Overview

This document provides the detail description about terminologies that are used in documentation, defining the methods used in ARES Dashboard and its integration with different automation frameworks.

Contents

Links

- **Dashboard URL:** http://testastra.com/
- **GitHub URL:** https://github.com/testastra/ARES

ARES Dashboard Terminologies

ARES dashboard: Name of the Dashboard.
token/UserToken/UserKey: A token will be generated upon creating project in dashboard for every user, which is user specific
ProjectKey/projectId: A project key will get generated upon creating project in dashboard that can be used while posting test results to dashboard, which is unique to every project.
run Id: An alpha-numeric number generated for every test suite run in ARES dashboard using createRunIdDetails method
ws_name: workspace name, you can get it from dashboard homepage (Dashboard URL)

ARES Dashboard Implementation Steps

ARES Dashboard is implemented by 3 steps
***Step1*:** At suite level
Step2: At module level
Step3: At test level

Posting results to Dashboard involves 3 Steps:

Step 1: At Suite Level

Creating Run ID: This will be created only once for each suite run. Following is the helper method to create Run ID (needed for step2 and step3.)

Method: createRunId (ws_name, status, project name)

- **ws_name** is workspace name which can be any for now.
- **status** is always'started'.
- **project_name** is name of project created in Dashboard.

Step 2: At Module Level:

Posting Module Data: After Run ID is created, we need to post module data. This API method with the following parameters is called twice for each module. Ideally, before and after module execution, which will help to represent module/functional area graph in dashboard where number of tests executed both passed and failed are shown in dashboard.

Method: createModule (runId, ws_name, project_name, moduleName, startTime, totalTests, moduleStatus)

- **runId** created from the above method (createRunId) is used here.
- **ws_name** is workspace name which can be any for now.
- **project_name** is name of project created in Dashboard.
- **moduleName** is name of the module.
- **startTime** is time at which test execution started.
- **totalTests** is no of tests in test suite.
- **moduleStatus** indicates starting and ending of test Execution.

Step 3: At Test Level:

Posting Test Results: After each test execution, the result should be posted to Dashboard. For this we use the following API Helper Methods

Method: apiRequest (runId, status, testStartTime, testEndTime, testBrowser, executionMode, failedStackTrace, errorMessage, testDevice, testOs, testDate, runBy, productName, moduleName, testSuite, testcaseTitle, testData, imageLink, videoLink, failType, testMachine)

- **runId** - Created from the above method (createRunId) is used here.
- **status** - Test script result i.e. either of Passed/Failed
- **testStartTime** - Test execution start time
- **testEndTime** - Test execution end time
- **testBrowser** - Browser name on which test execution happens
- **executionMode** - Test execution mode i.e. Linear/Remote
- **failedStackTrace** - Error stack trace for failed tests
- **errorMessage** - Cause of the failure for failed tests
- **testDevice** - Device on which scripts are executed
- **testOs** - Operating system on which scripts are executed
- **testDate** - The date when test scripts are executed
- **runBy** - The name of the person who triggered the tests
- **productName** - The name of the product
- **moduleName** - Module of the product
- **testSuite** - Currently running test suite name
- **testcaseTitle** - Test Case (Test method) being executed
- **testData** - Test Data used for the test case execution
- **imageLink** - The path to screenshot for failed test case
- **videoLink** - The path test execution video
- **failType** - Type of failure
- **testMachine** - Name of machine name on which scripts are executed

Dashboard integration with Automation Framework

Project creation and API details acquisition

- Navigate to dashboardURL
- Login to dashboard and create a project
- Copy API details i.e. **ProjectKey** and **UserKey** of the created project, project_ name and ws _name.

CSharp (MS Test):

Integration with automation framework

- Get/Clone automation framework repository from the **GitHub** URL
- Go to CSharpfolder
- Copy **Dashboard** folder from the project into your project
- Copy config.txt and rundetails.txt files into your project root folder
- Replace the following keys with your project properties acquired while creating project in dashboard.

  project_name, ws_Name, ProjectKey, User Token

Dashboard folder contents:

- **AresDashboard.cs** file that contains API methods defined, which are useful in posting results to dashboard.

Calling API Post Request methods at different levels in framework:

- **Calling Run ID Creation Method:**

  As this will be executed only once for entire test run, the ideal position to place this code is **Assembly Initialize** of **Base Class.**
- **Posting Module Details:**

  As we need to call this for every module, the ideal position to place this method is in **Class Initialize** method of **Test Class**
- **Posting Test Case Details:**

  As we need to post each test case details after execution, the ideal position to call this method is Test Cleanup method in Test Class

JAVA:

Integration with automation framework

- Get/Clone automation framework repository from the GitHub URL
- Go to java folder
- Copy **listener** and **ZenQ_Dashboard** folders resides in 'src/test/java' into your project
- Copy config. Properties file into your project root folder
- Replace the following keys with your project properties. Project name, ws_name, project key and user token can be copied from the created project in dashboard

  ProjectName_ARES, ws_Name_ARES, ProjectKey_ARES, UserToken_ARES, ProductName_ARES, ProjectUser_ARES, TestDevice_ARES

ZenQ_Dashboard folder contents:

- **AresDashboard.java** file that contains API methods defined, which are useful in posting results to dashboard and API methods are called from Test Listener java file
- **DashboardResources.java** where all dashboard properties getter methods defined

Note: Please specify your project config. properties path in DashboardResources java class static block
Note: Please specify listener class file at suite level or test class level

JAVASCRIPT:

Integration with automation framework

- Get/Clone automation framework repository from the GitHub URL
- Go to JavaScript folder
- Copy JavaScript folder from the project into your project
- Replace the following keys with your project properties acquired while creating project in dashboard.

  ws_name, project_name, token, projectId
- Write the ws_name and project_name in moduleBody.js file

JavaScript folder contents:

- apiMethods folder which have the **postCreatRunIdrequest.js, postCreateModuleRequest.js**, **postTestResultRequest.js** files, which are useful in posting results to dashboard.

Using dashboard from the project directly:

- Install Node V 7.x.x or greater
- NPM install
- Webdriver-manager update
- Webdriver-manager start
- We run the test by protractor conf.js

projectId and userToken are copied to **apiData.js**, where projectId is the id created after making the sample project and usertoken is the token created after making the sample project.