Basics

How do you deal with flaky tests?

Visualisation and Documentation

- By Visualising random test execution like executing the same test suite multiple times in same and different test environment,
- By changing the order of test case execution logically,
- By performing parallel testing
- Document the details of visualisation like date /day /time/ environment /duration /testcase failure count. It depicts whether flaky tests are increasing or decreasing over the time

Quarantine flaky test

Quarantine flaky test case and action to fix, so as to not disturb entire test suite to rerun. It saves time of executing non-flaky test case

Fix the issue

Try to fix it based on the failure reason, some of them are given below:

- Clean-up the state, data and cache, and start the analysis fresh
- Pay attention to dynamic waits and timeout, avoid static wait, unless necessary
- Avoid concurrency issues like dead lock
- Check for memory leakage, System clock

Let's suppose there is a test pipeline taking about 1 hour to finish, what would you do to decrease the time of it?

- Optimize the test suite to remove unwanted testcase or which has been covered already.
- Every Testcase should have single focus.
- Increase parallel execution
- Avoid use of static wait and always use dynamic wait
- Do not repeat yourself, split your build process up so that it does not do unnecessary work building, packaging and deploying code that hasn’t changed.
- Cache the modules - downloading modules at build time takes a significant portion of a build.
- Have the portfolio with maximum test coverage in API and less in GUI (Like straight through Scenario, popup).

Imagine you have the possibility to ask software engineers to develop tools for you that will increase your productivity as full-stack QA, please describe to them your requirements

- Develop the application with standard locators (avoid dynamically changeable ID property) which will ensure the test script developed to be reliable. It reduces the Maintenance cost associated with QA Automation
- In Agile environment, during daily scrum while developing code for new functionality/initiatives, if developers can show testers the code base and explain how the development team constructs new code as well as how their team pinpoints code defects and how they fix them. Testers can show developers their test cases and explain the reasoning that goes into their testing techniques. A tester that understands the code base can anticipate potential issues and code breakage, so stronger test cases are created. A developer who understands the testing suite can refine code construction so that it’s built to pass in testing.

Test Case Challenge

1. Test Plan

   Below test case is similar for 3 features – Android, ios and web application

   Pre-requisite:

   Application should be installed for mobile (Android and ios)

   Test Data:

   Register user credentials

   Test Plan:

Test Case	Step	Description
Verify login with valid credentials	1	Launch app
	2	Enter valid username & password, then click login
	3	Main Screen should be launched
Verify login with valid username and invalid password	1	Launch app
	2	Enter valid username & invalid password, then click login
	3	Login screen should throw error message, main Screen should not be launched
Verify login with invalid username and valid password	1	Launch app
	2	Enter invalid username & valid password, then click login
	3	Login screen should throw error message, main Screen should not be launched
Verify login with valid username and password less than required length	1	Launch app
	2	Enter valid username & password less than required length, then click login
	3	Login screen should throw error message, main Screen should not be launched

2. This will be tested at System Integration phase

Automation Test Challenge

- Test 1 and 2 are implemented and available in the following GitHub repo: https://github.com/sivamano02/mytheresa_website.git

- Test 3 – Extracting pull request using Ruby in the following repo: https://github.com/appwrite/appwrite
  - Install ruby
  - Create a "gemspec" file and provide the relevant details like github page, license used, dependencies etc.,
  - Create a "gemfile" and specify the source & gemspec
  - Open Terminal and execute command "gem install appwrite" .This installs the `epr` executable
  - Provide Configurations like token, default service etc., if required
  - Now to export the open pull request to csv format file, run the below command in terminal "epr -x pr -c 'appwrite' padrino/padrino-framework > pr.csv"