7. Run a CodeceptJS Test in GitLab´s Continuous Integration (CI) Environment

OK, lets take some time to recap what we have done so far: Setup CodeceptJS and all it´s dependencies, planned & created our first automated end to end test, run it locally, run it on the cloud in a different browser then Google Chrome, debugged a cross browser test automation issue and fixed that. Also we got a great understanding about all moving parts and where to look when things go wrong and how to update all of part.

One quick todo for you: Please change back the property `helpers.WebDriver.browser` to "chrome" in the `codecept.conf.js` file and remove the BrowserStack configuration from the previous chapters.

So what is the next piece missing on our "practical e2e test course" puzzle?

In my opinion, it is Continuous Integration.

The automated e2e tests should be prepared to run on every code change, automatically. Let us not loose any time and learn on a real example right away how to do that, are you ready to follow along?

7.1 GitLab CI & Git Repo Setup

Register for a free GitLab account at https://gitlab.com/users/sign_in and log in.

Then create a new project. After that, go to https://gitlab.com/profile/keys and add your public ssh key to be able to push to the freshly created git repo on gitlab.com.

In your codeceptjs project folder create a new file called `.gitignore` with the following contents:

```
# .gitignore

node_modules
output
```

Then execute the following commands to setup a local git repo, link it to the remote repo and push your code into it.

```
git init
git remote add origin git@gitlab.com:YOUR_GITLAB_USERNAME/YOUR_PROJECT_SLUG.git
git add .
git commit -m "Initial commit"
git push -u origin master
```

That´s it for the setup, let us head over to the next step.

7.2 Setup CI Chrome Test Run

GitLab CI is configured through a `.gitlab-ci.yml` file which just need to be added to the repo and pushed into the remote repo, GitLab will pick it up automatically and run the configured jobs.

Create a new file in the root of the repo called `.gitlab-ci.yml` with the following contents:

```
image: node:10-alpine

stages:
  - chrome-ci

variables:
  SCREEN_WIDTH: 1920
```

```
    SCREEN_HEIGHT: 1080

  services:
    - name: selenium/standalone-chrome:latest

  e2e-tests-chrome:
    stage: chrome-ci
    before_script:
      - npm ci
    script:
      - npx codeceptjs run --steps --profile=chrome-ci
    after_script:
      - cp -r output/ ci_artifacts/
    artifacts:
      name: "$CI_JOB_STAGE-$CI_COMMIT_REF_NAME"
      paths:
        - ci_artifacts/
      when: always
```

Also we need to adapt the `codecept.conf.js` a bit to rewire the selenium host when the "chrome-ci" profile is provided, to do so refactor your CodeceptJS config file to the following example:

```
let config = {
  tests: "./*_test.js",
  output: "./output",
  helpers: {
    WebDriver: {
      url: "http://the-internet.herokuapp.com",
      browser: "chrome",
      windowSize: "maximize"
    }
  },
  include: {
    I: "./steps_file.js"
  },
  bootstrap: null,
  mocha: {},
  name: "my-auto-e2e-tests",
};

if (process.profile === "chrome-ci") {
  config.helpers.WebDriver.host = process.env.SELENIUM_STANDALONE_CHROME_PORT_4444_TCP_ADDR;
  config.helpers.WebDriver.protocol = "http";
  config.helpers.WebDriver.port = 4444;
}

exports.config = config;
```

Basically we changed the following things:

1. Store the config in a variable to make it possible to override some parts of it when the "chrome-ci" profile is used.
2. Actually override the config when the profile is provided when running CodeceptJS
3. Finally export the config object

We are good to go, CI integration is almost there, how great is that?

You just need to run
`git add .gitlab-ci.yml codecept.conf.js && git commit -m "Add chrome ci test job." && git push`
and head over to https://gitlab.com/YOUR_GITLAB_USERNAME/YOUR_PROJECT_SLUG/-/jobs to watch your CodeceptJS

test running automatically in GitLab CI.

7.3 GitLab CI Artifacts

A nice bonus for you: The GitLab CI configuration already serves so called "artifacts" for you, as you can see in the configuration we first copy the "output" folder to a different location as GitLab ignores all git ignored paths and won´t serve them as artifacts and then using that temporary location as base for the artifacts.

As an example head over to https://gitlab.com/paulvincent/codeceptjs-e2e-testing/-/jobs/203393041/artifacts/browse and you can open the screenshot generated by the last step test `I.saveScreenshot("editor_test.png");`, executed in GitLabs free CI system.