📨 Gmail Selenium Test

Background

- Student ID: 260687719
- Name: Erick Zhao

This assignment was submitted as Assignment B for ECSE 428 at McGill University in the Winter 2019 semester.

Table of contents

Story statement

Story

```
As a user of the Gmail mail provider,
I would like to be able to send emails with image attachments
so I can communicate asynchronously via text and image  to anyone with an
internet connection across the world.
```

Flows

To respect the guidelines of this assignment, there is a normal, alternate, and error flow available, as well as 5 different test cases for each flow, which each have a different permutation of recipients and image attachment file types.

A separate scenario outline was used for each one of these flows, with a set of 5 variable combinations included for each one. See the Gherkin scripts below for more details, and the `/lib/users.js` file (link to GitHub here) for each specific email to be used.

Normal flow

```
Normal flow: Sending emails to a valid recipient using files from Computer
Given I have an email draft is addressed to someone with someone else as a
Cc
```

```
And I have chosen a single image to be attached from my local computer
When I send the email
Then I should be alerted that the email was sent successfully
And the draft should no longer be available
And the email should be sent
And the email's details should correspond to the original draft that was
sent
```

Alternate flow

```
Scenario Outline: Sending emails to a valid recipient using files from
Google Drive
Given I have an email draft is addressed to someone with someone else as a
Cc
And I have chosen a single image to be attached from Google Drive
When I send the email
Then I should be alerted that the email was sent successfully
And the draft should no longer be available
And the email should be sent
And the email's details should correspond to the original draft that was
sent
```

Error flow

```
Error flow: Sending emails with an image attachment to an invalid
recipient
Given an email draft is addressed to someone with someone else as a Cc
And I have chosen a single image to be attached from my local computer
When I send the email
Then the draft should remain open
And I should be warned that the recipients are invalid
And the email should not be sent
```

Gherkin scripts

These scripts are also accessible via the repository in the `/features/send_email.feature` file. (Link to GitHub here)

```
Feature: Sending email with image attachment

  Background:
    Given CurrentUser is logged into the Gmail web client
    And CurrentUser is composing a new message

  Scenario Outline: Sending emails to a valid recipient using files from
Computer (Normal Flow)
```

```
    Given an email draft is addressed to <recipient> with <cc> as a Cc
    And a single <filetype> image is attached from CurrentUser's local
computer
    When email is sent
    Then CurrentUser should be alerted that the email was sent
successfully
    And the draft should no longer be available
    And the email should be sent
    And the email's details should correspond to the original draft that
was sent

    Examples:
      | recipient     | cc            | filetype |
      | "CurrentUser" | "OtherUser-1" | ".png"   |
      | "CurrentUser" | "OtherUser-2" | ".jpg"   |
      | "OtherUser-3" | "CurrentUser" | ".tiff"  |
      | "OtherUser-4" | "CurrentUser" | ".gif"   |
      | "OtherUser-5" | "CurrentUser" | ".svg"   |

  Scenario Outline: Sending emails to a valid recipient using files from
Google Drive (Alternate Flow)
    Given an email draft is addressed to <recipient> with <cc> as a Cc
    And a single <filetype> image is chosen to be attached from Google
Drive
    When email is sent
    Then CurrentUser should be alerted that the email was sent
successfully
    And the draft should no longer be available
    And the email should be sent
    And the email's details should correspond to the original draft that
was sent

    Examples:
      | recipient     | cc            | filetype |
      | "CurrentUser" | "OtherUser-1" | ".png"   |
      | "CurrentUser" | "OtherUser-2" | ".jpg"   |
      | "OtherUser-3" | "CurrentUser" | ".tiff"  |
      | "OtherUser-4" | "CurrentUser" | ".gif"   |
      | "OtherUser-5" | "CurrentUser" | ".svg"   |

  Scenario Outline: Sending emails to invalid recipient using files from
Computer (Error Flow)
    Given an email draft is addressed to <recipient> with <cc> as a Cc
    And a single <filetype> image is attached from CurrentUser's local
computer
    When email is sent
    Then the draft should remain open
    And the user should be warned that the recipients are invalid
    And the email should not be sent
    Examples:
      | recipient       | cc              | filetype |
      | "CurrentUser"   | "InvalidUser-1" | ".png"   |
      | "CurrentUser"   | "InvalidUser-2" | ".jpg"   |
      | "InvalidUser-3" | "CurrentUser"   | ".tiff"  |
      | "InvalidUser-4" | "CurrentUser"   | ".gif"   |
      | "InvalidUser-5" | "CurrentUser"   | ".svg"   |
```

Test environment description

High-level testing approach

My test approach involved using three Scenario Outlines to mix and match the 3 flows described above with a combination of variables: Recipient email, Cc email, and image attachment type.

First, I set up the initial state before each scenario by ensuring login status and navigating to the inbox page. This was done in the `Background` step of my Gherkin feature.

For each case, I sent composed a draft with a unique subject and body (based on the timestamp of when the acceptance tests were run), which were saved later for assertion purposes. Then, I attempted to send the email to both an exterior user and to myself from my test account (whether I was the main recipient or the Cc varied between the permutations).

Since each email was eventually sent to my inbox, I checked both the Inbox and Sent folders of my test account for a matching subject (corresponding to my previous timestamp). I then clicked on the email and validated if the information corresponded to what I had composed in my draft (subject, body, sender, recipient, cc, and attachment file name).

After each scenario was run, I then cleared my Inbox and Sent folders, as well as discarded any existing drafts, to reset the system to its initial state.

Dependencies

This assignment was written in **JavaScript** using the NodeJS runtime and the npm package manager. The project was built using Selenium Webdriver (with ChromeDriver for running the acceptance tests), Cucumber to automate the user story flows into acceptance tests, and Chai as an assertion library to validate the `Then` steps in the acceptance tests.

Here are the versions of the dependencies specified by the `package.json`:

- `"chai": "^4.2.0",` - Chai is an BDD assertion library (See documentation here).
- `"chai-as-promised": "^7.1.1",` - A plugin for Chai to support assertions for asynchronous JavaScript Promises (see documentation here).
- `"chromedriver": "^2.46.0",` - npm wrapper used to fetch the corresponding version of ChromeDriver (see documentation here). ChromeDriver itself is a standalone Chromium server implementing the WebDriver protocol. It is used in conjunction with Selenium to run acceptance tests.
- `"cucumber": "^5.1.0",` - A JavaScript implementation of the Cucumber tool to run plain language automated tests (see documentation here).
- `"selenium-webdriver": "^4.0.0-alpha.1"` - JavaScript bindings for the Selenium automation library (see documentation here).

N.B. This notation follows the npm guidelines for semantic versioning (link here). In general, the caret (`^`) icon next to any `X.X.X` version means that any `X.X.Y` version downloaded would also be accepted.

Hardware

All tests were run on macOS Mojave 10.14.3 with a 2018-model Macbook Pro with a 2.7 GHz Intel Core i7 processor and 16 GB 2133 MHz LPDDR3 RAM.

Test account

A fresh Gmail account was created for the purposes of this exercise. Since the alternate flow for the project requires image attachments from Google Drive, I had to manually upload the five images from the `/images/` folder into my test account's Google Drive folder.

Pros and cons of approach taken

A few aspects of my approach had some tradeoffs,which I outline in the sections below.

Sending emails to myself

The main peculiarity of my approach was that I always sent the email back to myself. This allowed me to ensure that my email was successfully delivered, since I could check if one of the recipients (my test account) had indeed recieved the email. This is a more robust check than just inspecting my Sent folder (because there could have been a breakdown between the email going to my outbox and the recipient receiving the email). It also allows me to perform such a check without validating with an external API call (or navigation) to another email's inbox, which would add a lot more complexity to the project.

However, this approach has its downsides, since it does not reflect a very common real-world use-case. People rarely send emails to themselves, unless they want to send themselves a file for later.

Checking file validity

To validate that the image attachment was indeed uploaded correctly, I simply checked if the file that appeared as an attachment had a matching name (including the extension). This is a quick and easy way that naively trusts Gmail's image attachment system to not corrupt the image. However, the best way to see if the image was indeed uploaded correctly would be to download it and compare the binary to the originally uploaded picture. I believed this to be out of scope of the assignment, and did not take the time to do so.

Teardown

The post-scenario teardown step in my code implemented in `/features/step_definitions/after)steps.js` (GitHub link here) sets a clean slate by deleting all emails sent. This is fine and easy because I'm using a test account. However, to more accurately replicate real-world usage, we would want to only delete the email that was sent within the scenario.

Specific locators

The Selenium locators I used to run the browser automation are specific to the Document Object Model of the Gmail client, with many relying on ancestor, sibling, or descendant selectors, and some even relying on the inner texts within the elements. This makes the tests less robust than having specific IDs or CSS classes to target. However, this was probably my best approach given that we were running tests on the production build of Gmail, with the IDs and CSS classes obfuscated and changing with each instance of the web client. My approach might have been different had I had access to a development build with legible, static class names and IDs.

Using inner texts also means that I cannot testing across different localizations of the Gmail client. This was fine since testing in multiple locales was out of the scope of this project, but this is still a vulnerability nonetheless.

Future improvements (commercial use)

In the previous section, I discussed some tradeoffs with the robustness of my approach. Indeed, the aforementioned solutions lead to more robust software more suitable for commercial use, but definitely require the additional resources that come with a commercial product:

- Scenarios with only external recipients with an API to tell the acceptance tests that the email was successfully sent.
- Checking image attachment binaries to ensure validity of the image upload system.
- Only teardown emails that were sent in the current scenario.
- Use a development build with HTML DOM attributes that are static and legible.
- Store string constants to match all locales.

If the web interface changed

I discussed this in detail in the section above (link here), but the gist of it is that my approach is fairly vulnerable to changes in the HTML DOM (due to use of very specific locators being used to run Selenium tasks), but this is partially due to me having to run automation on a production build of Gmail. In its current state of my project, if the UI changed, I would have to rebind all outdated selectors.

Installation

Before starting installation, make sure that you have NodeJS installed, with the latest LTS version (`v10.15.3`). Also, ensure you have the latest version of npm (`6.4.1`).

To run this project, first clone the GitHub repository (link below). Next, install all packages via command line with:

```
npm install
```

You should then see a `/node_modules/` folder appear with all dependencies installed. For more information on each NPM dependency, check the `package.json` file or see the Dependencies section above.

Running tests

Once all packages are done installing, run the test command in your command line interface using:

```
npm test
```

This will execute the Cucumber scripts and run the whole test suite. Your output should look something like this:

```
[gmail-selenium-test] npm test

> gmail-selenium-test@0.1.0 test /Users/erickzhao/Code/gmail-selenium-test
> cucumber-js

..........................................................................
.......................................................................

15 scenarios (15 passed)
130 steps (130 passed)
0m54.203s
```

Adding new Gherkin scripts and steps

To add new Gherkin scripts to the same feature, append your script to `/features/send_email.feature`. To add additional features, create any `.feature` file within the `/features/` directory.

To add corresponding steps, add a new `.js` file to `/features/step_definitions/`, or append your steps to the most appropriate existing file:

- `after_steps.js`: teardown (in After hook)
- `background_steps.js`: setup (in Background or Before hook)
- `email_steps.js`: outline-specific commands for `send_email.feature`.

A note on modularity

I did a few things to make my code as modular as possible:

- Split step definitions between setup (`/features/step_definitions/background_steps.js`), teardown (`features/step_definitions/after_steps.js`), and main step code `features/step_definitions/email_steps.js`)
- Packaged all locators into an easily-accessible Object for reuse (`/lib/locators.js`)
- Packaged reusable helper functions into their own file (`/lib/utils.js`)
- Separated dependency configuration logic from step logic (`/lib/chai.js`, `/lib/cucumber.js`, and `/lib/driver.js`)
- Saved user information into its own constants file (`/lib/users.js`)

All files delivered

Inside the GitHub repository (and this submission), there are the following folders and files

```

|--features
|----step_definitions (contains all step definitions)
|------after_steps.js (contains after hook steps)
|------background_steps.js (contains background steps)
|------email_steps (contains email steps)
|--images (contains all locally hosted images)
|---- howdy.jpg
|---- howdy.png
```

```
|---- howdy.tiff
|---- howdy.svg
|---- howdy.gif
|--lib (contains helper code)
|---- chai.js (Chai config)
|---- cucumber.js (Cucumber config)
|---- driver.js (Driver config)
|---- locators.js (Selenium locator constants)
|---- users.js (Credentials and Email constants)
|---- utils.js (Various helper functions)
|-- .eslintrc.js (JS Linter configuration)
|-- .gitignore (Ignored files on git)
|-- package.json (List of all packages and run scripts)
|-- package-lock.json (Lockfile for dependency versions)
|-- README.md (Markdown version of this report)
|-- final-report.pdf (PDF version of this report)
```

Links

- 📼 Screen recording: https://youtu.be/tqemFJleJKU
- 👩‍💻 Code repository: https://github.com/erickzhao/gmail-selenium-test