QA challenge

Santiago Pérez

August 15, 2019

qa-challenge

Abstract

For this test environment we use Carina-framework, a build in framework that contains selenium and its components, a page object model that implements the user actions, and we use circleci to create the pipeline, we use docker hub as well to store the generated images from our builds. All tests are executed against a chrome driver and using java language for test automation scripts.

0.1 Repositories

We created a git repo in which the code is stored and receives commits from dev team, we find the repo here:

https://github.com/sanpsicro/qa-automation-challenge.git
in the repo its configured a .circleci/config.yml file that links our repo with circleci for the pipelne, also we have a docker-compose.yml with docker images for our execution environment, in particular pulls the latest built docker image found in our docker hub and pulls selenium and the browser drivers images as well

we also use this repos
1.- https://github.com/sanpsicro/qa-automation-challenge.git
2.- https:// github.com/sanpsicro/automation$_carina.git$

3.- sanpsicro/test-image-2:

0.2 CI setup

The setup for the CI is done via the following components.

1.- the repo where the code of te app lives.
https://github.com/sanpsicro/qa-automation-challenge.git

2.- the repo where the code of the tests lives.
https:// github.com/sanpsicro/automation$_carina.git$

3.- the dockerhub repo where the built images are stored

sanpsicro/test-image-2:

We follow the app project in circleci and every commit triggers via te config.yml file a build of the app, then we wait for the new build to be pushed to dockerhub for its later use in the test stage and then we execute the testcases in a serial way.

0.3 Testing framework setup

The testing framework is build with a page object model and we have three test cases in one class named demo_test.java under the test.java.com.test.qa_challenge folder this class takes the configuration from the page model under

java.com.test.qa_challenge.home and java.com.test.qa_challenge.search

we define the actions in the above classes and we execute the actions in demo_test.java

Then under main.resources we setup the execution configuration in _config.properties where we set the url of the app, the selenium hub url , the drivers and the tests to be executed

0.4 Workflow

The workflow is defined in the config.yml file in the circleci configuration, it looks like the diagram below

$$commit \triangleright build \triangleright dockerize \triangleright environment \triangleright test$$

we use a two step workflow that executes the jobs corresponding to the build that also include the pushed of the dockerized image and then the test job is executed that prepares the environment for the tests to follow. The file of the workflow is configured in the config.yml in circleci.

the app code and the tests code stays in separated repos, given this structure its possible to maintain both codes in isolation,the ci flow will always be updated in both repos.