Test Plan

for

MPSS - Motor Part Shop Software

Version 1.0 approved

Prepared by,

1. **Swapnil Yasasvi (20CS30054)**
2. **Kulkarni Pranav Suryakant (20CS30029)**
3. **Sidharth Vishwakarma (20CS10082)**

Group No. 5

Indian Institute of Technology, Kharagpur

22nd March 2022

Test Plan Outline

1. Test Plan Identifier

For the **MPSS** version 1.0, this paper specifies the framework and technique that must be utilized for GUI and backend testing.

2. References

Documents that support this test plan include

2.1. Software Requirements Specifications

2.2. Use Case Diagrams

2.3. Class Diagrams

2.4. Test Suite

3. Introduction

The following are the goals of our GUI software testing process:

3.1. Determine and specify the tools that will be utilized throughout the project.

3.2. Specify how the testing will be carried out.

4. Test Items

The testing procedure may be broken down into two categories:

4.1. Using Unit Test APIs, test the software's backend

4.2. Manually test the GUI frontend in accordance with the test suites. Both Linux and Windows-based systems will be used to test them.

5. Features to be Tested

At the owner level:

5.1. Login by the owner

5.2. Add/Order Item

5.3. Remove Item

5.4. View the Inventory

5.5. Recording a new sale

5.6. Generating the order list at the end of the day

5.7. Generating the revenue for the day

5.8. Generating a graph depicting the daily sales for a month

6. Features not to be Tested

Every effort has been made to guarantee that every function or feature of the code has been thoroughly tested through numerous unit tests and final egration testing. No feature should be left untested.

7. Approach(Strategy)

7.1. Testing the Backend

For testing the components used in the backend part, like the methods and modules in various classes, we write testing code using the pytest/unittest , which helps in automating the testing process. Here, we test the functional logic of each method and track changes made to the data members of the classes and the states of the objects. We make a testing class for each of the backend classes, which have unit testing methods for each of the individual methods in the classes. The output generated is verified against the golden output automatically here. If there is a mismatch then an appropriate message is displayed.

7.2. Testing the Database

The database queries and updates mostly happen inside the methods of the backend classes simultaneously with the update of the state of the objects. So, there is no need to write separate methods for testing this part. The tests for verifying that the database is always updated properly can be written in the unit testing functions described in the previous part itself

7.3. Testing the Frontend

This is the part where we need to verify that the GUI is functioning as expected and everything is positioned on the screen properly. Automating this task is extremely difficult, and hence this has to be done manually. So, here we just manually proceed, entering the appropriate inputs and going on checking if the correct results are displayed in an appropriate format.

7.4. Test Compliance Report

At the end, we create a test compliance report, which lists the various test results as PASS / FAIL.

8. Environmental Needs

The unit tests use the pytest runners or the unittest runner which is inbuilt in the Python standard library.These are very useful and handy in running the tests. Hence , it would be recommended to use that.

9. Test Scenarios for Frontend GUI Interface

9.1. Home Page

This window doesn’t demand any input thus not to be tested

9.2. Login Page

9.2.1. Both Username and Password are correct [Positive].

9.2.2. The username is incorrect and the Password is correct[Negative].

9.2.3. The username is correct and the Password is incorrect[Negative].

9.2.4. Both Username and Password are incorrect[Negative].

9.3. Dashboard

9.3.1. Working of all buttons[Positive].

9.4. Add Item

- 9.4.1. Working of all drop-down menus [Positive].
- 9.4.2. Working of all buttons[Positive].
- 9.4.3. Working of all text fields[Positive].
- 9.4.4. Data entered in all fields is valid[Positive].
- 9.4.5. Data entered is invalid[Negative].

9.5. Remove an Item

- 9.5.1. Working of all drop-down menus[Positive].
- 9.5.2. Working of all buttons[Positive].
- 9.5.3. All the fields chosen - Items Type , Manufacturer Name and Vehicle Type are valid[Positive].

9.6. Report a Sale

- 9.6.1. Working of all drop-down menus[Positive].
- 9.6.2. Working of all text fields[Positive].
- 9.6.3. Working of all buttons[Positive].
- 9.6.4. All the fields chosen and data entered are valid[Positive].
- 9.6.5. Quantity entered is a natural member[Positive].
- 9.6.6. Quantity entered is greater than the quantity in the inventory[Negative].

9.7. View Inventory

- 9.7.1. Working of all scrollable lists of items[Positive].
- 9.7.2. Working of all buttons[Positive].

9.8. EOD Tasks

- 9.8.1. Computation of the number of items to be ordered at the EOD[Positive].
- 9.8.2. Working of the generated order list , which will be a scrollable list [Positive]
- 9.8.3. Working of all buttons[Positive].

9.9. Graph for Daily sales at the month end

- 9.9.1. View the graph before the first month has ended [Negative].
- 9.9.2. View the graph on the day a month has ended[Positive].
- 9.9.3. View the graph in the middle of a month [Positive].

10. Test Scenarios for Database Management and Backend classes

10.1. Testing the Owner class

10.1.1. Testing the getName( ) function

- 10.1.1.1. Retrieve and verify the name of the owner[Positive].

10.1.2. Testing the setName(name) function

10.1.2.1. Set the name of the owner to a valid string[Positive].

10.1.2.2. Set the name of the owner to a invalid string[Negative].

10.1.3. Testing the getUserName( ) function

10.1.3.1. Retrieve and verify the name of the owner[Positive].

10.1.4. Testing the setUserName(username) function

10.1.4.1. Set the username of the owner to a valid string[Positive].

10.1.4.2. Set the username of the owner to an invalid string[Negative].

10.1.5. Testing the validate(username , password) function

10.1.5.1. Both the username and password passed are the same as the actual username and password of the owner[Positive].

10.1.5.2. username is incorrect and password is correct[Negative].

10.1.5.3. username is correct and password is incorrect[Negative].

10.1.5.4. Both username and password are incorrect[Negative].

10.2. Testing the Item class

10.2.1. Testing the Constructor Item(type , price , quantity , ManufacturerID , VehicleType , Data StartDate)

The constructor is called only after ensuring that all the parameters passed are valid. So , they will be tested either in GUI testing or in database testing. So there is no need to test the constructor here.

10.2.2. Testing the getID( ) function

10.2.2.1. Retrieve and verify the userID of an Item object[Positive].

10.2.3. Testing the getType( ) function

10.2.3.1. Retrieve and verify the type of an Item object[Positive].

10.2.4. Testing the getPrice( ) function

10.2.4.1. Retrieve and verify the price of an Item object[Positive].

10.2.5. Testing the getQuantity( ) function

10.2.5.1. Retrieve and verify the quantity of an Item object[Positive].

10.2.6. Testing the getManufacturerID( ) function

10.2.6.1. Retrieve and verify the manufacturerID of an Item object[Positive].

10.2.7. Testing the getVehicalType( ) function

10.2.7.1. Retrieve and verify the vehicalType of an Item object[Positive].

10.2.8. Testing the getStartDate( ) function

10.2.8.1. Retrieve and verify the startDate of an Item object[Positive].

10.2.9. Testing the save( ) function

10.2.9.1. Insert an item to the database when it is empty[Positive].

10.2.9.2. Insert an item to the database when it is non empty[Positive].

10.2.10. Testing the delete( ) function

10.2.10.1. Delete an item when multiple items are present in the inventory database[Positive].

10.2.10.2. Delete an item when only a single item is present in the inventory database[Positive].

10.2.11. Testing the updateSale(numSold) function

10.2.11.1. Update the quantity of an item being sold[Positive].

10.3. Testing the Login class

10.3.1. Testing the AuthenticLogin(username , password)

10.3.1.1. Both the username and password passed are the same as the actual username and password of the owner[Positive].

10.3.1.2. username is incorrect and password is correct[Negative].

10.3.1.3. username is correct and password is incorrect[Negative].

10.3.1.4. Both username and password are incorrect[Negative].

10.4. Testing the Manufacturer class

10.4.1. Testing the Constructor Manufacturer(name , address)

10.4.1.1. The constructor is called only after ensuring that all the parameters passed are valid. So,they will be tested either in the GUI testing or in the database testing. So, there is no need to test the constructor here.

10.4.2. Testing the getUID( ) function

10.4.2.1. Retrieve and verify the userID of a Manufacturer object.[Positive].

10.4.3. Testing the getName( ) function

10.4.3.1. Retrieve and verify the name of a Manufacturer object.[Positive].

10.4.4. Testing the getItemCount( ) function

10.4.4.1. Retrieve and verify the itemCount of a Manufacturer object.[Positive].

10.4.5. Testing the save( ) function

10.4.5.1. Insert a new manufacturer to the database when it is empty[Positive].

10.4.5.2. Insert a new manufacturer to the database when it is non-empty[Positive].

10.4.6. Testing the delete( ) function

10.4.6.1. Delete a manufacturer when multiple manufacturers are present in the inventory database[Positive].

10.4.6.2. Delete a manufacturer when only a single manufacturer is present in the inventory database[Positive].

10.5. Testing the Inventory class

10.5.1. Testing the retrieveData( ) function

10.5.1.1. The inventory database is empty[Positive].

10.5.1.2. The inventory database is not empty[Positive].

10.5.2. Testing the removeItem(itemID) function

10.5.2.1. Remove an item when multiple items are present in the inventory database[Positive].

10.5.2.2. Remove an item when only a single item is present in the inventory[Positive].

10.5.2.3. The manufacturer of the item being deleted does not make any other item[Positive].

10.5.2.4. The manufacturer of the item being deleted makes some other item(s) too. [Positive]

11. Pass/Fail

For our test plan, the conditions that must be completed are different. We may think about the following at the

11.1. Unit Test Level:

11.1.1. If all test cases are passed i.e. their output matches with our golden output

11.1.2. If all of the methods associated with a certain class have been thoroughly tested.

11.2. Front-end testing:

11.2.1. The visual result is the same as it ended.

11.2.2. All of the tools and buttons on all of the windows have been thoroughly tested.

12. Suspension criteria and resumption

The tests aren halted and the codebase is worked on if the code fails two or more tests. Resumption of test sets will be done after exhaustively running unit tests on the refactored source.

13. Software risk Issues

There are several inherent software dangers such as:

13.1. Database crashes given by MySQL.

13.2. MySQL's support has come to an end.

13.3. Changes to the **electron** library , which may no longer support some old features

13.4. Some functions are complicated , and they may disrupt the program's typical flow.

14. Schedule

Before the final completion of the codebase , the first set of tests will be formed everyday for a week. The last set of tests will be run 12 hours before the code is submitted to confirm that no errors remain.

15. Glossary

MPSS : Motor Parts Shop Software

API : Application Programming Interface

GUI	Graphical User Interface
SQL	Structured Query Language