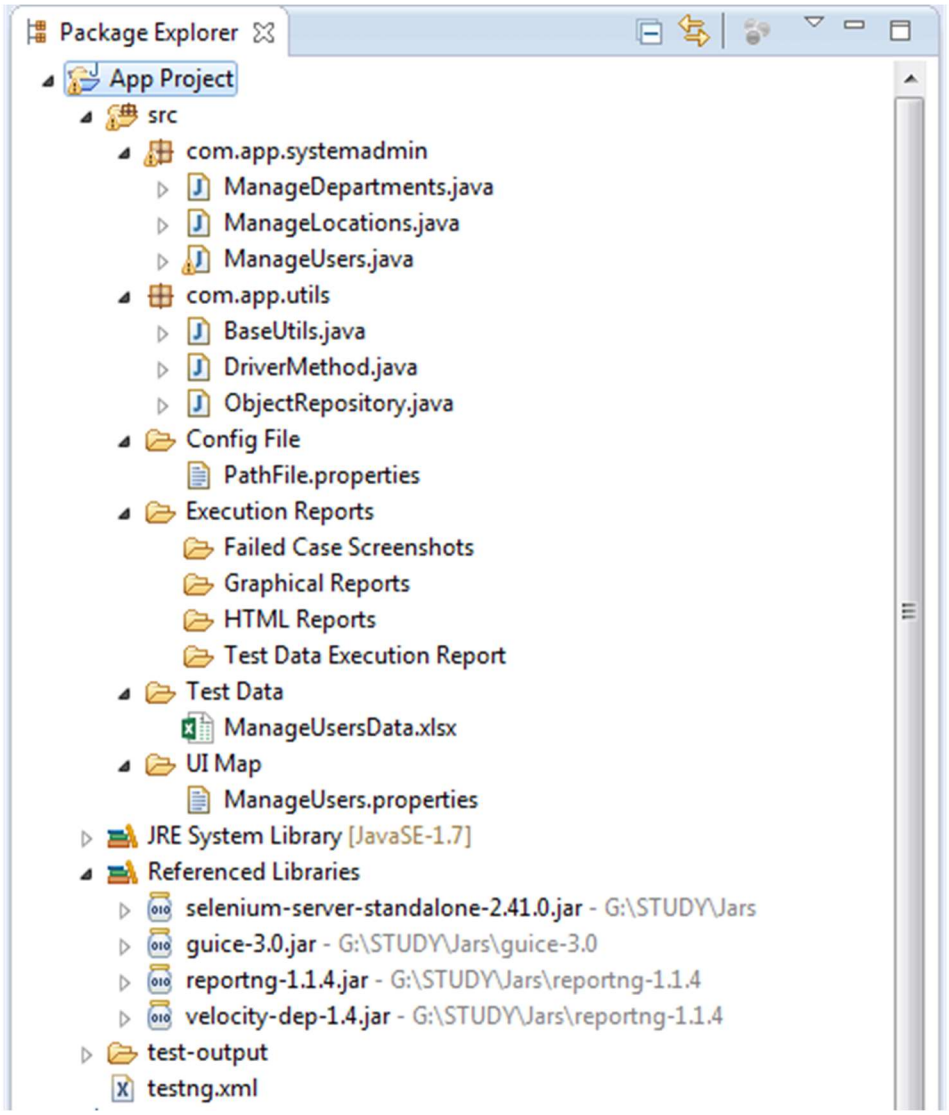
1. BASIC TESTNG SELENIUM JAVA PROJECT STRUCTURE WITH DETAILS ON EACH ITEM

Project Structure details 1 –



Project Structure details with additional packages –

src

com.app.Pages

Home_Page.java

LogIn_Page.java

com.app.Test

FirstTestCase.java

SecondTestCase.java

```
        lib
          drivers
              IEDreiverServer.exe
             Chromedriver.exe t
testng.xml
POM.xml
```

2. Sample UI Map / Object Repository

UIMap is a concept for defining, storing, and serving UI elements of an application or a website. The UIMap properties file contains a set of 'key-value' pairs, where key is an alias of the UI element, and a value is the locator.

We will create properties file for every single page and capture all the UI elements present on the page and use it as per needs.

UI Map or Object Repository Samples: -

Role - System Admin

ManageUsers.Properties
ManageDepartments.Properties
ManageLocations.Properties

Example:-

ManageUsers.Properties

ManageUsers.properties

```
#These properties belongs to Manage Users
#--------------------------------------

#PageTitle
userPageTitle=Manage Users

#To Click on Add User in Manage Users Grid
addUserButton=name=add user

#To enter user first name
firstName=id=fname

#To enter user middle name
middleName=id=mname

#To enter user last name
lastName=id=lname

#To Enter email address
emailAddress=id=email

#To enter contact number details
contactNumber=name=contact

#To click on submit button
submitButton=name=submit
```

Why and How to use UI Map / Object Repository / OR

To perform any action we need locators. Tool will perform any action based on the locators. For each and every action tool depends on these locators. If the tool does not identify the locators, it simply throws an error as Locator / Element Not Found or Identified.

In order to make sure the tool executes smoothly, we need to provide accurate unique identifier / locators. We need to keep all the locators at one place where we can easily modify the locators / identifies if there are any UI changes in the application.

If not, it will become difficult to change even one locator, as it will be used at many different places in test scripts. It is always better to keep locators in a separate file and at easily accessible location. It should be defined in such a way if any new person joins in the middle of the project, he/she should be capable of making changes / adding any new locators to the file.

In the above example, we have created properties file for each function which will store all the UI elements.
Example: In Manage Users, we will store all the locators' / UI elements information which can be used and accessed in "Add Users", "Edit Users" and "Delete Users" test cases

3. Sample Data Set / Test Data

Data set stores the data files, Script reads test data from external data sources and executes test based on it.

Typically the data input can be anything:

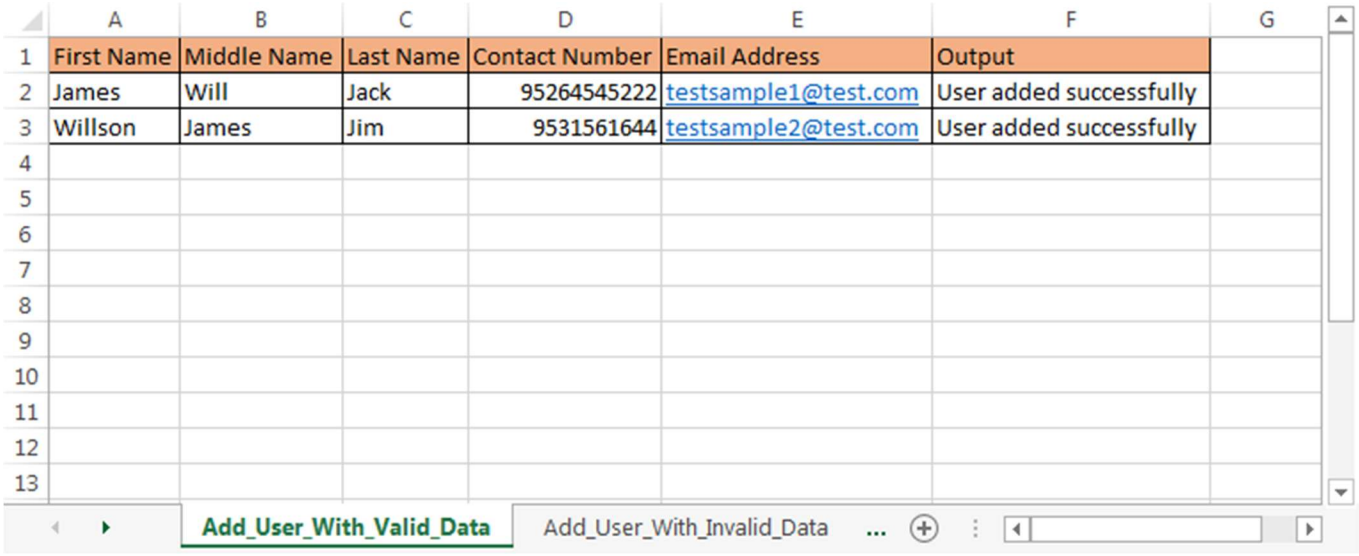
- MS Excel files
- Data base
- Text files
- XML files… etc.

External test data must be easily editable by test engineers without any programming skills. Excel files are the ones that used most often and are familiar with it.

Data sets increases test coverage by performing testing with various inputs and reduce the number of overall test scripts needed to implement all the test cases

Data Set / Test Data Examples

The below is the excel sheet for **"Add User with Valid Data"**

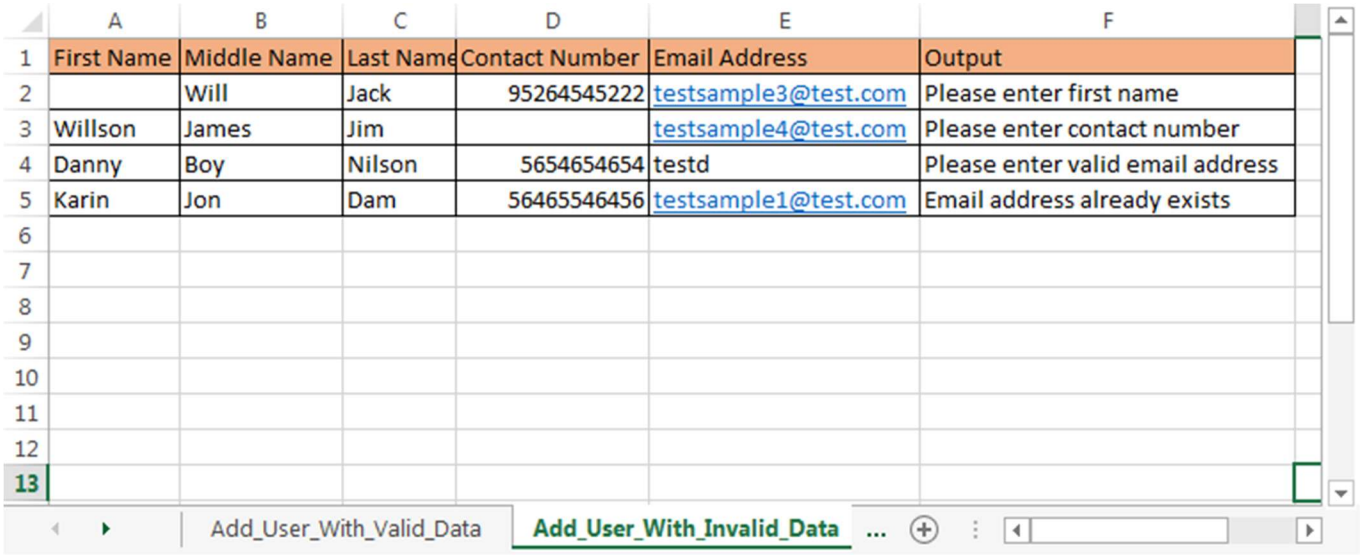


	A	B	C	D	E	F	G
1	First Name	Middle Name	Last Name	Contact Number	Email Address	Output	
2	James	Will	Jack	95264545222	testsample1@test.com	User added successfully	
3	Willson	James	Jim	9531561644	testsample2@test.com	User added successfully	
4							
5							
6							
7							
8							
9							
10							
11							
12							
13							

Add_User_With_Valid_Data | Add_User_With_Invalid_Data ...

In the above sheet, we are trying to add two records by filling all the fields. After entering the data in respective fields and clicking on submit button, it will display a success message which tool will get and compare with Actual and Expected.

The below is the excel sheet for **"Add User with invalid Data"**



	A	B	C	D	E	F
1	First Name	Middle Name	Last Name	Contact Number	Email Address	Output
2		Will	Jack	95264545222	testsample3@test.com	Please enter first name
3	Willson	James	Jim		testsample4@test.com	Please enter contact number
4	Danny	Boy	Nilson	5654654654	testd	Please enter valid email address
5	Karin	Jon	Dam	56465546456	testsample1@test.com	Email address already exists
6						
7						
8						
9						
10						
11						
12						
13						

Add_User_With_Valid_Data | Add_User_With_Invalid_Data ...

In the above example, we are trying to add four records with invalid details.

For the First row, we are passing an empty 'first name' which is a required field. When we click on submit button, it will display a message as 'Please enter first name'. We should now validate if the message displaying is as expected.

Why and How to use Data Set / Test Data:

We use test data to increase test coverage with valid and invalid data.

Say suppose, if there are 20 fields in the application. And we need to check different cases by passing different values.

If there are fields related to banking information, we need to validate and restrict the user not to enter junk or invalid data.

There are cases were you may need to check calculation / percentage by adding more number of records. In such cases we have to use data set and pass different values.

We need to make sure to identify the number of fields in the function / page and define the number of columns. On the basis of these columns, data will be provided to input fields in the application.

We should define a column called Output and provide the respective message based on the data provided by the user. To provide the result/status, tool should compare with expected and actual results. Here the output is nothing but the expected. Tool will get the actual and compare with the expected data provided by the user.

4. Test Automation Scripts

Consider the following automation script folder structure, page and test script location the pages are kept in com.app.Pages package and tests are kept in com.app.Test package. The test package is called in testng.xml and executed and result is generated.

```
src
        com.app.Pages
           Home_Page.java
           LogIn_Page.java

        com.app.Test
           FirstTestCase.java
           SecondTestCase.java
        lib
          drivers
             IEDreiverServer.exe
            Chromedriver.exe t
testng.xml
POM.xml
```

1. **Login_Page.java**

```
package com.ontestautomation.seleniumtestngpom.pages;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;

public class Login_Page {

   private WebDriver driver;
   public LoginPage(WebDriver driver) {

      this.driver = driver;
      if(!driver.getTitle().equals("ParaBank | Welcome | Online Banking")) {
         driver.get("http://parabank.parasoft.com");
      }
   }


   public void correctLogin(String username, String password) {
```

```
        driver.findElement(By.name("username")).sendKeys(username);
        driver.findElement(By.name("password")).sendKeys(password);
        driver.findElement(By.xpath("//input[@value='Log In']")).click();

    }
}
```

2. FirstTestCase.java

```
package com.ontestautomation.seleniumtestngpom.tests;

public class FirstTestCase {

    WebDriver driver;

    @BeforeSuite
    public void setUp() {

        driver = new FirefoxDriver();
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
    }

    @Parameters({"username","password"})
    @Test(description="Performs an unsuccessful login and checks the resulting error message")
    public void testLoginOK(String username, String password) {

        LoginPage lp = new LoginPage(driver);
        lp.correctLogin(username,password);
        Assert.asserTrue("The username and password are verified.");
    }

    @AfterSuite
    public void tearDown() {

        driver.quit();
    }
}
```

3. testng.xml

```
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >


<suite name="My first TestNG test suite" verbose="1" >
  <parameter name="username" value="john"/>
  <parameter name="password" value="demo"/>
  <test name="Login tests">
    <packages>
      <package name="com.ontestautomation.seleniumtestngpom.tests" />
```

```
  </packages>
 </test>
</suite>
```

5. Reports / Executed Results

Test report/results document which contains summary of test activities performed with pass/fail status and the time taken for execution.

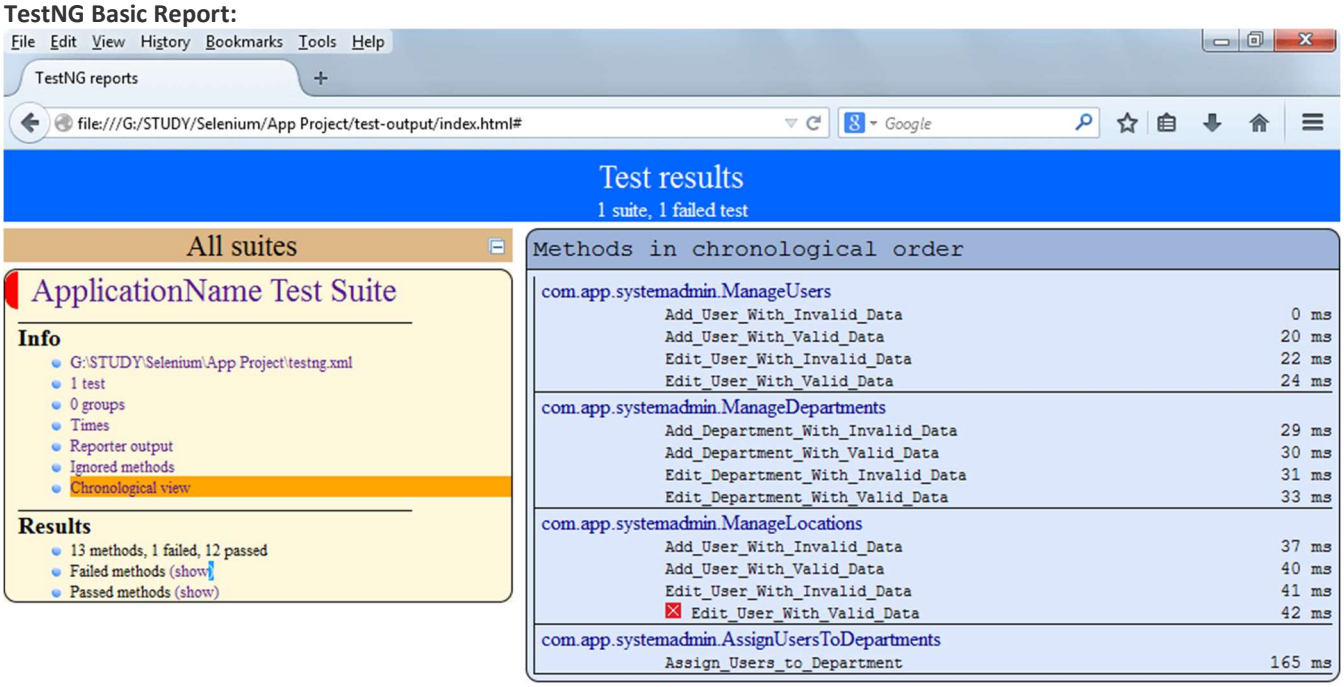
After completing the execution, it is very important to communicate the test results and findings to the project manager and with that decisions can be made for the release.

We can also capture and store all the screen shots for failed scenarios and it also store the executed data sheets with Pass/Fail status.

When we use TestNG, by default it generates the reports / execution status in 'html' formats.

The below is the sample html format report generated by TestNG.

TestNG Basic Report:



You can click here for article on How to Customize TestNG Report Emailable Report which will help user to quickly understand report.

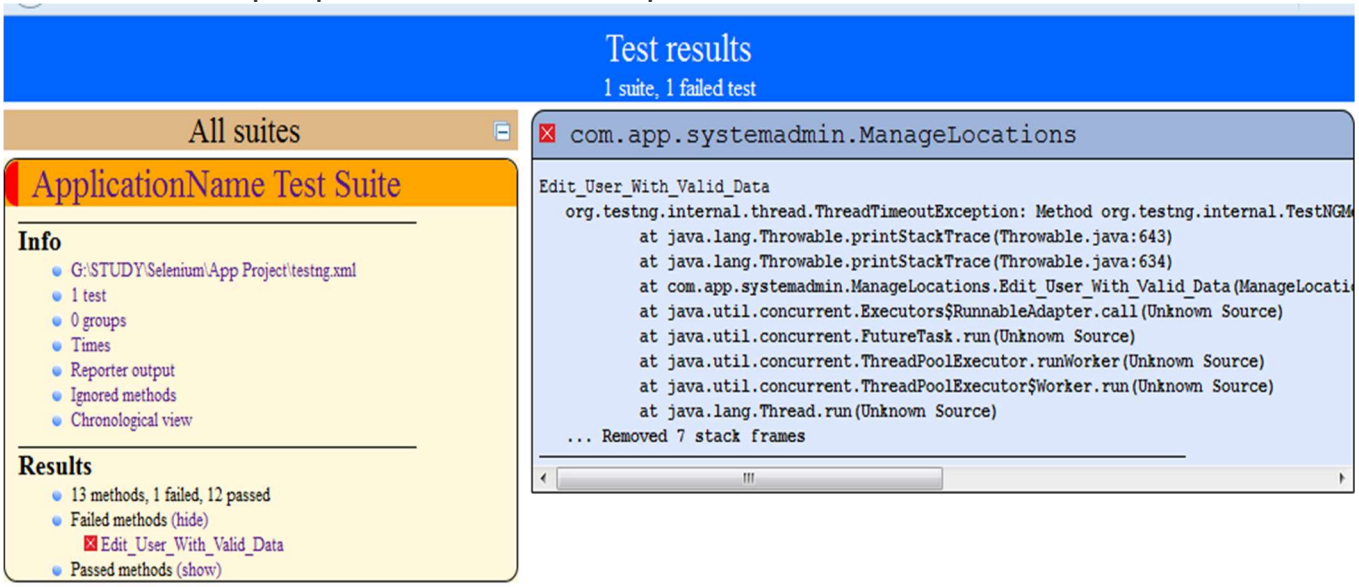
We can also view the reports in detail by using the options/links that are present in left-hand side.

The above report is in Chronological order, in which we can view all the executed methods and if there are any "Failed" cases, it will indicate with Red Color Cross icon (as shown at Edit_User_With_Valid_Data).

It will also display the time taken to execute test cases in the right side of each test cases.

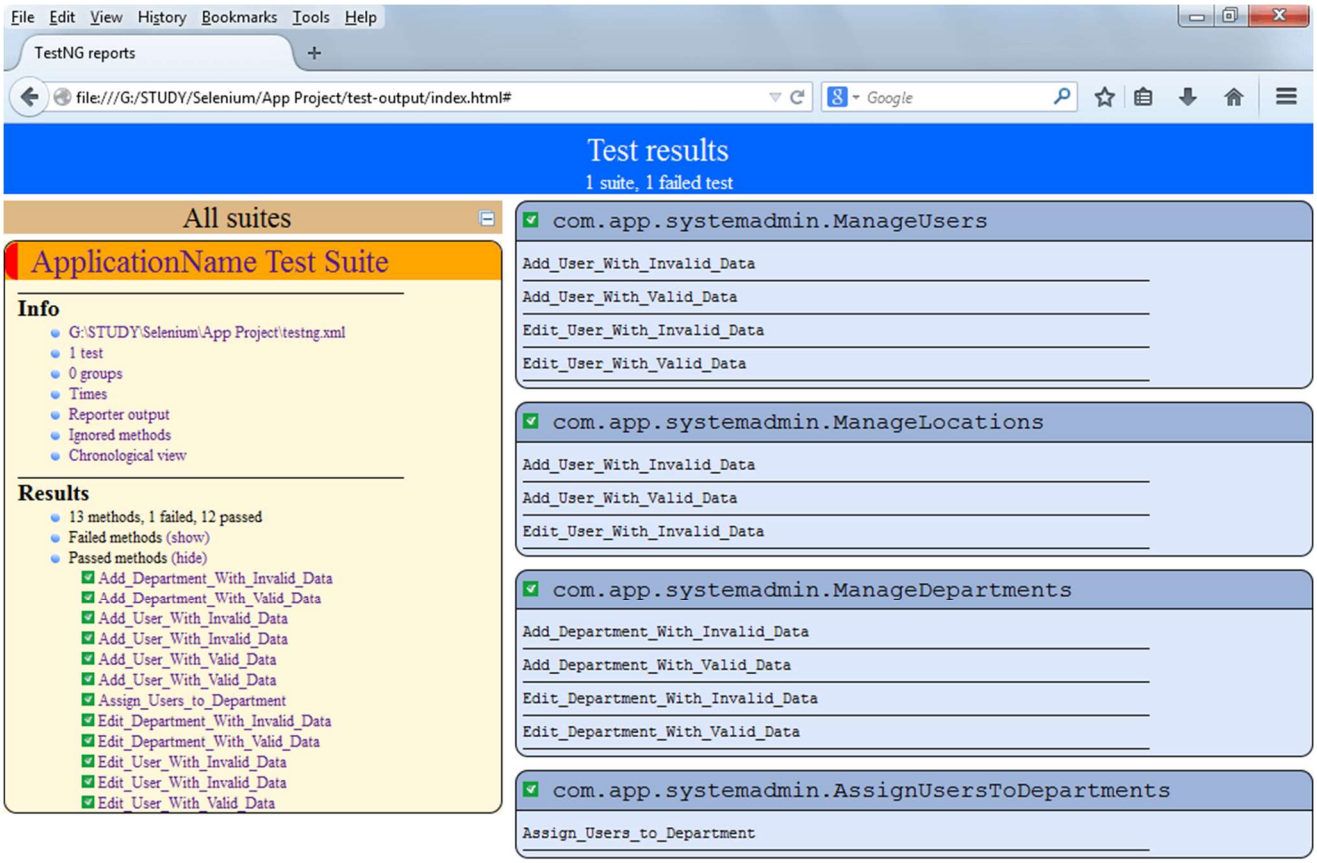
We can view the detailed report if there are any failed test cases, we can click on link “Show” beside Failed Methods under “Results” Grid.

The below is the example report to see failed test cases report in detail:



User can also see the Pass Methods details as Failed methods.

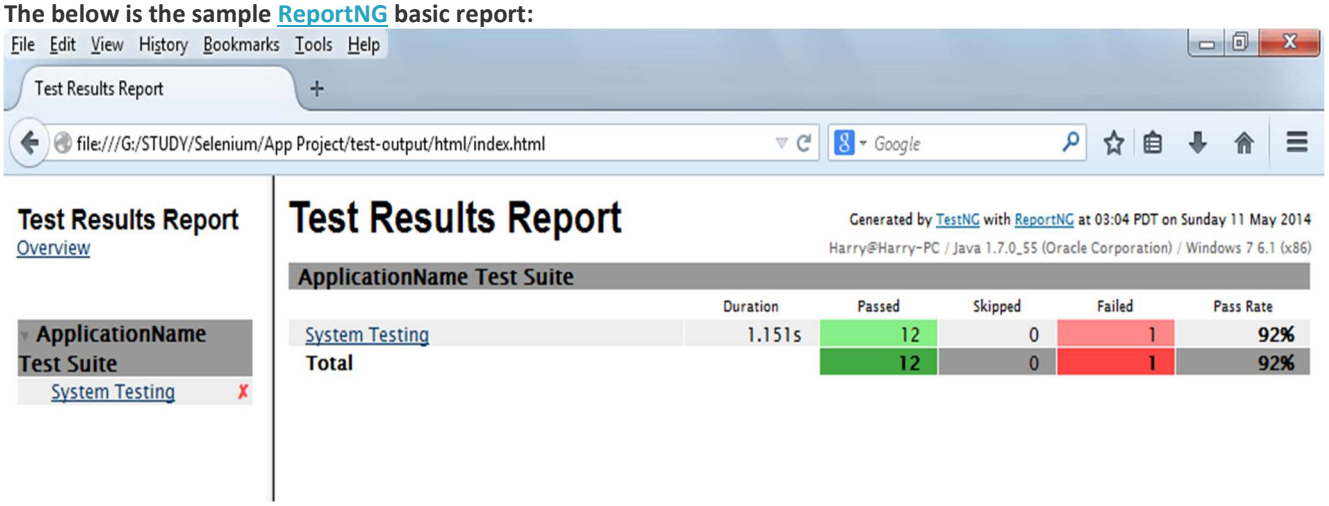
The below is the example report to see Passed test cases report in detail:



TestNG allows the user to use ReportNG Reports as well.

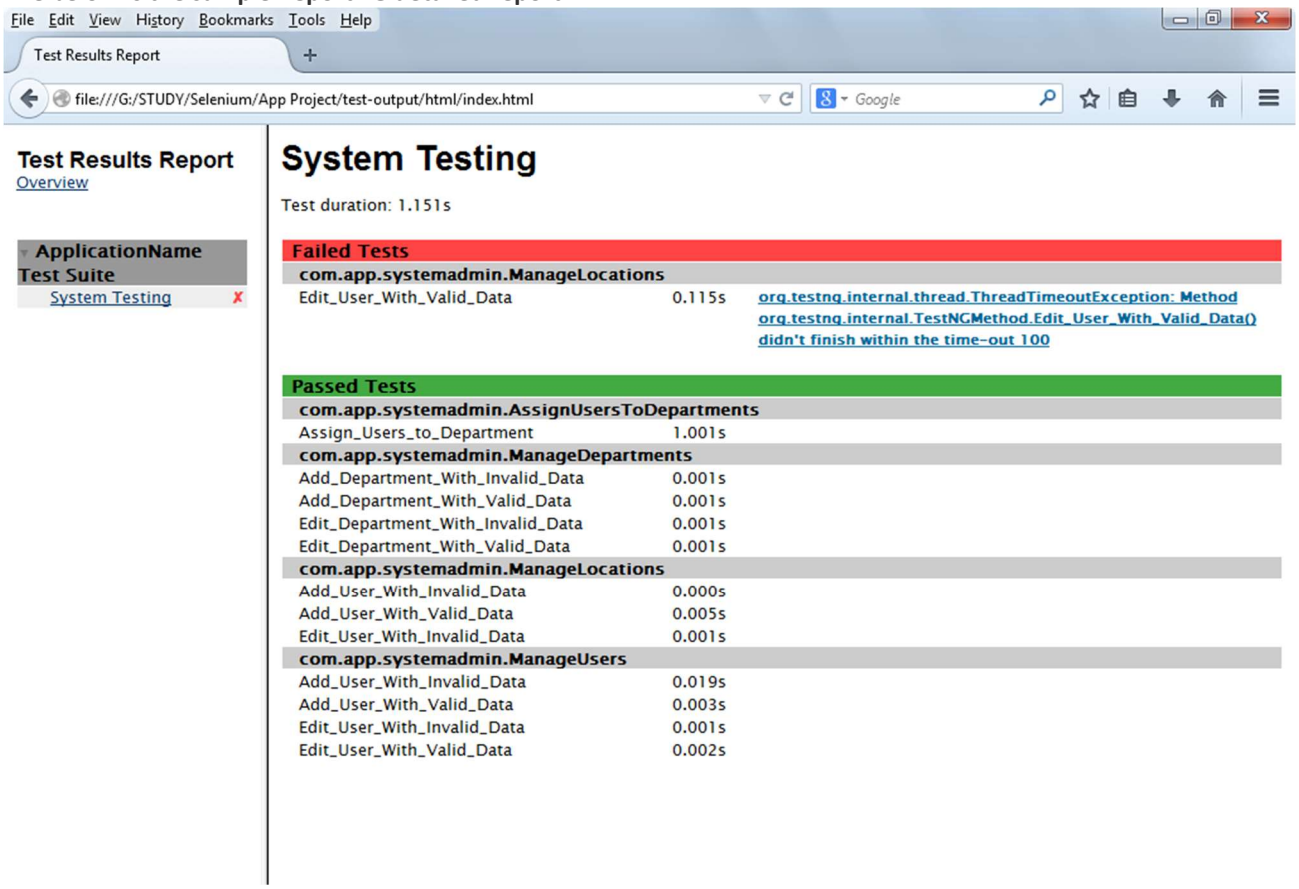
ReportNG is a simple plug-in for the TestNG unit-testing framework to generate HTML reports as a replacement for the default TestNG HTML reports.

The below is the sample ReportNG basic report:



Click on the hyperlink to view the detailed report. If there are any failed cases it will show the message/error that occurred.

The below is the sample ReportNG detailed report



6. TestNG XML example to execute Multiple Classes

In testng.xml file we can specify multiple name (s) which needs to be executed.
In a project there may be many classes, but we want to execute only the selected classes.
We can pass class names of multiple packages also. If say suppose, we want to execute two classes in one package and other class from some other package.
The below is the example testng.xml which will execute the class names that are specified.
Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<suite name="example suite 1" verbose="1" >
 <test name="Regression suite 1" >
  <classes>
   <class name="com.first.example.demoOne"/>
   <class name="com.first.example.demoTwo"/>
   <class name="com.second.example.demoThree"/>
  </classes>
 </test>
</suite>
```

We need to specify the class names along with packages in between the classes tags.
In the above xml, we have specified class name as “com.first.example.demoOne” and “com.first.example.demoOne” which are in “com.first.example” package. And class name demoThree is in package “com.second.example.”
All the classes specified in the xml will get executes which have TestNG annotations.
Below are the two example classes under “com.first.example” package which is executed.
Package: “com.first.example”
Classname: demoOne

```
package com.first.example;
import org.testng.annotations.Test;
public class demoOne {
		@Test
		public void firstTestCase()
		{
			System.out.println("im in first test case from demoOne Class");
		}

		@Test
		public void secondTestCase()
		{
			System.out.println("im in second test case from demoOne Class");
		}
}
```

Classname: demoTwo

```
package com.first.example;
import org.testng.annotations.Test;
public class demoTwo {
		@Test
		public void firstTestCase()
		{
			System.out.println("im in first test case from demoTwo Class");
		}

		@Test
```

```
	public void secondTestCase()
	{
		System.out.println("im in second test case from demoTwo Class");
	}
}
```

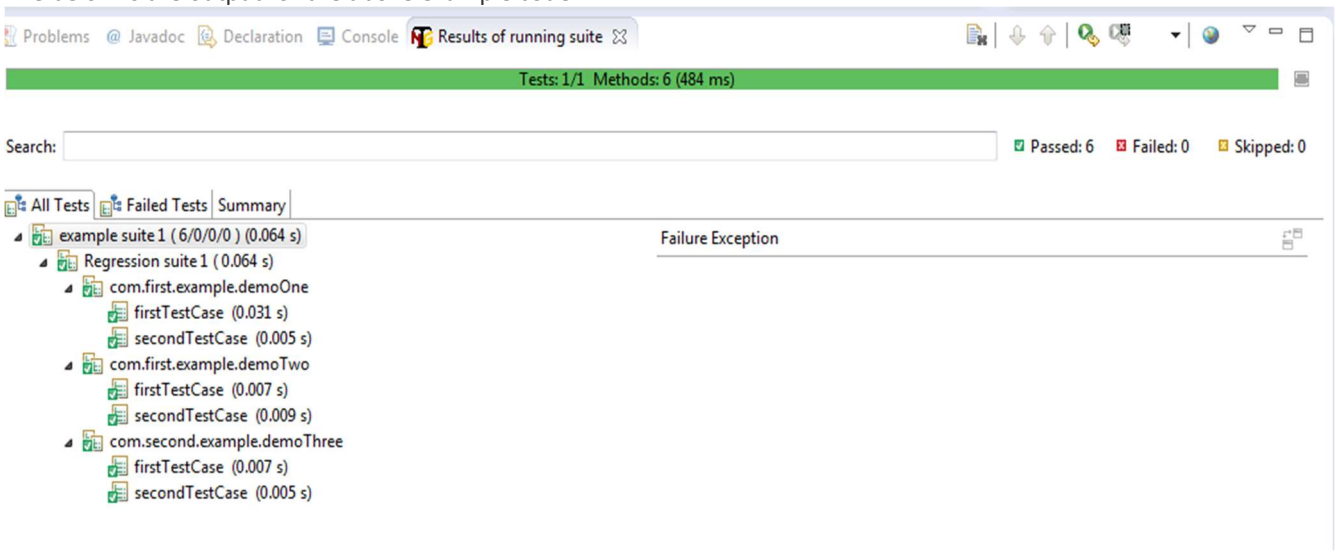
Package: "com.second.example"

Classname: demoThree

```
package com.second.example;
import org.testng.annotations.Test;
public class demoThree {
	@Test
	public void firstTestCase()
	{
		System.out.println("im in first test case from demoThree Class");
	}
	@Test
	public void secondTestCase()
	{
		System.out.println("im in second test case from demoThree Class");
	}
}
```

We need to run the testng.xml file. (Right click on testng.xml and select Run as 'TestNG Suite")

The below is the output for the above example code.



TestNG XML example to execute with package name:-

In testng.xml file we can specify the specific package name (s) which needs to be executed.

In a project there may be many packages, but we want to execute only the selected packages.

The below is the example testng.xml which will execute the specific packages.

```
<?xml version="1.0" encoding="UTF-8"?>
<suite name="example suite 1" verbose="1" >
```

```
 <test name="Regression suite 1" >
  <packages>
    <package name="com.first.example" />
  </packages>
 </test>
</suite>
```

We need to specify the names of the packages in between the package tags.

In the above xml, we have specified package name as “com.first.example” which will get executed. It will execute all the classes which have TestNG annotations

Below are the two example classes under “com.first.example” package which is executed .

Classname: demoOne

```
package com.first.example;
import org.testng.annotations.Test;
public class demoOne {
		@Test
		public void firstTestCase()
		{
			System.out.println("im in first test case from demoOne Class");
		}
		@Test
		public void secondTestCase()
		{
			System.out.println("im in second test case from demoOne Class");
		}
}
```

Classname: demoTwo

```
package com.first.example;
import org.testng.annotations.Test;
public class demoTwo {
		@Test
		public void firstTestCase()
		{
			System.out.println("im in first test case from demoTwo Class");
		}
		@Test
```

```
        public void secondTestCase()
        {
                System.out.println("im in second test case from demoTwo Class");
        }
}
```

We need to run the testng.xml file. (Right click on testng.xml and select Run as 'TestNG Suite")
The below is the output for the above example code.