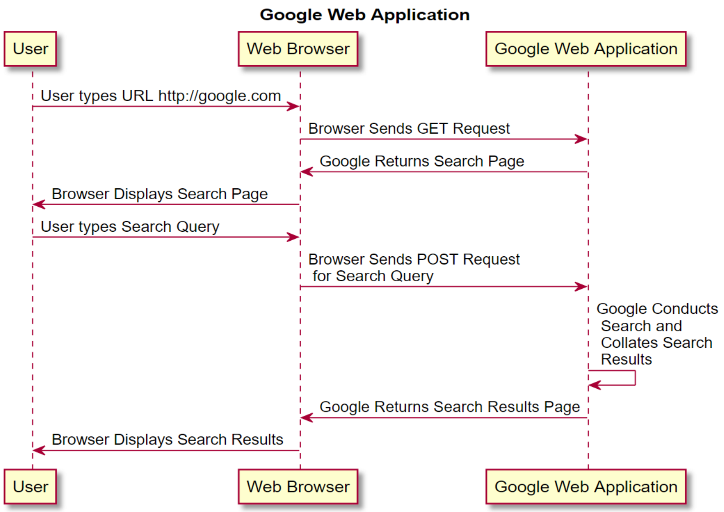
The requests that the Browser sends to the Web Server are HTTP requests. HTTP requests are a way of sending messages between machines over the Internet. Think of HTTP as the format of the message that Browser and Web Server send to each other.

When we first visit Google in a Browser we type in the URL or address for Google. i.e. `https://google.com`

The Browser then sends a type of HTTP request to Google called a `GET` request to 'get', or retrieve, the main search form. Google Web Application receives the request and replies with an HTTP response containing the HTML of the search page. HTML is the specification for the Web Page so the Browser knows how to display it to the user.

When the user types in a search term and presses the search button. The Browser sends a `POST` request to Google. The `POST` request is different from the `GET` request because it contains the details of the search term that the user wants the Google Web Application to search for.

The Google Web Application then responds with an HTTP response that contains the HTML containing all the search results matching the User's search term.


Google Web Application
User
Web Browser
Google Web Application
User types URL http://google.com
Browser Sends GET Request
Google Returns Search Page
Browser Displays Search Page
User types Search Query
Browser Sends POST Request for Search Query
Google Conducts Search and Collates Search Results
Google Returns Search Results Page
Browser Displays Search Results
User
Web Browser
Google Web Application

Google is an example of a Web Application with a GUI, and because the user accesses the Web Application through a Browser they are often unaware of the HTTP requests, or that different types of HTTP requests are being made.

When we test HTTP APIs we have to understand the details of HTTP requests.

What Is an API?

An API is an Application Programming Interface. This is an interface to an application designed for other computer systems to use. As opposed to a Graphical User Interface (GUI) which is designed for humans to use.

APIs come in many different forms with and technical implementations but this book concentrates on HTTP or Web APIs.

An HTTP based API is often called a Web API since they are used to access Web Applications which are deployed to Servers accessible over the Internet.

Applications which are accessed via HTTP APIs are often called Web Services.

Mobile Applications often use Web Services and REST APIs to communicate with servers to implement their functionality. The Mobile Application processes the message returned from the Web Service and displays it to the User in the application GUI. So again, the user is unaware that HTTP requests are being made, or of the format of the requests and responses.

What Is an HTTP Request?

HTTP stands for Hypertext Transfer Protocol and is a way of sending messages to software on another computer over the Internet or over a Network.

An HTTP request is sent to a specific URL and consists of:

- a VERB specifying the type of request e.g. GET, POST, PUT, DELETE
- A set of HTTP Headers. The headers specify information such as the type of Browser, type of content in the message, and what type of response is accepted in return.
- A body, or payload in the request, representing the information sent to, or from, the Web Application. Not all HTTP messages can have payloads: POST and PUT can have payloads, GET and DELETE can not.

HTTP requests are text based messages and over the course of this Case Study you will learn to read them e.g.

```
GET http://compendiumdev.co.uk/apps/mocktracks/projectsjson.php HTTP/1.1
Host: compendiumdev.co.uk
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
 AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/59.0.3071.115 Safari/537.36
Accept: text/html,application/xhtml+xml,
 application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
```

The above HTTP request is a GET request, which is a READ request:

- to read the page from the URL
  - `http://compendiumdev.co.uk/apps/mocktracks/projectsjson.php`
- request is made from the Chrome Browser version 59. You can see this in the 'User-Agent' header. Yes, the header also mentions 'Safari', 'AppleWebKit' and 'Mozilla', this is for various reasons of backwards compatibility, but it was sent from Chrome version 59. For more information on `User-Agent` visit useragentstring.com[6].

[6] http://www.useragentstring.com

What Is a URL?

URL is a Uniform Resource Locator and is the address we use to access websites and web applications.

When working with APIs you will often see this referred to as a URI (Uniform Resource Identifier).

Think of a URI as the generic name for a URL.

When we want to call an HTTP API we need the URL for the `endpoint` we want to call e.g

```
http://compendiumdev.co.uk/apps/mocktracks/projectsjson.php
```

This is the locator that says "I want to call the `apps/mocktracks/projectsjson.php` resource located at `compendiumdev.co.uk` using the `http` protocol".

For the purposes of this book I will use the phrase URL, but you might see URI mentioned in some of the source code. I use URL because the locator contains the protocol or *scheme* required to access it (`http`).

The above URL can be broken down into the form:

```
scheme://host/resource
```

- scheme - `http`
- host - `compendiumdev.co.uk`
- resource - `apps/mocktracks/projectsjson.php`

A larger form for a URL is:

```
scheme://host:port/resource?query#fragment
```

I didn't use a `port` in the URL, for some applications you might need to.

By default `http` uses port `80`, so I could have used:

```
http://compendiumdev.co.uk:80/apps/mocktracks/projectsjson.php
```

Also I haven't used a query because this endpoint doesn't need one.

The `query` is a way of passing parameters in the URL to the endpoint e.g. Google uses query parameters to define the search term and the page:

```
https://www.google.co.uk/?q=test&start=10#q=test
```

- scheme - `https`
- host - `www.google.co.uk`
- query - `q=test&start=10`
- fragment - `q=test`

The query is the set of parameters which are key, value pairs delimited by '&' e.g. `q=test` and `start=10` ("start" is a key, and "10" is the value for that key).

When working with APIs it is mainly the scheme, host, port and query that you will use.

You can learn more about URL and URI online[7].

What Are HTTP Verbs?

A Web Browser will usually make `GET` requests and `POST` requests.

- `GET` requests ask to read information from the server e.g. clicking on a link.
- `POST` requests supply information to the server e.g. submitting a form.

`GET` requests do not have a body, and just consist of the Verb, URL and the Headers.

`POST` requests can have a payload body e.g.

```
POST http://www.compendiumdev.co.uk/apps/mocktracks/reflect.php HTTP/1.1
Host: www.compendiumdev.co.uk
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
 AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/59.0.3071.115 Safari/537.36
Accept: text/html,application/xhtml+xml,
 application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8

{"action":"post"}
```

When working with a Web Application or HTTP API the typical HTTP Verbs used are:

- GET, to read information.

[7] https://en.wikipedia.org/wiki/Uniform_Resource_Identifier

- POST, to create information.
- PUT, to amend or create information.
- DELETE, to delete information, this is rarely used for Browser accessed applications, but often used for HTTP APIs.

POST and PUT requests would usually have a message body. GET and DELETE would not.

HTTP Verbs are described in the W3c Standard[8] and IETF standard[9].

What Is an HTTP Response?

When you issue an HTTP Request to the server you receive an HTTP Response.

The response from the server tells you if your request was successful, or if there was a problem.

```
HTTP/1.1 200 OK
Date: Fri, 30 Jun 2017 13:50:11 GMT
Connection: close
Content-Type: application/json

{
  "projects": {
    "project": [
      {
        "id":  1,
        "name": "A New Project",
        "position": 0,
        "description": "",
        "state": "active",
        "created-at": "2017-06-27T12:25:26+01:00",
        "updated-at": "2017-06-27T12:25:26+01:00"
      }
    ]
  }
}
```

The above response has:

[8] https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html

[9] https://tools.ietf.org/html/rfc7231

- A status code of 200, which means that the request was successful.
- A `Content-Type` header of `application/json` which means that the body is a JSON response.
- A body which contains the actual payload response from the server.

What Is an HTTP Status Code?

Web Services and HTTP APIs use HTTP Status Codes to tell us what happened when the server processed the request.

The simple grouping for HTTP Status Codes is:

- 1xx - Informational
- 2xx - Success e.g. 200 Success
- 3xx - Redirection e.g. 302 Temporary Redirect
- 4xx - Client Error e.g. 400 Bad Request, 404 Not Found
- 5xx - Server Error e.g. 500 Internal Server Error

The type of status code you receive depends on the application you are interacting with. Usually a 4xx error means that you have done something wrong and a 5xx error means that something has gone wrong with the application server you are interacting with.

You can learn more about status codes online:

- Wikipedia List[10]
- HTTP Statuses[11]

What Are Payloads?

A Payload is the body of the HTTP request or response.

When browsing the Web, the Browser usually receives an HTML[12] payload. This is the web page that you see rendered in the Browser.

Typically when working with an HTTP API we will send and receive JSON or XML payloads.

You saw JSON payloads in the examples above.

[10] https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

[11] https://httpstatuses.com/

[12] https://en.wikipedia.org/wiki/HTML

What Is JSON?

JSON stands for JavaScript Object Notation and is a text representation that is also valid JavaScript code.

```
{
  "projects": {
    "project": [
      {
        "id":  1,
        "name": "A New Projectaniheeiadtatd",
        "position": 0,
        "description": "",
        "state": "active",
        "created-at": "2017-06-27T12:25:26+01:00",
        "updated-at": "2017-06-27T12:25:26+01:00"
      }
    ]
   }
}
```

JSON can be thought of as a hierarchical set of key/value pairs where the value can be:

- Object - delimited by `{` and `}`.
- Array - delimited by `[` and `]`.
- String - delimited by `"` and `"`.
- Integer

An array is a list of objects or key/value pairs.

The keys are String values e.g. “projects”, “project”, “id”, etc.

What Is XML?

XML stands for Extensible Markup Language.

HTML is a variant of XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<projects type="array">
  <project>
      <id type="integer">1</id>
      <name>A New Projectaniheeiadtatd</name>
      <position type="integer">0</position>
      <description nil="true"/>
      <state>active</state>
      <created-at type="dateTime">2017-06-27T12:25:26+01:00
           </created-at>
      <updated-at type="dateTime">2017-06-27T12:25:26+01:00
           </updated-at>
      <default-context-id type="integer" nil="true"/>
      <completed-at type="dateTime" nil="true"/>
      <default-tags nil="true"/>
      <last-reviewed type="dateTime" nil="true"/>
  </project>
</projects>
```

XML is constructed from nested elements

- An element has an opening and closing tag e.g. `<state>` and `</state>`.
    - The tag has a name i.e. `state`.
    - The opening tag begins with `<` and ends with `>` e.g. `<state>`.
    - The closing tag begins with `</` and ends with `>` e.g. `</state>`.
- An element has a value, which is the text between the tags e.g. the `state` element has a value of `active`.
- An element can have attributes, these are always within the opening tag e.g. the `id` element (`<id type="integer">`) has an attribute named `type` with a value of `"integer"`.
- Elements can contain other Elements. These are called Nested Elements. e.g. the `projects` element has a nested element called `project`.

For XML to be valid, it must be well formed, meaning that every opening tag must have a corresponding closing tag, and strings must have an opening and closing quote.

Some elements do not have a closing tag, these are self closing. The opening tag, instead of ending with `>` actually ends with `/>` you can see this in the `<description nil="true"/>` element.

What Are HTTP Headers?

HTTP messages have the Verb and URL, followed by a set of headers, and then the optional payload.

```
POST http://www.compendiumdev.co.uk/apps/mocktracks/reflect.php HTTP/1.1
Host: www.compendiumdev.co.uk
Content-Type: application/json
Accept: application/json

{"action":"post"}
```

The headers are a set of meta data for the message.

Headers are a name, followed by `:`, followed by the value of the header.

The above HTTP message example has three headers:

- Host
- Content-Type
- Accept

The `Host` header defines the destination server domain name.

The `Content-Type` header tells the server that the content of this message is JSON.

The `Accept` header tells the server that the client (application sending the message) will only accept response payloads represented in JSON.

There are many headers available[13] for configuring the Authentication details, length of message, custom meta data, cookies etc.

What Is Authentication?

When we send a message to a server we might need to be authenticated i.e. authorised to send a message and receive a response.

For many Web Applications you authenticate yourself in the application by logging in with a username and password. The same is true for Web Services or HTTP APIs.

[13] https://en.wikipedia.org/wiki/List_of_HTTP_header_fields

If you are not authenticated and try to send a message to a server then you are likely to receive a response from the server with a 4xx status code e.g.

- 401 Unauthorized
- 403 Forbidden

There are many ways to authenticate HTTP requests for HTTP APIs.

Some common approaches you might encounter are:

- Custom Headers
- Basic Authentication Headers
- Session Cookies

Some HTTP APIs require **Custom Headers** e.g.

```
POST http://www.compendiumdev.co.uk/apps/mocktracks/reflect.php HTTP/1.1
X-APPLICATION_KEY: asds-234j-werw
```

Here the `X-APPLICATION-KEY` header has a secret value which authenticates the request.

Basic Authentication Headers are a standard approach for simple login details:

```
POST http://www.compendiumdev.co.uk/apps/mocktracks/reflect.php HTTP/1.1
Authorization: Basic Ym9iOmRvYmJz
```

The `Authorization` header specifies `Basic` authentication and is followed by a base64[14] encoded string.

- "`Ym9iOmRvYmJz`" is the base64 encoded version of the string "`bob:dobbs`"
- In Basic Authentication the string represents `username:password`

Session Cookies[15] are set by a server in a response message and are represented in a `Cookies:` header.

[14] https://en.wikipedia.org/wiki/Base64

[15] https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies

What Is REST?

REST stands for Representational State Transfer, and while it has a formal definition, which you can read in Roy Fielding's PHd thesis[16], it very often means that the API will respond to HTTP verbs as commands.

e.g.

- GET, to read information.
- POST, to create information.
- PUT, to amend information.
- DELETE, to delete information.

The documentation for the particular system you are testing will describe how the API has interpreted REST if they have described their API as a REST API.

What Tools Are Used for Accessing an API?

Since API stands for Application Programming Interface, we might expect all interaction with the API to be performed via program code. But it really implies that the interface is well documented and does not change very often.

Also that the input and output from the API are designed for creation and consumption by code - hence the use of formats like JSON and XML.

We can issue API requests from a command line with tools like cURL, which you will see later in this book.

Also GUI tools like Postman, which we cover in a later chapter, allow humans to easily interact with APIs.

When writing application code to interface with an API we are generally able to use a library for the specific programming language that we are working with.

In this book we are using Java and will use the REST Assured library.

[16] http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Example APIs

If you want a very simple API to experiment with at the moment, then I recommend the Star Wars API Web Application.

- swapi.co[17]

This is a very simple API that mainly uses GET requests and returns information about Star Wars characters and planets.

The API is well documented and has an online GUI that you can use to experiment with the API.

Recommended Reading

The REST API Tutorial[18] provides a good overview of REST APIs, HTTP Status codes and Verbs.

Summary

This chapter provided a very high level description of an API (Application Programming Interface) to differentiate it from a GUI (Graphical User Interface). An API is designed to be used for systems and applications to communicate, whereas a GUI is designed for humans to use.

Humans can use API interfaces. Tools such as cURL and Postman can help. We also have the advantage that for HTTP APIs, the messages are in text format and usually contain human readable JSON or XML format payloads.

The status codes that are returned when requests are sent to an API help you understand if the request has been successful (200, 2xx), or if there was something wrong with the request (4xx), or if something went wrong in the server application (5xx).

At the very least, you should now be familiar with the terms we will be using in this case study, and have seen some examples of HTTP messages for both requests and responses.

[17] https://swapi.co

[18] http://www.restapitutorial.com/

Introducing Tracks Case Study

This Case Study uses a Web Application called Tracks.

Tracks is a Todo Management and Productivity tool with a Web GUI and a REST API. The API uses XML as its Payload format.

This chapter will provide details of why we are using Tracks, where to find the source code and supporting information to help you get started.

Support Page

This book has an on-line support web page:

- compendiumdev.co.uk/page/tracksrestsupport[19]

The web page contains:

- Links to the important websites and tools mentioned.
- Links to the code.
- Links to the Postman collection file.
- Any errata or update information on the book.
- Links to any supporting blog posts.
- Links to any supporting videos.

Some of the web page sections will be mentioned in this book, but others will be added over time as it might be easier to add new content to the web page, than to add it into the book.

[19] http://compendiumdev.co.uk/page/tracksrestsupport

How to Use This Case Study

By the time you work through this case study, the version of Tracks will probably have increased. That might mean that some of the information here is out of date. You might not be able to exactly repeat all the steps and see the same results.

If you want to repeat the steps and achieve the same results then you could install the version of Tracks I mention in this book, and possibly the associated software that I used to test it at the time. Tracks is Open Source and the old versions are available to install; but I don't recommend you do that.

Instead, I recommend that you read the text, watch the videos (on the supporting web page), and perform 'looser' experiments. i.e. do the same, or similar things, but don't expect the result to be exactly the same, or look exactly the same, when you do it.

Over the course of the case study I used two versions of tracks: 2.2 and 2.3.

Tracks did change between these versions.

This required changes to the code that I wrote to automate Tracks, and later I'll show you what I changed in the code, and how I investigated the changes.

By the time you come to follow this case study, Tracks may have advanced again, and the code may need to change again.

You may need to change it. Since this is a case study, that is one of the exercises left to you.

Rather than starting from scratch, you can see what and why I created code, then you can change it to work for the current version. You can then take the code forward if you want.

A case study is a time bound body of work. It is what I did at a specific point in time. I'm communicating it so that you can learn from it, and build on it, not so you can repeat it exactly.

Source Code Location

All of the source code for REST Assured and Java mentioned in the book is available on GitHub:

- github.com/eviltester/tracksrestcasestudy[20]

[20] https://github.com/eviltester/tracksrestcasestudy

A chapter later in the book called "How to Use the Source Code" describes how to download the source code and install the JAVA SDK and IDE you'll need to run the code.

Case Study Contents Overview

- An overview of the Tracks application and its API
- How to use cURL to explore an API
- How to use Postman to explore an API
- How to use a Proxy to help automate and explore
- How to use REST Assured from Java:
  - Get URLs
  - Post forms
  - REST API testing
    - `GET`
    - `POST`
  - XML Response processing
  - JSON Response processing
  - Serialize payload objects to JSON and XML
  - Deserialize JSON and XML to Java objects
- Different stages of code
  - Code that gets something done
  - Code that we re-use
  - Code that we can use for the long term
  - How we refactor between these stages
- Scraping data from other sites to use as test data
- Thought processes and critical evaluation of the work
- Thoughts on REST Assured
- How to improve the code used in this case study
- Exercises for you to try

Why Test Tracks?

Why pick Tracks as an application to test?

Tracks[21] is an open source application written in Ruby on Rails so is relatively easy to install and use. Although, I should mention that, I have always used a pre-built virtual machine from Bitnami or Turnkey (more details later).

Practice Your Testing

I've been aware of Tracks as an application for quite a long time, and I've even experimented with it as a 'real' application i.e. one that I would use to track my TODO lists and manage my work.

I don't use Tracks to manage my work but I do use it as a target for practising my testing.

I've used it:

- personally to practice my testing
- personally to practice automating with WebDriver
- personally to learn new tools and experiment with proxies
- on training courses to provide a 'real' application to automate with WebDriver
- on training courses to provide a target for exploratory testing
- as part of the Black Ops Testing[22] workshops

I've primarily used it to practice testing. I hope that's one reason why you are working through this case study - because you want to practice your testing and improve.

Real Application

Tracks is a 'real' application.

I have built applications which are only used for the purposes of supporting training exercises. These tend to have limitations:

- specific aims - to exhibit particular flaws,
- they tend not to be rich in functionality,
- the GUI probably isn't polished,
- they might be deliberately limited e.g. no database, no security, etc.

[21] http://www.getontracks.org/

[22] http://blackopstesting.com

Since Tracks is built for production use it does not have artificial limits, and it changes over time which means that I can keep revisiting it to practice against new functionality.

It also has the type of bugs that slip through the net, some are easy to spot, some are not important, others are harder to spot and you have to work to find them.

Tracks has grown organically over time and isn't as basic as an application designed to be used for teaching testing and automating.

Tracks is built by a dedicated and passionate team of developers in their spare time. They also write a lot of code to help them test so we could review their automated execution coverage and use that to guide our testing.

Rich Functionality

Tracks has a lot of functionality and ways of accessing the functionality that a simpler 'test' application would not have:

- Tracks has a complex GUI.
- Tracks has a REST API.
- Tracks has a mobile GUI interface.
- Tracks is a client/server application.
- Tracks has a database.
- Tracks is multi-user.

Tracks has a complex GUI. The GUI also uses a lot of JavaScript, Ajax calls and DOM updating, this makes it appear more 'modern', but also offers challenges to the tester when automating through the GUI. This also opens up new technologies for the tester to learn.

Tracks has a REST API. Which allows us to experiment with more technical approaches to testing and going behind the GUI to learn new tools and approaches, exploring different risks than the GUI alone offers.

Tracks has a mobile GUI interface. Allowing us to experiment with device based testing in combination with browsers and APIs.

Tracks is a client/server application. Allowing us to focus on the client, or the server side functionality, and the interaction between the two. This also opens up server side logging and server side access. When you are testing a Browser application on a Windows machine and have to connect to a server running in a VM using Linux, you learn to switch between different operating system contexts and have to learn new technology to be able to test.

Tracks has a database. Allowing us to learn how to access it from a GUI tool, or from the command line. We don't have to access the database, but if we extend our testing to encompass the database then we will:

- learn new skills,
- spot new risks to test for,
- have the ability to investigate the cause of defects more deeply,
- manage our environment at a lower level.

Tracks is multi-user. This opens up risks around security, performance, testing under load etc.

Tracks also has the advantage that it is a very focused application, so while it offers a lot of features for testing, it is a small enough domain that we can understand the scope of the application.

Rich Testing Scope

I've been using Tracks as a test target for years. Every time I come to test it I can easily lose myself in it for days at a time.

This case study concentrates on the API and the client server interaction, but in no way does this approach to testing cover everything that Tracks offers. And I have not spent enough time to say that I have learned or tested everything about Tracks.

But I have spent enough time testing Tracks to be able to recommend it as a great testing target, and the time I've spent practising with it, has improved my testing skills and technical skills.

I think the time you spend working through this case study and applying the approaches to Tracks will improve your testing skills. Make sure you apply the techniques and approaches you see here, not just read about it, or watch the videos.

Only application of the approaches will help you advance.

What Is Tracks and GTD?

`Tracks` is an open source application written in Ruby on Rails which implements the David Allen "Getting Things Done" time management methodology.

- Tracks Home Page - getontracks.org[23]
- Getting Things Done[24]

Getting Things Done

For the purposes of this case study we don't really need to know the Getting Things Done methodology, we do have to understand the entities involved and their relationships, and how Tracks implements them, but if you'd like some additional domain knowledge then I'll cover it here.

GTD is a time management approach with a number of concepts:

- Projects - groups of Tasks which need to be done to implement the project
- Contexts - 'places' where Tasks can be done e.g. @Home, @Shops etc.
- Tasks - have a due date so they can appear on a calendar and you can track slippage or poor schedule estimating. I tend to think of these as TODOs and so I use the term 'Task' and 'TODO' interchangeably.

If you want to learn more about the GTD domain then the following links may also be helpful:

- Lifehacker summary[25]
- Additional documentation on Tracks Wiki[26]

[23] http://www.getontracks.org/
[24] https://en.wikipedia.org/wiki/Getting_Things_Done
[25] http://lifehacker.com/productivity-101-a-primer-to-the-getting-things-done-1551880955
[26] https://github.com/TracksApp/tracks/wiki/Help-%26-Support

Installing Tracks

Since we are using Tracks as a learning target we want to find easy ways to install it. I tend to use a pre-built virtual machine.

Because this chapter has the risk of becoming out of date I recommend that you visit the book support page if you experience any install difficulties:

- compendiumdev.co.uk/page/tracksrestsupport[27]

In this chapter I describe:

- how to use the official install from getontracks.org[28],
- pre-built virtual machines from Turnkey[29]
  - my preferred method for runnning Tracks

Official Tracks Install

The Tracks Download page has links to the install instructions. I include this for your reference, but this is not how I install Tracks, I use a Virtual Machine.

- getontracks.org/downloads[30]

The master install documentation is on GitHub:

- github.com/TracksApp/tracks/blob/master/doc/installation.md[31]

Each version may have a slightly different installation process:

[27] http://www.compendiumdev.co.uk/page/tracksrestsupport
[28] http://www.getontracks.org
[29] https://www.turnkeylinux.org/tracks
[30] http://www.getontracks.org/downloads/
[31] https://github.com/TracksApp/tracks/blob/master/doc/installation.md

- version 2.2 installation[32]
- version 2.3 installation[33]

You'll need to have Ruby installed, Bundler and a database (MySql, SQLite, PostgreSQL)

- Ruby[34]
- Bundler[35]

It is possible to use the 'official install' instructions with a pre-build 'Ruby' VM from Bitnami[36] or Turnkey. Instructions for installing Tracks into a Bitnami VM are on the book support web page.

Pre-Built Virtual Machines

My main focus with Tracks is having an application to practice testing on.

I prefer to download a pre-built virtual machine with Tracks already installed and running.

Throughout the case studies you will see references to `bitnami` because I used a Virtual Machine from Bitnami, unfortunately Bitnami have discontinued their Tracks VM, leaving Turnkey as the only pre-built Tracks VM that I know of.

Download a pre-build VM from Turnkey:

- Turnkey Tracks[37]

Many of the applications, on which I practice my testing, are run from pre-built virtual machines.

The disadvantage for this case study is that the virtual machines listed on Turnkey will only have one of the versions. Turnkey does not tend to maintain the older versions and make them available for download.

The advantage is that you have to do very little to get them working.

You can also easily install the virtual machines to Cloud instances if you want to have them running separately from your development machine.

[32] https://github.com/TracksApp/tracks/blob/2.2_branch/doc/installation.textile

[33] https://github.com/TracksApp/tracks/blob/2.3_branch/doc/installation.md

[34] https://www.ruby-lang.org/en/

[35] http://bundler.io/

[36] https://bitnami.com

[37] https://www.turnkeylinux.org/tracks

Using Virtual Machines

There are a few virtual machine applications that you can use.

- VirtualBox is open source and cross platform
- VMWare is free for some player options but is a commercial product
- Parallels on the Mac is also commercial, I haven't tried it with the virtual machines from Bitnami

I tend to use VMWare. On a Mac I use VMWare Fusion, and on Windows VMWare Workstation.

- vmware.com[38]

This is a paid project, but I find it more reliable.

The virtual machines that you can download from Bitnami and Turnkey will also work on VirtualBox.

- virtualbox.org[39]

VirtualBox is a free virtual machine host.

Turnkey provide a `.vmdk` file which is a Virtual Disk image which you can use with both VMWare and VirtualBox.

Turnkey also provide a `.ova` file which you can open with VirtualBox.

Summary

If we were to actually test the application then we would want to use the official install routines and have a well configured test environment. Since we are using it to practice, we can save ourselves some time and use one of the pre-built options.

I prefer to use virtual machines.

[38] http://www.vmware.com

[39] https://www.virtualbox.org

A Tour of Tracks

I have created a video to provide a quick walk-through of the Tracks GUI. You can find it on the book support page video section:

- compendiumdev.co.uk/page/tracksrestsupport[40]

Why Learn the GUI If We Are Testing the API?

Prior to automating an application, or working with its API, I usually try to use it from the GUI first.

I find this makes it easier for me to understand the basic functionality and build a mental model of the application.

I make an exception to this, when the application doesn't have a GUI at all.

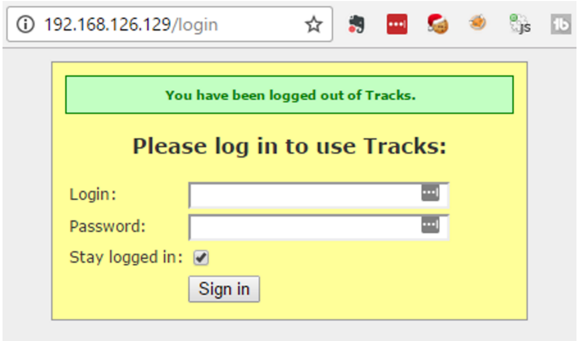
Since this is a mental model of the basic application, I simply use it in a browser with no additional tools. I will use an HTTP proxy in a later section. Initially I do not use it with a proxy because I don't want to become too distracted by the HTTP traffic. I use a proxy later when I want to understand the application at a more technical level in preparation for automating it or more technically focused exploratory testing.

Generally I'm doing this to build a model of how the application works and a list of questions and options that I will need to investigate prior to automating.

I'm just doing a quick tour to identify functionality that I can use via the GUI, and then I'll see if I can automate the same functionality using the API. Since we are concentrating on automating using the API, my notes will ignore the Ajax calls and GUI updating.

[40] http://compendiumdev.co.uk/page/tracksrestsupport#vtracksoverview

Login



192.168.126.129/login

You have been logged out of Tracks.

Please log in to use Tracks:

Login:

Password:

Stay logged in: ☑

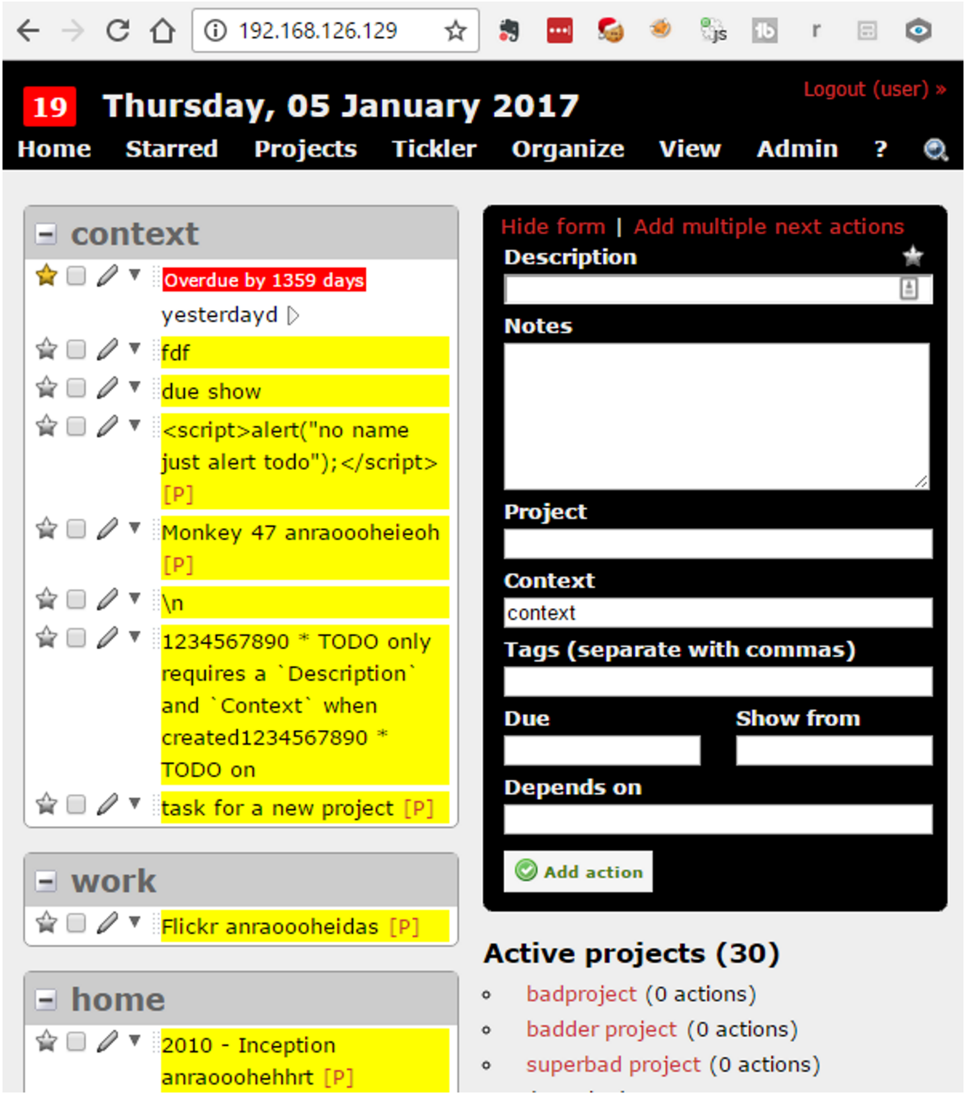
Tracks Login Screen

Login screen to the application, even though every user has their own set of Projects and TODO items.

This suggests I'll need to handle some sort of authentication when using the API. Also, if I were to automate via the API and the GUI at the same time I may be able to share the same session.

This case study used the (now discontinued) Bitnami pre-built virtual machines, the default admin user is 'user' with the password 'bitnami'.

Home Screen



Thursday, 05 January 2017

context

Overdue by 1359 days

yesterdayd

fdf

due show

<script>alert("no name just alert todo");</script> [P]

Monkey 47 anraoooheieoh [P]

\n

1234567890 * TODO only requires a `Description` and `Context` when created1234567890 * TODO on

task for a new project [P]

work

Flickr anraoooheidas [P]

home

2010 - Inception anraooohehhrt [P]

Context: context

Active projects (30)

- badproject (0 actions)
- badder project (0 actions)
- superbad project (0 actions)

Tracks Home Screen

You can see from the Home screen image that a tester has been using the application, just look at that data!

From the Home screen I can create TODO items (`action`):

- Projects and Contexts are created automatically when added to an `action`.
- Tagging provides another way of categorising and organising TODO items.
- TODO items can be dependent on one another.
- Validation rules applied to `action`,
  - `action` only requires a `Description` and `Context` when created,
  - `action` description length is limited to `100` in the GUI,
    - this is the only HTML validation, any other validation must be applied by the server.

The drop down menu suggests that I can 'star' actions. Since there is a `Starred` menu option.

I can do this on the GUI by clicking the 'star' at the side of the action.

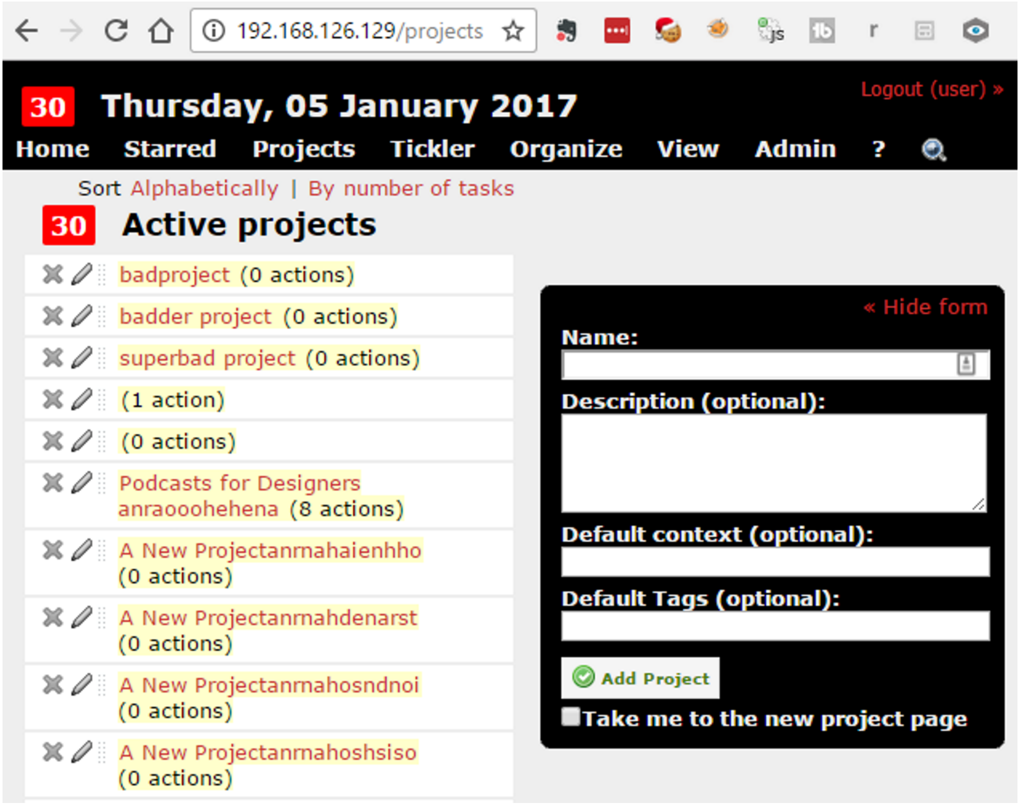
We can edit an `action` in-situ - I assume this is handled via Ajax, which would interesting to automate from the GUI, but we can ignore this since we are concentrating on the API.

Starred Screen

Starred screen shows starred `actions`.

I assume that the API will provide a way of retrieving a list of starred `actions`.

Projects



30 Thursday, 05 January 2017

Sort Alphabetically | By number of tasks

30 Active projects

- badproject (0 actions)
- badder project (0 actions)
- superbad project (0 actions)
- (1 action)
- (0 actions)
- Podcasts for Designers anraooohehena (8 actions)
- A New Projectanmahaienhho (0 actions)
- A New Projectanmahdenarst (0 actions)
- A New Projectanmahosndnoi (0 actions)
- A New Projectanmahoshsiso (0 actions)

Name:

Description (optional):

Default context (optional):

Default Tags (optional):

□ Take me to the new project page

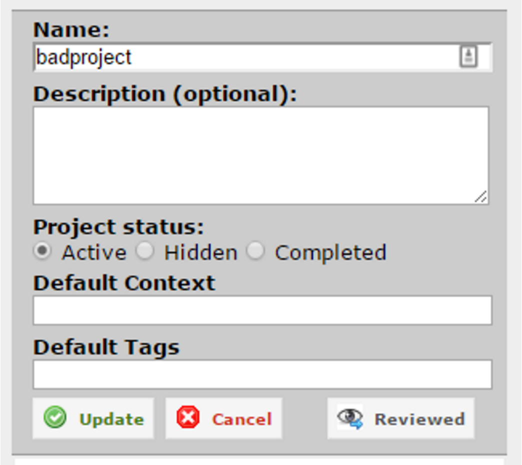
Tracks Projects Screen

Projects shows a list of `project` entities.

Can create a Project from this screen.

Clicking on a Project shows a `project` screen.

Project Edit



Name:
badproject

Description (optional):

Project status:
● Active ○ Hidden ○ Completed

Default Context

Default Tags

Tracks Project Edit Dialog

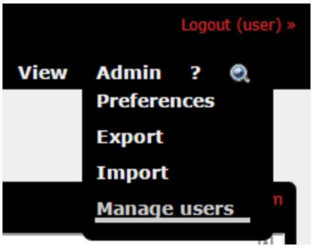
We can amend the details for a `project`.

A `project` can be `active`, `hidden`, `completed`.

A `project` can be amended after creation to have a `description`, a default `context` and `tags`.

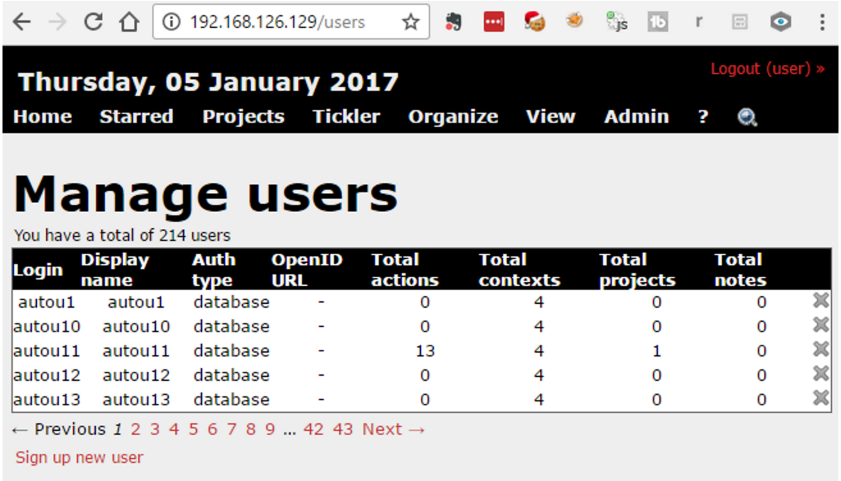
Validation looks as though it is done server-side.

Admin



Tracks Admin Menu

Admin allows editing of user details using `Manage Users` from the `Admin` menu.



Thursday, 05 January 2017

Manage users

You have a total of 214 users

Login	Display name	Auth type	OpenID URL	Total actions	Total contexts	Total projects	Total notes	
autou1	autou1	database	-	0	4	0	0	✖
autou10	autou10	database	-	0	4	0	0	✖
autou11	autou11	database	-	13	4	1	0	✖
autou12	autou12	database	-	0	4	0	0	✖
autou13	autou13	database	-	0	4	0	0	✖

← Previous 1 2 3 4 5 6 7 8 9 … 42 43 Next →

Sign up new user

Tracks Manage Users Screen

An Admin user can Manage and create users.

The 'admin' user and the 'normal' user have different options exposed here, so the API should also honour permissions.

Basic Functions to Use and Check in the API

When I test the application I will want to be able to create data in the system to support my testing. So I would like to be able to:

- Create users
- Create `action` and `project` and `context` for users

Basic functionality in the API that I would want to start writing API code to support testing would include:

- Login and authenticate
- Action
  - Create `Action`, and have `context` created automatically
    - Create `action` uses same data validation as GUI
      - `description` is 100 length maximum
      - must have a Context

  - List `Actions` in a `Context`
  - ‘Star’ an ‘Action’
  - Retrieve list of ‘starred’ `Actions`
  - Edit an `action`
- Project
  - Create Project
    * Add `action` to existing Project
  - Amend Project
  - Move Project into different status
  - Show `actions` for a Project
- Admin
  - Create Users
  - User can amend their details (but not another user)

Summary

When an application has a REST API and a GUI, I will use the application via the GUI first to identify the basic functional scope that might be available and to help me understand what entities the application manages and what features it offers.

The Tracks REST API

You can find a video overview of the REST API with cURL on the book support page.

- compendiumdev.co.uk/page/tracksrestsupport[41]

This chapter will provide a short introduction to the terminology and the tools we will initially use, with links to the documentation and downloads. We cover cURL and the REST API in much more detail in a later chapter.

What Is a REST API?

‘REST’ stands for ‘Representational State Transfer.

It basically means an API which uses HTTP and uses the ‘HTTP Verbs’ (`GET`, `PUT`, `POST`, `DELETE`, etc.) to retrieve data or amend data through the API. Any data retrieved by a `GET` would be in the body of the HTTP response, and any data in a `PUT` or `POST` request to create and amend data would be in the body of the HTTP request.

You can find information about REST:

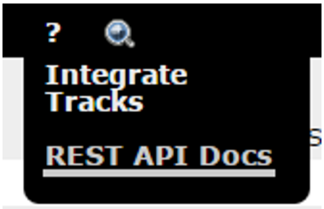
- REST Wikipedia page[42]
- An overview REST tutorial at restapitutorial.com[43]
- Roy Fielding[44] defined REST in his Ph.D. Thesis[45], you can read it as a .pdf[46]

[41] http://compendiumdev.co.uk/page/tracksrestsupport#vcurloverview
[42] https://en.wikipedia.org/wiki/Representational_state_transfer
[43] http://www.restapitutorial.com/
[44] https://www.ics.uci.edu/~fielding/
[45] http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
[46] https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

Tracks REST API Documentation

The Tracks API documentation is in the application itself.

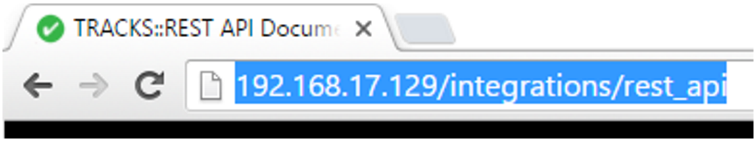
From the `?` menu, the `REST API Docs` menu.



The 'REST API Docs' menu

Clicking this takes us to a page with the API documentation.

For easier future reference I make a note of the URL.



URL docs located at `/integrations/rest_api`

- `/integrations/rest_api`

API Doc Examples

The API documentation is fairly minimal but it does have some examples - in cURL

- cURL official page[47]
- cURL Wikipedia description[48]

One of the things I will want to do is try the examples using cURL, before I try and use the API in other tools. This way I can learn if the API examples work as documented before

[47] https://curl.haxx.se/

[48] https://en.wikipedia.org/wiki/CURL

trying to translate them to another tool. If I can use the API via cURL then I can translate the examples to any other mechanism later.

From the documentation I can see that:

- System uses Basic HTTP Authentication for the API
  - This is different from the main application, which uses a logged on session cookie
- Message content is XML
- `GET` is used to retrieve data
- `DELETE` is used to delete data
- `PUT` is used to amend data
- `POST` is used to create data
  - If we are not familiar with cURL then it is hard to tell from the cURL request that `POST` is used

The following endpoints are used:

- `/todos.xml`
- `/todos/ID.xml`
- `/tickler.xml`
- `/done.xml`
- `/hidden.xml`
- `/calendar.xml`
- `/contexts.xml`
- `/contexts/ID.xml`
- `/contexts/ID/todos.xml`
- `/projects.xml`
- `/projects/ID.xml`
- `/projects/ID/todos.xml`

The documentation says that we can also limit returned information to the 'index' fields: `ID`, `created_at`, `modified_at`, `completed_at`. We do this by adding `?limit_fields=index` to the request.

We can get only active TODOs by adding a parameter `?limit_to_active_todos=1`.

Omissions in API Documentation

`user.xml` or `users.xml` is not mentioned as an endpoint so we might not be able to create or amend users through the API.

Also I suspect we might be able to have more 'limit' type parameter combinations.

General HTTP REST Return Codes

When we issue REST calls, there are a few HTTP return codes we would expect to see:

- `200` (OK) when we `GET` to retrieve information
- `401` (Unauthorized) when we try to access information without the correct credentials e.g. without logging in
- `404` (Not Found) when we `GET` information that does not exist, or `POST`, `PUT` `DELETE` to an end point that does not exist
- `201` (created) when we use `POST` to create a new entity
- `409` (Already Exists) we we try to create an existing entity
- `302` (Redirect) when we successfully issue a request, the system may redirect us somewhere else, e.g `POST` a login request and be redirected to the user dashboard

Useful information on HTTP Status Codes can be found at:

- httpstatuses.com[49]

Summary

The Tracks REST API is well documented, with examples of how to trigger it interactively.

Testing an application to ensure that it conforms to the documentation is an important part of any testing process, in addition to helping us learn and understand the system.

Although the Tracks REST API only uses XML, I do cover JSON processing in a later chapter.

[49] https://httpstatuses.com/

Using a Proxy to View GUI Traffic

Before I start using cURL and working with the API, I want to first check if the GUI itself uses the API.

I check by using the GUI in a browser, while all the browser traffic is captured by an HTTP Proxy.

You can find a video overview of using Tracks through a proxy in the book support page videos:

- compendiumdev.co.uk/page/tracksrestsupport[50]

Why?

I do this because, having read the documentation on the API, I now know what an API call looks like.

If the GUI itself uses the API then when we test or automate through the API we have also covered a lot of the GUI to server interaction. This would allow us to reduce the automated coverage that we might want to achieve through GUI automated execution.

If the GUI does not use API calls, then it suggests that the application may have multiple implementations of the same functionality. Or, possibly more than one route to the same functionality. Either way, we probably have to continue to test the same functionality through the GUI, as we do through the API.

In order to make a technically informed decision about the risk we would have to review the code, rather than rely on information obtained from observing the HTTP traffic.

How?

The easiest HTTP Proxies to use are:

[50] http://compendiumdev.co.uk/page.php?title=tracksrestsupport#vproxy

- For Windows: Fiddler[51]
- For Mac: Charles[52]

Both of these are ‘easiest’ because they hook in to the Operating System HTTP connections and start listening to traffic by default without any configuration.

You could also use:

- OWASP ZAP Proxy[53]
- BurpSuite Free Edition[54]

Both of these are free, and cross platform projects (they require a Java Runtime Edition installed). You have to do a little extra configuration in the browser to amend the network connection settings to use a proxy.

- Firefox Network Connection Configuration[55]
- Chrome Proxy Configuration[56]
- IE Proxy Configuration[57]

This ‘how to’ article[58] details proxy configuration for different browsers.

Viewing Traffic

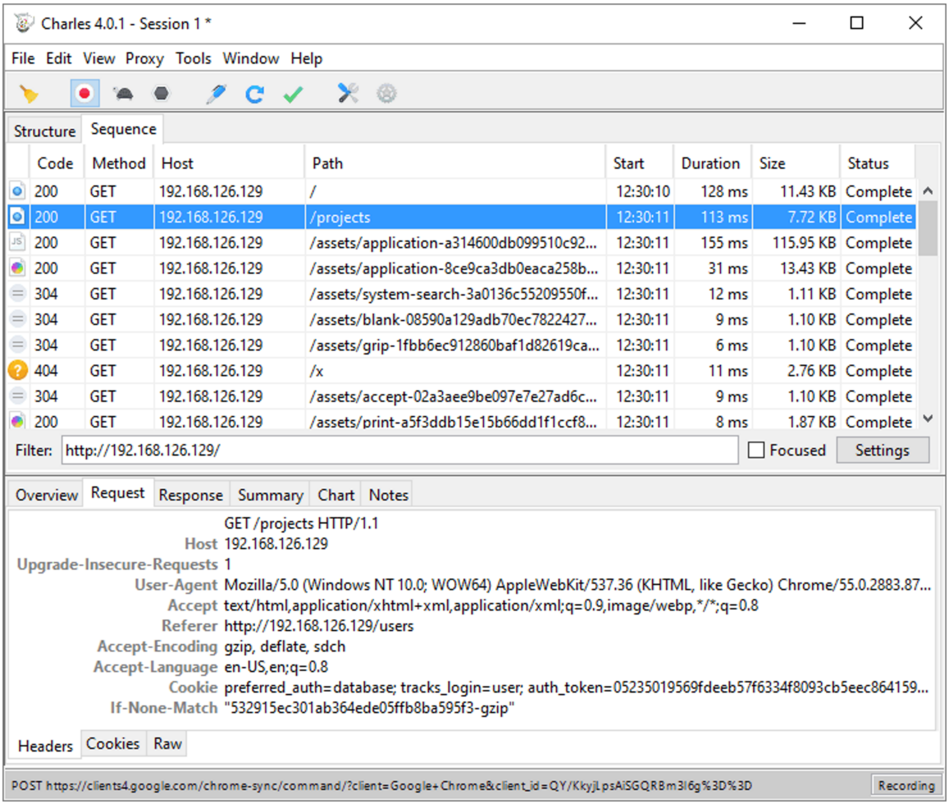
Assuming that you have installed the proxy correctly:

- You have a browser installed.
- You have one of the proxy tools installed.
- The proxy tool is running.
- The browser is running.

[51] http://www.telerik.com/fiddler
[52] https://www.charlesproxy.com/
[53] https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project
[54] https://portswigger.net/burp/download.html
[55] https://support.mozilla.org/en-US/kb/advanced-panel-settings-in-firefox#w_connection
[56] https://support.google.com/chrome/answer/96815?hl=en-GB
[57] http://windows.microsoft.com/en-gb/windows/change-internet-explorer-proxy-server-settings
[58] http://www.howto-connect.com/windows-10-customize-proxy-server-settings-in-browsers/

- The browser is configured to use the proxy.
- When you visit the URL for your tracks installation, the site still loads.

We can then start viewing and analysing the traffic to see if API URLs are used.



Code	Method	Host	Path	Start	Duration	Size	Status
200	GET	192.168.126.129	/	12:30:10	128 ms	11.43 KB	Complete
200	GET	192.168.126.129	/projects	12:30:11	113 ms	7.72 KB	Complete
200	GET	192.168.126.129	/assets/application-a314600db099510c92...	12:30:11	155 ms	115.95 KB	Complete
200	GET	192.168.126.129	/assets/application-8ce9ca3db0eaca258b...	12:30:11	31 ms	13.43 KB	Complete
304	GET	192.168.126.129	/assets/system-search-3a0136c55209550f...	12:30:11	12 ms	1.11 KB	Complete
304	GET	192.168.126.129	/assets/blank-08590a129adb70ec7822427...	12:30:11	9 ms	1.10 KB	Complete
304	GET	192.168.126.129	/assets/grip-1fbb6ec912860baf1d82619ca...	12:30:11	6 ms	1.10 KB	Complete
404	GET	192.168.126.129	/x	12:30:11	11 ms	2.76 KB	Complete
304	GET	192.168.126.129	/assets/accept-02a3aee9be097e7e27ad6c...	12:30:11	9 ms	1.10 KB	Complete
200	GET	192.168.126.129	/assets/print-a5f3ddb15e15b66dd1f1ccf8...	12:30:11	8 ms	1.87 KB	Complete

Filter: http://192.168.126.129/ ☐ Focused

```
                         GET /projects HTTP/1.1
                    Host 192.168.126.129
Upgrade-Insecure-Requests 1
              User-Agent Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.87...
                  Accept text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
                 Referer http://192.168.126.129/users
         Accept-Encoding gzip, deflate, sdch
         Accept-Language en-US,en;q=0.8
                  Cookie preferred_auth=database; tracks_login=user; auth_token=05235019569fdeeb57f6334f8093cb5eec864159...
           If-None-Match "532915ec301ab364ede05ffb8ba595f3-gzip"
```

POST https://clients4.google.com/chrome-sync/command/?client=Google+Chrome&client_id=QY/KkyjLpsAiSGQRBm3l6g%3D%3D

Viewing Traffic in Charles Proxy

When I use the application and:

- Login.
- Create an `action`.
- Create a `project`.
- Visit different parts of the GUI to see lists of entities.
- Amend an `action`.
- Amend a `project`.

- Delete an `action`.
- Delete a `project`.

I can see that only `POST` and `GET` are used, and none of the URLs are `.xml` URLs, so different parts of the application are being called, the API is not used by the GUI.

Implications

For our purposes of learning how to use the API, it means that we don't have any examples in the Proxy of the calls and results used for the API.

Which means that we have to rely on the documentation showing us how to use the API, rather than having 'live' examples taken from the actual application itself.

Summary

Use the GUI, with traffic directed through a proxy, to see if the GUI is using the API.

If the GUI is using the API then we have examples of the API calls being used. This might help us better understand the API or possibly spot any undocumented functionality. We might even spot nuanced usage that wasn't clear from the documentation.

If the GUI is not using the API then we are aware that testing the API in isolation does not mitigate all the risk associated with server side application functionality. The server side functionality is triggered in at least two ways: the GUI made HTTP calls and the REST API calls. This increases the potential scope of testing.

Exploring the Tracks API with cURL Through a Proxy

Let it be known that I have a 'thing' for HTTP proxies.

I like to:

- Have the ability to observe the traffic that I'm using in my testing, both requests and responses.
- Have a record of the actual messages sent and responses received.
- Look back through these records later when I start writing code to automate the API to make sure that the requests I send with code are the same as those I sent when exploring the API.

You can find a video overview of using cURL through a proxy in the book support page videos:

- compendiumdev.co.uk/page/tracksrestsupport[59]

Using a Proxy with cURL

- `-x <proxydetails>` - use a proxy for HTTP requests e.g. e.g. `-x localhost:8080`
  - `--proxy <proxydetails>` - an alias for `-x`
- `-U <user:password>` - set the proxy username and password

If I want to repeat any of the commands I have issued, but send the requests through a proxy, all I have to do is add the `-x` and include my proxy details.

Since none of the proxies I have mentioned need a password (Fiddler, Charles, BurpSuite and OWASP ZAP) I can ignore the `-U` flag and just use `-x`

e.g.

[59] http://compendiumdev.co.uk/page.php?title=tracksrestsupport#vproxy

```
curl -x localhost:8080 \
     -u username:p4ssw0rd -H "Content-Type: text/xml" \
      http://192.168.17.129/contexts.xml
```

For Debugging

If I know the cURL commands work successfully, and I capture the actual requests sent when I issue the cURL commands, then I have a baseline set of 'good' requests.

If I then encounter problems when I automate, or use a REST client, then I can compare the messages sent by the new tools with the original cURL message requests.

If there are any differences between the new requests and those sent by cURL then the differences are a good place to start looking for a source of any problems.

The cURL messages are likely to be the most minimal messages that I send, i.e. fewer headers and request paraphernalia so they can act as a very simple baseline.

For Exploration

cURL can feel a little clumsy to repeat messages and amend the content of longer messages.

One benefit of playing them through a proxy is that we can then use the replay and edit functionality in the proxy to resend requests, and amend the requests to explore different conditions.

For Simple Scope Coverage

Some of the proxy tools, e.g. BurpSuite and OWASP ZAP, have 'fuzzers'. I can use a Fuzzer to iterate over values, in much the same way as I fed a file of values into the commands from Bash or the command line.

I can do the same from the 'fuzzers' in the Proxy tool GUI. I find it easier to use the Fuzzer GUI than use the command line to create lots of data.

Summary

Having the ability to feed tools through HTTP debug proxies offers us enormous flexibility in our testing.

We can store the requests and responses for future review or support evidence of testing.

We can resend edited requests from the proxy GUI as well as from cURL, this can make it easier to experiment with subtle tweaks for more in-depth testing or just until we learn how to make the request work.

We can review the requests and responses in an environment designed to render them, often with pretty printing and formatting.

The increased ability to observe, interrogate and manipulate the requests will increase your understanding of the API and flexibility in how you test the API.

cURL Summary

The official cURL web site - curl.haxx.se/docs[60]

For quotes - remember `"` on Unix means allow expansion of variables, so use `'` on Unix and `"` on Windows.

Examples:

- `curl <url>` - `GET` the URL
- `curl -X <verb> <url>` - issue the verb request to the URL e.g. `GET`, `PUT`, `POST` etc.
- `curl --version` - see what version you are using
- `curl "eviltester.com?p1=a&p2=b"` - `GET` a URL with parameters

Options:

- `-v` for verbose mode to see response
- `-i` for 'include' to see response headers
- `--trace <filename>` - output full message as a HEX and ASCII representation to the `<filename>`
- `--trace-ascii <filename>` - output full message trace as an ASCII representation to the `<filename>`
- `-b <set-cookie-line>` - sends the cookies e.g. `N1=V1;N2=V2`
- `-b <filename>` - when no = the file is read as cookies
- `-c <filename>` - writes any cookies to this cookie jar file
- `-H "<header>"` - sets the header e.g. `-H "Content-Type: application/json"`
- `-x <proxydetails>` - use a proxy for HTTP requests e.g. `localhost:8080`
- `--proxy <proxydetails>` - use a proxy for HTTP requests e.g. `localhost:8080`
- `-U <user:password>` - set the proxy username and password
- `-A <user-agent>` - set the user agent
- `-o <file>` - send console output to a file

[60] http://curl.haxx.se/docs/

For PUT and POST payloads:

- `-d <data>` - send the data as form URL encoded
- `-d @<filename>` - sends data from a file (also `--data` etc.)
- `-F "name=@<filename>;type=text/plain"` - multi-part form data
- `-d "name=value"`

With `-d` options the Content Type is automatically set to:

- `application/x-www-form-urlencoded`

With `-F` options the Content Type is automatically set to:

- `multipart/form-data`

To split commands across multiple line use \ on Unix and `^` on Windows.

e.g. for Unix:

```
curl -u username:p4ssw0rd -H "Content-Type: text/xml" \
      http://192.168.17.129/contexts.xml
```

and for Windows:

```
curl -u username:p4ssw0rd -H "Content-Type: text/xml" ^
      http://192.168.17.129/contexts.xml
```

Exploring Tracks API with Postman REST Client

There are many REST Client GUIs available. I primarily use Postman.

- getpostman.com[61]

Postman is free and parts are Open Source. It was originally a Chrome Application and then converted into a Desktop Application.

This chapter will deal with the Desktop Application. Chrome Applications were deprecated in mid 2016 and will no longer be supported. I have moved the Chrome Application coverage to an appendix.

You can find a video overview of the Postman GUI on the book support page.

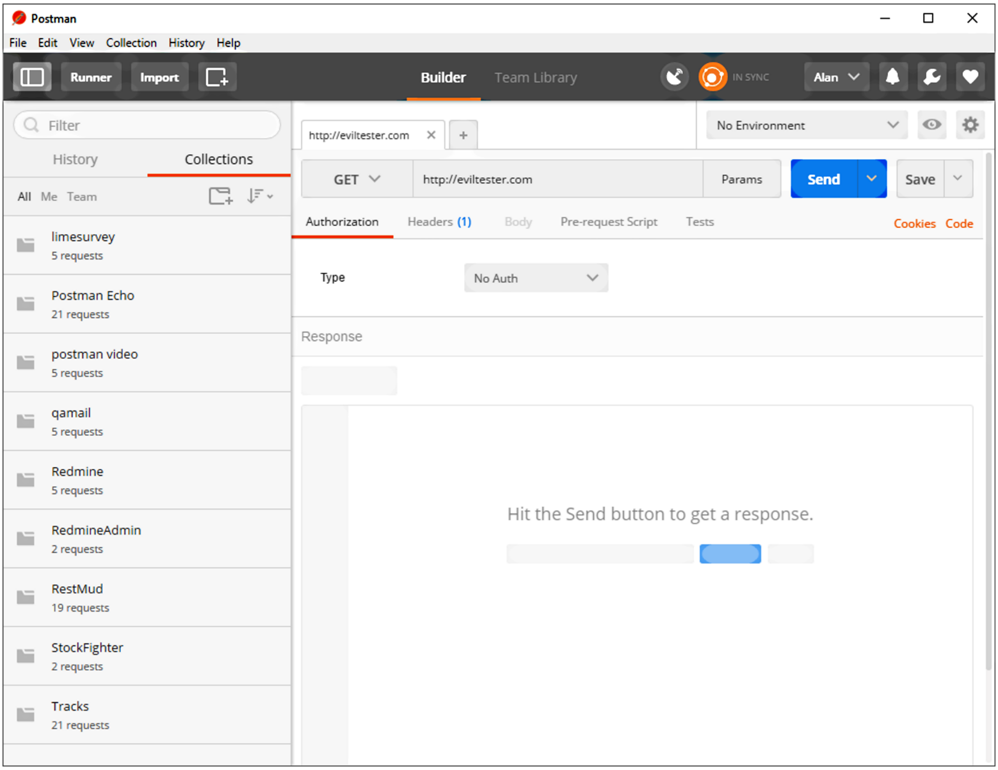
- compendiumdev.co.uk/page/tracksrestsupport[62]

The GUI

The Postman GUI is simple to use.

[61] https://www.getpostman.com/

[62] http://compendiumdev.co.uk/page/tracksrestsupport#vpostmangui



GET http://eviltester.com

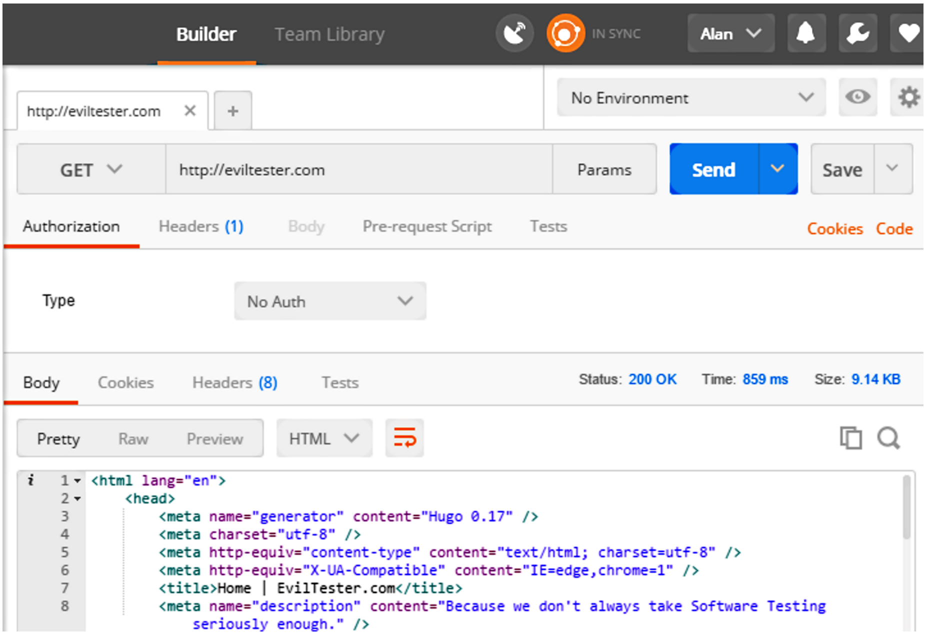
Type: No Auth

Postman GUI

- Central part of the screen is where we issue requests.
- Left hand side bar is a set of 'collections' of saved requests, and history of previous requests.

Issue Requests

We issue requests by amending the details in the main GUI.



No Environment

GET http://eviltester.com

Type: No Auth

Status: 200 OK Time: 859 ms Size: 9.14 KB

```
<html lang="en">
    <head>
        <meta name="generator" content="Hugo 0.17" />
        <meta charset="utf-8" />
        <meta http-equiv="content-type" content="text/html; charset=utf-8" />
        <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1" />
        <title>Home | EvilTester.com</title>
        <meta name="description" content="Because we don't always take Software Testing
            seriously enough." />
```

Issue a Request with Postman

We can change:

- The HTTP verb using the drop down. The screenshot shows a `GET` request but we can change that to any of the HTTP verbs e.g. `POST`, `DELETE`, `PUT` etc.
- The URL. The screenshot shows `http://eviltester.com`.
- The URL parameters by using the `Params` button, as this opens an easy way to edit any URL parameters.
- The `Authorization` used for Basic Auth, OAuth, etc.
- The `Headers` and add any header information we need.
- The `Body` if we are issuing a verb that allows body text e.g. `POST`.

Also:

- The `Send` button will issue the request.
- The `Save` button allows you to save the request to a Collection, use `Save As` if you are editing a request from a Collection.

After issuing a request the results will be shown in the `Body` tab and you can:

- See the HTTP Status code (in the above screenshot it was a `200 OK`).
- See the time it took to receive the response.
- See the size of the response.
- View the message response as raw text.
- Pretty print the response.
- Change the type that Postman has rendered the response e.g. `HTML`, `JSON` etc. to use a different Pretty Print view.

You can also view any cookies issued, and the headers used.

What you can't see is the actual request sent - which is why I like to configure Postman to use a proxy server. (see explanation later)

The GUI is pretty straight forward for sending basic requests. If you explore you should figure it out. Assuming of course that you know the semantics of the HTTP requests you are sending.

Postman Collections

When you have a request that works in Postman you can save it to a Collection for re-use.

Postman can 'sync' Collections of requests to all devices. I find this useful since I often switch between working on a Windows machine, and on a Mac. Postman will sync the changes I made to the Collection on one machine with the other so I always have an up to date set of messages to use.

If you create a Postman account then you can share your Collections with others.

You can find the Postman Collection for this case study shared as:

- getpostman.com/collections/b9e81b009bfd21ec5e83[63]

I created a 'bit.ly' link in case you want to type it manually:

- bit.ly/2iPaxgj[64]

[63] https://www.getpostman.com/collections/b9e81b009bfd21ec5e83

[64] http://bit.ly/2iPaxgj

To use the Collection in Postman you would import it. Either use the "import" button or the drop down menu item on the "Collection" menu.

You can import the Collection using the link above with the "Import From Link" option, or from the file in the source code repository in the `\postman` folder.

Requests in Collections can be organized into folders to make them easier to use.

Environment Variables

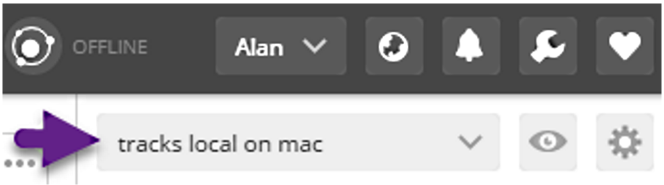
Environment variables are very useful because they allow you to have a saved Collection of responses which you can send to multiple environments.

If I'm running Tracks in a virtual machine then the IP address of the virtual machine might change and I don't want to have to amend every request before I send it to use the most up to date IP address. Instead I would use environment variables in the request:

e.g. I would want to `GET`:

- `http://{{url}}/contexts.xml` rather than
- `http://129.128.1.16/contexts.xml`

The Environment variables GUI section are in the top right of the main builder part of the GUI:



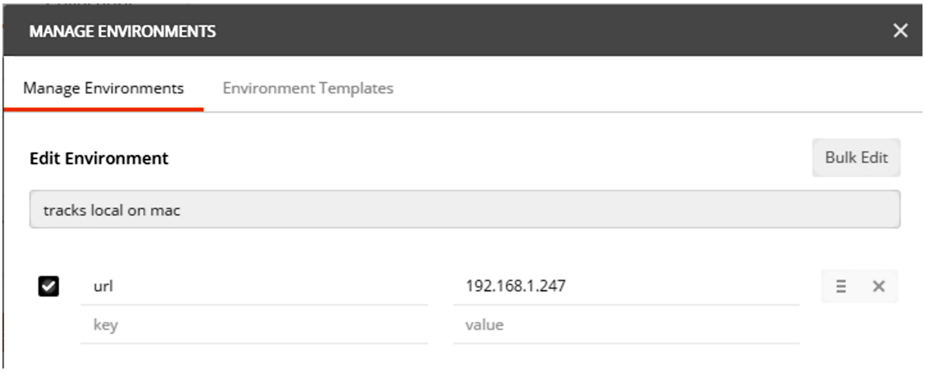
Environment Management Section

This consists of:

- A drop down showing the list of environments.
- An 'eye' which shows the values of the environment variables.
- A settings 'cog' where you can manage the environments.

An environment is essentially a named set of key value pairs.

In order to use the Collection for this case study you would need to create an environment which had a `url` key value pair:



tracks local on mac

☑ url 192.168.1.247

Environment Editing

Whatever `key` you create in the environment section, you can use in the requests with `{{key}}`.

e.g. if the `key` name was `url` you would write `{{url}}` in the request and it would be replaced by the value in the chosen environment.

If your requests fail, or you receive a result you don't expect, then make sure you check the environment variables exist and that they are set correctly.

Authentication

The `Authentication` tab lets you set a username and password for the authentication scheme used by the application.

You might need to amend this for the requests in the case study Collection. When you do, make sure you press the `Update Request` button prior to sending the request.

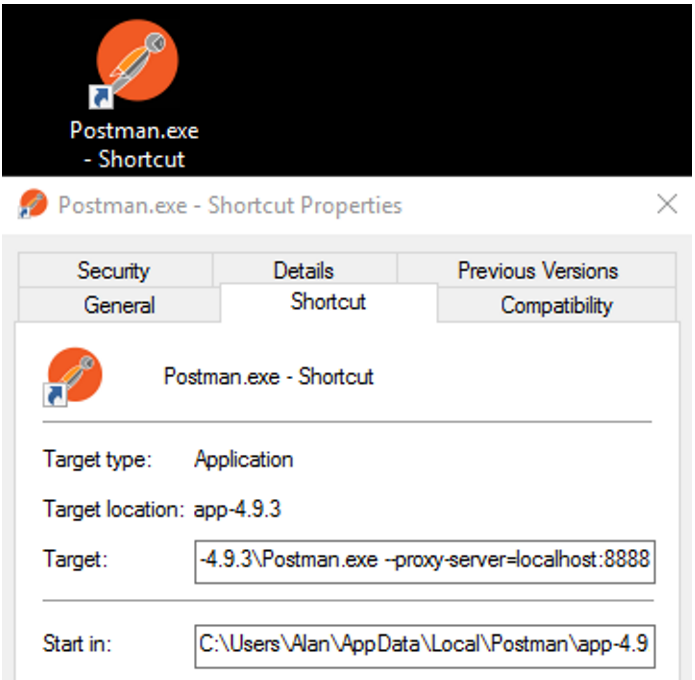
Using Postman Through a Proxy

To use Postman through a proxy, we have to start the application with a different command line argument.

```
Postman.exe --proxy-server=localhost:8888
```

You can find the location of Postman by looking at the shortcut created by the Postman installer.

After Postman has installed on Windows, I create a copy of the shortcut, edit the properties of the copied shortcut to add the command line arguments in the `Target` field.



Postman.exe - Shortcut

Target type: Application

Target location: app-4.9.3

Target: -4.9.3\Postman.exe --proxy-server=localhost:8888

Start in: C:\Users\Alan\AppData\Local\Postman\app-4.9

Postman Proxy Properties

You can find a video overview of using Postman through a proxy on the book support page.

- compendiumdev.co.uk/page/tracksrestsupport[65]

Proxy on Mac

To start a GUI application on Mac from the command line, you could use the `open` command from a terminal:

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

The above command uses:

[65] http://compendiumdev.co.uk/page/tracksrestsupport#vpostmanproxy

- `open` to start the application
- `/Applications/Postman.app` which is the path of the Postman application
- `--args` to tell the `open` command to accept command line arguments
- `--proxy-server=localhost:8888` the command line argument to set the proxy.

You can find other options for starting Mac applications from command line on this `Superuser.com` answer superuser.com/questions/16750[66]

Recreate cURL Requests in Postman

To start using Postman, the easiest thing to do is re-use the requests that you have been issuing from cURL.

Since you already know that the cURL request works, you can compare the cURL request with the Postman request to debug it.

If you are using a proxy then you can compare the sent request from both cURL and Postman, in detail, to spot any differences.

Summary

This chapter concentrated on the basics of the Postman desktop client and provided a simple overview of the GUI.

I fully expect this chapter to go out of date quickly, although I expect the basic functionality listed here to remain in Postman with much the same GUI.

The GUI isn't complicated, and if you experiment with it, you should be up and running with the basics very quickly.

To use the case study Collection, remember to create an 'environment' with a `url` and remember to change the Basic Auth values prior to sending a request. Or create an Admin user in your Tracks system with the username `user` which has a password of `bitnami`.

[66] https://superuser.com/questions/16750/how-can-i-run-an-application-with-command-line-arguments-in-mac-os

Starting to Automate

We have explored the API with cURL, proxies and Postman. Now we have a pretty good idea of how the API works and the type of responses it returns. We can start to think about automating it.

Why Explore First?

Part of the reason for doing the exploratory learning work first was to make sure we had several baselines to refer back to.

- cURL basic requests.
- Examples of the 'real' cURL requests and responses saved from the proxy.
- Saved requests in Postman with which we can quickly experiment.

Without the previous work, if something goes wrong when we automate we don't know if:

- We haven't understood the API.
- The API is broken.
- Our request is wrong.
- We are not receiving the correct responses.

But now we have evidence to compare back to if anything goes wrong. We can use the previous exploratory tools and approaches when we want to do something new in the GUI.

We have also built up basic coverage scenarios that we want to automate and now have a bit of a plan.

Choosing a Java Library

REST is essentially HTTP, so we could just use an HTTP library.

Using an HTTP library would be analogous to using cURL or the HTTP proxy. We gain a lot of low level configurable control, but it is a little slower and we work at the level of HTTP abstractions rather than REST requests.

Postman gives us a bit more of an abstraction on top of HTTP, since we can parse the responses more easily, but it is essentially an HTTP tool. The use of Collections is where the REST abstractions really start to come into play because we can organise the requests into a higher level set of categories.

There are always options when choosing a library for the programming language you will automate in.

For example in Java, a quick web search provided me with the following options:

- unirest.io[67]
- resty[68]
- jersey[69]
- JavaLite[70]
- REST Assured[71]
- OkHTTP[72]
- Apache HttpClient[73]

I chose to use REST Assured because I've used it before.

Although, really I'm cheating, because REST Assured uses Groovy, rather than Java, so adds some additional dependencies to the project that I don't really need.

But I have used REST Assured on a number of projects and it is pretty simple use. REST Assured also supplies some useful classes for parsing responses in XML and JSON.

In summary then, my decision to use the library is a pragmatic one based on previous experience. Not because it is the 'best' library. Not because I've evaluated them all and chosen the most suitable for the project. Instead, simply because I know I can create something quickly with it.

[67] http://unirest.io/java.html

[68] https://beders.github.io/Resty/Resty/Overview.html

[69] https://jersey.java.net/

[70] http://javalite.io/http

[71] https://github.com/rest-assured/rest-assured

[72] http://square.github.io/okhttp/

[73] https://hc.apache.org/

REST Assured Overview

REST Assured describes itself as a "Java DSL for simplifying testing of REST based services". I think 'DSL' is pushing it a bit far.

I think of REST Assured as a library, with a fluent interface, for automating REST services and HTTP applications easily.

All the examples for using REST Assured, use it as a set of `static` imports, which gives it the impression of looking more like a 'DSL' than a library.

I don't particularly like using lots of static imports, but since that is the convention for using REST Assured, that is what I will do.

You can find very good instructions for using REST Assured on the web site so I'm mainly going to describe how I used it, rather than all the features it provides.

The way that I use REST Assured is slightly different than the documentation describes and, I suspect, different from how many people use it. My usage differs because I don't really use the assertion mechanisms that REST Assured provides, or its BDD approach in the `@Test` methods.

Installation

Installation is very simple for Maven based projects - add the `rest-assured` dependency in the `pom.xml`.

- github.com/rest-assured/rest-assured/wiki/GettingStarted[74]

```
<dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>rest-assured</artifactId>
    <version>3.0.1</version>
</dependency>
```

Usage

I can use REST Assured to write `@Test` methods like the following:

[74] https://github.com/rest-assured/rest-assured/wiki/GettingStarted

```
@Test
public void aUserCanNotAccessIfNoBasicAuthHeaderUsingRestAssured(){

    given().
            contentType("text/xml").
    expect().
            statusCode(401).
    when().
            get("http://192.168.17.129/todos.xml");

}
```

Many older REST Assured examples are written in the `given`, `expect`, `when` style. And that is what I'm used to because I've used REST Assured before.

The interface for using REST Assured is very flexible and now the recommended style seems to be `given`, `when`, `then`, so I could equally have written:

```
@Test
public void aUserCanNotAccessIfNoBasicAuthHeaderUsingGivenWhenThen(){

    given().
            contentType("text/xml").
    when().
            get("http://192.168.17.129/todos.xml").
    then().
            statusCode(401);
}
```

You can see plenty of usage examples on the REST Assured web site:

- github.com/rest-assured/rest-assured/wiki/Usage[75]

I tend not to worry about which of those two conventions to use because I rarely use the REST Assured code in my actual `@Test` methods.

I use the REST Assured code in my lower level abstractions.

[75] https://github.com/rest-assured/rest-assured/wiki/Usage

Abstractions

I try to write code that is readable and is robust in the face of change.

It is possible to view the REST Assured library itself as providing a set of abstractions including:

- HTTP calls,
- Gherkin Given, When, Then,
- Assertions - e.g. in the form of `expect()`,
- JSON parsing,
- XML parsing,
- …

Actually, that is quite a lot of abstractions.

It would be tempting to write all of my `@Test` methods using REST Assured, because then I get assertions, given, when & then, for free, and I don't have to worry about JSON parsing and … etc.

However, if I have `@Test` methods which use the HTTP level abstractions then they will have to change when the API changes e.g. they call specific end points and use specific parameters. That approach would be suitable when I am specifically testing the structure of the API.

For example, if I have some end points on the API, and I want to check that each end point works, then this might be a valid abstraction level. I can mitigate some code changes by making endpoint URLs `String` constants.

Also, if I want to use the API to functionally test the application, and will be making multiple API calls in sequence, then I need to work at a level of abstraction above the HTTP endpoint calls.

In reality I want tests that look a little more like this:

```
@Test
public void aUserCanDeleteAProject(){

    TracksApi api = new TracksApi(TestEnvDefaults.getTestEnv());

    api.createProject("A New Project" +
                         new RandomDataGenerator().randomWord());
    Assert.assertEquals(201,
                        api.getLastResponse().getStatusCode());
    String projectId = new TracksResponseProcessor(
                                api.getLastResponse())
                                    .getIdFromLocation();

    // check we can get it
    api.getProject(projectId);
    Assert.assertEquals(200,
                        api.getLastResponse().getStatusCode());

    // check we can delete it
    api.deleteProject(projectId);
    Assert.assertEquals(200,
                        api.getLastResponse().getStatusCode());

    // check it has been deleted
    api.getProject(projectId);
    Assert.assertEquals(404,
                        api.getLastResponse().getStatusCode());
}
```

The above `@Test` method does use the REST Assured abstractions in the code - the `Response` class from REST Assured is used.

I have created a level of abstraction that maps on to the documented API i.e. `TracksApi` and this has methods which allow me to call the API and receive responses. Sometimes the responses are HTTP responses, sometimes they are domain objects. I always have access to the physical HTTP response via the `getLastResponse` method.

I assert in the `@Test`, but I don't use the REST Assured assertion mechanism, I use the `JUnit` assertions because these maintain a separation between the 'use' of the API, and the 'checking' or 'assertion' of the API results.

This means that if the physical structure of the API changes then my `@Test` code does not have to change, since the functionality that I'm using the API to test does not change. I will always be able to use the API to create, get, and delete Projects. The status code returned might change, but I doubt it.

The end point might change, and that won't impact my `@Test`, but it will require me to change other parts of the abstraction code. The data used to create the `Project` might change, but that probably won't require me to change the `@Test` since I don't really care what the details of the Project are, just that one is created.

This `@Test` isn't perfect.

- I could refactor this so that instead of sending the `createProject` method a 'name', I send it a `Project.newRandomProject()`.
- If the API were larger then having a single `TracksApi` with a long series of methods might not be readable or manageable, I might want to logically organise the API: `api.projects().get(id)` or `api.projects().delete(id)` etc.

But I don't worry about that now because I can amend the API abstractions over time.

Other Examples of REST Assured

I wanted to compare my use of REST Assured with other people so I used the search facility on github.com[76] to look for Java projects using the `rest-assured` library.

- GitHub search for Java projects using rest-assured[77]

I found the following projects which all use REST Assured directly in their `@Test` methods:

- github.com/testvagrant/RESTTests_RestAssured[78]
- github.com/OnBoardInformatics/WebMavenRestAssured[79]
- github.com/GSMADeveloper/RCS-REST-Tests[80]
  - This `GSMADeveloper` project is interesting because it imports the top level `RestAssured` class, rather than statically importing the `given` class. I found a few other personal projects doing this but haven't listed them as I thought one example would suffice.

[76] https://github.com

[77] https://github.com/search?l=Java&q=rest-assured&type=Repositories&utf8=%E2%9C%93

[78] https://github.com/testvagrant/RESTTests_RestAssured

[79] https://github.com/OnBoardInformatics/WebMavenRestAssured

[80] https://github.com/GSMADeveloper/RCS-REST-Tests

  - I also have some examples using this style in the later JSON and XML processing chapter.

I found the following projects which adopted a similar approach to that used in this case study i.e. the use of abstractions on top of REST Assured which are used in the `@Test` methods:

- github.com/moolya-testing/rest-assured[81]

The basic criteria for choosing to list the projects here was that they looked like they were written by companies rather than individuals and they had enough code to make reading them interesting. I haven't included them because I think they are exemplars of how to write automated code. Their inclusion does not mean that I endorse them in any way. But I think it is useful to have examples to read.

REST Assured Related Reading

A quick web search revealed the following resources, if you want to learn more about REST Assured:

- Bas Djkstra's Open Source REST Assured Workshop[82]
  - Bas has open sourced a workshop that has basic REST Assured functionality for pre-emptive Basic Auth, REST Assured assertions, Oath2, `POST`, `GET`, URL Path parameters.
- Joe Colantonio has a few REST Assured blog posts written in a tutorial format:
  - Part 1 Getting Started[83]
  - Part 2 GET[84]
  - part 3 POST[85]
- Code examples on programcreek for REST Assured[86]
- Mark Winteringham has some REST Assured example code in his API Framework project on GitHub[87]

[81] https://github.com/moolya-testing/rest-assured

[82] http://www.ontestautomation.com/open-sourcing-my-workshop-an-experiment/

[83] https://www.joecolantonio.com/2014/02/07/rest-testing-with-java-getting-started-with-rest-assured/

[84] https://www.joecolantonio.com/2014/02/26/rest-testing-with-java-part-two-getting-started-with-rest-assured/

[85] https://www.joecolantonio.com/2014/04/24/rest-assured-how-to-post-a-json-request/

[86] http://www.programcreek.com/java-api-examples/index.php?api=com.jayway.restassured.RestAssured

[87] https://github.com/mwinteringham/api-framework

Summary

Try to explore the API interactively before you automate. This allows you to learn more about the API, and often allows you to experiment very quickly.

Prior to committing strategically to a library or tool, it is worth spending some time tactically automating the API to learn more and identify specific risk areas or problematic areas of the API.

As you automate, do keep thinking about the structure of the code and the abstractions you are using to make sure that you continually refactor to code that is maintainable and readable.

About the Author

Alan Richardson has more than twenty years of professional IT experience, working as a programmer, and at every level of the testing hierarchy from Tester through Head of Testing. Author of the books "Dear Evil Tester", "Selenium Simplified" and "Java For Testers". Alan has also created on-line training courses to help people learn Technical Web Testing and Selenium WebDriver with Java.

Alan works as an independent consultant, helping companies improve their automating and use of agile, and exploratory technical testing.

You can find Alan's writing and training videos on:

- SeleniumSimplified.com[88],
- EvilTester.com[89],
- JavaForTesters.com[90], and
- CompendiumDev.co.uk[91].

Alan posts information and videos regularly to social media on:

- Twitter - @eviltester[92]
- Instagram - @eviltester[93]
- Linkedin - @eviltester[94]
- Youtube - EvilTesterVideos[95]
- Pinterest - @eviltester[96]

To contact Alan for custom training or consultancy, visit:

- compendiumdev.co.uk/contact[97]

[88] http://SeleniumSimplified.com
[89] http://EvilTester.com
[90] http://javafortesters.com
[91] http://compendiumdev.co.uk
[92] https://twitter.com/eviltester
[93] https://www.instagram.com/eviltester
[94] https://uk.linkedin.com/in/eviltester
[95] https://www.youtube.com/user/EviltesterVideos
[96] https://uk.pinterest.com/eviltester/
[97] http://compendiumdev.co.uk/page/contact_us

Thanks for Reading This Sample

Thanks for reading this sample of my “Automating and Testing REST APIs” book.

The sample has the general introductory information to help you get started with testing REST APIs, specifically the Tracks API.

And the full book has so much more.

The full book is a case study so it contains:

- working code
- practical examples
- thought processes
- step by step analysis

All against a real system.

We have full analysis of exploring the Tracks API with cURL. Showing cURL commands for many different HTTP REST calls.

Then we build on that to figure out how to automate the Tracks API.

We have to overcome the hurdle that the Tracks API does not support the creation of users. So how do we automate it?

I explain the concept of APP as API with code to illustrate the concept in detail.

Full explanations of three different random data creation strategies:

- one very simple
- one that relies on scraping data from a live site
- one that uses pure code to generate data

I explain how to use REST Assured so if you want to use that in production you’ll have enough information to go quite far.

I also explain the unconventional way in which I use it for strategically automating with abstraction layers to make the test code maintainable.

I also provide explanations of what I would do next - what refactoring to undertake.

And at that point in the book you should have enough information to carry that forward as an exercise so you can really build on the information in the book in a very practical way by taking the code forward.

The code and coverage in this case study was created for workshops where I taught how to test a REST API. I pulled on all my previous REST API testing experience, and I learned a bunch of stuff as I conducted the training, and I've tried to explain it fully in this book.

I'm sure you'll learn something you didn't know before, and I hope you're eager enough to move forward with your API testing that you'll pick up the full copy of the text.

- compendiumdev.co.uk/page/tracksrestapibook[98]

[98] http://compendiumdev.co.uk/page/tracksrestapibook