We can test front-end, catalog and order services by using test.sh present in /src/test.

1. 1st start all servers using sh runAll.sh at root level
2. Teh run test.sh present in /src/test

```
mitaleeminde@Mitalees-MacBook-Air test % sh test.sh
Frontend Service Testing case 4: Testing insufficient quantity buy
Order Service Testing case 2: Testing insufficient quantity buy query
Order received for toy: Elephant and for quantity: 100000000
Response OrderNumber:-1 and expected OrderNumber: -1 both match


.Tested insufficient quantity buy


.Frontend Service Testing case 5: Testing invalid toyname buy
Order Service Testing case 3: Testing invalid item name buy query
Order received for toy: InValid and for quantity: 1
Tested invalid toyname buy


.Frontend Service Testing case 2:Testing invalid toyname query
Response OrderNumber:-1 and expected OrderNumber: -1 both match


.Tested invalid toyname query


.Frontend Service Testing case 3: Testing successful toyname buy
Order Service Testing case 1: Testing Successful buy query
Order received for toy: Tux and for quantity: 1
Order has been processed with OrderNumber 19
Order processed for toy: Tux and for quantity: 1
Response OrderNumber:19 and expected OrderNumber to be greater than 0 both match


Tested successful buy


.Frontend Service Testing case 1: Testing valid toyname query in
.
----------------------------------------------------------------------
Ran 3 tests in 0.132s

OK
Tested valid toyname query


.
----------------------------------------------------------------------
Ran 5 tests in 0.117s
```

```
 OK
 mitaleeminde@Mitalees-MacBook-Air test % <frozen importlib._bootstrap>:1047: ImportWarning: _SixMetaPathImporter.find_spec() n
 ot found; falling back to find_module()
 Catalog Service Testing case 4: Testing insufficient stock Buy
 Response:0 and expected Response: 0, both match


 .Catalog Service Testing case 5: Testing invalid toyName Buy
 Response:-1 and expected Response: -1, both match


 .Catalog Service Testing case 3: Testing successful Buy
 Response:1 and expected Response: 1, both match


 .Catalog Service Testing case 2: Testing invalid toyname Query
 Tested invalid toyname query, giving no response, working as expected


 .Catalog Service Testing case 1: Testing successful Query
 Response toyname:Elephant and expected toy name: Elephant, both match


 .
 ----------------------------------------------------------------------
 Ran 5 tests in 0.071s

 OK
```

a. Above Unit tests include for all of them combined:

i. Query: Valid and InValid products case
ii. Buy: Valid and InValid products, and Insufficient stock case.

Also we can test all services individually using below steps

2. **Catalog services testing**
    a. Run the unit_test.py file, present in the /src/services/catalog folder

```
mitaleeminde@Mitalees-MacBook-Air catalog % python3 unit_test.py
<frozen importlib._bootstrap>:1047: ImportWarning: _SixMetaPathImporter.find_spec() not found; falling back to find_module()
Testing insufficient stock Buy
Response:0 and expected Response: 0, both match


.Testing invalid toyName Buy
Response:-1 and expected Response: -1, both match


.Testing successful Buy
Response:1 and expected Response: 1, both match


.Testing invalid toyname Query
Tested invalid toyname query, giving no response, working as expected


.Testing successful Query
Response toyname:Elephant and expected toy name: Elephant, both match


.
----------------------------------------------------------------------
Ran 5 tests in 0.072s

OK
```

    b. Above Unit tests include:
        i. Query: Valid and InValid products case
        ii. Buy: Valid and InValid products, and Insufficient stock case.

3. **Order services testing**
    a. Run the unit_test.py file, present in /src/services/order folder

```
mitaleeminde@Mitalees-MacBook-Air order % python3 unit_test.py
Testing insufficient quantity buy query
Order received for toy: Elephant and for quantity: 100000000
Response OrderNumber:-1 and expected OrderNumber: -1 both match


.Testing invalid item name buy query
Order received for toy: InValid and for quantity: 1
Response OrderNumber:-1 and expected OrderNumber: -1 both match


.Testing Successful buy query
Order received for toy: Tux and for quantity: 1
Order has been processed with OrderNumber 12
Order processed for toy: Tux and for quantity: 1
Response OrderNumber:12 and expected OrderNumber to be greater than 0 both match


.
----------------------------------------------------------------------
Ran 3 tests in 0.019s

OK
```

b. Unit tests include:
    i. Buy: Valid and InValid products, and Insufficient stock case.

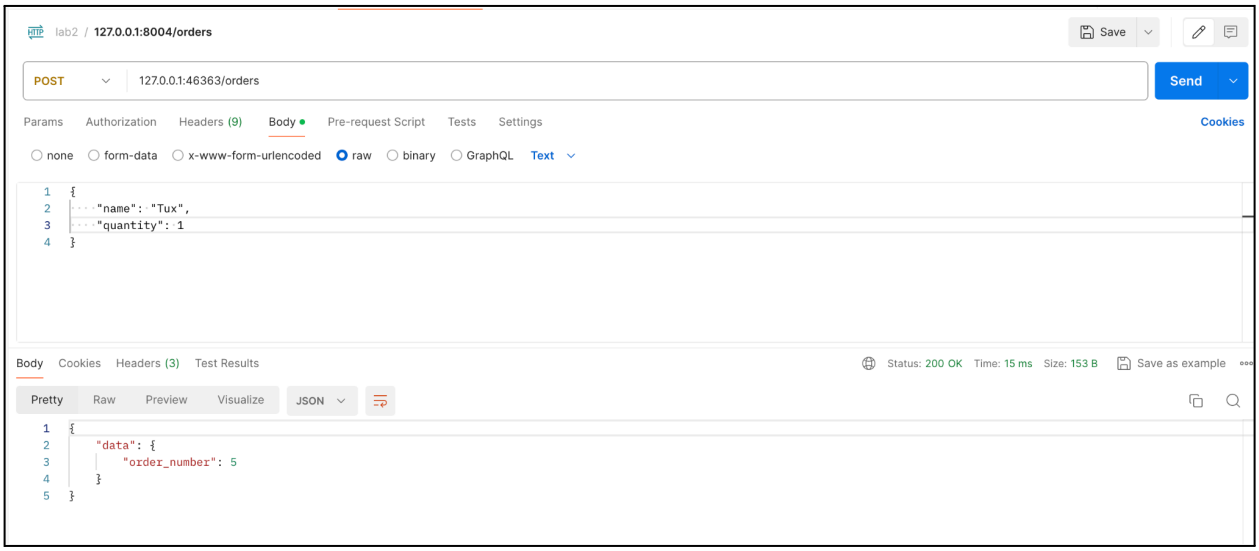
4. Frontend service testing
    a. 1st start all servers using sh runAll.sh at root level
    b. Run the unit_test.py file, present in /src/frontend folder

```
mitaleeminde@Mitalees-MacBook-Air frontend % python3 unit_test.py
Frontend Service Testing case 4: Testing insufficient quantity buy
Tested insufficient quantity buy


.Frontend Service Testing case 5: Testing invalid toyname buy
Tested invalid toyname buy


.Frontend Service Testing case 2:Testing invalid toyname query
Tested invalid toyname query


.Frontend Service Testing case 3: Testing successful toyname buy
Tested successful buy


.Frontend Service Testing case 1: Testing valid toyname query in
Tested valid toyname query


.
----------------------------------------------------------------------
Ran 5 tests in 0.047s

OK
```

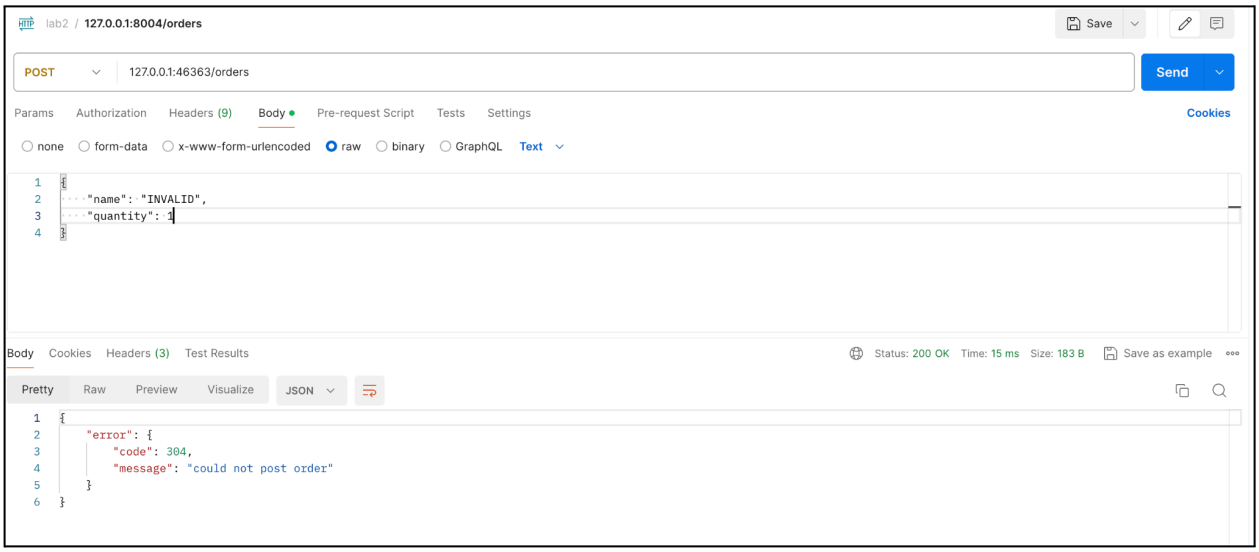
    c. Above Unit tests include:
        i. Query: Valid and InValid products case
        ii. Buy: Valid and InValid products, and Insufficient stock case.

5. **Postman screenshots for various test cases**
    a. To test in postman run the runAll.sh file present on root level

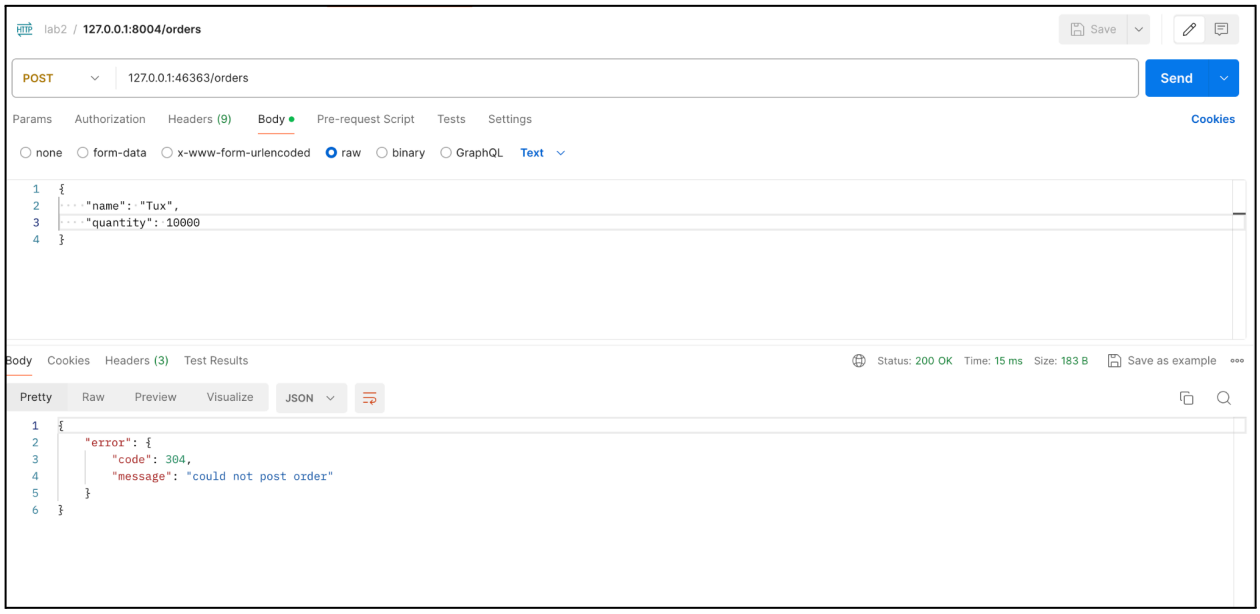
    b. Valid order test case



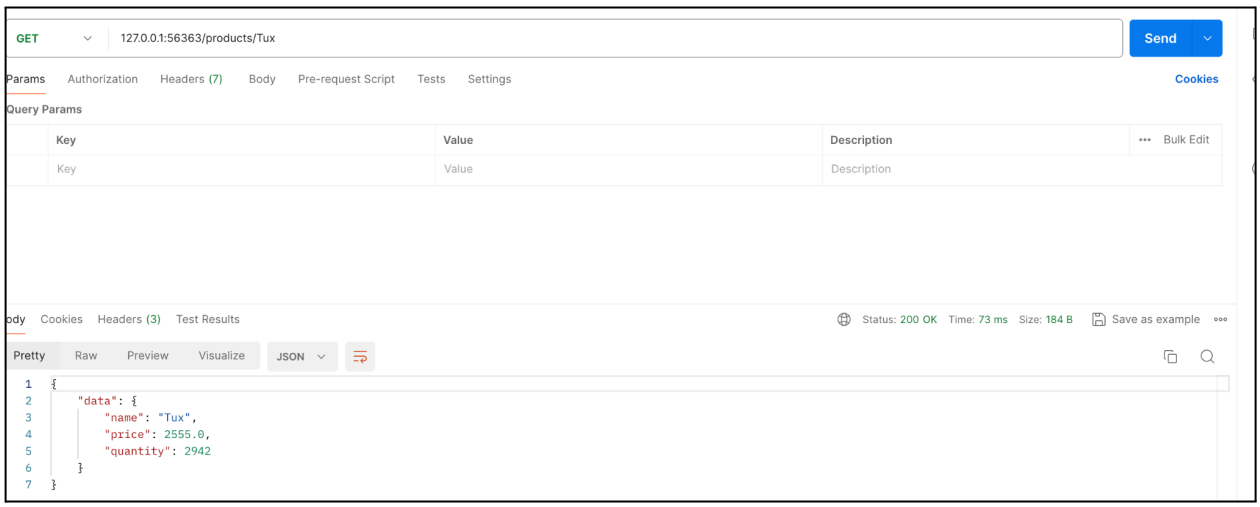
c. Invalid name test case



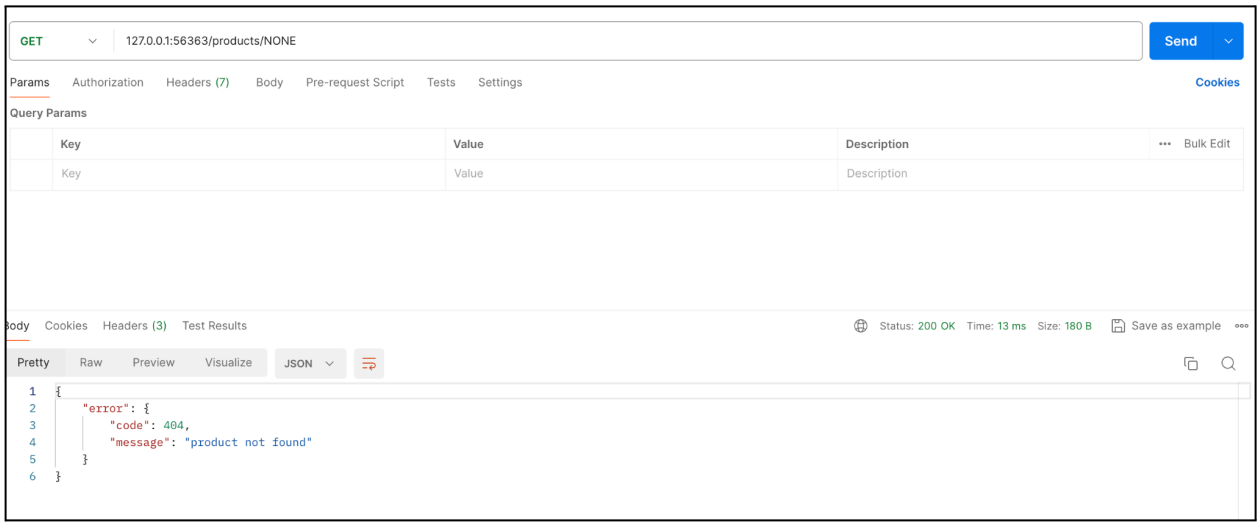
d. Invalid quantity (insufficient) test case



e. Valid test case



f. Invalid Query

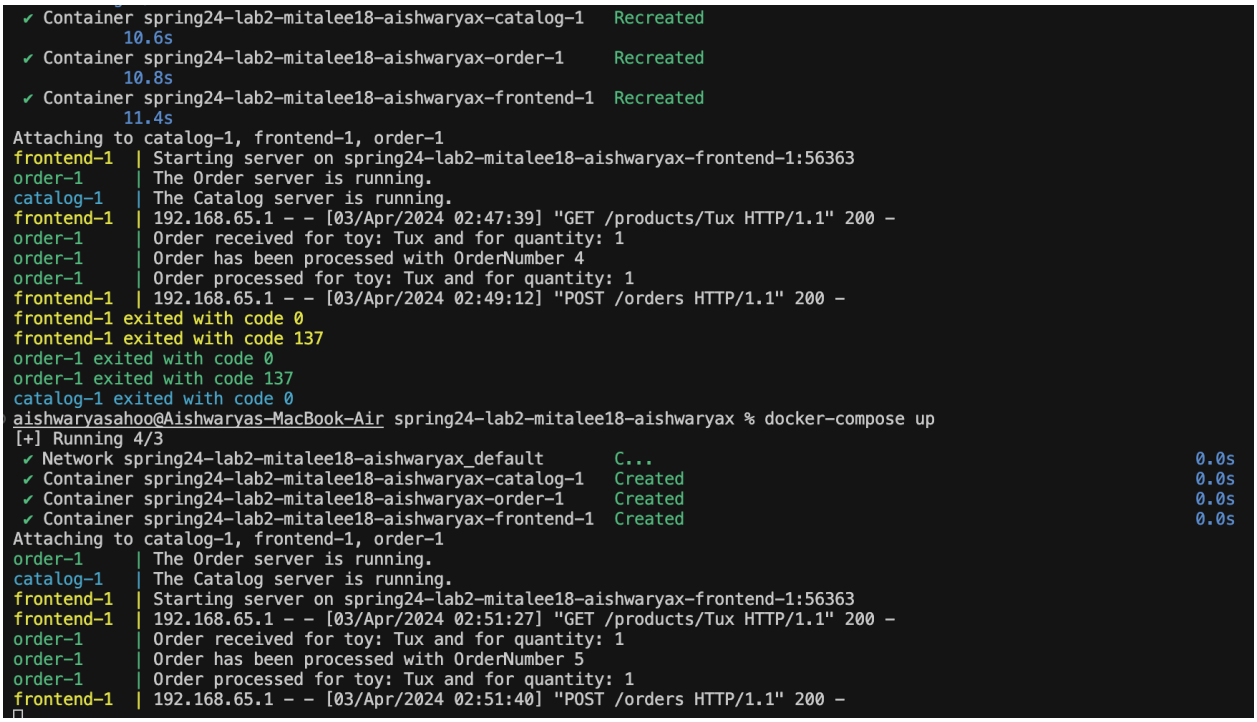


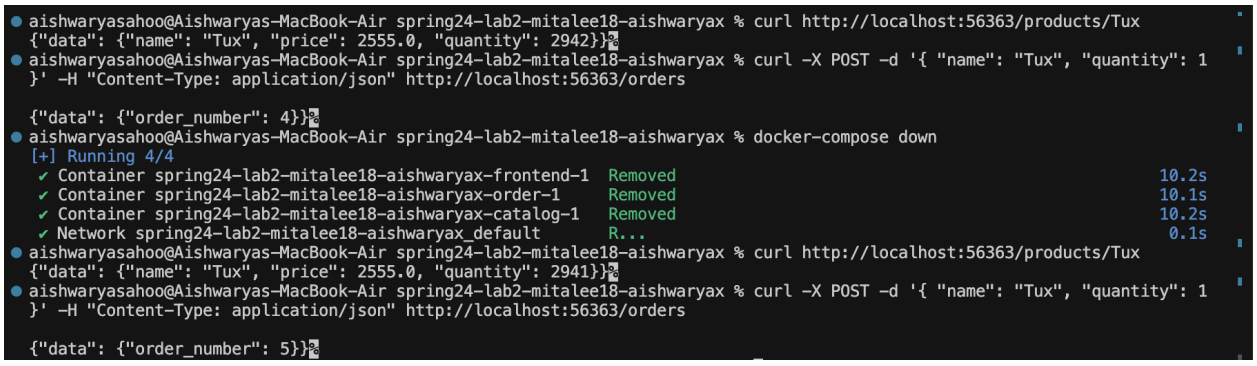
6. **Multithreading approach with client maintaining sessions in HTTP Service Testing**

a. Thread assigned : 13072687104, Client ID: 2
b. Thread assigned : 13096452096, Client ID: 5
c. Thread assigned : 13113241600, Client ID: 3
d. Thread assigned : 13130031104, Client ID: 8
e. Thread assigned : 13146820608, Client ID: 1
f. Thread assigned : 13163610112, Client ID: 4
g. Thread assigned : 13180399616, Client ID: 9
h. Thread assigned : 13096452096, Client ID: 5
i. Thread assigned : 13197189120, Client ID: 10
j. Thread assigned : 13096452096, Client ID: 1
k. Thread assigned : 13146820608, Client ID: 6
l. Thread assigned : 13197189120, Client ID: 10
m. Thread assigned : 13213978624, Client ID: 7
n. Thread assigned : 13197189120, Client ID: 6
o. Thread assigned : 13072687104, Client ID: 2
p. Thread assigned : 13096452096, Client ID: 1
q. Thread assigned : 13113241600, Client ID: 3
r. Thread assigned : 13163610112, Client ID: 8

Client ID and Thread ID mappings are mostly maintained but in some cases they are not due to thread being already assigned to some other Client ID.

7. **Persistence Testing**





From the above screenshot we can see that even after killing docker and restarting it the data in order.csv and catalog.csv remains persistent. We first run the containers and then perform curl commands, then kill the containers and re-run curl commands and observe that order ID and quantity persists despite removing the containers due to volume mounting.