GEB-Page Object Modelling with Power Of Selenium WebDriver and Groovy’ s expressiveness

Agenda of this Document: Introduction of GEB as a Solution of Browser Automation.

What is GEB: GEB is browser automation solution, which is a wrapper over Selenium WebDriver. It uses:

1. Groovy's expressiveness to deal with the Browser Automation.
2. JQuery to deal with the WebElements.
3. Robust Page Object Modelling Framework.

It can be Integrated with different Unit Test Framework such as **JUnit**, **TestNg**, **Spock** as well as different build tools like **Maven**, **Gradle** and **CI tool like Jenkins**.

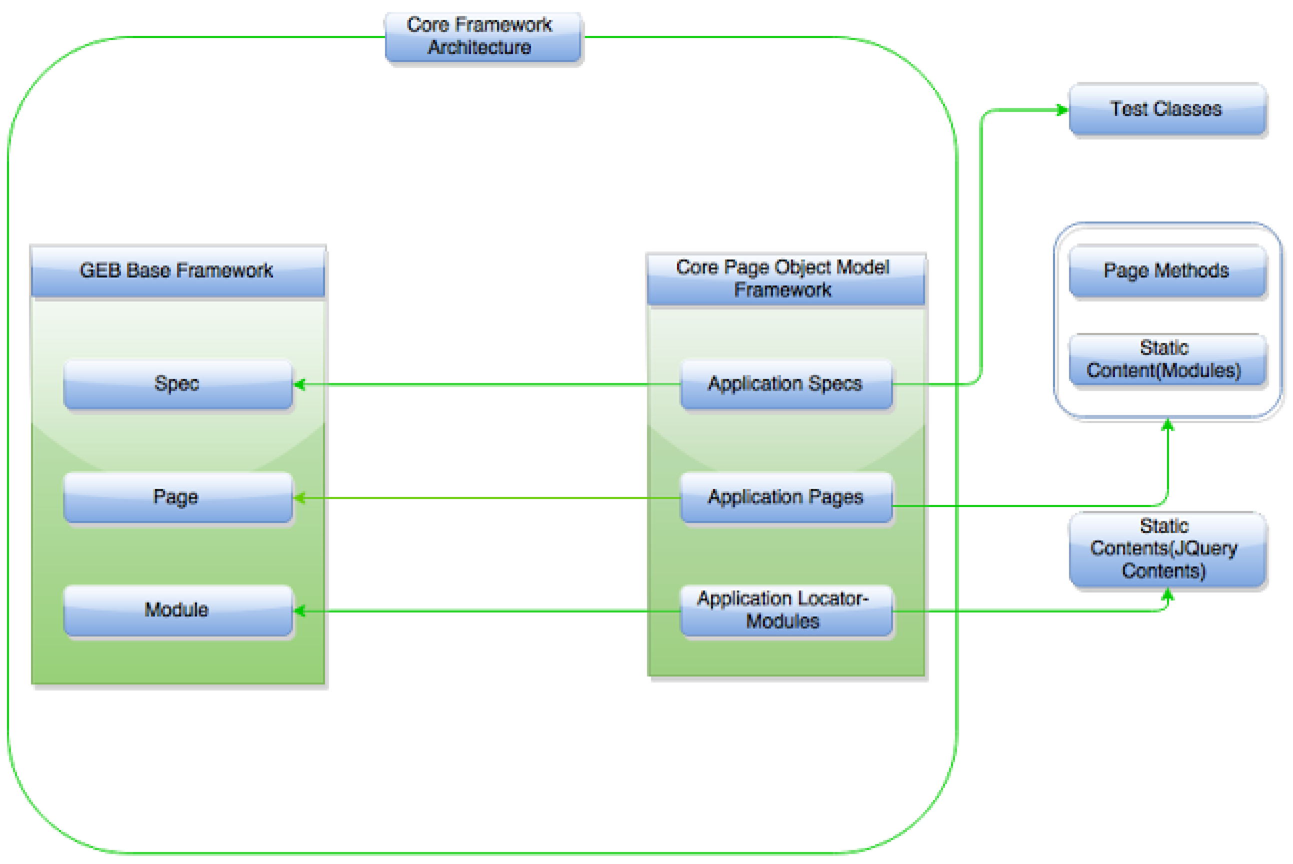
Is it Really Useful when we have Selenium **for Automation :**
The answer to the above is **YES,** because it comes with few very good features along with it.

1. GEB uses Groovy language which itself is very powerful having features like **Closure**, **GroovyMarkup** and **GPath** support,dynamic and static typing is supported and many more.
2.As it is supported by Groovy, so **less coding is required to write automation scripts compared to JAVA.**
3. GEB supports JQuery to handle WebElements,so Scripts become more **faster**.
4.Asynchronous Content Lookup.
5.**Selenium API** can be used easily if need arises.

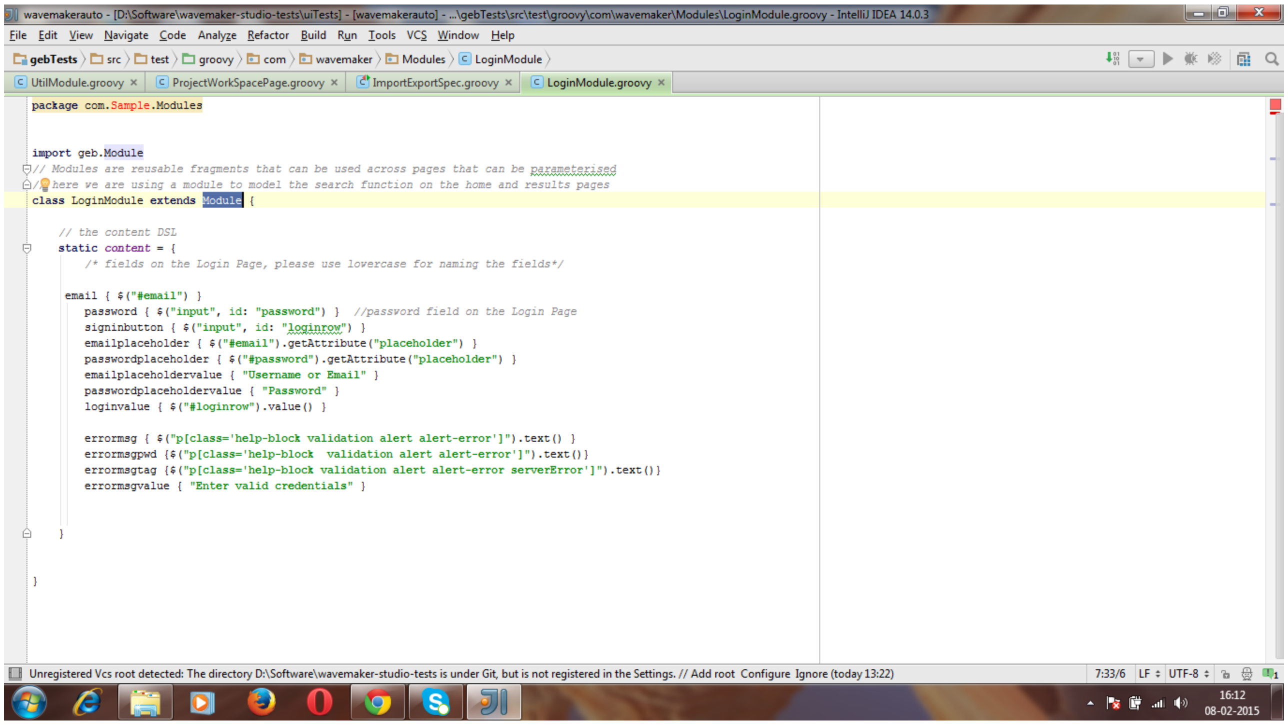
Structure of Framework:

If we are using GEB then generally we follow 3 main building blocks of it:
1.**Module**:Where we keep our WebElements/Locators/JQuery Contents as static contents.
2.**Pages**:Place for the re-usable methods of Page/Module functionalities .
3.**Specs**: Where the Test Classes are kept.

Please refer the below screenshot for the Core framework structure.


Core Framework Architecture
Test Classes
GEB Base Framework
Core Page Object Model Framework
Page Methods
Static Content(Modules)
Spec
Application Specs
Page
Application Pages
Static Contents(JQuery Contents)
Module
Application Locator-Modules

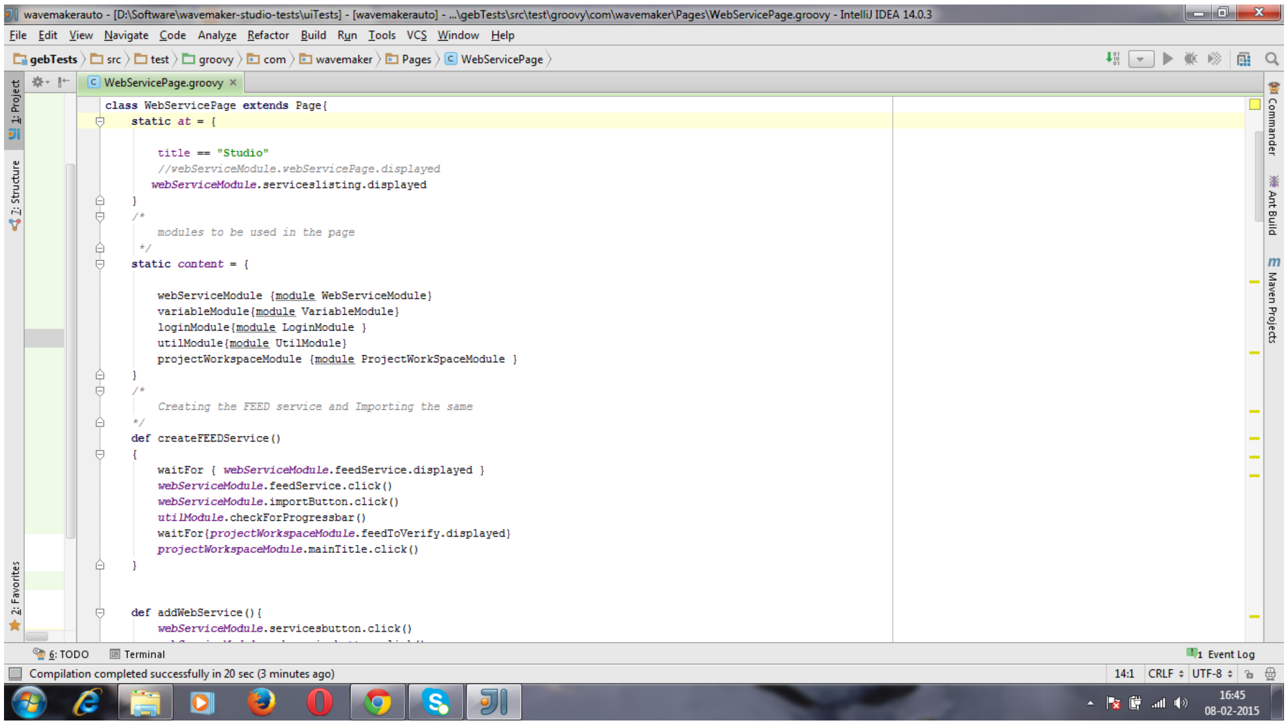
Sample of Module given below:



```
package com.Sample.Modules


import geb.Module
// Modules are reusable fragments that can be used across pages that can be parameterised
//here we are using a module to model the search function on the home and results pages
class LoginModule extends Module {

    // the content DSL
    static content = {
        /* fields on the Login Page, please use lowercase for naming the fields*/

     email { $("#email") }
        password { $("input", id: "password") }  //password field on the Login Page
        signinbutton { $("input", id: "loginrow") }
        emailplaceholder { $("#email").getAttribute("placeholder") }
        passwordplaceholder { $("#password").getAttribute("placeholder") }
        emailplaceholdervalue { "Username or Email" }
        passwordplaceholdervalue { "Password" }
        loginvalue { $("#loginrow").value() }

        errormsg { $("p[class='help-block validation alert alert-error']").text() }
        errormsgpwd {$("p[class='help-block  validation alert alert-error']").text()}
        errormsgtag {$("p[class='help-block validation alert alert-error serverError']").text()}
        errormsgvalue { "Enter valid credentials" }


    }


}
```

Unregistered Vcs root detected: The directory D:\Software\wavemaker-studio-tests is under Git, but is not registered in the Settings. // Add root Configure Ignore (today 13:22)

Sample for Page :

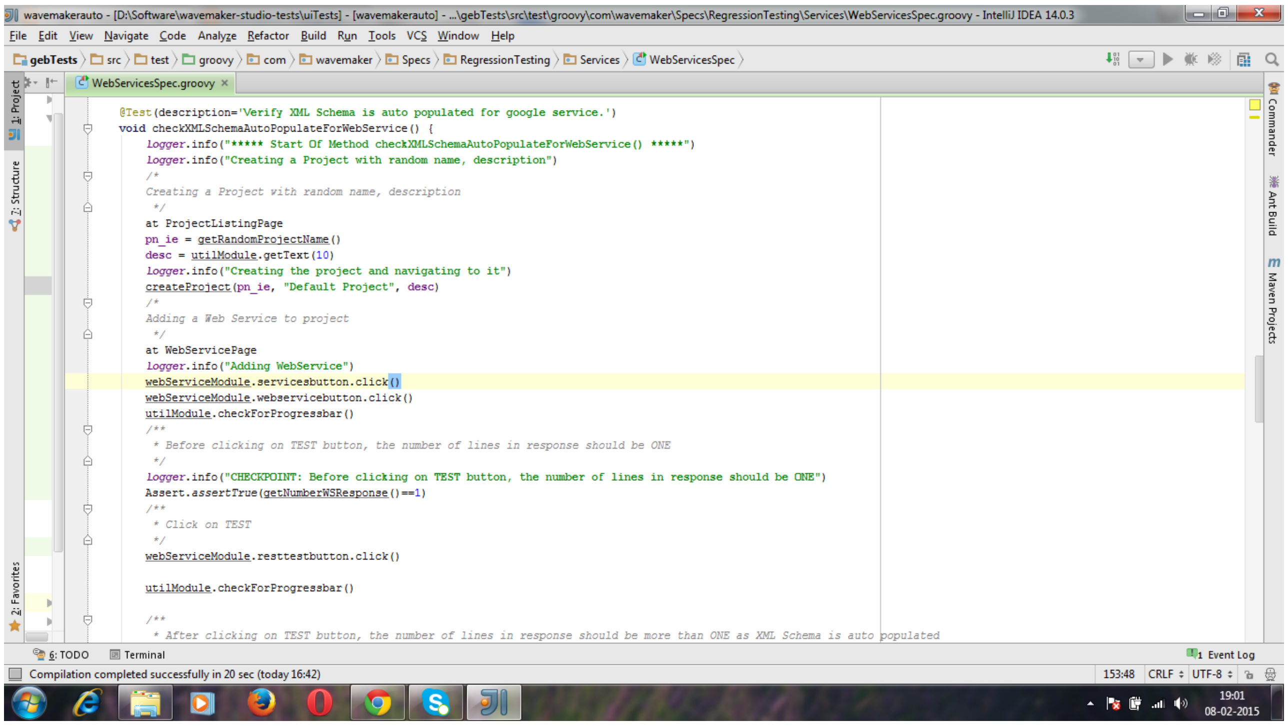


```
class WebServicePage extends Page{
    static at = {

        title == "Studio"
        //webServiceModule.webServicePage.displayed
      webServiceModule.serviceslisting.displayed
    }
    /*
        modules to be used in the page
     */
    static content = {

        webServiceModule {module WebServiceModule}
        variableModule{module VariableModule}
        loginModule{module LoginModule }
        utilModule{module UtilModule}
        projectWorkspaceModule {module ProjectWorkSpaceModule }
    }
    /*
        Creating the FEED service and Importing the same
    */
    def createFEEDService()
    {
        waitFor { webServiceModule.feedService.displayed }
        webServiceModule.feedService.click()
        webServiceModule.importButton.click()
        utilModule.checkForProgressbar()
        waitFor{projectWorkspaceModule.feedToVerify.displayed}
        projectWorkspaceModule.mainTitle.click()
    }


    def addWebService(){
        webServiceModule.servicesbutton.click()
```

Compilation completed successfully in 20 sec (3 minutes ago)

GEB provides few methods like "at" which is used to verify whether we are on the stated page or not and "to" method to redirect the driver to the stated page.
If we are in certain page then to access the reusable methods of that page class i.e Page Functionalities we Don't need to create Object for that class,we can simple call the method.

Sample of a Test method :



wavemakerauto - [D:\Software\wavemaker-studio-tests\uiTests] - [wavemakerauto] - ...\gebTests\src\test\groovy\com\wavemaker\Specs\RegressionTesting\Services\WebServicesSpec.groovy - IntelliJ IDEA 14.0.3

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
gebTests › src › test › groovy › com › wavemaker › Specs › RegressionTesting › Services › WebServicesSpec
WebServicesSpec.groovy
1: Project
2: Structure
2: Favorites
Commander
Ant Build
Maven Projects
6: TODO
Terminal
Event Log

```
    @Test(description='Verify XML Schema is auto populated for google service.')
    void checkXMLSchemaAutoPopulateForWebService() {
        logger.info("***** Start Of Method checkXMLSchemaAutoPopulateForWebService() *****")
        logger.info("Creating a Project with random name, description")
        /*
        Creating a Project with random name, description
         */
        at ProjectListingPage
        pn_ie = getRandomProjectName()
        desc = utilModule.getText(10)
        logger.info("Creating the project and navigating to it")
        createProject(pn_ie, "Default Project", desc)
        /*
        Adding a Web Service to project
         */
        at WebServicePage
        logger.info("Adding WebService")
        webServiceModule.servicesbutton.click()
        webServiceModule.webservicebutton.click()
        utilModule.checkForProgressbar()
        /**
         * Before clicking on TEST button, the number of lines in response should be ONE
         */
        logger.info("CHECKPOINT: Before clicking on TEST button, the number of lines in response should be ONE")
        Assert.assertTrue(getNumberWSResponse()==1)
        /**
         * Click on TEST
         */
        webServiceModule.resttestbutton.click()

        utilModule.checkForProgressbar()

        /**
         * After clicking on TEST button, the number of lines in response should be more than ONE as XML Schema is auto populated
```

Compilation completed successfully in 20 sec (today 16:42)

153:48 CRLF UTF-8

19:01
08-02-2015

Geb can be very effective and robust as a solution to the browser Automation.