OpenTest - Getting Started

Installation Guide

Pre-requisite software needed to be downloaded and installed:

1. Download Git
2. Visual Studio code
3. Add Extension yaml validation
4. Notepad++
5. Node.js
6. Java 8 runtime

Step 1: Install and setup the Latest OpenTest Server, Actor and Package:

You should have Node.js installed (any recent version):

Follow the Procedures in the below link to do latest Opentest Package installation

https://github.com/mcdcorp/opentest

Step 2: The test scripts repo

1. Create a “OpenTest” directory on the local disk
2. Have Git installed on machine
3. To check if Git is installed successfully: Cmd> git
4. Go to Stash https://coderepository.mcd.com/projects/SDKPIL/repos/sdk-automation-test
5. Clone Repository: Cmd> change directory to OpenTest> git clone <tests URL>
6. Provide Username and Password

This is just one of the repos with test scripts that we are currently using.

Step 3: Using Visual Studio Code

You must have Visual Studio Code installed:

1. “Open Folder” in Visual Studio Code, Open “Test-defs” folder here
2. Here you can explore example code for yaml tests
3. Add Extension yaml validation to Visual Studio code

Step 4: Create test sessions

When both the sync service and the actors are running properly, you can go into the sync service web UI and:

1. Check that the test actors you created are visible.
2. Click the Session/Create test Session menu.
3. Create a new test session, selecting one or more of the example tests (“01_UsingMacroActions”, “02_UsingSharedData”, etc.)
4. Watch the test session complete (the refresh is manual, for now) and click on its label to view more details about it.

Step 5: Create Automated Test Scripts:

API Automation Using Open Test Framework

[**NOTE:** Examples are described using dcs or mw test files as the approach is same in both]

API Automation deals with automating the Testing for RestFul Services.

There are two framework approaches used as described below:

1. Keyword Driven Approach:

In this approach, all the necessary test data are maintained in a single input yaml file for all the test cases.

All the related data files are maintained under the common folder called "api-testing".

For E.g.:

All the header information’s and endpoint for API can be updated in the below file:

1.1 Test Data:

1.1.1 Api-testing. Yaml: - Base URL Entries

This is a Data File that contains the endpoint URL path which will be used as common across the test scripts which can be provided in this file by giving a meaningful variable name.

DCS:

BaseEndpoint URL entries for Identity Profile and Data Virtualization:

```
dcsDevBaseIdentityURL: http://mcd-dcs-identity-elb-907765986.us-east-1.elb.
amazonaws.com/sys/v1/customer - Identity
dcsDevBaseProfileURL: http://mcd-dcs-profile-elb-1239983636.us-east-1.elb.
amazonaws.com/sys/v1/customer - Profile
dcsSVLogoutBaseURL: http://vse.mcdecp.datapipe.net:10084/sys/v1/customer -
Virtualized EndPoints
```

MW:

```
mwDevBaseURL: http://dev2.lb.anypointdns.net/exp/v1/customer
mwDevLogoutURL: http://dev2.lb.anypointdns.net/exp/v1/customer
mwSVLBaseURL: http://vse.mcdecp.datapipe.net:10085/dev/exp/v1/customer -
Virtualized Endpoint
```

1.1.2. API Headers:

The API Headers which are Common accross various API Test Scripts can be provided in a separate header.Yaml file under section RequestHeaders which can be referred in test files as

$data(“path”) -> $data(“api-testing/headers).RequestHeaders

dcsHeader.yaml has the below entry with all common headers mentioned inside { bracket with appropriate header values

Data for headers can be updated like below:

ContentType: application/json under dcsRequestHeaders

Common Headers for DCS:

```
dcsRequestHeaders:
{Content-Type: application/json,
     mcd-marketid: "US",
     mcd-locale: "en-US",
     mcd-sourceapp: "WEB",
     mcd-uuid: "644e1dd7-2a7f-18fb-b8ed-ed78c3f92c2b"}
```

Common Headers for MW:

```
mwRequestHeaders:
   {
        Content-Type: "application/json",
        Accept-Language: "en-US",
        mcd-clientid: "508747ceaec649079c0022b6c8d8052e",
        mcd-uuid: "a348b4c3-7915-4f5f-942b-b79f3589759d",
        mcd-sourceapp : "MOT",
    }
```

1.1.3. Input Data File

This is the input file for the automation test scripts. Create data file called "inputdata.yaml" where all the test dataSet has to be given and use the data path in the test script.

Input data required for both MW and DCS can be provided like below:

Test data required for each API test cases and also for macro can be provided with appropriate meaningful section names like below, so that it will be easy to track the data according to the test case and API

```
mwRegistration:
loginUsername: aru12Oct2017Test123@mailinator.com #need to update
dcsRegistration:
loginUserName: "testhoghyone@gmail.com"
```

1.1.4. API Status Codes:

All the success and failure status codes For API's are updated in the status code yaml file.

Status codes related to dcs are maintained in dcsStatusCodes.yaml and MW is updated in “mwStatusCodes”

This yaml has the below sections that includes:

Codes:

httpCode - under which all the http codes are mentioned with appropriate data name and the status code(For Eg: successRequest: 201)
dcscode or mwCode - under which all the success and error codes are mentioned(For Eg: success: 20000 whereas success is the data name and 20000 is the corresponding status code)

exception message, success message - are used to define the messages.

Macro:

This contains the action. In order to perform the Integration, Macro calls can be created to get the response from one api into another api.

DCSCallCreateProfile.yaml macro is used to get the dcsid from the response which is passed as an input to registration api

```
- script: |
var dscId = $readOutput("body").response.dcsId; - > is used to get the dcs
id response from the createprofile API which can be passed  to other API's
by calling the macro that provides this response
```

The readOutput(“body”) will read the output response value from the body and store it in a variable called dcsId which will be passed to the testcase request or header during runtime through the macro.

or

```
var dscId = $readOutput("body").response.dcsId;
$writeMacroOutput("dcsId", dscId);
```

NOTE: Any java scripts can be writtern in the yaml file using the $ symbol and | (Pipe Symbol) is used for multiple lines

JavaScript Example:

Any java scripts can be written under actions

```
actions:
            - script: |
                  var timestamp = String(Date.now());
                  var emailAddress = "user" + timestamp + "@mailinator.
com";
                  var number = timestamp.substring(timestamp.length - 10);
```

The above script is used to generate unique email id and phone number while executing the test script each time. In the above case, email id is appended with the timestamp each time to generate unique value and a random phone number

1.2. Test Script File:

The test script file contains the automated script which should have step numbers, description of each step, macro (optional) in case of integration (Calling one API inside another API), endpoint url entry, request and expected error code entry.

1.2.1. Steps:

This will be displayed in the output log while running the test session to clearly display the step by step actions executed for the test script.

```
steps:
        - step: 1
          actions:

StatusCodes:(dcsStatusCodes.yaml)
```

All the success and failure status codes are updated in the above file.
Codes:
httpCode - under which all the http codes are mentioned
dcscode - under which all the success and error codes are mentioned
exception message, success message - are used to define the messages

1.2.2. Macro (Optional):

In order to use the response of another API, a macro can be created and called under actions using (macro: path of the macro)

For E.g.: macro: api.DCSCallCreateProfile and the input values to the macro can be passed using arguments like:

args:

emailAddress: $script emailAddress

number: $script number

The macro can be called in the test file as below:

```
actions:
- description: Call DCS create profile API
   macro: api.DCSCallCreateProfile
args:
emailAddress: "dataUser06@mailinator.com"
number: "5478877851"
and the response "dscId" can be called in the header or body as below:
body:
"dcsId": dscId
header:
$Script dscId
The response can be passed as local data using the below code:
- step: 1
   actions:
- description: Call DCS create profile API
  macro: api.DCSCallCreateProfile
args:
$localData:
   dcsId: $readOutput("dcsId")
To use the above code, the below code has to be given in the corresponding
macro
var dscId = $readOutput("body").response.dcsId;
$writeMacroOutput("dcsId", dscId);
The above approach will be useful, if a test script requires multiple
macros to be called and the above provides the clarity on which response
is output by which macro and the value should be passed in the body like
below if the local data should be passed as runtime value in the request
body:
body
"dcsId": $localData("dscId")
```

1.2.3. URL Entry and Verb:

URL is the Endpoint URL and Verb is the method like POST, GET, etc

Pass the URL and method as verb in action argument

Give the entry for Endpoint URL as:

url: $format("{0}/profile?type=traditional", $data("api-testing/apiData").apiEndpoint.dcsDevBaseProfileURL)

{0} - is a place holder which will be replaced with the "/profile?type=traditional" and will be appended with the base URL

For Eg:

```
action: dtest.actions.HttpRequest
              args:
                url: $format("{0}/identity", $data("api-testing/apiData").
apiEndpoint.dcsDevBaseIdentityURL)
                verb: POST
```

1.2.4. API Header:

HTTP **Headers** are an important part of the **API** request and response as they represent the meta-data associated with the **API** request and response. Path for the common headers can be provided like below:

```
 headers: $data("api-testing/dcsHeader").dcsRequestHeaders - DCS
 headers: $data("api-testing/mwHeader").mwRequestHeaders – MW
```

1.2.5. API HttpCode:

The “successStatusCode” is used to define the http code and the path for the http code should be mentioned like below:

```
successStatusCode: $data("api-testing/dcsStatusCodes").httpCode.
successRequestLogin
```

1.2.6. Request Body:

The request body contains the actual request that should be passed inside the $json

```
body: |
                  $json(
                   {
                      "traditional": {
                      "loginUsername": $data("api-testing/inputdata").
dcsLogin.LoginUsername,
                      "type": $data("api-testing/inputdata").dcsLogin.type,
                      "password": $data("api-testing/inputdata").dcsLogin.
password
                      }
                    }
                  )
whereas the data values for the request can be mapped from the inputdata.
yaml file
$data("api-testing/inputdata"). – specifies the path of the inputdata file
Whereas LoginUsername is the parameter variable provided under dcsLogin
section in the inputdata file.
```

1.2.7. Response Verification:

How to compare the Expected and Actual Result:

The comparison code has been writtern in the Validate Json file and is placed under the path macro api ValidateJson.yaml

1.2.7.1. Success Response:

The Success response displays the success status code.

The validate json macro should be called which has the logic behind the json response comparison

```
- description: Verify that response JS objects are equal
macro: api.ValidateJson
args:
expected:
status:
code: $data("api-testing/dcsStatusCodes").dcsCode.success

actual: $readOutput("body")


validateJson macro  - does the comparison between the expected and the
actual result
actual: $readOutput("body") – reads the actual output from the response
code :The expected code path has been provided against which the actual
output code will be compared.
```

1.2.7.2. Error Response:

The error response displays the appropriate error code.
The error response are displayed in array as below:

Code Snippet:

```
{
"status": {
"code": 40000,
"type": "BusinessException",
"errors": [ - the error code is defined in an array
{
"code": 40092,
"type": "InvalidDataException",
"message": "The account already exists and is currently inactive for
provided email address.",
"service": "Profile"
}]}}
Hence, the yaml code can be written as below:
- script: var body = $readOutput("body"); - reads the output from the body
first
- description: Verify that body JS objects are equal
macro: api.ValidateJson
args:
expected:
status:
code: $data("api-testing/dcsStatusCodes").dcsCode.validException
actual: $script body – pass the body output
# errorMessage: "Expected object did not match the actual"
- description: Verify that response body
macro: api.ValidateJson
args:
expected:code: $data("api-testing/dcsStatusCodes").dcsCode.
invalidDataException
actual: $script body.status.errors[0]  For displaying error code in an
array
```

2. Data Driven Approach

It is an approach by which testing is done using a table of conditions directly as test inputs and verifiable outputs as well as the process where test environment settings and control are not hard-coded.The data driven approach is used to avoid writting multiple scripts based on various data:

OpenTest Technical Reference - OpenTestTechnicalReference-Data-drivenTesting This link is a reference for writting scripts using data driven approach

(For Eg: Request header or body requires validation testing for which variety of dataset is required)

Various Validation scenarios are:

1.header validation required key missing

2.body validation

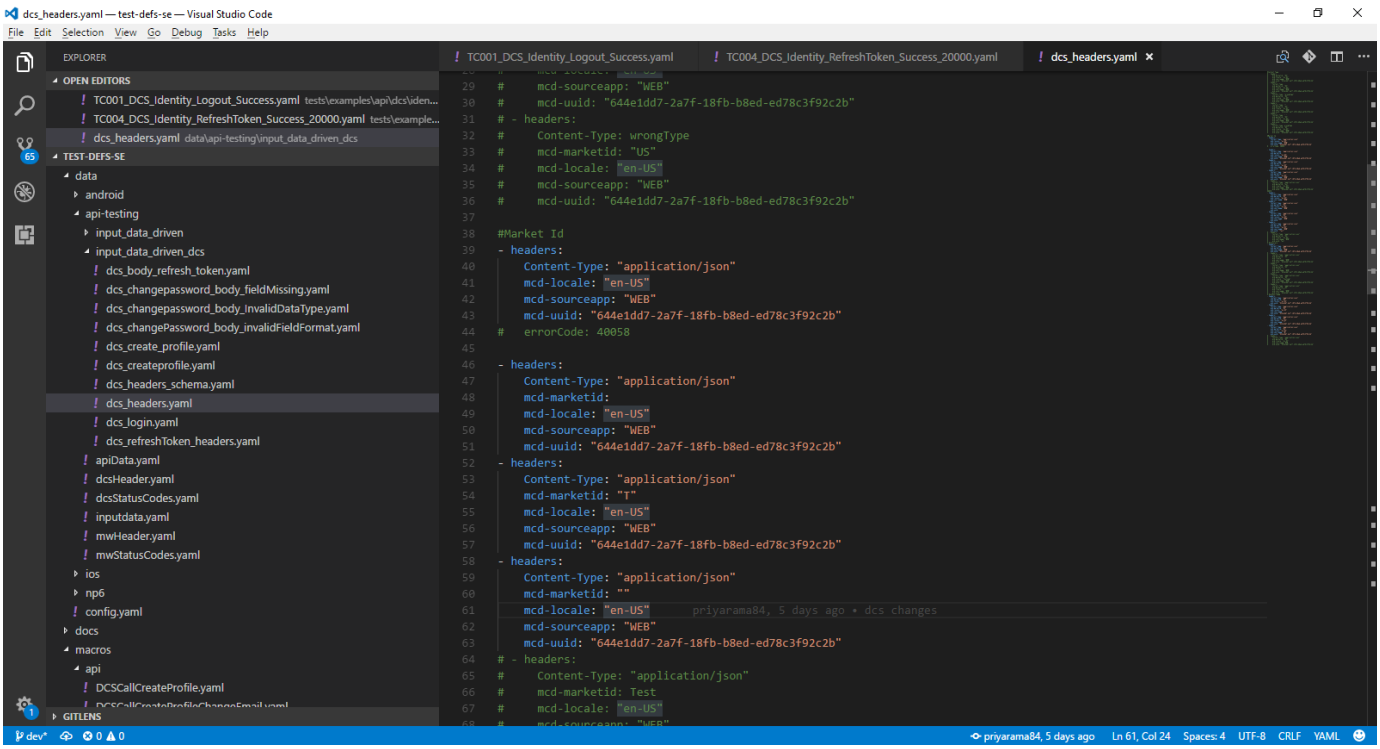
3.syntax validation

4.url validation

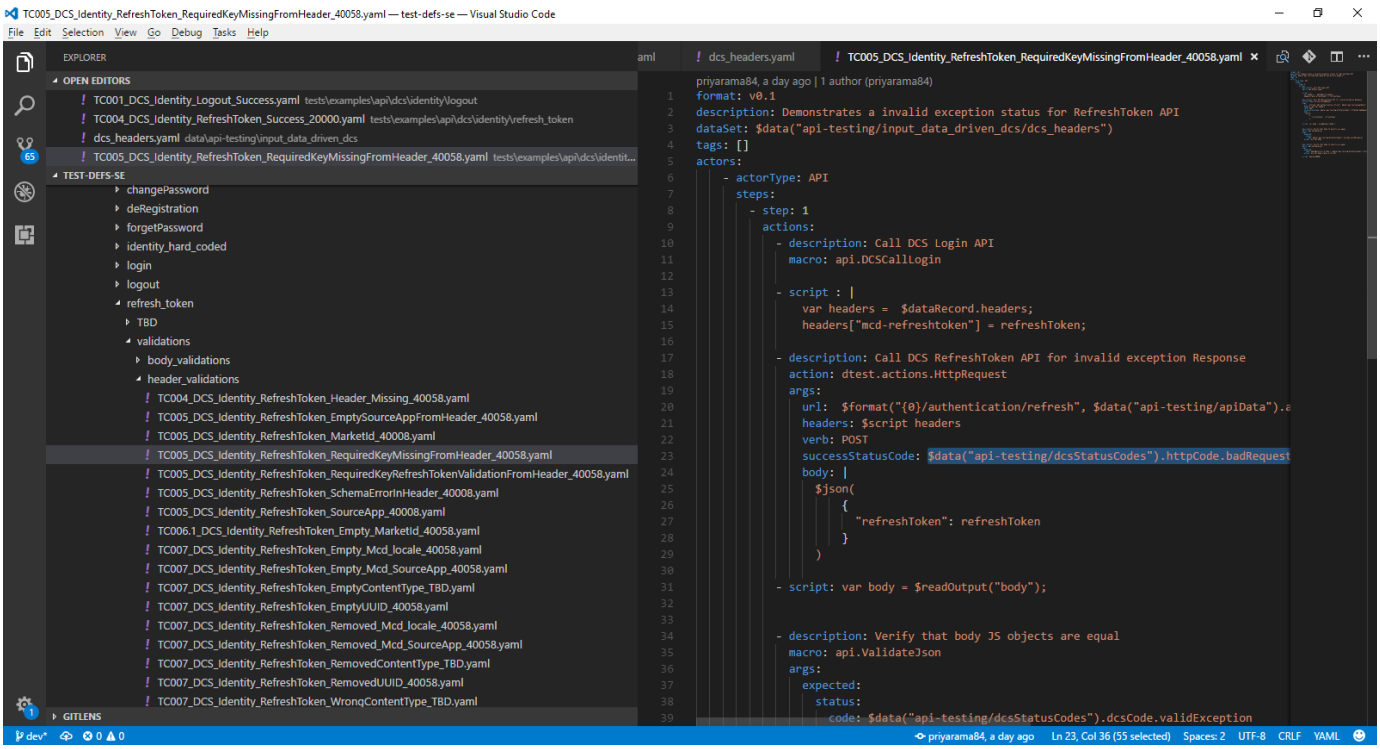
2.1. Header validation:

This deals with validating the headers.

Sample header data file:



Sample Test File for calling the data file:



dataSet:

This keyword represents the dataset to be used in the data driven approach.

The data file can be called using dataset as below:

dataSet: $data("api-testing/input_data_driven_dcs/dcs_headers") - If the datas are stored in an external file
or the values can also be defined in the same file like below

```
Dataset:
  - emailAddress: TestUser001@gmail.com
     phoneNumber: 9876543209
 - emailAddress: TestUser002@gmail.com
     phoneNumber: 9877656789
 - emailAddress: TestUser003@gmail.com
     phoneNumber: 9878776549
 - emailAddress: TestUser004@gmail.com
     phoneNumber: 9665789009
```

The refreshtoken which is passed dynamically in the header from the macro can be writtern as below:

```
- step: 1
actions:
- description: Call DCS Login API
macro: api.DCSCallLogin

- script : |
var headers = $dataRecord.headers;

headers["mcd-refreshtoken"] = refreshToken;


All the dataset for headers can be specified as below:
headers: $script headers
```

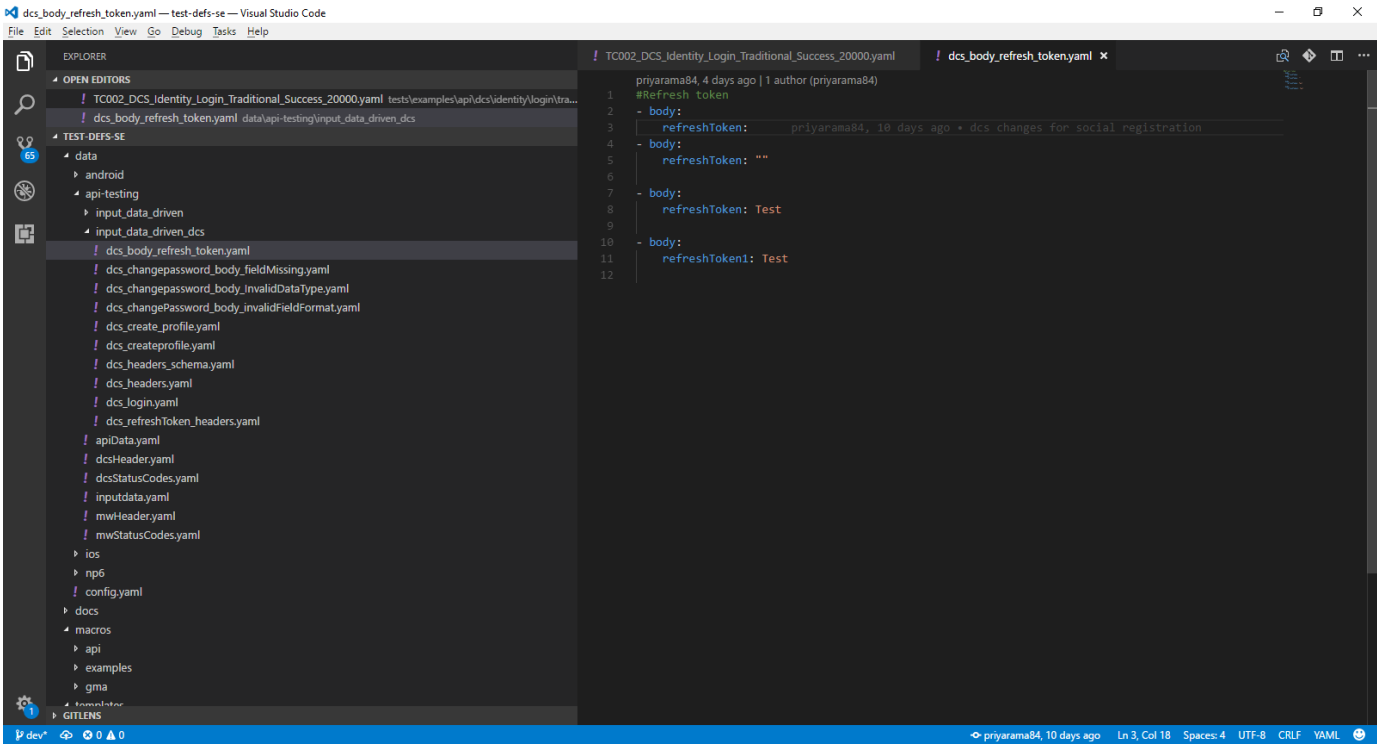
2.2. Body validation:

This section deals with validating the request body.
How to use the data from the data file in the request body:

The values from the data file can be called using

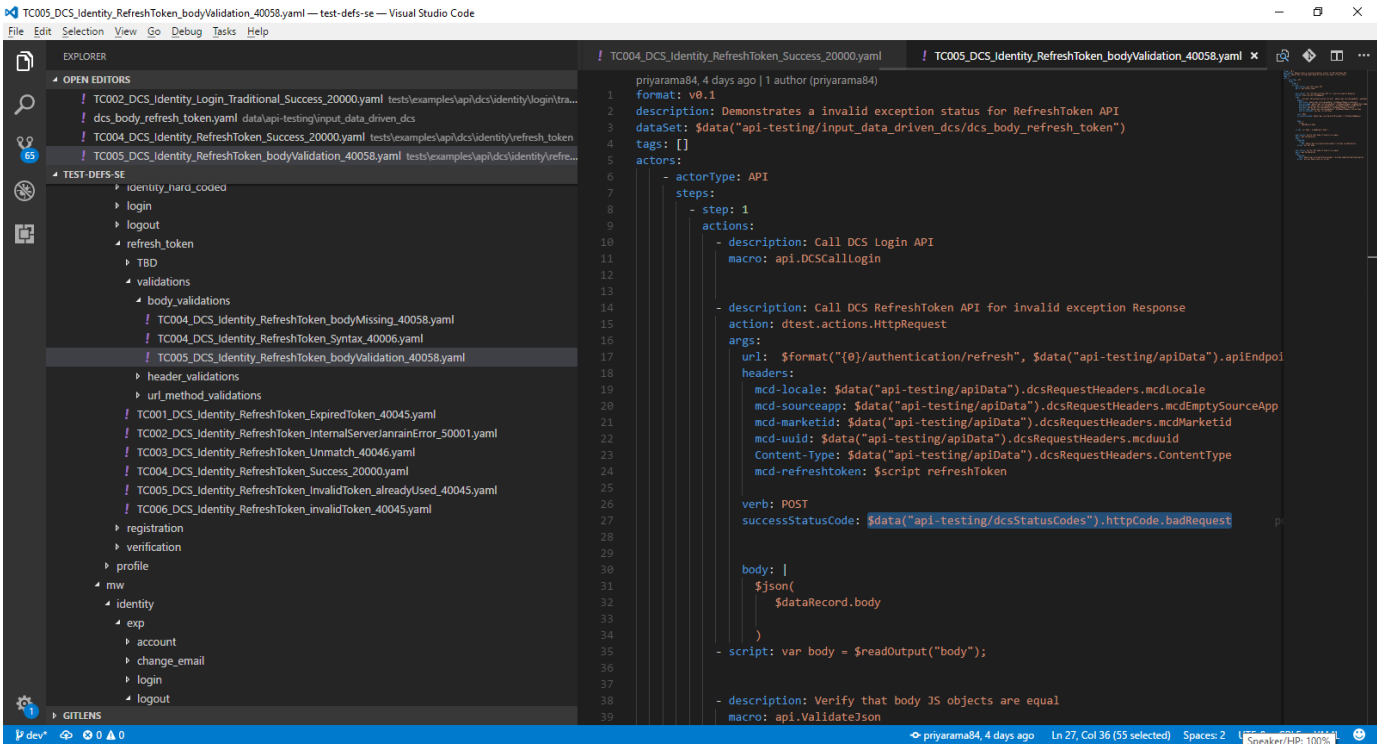
$dataRecord.body

The data file for request body data validation scripts can be created as below:



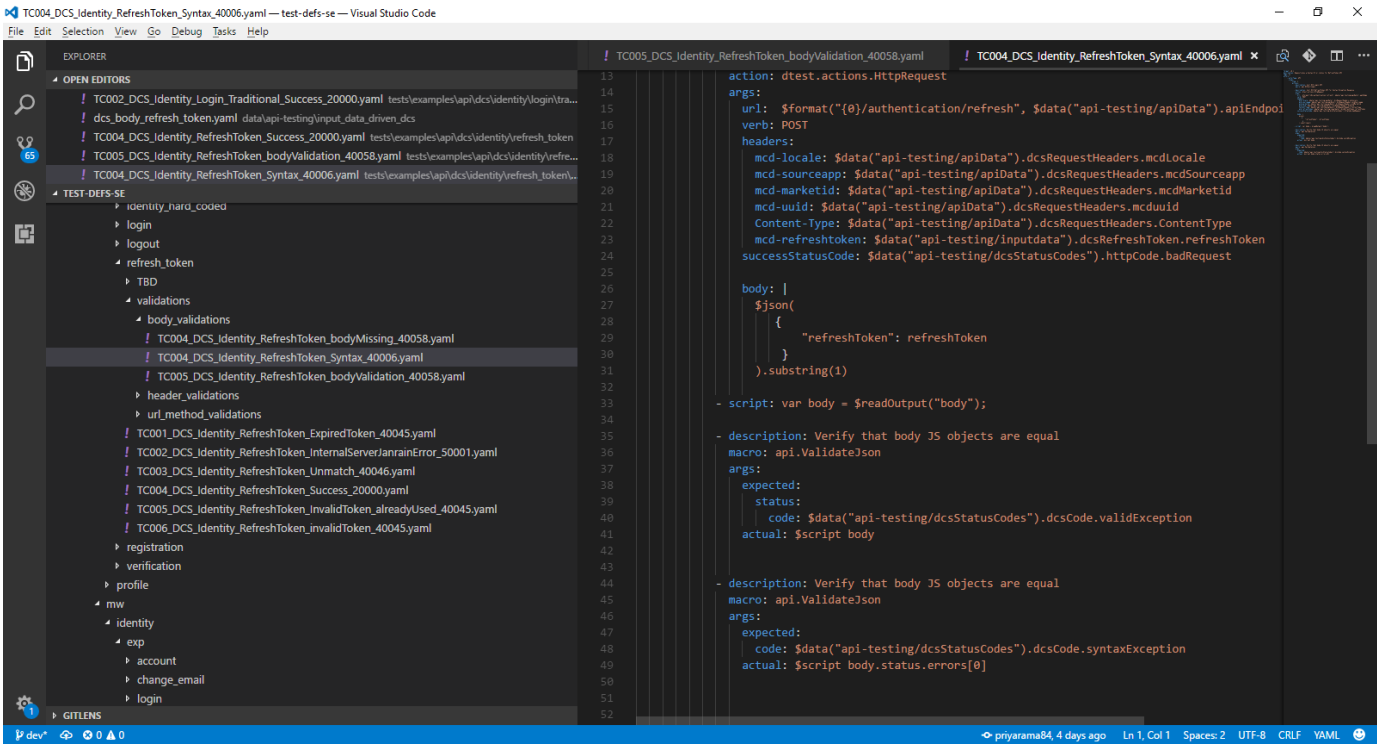
The testcase file should be created as below:

The testcase file contains the test scripts writtern in Yaml.



2.3. Syntax validation:

This section details about validating the syntax error in the request body. Testscripts for testing the validation error codes due to the body syntax error can be writtern as below:



The above implies that the open curly brace has been removed or is not passed in the json.

In the yaml file, if we remove the curly brace or make a syntax error then the script will not pass the proper json request.

The json request has to be passed and it has to be interpreted as a valid json with syntax error

substring function will extract only the required string part - substring(1) , which means the first passed string value is the open curly brace

Eg:

var s = "priya"
s.substring(1) --gives you "riya"

2.4. Url Validation:

This details about validting the EndPoint URL's

%20 - is used for space

Below url passes the empty space as a delete reason

url: $format("{0}/profile?dcsId=11840000000000624&deleteReason=%20", $data("api-testing/apiData").apiEndpoint.dcsDevBaseProfileURL)

- script: $delay(60000) This script can be used for timedelay (In DCS, If we do login continuously several times with the same credentials then due to security, server error will be thrown. In order to avoid the same, we can use a time delay of 1 min in the test script while everytime a login macro has been executed)

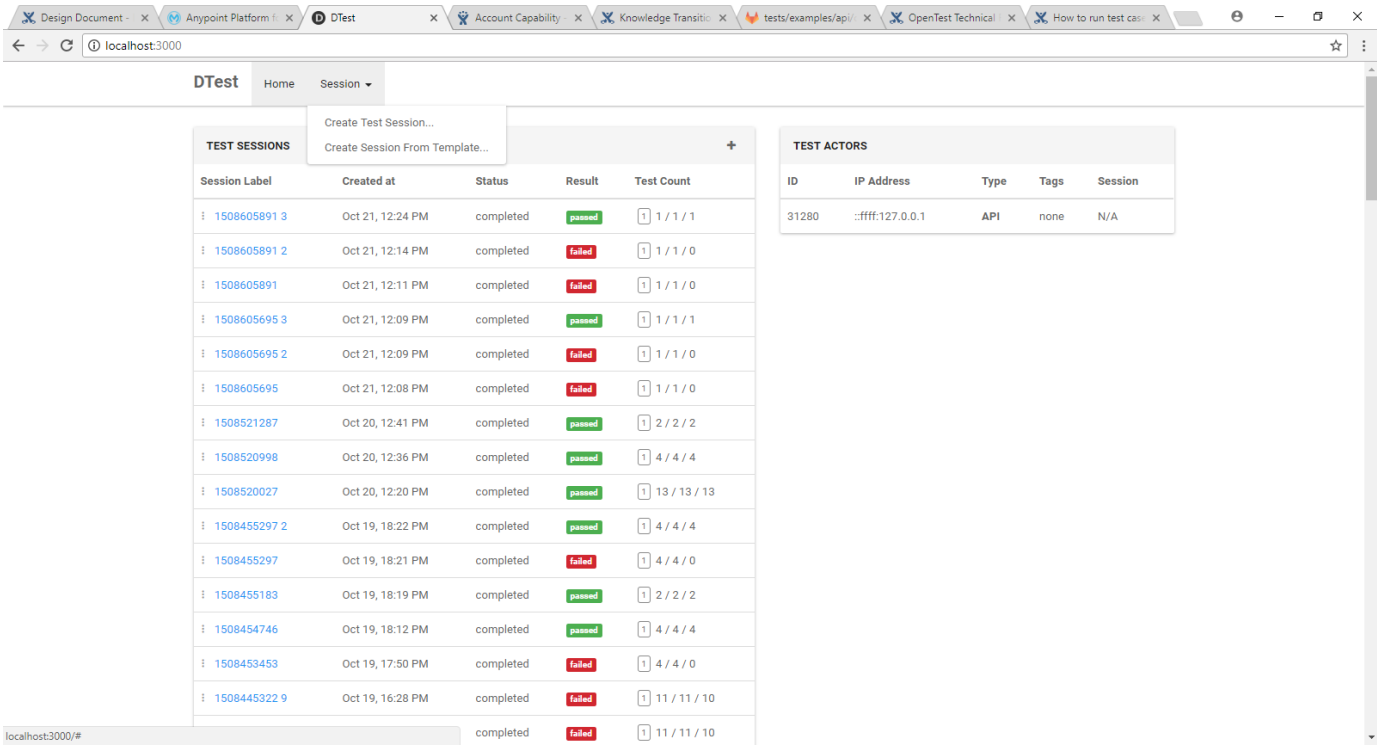
3.Templates:

How to run test cases in OpenTest using template - please refer this link for template creation and execution

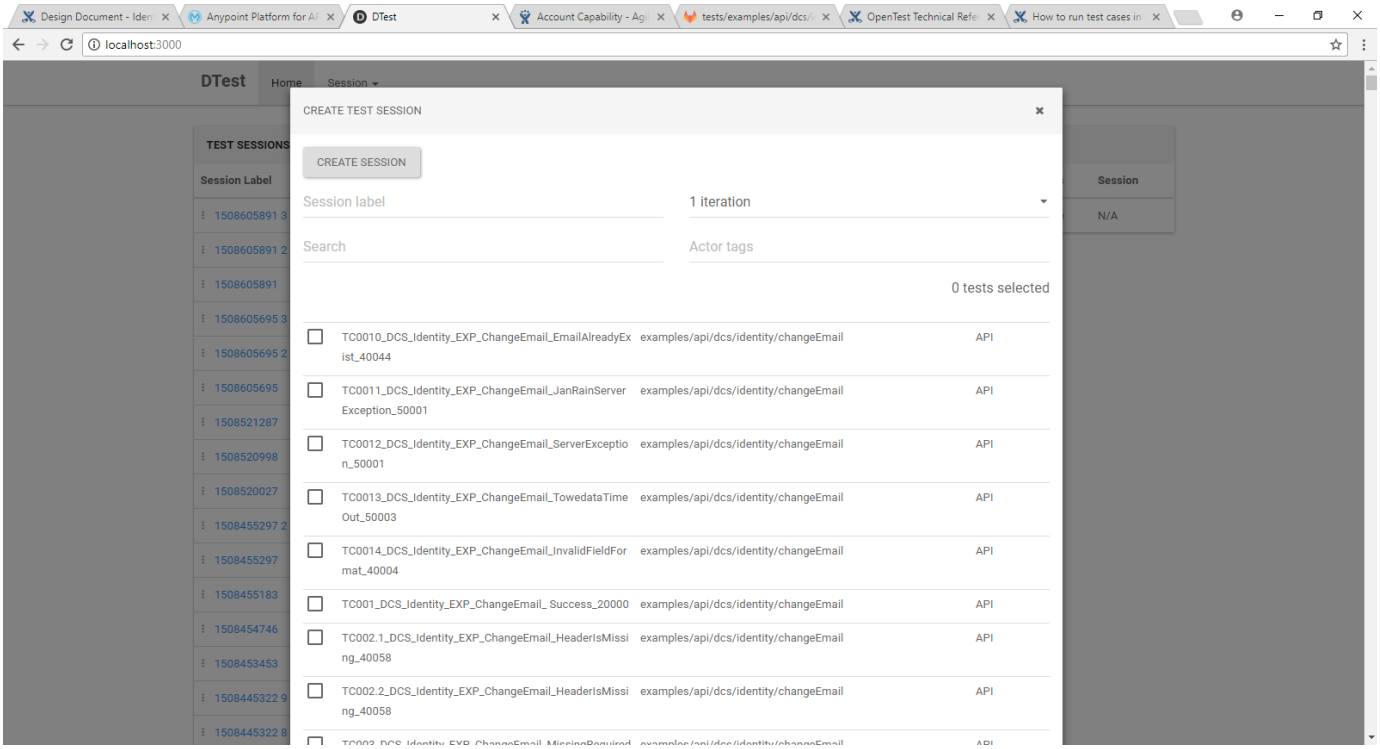
Sample: Template

```
sessionLabel: DCS Smoke Build Login
maxIterations: 1
tests:
#Login
- name: TC001_DCS_Identity_Login_InvalidEmail_30040
path: examples/api/dcs/identity/login/traditional
- name: TC001_DCS_Identity_Login_WrongEmail_30040
path: examples/api/dcs/identity/login/traditional
- name: TC002_DCS_Identity_Login_Traditional_Success_20000
```

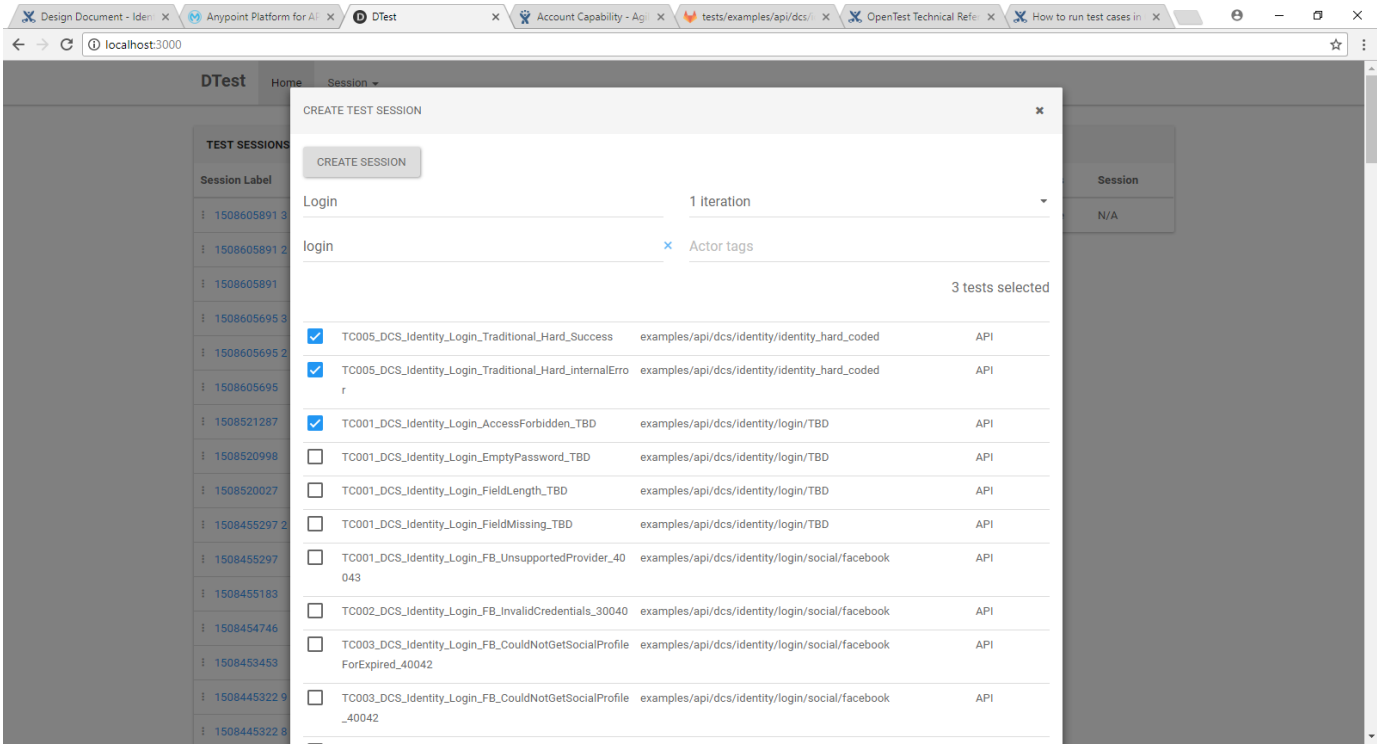
4. How to create test session locally:



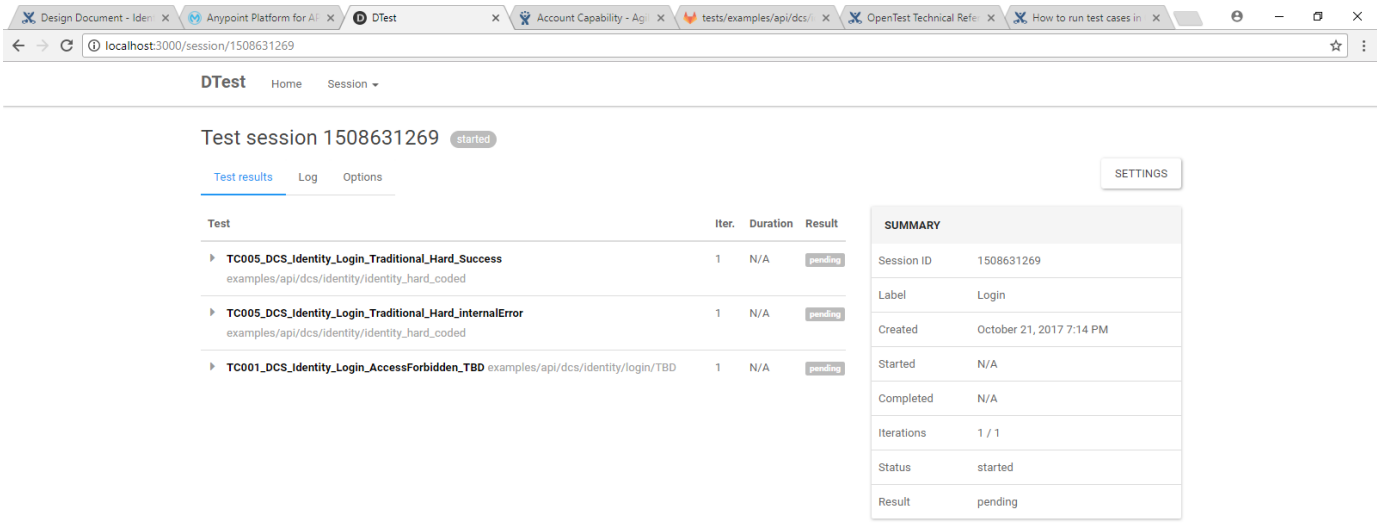
Click on the "create Test Session", all the available testcases will be loaded like below:



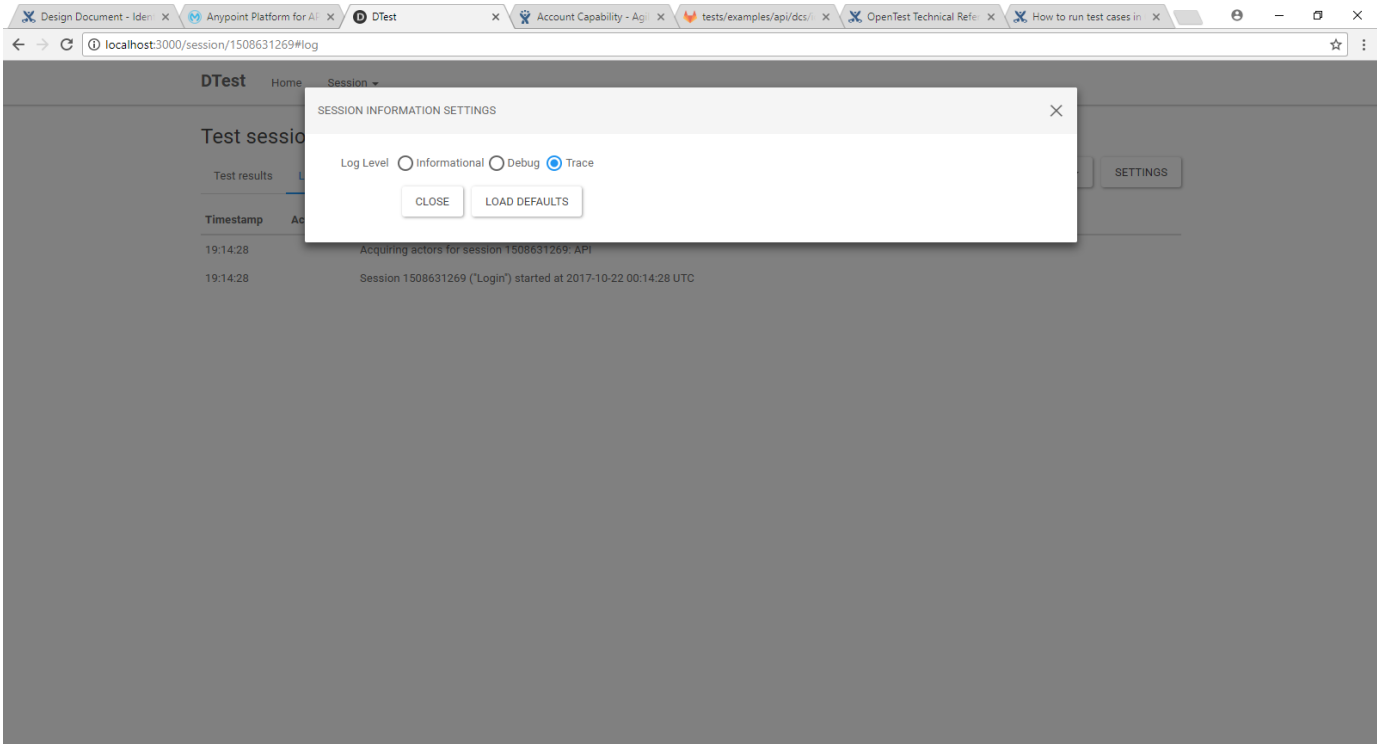
Select the testcase that you want to run by searching and you can give the session lable,
The no of iterations that you can choose are 1,2 and 3.



Click on "Create Session" to run



The logs can be set to verious log levels like 1. informational 2.Debug and 3.trace by clicking on the settings



API Automation Using GMA - (Android)

Install the below software's for running android mobile applications apart from those listed above under installation guide section:

1. Android SDK:

Pre-requisite – Need Java Development Kit for the latest version of Java

Go to https://developer.android.com/studio/index.html

Launch the .exe file you downloaded.

Follow the setup wizard to install Android Studio and any necessary SDK tools.

To set path - Go to Control panel > system> advanced system settings > environment variables

Copy and paste the below path into the edit box separated by semi-colons (;) and click on the Ok button to save changes for path.

C:\Android\sdk\tools;C:\Android\sdk\platform-tools

Also, create a system variable ANDROID_HOME with value C:\Program Files (x86)\Android\android-sdk

2. Appium 1.7:

Run below commands in sequence to install appium 1.7

appium install steps:
$ npm install -g appium
$ appium
$ npm install -g appium-doctor
$ appium-doctor
$ npm install appium-uiautomator2-driver

$ appium -a 127.0.0.1 -p 4723

The following desired capabilities should be provided under actor properties file

appium:

```
    appiumServerUrl: http://127.0.0.1:4723/wd/hub
    desiredCapabilities:
        platform Name: Android
        app: C:/Users/MC58412/Desktop/GMA/Android/Builds/Nov3
/IntegratedBuilds/SPRINT14/Integrated Build/app-QA-release.apk
        udid               : $data("android/inputdata").
androidCapabilities.udid
        app                : $data("android/inputdata").
androidCapabilities.app
        deviceName         : $data("android/inputdata").
androidCapabilities.deviceName
        automationName     : $data("android/inputdata").
androidCapabilities.automationName
        autoLaunch         : $data("android/inputdata").
androidCapabilities.autoLaunch
        noReset            : $data("android/inputdata").
androidCapabilities.noReset
        fullReset          : $data("android/inputdata").
androidCapabilities.fullReset
        newCommandTimeout  : $data("android/inputdata").
androidCapabilities.newCommandTimeout
        appPackage         : $data("android/inputdata").
androidCapabilities.appPackage
        appActivity        : $data("android/inputdata").
androidCapabilities.appActivity
        appWaitPackage     : $data("android/inputdata").
androidCapabilities.appWaitPackage
        appWaitActivity    : $data("android/inputdata").
androidCapabilities.appWaitActivity
```

3. Project URLs:

- Test actor common dependencies: https://gitlab.com/adrianth/dtest-base
- Test actor main project: https://gitlab.com/adrianth/dtest-actor
- Test actor mobile actions: https://gitlab.com/adrianth/dtest-mobile-actions
- Test actor NP6 actions: https://gitlab.com/adrianth/dtest-np6-actions

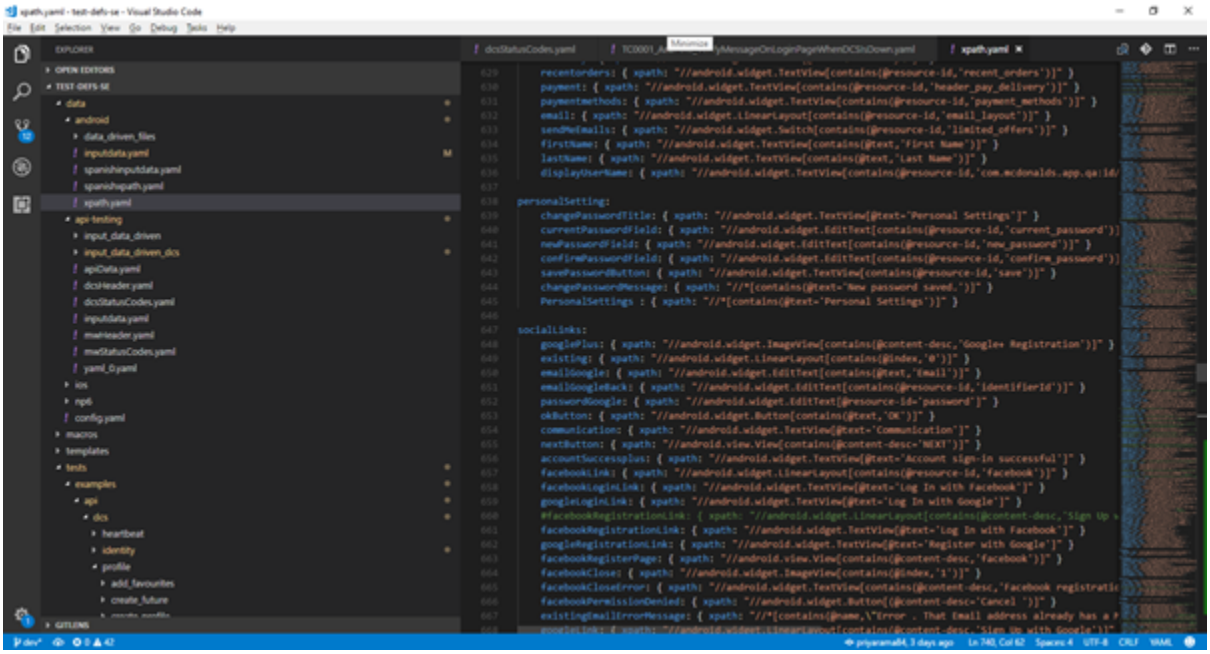
4. Pre-build Jar link:

https://drive.google.com/file/d/0BzbbPjqA1I4RX2ZVeVNMVk5yQWs/view?usp=sharing

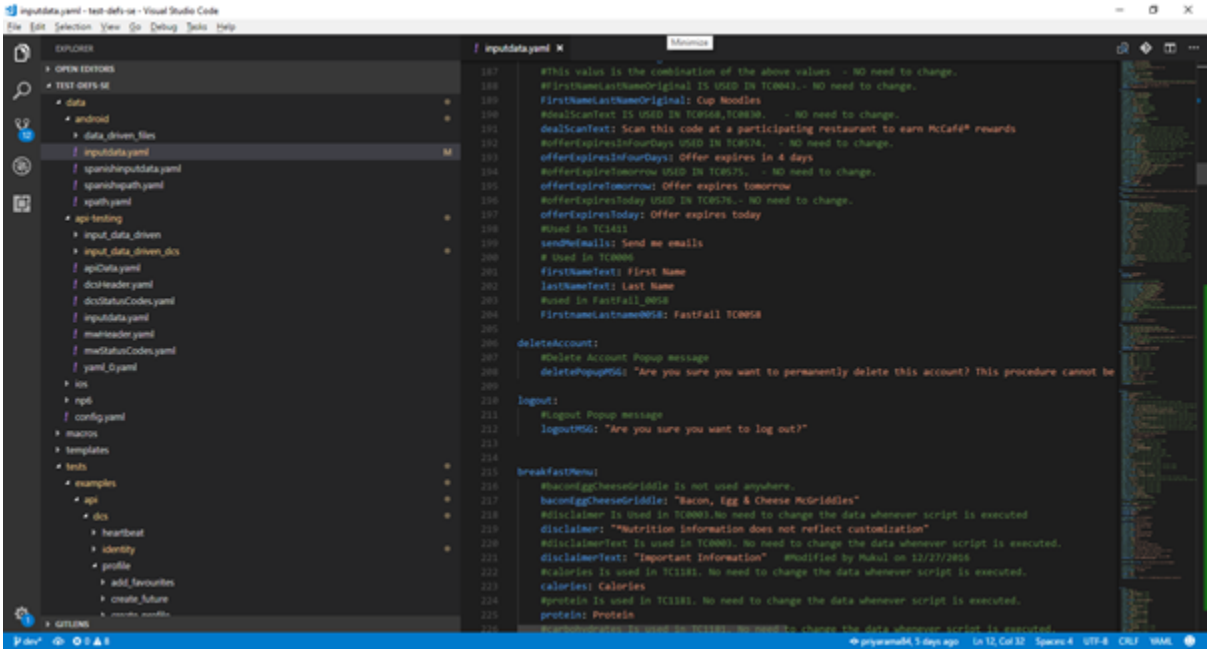
5. Latest URL:

https://drive.google.com/open?id=0BzbbPjqA1I4RblVkNXhibmxuSkU

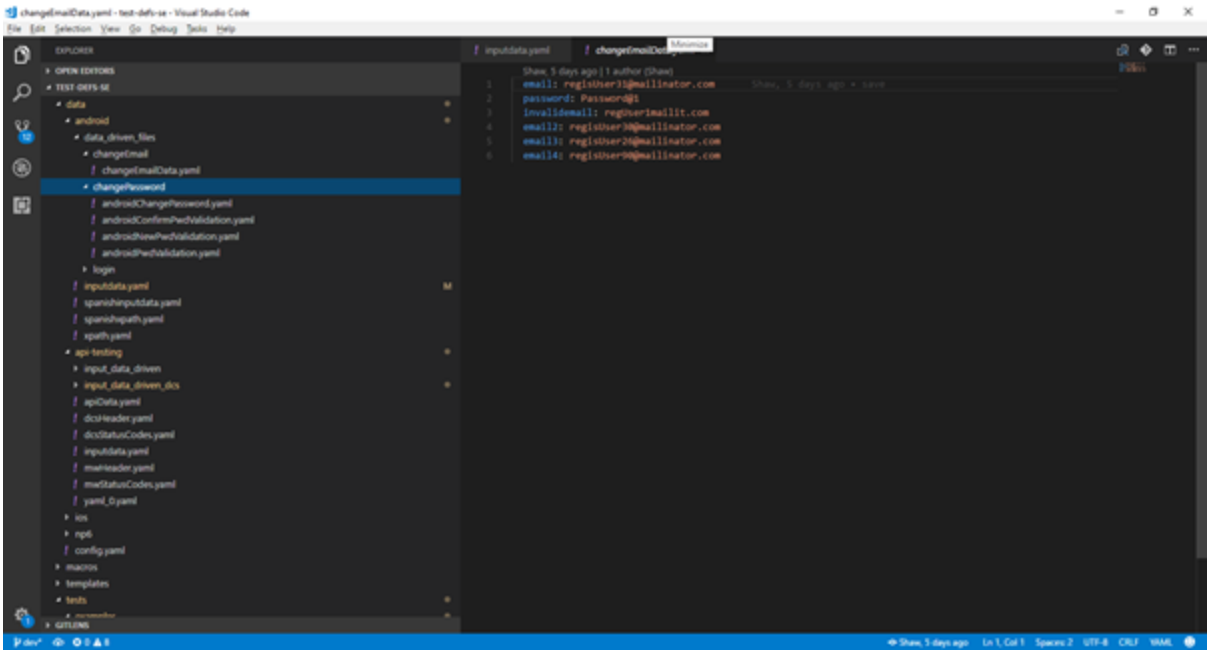
All the Xpath related to app in android is maintained in the xpath.yaml under data android



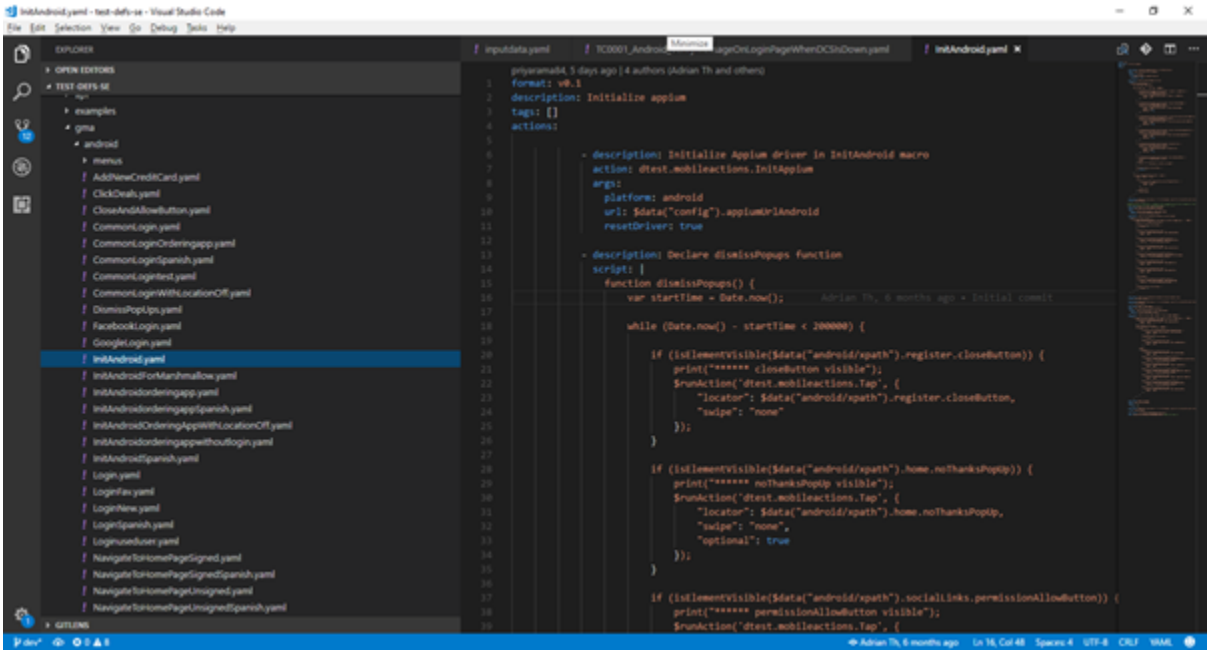
Input is maintained under inputdata.yaml



Input data user for data driven approach is maintained under data -> android -> data_driven_files



Macros should be created under -> Macros ->gma -> android



AWS Walk Through for Android and IOS

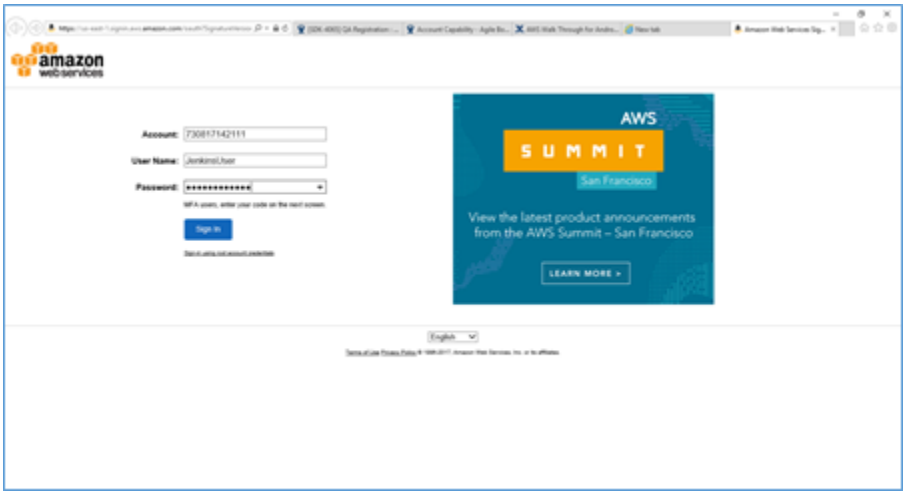
AWS setup for Android:

Step 1:

Open the AWS Server - https://730817142111.signin.aws.amazon.com/console

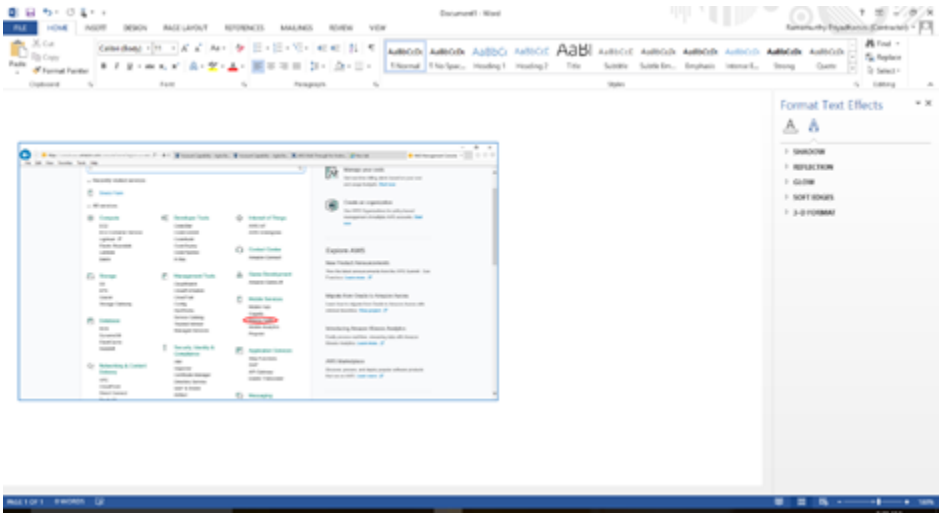
Credentials:

Username: JenkinsUser, Password: tSF7H7GUlQwo



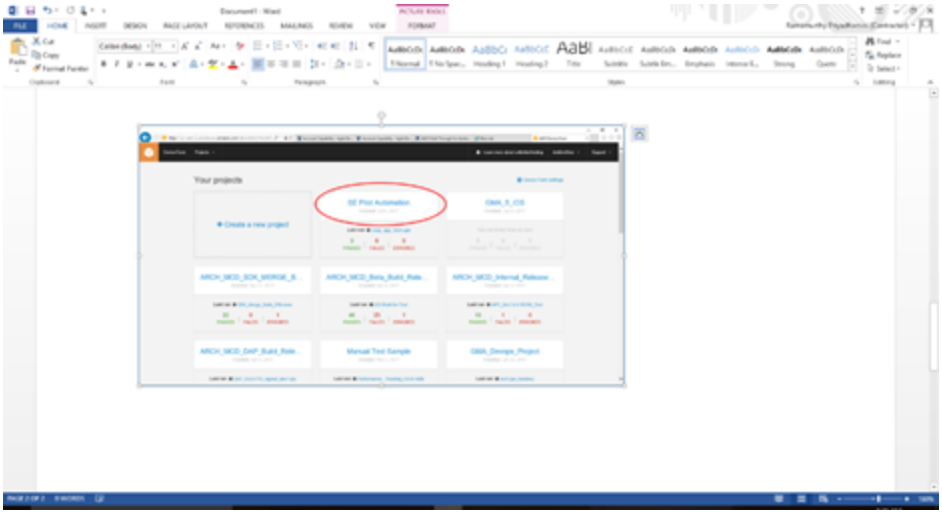
Step 2:

Click on the Device Farm link under mobile services



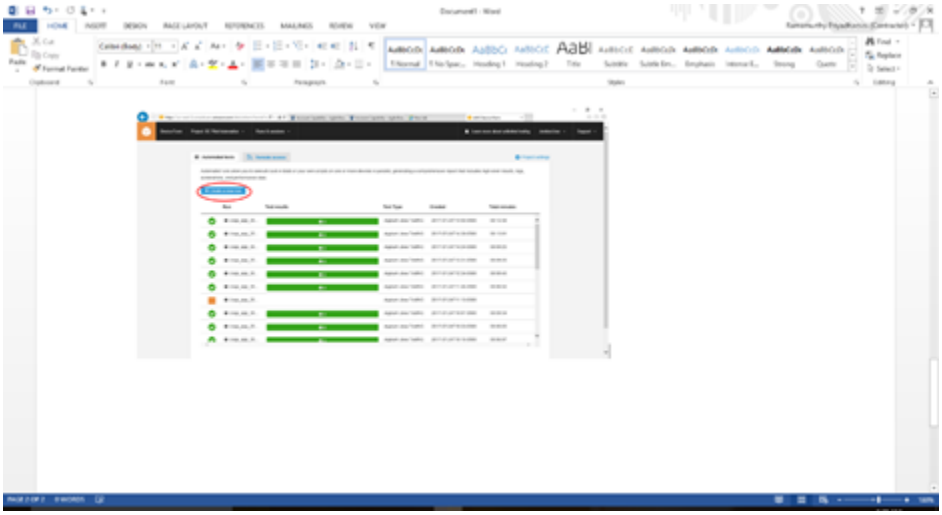
Step 3:

Under "Projects", Click on “SE Pilot Automation"



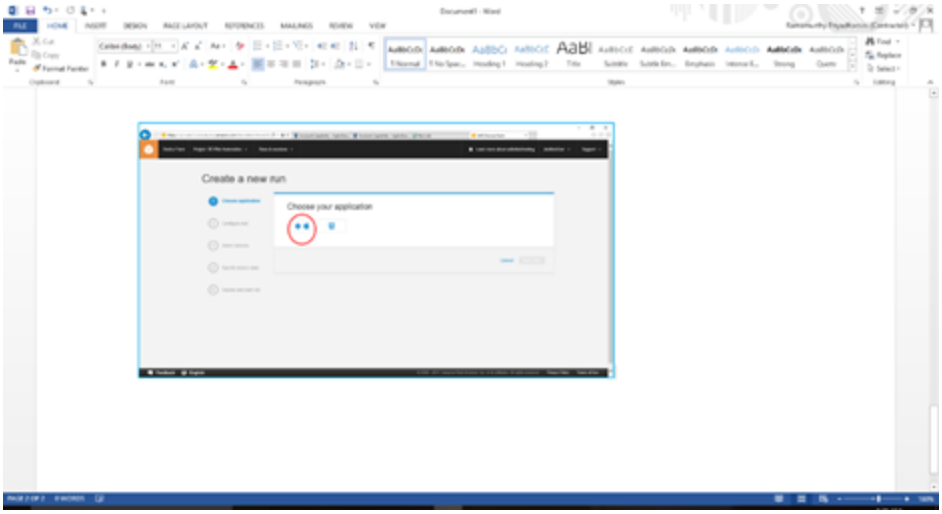
Step 4:

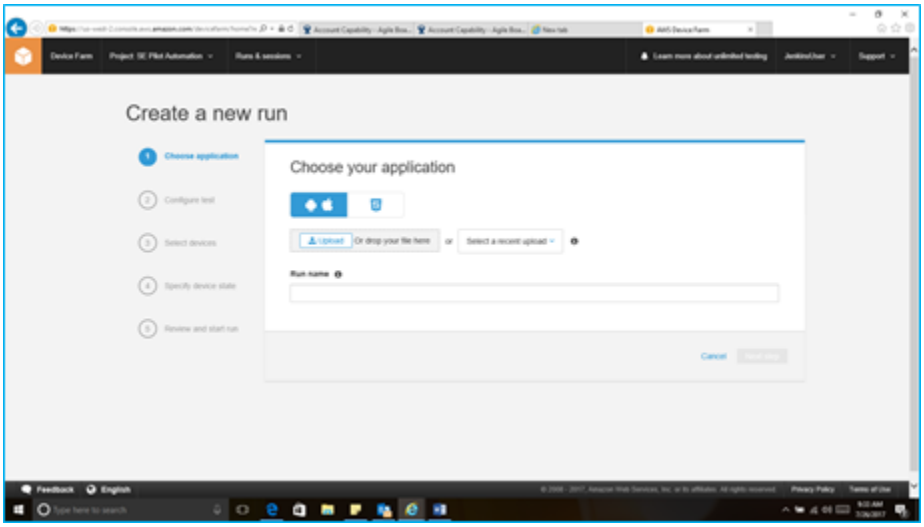
Click on "create a new run" button



Step 5:

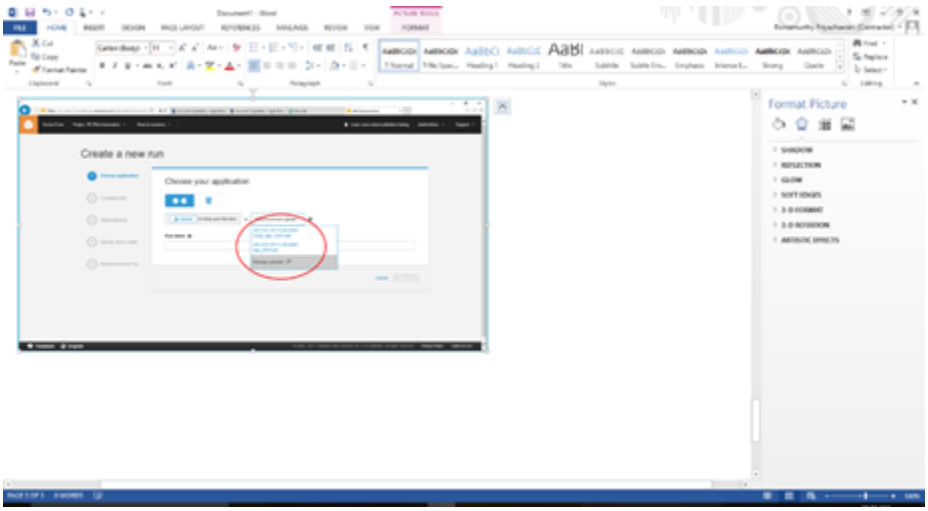
Click on the icon marked below





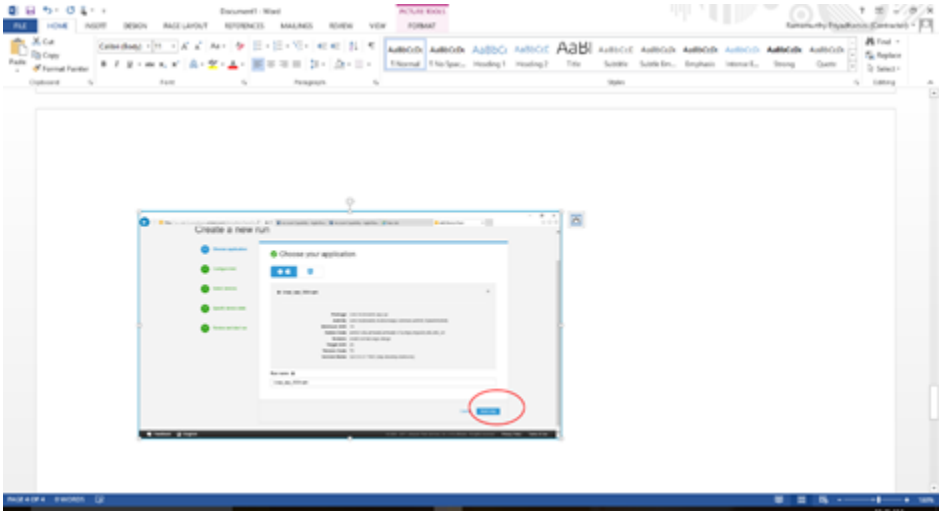
Step 6:

Upload a .apk (Android) or .ipa(IOS) file and once uploaded the file name will be listed in the "Select an recent upload" drop down



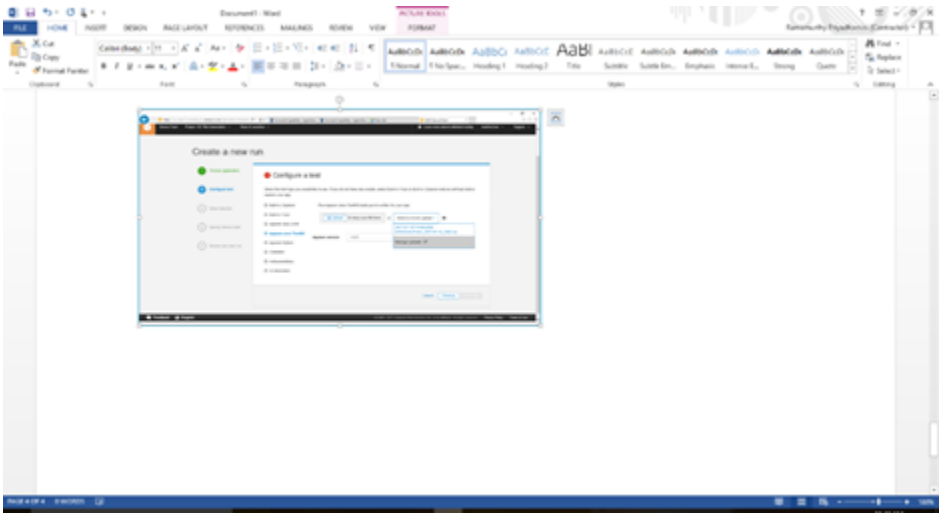
Step 7:

Click on the "Next Step" button



Step 8:

Select the "Appium Java TestNG" radio button and clicking on which will display a file upload field to upload the "AWS Device Farm.zip" file



Upload the file and click on "Next Step" button

Step 9:

Click on "Create a new device pool " button

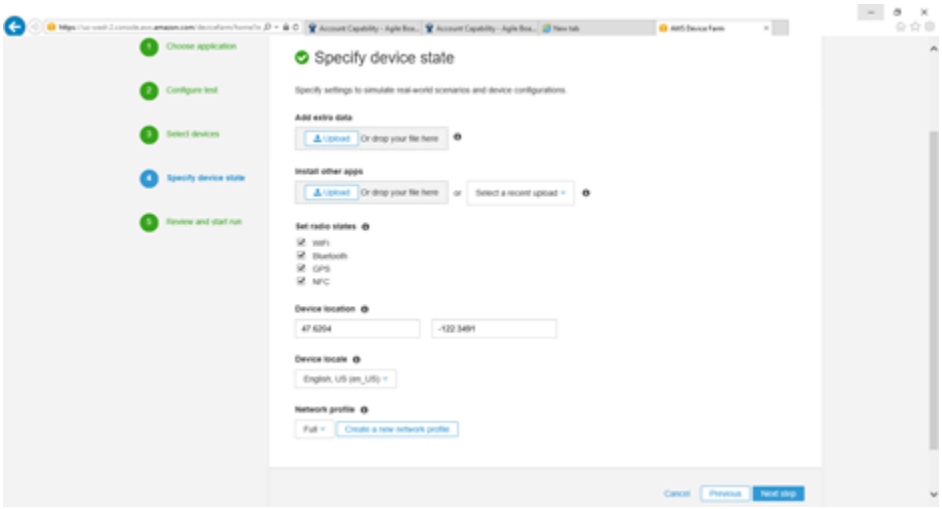
Enter the phone name that you are choosing, select the phone model that you want to test (For Eg: "Samsung Galaxy S6(Verizon)) and then click "Save a device pool" button

The chosen device will appear under "Device Pool" dropdown., select it and then click on "Next step" button

Step 10:

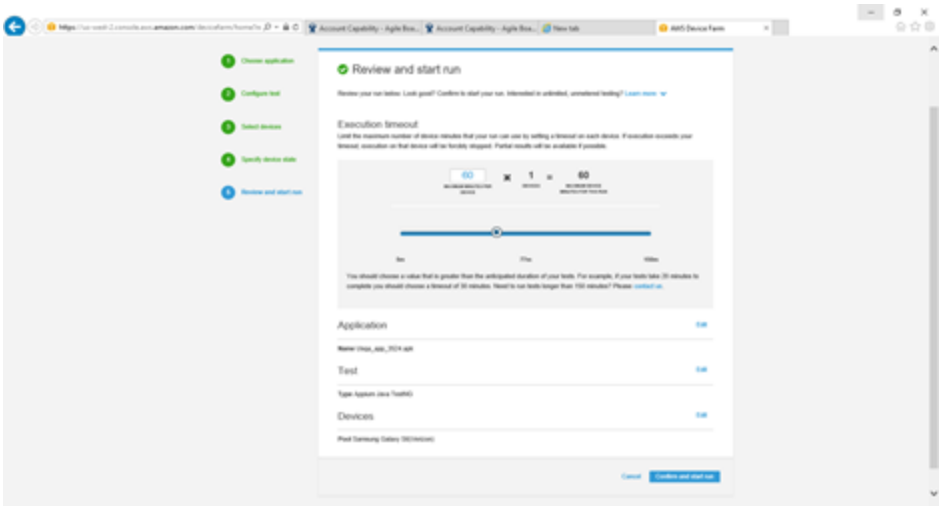
In the "Specify device state" page, No need to select anything as everything is preset

Click on the " Next Step " button



Step 11:

Based on the test cases that you are picking, you can increase the " Maximum Minutes per device"



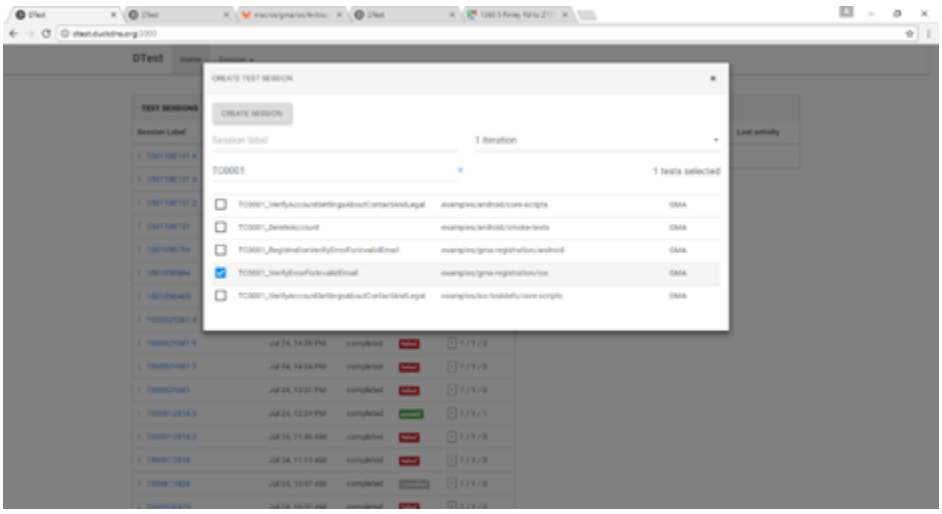
Before clicking on the "Confirm and start run" button , please do the below step

Step 12:

Get access for the "sample" branch and commit(push) all the codes that you want to test in the sample branch. As this branch is synced with the AWS Sync Service

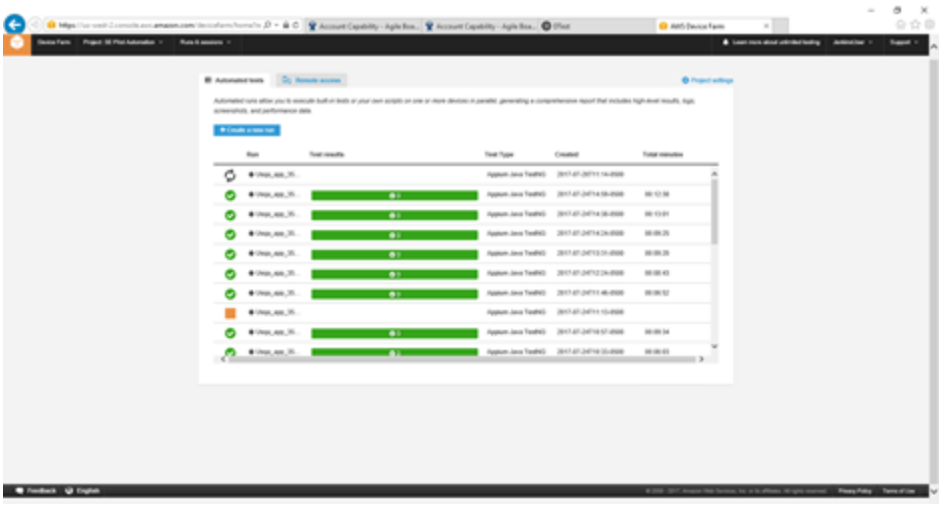
Open the "http://dtest.duckdns.org:3000/" in a separate browser and create the test session to run and then go to the AWS to click on the "Confirm and Run" button

[**NOTE:** Please get permission for accessing "sample" branch and commit(push) all the codes to the branch "sample" in the git before creating test session as this git is linked with AWS sync service]

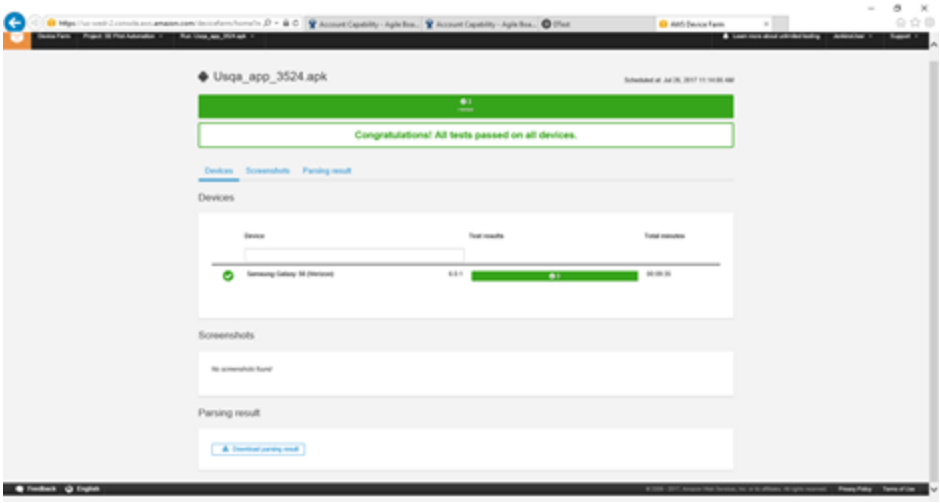


Step 13:

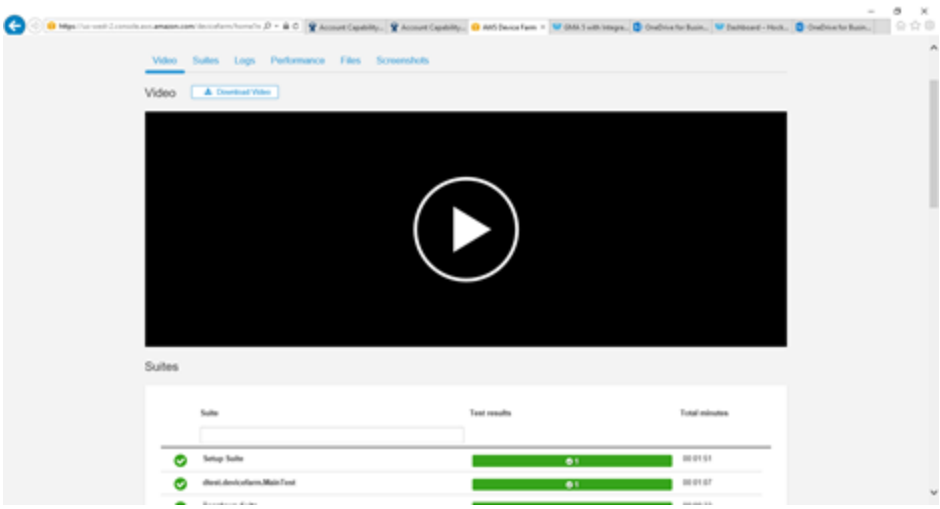
The below page will be displayed



Click on the latest run on the top to view the below page



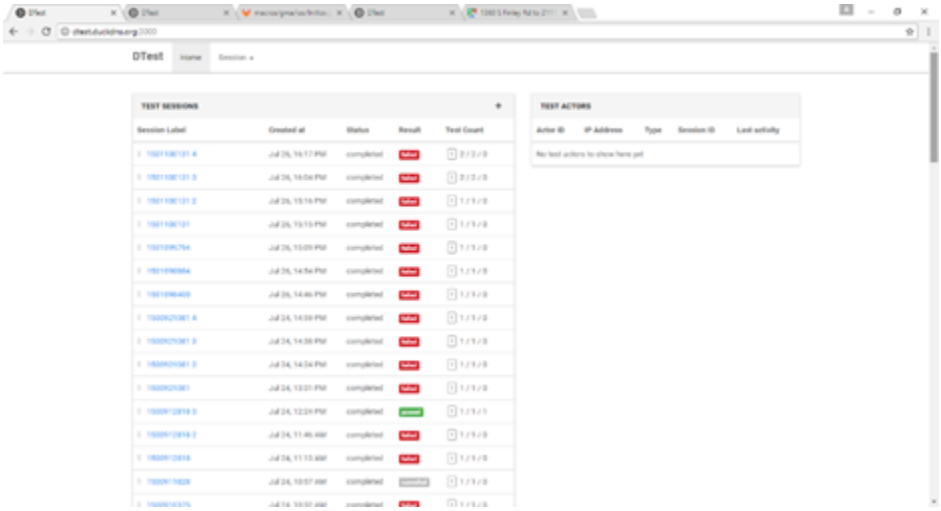
This shows that the device farm has been executed successfully and you will be able to view the video of the complete execution



The screenshots can also be viewed under the screenshots section

Step 14:

The test session results can also be viewed in the "http://dtest.duckdns.org:3000/"



AWS setup for IOS: - Will be updated soon

Basic GIT commands:

1. CLONE:

1) To clone server code into your local machine:

Create folder in your local. Go to that folder in command prompt and run below command:

git clone <repositoryLink>

To clone a particular branch.

git clone –b <branchName> <repositoryLink>

2. Branches:

1) List all branches :

git branch

2) Change current pointing branch

git branch <branch Name>

3) Check out the branch

First check all the branches in your local. If this branch is not available then only perform check out. If branch is already available in your local and running below command will generate new folder, which is not recommended.

git checkout <branchName>

3. Pull Updates:

1) Pull the latest code from the dev branch(If you have not done any changes in your local repository and if you want to take latest code) :

git pull

2) If you have updated some files in your local, then performing “git pull” will show below error message:

It means that other team member had already updated the same file and “git pull” may overwrite those files and you will lose your local changes.

- Perform below command to save your local work in some temporary folder:

  git stash

  This will store all your local changes to some temporary folder.

- Now fetch latest code from master:

  git pull

  Git pull will work successfully now.

- Once you pull the latest code from the master using “Git pull”, now you can apply your changes to latest code.

  You can pop your changes from temporary directory to local copy through below command.

  git stash pop

This will apply all your changes to local copy.

By performing this command git will show you all merge and conflicts.

3) Now you have to manually go to that files and remove conflicts and extra comments/lines added by Git.

For example from below code remove – Accepted current ….. , <<<<<<, ===== - all 3 lines.

4. COMMIT your changes:

Always fetch updated code from the dev branch before committing your changes (always perform “git pull”, solve merge conflicts and then run below commands)

- Add all your modified files to commit action

  git add . -->

  “.” Will add all your modified files to commit action. If you want to commit specific file then

  Run below command:

  git add <filename>

- Commit your files

  git commit –m “Commit message”

- Push your changes

  git push

This will push all your changes to the branch.

5. GIT Revert Process:

1) Clone your project on new location with require branch:

Git clone “Project URL”

Go to test-def folder

Git checkout dev

Git pull

Now you are on Dev branch with the latest commit.

2)Copy SHA (commit code), on which you want to revert your repository.

From Gitlab (web), go to “Commit” option, select your branch and you can get list of all commits. Select one from that list, for which you want to reset.

3)In your local folder, do hard reset on that commit.

Git reset --hard “sha number”

Now your local folder is on select commit. All your changes till that commit is available in your local folder <It will not include commits which had done after that select SHA)

4)Your local folder is now on require state.

5)If you check git status, you will get below message:

Your branch is behind 'origin/dev' by 5 commits because we have selected previous commit SHA, which doesn’t not include commits happed after that SHA.

6) To push your current state of repository on dev branch – git lab, do fast forward.

Git push –ff

This will update your Dev branch with the selected SHA.

6. Additional GIT Commands:

use git push/pull origin branchname -> if remote branch

if already on branch ---> do git push/git pull

git pull -all -> to pull all the files from remote to local
git status To check the current status (like how many files modified/added in local to be pushed to remote)
git checkout branch - switch to the branch
git checkout -b branch name
git merge sample(branch to be merged)

git stash apply - put the code back to the workspace in local --> we can use this when we try to pull the code from git

git gui --> is a gui to perform git commands like clone etc

git reset --hard copy the commitnumber from branch and paste here in order to reset to the previous commits

git rm -r folder name--> removes folder name for renaming directory/folder

gir rm filename removes file name

git mv oldfoldername newfoldername --> renaming folder

Always remove and then move in order to rename the files/folders

7. STEPS involved in pushing code to stash:

-> git clone stash url

->if the branch is empty, copy all the sources from local folder to the newly cloned stash folder
->add the changes (stage all changes) to git and then commit
git add . - staging all the files
git commit
->Then create a new branch
git checkout -b branch name

-> push the code to the new branch from local for which the code changes are already committed
git push origin branch name
-> We can use this for AWS Sync service

8. VSCode Useful Commands:

In order to format the test script into a proper language format like JavaScript, JSON, YAML files format, use the below:

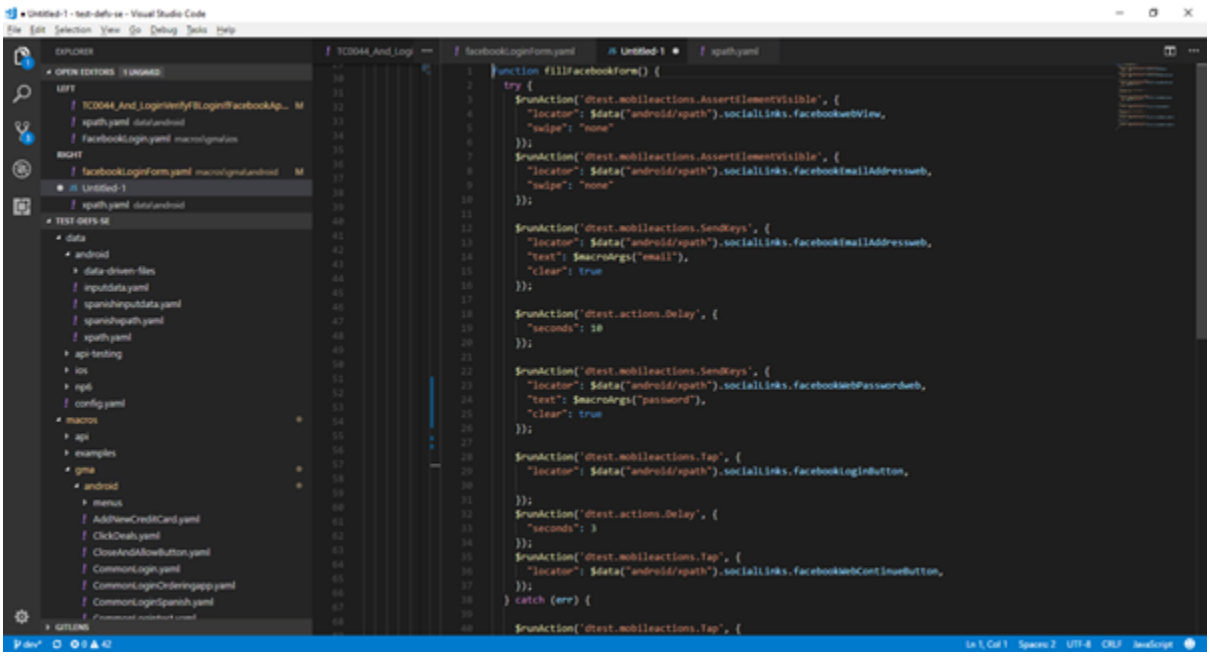
1. Create a new file, paste the script and in the bottom there is an option to select Language mode

For Ex:

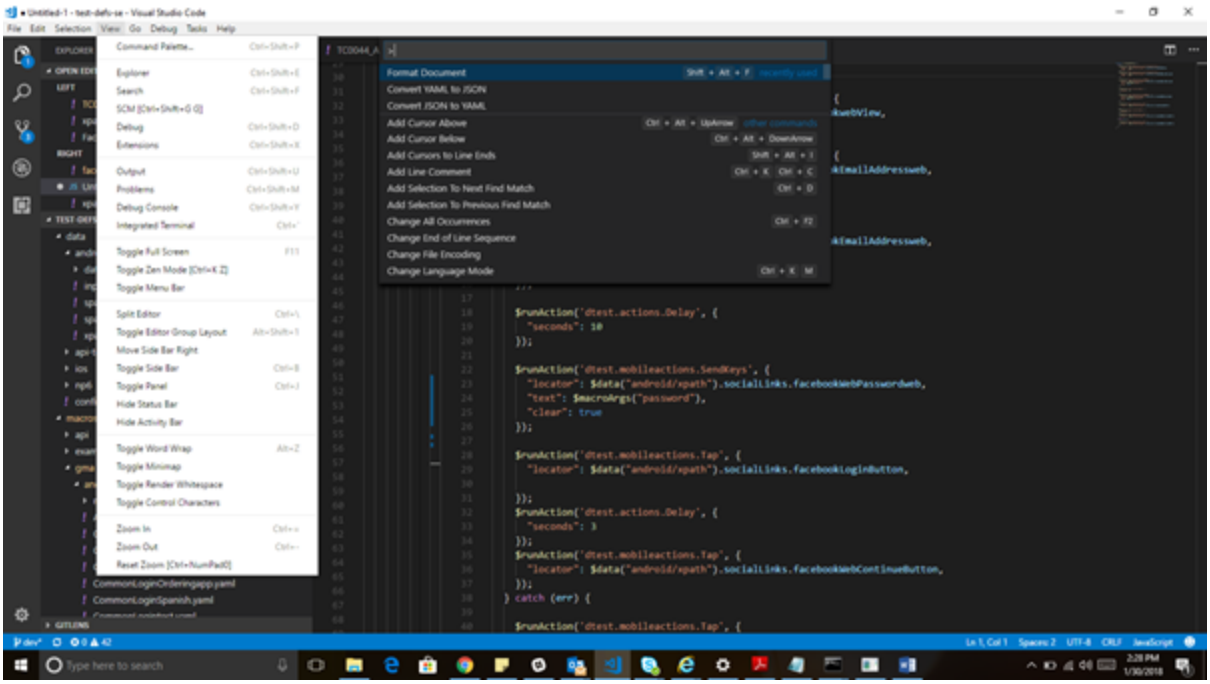
Select JavaScript for formatting JavaScript file and select the indentation –

Apply “Alt+Shift+F” to format it further

Ex:



Also, you can use “View ->Command Palette” to identify the key assist operations for VSCODE



References:

1. Open Test Technical Reference **-** OpenTestTechnicalReference-Data-driven Testing **This link is a reference for writing scripts using data driven approach**
2. How to run test cases in OpenTest using template **-** please refer this link for template creation and execution
3. OpenTest Tutorial: https://us-confluence.mcd.com/display/SE/OpenTest+Training+Videos
4. Query Tracker: https://us-confluence.mcd.com/display/SE/Dtest+Framework+-+Sapient+Query+Tracker

Faq:

1. Usage of Dev/CI integration environment for execution:

We can use Dev/CI integration envs for our execution. First they will deploy code in Dev env, if everything will go smooth in Dev env then they will deploy same code into CI integration env. If Dev env will be down cause of deployment then you can use CI integration env to compete your stories. To access CI integration urls, open https://dev2.lb.anypointdns.net/ci/exp/v1/customer/login url once in browser where you'll see browser details and show details options, click on show details option and click 'OK'. You'll get below response

{"status": {"code": 40000,"type": "ValidationException","message": "Mandatory Informaiton Missing.","errors": [{"code": 41402,"type": "BadRequestException","message": "Client id is missing in the request.","property": "mcd-clientid","path": "/exp/v1/customer/login","service": "ci-identity-exp-api"}]}}

It'll basically install certificate on your machine, so you need to do this once. After that, you can hit any CI URL from postman.