|  |  |
| --- | --- |
| BGP Assessment Completion Report | |
|  | Syed Akeel Ascar |
|  | Cognizant Technology Solutions |

**Table of Contents**

[**Overview** 3](#_Toc5825085)

[**Testing Scope** 3](#_Toc5825086)

[**Test Automation Tools used** 4](#_Toc5825087)

[**Challenges faced and Resolutions** 4](#_Toc5825088)

[**Locating Dynamic Web Elements** 4](#_Toc5825089)

[**Wait for Page Load Complete** 5](#_Toc5825090)

[**Capturing Scrolling Screenshot** 6](#_Toc5825091)

[**Test Automation Framework** 7](#_Toc5825092)

[**POM.xml and WebDrivers** 7](#_Toc5825093)

[**Program structure** 7](#_Toc5825094)

[**Steps to run Test** 9](#_Toc5825095)

[**Command Prompt Output** 10](#_Toc5825096)

[**Cucumber Report** 11](#_Toc5825097)

# **Overview**

The objective of this project is to develop an automation framework to automate the test scenario mentioned in Customer’s Business Grants Portal (BGP) project’s Tech Challenge using open source automation tool / framework.

# **Testing Scope**

Below is the Test scenario and Test Cases provided in the BGP project’s Tech Challenge.

|  |  |
| --- | --- |
| Test Scenario | Test Steps |
| Test Scenario | - Launch test environment URL : https://bgp-qa.gds-gov.tech  - Login as public user: public / Let$BeC001  - Login as BGP user: S9111111A  - Select ‘Get new grant’  - Select ‘Building & Constructions’ sector, select the sub-sector ‘Builders (Constructors)’  - Select ‘Upgrade key business area…’  - Select ‘Pre-Approved Productivity Solutions’  - Select ‘Apply’  - Select ‘Proceed’ |
| Acceptance Criteria | Test Steps |
| AC 1-1 | When no answer is selected for the question ‘Does the applicant meet the eligibility criteria’, the following items shall be disabled:  - the ‘Next’ button  - all side menu except Eligibility menu |
| AC 1-2 | When answer ‘Yes’, system shall enable the items indicated in AC1-1. User is allowed to navigate through side menus. |
| AC 1-3 | When answer ‘No’, system shall disable the items indicated in AC1-1 and, triggers warning message ‘Visit Smart Advisor on the SME Portal for more information on other government assistance.’. |
| AC 1-4 | When click on the link of Smart Advisor, system shall launch the website on the same windows tab with URL: <https://www.smeportal.sg/content/smeportal/en/moneymatters.html> |
| AC 1-5 | When I click on the ‘Save’ button, I should be able to save the form as draft. |
| AC 1-6 | When I go to the home page, I should see my draft form and when I click on it system should retrieve the form for editing. |
| AC 1-7 | When I fill up all the various sections (Contact Details, Proposal, Cost, Business Impact, Declare & Review), I should be able to submit the form and get back an application ref ID (success message). |

# **Test Automation Tools used**

|  |  |
| --- | --- |
| Automation Tools / Framework | Version |
| Cucumber | *4.2.4* |
| Selenium WebDriver | *3.14.0* |
| Junit | *4.12* |
| Ashot | *1.5.4* |
| Java | *1.8.0\_201* |
| Maven | *3.6.0* |

# **Challenges faced and Resolutions**

Below are the challenges faced during creating the Test automation framework for the given requirement.

* Location Dynamic Web Elements.
* Wait for Page Load Complete.
* Capturing Scrolling Screenshot.

Implemented the below resolutions to overcome the challenges mentioned.

## **Locating Dynamic Web Elements**

Most of the Web elements are dynamic and they will appear only after performing some action on some other elements, it’s not possible to just locate an element using the xpath.

Below logic was implemented to ensure the element is visible or clickable before performing any action on the web element.

|  |
| --- |
| **Method: clickElement** |
| public static void clickElement(WebElement element) {  waitForCondition(ExpectedConditions.elementToBeClickable(element), BGPConstant.WAIT\_LONG\_TIMEOUT);  focusElement(element);  executeJavaScript("arguments[0].click();", new Object[]{element});  waitForPageLoadComplete();  } |

Added explicit wait with ***ExpectedCondition*** to make sure the web element is visible or clickable.

|  |
| --- |
| **Method: waitForCondition** |
| public static void waitForCondition(ExpectedCondition<?> conditionToWaitFor, int timeToWait) {  (new WebDriverWait(BrowserManager.getDriver(), (long) timeToWait)).until(conditionToWaitFor);  } |

Once the status of element is located, ***Actions*** class was used to focus the element before performing any action.

|  |
| --- |
| **Method: focusElement** |
| public static void focusElement(WebElement elementToFocus) {  if (!elementToFocus.getTagName().equals("script")) {  try {  (new Actions(BrowserManager.getDriver())).moveToElement(elementToFocus).perform();  } catch (MoveTargetOutOfBoundsException var2) {  ;  }  }  } |

Using ***JavascriptExecutor*** to perform the actions on the elements pointed by ***Actions*** class.

|  |
| --- |
| **Method: executeJavaScript** |
| public static Object executeJavaScript(String script, Object... args) {  if (BrowserManager.getDriver() instanceof JavascriptExecutor) {  JavascriptExecutor jsExec = (JavascriptExecutor) BrowserManager.getDriver();  return args.length > 0 ? jsExec.executeScript(script, args) : jsExec.executeScript(script, new Object[0]);  } else {  throw new SessionNotCreatedException("No WebDriver session found");  }  } |

## **Wait for Page Load Complete**

In any web based application waiting for page load to complete is a common challenging factor. In BGP project clicking on buttons like Proceed, Save, Next and Submit takes long time to load the page.

Utilized ***JavascriptExecutor*** and implemented below logic to get the status of the page load to ensure the page load is completed before proceeding with the next step in the test script.

|  |
| --- |
| **Method: waitForPageLoadComplete** |
| public static void waitForPageLoadComplete() {  ExpectedCondition<Boolean> pageLoadCondition = new ExpectedCondition<Boolean>() {  public Boolean apply(WebDriver driver) {  boolean check1 = "complete".equals(executeJavaScript("return document.readyState", new Object[0]));  boolean check2 = true;  try {  String var4 = executeJavaScript("return jQuery.active", new Object[0]).toString();  byte var5 = -1;  switch (var4.hashCode()) {  case -1038130864:  if (var4.equals("undefined")) {  var5 = 0;  }  break;  case 48:  if (var4.equals("0")) {  var5 = 1;  }  }  switch (var5) {  case 0:  break;  case 1:  check2 = true;  break;  default:  check2 = false;  }  } catch (WebDriverException var6) {  check2 = true;  }  return check1 && check2;  }  };  waitForCondition(pageLoadCondition, BGPConstant.WAIT\_LONG\_TIMEOUT);  } |

## **Capturing Scrolling Screenshot**

Capturing Screenshot, as Selenium build-in ***TakeScreenshot*** interface doesn’t support scrolling option, so it fails to capture the full details as BGP pages are too long.

To capture the screenshot with scrolling feature implemented the logic using ***Ashot*** Class. The logic is implemented as below capture the screenshot and convert it into Byte Array. Finally embed the Byte Array to the scenario using *Hooks*.

|  |
| --- |
| **Method: captureAndAttachScreenshotToScenario** |
| public static void captureAndAttachScreenshotToScenario(Scenario scenarioToAttach) throws IOException {  byte[] imagesBytesToAttach = convertBufferedImageToByteArray(new AShot().  shootingStrategy(ShootingStrategies.viewportPasting(1000)).takeScreenshot(BrowserManager.getDriver()).getImage());  attachBytesToScenarioAsImage(scenarioToAttach, imagesBytesToAttach);  } |

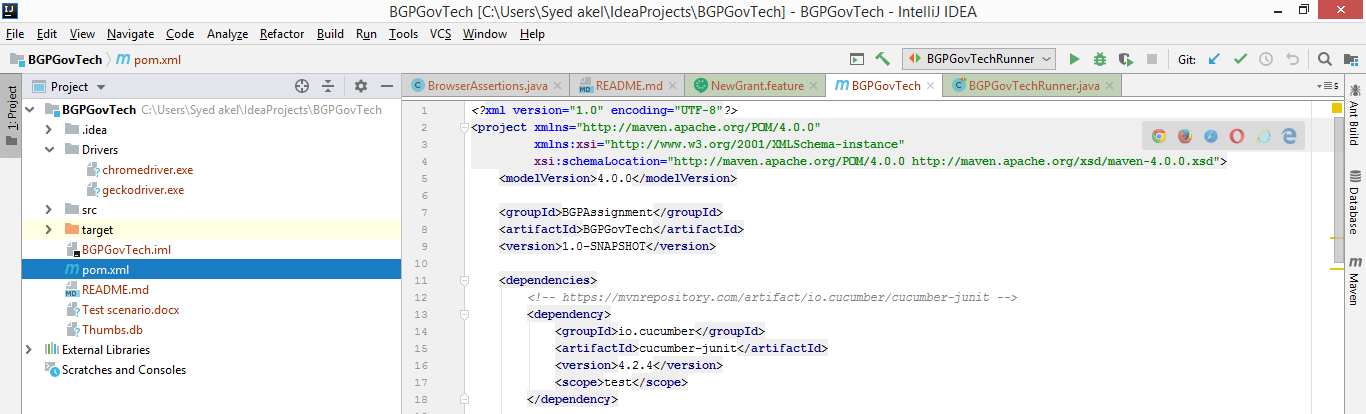
|  |
| --- |
| **Method: convertBufferedImageToByteArray** |
| public static byte[] convertBufferedImageToByteArray(BufferedImage bufferedImage) throws IOException {  ByteArrayOutputStream baos = new ByteArrayOutputStream();  ImageIO.write(bufferedImage, "png", baos);  baos.flush();  byte[] imageInByte = baos.toByteArray();  baos.close();  return imageInByte;  } |

|  |
| --- |
| **Method: attachBytesToScenarioAsImage** |
| private static void attachBytesToScenarioAsImage(Scenario scenarioToAttach, byte[] imagesBytesToAttach) {  scenarioToAttach.embed(imagesBytesToAttach, "image/png");  } |

# **Test Automation Framework**

Implemented Behavior Driven Development (BDD) test automation framework using Cucumber, Selenium, Java, Junit and Maven based on Project Objet Model (POM) design pattern for better test maintenance and reducing code duplication. According to POM, tests scripts and element locators kept separately, this will keep code clean and easy to understand and maintain.

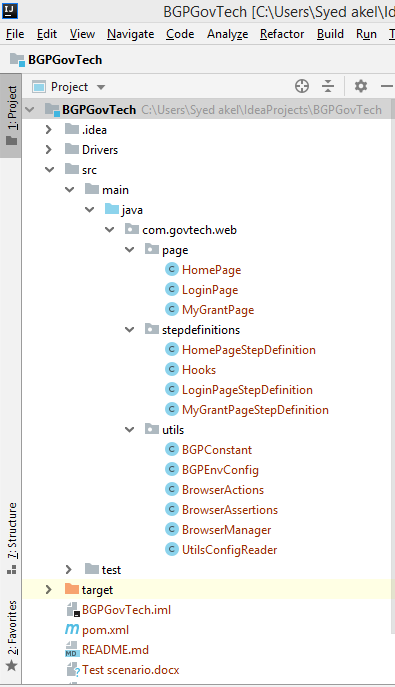
## **POM.xml and WebDrivers**



## **Program structure**

**/src/main/java/com.govtech.web**

* **page:** contains Java classes which defines elements locators of respective pages in BGP portal.
* **stepdefinitions:** contains Java classes that access element locators Java files and implements step definitions of respective pages for the steps write in feature files.
* **utils:** contains Browser and other util related Java classes.

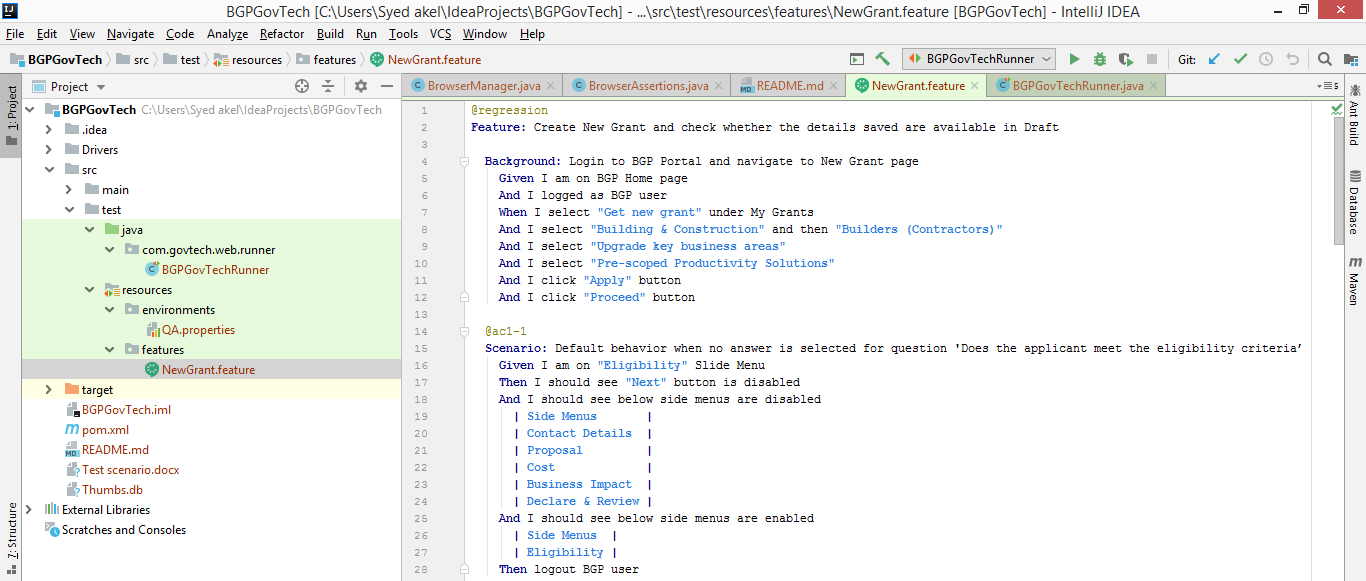


**/src/test/java/com.govtech.web.runner/BGPGovTestRunner:**

* This is the main class that binds the Cucumber feature file with the step definitions and also for reporting.
* This also contains the base test runner class that implements the methods to trigger ***setUp()*** and ***tearDown()*** to initiate and quit the WebDriver to run test automation.

**/src/test/java/resources:**

* ***environments/QA.properties:*** contains BGP QA environment properties such as protocol, URL, username, password.
* ***Features:*** contains all Cucumber features files.



# **Steps to run Test**

1. Download BGP GovTech Project from GitHub using following link <https://github.com/syedakeelascar/BGPGovTechProject.git>
2. Extract the project into local destination project folder.
3. Open the command prompt and go to the project folder location where pom.xml is located.
4. Run the below command to execute the BGP regression automation suite.
5. *mvn clean install -U -Dbrowser=CHROME -Dtest="BGPGovTechRunner.class" -Dbgpenv=QA -Dcucumber.options="--tags @regression"*
6. Once the execution is completed report will be generated under *target/cucumber/html-report*, open *index.html* file to view the report.

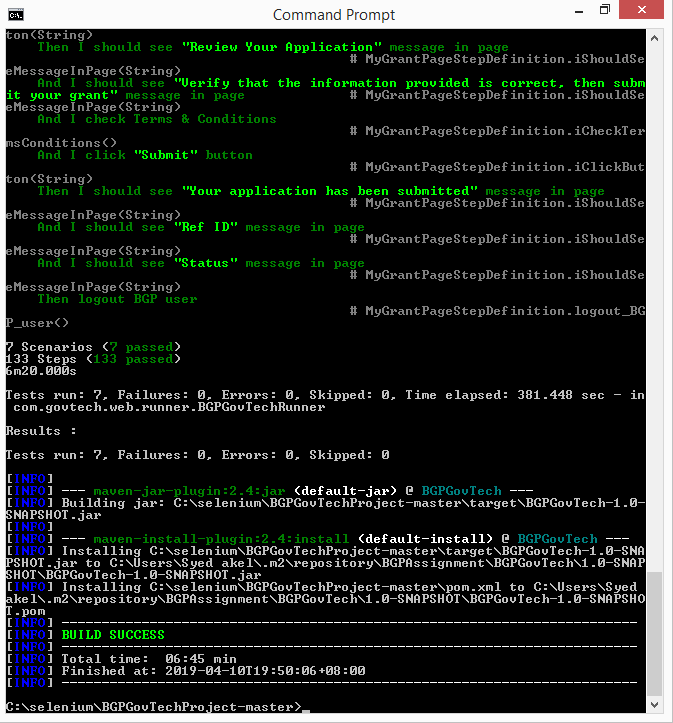
**Note:** cucumber options tags are customizable based on the requirement.

*-Dcucumber.options="--tags @regression"  - to run all test cases.*

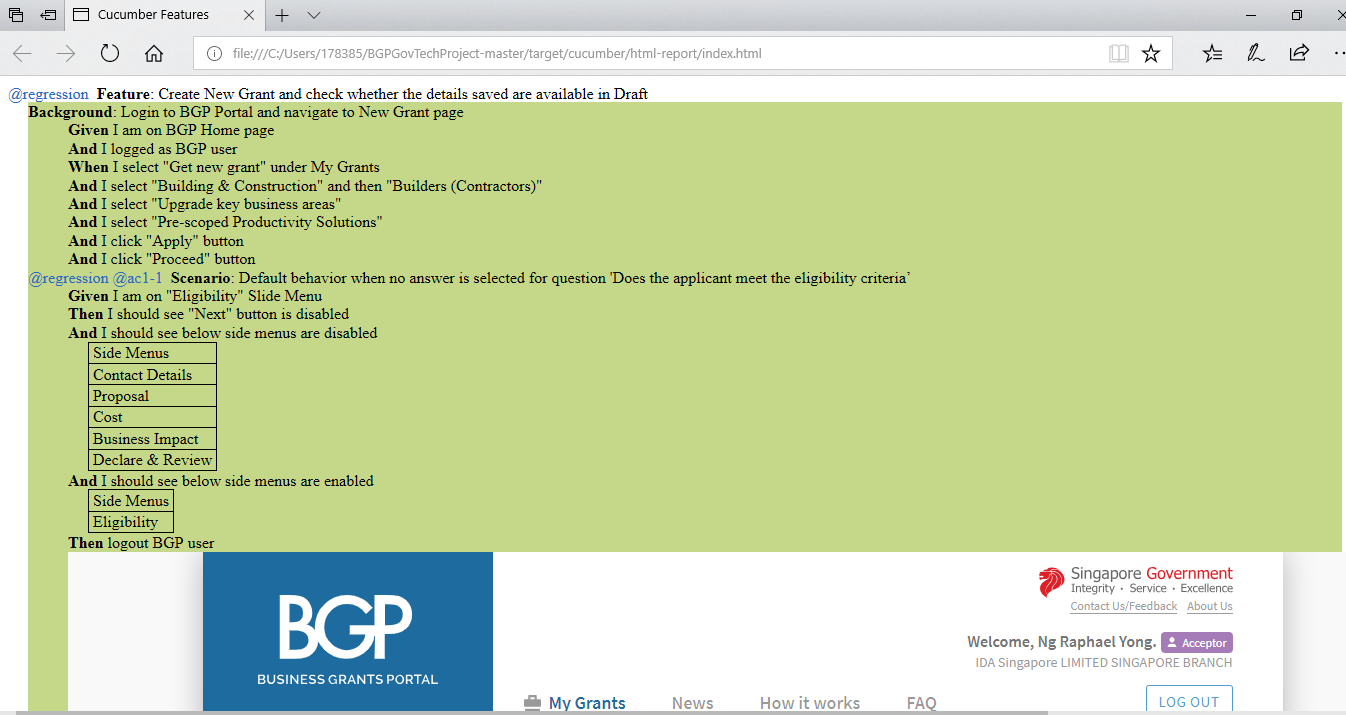
*-Dcucumber.options="--tags '@ac1 or @ac2'" - to run only test cases - @ac1-1 and @ac1-2*

## **Command Prompt Output**





## **Cucumber Report**

****