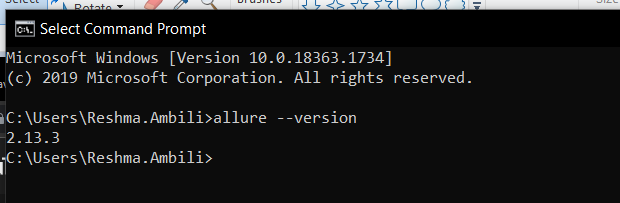
1st check whether we have allure is installed in our system by allure - -version

If not installed it

<https://repo.maven.apache.org/maven2/io/qameta/allure/allure-commandline/2.13.3/>

download—unzip—copy path till bin and add in environment variables

select path—new—add above path and then check in cmd.



Java se should be 1.8 minimum

Now add all the dependencies for the project + allure dependency

<!-- https://mvnrepository.com/artifact/io.qameta.allure/allure-testng -->

<dependency>

<groupId>io.qameta.allure</groupId>

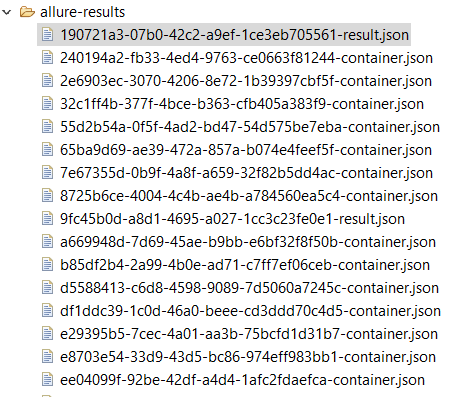
<artifactId>allure-testng</artifactId>

<version>2.13.3</version>

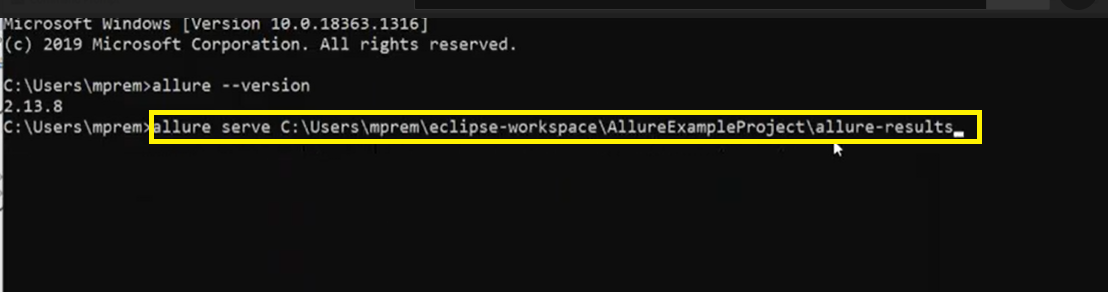
</dependency>

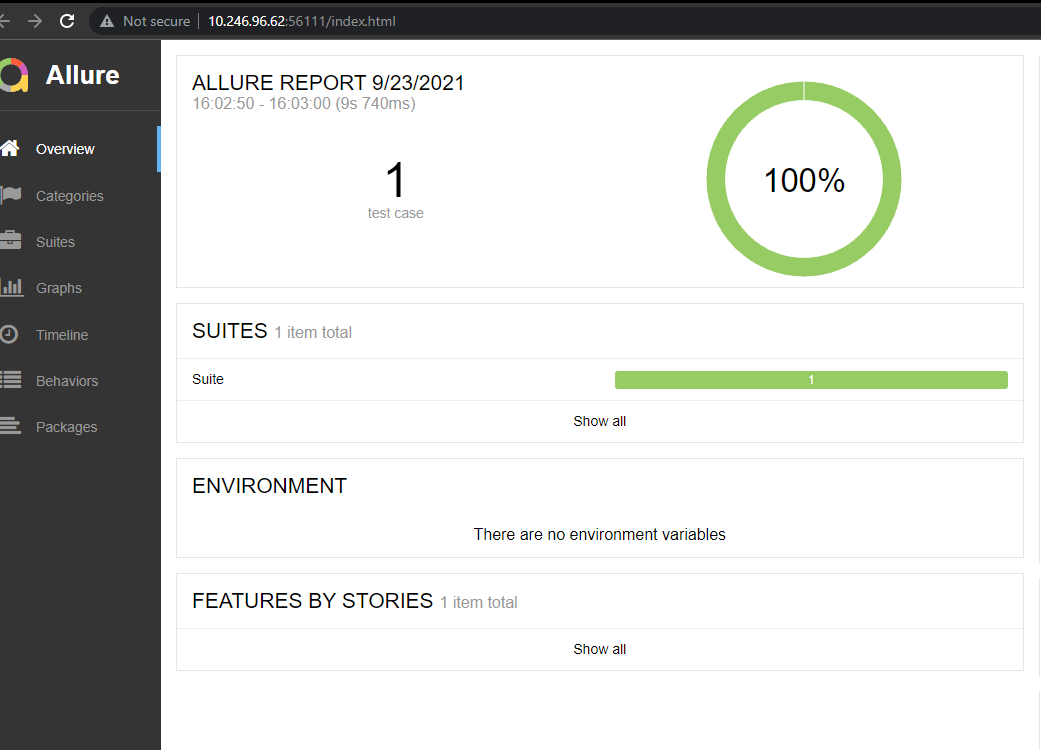
Run test through pom.xml

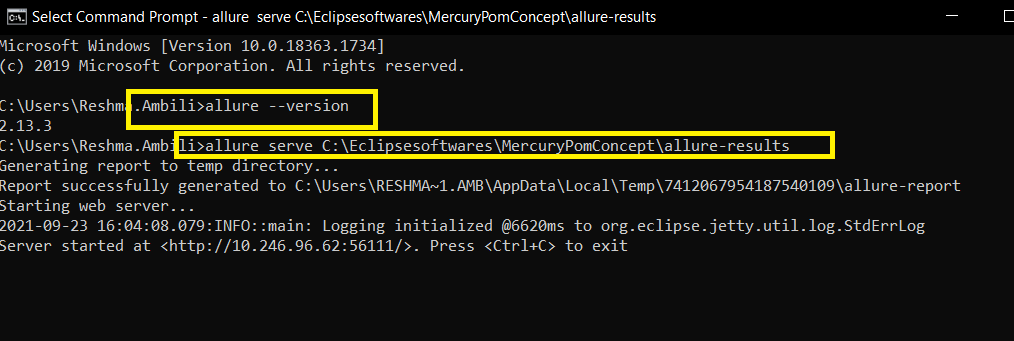
Refresh the project will get below allure reports folder with json files. So for reading this file we need allure command line tools which has already downloaded from <https://repo.maven.apache.org/maven2/io/qameta/allure/allure-commandline/2.13.3/>



Now copy allure results path from properties and do below action, it will start loading the results in html format







So to get Step annotation and all…need to run with pom xml rather than testng.xml alone.

Customized report and lightweight

After compiler plug in we have to add out surefire plug in

It’s a plugin inside our maven…how maven works

**Setting the -source and -target of the Java Compiler**

Sometimes when you may need to compile a certain project to a different version than what you are currently using. The javac can accept such command using -source and -target. The Compiler Plugin can also be configured to provide these options during compilation.

For example, if you want to use the Java 8 language features (-source 1.8) and also want the compiled classes to be compatible with JVM 1.8 (-target 1.8), you can either add the two following properties, which are the default property names for the plugin parameters:

1. <project>
2. [...]
3. <properties>
4. <maven.compiler.source>1.8</maven.compiler.source>
5. <maven.compiler.target>1.8</maven.compiler.target>
6. </properties>
7. [...]
8. </project>

<plugins>

<!-- Compiler plug-in -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

Javaagent will be downloaded when we build the project

@step annotation can write in every methods in page classes, which will be available in html report.

Login with username and password—append with number of parameters if it 4parameters then 0,1…4

Username is 0 parameter and password is 1

Allure plugin is added in POM .xml so have to be executed by maven way from pom.xml,if we execute from testng allure plugin wont be triggered

Maven will check for sugarfire plugin which is responsible for running tcs we have added the suite testng file here

Same time it will download java agent and the aspect dependency for allure results

**Retry Analyzer**

If 8/10 got failed so that time I am giving one more chance to that failures(may be coz of xpath issue/elemnt o found/not interactable etc)

2ways

@test level

@run time please handle those testcases

IRetryAnalyzer—testng listener available. with the help of this we can implement this logic

Method in this is Retrymethod

Best approach is @run time

**Flow**

1st give countereg Counter=3

Each and every failures that counter count will be given to that tcs

1st chance failed 2nd passed then Tcase=passed

So if we haven’t implemented retry analyser and 1st chance failed means testcase=failed

RetryAnalyzer class(any name for this class) will implements IRetryAnalyzer interface

Now set the counter =0

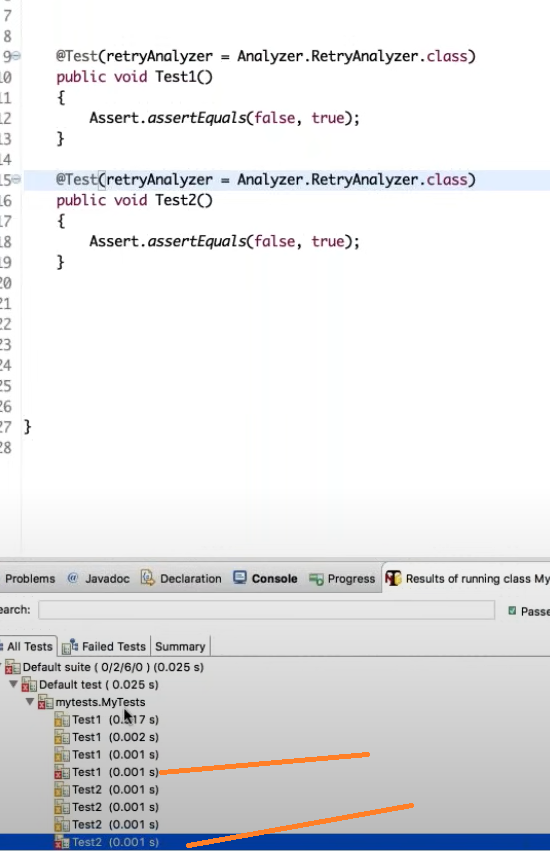
Wherever in @test we are applying this analyzer that will be giving the chance based on counter given

Here counter was 3

So got a chance to retry/run those faild cases for 3 times and 4th time it got failed.

So its not a good approach coz if we have multiple/50 test cases we have repeat this in every @test

So better approach is

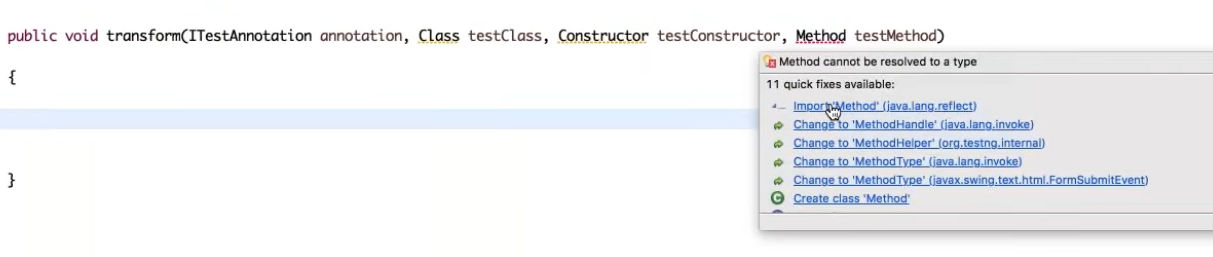


So better approach is check @run time

Another method called transform() from IAnnotationTransformerlistener

So create another class (any name) and implements IAnnotationTransformer

Then method transform from above listener

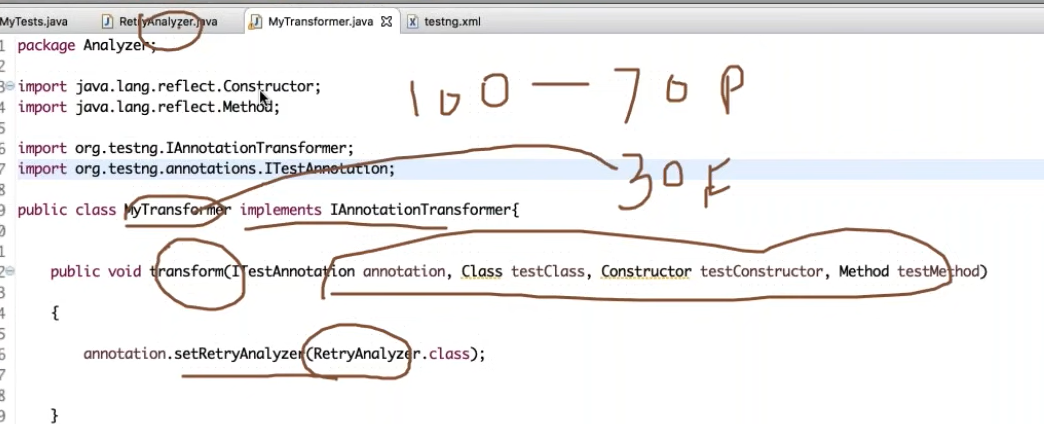


TestNg Listeners all are build up on reflections api in java

After done this transform code

Just add this listener in testg.xml

Flow when failed



It will come to below class and check the limit

