My Testing Practices

Before integration to source (Pull Requests):

Our group was fortunate to have a dedicated Q.A. member (Emma) who executed and verified most of the tests, my role in this process was to define as clearly as possible the test criteria for the feature I was submitting to the repository. I then supported the testing efforts and made any changes necessary to pass the testing criteria and updated pull requests until the feature reached our groups “definition of done”.

Initially our testing processes were not unified and this resulted in a lot of cross-talk about what things should be doing in the code, how they should be doing it etc. As time went on, my skills at writing these tests in a clear and precise manner increased, as did the quality of communication with our dedicated tester. This resulted in a much higher code quality than individual development would have created.

During Development:

As part of my initial assessment of features needed to scaffold development and create a unified testing architecture I created a population script that programmatically generated a sqlite file that contained an admin user, regular users, and items, and then it assigned those items to specific users creating relations among the tables. This served to establish a fairly cohesive database state that greatly expedited the rest of development and automated all of the tedious manual entry that would have been required as the requirements changed throughout the project.

When writing the backend code for a feature I would determine the set of possible usages a user would have for that functionality. I would then do my best to find all of the permutations of the possible actions a user could perform on the fly. As I was writing all control logic with no corresponding view logic, I had to create a rough sketch of the view logic in a template that was later discarded. I used these templates to verify the validity of data.

Additionally, I interacted directly with the database through queries on the command line; creating items, users, and relations between them to verify their representation on the views of our MVC architecture.

Example Module (retrieving offers associated with a given user's items):

- Equivalence Classes
    - Transaction Item Type
        - Cash Only
        - Free
        - Items Only
    - Offer View
        - My Offers Placed
        - My Offers Received

- Test Cases:
    - Case 1:
        - Cash only item, viewing in My Offers Placed
    - Case 2:
        - Cash only item, viewing in My Offers Received
    - Case 3:
        - Free item, viewing in My Offers Placed
    - Case 4:
        - Free item, viewing in My Offers Received
    - Case 5:
        - Items only, viewing in My Offers Placed
    - Case 6:
        - Items only, viewing in My Offers Received