BDD - Behavior Driven Development

BDD is an extension of TDD (Test Driven Development). The idea of TDD is that you start writing a unit test that fails, then you code until it succeed and repeat that cycle until the work is done.

BDD propose to apply this same workflow on another level: you start writing the acceptance tests with your clients, then you develop the features that will make those tests succeed.

BDD defines a Domain Specific Language based on Natural Language to write these acceptance tests, in a way that non-technical collaborators can easily understand them.

The development team will then code the features and the necessary “glue” that will translate the acceptance test in executable code.

Cucumber is a framework/tool that helps creating tests in this way. We can use Cucumber without BDD, as a tool to make functional tests very well readable.

LFRGS Selenium Commons Setup

How start running the tests

1. Create file **defaultProperties.properties** in path **/opt/liferay/ <defaultProperties.properties *here*>**

2. Download the browser drivers
   a. Gecko driver (if you want running Firefox)

   b. Update firefox just for sure

   c. Unzip gecko driver and put it in the path */opt/liferay/web_drivers/*

   2.1 If you don't want to run the tests with Firefox, just change the variables in **defaultProperties.properties**:

a. It can be defaultGC (chrome), defaultFF (firefox) -> **browser=defaultFF**

References

1. Wiki of LFRGS Selenium Commons: Wiki link

2. It was used a template project made by GS-BR. Template Project link.

Code Structure

Cucumber Structure

In your code you will have two kinds of files, the first is the ".feature" file and the second is the ".java" files. This files have a different functions on the code.
The ".feature" will received the scenarios related with the feature that team are developing, example of one ".feature" below:

```
#language:en
Feature: Login

@Sanity
Scenario Outline: Perform the Login with success
Given I am on Home Page
When I fill the <login> and <password> I will be logged
Then The <usernameAcronym> is displayed

Examples:
| login | password | usernameAcronym |
| test@liferay.com | test | TT |

Scenario Outline: Perform the Login with fail
Given I am on Home Page
When I fill the <login> and <password> I will be logged
Then The error message will appear related with <wrongField>

Examples:
| login | password | wrongField |
| unexistUser@liferay.com | unexistPass | both |
| | noUserPass | user |
| unexistUser@liferay.com | | password |
```

The ".java" files will have the annotations of the file above to make the translation of the "cucumber language" to "selenium" or "java language". Example of one ".java" below:

```
public class LoginStepDefinitions {

LoginPage loginPage = new LoginPage();
WelcomePage welcomePage = new WelcomePage();

@Given("^I am on Home Page$")
public void i_am_on_home_page() {
UtilsKeys.DRIVER.get(UtilsKeys.getUrlToHome());
}

@When("^I fill the (-?[^\"]*) and (-?[^\"]*) I will be logged$")
public void i_fill_the_login_and_password_i_will_be_logged(String emailAddress, String
password) {
loginPage.clickOnSignIn();
loginPage.fillEmailAddressField(emailAddress);
loginPage.fillPasswordField(password);
loginPage.clickOnSignInOfTheModal();
}

@Then("^The (-?[^\"]*) is displayed$")
public void the_username_is_displayed(String username) {
assertEquals(true, welcomePage.usernameIsDisplayed(username));
}

@Then("^The error message will appear related with (-?[^\"]*)$")
public void the_error_message_will_appear_related_with(String wrongField) {
switch (wrongField) {
case "both":
assertEquals(true, loginPage.alertErrorIsDisplayed());
break;
case "user":
assertEquals(true, loginPage.loginHelperIsDisplayed());
break;
case "password":
assertEquals(true, loginPage.passwordHelperIsDisplayed());
break;
default:
//This switch will catch the failure if the others cases weren't mapped
assertEquals(true, false);
}
}
```

Using this approach any member of the team can contribute with the quality of the code, because anyone can write the ".feature" file, and other technical member of the team can create the annotations related with the action of the respective annotation.

References

1. It was used a template project made by GS-BR. Template Project link.

2. Know more on: https://cucumber.io

Page Object Structure

This approach is one easy technique to keep your code more understandable and easier to make the maintenance. You should create two initial structures.

- The **src/test/java/** must have four folders/packages:

- **com.liferay.<PROJECT>.test.functional.pages:** This path contains all classes mapped between functions per pages, for example, the Login Page, should have all elements and methods that the test will need to perform the actions on the respective page, the same logic is applicable on Welcome Page.

- **com.liferay.<PROJECT>.test.functional.steps:** This path have all the steps definitions, in other words, is the path who have the annotations mentioned on topic above.

- **com.liferay.<PROJECT>.test.functional.utils:** This path have all the things that can be util to the code, for example the class "CommonMethods".

- **com.liferay.<PROJECT>.test.functional:** This path have only the "run" class.

- The **src/test/resources/** must have only the ".feature" files.

In this kind of organization will be easier to developer or tester, developing the code and looking for the future, realize the maintenance.

References

1. It was used a template project made by GS-BR. Template Project link.

Common Methods

LFRGS Selenium Commons

About the shared classes on "SeleniumCommonMethods" section, you can know this information, on this link: Existing Classes on lfrgs-selenium-commons

Why would you use ID attributes

1. Clean automation code. Your automation code will be more readable if you get the attribute's locator by ID instead of getting it by XPath or CSS selectors.

2. In some cases, it's inevitable to find an element by ID, but if you create a XPath with the ID element, this XPath will be more robust.

3. Fastest way to locate elements on page because selenium gets it down executing document.getElementById().

4. Fragility in UI tests is hard to manage, so other issues might also need to be resolved before you see the benefits of adding IDs. This point can achieve everybody's confidence in the team about the automated tests.

5. Best discussions and articles about this subject can be found here:
    - Good practices for thinking in ID and Class name by Google.
        - Google's HTML code has ID's with some_id and also with some-id. I believe that the two one is correct. It must be chosen one way to follow and make the code readable.
    - Is adding ids to everything standard practice when using selenium
    - Which is the best and fastest way to find the element using webdriver by xpath

Auxiliary tool during test automation process

1. Extension to help to generate step definitions classes. If the IDE doesn't support generating it. But if you prefer, install cucumber plugin to your IDE and it will generate from there too.
    a. Tidy Gherkin

2. Firebug with FirePath - a great help when you need to find XPath expressions, he creates one for you by inspecting the element, work only on Firefox.

Selenium GRID

Selenium GRID in a remote machine

1. Configure one machine that your selenium will be running
2. Install the browser latest version that the selenium GRID will be running
3. Download of the driver that the browser installed on step above.
4. Install the java JDK, I recommend the java JDK latest version.
5. Download the selenium server standalone jar file
6. Create one folder on the selenium GRID machine:
    a. Move the browser driver to this folder.
    b. Move the selenium server standalone jar file to this folder.
    c. Create a JSON files:
        i. **Hub.json:** This file must contains the configuration below:

```
{
"host": "10.0.40.32",
"port": 4444,
"newSessionWaitTimeout": -1,
"servlets" : [],
"prioritizer": null,
"capabilityMatcher":
"org.openqa.grid.internal.utils.DefaultCapabilityMatcher",
"throwOnCapabilityNotPresent": true,
"nodePolling": 5000,
"cleanUpCycle": 5000,
"maxSession": 15,
"jettyMaxThreads":-1
}
```

ii. **Node.json:** This file must contains the configuration below:

```
{
"capabilities":[
{
"platform": "WINDOWS",
"browserName": "firefox",
"maxInstances": 10,
"seleniumProtocol": "WebDriver"
}
],
"maxSession":15,
"port":5555,
"host":"10.0.40.32",
"register":true,
"registerCycle": 5000,
"hub":"http://10.0.40.32:4444"
}
```

7. Start the Hub.json and the Node.json
   a. Use the command to start Hub.json:

```
java -jar <path to selenium-server-standalone jar file> -role hub -hubConfig <path to hub.json file>
```

i. Example:

```
java -jar C:\selenium\selenium-server-standalone-3.0.1.jar -role hub -hubConfig C:\selenium\hub.json
```

b. Use the command to start Node.json:

```
java -jar -Dwebdriver.gecko.driver=<path to geckodriver to use the Firefox browser> <path to selenium-server-standalone jar file> -role node -nodeConfig <path to node.json file>
```

i. Example:

```
java -jar -Dwebdriver.gecko.driver=C:\selenium\<PROJECT>\geckodriver.exe -Dwebdriver.ie.driver=C:\selenium\<PROJECT>\IEDriverServer.exe -Dwebdriver.chrome.driver=C:\selenium\<PROJECT>\chromedriver.exe C:\selenium\selenium-server-standalone-3.0.1.jar -role node -nodeConfig C:\selenium\nodes<PROJECT>.json
```

8. Change the value of the parameter "SeleniumGridMachine" on defaultProperties.properties, to use the right IP to connect on Selenium GRID Machine.
   a. For example: "SeleniumGridMachine=http://<SERVER>:5555/wd/hub"

9. Run the tests.

Selenium GRID with Docker

1. Trying GRID with Docker browser hub.
   a. First things to do - Setup

i. Install Docker on your machine & start Docker.

ii. To check containers in docker run **docker ps -a**

iii. To setup selenium + Docker:
   1. download container image running
      a. **docker pull selenium/hub**

      b. **docker pull selenium/standalone-chrome**

      c. **docker images** and check selenium standalone there

      d. start container **docker run -p 4444:4444 --name selenium-chrome selenium/standalone-chrome**

      e. Change browser=dockerGC (via defaultProperties.properties)

      f. And run tests.