TEST PLAN

for

Zulu - A Motor Part Shop Software

Prepared by -

Ashutosh Kumar Singh (19CS30008)

Vanshita Garg (19CS10064)

Suhas Jain (19CS30048)

Indian Institute of Technology, Kharagpur

March 26, 2021

Contents

1 Introduction

This is the Master Test Plan for Zulu - a Motor Part Shop Software. This document mainly addresses the various features that will be tested. The document then lists down the various scenarios that can arise for each of the mentioned features. The primary motive of this plan is to ensure that at the end of the testing process, the software performs all the tasks as expected. It will also ensure that the software never goes to an inconsistent state or crashes unexpectedly.

Presently, only the unit testing is being done, i.e, all the functions and modules will be tested individually. The software is executed for various scenarios and then the output produced is cross-checked with the theoretical golden output either manually or automatically.

Note : The words *Positive* or *Negative* in brackets after every test scenario denote whether it is a Positive scenario or a Negative scenario, respectively.

2 Features to be Tested

The following are the features at the user level that need to be tested :

- Adding a new item to the inventory
- Removing an item from the inventory
- Viewing the current inventory
- Recording a new sale
- Generating the order list at the end of the day
- Generating the revenue for the day
- Generating a graph depicting the daily sales for a month

3 Approach (Strategy)

3.1 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 JUnit 4 framework, 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.

3.2 Testing the Database

The database queries and updates mostly happen inside the methods of the backend classes simultaneously with the updation 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.

3.3 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 go on checking if the correct results are displayed in an appropriate format.

3.4 Test Compliance Report

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

4 Environmental Needs

The unit tests use the JUnit 4 framework. Also, Apache Netbeans IDE automatically generates the test code skeleton using the JUnit 4 framework, and is also very useful and handy for running the tests. Hence, it would be recommended to use that.

5 Test Scenarios for the Frontend GUI Interface

5.1 Home Page

This window does not demand any input and thus need not be tested.

5.2 Login Page

1. Both Username and Password are correct [*Positive*]
2. Username is correct but Password is incorrect [*Negative*]
3. Username is incorrect but Password is correct [*Negative*]
4. Both Username and Password are incorrect [*Negative*]

5.3 Dashboard

1. Working of all buttons [*Positive*]

5.4 Add an Item

1. Working of all drop-down menus [*Positive*]
2. Working of all text fields [*Positive*]
3. Working of all buttons [*Positive*]
4. The data entered in all the fields is valid/correct [*Positive*]
5. Item Type entered is invalid [*Negative*]
6. Manufacturer Name entered is invalid [*Negative*]
7. Manufacturer Address is invalid [*Negative*]
8. Vehicle Type entered is invalid [*Negative*]
9. Price entered is not a number [*Negative*]
10. Price of the item entered is zero [*Negative*]
11. Price of the item entered is negative [*Negative*]
12. Initial Quantity entered is not a number [*Negative*]
13. Initial Quantity entered is zero [*Negative*]
14. Initial Quantity entered is negative [*Negative*]

Note : Any Item Type, Manufacturer Name, Manufacturer Address or Vehicle Type is considered invalid if it has any character other than alphabets, numbers, comma, dot, apostrophe or ampersand.

5.5 Remove an Item

1. Working of all drop-down menus [*Positive*]
2. Working of all buttons [*Positive*]
3. All the fields chosen - Item Type, Manufacturer Name and Vehicle Type are valid [*Positive*]

5.6 Report a Sale

1. Working of all drop-down menus [*Positive*]
2. Working of all text fields [*Positive*]
3. Working of all buttons [*Positive*]
4. All the fields chosen and data entered are valid [*Positive*]
5. Quantity entered is not a number [*Negative*]
6. Quantity entered is zero [*Negative*]
7. Quantity entered is negative [*Negative*]
8. Quantity entered is greater than the quantity in the inventory [*Negative*]

5.7 View Inventory

1. Working of the scrollable list of items [*Positive*]
2. Working of all buttons [*Positive*]

5.8 Day - End Tasks

1. Computation of number of items to be ordered at the end of a day [*Positive*]
2. Working of the generated order list, which will be a scrollable list [*Positive*]
3. Working of all buttons [*Positive*]

5.9 Graph for Daily Sales of a Month

1. View the graph before the first month has ended [*Negative*]
2. View the graph on the day a month has ended [*Positive*]
3. View the graph in the middle of a month [*Positive*]

6 Test Scenarios for Backend Classes and Database Management

6.1 Testing the `Owner` class

6.1.1 Testing the `getName()` function

1. Retrieve and verify the `name` of the owner [*Positive*]

6.1.2 Testing the `setName(String name)` function

1. Set the `name` of the owner to a valid string [*Positive*]
2. Set the `name` of the owner to an invalid string [*Negative*]

6.1.3 Testing the `getUsername()` function

1. Retrieve and verify the `name` of the owner [*Positive*]

6.1.4 Testing the `setUsername(String username)` function

1. Set the `username` of the owner to a valid string [*Positive*]
2. Set the `username` of the owner to an invalid string [*Negative*]

6.1.5 Testing the `validate(String username, String password)` function

1. Both the `username` and `password` passed are the same as the actual `username` and `password` of the owner [*Positive*]
2. `username` is correct but `password` is incorrect [*Negative*]
3. `username` is incorrect but `password` is correct [*Negative*]
4. Both `username` and `password` are incorrect [*Negative*]

6.2 Testing the `Item` class

6.2.1 Testing the Constructor `Item(String type, double price, int quantity, int manufacturerID, String vehicleType, Date startDate)`

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.

6.2.2 Testing the `getUID()` function

1. Retrieve and verify the `uID` of an `Item` object. [*Positive*]

6.2.3 Testing the `getType()` function

1. Retrieve and verify the `type` of an `Item` object. [*Positive*]

6.2.4 Testing the `getPrice()` function

1. Retrieve and verify the `price` of an `Item` object. [*Positive*]

6.2.5 Testing the `getQuantity()` function

1. Retrieve and verify the `quantity` of an `Item` object. [*Positive*]

6.2.6 Testing the `getManufacturerID()` function

1. Retrieve and verify the `manufacturerID` of an `Item` object. [*Positive*]

6.2.7 Testing the `getVehicleType()` function

1. Retrieve and verify the `vehicleType` of an `Item` object. [*Positive*]

6.2.8 Testing the `getStartDate()` function

1. Retrieve and verify the `startDate` of an `Item` object. [*Positive*]

6.2.9 Testing the `save()` function

1. Insert an item to the database when it is empty [*Positive*]
2. Insert an item to the database when it is not empty [*Positive*]

6.2.10 Testing the `delete()` function

1. Delete an item when multiple items are present in the inventory database [*Positive*]
2. Delete an item when only a single item is present in the inventory database [*Positive*]

6.2.11 Testing the `updateSale(int numSold)` function

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

6.3 Testing the `Manufacturer` class

6.3.1 Testing the Constructor `Manufacturer(String name, String address)`

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.

6.3.2 Testing the `getUID()` function

1. Retrieve and verify the `uID` of a `Manufacturer` object. [*Positive*]

6.3.3 Testing the `getName()` function

1. Retrieve and verify the `name` of a `Manufacturer` object. [*Positive*]

6.3.4 Testing the `getAddress()` function

1. Retrieve and verify the `address` of a `Manufacturer` object. [*Positive*]

6.3.5 Testing the `getItemCount()` function

1. Retrieve and verify the `itemCount` of a `Manufacturer` object. [*Positive*]

6.3.6 Testing the `save()` function

1. Insert a new manufacturer to the database when it is empty [*Positive*]
2. Insert a new manufacturer to the database when it is not empty [*Positive*]

6.3.7 Testing the `delete()` function

1. Delete a manufacturer when multiple manufacturers are present in the inventory database [*Positive*]
2. Delete a manufacturer when only a single manufacturer is present in the inventory database [*Positive*]

6.4 Testing the `Inventory` class

6.4.1 Testing the `retrieveData()` function

1. The inventory database is empty [*Positive*]
2. The inventory database is not empty [*Positive*]

6.4.2 Testing the `removeItem(int itemID)` function

1. Remove an item when multiple items are present in the inventory database [*Positive*]
2. Remove an item when only a single item is present in the inventory [*Positive*]
3. The manufacturer of the item being deleted does not make any other item. [*Positive*]
4. The manufacturer of the item being deleted makes some other item(s) too. [*Positive*]