Creating a Strong Testing Framework with Google Translate Automation

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science
University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science, School of Engineering

Mehmet Faruk Yaylagul
Fall 2023

Advisor:
Briana Morrison, Department of Computer Science

Creating a Strong Testing Framework with Google Translate Automation

CS 4991 Capstone Report, 2023

Mehmet Faruk Yaylagul
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia
urj5rb@virginia.edu

ABSTRACT
Daily more than 500 million people use Google Translate to translate texts to a different language and many encounter problems during translation. To address this challenge, I decided to develop a framework that tests the functionality and accuracy of Google Translate. I wrote the framework in Java with Cucumber and Selenium testing tools. This testing framework shows the accuracy and usability of the website, including quality assurance for combinations of seven languages (English, Spanish, Dutch, French, Turkish, German, and Swedish) and functional buttons on the website. As a result, the tests, accessing the website, the buttons available on the website, and 14 language translation tests all passed with 100% accuracy with no issues. In the future, this testing framework will be a good example for QA testing engineers. It can also be used on different websites to test the website's usability, accuracy, and functionality.

1. INTRODUCTION

According to Google, over 600 million people visit GoogleTranslate.com daily. The website offers translation capabilities in 133 different languages and people can even translate their documents, images, or websites they visit. However, despite its widespread use and advanced technology, Google Translate may also not work functionally without facing errors. Considering the website's importance, it is significant to test the reliability of such important websites in an easier and faster way. Therefore, I designed an automation testing framework for GoogleTranslate.com based on its common usage and different functional combinations. This framework will be a good model for future QA engineers who want to work in the website automation field.

The main purpose of the project is to create a user-friendly, easy-to-implement, fast, and reliable automation testing framework for QA engineers working on similar web-based applications. Writing a good testing framework is a very important task for Quality Assurance Engineers. To automate a website, every QA tester must write a good testing framework in order to validate the usability and functionality of websites. I divided the parts of the framework to make it more understandable and easier to edit to be a good example to other QA engineers when they work on their projects or GoogleTrasnalte.com. In automation testing, the hard part is to build the framework so that developing automated tests will be easier. This framework will enable anyone using the framework to test Google Translate and easily implement their methods based on already found elements and built functions.

2. RELATED WORKS

Testers have the option of scripting or recording tests by using an automation tool. However, integrating this into a structured framework usually offers extra advantages. An effectively established test automation framework assists the testing team by achieving higher reusability of test components, developing scripts that are easily maintainable, and obtaining high-quality test automation scripts (Umar, 2019). Utilizing a framework for automated testing also increases test speed and efficiency, improves test accuracy, and reduces test maintenance costs, as well as lowering risks (Aebersold, 2019). Test automation frameworks provide a support foundation for a variety of automated software tests including Unit testing, Functional testing, Performance testing, etc. (Smartsheet, 2019).

Using any of the traditional scripting techniques (linear, data-driven, keyword-driven, etc.), the tester has to write down test case steps in detailed and complex sheet format. In the proposed framework,the algorithm can determine the appropriate keyword automatically according to the HTML input type (Hanna, 2018). A well-built testing framework helps engineers test their projects more easily and professionally because these tools help them track their testing coverages. A keyword may be SetText / SelectValue / OpenURL / ClickButton / etc. This eliminates the need to manually select the appropriate keyword that matches the HTML input control (Hanna, 2018).

3. DESIGN

The design Section will dive into the details of the software side of the project and how it is developed.

3.1 The Website Under Test

The URL of the website that I worked on is (https://translate.google.com) (Figure 1). This page gets the user input in any language and translates it to any requested language. This website is owned by Google company. All the codes are written in OpenJDK java version 20.0.1 in IntelliJ IDEA (latest version), using Junit, Selenium, and Cucumber.

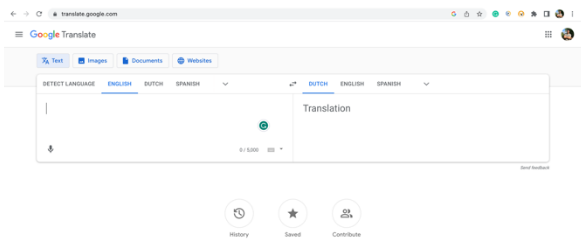
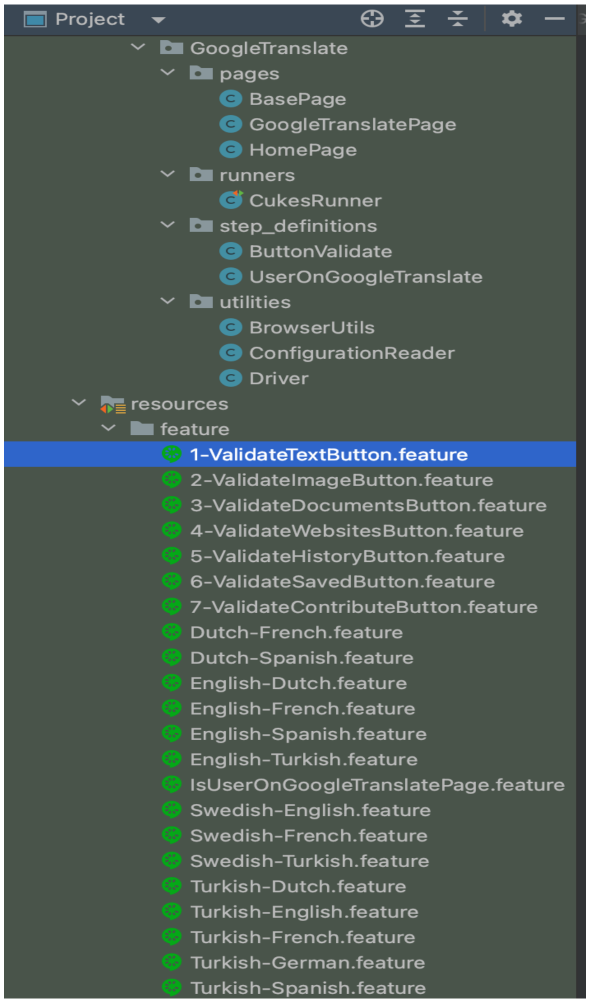


Figure 1: Main Page of GoogleTranslate.com

Google owns the SUT so I cannot access the source code. I inspected the website API and reached its HTML and CSS code so I would be able to test the website elements. For the functionality testing, I applied the Input Space Partition Coverage Criteria to test the website for the project, which helped me to test the different combinations of input variables to identify possible defects.

3.2. General View of the Framework

I stored the HTML of each element under test in the **GoogleTranslatePage** file. I created the testing features in the **feature** folder by following the input space partition method and created my test methods in the **step_definitions** folder. All the test definitions can easily be understood by checking these **.feature** files. This folder has two files: one **UserOnGoogleTranslate** which includes the test method if the website is reachable by the user, and the other one has **ButtonValidate** file which has all the other test methods related to the Google Translate website. The **utilities** folder has the helper functions and utilities that help to write my test methods (Figure 2).

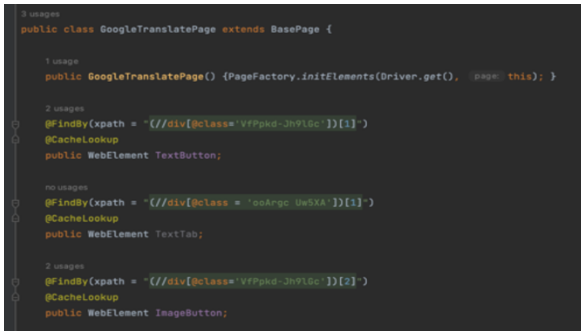


```
Project
  GoogleTranslate
    pages
      BasePage
      GoogleTranslatePage
      HomePage
    runners
      CukesRunner
    step_definitions
      ButtonValidate
      UserOnGoogleTranslate
    utilities
      BrowserUtils
      ConfigurationReader
      Driver
resources
  feature
    1-ValidateTextButton.feature
    2-ValidateImageButton.feature
    3-ValidateDocumentsButton.feature
    4-ValidateWebsitesButton.feature
    5-ValidateHistoryButton.feature
    6-ValidateSavedButton.feature
    7-ValidateContributeButton.feature
    Dutch-French.feature
    Dutch-Spanish.feature
    English-Dutch.feature
    English-French.feature
    English-Spanish.feature
    English-Turkish.feature
    IsUserOnGoogleTranslatePage.feature
    Swedish-English.feature
    Swedish-French.feature
    Swedish-Turkish.feature
    Turkish-Dutch.feature
    Turkish-English.feature
    Turkish-French.feature
    Turkish-German.feature
    Turkish-Spanish.feature
```

Figure 2: Files and Features in the Framework.

3.3. Finding The Right Elements to Test

The source code of the website is not accessible, so I inspected the HTML of the website and saved the element IDs, or classes in the **GoogleTranslatePage.java** file (Figure 3) to use them in test methods.



```java
3 usages
public class GoogleTranslatePage extends BasePage {

    1 usage
    public GoogleTranslatePage() {PageFactory.initElements(Driver.get(),  page: this); }

    2 usages
    @FindBy(xpath = "(//div[@class='VfPpkd-Jh9lGc'])[1]")
    @CacheLookup
    public WebElement TextButton;

    no usages
    @FindBy(xpath = "(//div[@class = 'ooArgc Uw5XA'])[1]")
    @CacheLookup
    public WebElement TextTab;

    2 usages
    @FindBy(xpath = "(//div[@class='VfPpkd-Jh9lGc'])[2]")
    @CacheLookup
    public WebElement ImageButton;
```

Figure 3: All the Elements Saved in GoogleTranslatePage.java
Storing the crucial elements in a different file helps the developer to automate the website more easily.

The biggest problem in website testing without a source code is finding the correct xpath of the element. Some indexes in HTML and CSS are dynamic and they change when the page is refreshed. Therefore, choosing the right class or ID of the element is crucial in automating a website. There can be a xpath corresponding to more than a hundred elements located on the website (Figure 4).

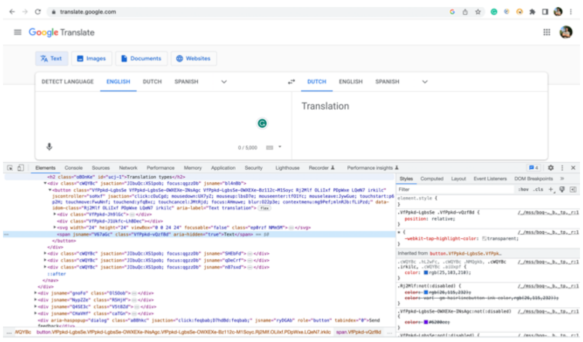


Figure 4: Xpaths, IDs, and Classes of the Elements in the HTML of the Website

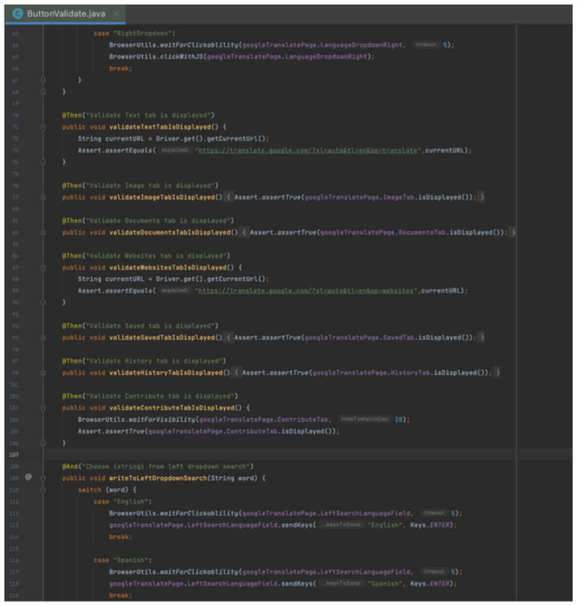
3.4. Testing Coverage Technique and Implementation

I mainly focused on input space partition coverage criteria when I developed the framework. I divided the input domain of a program into subsets, based on certain characteristics and properties of the website. The goal of input space partitioning was to identify representative test cases that test different combinations of inputs.

Google Translate gives different outputs based on the language model the user chooses so I tested if the website was accessible by the user and then checked the main buttons on the website if they were clickable and worked correctly. These buttons are text, images, documents, websites, history, saved, contribute, right language dropdown menu, and the left language dropdown menu. I stored their xpaths and classes in the page file as well (Figure 3).

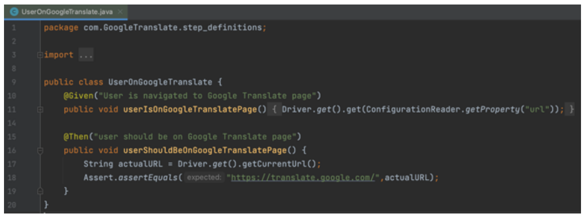
After I made sure the buttons worked, I tested if the translation worked correctly by checking different combinations of 7 languages (English, Spanish, Dutch, French, German, Turkish, Swedish).

I wrote the test methods for every step in **UserOnGoogleTranslate.java** and **ButtonValidate.java** files (Figure 5.1, 5.2). If the assert methods implemented in the Java files return False, the related step cannot pass so the test scenario fails.



```
ButtonValidate.java
            case "RightDropdown":
                BrowserUtils.waitForClickablility(googleTranslatePage.LanguageDropdownRight,  timeout: 5);
                BrowserUtils.clickWithJS(googleTranslatePage.LanguageDropdownRight);
                break;
        }
    }

    @Then("Validate Text tab is displayed")
    public void validateTextTabIsDisplayed() {
        String currentURL = Driver.get().getCurrentUrl();
        Assert.assertEquals( expected: "https://translate.google.com/?sl=auto&tl=en&op=translate",currentURL);
    }

    @Then("Validate Image tab is displayed")
    public void validateImageTabIsDisplayed() { Assert.assertTrue(googleTranslatePage.ImageTab.isDisplayed()); }

    @Then("Validate Documents tab is displayed")
    public void validateDocumentsTabIsDisplayed() { Assert.assertTrue(googleTranslatePage.DocumentsTab.isDisplayed()); }

    @Then("Validate Websites tab is displayed")
    public void validateWebsitesTabIsDisplayed() {
        String currentURL = Driver.get().getCurrentUrl();
        Assert.assertEquals( expected: "https://translate.google.com/?sl=auto&tl=en&op=websites",currentURL);
    }

    @Then("Validate Saved tab is displayed")
    public void validateSavedTabIsDisplayed() { Assert.assertTrue(googleTranslatePage.SavedTab.isDisplayed()); }

    @Then("Validate History tab is displayed")
    public void validateHistoryTabIsDisplayed() { Assert.assertTrue(googleTranslatePage.HistoryTab.isDisplayed()); }

    @Then("Validate Contribute tab is displayed")
    public void validateContributeTabIsDisplayed() {
        BrowserUtils.waitForVisibility(googleTranslatePage.ContributeTab,  timeToWaitInSec: 20);
        Assert.assertTrue(googleTranslatePage.ContributeTab.isDisplayed());
    }

    @And("Choose {string} from left dropdown search")
    public void writeToLeftDropdownSearch(String word) {
        switch (word) {
            case "English":
                BrowserUtils.waitForClickablility(googleTranslatePage.LeftSearchLanguageField,  timeout: 5);
                googleTranslatePage.LeftSearchLanguageField.sendKeys( ...keysToSend: "English", Keys.ENTER);
                break;

            case "Spanish":
                BrowserUtils.waitForClickablility(googleTranslatePage.LeftSearchLanguageField,  timeout: 5);
                googleTranslatePage.LeftSearchLanguageField.sendKeys( ...keysToSend: "Spanish", Keys.ENTER);
                break;
```

Figure 5.1 ButtonValidate.java

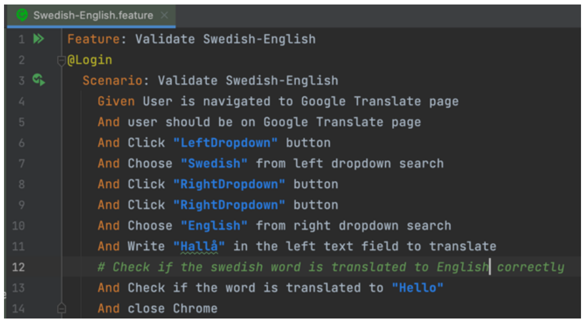


```
UserOnGoogleTranslate.java
package com.GoogleTranslate.step_definitions;

import ...

public class UserOnGoogleTranslate {
    @Given("User is navigated to Google Translate page")
    public void userIsOnGoogleTranslatePage() { Driver.get().get(ConfigurationReader.getProperty("url")); }

    @Then("user should be on Google Translate page")
    public void userShouldBeOnGoogleTranslatePage() {
        String actualURL = Driver.get().getCurrentUrl();
        Assert.assertEquals( expected: "https://translate.google.com/",actualURL);
    }
}
```

Figure 5.2: Test Methods Implemented in UserOnGoogleTranslate.java and ButtonValidate.java Files

3.4. Testing Scenarios

The good side of writing a framework with Cucumber is to create scenarios in separate files. This makes the framework tidier and easier for the user to understand. When encountering an error, one can easily see which method or test case fails ButtonValidate.java in the scenario. I developed 22 different automation testing scenarios, but every scenario checks more than one functionality of the website. For example, to test the Swedish-English translation, we have to ensure that the left and right language dropdowns work well. The text field should also pop up correctly to write the text that we want to translate so we can compare the original and translated texts correctly (Figure 6).



```
Swedish-English.feature
Feature: Validate Swedish-English
@Login
  Scenario: Validate Swedish-English
    Given User is navigated to Google Translate page
    And user should be on Google Translate page
    And Click "LeftDropdown" button
    And Choose "Swedish" from left dropdown search
    And Click "RightDropdown" button
    And Click "RightDropdown" button
    And Choose "English" from right dropdown search
    And Write "Hallå" in the left text field to translate
    # Check if the swedish word is translated to English correctly
    And Check if the word is translated to "Hello"
    And close Chrome
```

Figure 6: Test Scenario for Swedish-English Translation

4. RESULTS

The framework, I built in Java with Cucumber and Selenium, and effectively tested the functionality and accuracy across seven languages. The testing covered various aspects of the website, including its user interface elements and translation accuracy. Remarkably, all tests, including website accessibility, button functionality, and language translation accuracy, passed with 100% success. This outcome demonstrates the framework's effectiveness in ensuring the quality and reliability of Google Translates. The test scenarios are demonstrated for future documentation. The framework works accurately and it is ready to be adapted to test different web-based applications.

5. CONCLUSION

This project demonstrates how a good testing framework should be implemented. I aimed to design a professional automation framework for QA engineers so they can get a reference from this project when they design their own frameworks. The structure of the framework can be easily applied to different web-based applications just by changing the element IDs when testing similar functionalities. Using such a framework will also make QA engineers' jobs easier when writing testing reports.

6. FUTURE WORK

This framework can be shown in testing classes for education purposes in the future. The main issue that students encounter in testing classes is how to use selenium and Junit tests in a professional work environment when they have to test web-based applications. Web-based applications might have a lot of functionalities and trying to test them without a well-built framework will be complicated and hard. Showing this example framework to students, and QA engineers will be beneficial for their professional work.

7. ACKNOWLEDGMENTS

I would like to extend my sincere gratitude to Professor Upsorn Praphamontripong, Professor Rosanne Vrugtman, and Professor Briana Morrison for their invaluable support and guidance throughout this project.

REFERENCES

[1] Umar, M. A., & Zhanfang, C. (2019). A study of automated software testing: Automation tools and frameworks. International Journal of Computer Science Engineering (IJCSE), 8(6).

[2] Aebersold, K. Test automation frameworks. SmartBear. Retrieved August 26, 2019, from https://smartbear.com/learn/automated-testing/testautomation-frameworks/

[3] Smartsheet. A guide to automation frameworks. Retrieved August 30, 2019, from https://www.smartsheet.com/test-automation-frameworkssoftware

[4] Hanna, M., Aboutabl, A. E., & Mostafa, M.-S. M. (2018). Automated software testing framework for web applications. International Journal of Applied Engineering Research, 13(11), 9758-9767.