CS 677 Lab 2 Test Report

Dan Cline, Kyle Stevens, Tian Xia *UMass Amherst*

This document is a report of all the tests ran on the system, and presents sample outputs of each test.

Catalog Server Unit Tests

Test Description: All three servers are up running remotely. Run `TestCatalog.py` in the `tests/unit`. Make sure that the dictionary fields in the unit test is the same as the remote server on initialization. The goal is to make sure that the catalog server functions are working. There are three units in this test:

- `test_query_by_topic`: make sure that querying by topic returns the correct dictionaries.
- `test_query_by_item`: make sure that the results for item querying returns correctly the `price`, `topic`, `item_id` and `amount` field.
- `test_update`: make sure that the changing the price and stock of the items yields the correct value.

Program Output:

```
PASSED: Query by item_id yields correct keys
.PASSED: Query by topic yields correct dictionaries
.PASSED: Decrease stock & price operation correct
.
----------------------------------------------------------------------
Ran 3 tests in 0.011s

OK
```

Order Server Unit Tests

Test Description: All three servers are up running remotely. Run `TestOrder.py` in the `tests/unit`. The goal is to make sure that the order server functions are working. There is one unit in this test:

- `test_buy`: makes sure that only one item is left for an `item_id`. The first buy should be successful and the second should fail.

Program Output:

```
PASSED: Buy with positive stock successful
PASSED: Buy with zero stock unsuccessful
.
----------------------------------------------------------------------
Ran 1 test in 0.017s

OK
```

Concurrent Buy Tests

Test Description: All the three servers are up running remotely. The goal is verify that concurrent requests are processed correctly, returning success and failure messages and that no client makes illegal stock decrements. The test spawns 3 clients which each concurrently make 100 buy requests to item 4, which is initialized to have 100 items. Only 100 items should be sold. The test can be found in `tests/concurrent/TestRaceCondition.sh`

Program Output: We run the script and count the number of fails and successes:

```
$ bash TestRaceCondition.sh > 1.txt
$ grep -o -c Failed 1.txt
$ grep -o -c Successfully 1.txt
```

The first `grep` command returns 200 and second returns 100. Thus the program works as designed to handle concurrent requests.