Bharatiya Vidya Bhavan's

Sardar Patel Institute of Technology

(Autonomous Institute Affiliated to University of Mumbai)

Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India

EXPERIMENT 7

Prepare unit test case and perform unit testing

Name: Adwait Purao UID: 2021300101
Name: Viraj Bhalerao UID: 2021301002

Aim: To understand and implement the concept of Unit Testing by preparing test cases andperforming Unit Testing for some program in each language/framework.

Problem Statement:

In the context of modern transportation challenges, our proposed Ferry Management System aims to provide a seamless solution. It offers a user-friendly platform with a diverse range of ferry routes and schedules, ensuring convenient travel options for passengers. The system empowers users to effortlessly book tickets, track ferry status, and receive real-time updates on departure and arrival times. Passengers can personalize their travel experience by selecting cabin preferences and accessing on-board amenities. Additionally, our system prioritizes employee efficiency by offering analytical tools for route optimization, insights into passenger preferences, and real-time financial data access, facilitating informed decision-making for enhanced ferry service management.

Theory:

A test case is exactly what it sounds like: a test scenario measuring functionality across a set of actions or conditions to verify the expected result. They apply to any software application, can use manual testing or an automated test, and can make use of test case management tools.

A key thing to remember when it comes to writing test cases is that they are intended to test a basic variable or task such as whether or not a discount code applies to the right product on an e-commerce web page. This allows a software tester more flexibility in how to test code and features.
Test cases can measure many different aspects of code. The steps involved may also be intended to induce a Fail result as opposed to a positive expected result such as when a user inputs the wrong password on a login screen.
Writing test cases varies depending on what the test case is measuring or testing. This is also a situation where sharing test assets across dev and test teams can accelerate software testing.

Test cases have a few integral parts that should always be present in fields. However, every test case can be broken down into 8 basic steps.

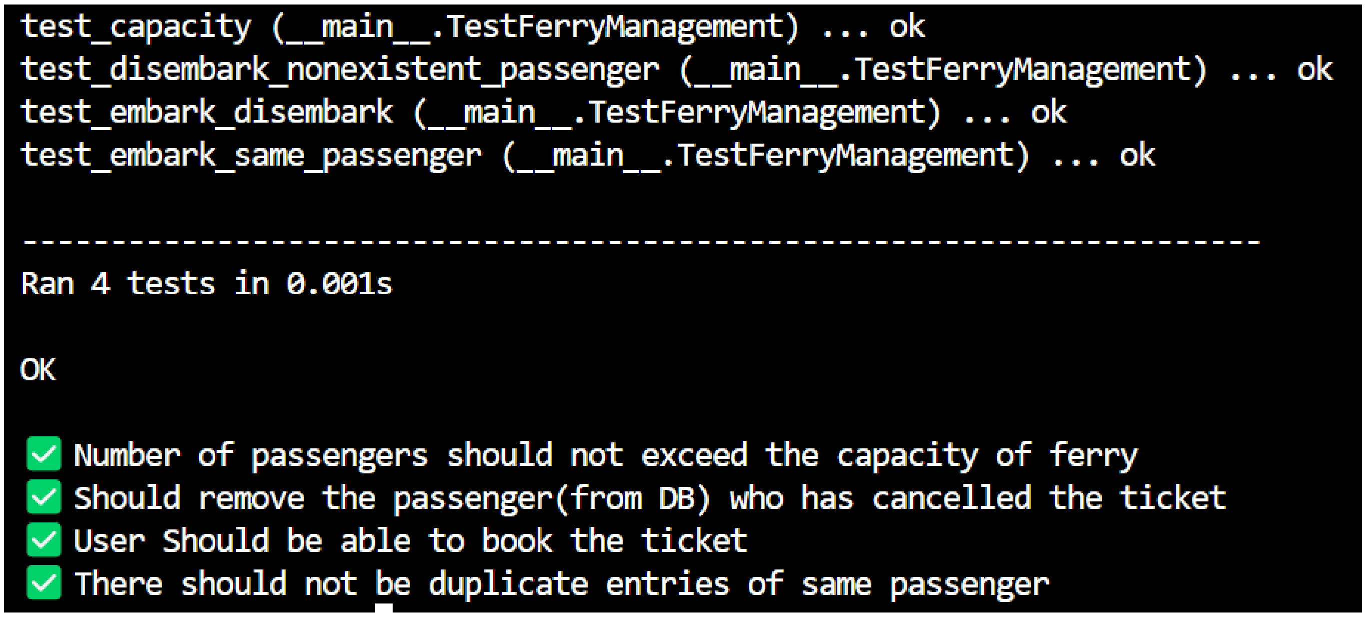
- Test Case ID
- Test Description
- Test Data
- Steps To Be Executed
- Expected Results
- Actual Condition and Post Conditions
- Pass/Fail
- Assumptions and Pre-Condition

Implementation:

```
import unittest
import time

class Ferry:
    def __init__(self, capacity):
        self.capacity = capacity
        self.passengers = []

    def embark(self, passenger):
        if len(self.passengers) < self.capacity and passenger not in self.passengers:
            self.passengers.append(passenger)
            return True
        else:
            return False

    def disembark(self, passenger):
        if passenger in self.passengers:
            self.passengers.remove(passenger)
            return True
        else:
            return False

    def passenger_count(self):
        return len(self.passengers)

class TestFerryManagement(unittest.TestCase):
    def setUp(self):
        self.ferry = Ferry(capacity=5)

    def test_embark_disembark(self):
        self.assertEqual(self.ferry.passenger_count(), 0)

        self.assertTrue(self.ferry.embark("Passenger1"))
        self.assertTrue(self.ferry.embark("Passenger2"))
        self.assertTrue(self.ferry.embark("Passenger3"))

        self.assertEqual(self.ferry.passenger_count(), 3)

        self.assertTrue(self.ferry.embark("Passenger4"))
        self.assertTrue(self.ferry.embark("Passenger5"))

        self.assertFalse(self.ferry.embark("Passenger6"))

        self.assertTrue(self.ferry.disembark("Passenger2"))
        self.assertEqual(self.ferry.passenger_count(), 4)
```

```python
        self.assertFalse(self.ferry.disembark("Passenger2"))
        self.assertEqual(self.ferry.passenger_count(), 4)

    def test_embark_same_passenger(self):
        self.assertTrue(self.ferry.embark("Passenger1"))
        self.assertFalse(self.ferry.embark("Passenger1"))  # Passenger1 is already on
the ferry

    def test_disembark_nonexistent_passenger(self):
        self.assertFalse(self.ferry.disembark("Passenger1"))  # Passenger1 has not
embarked the ferry

    def test_capacity(self):
        self.assertEqual(self.ferry.capacity, 5)

        new_ferry = Ferry(capacity=10)
        self.assertEqual(new_ferry.capacity, 10)

if __name__ == '__main__':
    # Run the tests with formatted output
    runner = unittest.TextTestRunner(verbosity=2)
    start_time = time.time()
    result =
runner.run(unittest.TestLoader().loadTestsFromTestCase(TestFerryManagement))
    end_time = time.time()

    if result.testsRun == 4:
        print("\n✅ Number of passengers should not exceed the capacity of ferry")
        print("✅ Should remove the passenger(from DB) who has cancelled the ticket")
        print("✅ User Should be able to book the ticket")
        print("✅ There should not be duplicate entries of same passenger")

    else :
        print("Tests failed")
```

Output:



```
test_capacity (__main__.TestFerryManagement) ... ok
test_disembark_nonexistent_passenger (__main__.TestFerryManagement) ... ok
test_embark_disembark (__main__.TestFerryManagement) ... ok
test_embark_same_passenger (__main__.TestFerryManagement) ... ok

----------------------------------------------------------------------
Ran 4 tests in 0.001s

OK

✅ Number of passengers should not exceed the capacity of ferry
✅ Should remove the passenger(from DB) who has cancelled the ticket
✅ User Should be able to book the ticket
✅ There should not be duplicate entries of same passenger
```

Conclusion:

From the above experiment, we learned about Theory of unit test cases and performing test cases for the given problem statement for the food delivery service which efficiently captures user interactions, order management, and financial monitoring, emphasizing a user-centric and data-driven approach fora seamless experience and understood relationship between various classes.