As part of the QA Engineer tech test I was asked to prepare automated user acceptance tests for a frontend app. In this document I will provide insights into how I created them.

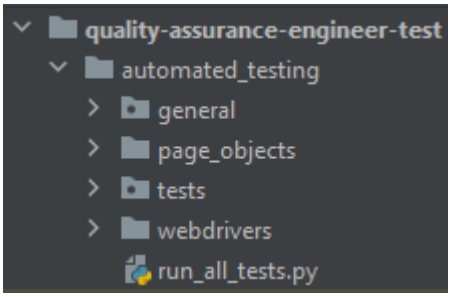
1. Technology used

I decided to create my tests using Python (3.10 version) programming language, Selenium WebDriver and PyTest frameworks. I prepared my tests to run on the Firefox browser.

2. Set up

Before I started working on my implementation I created my own branch - “qa_tech_test_julia_jaroszuk” where all of my changes/implementations will be added. Then I imported all the necessary libraries (pytest, selenium, pytest-depends). After checking my version of Firefox browser (95.0) I downloaded the corresponding webdriver which will be used when running tests.

Next I created a simple folder structure in which my tests will be kept:



Everything is contained inside the “automated_testing” directory.
Folders inside have following purposes:

- general - all common actions should be defined there
- page_objects - locators and common (oftenly repeated) actions for each element that will be used in tests
- tests - all files which contain tests will be stored there
- webdrivers - this folder contains webdrivers used to perform tests (important note: location of the webDrivers used needs to be added to the PATH system variable in order to run the tests)

The “run_all_tests.py” can be used to run all tests at once. It has the following content:

```
import pytest

# Url used for all test cases
URL = "localhost:8000"
# Browser used to run automated tests
browser_type = 'Firefox'
# Directory containing tests that should be run is defined
directory_with_tests = ['tests/']
# All files that start with "test" will be found in the given directory and all of them will be executed one by one
pytest.main(directory_with_tests)
```

When using PyCharm IDE, it is enough to simply select the ‘Run’ option for this file and the above code will be run which will prompt the execution of all tests found in the given directory. This is possible due to the PyTest framework. After deciding on the basics I moved on to creating the tests.

3. Page Objects

In order to test the application I need to be able to interact with its elements (buttons, texts etc). Those elements are used a lot through different tests so it is better to define them separately so that each test can reuse them. I divided the tested application into 4 sections: header, sidebar, created questions and new questions. For each section I created a separate class which contains locators for elements inside that section and actions that can be performed on them. I decided to divide the locators to have better control over the files and to make any maintenance easier. Also I created functionalities only for the actions that I knew I would use in tests.
All classes in this folder inherit from the Page class which contains a function that makes sure that all actions are performed on the already existing instance of the browser.

4. Browser actions

Before tests can be created I needed to establish connection to the browser and I needed to be able to control it. I decided on 3 functionalities:

- creating browser instance => automated_testing.general.browser_actions.setup_browser
- checking if there is an instance and accessing it => automated_testing.general.browser_actions.get_browser
- clearing and closing the browser => automated_testing.general.browser_actions.clear_browser

5. Tests

After considering the given application I decided on 4 areas for testing:

- initial layout - checking the initial state of the app (no input/action from the user)
- add question - checking if the functionality for adding a question works as expected
- created questions - checking behavior of existing questions i.e. sort functionality, if answers match questions, sidebar text gives correct number of questions
- delete questions - checking if removing functionality works as expected - no question visible and appropriate text is displayed and sidebar msg is also affected

Using pytest allows for better test management - I was able to use fixtures to dictate when certain actions should occur. For example I decided that the browser should not be restarted for each test but for each test class (test classes correspond to the 4 areas described above). Restarting of the browser restores the initial state of the app. Each test has only 1 assert so that if something fails it is easy to detect what happened. Most of the tests created are independent of each other so the order of execution does not matter. The only exception occurs in the TestRemoveQuestion class. In order to check the sidebar text, the initial question must be deleted which happens in test_remove_questions. I decided to leave this dependency and show how we can connect the tests so that the order of execution

aligns with the dependency (this is achievable due to pytest-depends package). However I could have removed that dependency by separating those two tests and using a fixture specific to this class which would have deleted the question. I showcased how this can be done in test_created_questions_functionality.py

In this assignment I decided to implement only happy path test cases and have the test data integrated with the code.

6. Execution of tests

Precondition for running the tests: application must be running

Screenshot of the summary of a complete test run: