ECSE 428
March 9th, 2019

Assignment B

Mahad Khan
260678570
mahad.khan@mail.mcgill.ca

Summary of Deliverables:

Story Statement

Story:
As a Gmail user,
I would like to be able to send an email with an image attachment
So the recipient can view the image

Normal Flow: Sending an email after uploading an image file saved on my device

Given I am on Gmail's compose email page
And I have filled in the "Recipients" and "Subject" fields
When I attach an image file from my device
And the image is successfully uploaded from device
Then I can send the email with the image attached

Normal Flow: Sending an image attachment as a reply

Given I am signed in
And I have selected to reply to an email in my inbox
When I attach an image file
Then I can send the reply with the image attached

Normal Flow: Sending an image attachment to one regular and one cc email

Given I am on Gmail's compose email page
And I have filled in the "Recipients" and "CC" fields
And I have filled in the subject field
When I attach an image file from my device
And the image is successfully uploaded from device
Then I can send the email with the image attached

Alternate Flow: Sending an email with an image attachment and without a subject

Given I am on Gmail's compose email page
And I have filled in the "Recipients" fields
When I attempt to send an image file from my device
And I accept the warning

Then the email will be sent

Error Flow: Sending an email after removing an attached image file

Given I am on Gmail's compose email page
And I have filled in the "Recipients" and "Subject" fields
And I have attached an image file
When I press "X" next to the uploaded attachment
Then the email is sent without the image attachment

Test Environment Description

General Setup

The step definitions and features for Cucumber were created in a single project using Java 8 Update 101 and Maven 3 with IntelliJ IDEA 2016 Community version. Selenium v3.1 was used for web automation and was essential to testing Gmail's image attachment email feature. The Chrome Web Driver was used with Selenium for all acceptance tests. The OS on which tests were run was macOS Mojave Version 10.14.2. More detail about the source code is available later in this document.

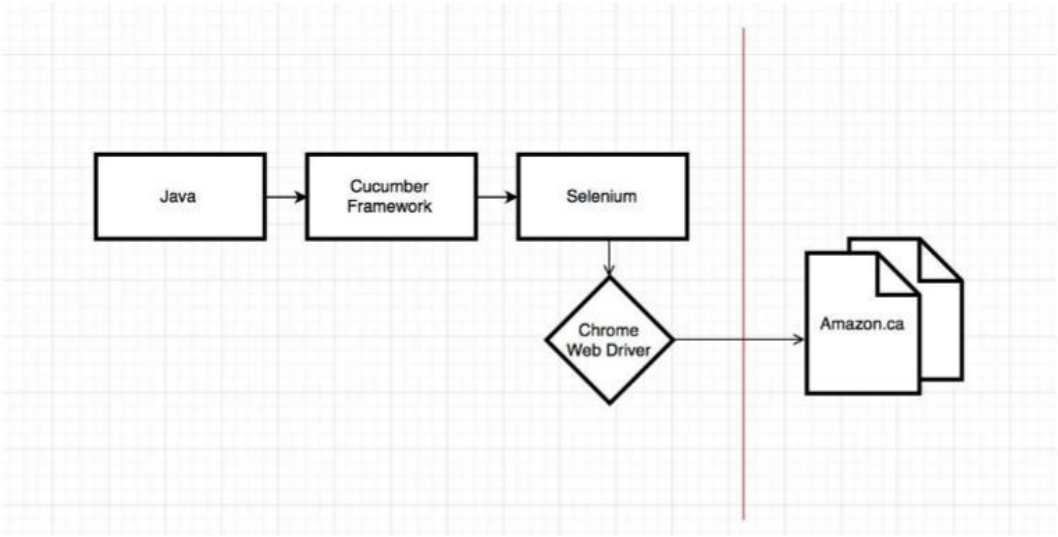
Cucumber

Maven was used to install the dependencies for Cucumber with version 1.0.11. All the Cucumber tests were run locally on the machine using IntelliJ. The scenarios are contained within the feature file, gmail.features. Using IntelliJ we can run the feature file locally to execute our tests. The configurations need to be set accordingly. A screenshot of the configurations and further deatials on installing and running the cucumber-based tests can be found later in this document. The following are the Maven Cucumber dependencies required for our project:

```
<dependency>
    <groupId>info.cukes</groupId>
    <artifactId>cucumber-java</artifactId>
    <scope>test</scope>
    <version>1.0.11</version>
</dependency>
<dependency>
    <groupId>info.cukes</groupId>
    <artifactId>cucumber-jvm</artifactId>
    <version>1.0.11</version>
    <type>pom</type>
</dependency>
<dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>cucumber-junit</artifactId>
    <version>4.2.5</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
</dependency>
```

Block Diagram

The following block diagram (copied from an assignment posted on myCourses) illustrates how Java encompasses all of the Cucumber Framework. We create the Gherkin scripts with Java and the step definitions with Selenium to automate actions through Chrome. In the block diagram we should assume that Amazon.ca is replaced with Gmail's login page as the communication remains the same.



Cucumber Gherkin Scripts

The following Cucumber Gherkin script is used for the tests we are performing:

```
Feature: Gmail

  ## Normal Flow
  Scenario: Sending an email after uploading an image file saved on my device
    Given I am on Gmail’s compose email page
    And I have filled in the Recipients and Subject fields
    When I attach an image file from my device
    And the image is successfully uploaded from device
    Then I can send the email with the image attached

  ## Normal Flow
  Scenario: Sending an image attachment as a reply
    Given I am signed in
    And I have selected to reply to an email in my inbox
    When I attach an image file
    Then I can send the reply with the image attached

  ## Normal Flow
  Scenario: Sending an image attachment to one regular and one cc email
    Given I am on Gmail’s compose email page
    And I have filled in the Recipient and cc field
    And I have filled in the subject field
    When I attach an image file from my device
    And the image is successfully uploaded from device
    Then I can send the email with the image attached

  ## Alternate Flow
  Scenario: Sending an email with an image attachment and without a Subject
    Given I am on Gmail’s compose email page
    And I have filled in the Recipient field
    When I attempt to send an image file from my device
    And I accept the warning
    Then the email will be sent

  ## Error Flow
  Scenario: Sending an email after removing an attached image file
    Given I am on Gmail’s compose email page
    And I have filled in the Recipients and Subject fields
    And I have attached an image file
    When I press X next to the uploaded attachment
    Then the email is sent without the image attachment
```

New Gherkin scripts can be added too. To do so, we first need to access the feature file that can be found under src/test/resources/ **gmail.feature**. The Gherkin keywords that we are using and can be used to add more tests are Feature, Scenario, Given, When, Then and And. To add more

tests, we first define the Scenario which just illustrates what we are attempting to do. Once we have defined the Scenario, we follow a pattern such that we use the Given step to describe an initial context, the When step to describe an event and the Then step to describe an expected outcome. Once we have our completed Cucumber Gherkin script, we have to define the step definitions for each step. To do so, we access the step definitions file that can be found under src/test/java/com/mahadkhan/cucumber/**StepDefinitions.java**. A step definition is a Java method with an expression that links it to one or more Gherkin steps. When Cucumber executes a Gherkin step in a scenario, it will look for a matching step definition to execute. The step definition should include all the actions to be performed for that step to be completed. To illustrate how this works, look at the following code snipped from out StepDefinitions.java file:

```
@Given("^I am on Gmail's compose email page$")
public void givenOnComposeEmail() throws Throwable {
    setupSeleniumWebDrivers();
    signIn();
    System.out.println("Attempting to find Compose button");
    WebElement composeBtn = (new WebDriverWait(driver, 10))

.until(ExpectedConditions.elementToBeClickable(By.className(COMPOSE_BTN)));
    System.out.println("Found!");
    composeBtn.click();
    try {
        System.out.println("Attempting to open email composer");
        //Verifying if the composer is actually visible
        WebElement composer = (new WebDriverWait(driver, 10))
                .until(ExpectedConditions.presenceOfElementLocated(By.id(":6g")));
        System.out.println("Email composer successfully opened");
    } catch (Exception e) {
        System.out.println("Did not successfully open email composer");
    }
}
```

This illustrates a Given step used in our scenarios and can be used as a reference when adding new step definitions. Keep in mind that the Then step should usually include a JUnit test that asserts whether our test has passed or failed.

Source Code

The source code has been submitted along with this file on myCourses. The zipped file name ECSE428-Assignment_B.zip can be unzipped and opened as a project in an IDE of your choice. For this project, I used the IntelliJ IDE. The source code can also be found under: https://github.com/mahad96/ECSE428-Assignment_B.

For Cucumber:

Step definitions found under:
src/test/java/com/mahadkhan/cucumber/**StepDefinitions.java**
Feature file found under:
src/test/resources/**gmail.feature**

The pom.xml file contains all the dependencies required for the project and will be installed using Maven.

The images folder contains all the images used for testing purposes.

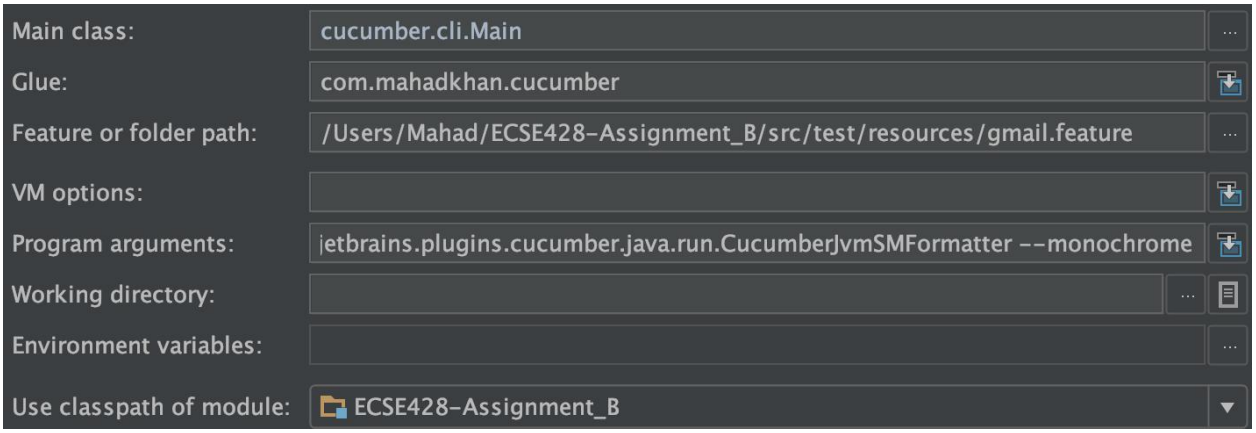
Instructions to install test environment and run tests (Cucumber)

To start off, we will first need the source code. This can be cloned from the github repository using the link provided below:
https://github.com/mahad96/ECSE428-Assignment_B
The source code has also been delivered as a zip file named ECSE428 - Assignment_B.zip. This file can be unzipped and opened as a project in an IDE.

Secondly, in order for Selenium to properly utilize the Chrome Web Driver, please download the driver here:
https://sites.google.com/a/chromium.org/chromedriver/downloads

Once the project is opened in an IDE of your choice, edit the PATH_TO_CHROME_DRIVER variable (line 31 in StepDefinitions.java) to be the absolute path to the newly downloaded Chrome Web Driver.

Using IntelliJ, we then need to build and compile the project and run gmail.feature with the following configurations:



The entire feature can be tested at once or separate scenarios can be tested individually. To test the entire feature, we run the project using the above configurations. To test each scenario separately, we can right-click the Scenario keyword in IntelliJ and select Run.

Cucumber Implementation

The Cucumber testing framework is ideal for technical users for creating acceptance tests and scenarios based on Gherkin scripts. This framework focuses on behavior-driven-development. The tests can be configured entirely on one IDE (e.g. IntelliJ) and can be run directly via a feature file. This feature file contains the Gherkin scripts. Implementation is fairly easy. The tests can be directly run from the IDE and are light weight. For the implementation of Cucumber, no new programming language need to be learnt other than the creation of Gherkin scripts.

In our case, the benefits of using Cucumber were that implementation of the test cases was fairly simple. Once the Gherkin script is created, we just need to define the actions to be taken in each step. The step approach allows for different sets of actions to be isolated. This makes debugging easier as one knows on which step the test has failed and therefore has a reference point to start debugging.

There are risks associated with our implementation of Cucumber too. Because we are testing a feature of Gmail, we can not ensure that the element identifiers remain the same on every test run. Some identifiers are set by Google to change dynamically. Problems in this regard were faced while testing and some identifiers had to be changed frequently.

If we were to use Cucumber in a commercial project, tests would have to be way more thorough. Greater number of checks for elements would have to be made. A major improvement would be automatic failure reporting to responsible teams that will get a pager alert when the application starts to fail. Another improvement would be to run the tests on frequent intervals using a build tool like TravisCI.

In the case that the web based email GUI changed, we will have to identify what element identifiers have been changed and replace them accordingly. In some cases the flow of events may change too and the tests will have to be re-written accordingly.

Summary of Files Delivered

The files that have been delivered are a zip file with the source code, a zip file containing videos of the test scenarios and this document. The zip file with the source code is named ECSE428-Assignment _B.zip and contains all the source code and relevant images for testing purposes. The zip file containing the videos is named Videos.zip. This contains videos in .mov format for all five scenarios. This document has also been delivered and can be used as a reference.