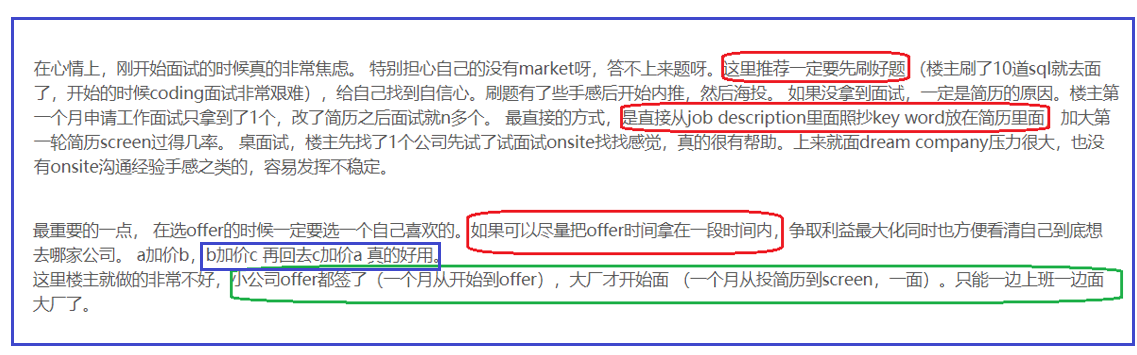
9/27/22



1. **Overall Structure**

Our Application provides connectivity test solution for mobile devices and it offers test management for tests, workflows, devices and analytics and reporting through UI and restAPIs.

We have a CI/CD pytest-based automation platform to run our QA test. Once a developer checks in new code in feature branch, automation test is triggered to run. We also run nightly regression test on our master branch.

We use selenium to run UI test. All API tests are run based on python request module.

Device is in prototype, with chipset running on host pcs.

By combining connectivity Benchmarking Solution with LP workflow generator, analysis and reporting. you can create an **end-to-end automated data processing environment** from the collection of measurement data to analytics and reporting.

1. Daily work routine
2. Review logs/test results of the nightly regression test.
3. Maintain pytest automation platform.
4. New feature verification test.

* Review design API/UI documents
* Create test cases/test scripts.
* Quickly verify basic functionalities manually.
* Performance test is done through python locust.

1. Debug/verification of bug fixes.

8/31/22, 9/6,

1. Overall
   1. Application Stack
      * 1. HTML/CSS/js/ajax (react maybe) + Java web application
        2. Backend (Tomcat) on Linux
        3. Testing mainly on Linux, support windows too
   2. Summary

Our application or test solution (web-based /desktop application) 🡪 connectivity testing. As SW test engineer, we focus on UI and API testing.

1. CI/CD
   1. Done on linux.
   2. Source/code and test code are in the same codebase, but in different folders.
   3. Figure out pipeline or not.
   4. Procedures.
      * 1. Linux Ubuntu LTS: 1.19
        2. Dev check-in feature-branch after unit-testing pass.
        3. CI/CD deploys app (.war file) to test server.
        4. Kick off API/UI QA testing. API and UI run separately. So basically, two Jenkins test jobs. UI first, followed by API test.
        5. Nightly Regression test.
           1. Run daily, testing on master-branch.

Cover cross-browser: Chrome, Firefox, (Microsoft edge)

* + - * 1. Run weekly, cover windows 10/11 OS.

1. API/UI functionality test.
   1. Python pytest-based automation platform.
2. API/UI performance test
   1. Python locust.
3. Detail BQ questions.
   1. **Most Challenges** 
      * 1. Struggling with adding general or more specific function for new feature testing.

Lots of time, writing a test function for individual feature without having a bigger picture that the test function could be re-used for other similar test scenario. Always wait until testing is done, and then trying to merge/consolidate similar functionalities, to put in fixture that will be shared by similar test scenario, high maintenance cost and lots of times, there is no time to even do that.

For functions that apply to multiple cases, make it specific means lots of copy/paste with minor changes, hard to maintain and if there is changes, it ends up change many relevant functions.

Make function in more general way, at the current time, takes much more time and effort to make it work and is error-prone since we have to consider test scenarios that are not possible now, but reserved for future. Consider overall structure and features, make necessary/relavant trade off.

Solution would be, first , try to understand the modules that the new feature could affect, and based on that writing a general function that will cover those cases first, while leaving room expandable future.

**Provide an example**

* + - 1. How to verify a search or sort feature in API

How to find a bug

How to write functions to do that

Can you sort based on all output parameters or just certain ones that defined by dev.

* + - 1. Manage conftest.py
         1. Conftest.py is used to keep fixture functions, which is automatically called by pytest. Over the time, conftest.py gets bigger and bigger as more fixture functions are added.
         2. One way to solve the problem is hierarchy conftest, for each main feature, which is of its own folder, creates a conftest and put all the feature relevant fixture function in this conftest.
         3. Another way, by researching. We can create a stand-alone folder called fixture\_test, and put all fixture functions under this folder.

Then add a plug-in to conftest.py, thus only need to maintain one simple conftest.py file.

* + - 1. Avoid dev says I can’t see it on my setup
         1. Issues: env is not the same; lib not the same version. Or not running the same script or calling the same function.
         2. Provide detailed steps to dev and makes it easier for them reproduce.
         3. Provide working script that dev can try without extra configuration.
         4. Set up test station in a ready stage, so dev can just run the script and easy to see the issue.
         5. Selenium: deal with issues due to staleness

[Selenium - stale element reference: element is not attached to the page](https://stackoverflow.com/questions/45002008/selenium-stale-element-reference-element-is-not-attached-to-the-page)

* + - * 1. Identify bugs is from UI or backend.

Use network or fiddle to capture to see if backend returns correct info or just UI related issues.

* + 1. UI/API: positive/negative and corner case testing and results verification.
       - 1. UI test

Focus on GUI .

Result is normally checked against page display.

* + - * 1. API test

Focus on :

1. Json results comparison (comprehensive)
2. Or key parameter output: comparison
3. Status code
4. Error message
   * 1. Pytest Auto-Framework
        1. Tools/utilities/libs
     2. Keep studying other company’s good API+