Assignment B

Cucumber in Agile Story Test Automation

Presented to Professor Robert Sabourin
ECSE 428: Software Engineering Practice
On 03/10/19

Presented by:

Alexander Harris 260688155 alexander.harris@mail.mcgill.ca
Abbas Yadollahi 260680343 abbas.yadollahi@mail.mcgill.ca

TABLE OF CONTENTS

STORIES

Story: Send email with an image attachment

- **As a** Google user
- **I can** use Google's email service
- **So I can** send an email with an image attached

Normal Flow: Send an email with an attachment to another email address

1. **Given** I am composing an email
2. **And** it is addressed to a recipient
3. **And** the subject contains a given subject
4. **And** a sample text is entered in the body
5. **And** we attached an image to the email
6. **When** we send the email
7. **Then** the recipient should receive the email with the given subject
8. **And** it should be from the Sender's email
9. **And** the email's body should match the sample text
10. **And** the email's attachment should match the image

Alternate Flow: Send an email with too large attachment(s) to another email address

1. **Given** I am composing an email
2. **And** it is addressed to a recipient
3. **And** the subject contains a given subject
4. **And** a sample text is entered in the body
5. **And** we attached an array of images to the email
6. **Then** I should be prompted to upload the attachments to Google Drive
7. **And** we should be able to send the email
8. **And** the recipient should receive the email with the given subject
9. **And** it should be from the Sender's email
10. **And** the email's body should match the sample text
11. **And** the email's attachment(s) should match the images

Error Flow: Send an email with an attachment to an invalid email address

1. **Given** I am composing an email
2. **And** it is addressed to a recipient
3. **And** the subject contains a given subject
4. **And** a sample text is entered in the body
5. **And** we attached an image to the email
6. **When** we try to send the email
7. **Then** we should receive an email error

REQUIREMENTS

- Python 3.6 or higher
- Linux-based OS
- Chrome 72 or higher
- Selenium 3.141.0 or higher
- Behave 1.2.6 or higher
- Webdriver-manager 1.7 or higher
- Retrying 1.3.3 or higher

All package dependencies are automatically installed using pip, as detailed in the following section.

SETUP

Setup is fairly simple for this project, thanks to pip and Webdriver manager:

- Clone the repository at https://github.com/alexH2456/ECSE428-AssignmentB and change into the project directory.
- Optionally, activate your Python 3.6 virtual environment.
- Run `pip install -r requirements.txt` to install all dependencies.

TESTS

Running Tests

- Run `behave` in the linux terminal to start the test suite.
- Script should automatically download the Chrome Web Driver for your platform.
- All tests will run automatically and you will receive a summary upon completion.

Adding New Tests

To add new Gherkin tests, first add your Gherkin acceptance test definition as a .feature file under the `features` folder. Then create a .py file with the same name under the `features/steps` folder. Use the decorators provided by Behave to link your Gherkin steps with the functions you define in this file. For example:

Gherkin step:

```
Given we are at the login page
```

Python step:

```
@given('we are at the login page')
def step_given_login(context):
    pass
```

Please see the Behave documentation for more information.

TEST ENVIRONMENT

General Description

This suite uses Behave, which is a behaviour-driven development (BDD) based testing framework built on the Gherkin language. The Gherkin feature file that describes the scenario outlines for our acceptance tests can be found in `features/email.feature`. The step definitions for this feature file are found in `features/steps/email.py`. The `features/environment.py` file defines the before and after functions for Gherkin steps, as well as instantiating the web driver object for our tests. It also includes methods like `delete_emails` to help return the system to its initial state after running all tests.

The UI component of the testing is implemented using Selenium and the Chrome Web Driver. We use Webdriver Manager to automatically download the required Web Driver for the user's platform. We also use Retrying for adding retry decorators to certain methods. All relevant dependency information for the project can be found in `requirements.txt`.

Additionally, `gmail.py` implements many helper methods needed for our UI tests, like `check_email_received`, a method to verify that an email was sent by the sender and received by the recipient. All UI element mappings can be found in `locators.py`. These locator classes were created to make the test suite easier to update in the event that UI elements change.

Benefits and Risks

Since UI testing is fairly unstable by definition, our test suite can still be subject to errors caused by circumstances outside of our control. Additionally, since we do not own the product we are testing, we do not always have full control over the initial conditions for our tests. An example of this we encountered during development was that Google would sometimes give us a Captcha during login, which means that we would be unable to perform any tests. Developers at Google likely have a version of their login form with Captcha disabled for their UI testing so they would not encounter this issue.

One of the benefits of our design is that it is fairly modular. Many step definitions are reused and we wrote a lot of helper methods for performing common tasks. This helps reduce code duplication and

makes debugging and maintainability much easier. We also put all locators (XPath, CSS Selector, ID, etc.) in a separate file. This means that if just the properties of the UI elements change and not the entire flow of the scenario, we should be able to simply update these locators to make our scripts work again. Obviously, if the entire process of composing an email changes in a significant way it will necessitate a more comprehensive rewrite, such as rewriting the step definitions or modifying certain helper methods to reflect the new system behaviour.

However we believe the Gherkin scripts themselves are generic enough that they should not need to be updated unless porting to another email platform, in which case our current alternative flow might need to be changed slightly, as the other platform would likely not have the Google Drive functionality provided in Gmail.

Improvements

In order to improve the modularity of our code, in the future we would want to organize helper methods into classes corresponding to different web pages in the UI, since common functions such as deleting emails can have different flows depending on the page. Currently we are just passing parameters into the functions where this is the case, but this change would help make our code more readable, which is important in a commercial setting where many people would be working on the same codebase.

Additionally, we would like to add support for other browsers such as Firefox and Internet Explorer, as well as automate a process for running our test suite on different versions of each of these browsers, possibly using a CI/CD tool like Travis CI. We can also add more alternative flows, such as having multiple recipients or using CC/BCC, as well as more varied image formats. Currently, for checking attachments on the recipient, we only verify that the attachment name is the same, however we could implement hashing to verify the images themselves are identical.

GHERKIN SCRIPT

```
Feature: Send email with image attachment

    Background: At inbox page
        Given we have logged into gmail
        And we are on the inbox page

    Scenario Outline: Normal Flow: Send an email with "<subject>" as subject to
<recipient> with <image> attachment
        Given we are composing an email
        And it is addressed to "<recipient>"
        And the subject contains "<subject>"
        And a sample text is entered in the body
            """
            Hello, this email contains an image.

            Regards,
            Sender
            """
        And we attached "<image>" to the email
        When we send the email
        Then "<recipient>" should receive the email with subject "<subject>"
        And it should be from the Sender
        And the body should match the sample text
        And the attachment/s should match "<image>"

        Examples:
            | subject       | recipient  | image       |
            | Normal Flow 1 | recipientA | imageA.jpg  |
            | Normal Flow 2 | recipientB | imageB.jpg  |
            | Normal Flow 3 | recipientA | imageB.jpg  |
            | Normal Flow 4 | recipientB | imageA.jpg  |
            | Normal Flow 5 | recipientA | imageC1.png |

    Scenario Outline: Alternative Flow: Send an email with "<subject>" as subject to
<recipient> with <images> attachment too large
        Given we are composing an email
        And it is addressed to "<recipient>"
        And the subject contains "<subject>"
        And a sample text is entered in the body
            """
            Hello, this email contains image/s.

            Regards,
            Sender
            """
        When we attached "<images>" to the email
```

```
        Then we should be prompted to upload to Google Drive
        And we should be able to send the email
        And "<recipient>" should receive the email with subject "<subject>"
        And it should be from the Sender
        And the body should match the sample text
        And the attachment/s should match "<images>"

        Examples:
            | subject            | recipient  | images                                |
            | Alternative Flow 1 | recipientA | imageLarge.jpg                        |
            | Alternative Flow 2 | recipientB | imageLarge.jpg                        |
            | Alternative Flow 3 | recipientA | imageC1.png imageC2.png imageC3.jpg |
            | Alternative Flow 4 | recipientB | imageC1.png imageC2.png imageC3.jpg |
            | Alternative Flow 5 | recipientA | imageA.jpg imageLarge.jpg           |

    Scenario Outline: Error Flow: Send an email with "<subject>" as subject to
<recipient> with invalid email address and <image> attachment
        Given we are composing an email
        And it is addressed to "<recipient>"
        And the subject contains "<subject>"
        And a sample text is entered in the body
            """
            Hello, this email should not send.

            Regards,
            Sender
            """
        And we attached "<image>" to the email
        When we try to send the email
        Then we should receive an email error

        Examples:
            | subject      | recipient            | image      |
            | Error Flow 1 | @not.valid.gmail@com | imageA.jpg |
            | Error Flow 2 | wotisdis             | imageB.jpg |
            | Error Flow 3 | abc.123.gmail.com    | imageB.jpg |
            | Error Flow 4 | ilikecucumbers@gmail | imageA.jpg |
            | Error Flow 5 | gherkin@.gmail.com   | imageB.jpg |
```

FILE SUMMARY

`features:` Folder containing all behave associated files and Gherkin feature files.

`steps:` Folder containing all Gherkin step definitions.

- `email.py:` Contains all step definitions for email.feature Gherkin script.

- `email.feature:` Gherkin script.
- `environment.py:` Environment description for behave.

`images:` Folder containing images of different sizes used as attachments in the tests.

- `gmail.py:` Contains helper methods used for UI tests.
- `locators.py:` Contains locators for UI elements used in tests.
- `README.MD:` Contains project description and instructions.
- `requirements.txt:` Project dependencies.
- `test_run_success.webm:` Video of successful test run. Also available at: https://youtu.be/XWnZJmQW-lU

The source code for this project is hosted on GitHub in the following repository: https://github.com/alexH2456/ECSE428-AssignmentB