Selenium Page Object Model using Python

Please find a ‘mod.zip’ file attached to the email. I will try to walk you through my idea of Selenium modeling that I’ve created in the past few days. Please note that it’s not finished or perfect and I want to use it to share my vision about this matter.

There are 3 levels of code in my POM:

1. High level - contains highly reusable code

   action.py: contains everything you need to make an action (like click, text input, etc.). Functions already contain Selenium code, try/catch statements, logs/test output file, screen shot, etc. There are more options that could be added depending on project needs, like implicit wait or retry the action if it failed the first time, which could be helpful if application has a dynamic parts.
   Instead of doing everything above in a test, test engineers reuse the high level function.

   Example (from test car01.py) for single click:
   fns.actions.click(**‘’**, **"//input[@data-qa-id='addToCart']",** ‘**Add to Cart’**, **‘y’**)

   log.py: when you run the test, the test_name.html file will be created with recording of each step, checkpoint validation (actual and expected values) and screenshots. Functions are embedded into actions.py so test engineers don't have to worry about it.

   You can find an example in the ‘mod/tstlog/car01.html’ file. Currently it records the step name passing in the action function and screenshot value (if ‘y’ it will add a screenshot). It will aways take a screenshot if validation (assertion) fails.

   Potentially, I would want to add some short traces option into the html file when validation or action fails. There is also an option to link all the test output to a QA web page (html output, log file and output file for comparison) so most of debugging could be done from one place.

   settings.py: place for all Selenium preferences, grid implementation, etc. This part is yet to be developed in my example, but I will continue to work on it in my free time.

2. Middle level - common reusable functions

   cmn.py - store functions that could be accessed from different places and reused by different members of a team. Example: perform log in

   xpathv.py - stores all commonly used locators (xpath or css) in a dictionary. To access a locator user needs to simply insert a key value (name) to the action function. It's very handy to store all common locators at the mid level as it helps a lot when changes made at software/design. Instead of fixing the locators test-by-test, it could be fixed much faster and in one file.

3. Lower lever - tests that use functions from the top levels. Please see an example car01.py.