

Project #3 Graphical User Interface Testing Project

a) Description of the application

For this project, I created a GUI written in Python for a term life insurance shopping scenario. The user navigates to the landing page, where it prompts to enter basic information such as age, gender, and duration of the term. Once user clicks Next, it navigates to the next page where user chooses a coverage amount, whether he/she is shopping for her own life insurance, and the beneficiary's name. User can go back to modify user input on page 1 or proceeds to next page by clicking Back or Next. On page 3, user continues to enter information such as past tobacco use and current health status. User then submits these information for term life insurance quote, which is not part of this GUI. Source codes are provided separately.

Below describes types of elements used on each page in version 1.

Version 1:

Page 1.

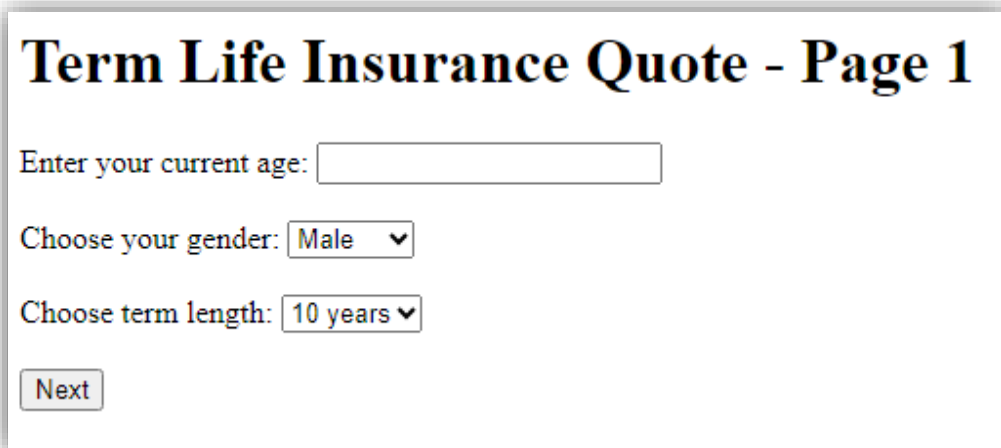
Label: Term Life Insurance Quote – Page 1

Age = Scrollbar, min=0, increment by 1

Gender = Dropdown list (Male, Female)

Term Length = Dropdown list (10,15,20,30)

Next button to proceed to page 2



Term Life Insurance Quote - Page 1

Enter your current age:

Choose your gender:

Choose term length:

Page 2.

Label: Term Life Insurance Quote – Page 2

Coverage amount: Slider, min=100,000 max = 1,000,000


Radio Button (yes or no)

Beneficiary name: Text box

Next button to proceed to page 3

Back button to go back to page 2

Term Life Insurance Quote - Page 2

Choose coverage amount: 
100000

Are you applying to insure your own life? ☐ Yes ☐ No

Enter your beneficiary's name:

Next

Back

Page 3.

Label: Term Life Insurance Quote – Page 3

Other insurance: Radio button (yes or no)

Admitted to: Dropdown list

Past tobacco use: Radio button (yes or no)

Submit button

Term Life Insurance Quote - Page 3

Do you currently have any life insurance in force? ☐ Yes ☐ No

Are you currently admitted to:

In the past 12 months, have you used tobacco items? ☐ Yes ☐ No

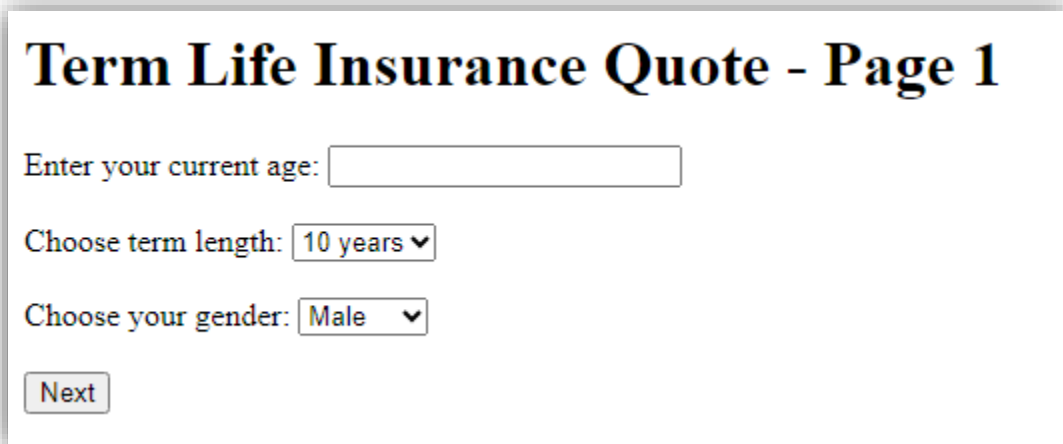
Submit

In version 2 of GUI, the basic GUI elements remain the same. However, locations of some of the design elements and the flow of pages changed. Below shows screenshots from version 2 and description of changes made from version 1.

Version 2:

Page 1:

Change 1. Locations of Gender and Term length lists are switched.



Term Life Insurance Quote - Page 1

Enter your current age:

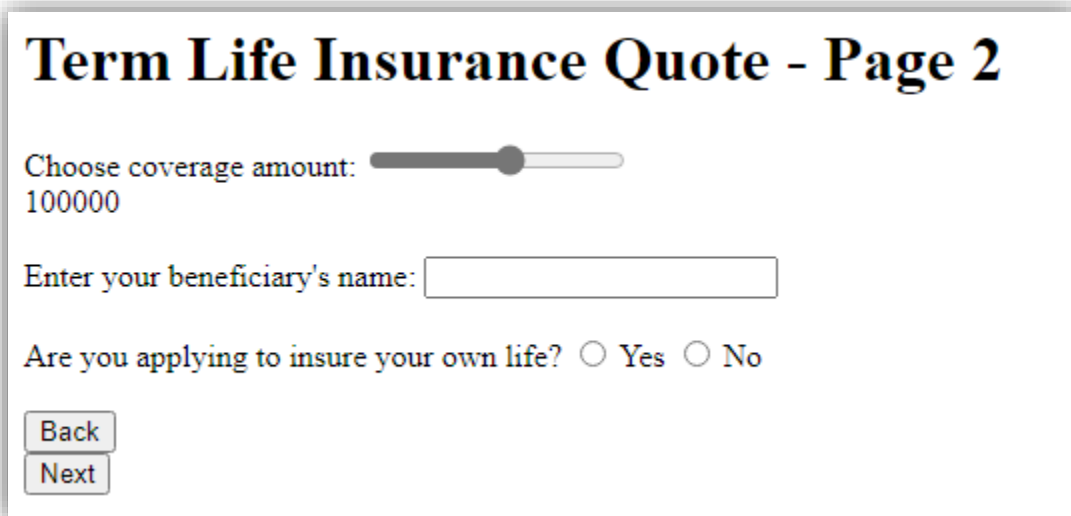
Choose term length:

Choose your gender:

Page 2:

Change 2. Locations of text box and radio buttons are switched.

Change 3. The placement of "Back" and "Next" buttons are switched.



Term Life Insurance Quote - Page 2

Choose coverage amount:
100000

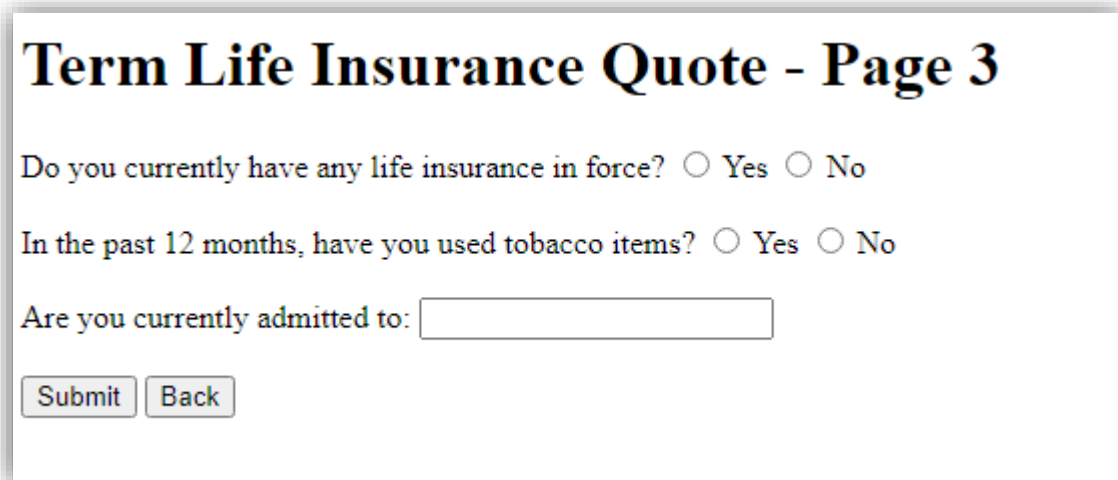
Enter your beneficiary's name:

Are you applying to insure your own life? ☐ Yes ☐ No

Page 3:

Change 4. Locations of list and radio buttons are switched.

Change 5: Added Back button to go back to page 2.



Term Life Insurance Quote - Page 3

Do you currently have any life insurance in force? ☐ Yes ☐ No

In the past 12 months, have you used tobacco items? ☐ Yes ☐ No

Are you currently admitted to:

b) Description of GUI testing tool

I chose Selenium to test various test cases of these two versions of GUI.

Features:

- **Browser Compatibility:** Selenium supports major web browsers like Chrome, Firefox, Safari, Internet Explorer, and Edge, allowing cross-browser testing of web applications.
- **Multi-Platform Support:** Selenium can be used on different operating systems such as Windows, macOS, and Linux, making it highly versatile.
- **Language Support:** Selenium supports multiple programming languages, enabling testers to write automation scripts in their preferred language.
- **Powerful APIs:** Selenium provides a rich set of APIs to interact with web elements, handle dynamic content, perform validations, and extract data from web pages.
- **Test Framework Integration:** Selenium can be integrated with various testing frameworks like TestNG, JUnit, and NUnit, enhancing test management and reporting capabilities.
- **Parallel Execution:** Selenium supports parallel test execution, allowing for faster testing cycles and efficient resource utilization.
- **Extensibility:** Selenium can be extended with additional libraries and plugins to enhance its capabilities and integrate with other tools.
- **Cross-Domain Testing:** Selenium supports testing across different domains and handles security restrictions imposed by browsers.

Scope and Area of Usage:

- **Functional Testing:** Selenium is primarily used for functional testing of web applications, ensuring that the application behaves as expected.
- **Regression Testing:** Selenium facilitates automated regression testing, allowing testers to quickly validate that new changes or updates do not introduce unexpected issues.
- **Cross-Browser Testing:** Selenium helps ensure consistent behavior across different browsers and versions, verifying that web applications work correctly on various platforms.
- **Data-Driven Testing:** Selenium supports data-driven testing, enabling testers to execute tests with different sets of test data to validate application behavior under various scenarios.
- **Integration Testing:** Selenium can be integrated with other tools and frameworks to perform integration testing of web applications.
- **Continuous Integration:** Selenium can be integrated into continuous integration and delivery pipelines, enabling automated testing as part of the software development lifecycle.
- **Web Scraping:** Selenium can be utilized for web scraping tasks, extracting data from websites for various purposes.

c) **Description of the test cases**

Below describes various test cases developed by me to test and validate the functionality of GUIs. For version 1 and version 2 test cases are identical except “Back” button test case from page 3 since “Back” button is added to the version 2 only. In terms of coverage, page 1 test cases cover close to 100% of the page functionality while page 2 and page 3 test cases do not cover 100%, missing a test case such as testing back button on page 2.

Version 1:

A total of 12 test cases are tested

Page 1 Test Cases

1. Verify the page title
2. Test for a valid age (a positive number, 30) input
3. Test for an invalid age (a negative number, -5) input
4. Test whether a selected gender is in the list of options (female)
5. Test whether a selected term option is in the list of options (40year)
6. Test Next button to see it proceeds to page 2

Page 2 Test Cases

1. Test for a valid slider value (300,000)
2. Test for selecting a radio button option (Yes)
3. Test Back button to verify it goes back to page 2
4. Test Next button to verify it proceeds to page 3

Page 3 Test Cases

1. Test for selecting a radio button option for tobacco use (No)
2. Test for selecting a radio button option for other insurance products (Yes)

Version 2:

In version 2, all 12 test cases from version 1 are reused and re-tested. Additionally, I added one more test case to test the functionality of “Back” button added on page 3.

Additional test case on page 3:

- Test Back button to verify it goes back to page 2

d) Explanation of the test results

To generate test results, I used unittest and htmlTestRunner along with Selenium. Below shows screenshots of test results.

Version 1:

Page 1.

It shows that unit test passed 4/6 test cases and returned 1 failed and 1 error.

Unittest Results

Start Time: 2023-06-10 10:59:00

Duration: 60.89 s

Summary: Total: 6, Pass: 4, Fail: 1, Error: 1

__main__.TestCasePage1	Status
test_page1_age_input_1	Pass
test_page1_age_input_2	Fail View
test_page1_gender_input	Pass
test_page1_page2_button_test	Pass
test_page1_term_input	Error View
test_page1_title	Pass

Total: 6, Pass: 4, Fail: 1, Error: 1 -- Duration: 60.89 s

Looking at the description of the failed test case, we see that a negative age input (-5) was not allowed as intended.

test_page1_age_input_2	Fail	Hide
------------------------	------	----------------------

AssertionError: -5 not greater than or equal to 0 : Negative age input is not allowed

Traceback (most recent call last): File "test_script.py", line 55, in test_page1_age_input_2 self.assertGreaterEqual(entered_age, 0, "Negative age input is not allowed") AssertionError: -5 not greater than or equal to 0 : Negative age input is not allowed

Similarly, looking at the description of term test case, 40 year term option was not in the provided list of values (10,15,20,30), so it returned an error.

test_page1_term_input Error Hide

NoSuchElementException: Message: Could not locate element with visible text: 40 years; For documentation on this error, please visit: <https://www.selenium.dev/documentation/webdriver/troubleshooting/errors#no-such-element-exception>

Traceback (most recent call last): File "test_script.py", line 99, in test_page1_term_input dropdown.select_by_visible_text(selected_term) File "C:\Users\12132\anaconda\lib\site-packages\selenium\webdriver\support\select.py", line 137, in select_by_visible_text raise NoSuchElementException(f"Could not locate element with visible text: {text}") selenium.common.exceptions.NoSuchElementException: Message: Could not locate element with visible text: 40 years; For documentation on this error, please visit: <https://www.selenium.dev/documentation/webdriver/troubleshooting/errors#no-such-element-exception>

Page 2.

Page 2 test cases are relatively simple, checking for a valid value along a slider, radio button, and back/next button to move to the next page. The below report shows that it passed all 4 test cases.

Unittest Results

Start Time: 2023-06-10 21:43:41

Duration: 72.32 s

Summary: Total: 4, Pass: 4

__main__.TestCasePage2		Status
test_page2_page1_back_button_test	test_page2_page1_back_button_test	Pass Hide
Successfully navigated back to page 1.		
test_page2_page3_next_button_test	test_page2_page3_next_button_test	Pass Hide
Successfully navigated to page 3.		
test_page2_radio_button_yes	test_page2_radio_button_yes	Pass
test_page2_slider_value	test_page2_slider_value	Pass
Total: 4, Pass: 4 -- Duration: 72.32 s		

Page 3.

Below shows that two test cases for Page 3 have passed.

Unittest Results

Start Time: 2023-06-10 22:04:43

Duration: 14.40 s

Summary: Total: 2, Pass: 2

__main__.TestCasePage3		Status
test_page3_insurance_yes	test_page3_insurance_yes	Pass
test_page3_tabacco_no	test_page3_tabacco_no	Pass
Total: 2, Pass: 2 -- Duration: 14.40 s		

Version 2:

Page 1

For page1 of version 2 tests, I reused the page 1 test cases from version 1 as the only change made in version 2 is locational changes of the existing lists (Gender and Term length). As seen below, version 2 test results are identical to those of version 1.

Unittest Results

Start Time: 2023-06-10 22:16:10

Duration: 55.90 s

Summary: Total: 6, Pass: 4, Fail: 1, Error: 1

__main__.TestCasePage1		Status
test_page1_age_input_1		Pass
test_page1_age_input_2		Fail Hide
AssertionError: -5 not greater than or equal to 0 : Negative age input is not allowed		
Traceback (most recent call last): File "page1_test.py", line 55, in test_page1_age_input_2 self.assertGreaterEqual(entered_age, 0, "Negative age input is not allowed") AssertionError: -5 not greater than or equal to 0 : Negative age input is not allowed		
test_page1_gender_input		Pass
test_page1_page2_button_test		Pass View
test_page1_term_input		Error Hide
NoSuchElementException: Message: Could not locate element with visible text: 40 years; For documentation on this error, please visit: https://www.selenium.dev/documentation/webdriver/troubleshooting/errors#no-such-element-exception		
Traceback (most recent call last): File "page1_test.py", line 89, in test_page1_term_input dropdown.select_by_visible_text(selected_term) File "C:\Users\12132\anaconda\lib\site-packages\selenium\webdriver\support\select.py", line 137, in select_by_visible_text raise NoSuchElementException(f"Could not locate element with visible text: {text}") selenium.common.exceptions.NoSuchElementException: Message: Could not locate element with visible text: 40 years; For documentation on this error, please visit: https://www.selenium.dev/documentation/webdriver/troubleshooting/errors#no-such-element-exception		
test_page1_title		Pass

Total: 6, Pass: 4, Fail: 1, Error: 1 -- Duration: 55.90 s

Page 2

Page 2 test results are also identical to those of version 1, as these test cases are reused.

Unittest Results

Start Time: 2023-06-10 22:17:31

Duration: 71.24 s

Summary: Total: 4, Pass: 4

__main__.TestCasePage2		Status
test_page2_page1_back_button_test		Pass Hide
Successfully navigated back to page 1.		
test_page2_page3_next_button_test		Pass Hide
Successfully navigated to page 3.		
test_page2_radio_button_yes		Pass
test_page2_slider_value		Pass

Total: 4, Pass: 4 -- Duration: 71.24 s

Page 3

For page 3 test cases of version 2, I added one additional test case to test the functionality of “Back” button as this button was newly added to version 2. Otherwise, the other test cases were reused.

Unittest Results

Start Time: 2023-06-10 22:20:18

Duration: 45.27 s

Summary: Total: 3, Pass: 3

__main__.TestCasePage3		Status
test_page3_back_button		Pass Hide
Successfully navigated back to page 2.		
test_page3_insurance_yes		Pass
test_page3_tabacco_no		Pass
Total: 3, Pass: 3 -- Duration: 45.27 s		

e) Assessment of the tool

Coverage

In this testing scenario, I mainly tested the functionality of two GUIs by simulating user interactions. I found that Selenium is an excellent testing tool to test for functionality. I have tested for other types of testing such as performance testing, in which Selenium may not be a suitable testing tool for that.

Reuse of test cases

I created various test case for version 1 and reused those test cases for version 2 along with a new test case specifically designed for test 2. So, I found that Selenium is a great tool for automated test cases that can be used across different versions of web applications. This make testing a lot easier and more consistent.

Test results produced

To generate test results and reports, I chose a tool “HtmlTestRunner”. I found the tool is useful generating a report that shows test case name, test results, fail/error logs, and duration. However, I don’t find the tool to be particularly user-friendly as it took me some time to figure out the functionality of tool and how to properly set up and code up the selenium test cases.

Usability

Overall, I found that Selenium is a very useful tool to test unit test cases I developed. It requires some learning in the beginning as developing test cases as well as generating html reports requires a fair amount of Python coding. As I developed a relatively simple GUI for both versions, coding part was not that complicated for me. However, I can see with more complex GUIs and more testing cases, both creating test cases and code conversion may require some development time. However, once test cases are created, those can easily be re-used in multiple versions, so I can see why Selenium is widely used for testing web applications.

Reference

<https://github.com/SeleniumHQ/>