Selenium – Day 4

November 2016

TEKSTROM Nurturing quality. Cultivating talent.

Learning Objectives– Day 4

- ❖DAY 3 - Recap
- ❖Database Interactions
- ❖Selenium Grid Overview and Benefits
- ❖Selenium Grid Implementation
- ❖Framework Design Patterns - Keyword Driven, Data Driven, Hybrid Model, Page Object Model
- ❖Exercise : Tests using Grids, Parallel Execution, Accessing Database

TEKSTROM Nurturing quality. Cultivating talent.

Database Interactions : JDBC

Java Database Connectivity (JDBC) is an application programming interface (API) for the programming language Java, which defines how a client may access a database. It is part of the Java Standard Edition platform, from Oracle Corporation.

The JDBC classes are contained in the Java package java.sql and javax.sql.

The API provides a mechanism for dynamically loading the correct Java packages and registering them with the JDBC Driver Manager.

The Driver Manager is used as a connection factory for creating JDBC connections.

JDBC connections support creating and executing statements. Query statements return a JDBC row result set. The row result set is used to walk over the result set. Individual columns in a row are retrieved either by name or by column number

JDBC drivers are client-side adapters (installed on the client machine, not on the server) that convert requests from Java programs to a protocol that the DBMS can understand.

Steps to read data from Database

Identify and download the JDBC driver for the corresponding database type.
(ex: mysql jdbc driver - https://jdbc.postgresql.org/download/postgresql-9.4.1212.jar)

Add the jdbc driver jar file to the build path.

Import java.sql classes to interact with database.

Make a connection to the database using DriverManager.getConnection.

Use Statement object to query database.

Use ResultSet object to retrieve recordsets returned by query and process results.

Close connection to the database.

Read data from Database - Example

```
package myPackage;

import java.sql.*;

public class Postgres_Test {

public static void main(String[] args) throws SQLException, ClassNotFoundException {

                    //Connection to postgres DB
                    String dbUrl = "jdbc:postgresql://localhost:5432/demo";
                    //Database Username
                    String username = ”postgres";
                    //Database Password
                    String password = "test";
                    //Query to fetch order details from orders table
                    String query = "select * from orders;";
                    //Load postgresjdbc driver
                    Class.forName("org.postgresql.Driver")
                    //Create Connection to DB
                    Connection con = DriverManager.getConnection(dbUrl,username,password);
                    //Create Statement Object
                    Statement stmt = con.createStatement();
                    // Execute the SQL Query. Store results in ResultSet
                    ResultSet rs= stmt.executeQuery(query);
                    // While Loop to iterate through all data and print results
                    while (rs.next()){
                    String orderid = rs.getString(1);
                    String itemid = rs.getString(2);
                    String itemqty = rs.getString(3);
                    String itemprice = rs.getString(4);
                    String total = rs.getString(5);
                    System.out.println(orderid+" , "+ itemid +" , " + itemqty + " , " + itemprice + " , " + total);
                }
                    // closing DB Connection
                    con.close();

                    }
}
```

TEKSTROM
Nurturing quality. Cultivating talent.

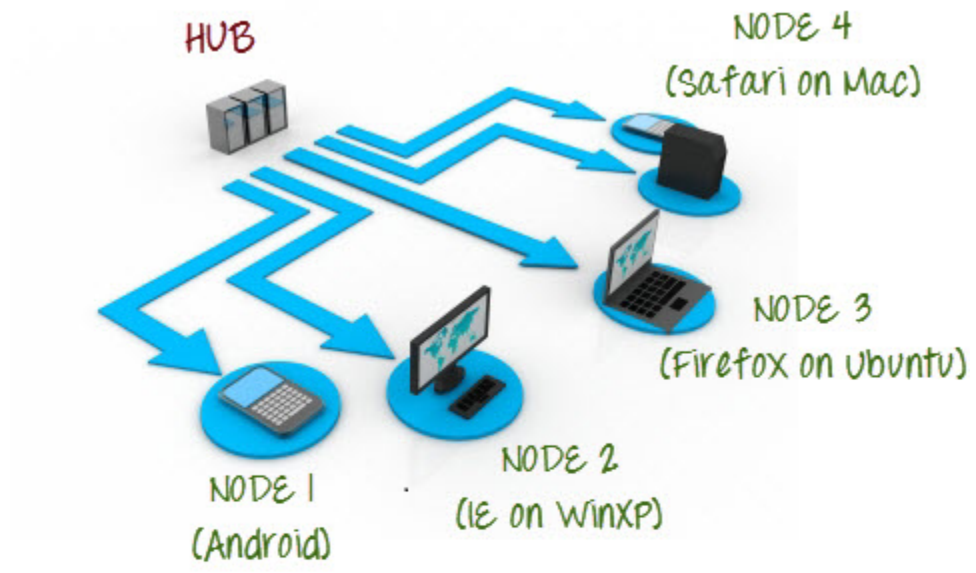
Selenium Grid

Selenium Grid allows to run tests in parallel

Different tests can be run at the same time on different remote machines

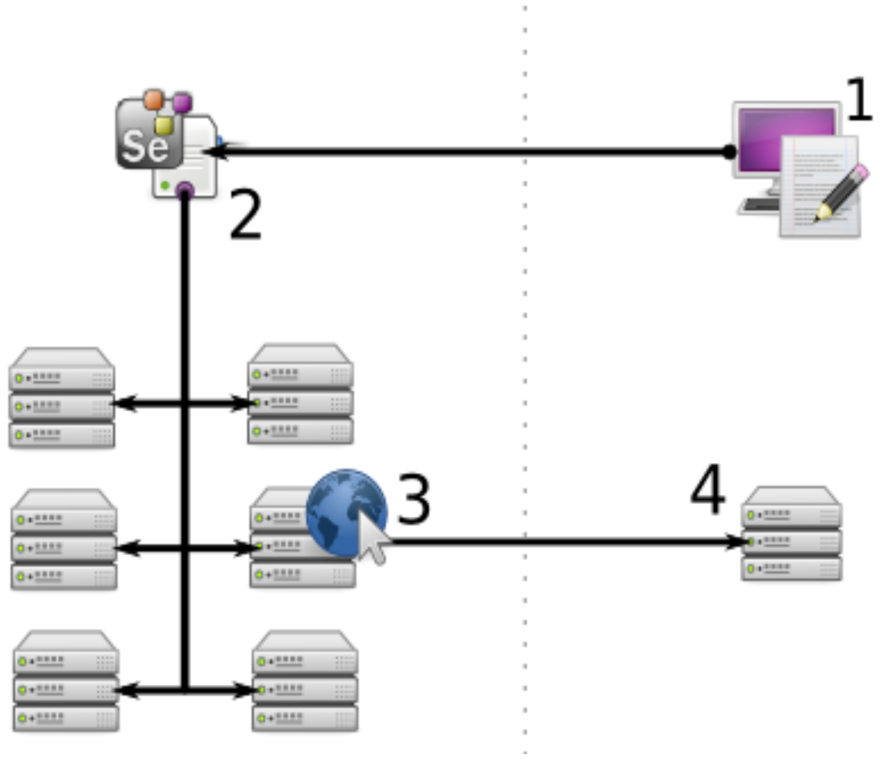
Advantages
Execution of large test suite or slow-running test suite
Multiple OS compatibility
Supporting Multiple Browsers

Selenium GRID



- Selenium Grid is a server that allows tests to use web browser instances running on remote machines
- With Selenium Grid, one server acts as the hub. Tests contact the hub to obtain access to browser instances. The hub has a list of servers that provide access to browser instances (WebDriver nodes), and lets tests use these instances
- Selenium Grid allows running tests in parallel on multiple machines, and to manage different browser versions and browser configurations centrally (instead of in each individual test)

Selenium GRID…



1. Test client sends commands to the Selenium Hub
2. Selenium Hub redistributes tests on the Grid environment
   - Each server can have a different environment
   - You target your tests to a specific environment
3. Selenium RC servers run on servers and drives browser instances
4. Tests run in parallel against the application server

TEKSTROM
Nurturing quality. Cultivating talent.

GRID Benefits

Distributed testing

Configure tests for different environments

Parallelize the tests

load distribution of testing across several machines

Minimize the maintenance time for the grid by allowing you to implement custom hooks to leverage virtual infrastructure for instance

TEKSTROM
Nurturing quality. Cultivating talent.

GRID – When to Use

To reduce the time it takes for the test suite to complete a test pass.

Selenium-Grid speeds up the execution

For example, a suite of 100 tests, Selenium-Grid to support 4 different machines (VMs or separate physical machines) to run those tests, test suite will complete in (roughly) one-fourth the time.

For large test suites, and long-running test suite such as those performing large amounts of data-validation, this can be a significant time-saver. Some test suites can take hours to run.

Faster feedback on test results after developers check-in code for the AUT.

Selenium-Grid is also used to support running tests against multiple runtime environments, specifically, against different browsers at the same time

Selenium-Grid With a Hub and Nodes

A grid consists of a single hub, and one or more nodes. Both are started using the selenium-server.jar executable. We’ve listed some examples in the following sections of this chapter

The hub receives a test to be executed along with information on which browser and ‘platform’ (i.e. WINDOWS, LINUX, etc) where the test should be run. It ‘knows’ the configuration of each node that has been ‘registered’ to the hub. Using this information it selects an available node that has the requested browser-platform combination. Once a node has been selected, Selenium commands initiated by the test are sent to the hub, which passes them to the node assigned to that test. The node runs the browser, and executes the Selenium commands within that browser against the application under test.

Installation

Download the Selenium-Server jar file from the SeleniumHq website's download page. You want the link under the section "Selenium-Server (formerly Selenium-RC)"

Install it in a folder of your choice. You'll need to be sure the java executable is on your execution path so you can run it from the command-line. If it does not run correcly, verify your system's path variable includes the path to the java.exe.

Selenium GRID – Starting HUB

Generally hub is first started since nodes depend on a hub.
This is not absolutely necessary however, since nodes can recognize when a hub has been started and vice-versa.

Starting a Hub

To start a hub with default parameters, run the following command from a command-line shell. This will work on all the supported platforms, Windows Linux, or Mac OSX.

```
Java –jar <selenium jar file> -role hub
```

This starts a hub using default parameter values. We'll explain these parameters in following subsections. Note that you will likely have to change the version number in the jar filename depending on which version of the selenium-server you're using

Selenium GRID – Starting Node

To start a node using default parameters, run the following command from a command-line.

Example:

```
java -jar selenium-server-standalone-2.53.1.jar -role node  -hub http://localhost:4444/grid/register
```

This assumes the hub has been started above using default parameters. The default port the hub uses to listen for new requests is port 4444. For getting started this is probably easiest. If running the hub and node on separate machines, simply replace 'localhost' with the hostname of the remote machine running the hub.

WARNING Be sure to turn off the firewalls on the machine running your hub and nodes. Connection errors might occur.

Selenium GRID – Hub Configuration

To get help on the commands

```
java –jar <selenium-standalone.x.jar> -h
```

To run the hub using the default options simply specify -role hub to the Selenium-Server with default port 4444

```
java –jar <selenium-standalone.x.jar> -hub
```

For different port

```
java –jar <selenium-standalone.x.jar> -hub –port 4441
```

Nurturing quality. Cultivating talent.

Selenium GRID – Node Configuration

By default, starting the node allows for concurrent use of 11 browsers : 5 Firefox, 5 Chrome, 1 Internet Explorer.

```
-browser browserName=firefox,version=3.6,maxInstances=5,platform=LINUX
```

This setting starts 5 Firefox 3.6 nodes on a linux machine.

```
-browser “browserName=firefox,version=3.6,firefox_binary=c:\Program Files\firefox,
maxInstances=3, platform=WINDOWS”
```

This setting starts 5 Firefox 3.6 nodes on a windows machine

Selenium GRID – Node Configuration…

Optional Parameters:

-port 4444 (4444 is default)
-host <IP | hostname> - The host name or IP..
-timeout 30 (300 is default) - The timeout in seconds.

Note: This is NOT the WebDriver timeout for all "wait for WebElement" type of commands.

-maxSession 5 (5 is default) - The maximum number of browsers that can run in parallel on the node
-browser < params > If -browser is not set, a node will start with 5 firefox.

Ex: -browser: browserName={android, chrome, firefox, htmlunit, internet explorer, iphone, opera} version={browser version} firefox_binary={path to executable binary} chrome_binary={path to executable binary} maxInstances={maximum number of browsers of this type} platform={WINDOWS, LINUX, MAC}

TEKSTROM
Nurturing quality. Cultivating talent.

Selenium GRID – Node Configuration…

Optional Parameters:

-registerCycle N - how often in ms.

Configuring timeouts (Version 2.21 required)

Timeouts in the grid should normally be handled through webDriver.manage().timeouts(), which will control how the different operations time out.

To preserve run-time integrity of a grid with selenium-servers, there are two other timeout values that can be set.

On the hub

- Setting the -timeout command line option to "30" seconds will ensure all resources are reclaimed 30 seconds after a client crashes.
- also set -browserTimeout 60 to make the maximum time a node is willing to hang inside the browser 60 seconds.

Note: The browserTimeout should be: Higher than the socket lock timeout (45 seconds)

Framework Design Patterns

Data driven framework

Keyword driven framework

Hybrid framework

POM (Page object model)

Data Driven Framework

One test class which covers multiple application workflows/scenarios.

Test data is maintained separately in an XLS, CSV or XML file format.

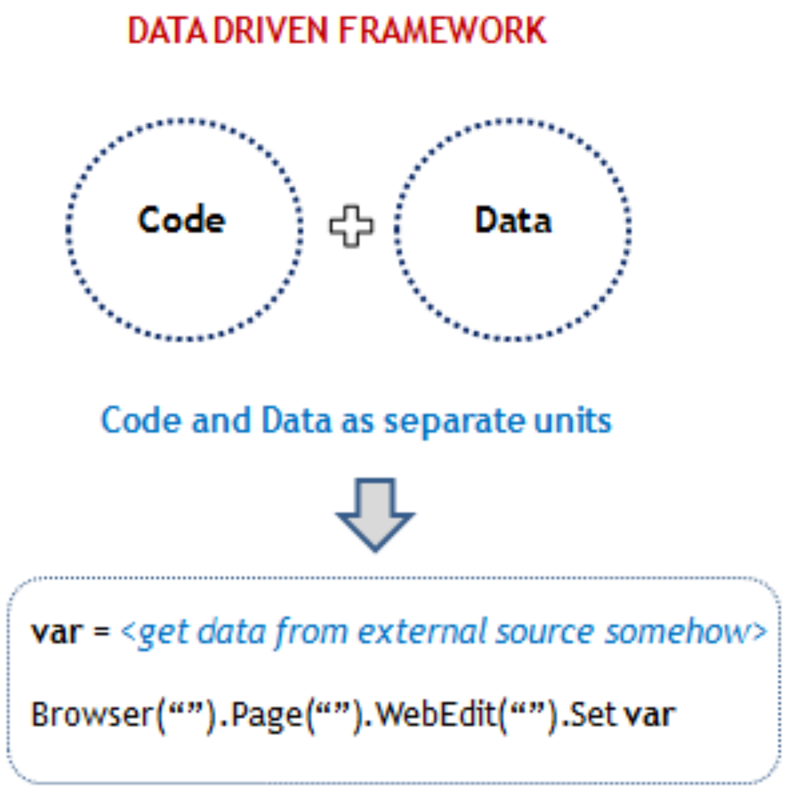
Data drives (or decides) which scenario or application flow to be executed.

Each row in test data file represents one data combination.

Test class read each row of the data and determines the flow of script.

Less changes to code as new data variations are maintained in test data file.

Suitable for the applications with fixed workflows and varied data conditions.



```
var = <get data from external source somehow>
Browser("").Page("").WebEdit("").Set var
```

TEKSTROM
Nurturing quality. Cultivating talent.

Keyword Driven Framework

Keyword driven framework separates test case definition from code.

Makes it easy for non-technical teams in building and maintaining automation suite.

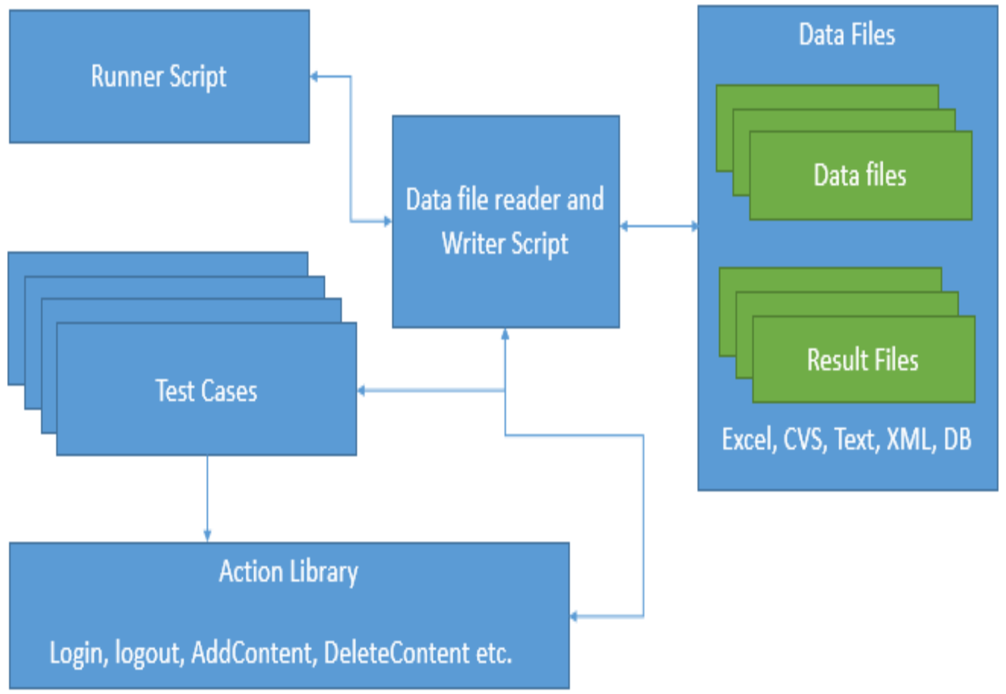
Test steps (Object, Action, Value) are provided in a separate file (xls file).

Easy to maintain, as any changes to the object definitions/actions/values does't need code changes.

One excel sheet represents one test case with each row representing the test step details.

A driver excel sheet maintains the list of test cases with a flag to determine if the test needs to be executed or not.

Optionally test cases can be grouped into modules and module level flags can be created.


Runner Script
Data Files
Data files
Data file reader and Writer Script
Result Files
Test Cases
Excel, CVS, Text, XML, DB
Action Library
Login, logout, AddContent, DeleteContent etc.

Hybrid Framework

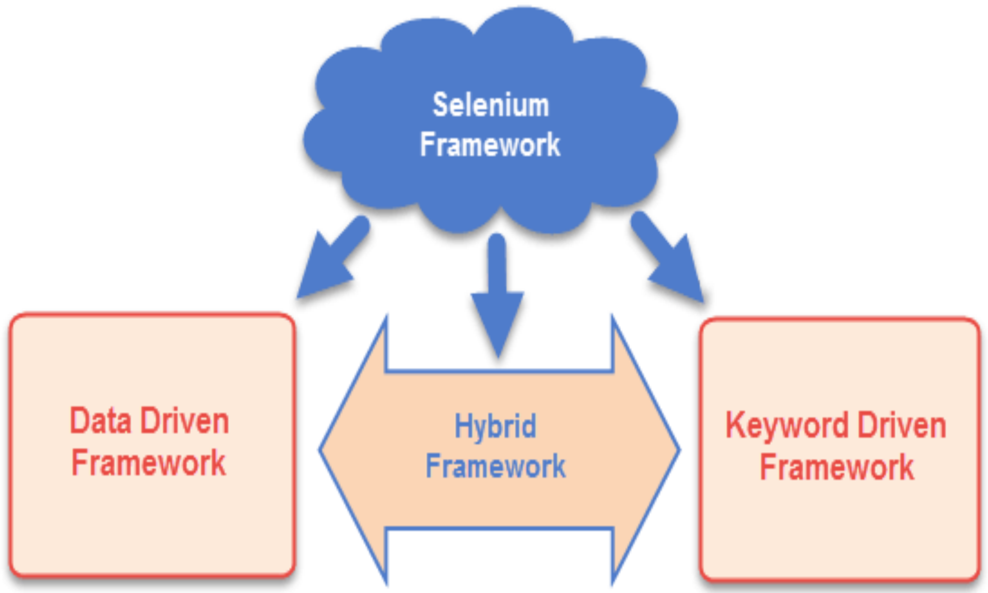
Hybrid framework is combination Keyword, Data Driven framework

Combines benefits of both keyword and data driven frameworks.

Test data, test steps and test suites are all maintained in separate excel files.

Any changes to application flow/objects or test data are isolated to changing the values in the corresponding excel sheets.

Easy to create/maintain for non-technical teams.



Page Object Model (POM)

Page Object is a Design Pattern which has become popular in test automation for enhancing test maintenance and reducing code duplication.

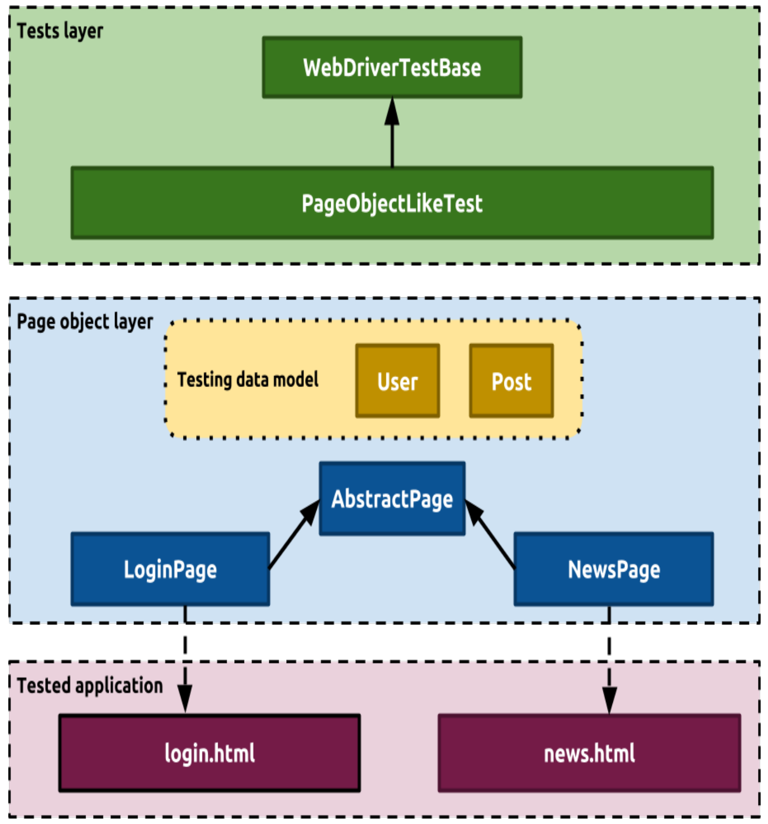
Models the pages/screen of the application as objects called Page Objects.

All the functions that can be performed in the specific page are encapsulated in the page object of that screen.

Increases code reusability - code to work with events of a page is written only once and used in different test cases

Improves code maintainability - any UI change leads to updating the code in page object classes only leaving the test classes unaffected.

Makes code more readable and less brittle.



TEKSTROM Nurturing quality. Cultivating talent.

Exercise

- Launching Tests using Grids
- Parallel Execution of Tests
- Accessing Database

Day 4 - Summary

Today we learnt :

- ❖Database Interactions
- ❖Selenium Grid Overview and Benefits
- ❖Selenium Grid Implementation
- ❖Framework Design Patterns
  - ➢Keyword Driven
  - ➢Data Driven
  - ➢Hybrid Model
  - ➢Page Object Model