



TESTINGMIND

ANNUAL SOFTWARE TESTING SYMPOSIUM

TALK: BASIC POSTMAN API TESTING

PRESENTER: ARNOLD MILLER

#STS18

Denver, Colorado, USA
28 August 2018
Arnold.miller0@gmail.com

BASIC POSTMAN API TESTING

Presenter: Arnold Miller, Senior Software QA-Tester

- Lessons learned using the basic Postman API tool to automate ReST API messages (micro services) validation along with manual checking the API supported Web, Mobile, other applications.
- **First, Basic (Free) Postman API features**
- Second, API Request/Response checks
- Third, Runner and Command Line Execution
- Fourth, Web spot checks of API Failures
- Fifth, Conclusion and Summary

POSTMAN API OVERVIEW

- Postman API, <https://www.getpostman.com>, Java Script internet based testing tool that automates building, sending, receiving and evaluating ReST API messages (**Plus**)
 - Able to export to file system and Import from file system (**Plus**)
 - Developers use and like more than cURL command line (**Plus**)
 - Internet documents, videos, training and support (**Plus**)
- Newman, <https://www.npmjs.com/package/newman>, command line tool to execute Postman API tests with JUNIT results for CI/CD (Jenkins, TeamCity, Scripts) tool integration (**Plus**)

VALIDATE API AND WEB OVERVIEW

- Public SaaS ReST API messages and supported Web App.
- Web App: <http://www.dishanywhere.com/>
- Provide Dish Network subscribers with Shows and Movies videos for on demand watching. Along with
 - Promote up-sales and new customers
 - Subscriber's Guide and Network Information
 - Subscriber's DVR Remote Recording and Watching on demand
 - Sport Team and League Information
- The Dish Network on-demand ReST APIs controls about 90% of what this Web App and similar Mobile App displays, along with providing data to other applications.

SAMPLE WEB SITE



POSTMAN API REQUEST GROUPS (**PLUS**)

- **API requests grouped in Collections, Folders, Sub-Folders**
- These all have Description, Authorization, Pre-requests Script, Test Script and Requests
- **Collection:** Top level group also has Variables
 - **Request** (Send, Request, Get Response)
 - **Folder:** Next Level group
 - **Request** (Send, Request, Get Response)
 - **Sub-Folders:** Next level groups
 - **Request** (Send, Request, Get Response)

COLLECTION, FOLDER, SUB-FOLDER

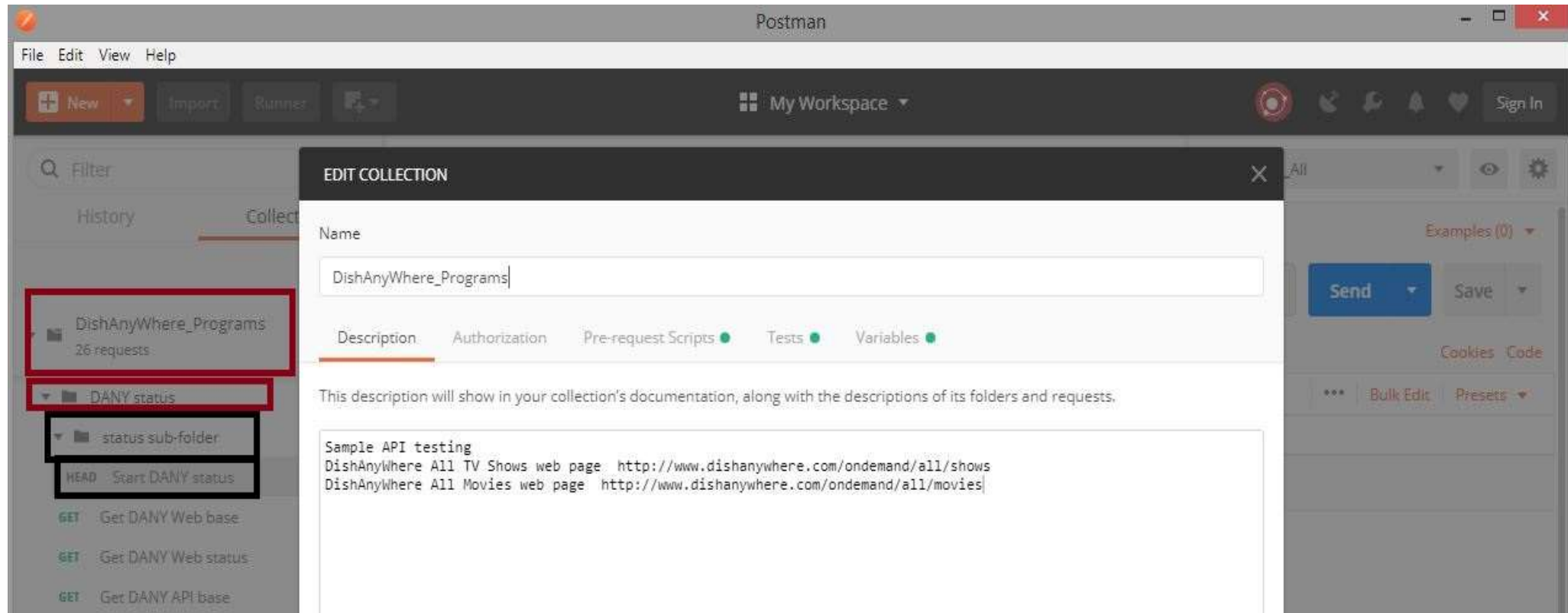
Collection: DishAnyWhere_Programs

Folder: DANY status

Sub: status sub-folder

Req: HEAD Start DANY status

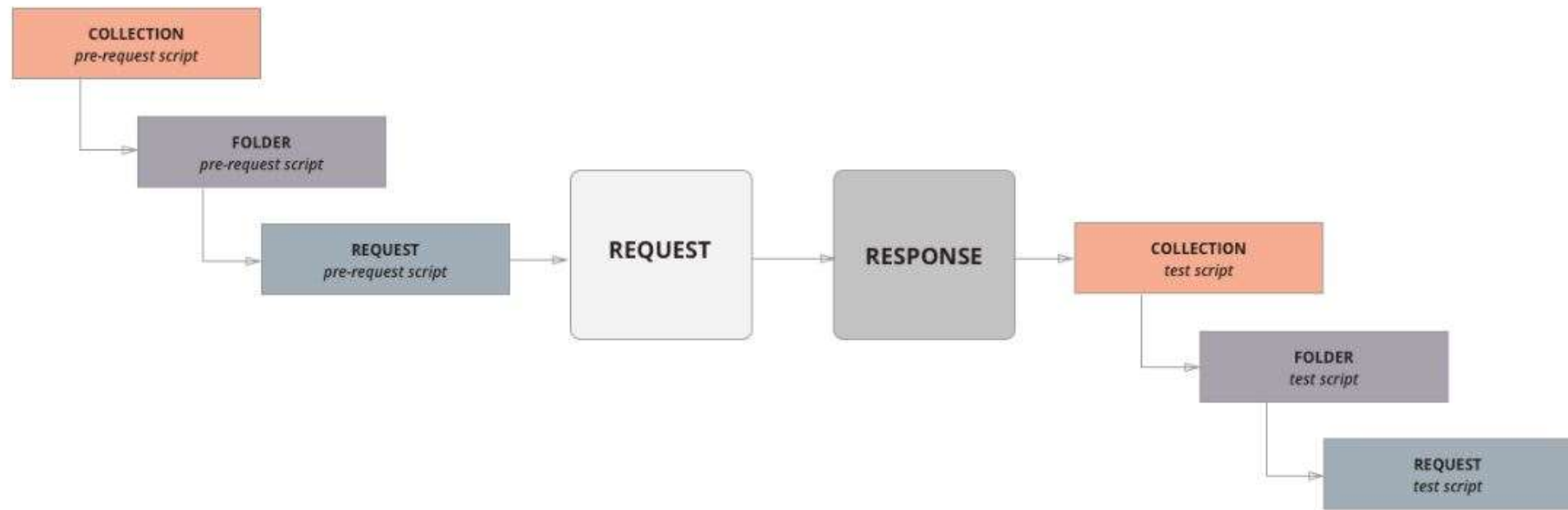
Req: Get DANY Web base



POSTMAN SCRIPT EXECUTION ORDER

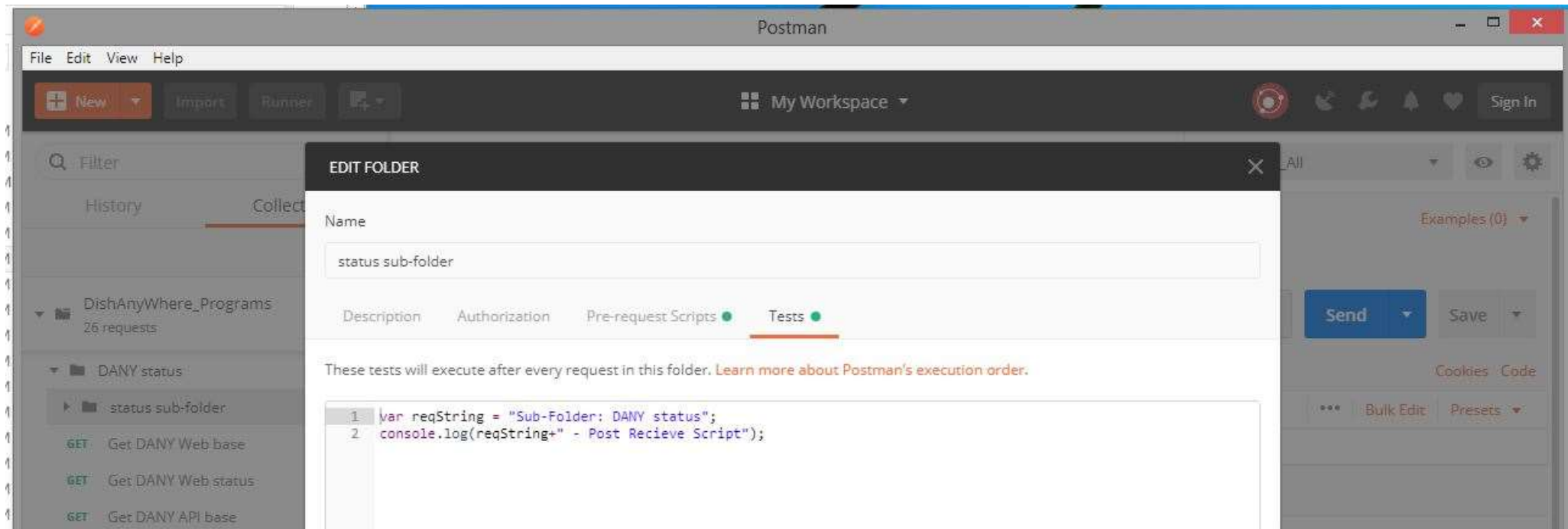
Per Request Script Execution Order:

https://www.getpostman.com/docs/v6/postman/scripts/intro_to_scripts



SUB-FOLDER TEST SCRIPT

PLUS: `console.log(<string>)` sends <string> to **Postman Console**



POSTMAN OUTPUT CONSOLE (**PLUS**)

- Besides `console.log(<string>)` also has
 - Request: Method, URL, Path, Query parameters
 - Request Headers; Response Headers; Response Body



API REQUEST AREA

- Method, URI, arguments
 - Authorization, Headers, Body, Pre-request script, Tests script

The screenshot displays the Postman API client interface. On the left, a sidebar shows a collection named 'DishAnyWhere_Programs' with 26 requests. Under the 'DANY status' folder, there is a 'status sub-folder' containing a 'HEAD Start DANY status' request. The main panel shows the details of this request, including the method 'HEAD' and the URL '{{\$e_host_name_web}}.dishanywhere.com'. The 'Tests' tab is selected, showing a JavaScript script for schema validation. The script includes comments and code for loading helpers, checking response codes, and logging request details.

```
1 // load helper functions
2 var helpers = eval(globals.loadHelpers);
3 var _ = require('lodash');
4
5 var rspCode = Number(pm.variables.get("c_ok_code"));
6 var reqString = request.name + " ";
7 console.log("Request: "+reqString+" - Test Script, response msec " + responseTime);
8 helpers.checkStatusCode(rspCode, reqString + ": ");
9
10 // need expected code to do schema validation
11 if (responseCode.code !== rspCode) {
12     console.log(reqString + ", responseCode=" + responseCode.code);
13     return;
14 }
15
```

API RESPONSE AREA

- **Response Body** display in different formats: HTML, XML, JSON, Text.
 - Default format based on response header: Content Type value
- **Response Cookies, Headers**
- **Response Test Results**
 - Each test compare result with output tests with AssertionError when False
 - Via tests[<output string>] = <Boolean>

The screenshot displays an API testing tool interface. On the left, a sidebar lists various API endpoints under categories like 'DANY Programs', 'DANY TV Shows', and 'DANY Movies'. The main panel shows the 'Test Results' tab for a specific test. The test is labeled 'GET Get DANY API status' and has a status of '200 OK', a time of '652 ms', and a size of '5.88 KB'. The test results are listed in a table with columns 'All', 'Passed', 'Skipped', and 'Failed'. All tests passed, and the results are as follows:

Test Result
PASS Get DishAnywhere Web base: Status code: '200' eq '200'
PASS Chai to.equal responseCode
PASS chai expect: AssertionError: expected 200 to equal 404
PASS Chai should.equal responseCode
PASS chai should: AssertionError: expected 200 to equal 404
PASS cheerio <head> has 12 <meta> tags
PASS <meta>[2] description: Watch your favorite TV shows and movies free online with your DISH subscription. Set recordings directly to your DISH DVR
PASS <meta>[3] keywords: DISH, Dish Anywhere, Dishanywhere, dishanywhere, Dish Online, movies, shows, online, watch, stream, on demand, ondemand, record, show, movie, Sling, Sling Media, Sling Box, Dish Network, TV, television, DVR, Guide, Live TV, DISH Movie Pack

API SCRIPT VARIABLES (**PLUS**)

API Script Variables and their priority

https://www.getpostman.com/docs/v6/postman/environments_and_globals/variables

- **Global, Collection, Environment** : String type (**Minus**)
 - Store via to-string functions; Use via from-string functions
- **Global**: share values between Pre-Request and Test scripts
- **Collection**: Initial value via Collection Set,
 - Only in its Collection (folders, subfolders and request)
- **Environment**: Initial set via Environment. (**Minus**)
 - Have multiple Environments for different test execution types
- **Local**: Any JavaScript types only in its Pre-Request or Test script
- **Data**: only Execution Iteration via Runner or Command Line

INITIAL VARIABLE VALUES (PLUS)

EDIT COLLECTION

Name

DishAnywhere_Programs

Description

Authorization

Pre-request Scripts

Tests

Variables

These variables are specific to this collection and its requests. [Learn more about collection variables.](#)

	VARIABLE	INITIAL VALUE ⓘ	CURRENT VALUE ⓘ	...	Persist All	Reset All
<input checked="" type="checkbox"/>	c_ok_code	200	200			
<input checked="" type="checkbox"/>	c_bad_code	404	404			
	Add a new variable					

MANAGE ENVIRONMENTS

Environment Name

Dish_Prod_All

	VARIABLE	INITIAL VALUE ⓘ	CURRENT VALUE ⓘ	...	Persist All	Reset All
<input checked="" type="checkbox"/>	e_branch_api	origin/18.3.2	origin/18.3.2			
<input checked="" type="checkbox"/>	e_host_name_web	www	www			
<input checked="" type="checkbox"/>	e_branch_web	18.3.2	18.3.2			
<input checked="" type="checkbox"/>	e_exec_tests	programs	programs			
<input checked="" type="checkbox"/>	e_exec_type	all	all			
	Add a new variable					

GENERATE CODE SNIPPETS (**PLUS**)

Generate API Request Code Snippets

- Uses current variable key values
- Formats: HTTP, cURL, C#, Java, JavaScript, PHP, Python, Ruby

Request: Get DANY Web base

```
GET {{e_host_name_web}}.dishanywhere.com
```

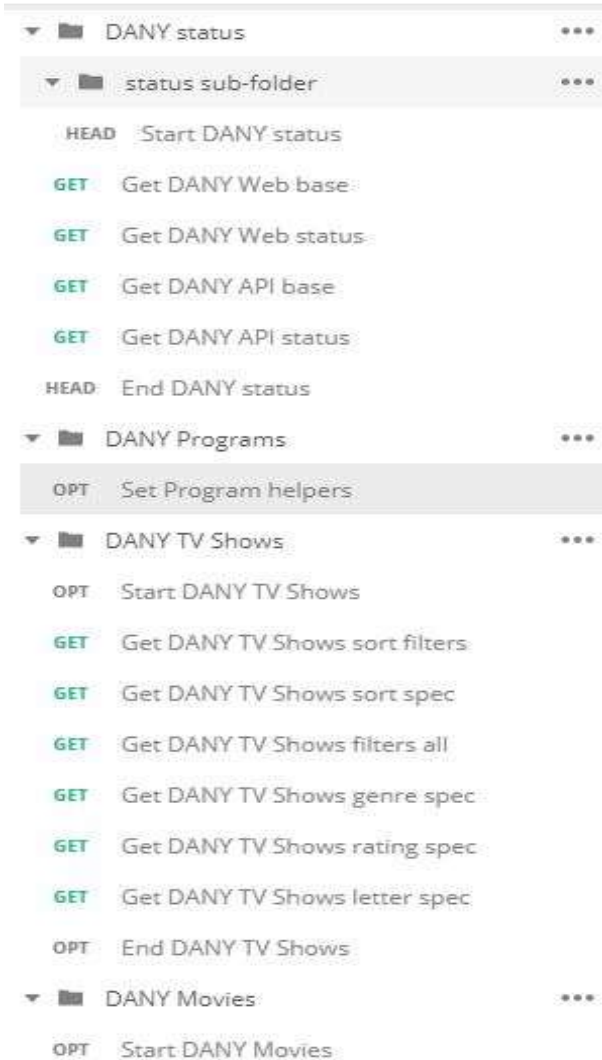
```
curl -X GET \  
  http://www.dishanywhere.com \  
  -H 'Cache-Control: no-cache' \  
  -H 'Postman-Token: dba1dab9-7204-4585-ad49-  
59b094e11f08'
```

REQUEST EXECUTION ORDER (**PLUS**)

https://www.getpostman.com/docs/v6/postman/scripts/branching_and_looping

- Normal API Request order is linear from top to bottom Request in Collection, Folder, Sub-Folder
 - State-machine like execution order
 - Change this order via `postman.setNextRequest(<request_name>)`
 - Where `<request_name>` the name or ID of the subsequent request and the collection runner will take care of the rest
 - May be used in the pre-request or the test script. In case of more than one assignment, the last set value is used.
 - When `postman.setNextRequest()` is absent in a request, the collection runner defaults to linear execution and moves to the next request
 - Stop execution, when `<request_name>` is null

CHANGE EXECUTION ORDER (PLUS)



Request: Set Program helpers (Test Script)

```
// set next test group via environment variable
switch(environment.e_exec_tests.toLowerCase()) {
  case "status":
    postman.setNextRequest(null);
    break;
  case "movies":
    postman.setNextRequest("Start DANY Movies");
    break;
  case "programs":
    postman.setNextRequest("Start DANY TV Shows");
    break;
  case "shows":
    postman.setNextRequest("Start DANY TV Shows");
    break;
  default:
    postman.setNextRequest(null);
    break;
}
```

BUILT-IN JAVA SCRIPT PACKAGES (**PLUS**)

https://www.getpostman.com/docs/v6/postman/scripts/postman_sandbox_api_reference

- **chai** for BDD expect and should validation
 - **Request:** GET DANY Web base (Test Script)
- **cheerio** for HTML paring and validation
 - **Request:** GET DANY Web base (Test Script)
- **lodash** for numbers strings arrays and objects utilities
 - **Request:** GET DANY API base (Pre-Request Script, Test Script)
- **momentum** for date-time utilities
 - **Folder:** DANY status (Pre-Request Script, Test Script)
- **tv4** for JSON schema validation
 - **Helper Function:** **tv4** to define verify schema, pattern-validate functions
 - **Request:** Get DANY API base (Pre-Request Script, Test Script)

CREATE, USE COMMON FUNCTIONS

Helper functions definition via Set Global variables (Minus)

Request: Start DANY status (Pre-request script)

```
// define helper functions
postman.setGlobalVariable("loadHelpers", function loadHelpers() {
  let helpers = {};
  helpers.CompareSame = function testCompare(compString, compA, compB) {
    tests[compString + ": '" + compA + "' is '" + compB + "'] = compA == compB; };
  helpers.CompareEqual = function testEqual(compString, compA, compB) {
    tests[compString + ": '" + compA + "' is '" + compB + "'] = compA === compB; };
  . . .
  // check response status code value
  helpers.checkStatusCode = function verifyStatusCode(
    statusCode, msgString = "") {
    helpers.CompareEqual(msgString + " Status code", responseCode.code, statusCode); };
  . . .
  // ...additional helpers
  return helpers;
} + '; loadHelpers();');
```

CREATE, USE COMMON FUNCTIONS

Helper functions usage via eval Global variables (Minus)

Request: Start DANY status (Test script)

```
// load helper functions
var helpers = eval(globals.loadHelpers);
var _ = require('lodash');
var rspCode = Number(pm.variables.get("c_ok_code"));
var reqString = request.name + " ";
console.log("Request: "+reqString+" - Test Script, response msec "
    + responseTime);
helpers.checkStatusCode(rspCode, reqString + ": ");
```

BASIC POSTMAN API TESTING

- First, Basic (Free) Postman API features
- **Second, API Request/Response checks**
- Third, Runner and Command Line Execution
- Fourth, Web spot checks of API Failures
- Fifth, Conclusion and Summary

CHECK DISH WEB'S BRANCH RELEASE

GET <http://www.dishanywhere.com/>

Contain web html meta tag with release information

```
<meta content="Dish Anywhere" name="dany" data-branch="18.3.3" data-hash=". . . ">
```

Request: Get DANY Web base (Test Script)

```
var webDataBranch = metaDanyHTML.attr('data-branch');  
helpers.CompareSame("Web Deploy Branch: ", webDataBranch,  
environment.e_branch_web);  
postman.setGlobalVariable("g_Web_Branch", webDataBranch);
```

CHECK DISH WEB'S API URL PATH

GET www.dishanywhere.com/health/config_check

Contains web release/build information

```
"branch": "18.3.3", "hash": ". . . ."
```

Contains URL for its ReST API messages (micro services) path

```
"radish_url": "http://radish.dishanywhere.com/",
```

Request: Get DANY Web status (Test Script)

```
var webBranch = globals.g_Web_Branch;  
helpers.CompareSame(reqString + " branch:", responseBody.git.branch,  
webBranch);  
.  
.  
.  
var danyBaseURL = "http://" + environment.e_host_name_web +  
".dishanywhere.com";  
helpers.CompareSame(sesString + ".dany_base_url:",  
session.dany_base_url, danyBaseURL);
```

CHECK DISH API BRANCH RELEASE

GET http://radish.dishanywhere.com/health/config_check

Contains API release/build information

```
"branch": "origin/18.3.3", "hash": " . . ."
```

Request: Get DANY API status (Test Script)

```
var respBody = JSON.parse(responseBody);  
var apiBranch = environment.e_branch_api;  
helpers.CompareSame(reqString + " branch:", respBody.git.branch,  
apiBranch);
```


DISH WEB PROGRAMS

Displays Subscriber viewable (unlocked) TV Shows or Movies

- GET <http://www.dishanywhere.com/ondemand/all/shows>
- GET <http://www.dishanywhere.com/ondemand/all/movies>

1. Get Sort By Values (API Request)

- GET <http://radish.dishanywhere.com/v20/dol/shows/sorts.json>
- GET <http://radish.dishanywhere.com/v20/dol/movies/sorts.json>

2. Get Filter (Genres, Ratings, Starting Letters) Values (API Request)

- GET <http://radish.dishanywhere.com/v20/dol/shows/filters.json>
- GET <http://radish.dishanywhere.com/v20/dol/movies/filters.json>

3. Display Programs via specific Sort and Filter Value (API Request)

DISH API PROGRAMS SORTS

GET <http://radish.dishanywhere.com/v20/dol/shows/sorts.json>

```
[{"name": "What's Hot",      "slug": "whats_hot"},
 {"name": "Recently Added", "slug": "recent"},
 {"name": "Title",          "slug": "name"},
 {"name": "Most Popular",   "slug": "most_popular"}
]
```

GET <http://radish.dishanywhere.com/v20/dol/movies/sorts.json>

```
[{"name": "Release Date",      "slug": "date"},
 {"name": "Critics Rating",    "slug": "rating"},
 {"name": "Tomatometer Rating", "slug": "tomatometer"},
 {"name": "Title",             "slug": "name"},
 {"name": "Most Popular",      "slug": "most_popular"}
]
```

DISH API PROGRAMS FILTERS

GET <http://radish.dishanywhere.com/v20/dol/shows/filters.json>

```
{
  "genres": [
    { "name": "All", "slug": "all" },
    { "name": "Action", "slug": "action" },
    ... ,
    { "name": "Thriller", "slug": "thriller" }
  ],
  "ratings": [
    { "name": "All", "slug": "all" },
    { "name": "TVY", "slug": "tvvy" },
    ... ,
    { "name": "TV14", "slug": "tv14" },
    { "name": "TVMA", "slug": "tvma" }
  ],
  "a-z": [
    { "name": "A", "slug": "a" },
    { "name": "B", "slug": "b" },
    ... ,
    { "name": "Z", "slug": "z" },
    { "name": "#", "slug": "123" }
  ]
}
```

GET <http://radish.dishanywhere.com/v20/dol/movies/filters.json>

```
{
  "genres": [
    { "name": "All", "slug": "all" },
    { "name": "Action", "slug": "action" },
    ... ,
    { "name": "Thriller", "slug": "thriller" }
  ],
  "ratings": [
    { "name": "All", "slug": "all" },
    { "name": "G", "slug": "g" },
    ... ,
    { "name": "NC17", "slug": "nc17" },
    { "name": "NR", "slug": "nrao" }
  ],
  "a-z": [
    { "name": "A", "slug": "a" },
    { "name": "B", "slug": "b" },
    ... ,
    { "name": "Z", "slug": "z" },
    { "name": "#", "slug": "123" }
  ]
}
```

DISH DISPLAY PROGRAMS

Display Programs via specific Sort and Filter Values (API Request)

Via Only by Sort Value

- GET <http://radish.dishanywhere.com/v20/dol/shows.json?sort=name>
- GET <http://radish.dishanywhere.com/v20/dol/movies.json?sort=name>

Via Only by Filter Genre Value

- GET <http://radish.dishanywhere.com/v20/dol/shows.json?genres=newscast>
- GET <http://radish.dishanywhere.com/v20/dol/movies.json?genres=drama>

Via Only by Filter Rating Value

- GET <http://radish.dishanywhere.com/v20/dol/shows.json?ratings=tv14>
- GET <http://radish.dishanywhere.com/v20/dol/movies.json?ratings=r>

Via Only by Filter Starting Letter Value

- GET <http://radish.dishanywhere.com/v20/dol/shows.json?a-z=p>
- GET <http://radish.dishanywhere.com/v20/dol/movies.json?a-z=s>

CHECK DISH API PROGRAMS

General API Response:

- Response Code Ok
- Positive item count (Allow Zero for special cases)
- **Program General:** (General TV Shows, Movies)
 - **Name** has positive length
 - **Type** is Program with **Kind** as Show or Movies via Request
 - **Description** at least 5 characters long
 - **Rating** array values same as **Rating List** of array values
 - Has at least 1 **Genre**
 - Has at least 1 **Network (Channel)**
- **Specific Sort, Genre, Rating, Starting Letter**(Specific Sort, Filter)
 - **Sort:** Items in request Order (only **Title** sorted request)
 - **Genre:** Each Item's **Genre** Array has the Filter **Genre**
 - **Rating:** Each Item's **Rating** Array has the Filter **Rating**
 - **Starting Letter:** Each item's **Name** begins with the Filter **Letter**

BASIC POSTMAN API TESTING

- First, Basic (Free) Postman API features
- Second, API Request/Response checks
- **Third, Runner and Command Line Execution**
- Fourth, Web spot checks of API Failures
- Fifth, Conclusion and Summary

POSTMAN COLLECTION RUNNER (PLUS)

- Execute Collection, Folder, Sub-Folder Requests with a specific Environment with for N iteration and Data-file.
- Has Recent Runs with
 - Iteration counter
 - API Request
 - Pass/Failed Test Results
 - Filter for Passed, Failed or Both

COLLECTION RUNNER: SETUP (PLUS)

The screenshot displays the 'Collection Runner' application window. The interface is divided into two main sections: a left sidebar for configuration and a right pane for recent runs.

Left Sidebar (Configuration):

- Choose a collection or folder:** A search bar and a list of collections including 'DishAnywhere_Programs', 'DANY status', 'DANY Programs', 'DANY TV Shows', 'DANY Movies', and 'DANY Others'.
- Environment:** A dropdown menu currently set to 'Dish_Prod_sample'.
- Iterations:** A text input field with the value '1'.
- Delay:** A text input field with the value '0' and a unit selector set to 'ms'.
- Log Responses:** A dropdown menu set to 'For all requests' with an information icon.
- Data:** A 'Select File...' button.
- Keep variable values:** An unchecked checkbox with an information icon.
- Run Button:** A large blue button labeled 'Run DishAnyWher...'.

Right Pane (Recent Runs):

- Header:** 'Recent Runs' with a search bar 'Type to Filter' and an 'Import Test Run' button.
- Table of Recent Runs:**

Collection Name	Status	Time
DishAnywhere_Programs Dish_Prod_All	17 FAILED	Yesterday, 6:14 am
DishAnywhere_Programs Dish_Prod_sample	5 FAILED	29 Jul, 2018
DishAnywhere_Programs Dish_Prod_sample	7 FAILED	29 Jul, 2018
DishAnywhere_Programs Dish_Prod_sample	4 FAILED	29 Jul, 2018
DishAnywhere_Programs Dish_Prod_sample	7 FAILED	29 Jul, 2018

COLLECTION RUNNER: EXECUTING

The screenshot shows the 'Collection Runner' application interface. At the top, there's a menu bar with 'File', 'Edit', 'View', and 'Help'. Below it, a dark header bar contains 'Collection Runner', 'Run Results', 'My Workspace', and buttons for 'Run In Command Line' and 'Docs'. The main area displays a progress indicator at 73% and the title 'DishAnywhere_Programs' with a subtitle 'Dish_Prod_sample' and the status 'Running 1 iteration...'. To the right are 'Stop Run' and 'Pause' buttons. Below this, a section for 'Iteration 1' lists test results. A table-like structure shows the following:

Result	Test Name	Details
PASS	<head><meta>[name="dany"].html().length: 0, html():	
PASS	<head><meta>[name="dany"].text().length: 0, text():	
FAIL	Web Deploy Branch: : '18.3.3' is '18.3.2'	
PASS	Web Deploy Branch: 18.3.3, Hash: eef88e3e1916ee49e26f2675b8459df418d45f5	
PASS	<head><title> HTML.length: 28, HTML: <title>Dish Anywhere</title>	
PASS	<head><title>.html().length: 13, html(): Dish Anywhere	
PASS	<head><title>.text().length: 13, text(): Dish Anywhere	
PASS	<head><title>.text(): 'Dish Anywhere' is 'Dish Anywhere'	
PASS	<body>[id="mainContainer"] HTML.length: 402, HTML: <div id="mainContainer"> <div id="applicationSkinLeft" class="applicationSkin"></div> <div id="applicationSkinGradientLeft...	
PASS	<body>[id="mainContainer"].html().length: 372, html(): <div id="applicationSkinLeft" class="applicationSkin"></div> <div id="applicationSkinGradientLeft" class="applicationSkinGr...	
PASS	<body>[id="mainContainer"].text().length: 52, text():	
GET	Get DANY Web status: www.dishanywhere.com...	...NY status / Get DANY Web status 200 OK 167 ms 5.432 KB
PASS	Get DishAnywhere Web Health: Status code: '200' eq '200'	

At the bottom, there's a status bar showing 'Sending Request'.

COLLECTION RUNNER: DONE (PLUS)

The screenshot displays the 'Collection Runner' application window. The top menu bar includes 'File', 'Edit', 'View', and 'Help'. Below the menu, there are tabs for 'Collection Runner' and 'Run Results', along with a 'My Workspace' dropdown and buttons for 'Run In Command Line' and 'Docs'. The main area shows a summary for 'DishAnyWhere_Programs' with a 'Dish_Prod_sample' label. It indicates '2.29K PASSED' and '4 FAILED' tests. Buttons for 'Run Summary', 'Export Results', 'Retry', and 'New' are visible. The test results are listed under 'Iteration 1', showing a mix of 'PASS' and 'FAIL' outcomes. A specific failure is highlighted: 'Web Deploy Branch: : '18.3.3' is '18.3.2''. Below the test results, a 'GET' request to 'www.dishanywhere.com...' is shown with a status of '200 OK', a response time of '167 ms', and a size of '5.432 KB'. The bottom of the window shows the details of the 'Get DishAnyWhere Web Health' test, which passed with a status code of '200' and a branch of '18.3.3'.

Collection Runner

File Edit View Help

Collection Runner Run Results My Workspace Run In Command Line Docs

2.29K PASSED 4 FAILED DishAnyWhere_Programs Dish_Prod_sample just now

Run Summary Export Results Retry New

Iteration 1

- PASS <head><meta>[name="dany"].html().length: 0, html():
- PASS <head><meta>[name="dany"].text().length: 0, text():
- FAIL Web Deploy Branch: : '18.3.3' is '18.3.2'
- PASS Web Deploy Branch: 18.3.3, Hash: eef88e3e1916ee49e26f2675b8459df418d45f5
- PASS <head><title> HTML.length: 28, HTML: <title>Dish Anywhere</title>
- PASS <head><title>.html().length: 13, html(): Dish Anywhere
- PASS <head><title>.text().length: 13, text(): Dish Anywhere
- PASS <head><title>.text(): 'Dish Anywhere' is 'Dish Anywhere'
- PASS <body>[id="mainContainer"] HTML.length: 402, HTML: <div id="mainContainer"> <div id="applicationSkinLeft" class="applicationSkin"></div> <div id="applicationSkinGradientLeft...
- PASS <body>[id="mainContainer"].html().length: 372, html(): <div id="applicationSkinLeft" class="applicationSkin"></div> <div id="applicationSkinGradientLeft" class="applicationSkinGr...
- PASS <body>[id="mainContainer"].text().length: 52, text():

GET Get DANY Web status www.dishanywhere.com... ..NY status / Get DANY Web status 200 OK 167 ms 5.432 KB

PASS Get DishAnyWhere Web Health: Status code: '200' eq '200'

PASS Get DishAnyWhere Web Health branch: '18.3.3' is '18.3.3'

COLLECTION RUNNER: FAIL (PLUS)

The screenshot shows the 'Collection Runner' application interface. At the top, there's a menu bar with 'File', 'Edit', 'View', and 'Help'. Below it, a dark header bar contains 'Collection Runner', 'Run Results', 'My Workspace', and buttons for 'Run In Command Line' and 'Docs'. The main area displays test results for 'DishAnyWhere_Programs' (Dish_Prod_sample) run 'just now'. It shows a summary with '2.15K PASSED' and '6 FAILED'. Buttons for 'Run Summary', 'Export Results', 'Retry', and 'New' are present. The test results are listed in a table under 'Iteration 1'.

Method	Test Name	URL	Status	Response Time	Response Size
GET	Get DANY Web base	www.dishanywhere.com	200 OK	115 ms	15.993 KB
FAIL	Web Deploy Branch: : '18.3.3' is '18.3.2'				
GET	Get DANY API base	radish.dishanywhere.com	200 OK	187 ms	196 B
FAIL	Get DishAnyWhere API Base 'sge' status: 'red' is 'green'				
GET	Get DANY API status	radish.dishanywhere.co...	200 OK	85 ms	507 B
FAIL	Get DishAnyWhere API Health branch: 'origin/18.3.3' is 'origin/18.3.2'				
GET	Get DANY TV Shows genre spec	radish.dishanywhere.co...	200 OK	113 ms	22 B
FAIL	Get DANY TV Shows genre 'Thriller' via search thriller: Total: '0' gt '0'				
GET	Get DANY TV Shows letter spec	radish.dishanywhere.co...	200 OK	111 ms	22 B
FAIL	Get DANY TV Shows letter '#' via search 123: Total: '0' gt '0'				
GET	Get DANY Movies letter spec	radish.dishanywhere.co...	200 OK	149 ms	58.01 KB
FAIL	a-z=q; movie[4] name=¿Qué le dijiste a Dios?; name[0]: '¿' in [q,Q]				

NEWMAN COMMAND LINE (PLUS)

Execute Collection Requests with a specific Environment generating reports
(Unix/Linux Shell version)

```
newman run $postman_run -e $post_out_env -r junit,cli --timeout-request  
$req_timeout
```

Have Problem maintaining many Postman Environment files (**Minus**)

- Branch Build/Release ID (Web, API, tools)
- What APIs to test (Status, Shows, Movies, Networks)
- Where is API URL target (Production, Pre-Prod, Beta, SQA-test)
- Execution type (Smoke, Install, Feature, Regression)

Noticed: Postman Environment file are text editor JSON readable

```
"values": [  
  {"enabled": true, "key": "e_branch_web", "value": "18.3.2", "type": "text"},  
  {"enabled": true, "key": "e_branch_api", "value": "origin/18.3.2", "type": "text"},  
  . . . ]
```

NEWMAN COMMAND LINE

Solution to maintaining many Postman Environment files (**Plus**)

Write program that inputs a Postman Environment file and CSV Key-Value pair file
Outputs a Postman Environment File with changing any found Key's to have their pair value

Example CSV Key-Pair file

```
row,Variable,Value,Meaning
1,e_branch_web,18.3.0,web version
2,e_branch_api,origin/18.3.0,api version
```

Output Postman Environment File

```
"values": [
  {"enabled": true, "key": "e_branch_web", "value": "18.3.0", "type": "text"},
  {"enabled": true, "key": "e_branch_api", "value": "origin/18.3.0", "type": "text"},
  . . . ]
```

Found easier to maintain many CSV Key-Pair files (**Plus**)

NEWMAN EXECUTING (PLUS)

Get DANY API status

```
┌  
| 'Collection: DANY - Pre Request Script'  
| 'Folder: before Get DANY API status - Pre Request  
|   Script at 2018-08-12 20:09:14.759 -06:00'  
└
```

GET radish.dishanywhere.com/health/config_check [200 OK, 1.02KB, 124ms]

```
┌  
| 'Collection: DANY - Post Receive Script'  
| 'Folder: Get DANY API status after - Post Receive Script at 2018-08-12 20:09:14.926 -  
06:00'  
| 'Folder: Get DANY API status after - milliseconds since PreReq 167'  
└
```

✓ Get DishAnyWhere API Health: Status code: '200' eq '200'

3. Get DishAnyWhere API Health branch:: 'origin/18.3.3' is 'origin/18.3.0'

NEWMAN SUMMARY RESULTS (PLUS)

Postman start at Sun, Aug 12, 2018 8:09:08 PM

	executed	failed
iterations	1	0
requests	6	0
test-scripts	19	0
prerequisite-scripts	19	0
assertions	47	3
total run duration: 2s		
total data received: 19.23KB (approx)		
average response time: 134ms		

Postman ended at Sun, Aug 12, 2018 8:09:15 PM

NEWMAN FAILURE DETAILS (PLUS)

Used: dish_chg_vars.csv

Postman start at Sun, Aug 12, 2018 8:09:08 PM

#	failure	detail
1.	AssertionError	Web Deploy Branch: : '18.3.3' is '18.3.0' expected false to be truthy at assertion:14 in test-script inside "DANY status / Get DANY Web base"
2.	AssertionError	Get DishAnyWhere API Base 'sge' status: 'red' is 'green' expected false to be truthy at assertion:3 in test-script inside "DANY status / Get DANY API base"
3.	AssertionError	Get DishAnyWhere API Health branch:: 'origin/18.3.3' is 'origin/18.3.0' expected false to be truthy at assertion:1 in test-script inside "DANY status / Get DANY API status"

Postman ended at Sun, Aug 12, 2018 8:09:15 PM

NEWMAN JUNIT XML RESULT FILE (PLUS)

```
<testcase name="Web Deploy Branch: : '18.3.1' is '18.1.1'" time="0.282"
classname="JUnitXmlReporter.constructor">
  <failure type="AssertionFailure">
    <![CDATA[Failed 1 times.]]>
    <![CDATA[Collection JSON ID: fdeb44ba-76ea-424b-9a8c-39645bca44b9.]]>
    <![CDATA[Collection name: DishAnyWhere_Programs.]]>
    <![CDATA[Request name: DANY status / Get DANY Web base.]]>
    <![CDATA[Test description: Web Deploy Branch: : '18.3.1' is '18.1.1'.]]>
    <![CDATA[Error message: expected false to be truthy.]]>
    <![CDATA[Stacktrace: AssertionError: expected false to be truthy
at Object.eval sandbox-script.js:15:2).]]>
  </failure>
</testcase>
```

BASIC POSTMAN API TESTING

- First, Basic (Free) Postman API features
- Second, API Request/Response checks
- Third, Runner and Command Line Execution
- **Fourth, Web spot checks of API Failures**
- Fifth, Conclusion and Summary

API PROGRAMS SAMPLE FAILURES

Used: dish_progs_smp_vars.csv

Failures not reported on previous slides

Postman start at Sun, Aug 12, 2018 8:24:26 PM

#	failure	detail
3.	AssertionError	a-z=q; show[4] name=¿Quién Da Más?; name[0]: '¿' in [q,Q]
4.	AssertionError	a-z=q; show[5] name=¿Quién da más?; name[0]: '¿' in [q,Q]
5.	AssertionError	sort=name; movie[8] name=1 Message; desc.length: '0' ge '5'
6.	AssertionError	a-z=q; movie[4] name=¿Qué le dijiste a Dios?; name[0]: '¿' in [q,Q]

Postman ended at Sun, Aug 12, 2018 8:25:17 PM

API PROGRAMS ALL FAILURES

Used: dish_progs_all_vars.csv

Failures not reported on previous slides

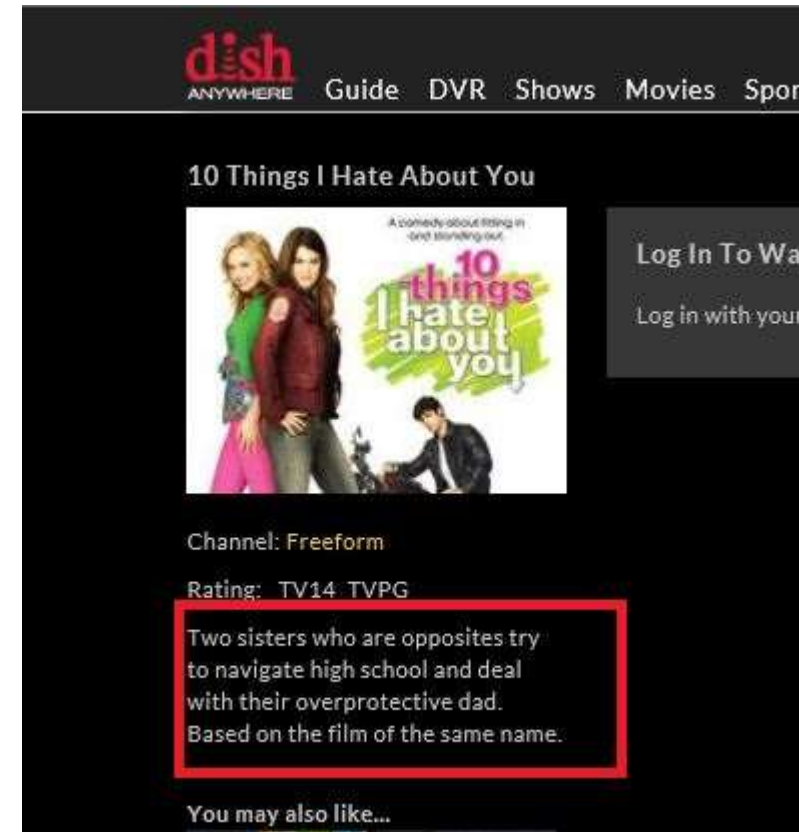
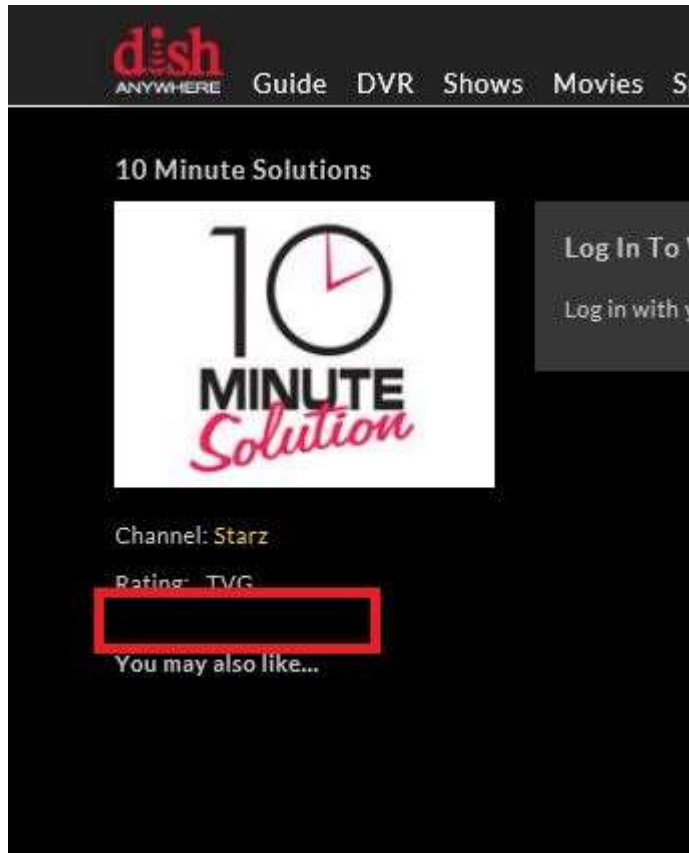
Postman start at Sun, Aug 12, 2018 8:29:44 PM

#	failure	detail
04.	AssertionError	sort=name; show[1] name=10 Minute Solutions; desc.length: '0' ge '5'
05.	AssertionError	sort=name; show[3] name=100 Code; desc.length: '0' ge '5'
06.	AssertionError	sort=name; show[6].name > show[5].name: '12 Corazones' gt '12 Corazones'
07.	AssertionError	sort=name; show[6].search > show[5].search: '12_corazones_e1570408' gt '12_corazones_e32940'
08.	AssertionError	Get DANY TV Shows genre 'Thriller' via search thriller: Total: '0' gt '0'
11.	AssertionError	Get DANY TV Shows letter '#' via search 123: Total: '0' gt '0'
13.	AssertionError	a-z=1; movie[9] name= La Casa De Beatriz; name[0]: ' ' in [1,L]
15.	AssertionError	Get DANY Movies letter '#' via search 123: Total: '0' gt '0'

Postman ended at Sun, Aug 12, 2018 8:34:03 PM

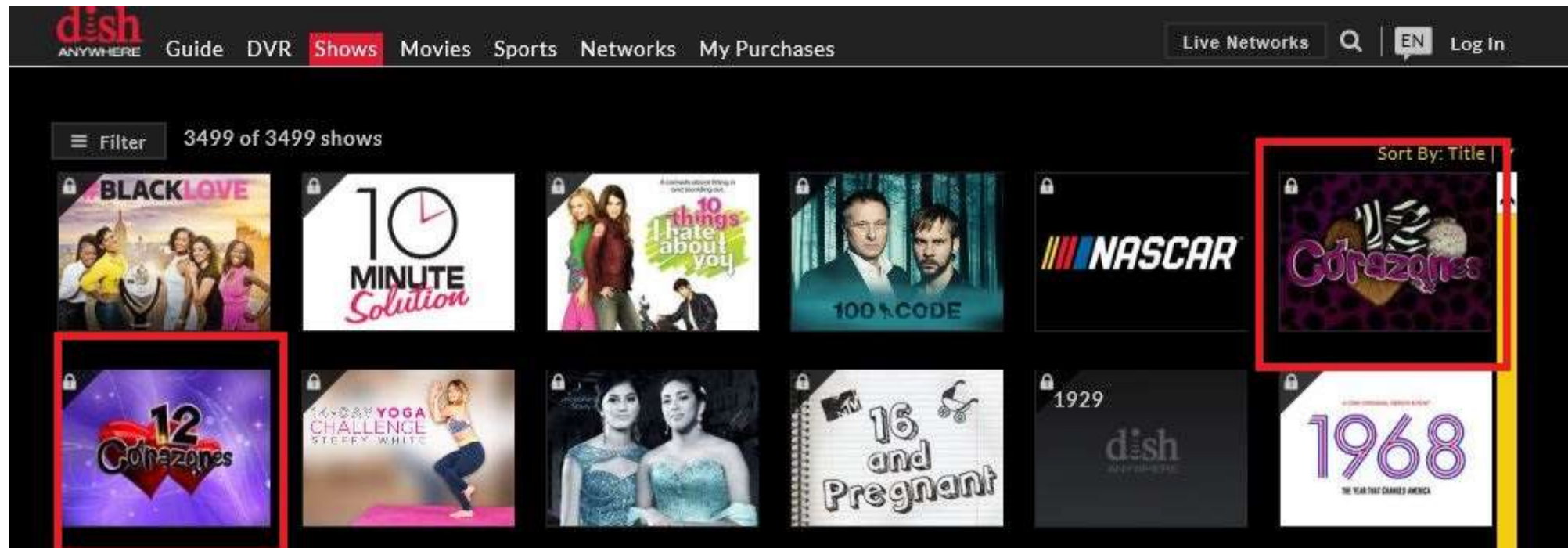
TV SHOW: 10 MINUTE SOLUTIONS

```
AssertionError sort=name; show[1] name=10 Minute Solutions; desc.length: '0' ge '5'
```



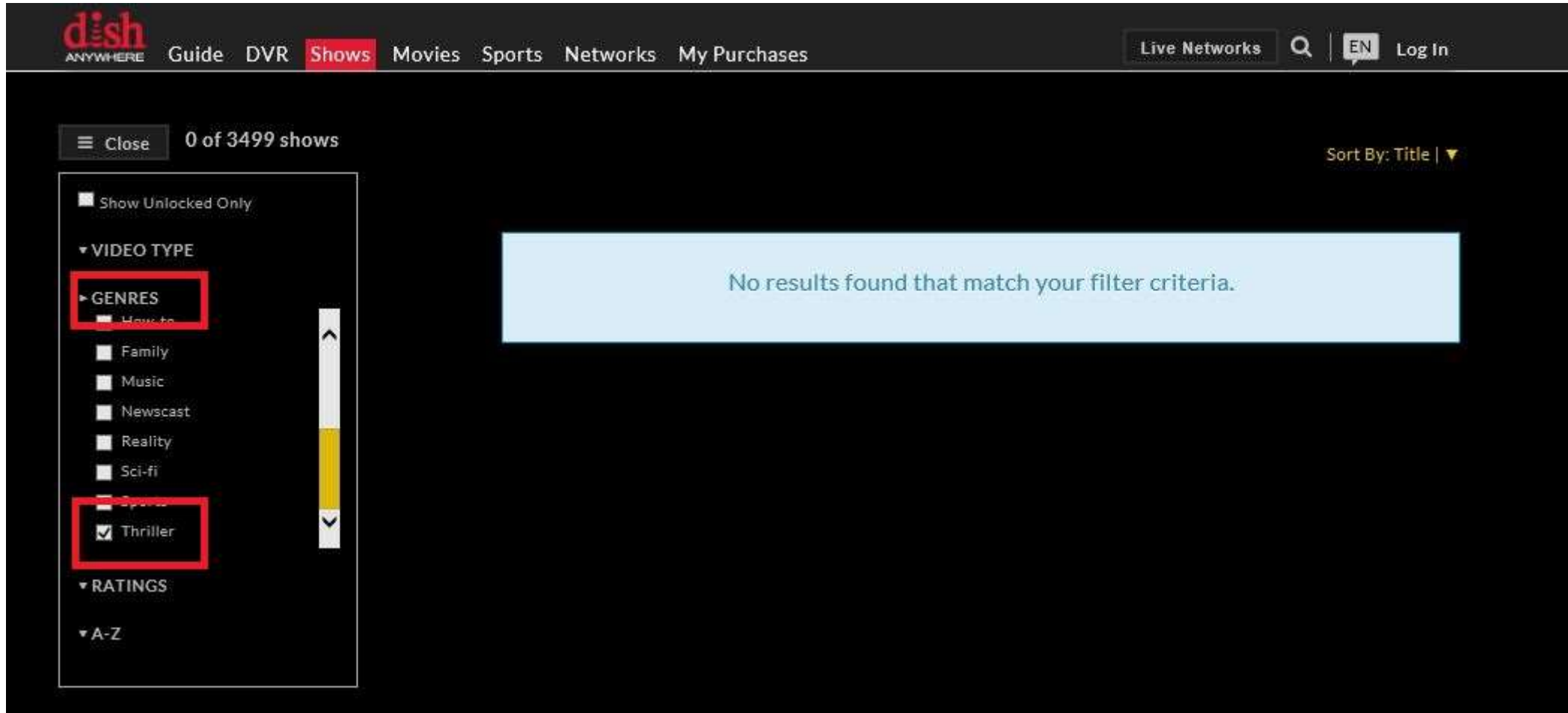
TV SHOW: '12 CORAZONES'

```
AssertionError sort=name; show[6].name > show[5].name: '12 Corazones' gt '12 Corazones'  
AssertionError sort=name; show[6].search > show[5].search: '12_corazones_e1570408' gt  
'12_corazones_e32940'
```



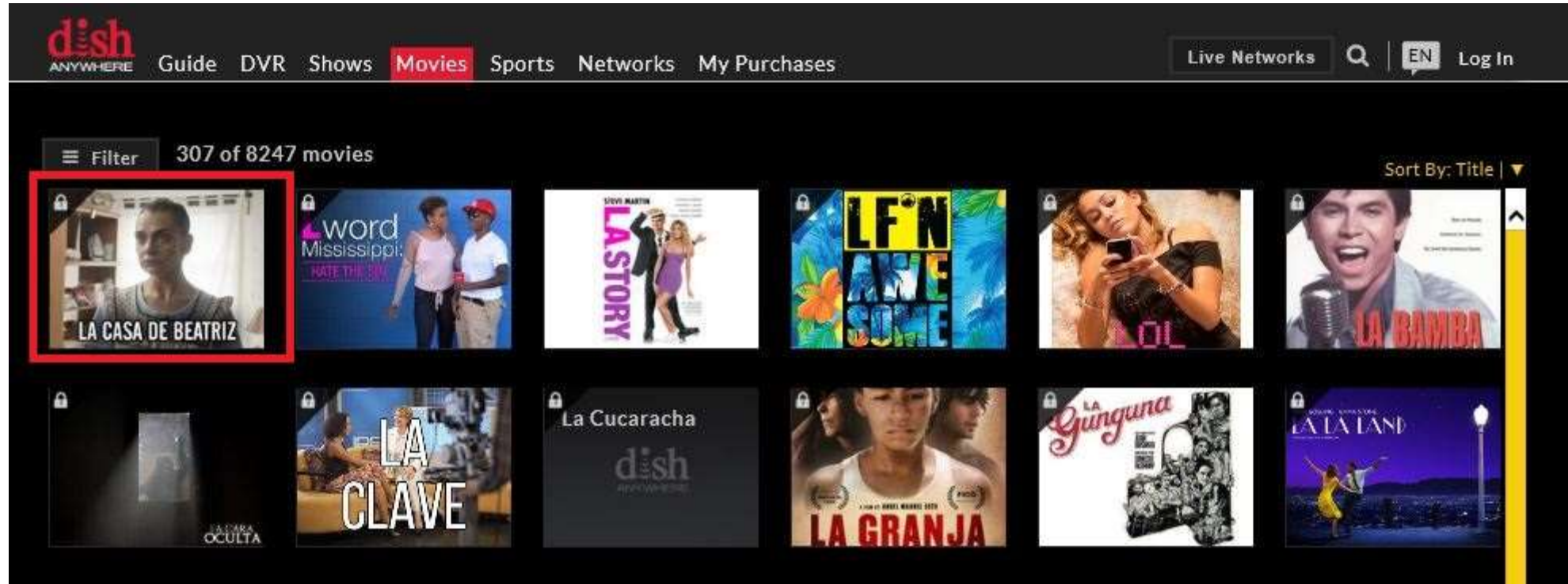
TV SHOW: GENRE THRILLER

AssertionError Get DANY TV Shows genre 'Thriller' via search thriller: Total: '0' gt '0'



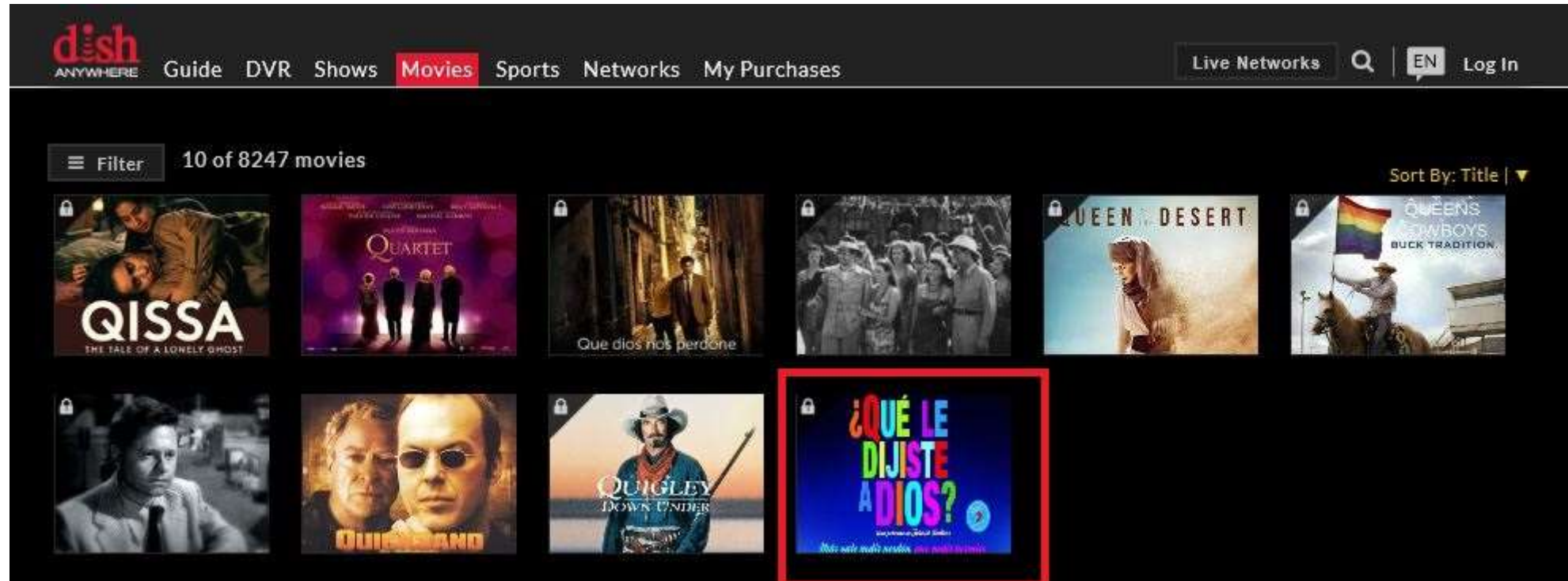
MOVIE : STARTS LETTER L

```
AssertionError a-z=l; movie[9] name= La Casa De Beatriz; name[0]: ' ' in [l,L]
```



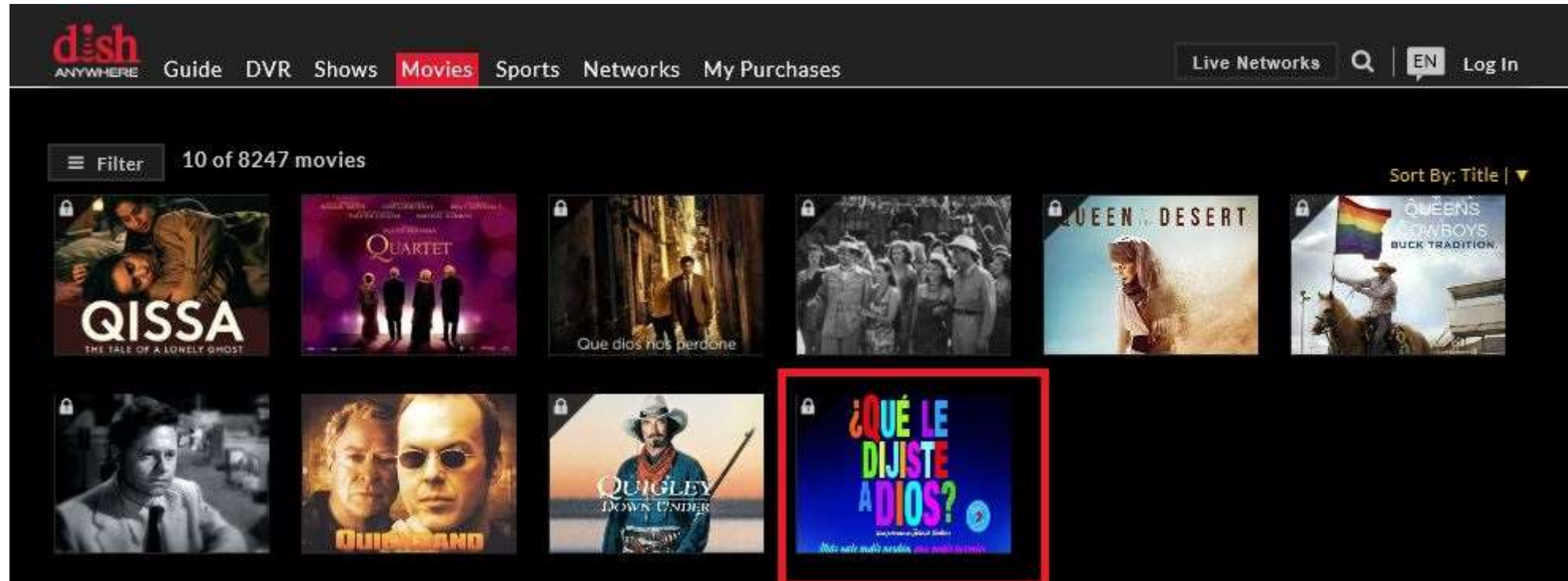
MOVIE : STARTS LETTER Q

```
AssertionError a-z=q; movie[4] name=¿Qué le dijiste a Dios?; name[0]: '¿' in [q,Q]
```



MOVIE : STARTS LETTER Q

```
AssertionError a-z=q; movie[4] name=¿Qué le dijiste a Dios?; name[0]: '¿' in [q,Q]
```



BASIC POSTMAN API TESTING

- First, Basic (Free) Postman API features
- Second, API Request/Response checks
- Third, Runner and Command Line Execution
- Fourth, Web spot checks of API Failures
- **Fifth, Conclusion and Summary**

BASIC POSTMAN API: PLUS

- Java Script: builds, sends, receives, evaluates API ReST messages
- Export to File System, Import from File System
- Developers use it and like more than cURL command line
- Internet documents, videos, training and support
- Command Line Execution tool
 - Console Output, Summary Results, Failure Details
 - Support Ci/CD Junit XML result format
 - Environment exported format supports variable value changes
- Groups API Requests in to Folder-like Hierarchy
- Support Output Console information
- Different variable types (Global, Environment, Collection, Local, Data)
 - Initialize Global, Environment, Collection variable values
- Generate Code Snippets in different languages
- State Machine Request Execution order with override ability
- Built-in Popular Java Script Node Packages
- Tool based Collection Execution: Results Pass, Fail summary

BASIC POSTMAN API: CONCLUSION

Minus:

- Global, Environment, Collection variable stored as string
 - Store via to-string functions (example: JSON.stringify, toString)
 - Use via from-string functions (example: JSON.parse, parseInt)
- **Helper Variables that define common functions and libraries**
 - Paid Versions support libraries and other shareable functions
 - Would like ability to load non-built-in Java Node Packages and personal packages
- Too Many Environments (group of Environment variables)
 - Convert Environment via CSV files before Command Line execution
 - Prefer many CSV files to many Environments

Conclusion:

- Overall Very Good Tool for API ReST testing
- Major problem: Helper Variables for common functions and libraries

DONE

Slides, Postman Collection, Shell Scripts, etc.

<https://github.com/arnold-miller0/Postman-Denver-Aug-2018>