SDET Bootcamp May 2022

Individual Project

Pick a site

Pick any site you like, or one from the following list:

- Nab or Commbank home loan calculators (any of deposit/repayments/loan amount calculators)
    - https://www.nab.com.au/personal/home-loans/calculators/loan-repayments-calculator
- https://www.saucedemo.com/
    - Automation training site that changes selector difficulty based on what login you use
- https://demo.applitools.com/
- http://automationpractice.com/index.php
    - A mock shopfront training site
- https://react-shopping-cart-67954.firebaseapp.com/
    - A mock shopfront training site
- https://www.globalsqa.com/angularJs-protractor/BankingProject/#/login
    - A simple mock banking site
- Bunnings
    - https://www.bunnings.com.au/
    - This is a harder choice than the others on this list, but still very doable

(If you pick a site not on the list, ask us about it first.)

Design test cases

In an excel spreadsheet, word document, text file, etc. write 5-6 test cases for the site of your choice. (You don't need to implement all these test cases in your solution).

Create an automation solution

Set up a Java automation solution (separate to the Jupiter Toys solution we've been working on) for your project. Create a new repository on GitHub to store this solution.

Writing tests

Aim to complete 2-3 test cases from the test cases you wrote earlier. Quantity of test cases doesn't matter, we're looking for quality.

What you'll be evaluated on

At the end, we'll be evaluating your work based on what you've learned so far and whether you can apply the principles from the past two weeks. If your project ends up being a lot more difficult than you expected and you can't do as much as you hoped for, don't worry - if you can explain what went wrong, why it was difficult, and how you tried to work around the problems, that will be ok.

We're not looking for you to provide a perfect solution. Given the 2-3 days of technical training, we haven't taught you all the skills and techniques we'd like you to be able to use once you're working as an SDET. What we want to see is:

1. Effective application and understanding of the techniques you've learned so far, and
2. Willingness to try out your own techniques to fill in the gaps (there's things we haven't taught you!). It doesn't matter if what you tried didn't work well, as long as you can explain why you tried it and why it's not great.

Between your 2-3 tests, I recommend having at least one that's simple and sticks to the criteria below, and if you want to do something experimental do that in the other tests.

Criteria

- Test-Model separation:
    - Your tests should not have any Selenium code or details of the automation. Selectors, Selenium types (e.g. WebElement and WebDriverWait) should not be present in the tests. Tests should interact only with the model code.
    - The model shouldn't include any test code. That is, model functions shouldn't perform any assertions, and shouldn't do logic to check test conditions (e.g. a model function like "checkTextIsEqualTo("some text")" that returns a Boolean is leaking test logic into the model – try to avoid this.
- Test case design:
    - We expect the tests written to adhere to the Arrange-Act-Assert pattern: https://automationpanda.com/2020/07/07/arrange-act-assert-a-pattern-for-writing-good-tests/
    - This means your tests should be test steps, then assertions. If this is out of order, you're not doing Arrange-Act-Assert, and should probably split the test into multiple tests.

Some notes on designing a model

Our model uses the programming language's type system to accurately model the entire current state of the app. e.g. on the home page, our model is a HomePage object.

How do we control changes in the model? If an action the user takes (e.g. clickContactMenu) results in a change in the webapp, **it should change our model too.**

Changes in the model are communicated by returning a new model object

App: On the home page, the user clicks the contact menu link. As a result, the user is now on the contact page.

Model: We call HomePage.clickContactMenu(), which returns a ContactPage.

For pages with shared state, we can use **inheritance** to share common code (e.g. a nav bar shared across all pages).

Modelling User Actions

- We’re automating user actions. What actions can the user take?
    - CLICK
    - SET (e.g. typing in a field, selecting from a dropdown)
    - HOVER
    - DRAG and DROP
    - GET*
- *GET is kind of a special case: it’s the user processing information on the screen. It may return data that isn’t a part of the model (like a string or an integer).
- All model functions should start with a verb!
    - E.g. **click**SubmitButton()
    - **get**Email()
    - **set**Message()
- After the verb, describe what the thing is as short and unambiguously as possible
    - A button with text submit may be a “SubmitButton”
    - It could also be a “SignUpButton” if the submitted form is a sign-up form