RS

IQ

IST_Quality

Private group

Not following Next steps

6 members

New Send by email Page detai Draft saved 11/29/2018 Edit Publish

Test case for automation

Writing Test Case For Automation

Overview

The application under test is automated using the hybrid design concepts of feature and data driven automation framework. Framework-utils is added as a dependency to leverage the standard utilities.

A test project is created for the application and two separate packages are created to distinguish or separate the tests from the features of the applications.

1) The features of the application are categorized into one package group called as support-module under /src/main/java. Assertions should not be written here and the modules should return objects which can be used within the tests for assertions and validations.

2) The test cases are written in the test package under /src/test/java. All the assertions are to be performed in test cases.

In this way, the functionality of the application is separated to the actual test cases. The support module packages are a complete separate project by itself and can be used by external projects to run any feature of the application by adding the project as a dependency.

Features:

- Gradle for build management and execution. Maven can be chosen as well depending upon the comfort level within the team.
- TestNg as test harness to execute the test cases.
- Selenium wrappers are used from the framework-utils to interact with the web components.
- CSV files with input data are used in case of multiple input parameters to the test cases.

- Support continuous integration and execution of test cases under different environment by changing system properties via Jenkins or gradle.
- Reporting using extent reports tool. it reports all the assertions and the stack trace if there are failures. Test reports are available on report-dashboard as well.
- Soft assertions are used; This helps in executing all the assertions instead of failing the test case upon first assertion failure.
- Terminate the test execution if there is an exception at any step. This can be achieved by carefully automating the features of the application under support modules packaged and throwing the exceptions.
- Properties management is provided based on different environment, properties can be changed during run time by passing command line parameters.
- Multi-threading or parallel test cases execution is supported but the test cases must be executed in a certain order that they don't interfere with each other on their reliance on database or environment. TestNg xml needs to be configured for it.

Test cases structure for Selenium framework

- **Structure**: The support modules classes can have methods to automate the features of the application. The test case classes/methods will create the object of the classes from the support module packages and then appropriate methods should be called to perform validations. For more details please check the existing test cases.

- **Groups**: Each test case should have the group categorization, the group categorization is done based on the understanding whether the test should be part of regression, BVT or any feature of the application. This will help in running the specific test when needed. Convention for grouping of test cases. use hyphen between words.

@Test (groups ="image-submission-portal", "regressions"})

- **Execution**: The execution of test will be done from the testng.xml file. Different xml files are created based on the execution requirements. for example, regressiontest.xml, BVT.xml, application.xml etc. The xmls will be defined and executed in the build.gradle file and can be overwritten from command line parameters. The xml file can be executed from eclipse ide as well during local development.

- **Multi Browser config**: UI test cases should be enabled to run on any browser and platform. Such configurations are passed to the tests from the <testng.xml> file as parameters along,

<test name=mageSubmissionPortalOnChrome'>

<parameter name=uot;bType" value=uot;chrome" />

<parameter name=uot;bVersion" value=uot;" />

<parameter name=uot;platform" value=uot;" />

..... so on

- **Assertions**: As mentioned before, custom soft assertion are used in test classes/methods. The assertions are captured to be printed in the output report as well. Usage sample code:

```
@Test (dataProvider =/span>"csv", groups =“image-submission-portal", "regressions" })

public void ValidateLandingPage(Map<String, String> inputDatamap) {

        ReportLogger reportLogger =ew ReportLogger();

        int status 0;

reportLogger.assertEquals(status, 200, "Validate response status is 200" + status);

        --other code

}
```

- **Parameterization**: While creating application support classes and test cases, take into consideration on how many input variables need to be passed to the test. More detailed test samples are below.

There are three ways in which we can write test cases based on the number of varying input parameters that we need to send to the test case.

1) **No input variables**: In this case, the test can be written using standard @test annotation from testing.

2) **Max of up to 2-3 input variables**: In this case, pass the test parameters from the testng.xml file.

Sample Test case and xml content is

```
@Test(groups = regressions" })

@Parameters("url")

public void ValidateGetRoleDataService(String url, ITestContext context) {

        //below statement is added to print the params in the html report.

                context.setAttribute("URL", url);


 --Some code suing url

 }
```

To run this test case the xml content is

```
<test name=uot;Sample Test">

    <parameter name=uot;url" value=uot;someurl" />

    <classes>

       <class

          name=uot;sometest" />
```

```
    </classes>

 </test>
```

More than 3 parameters: It is hard to pass parameters from xml file when the number of input parameters are more than three. CSV files can be used to pass parameter and use testng data provider to parse and pass them to the test cases. The CSV file needs to have a header with field names and multiple data rows mapping with the header. A map is created by data provider where the keys are the first line of the CSV file and data row forms the data. Start and end row defines how many iterations of the csv file will be performed, each row constitutes a test case. The CSV file should have first two columns as “TestCaseNo” and “TestDescription”.

Sample Test case and xml content is

```
public class SampleClass {

@Test(groups = "regressions" }, dataProvider =quot;csv")

public void Test(Map<String, String> inputDataMap) {

    // test code

}
```

To run this test testng xml cotent is

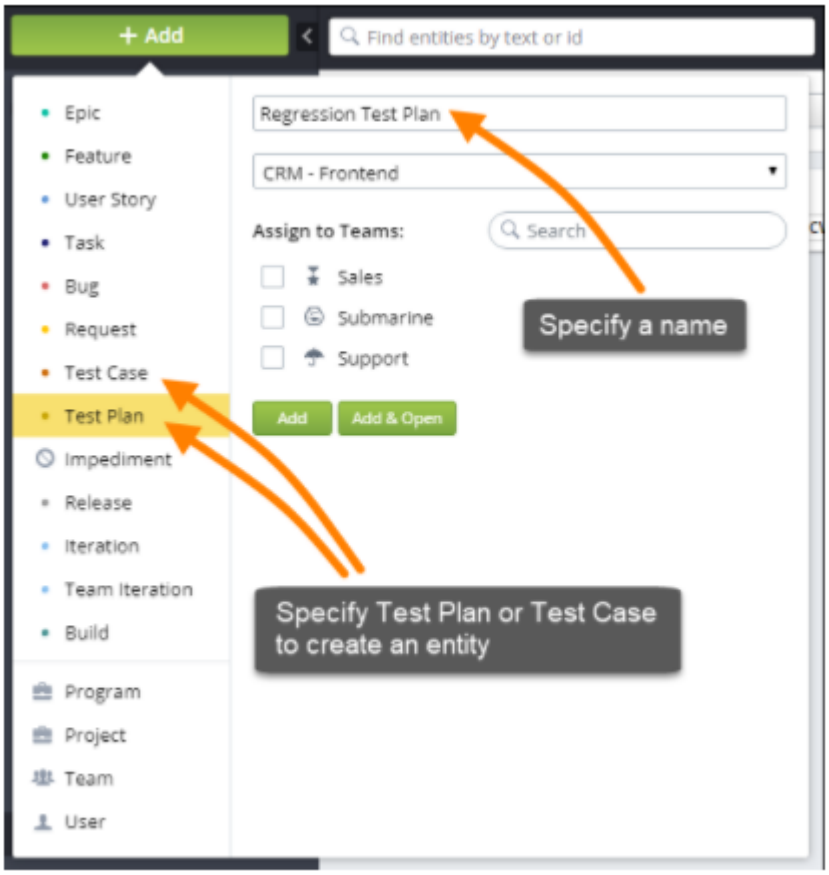
```
<test name=uot;Sample test">

    <parameter name=uot;fileName"  value=uot;CsvFilePath.csv" />

    <parameter name=uot;startRow" value=uot;1" />

    <parameter name=uot;endRow" value=uot;9" />

    <classes>

        <class

            name=uot;SampleTestWithRelativePathFromProjectHome" />

    </classes>

</test>
```

Target process Test case Creation

There are many places in Targetprocess where you can create a new test case and test plan from.

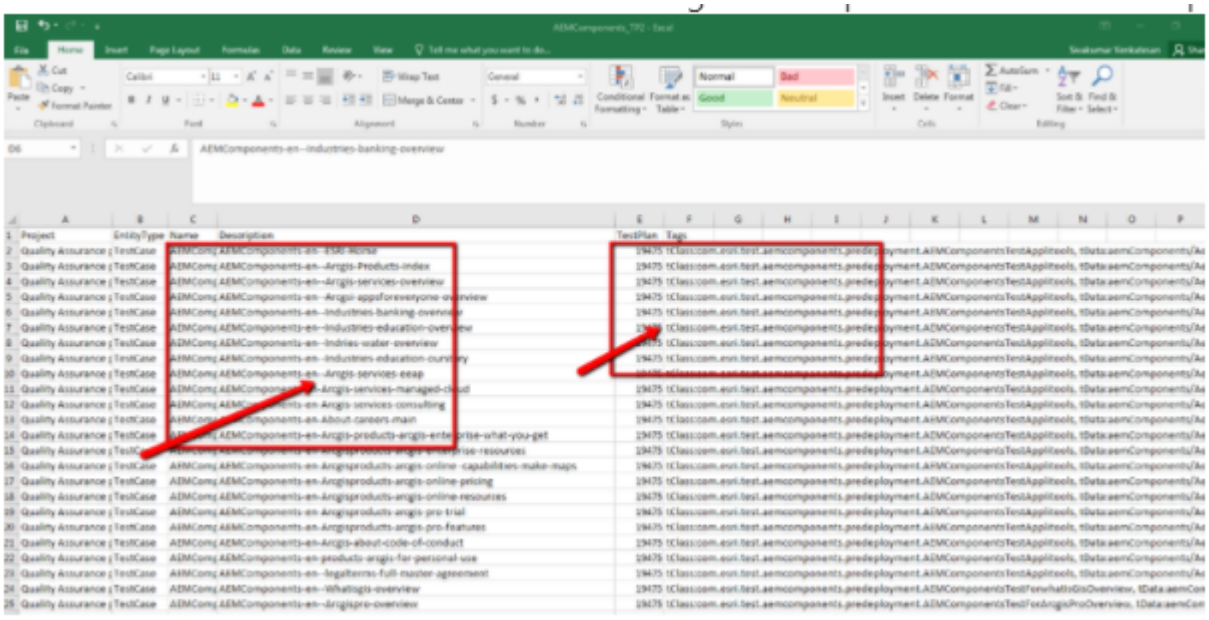
Green Add button

Click [+] Add green button, select a test plan/case entity and provide a name. Test case/plan will be added to the project and a team which you specify during the creation.

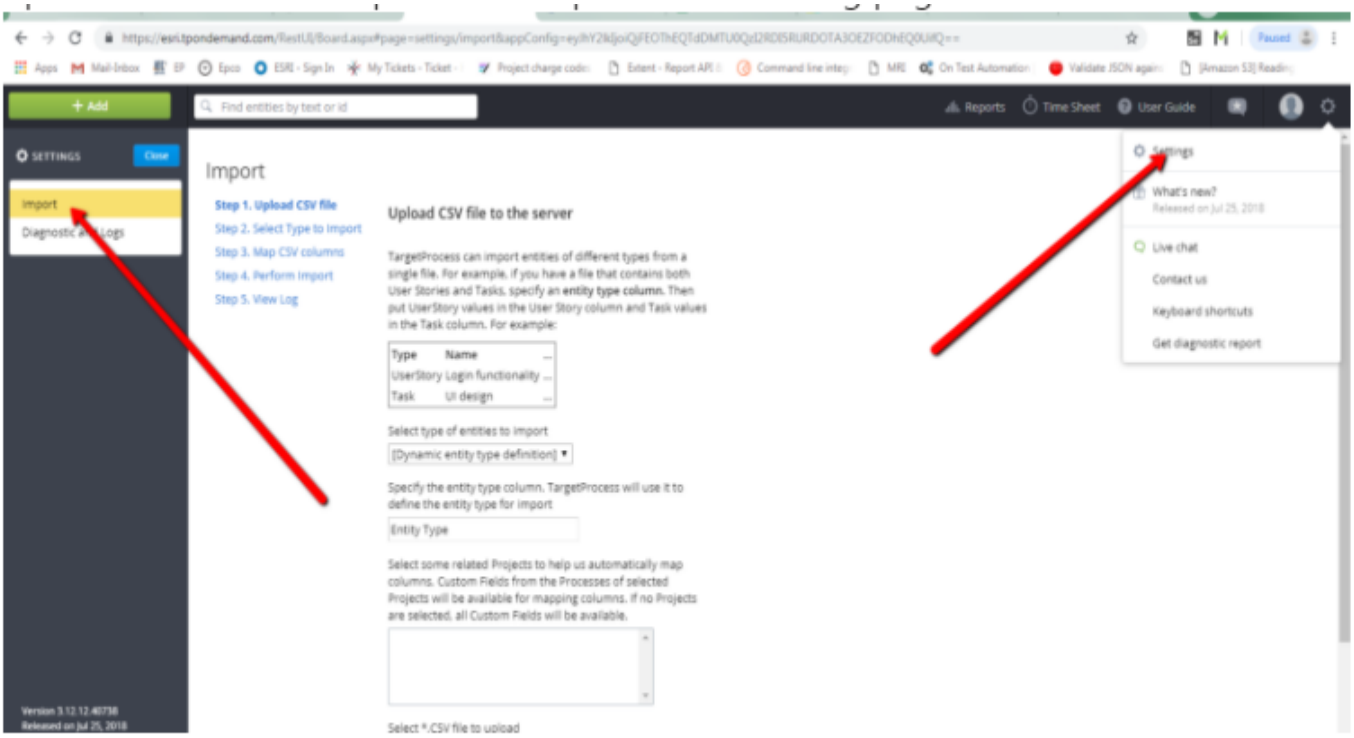


Step to create Test case and Test plan:

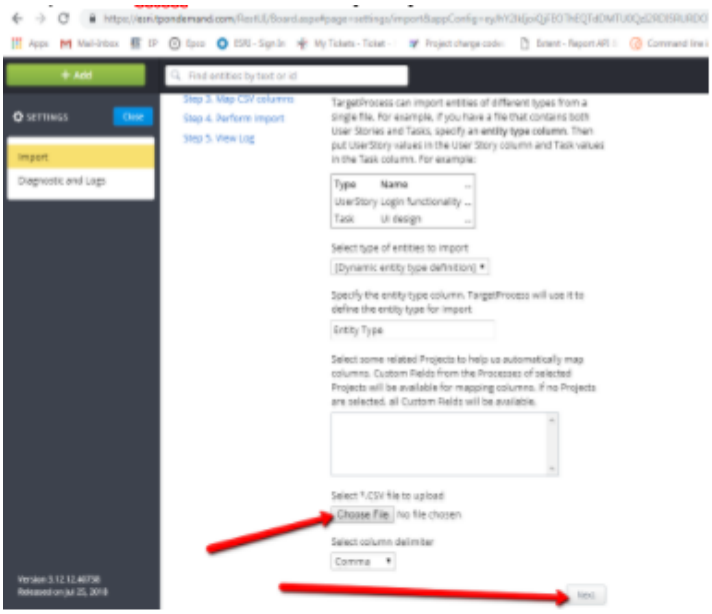
1. Create new test plan
2. Create CSV file Test Case for Different test class as tag and test plan id in CSV file Sample csv :

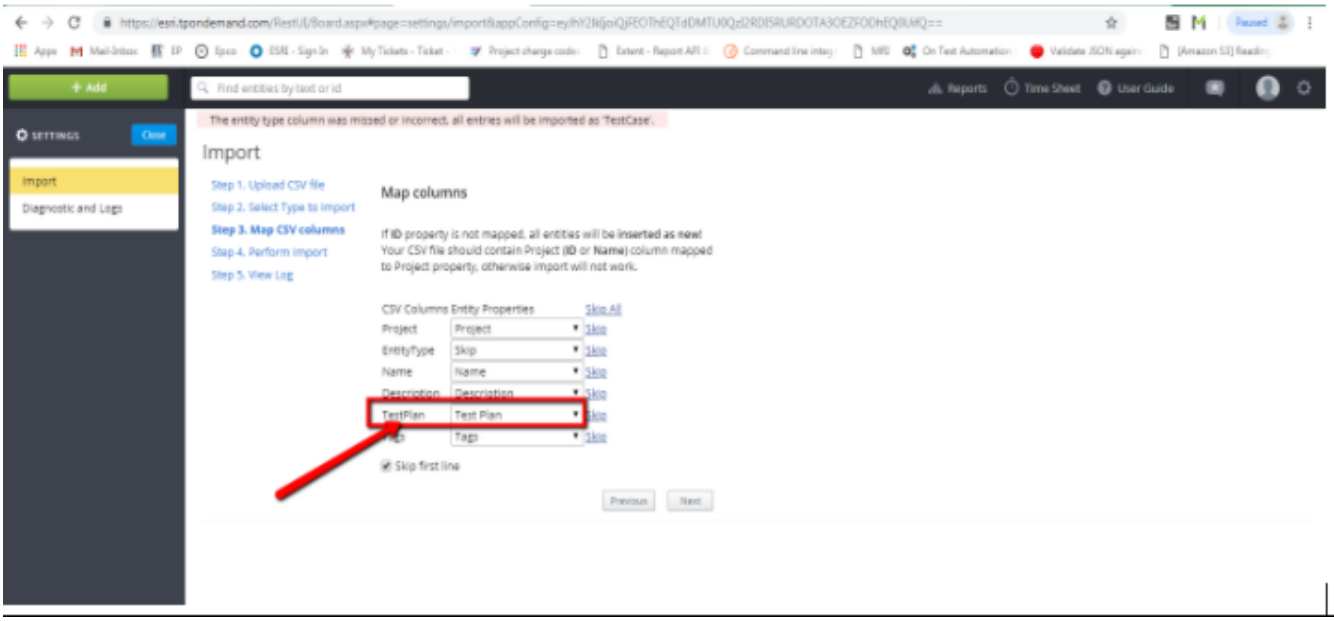


3.upload Test Case in respective Test plan throw setting page.



4.Upload CSV with Primary key as Test Plan id





5. Once Successful Upload test case will displayed under Test plan Id.