Test Suite

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 Suite Contents

1. Introduction and Important Points

This is the Motor Part Shop Software (MPSS) Test Suite document. For each scenario indicated in sections 5 and 6 of the Test Plan specification, we present test cases together with their expected golden result.

Note: The numbers put in parentheses following the heads of a section or subsection represent the relevant section number in the Test Plan Text at all times in this document.

2. Test Cases for Frontend GUI interfaces

2.1. Home Page

This page cannot be checked since it is only an intermediary page used to speed up the download of the program, which then automatically redirects to the login page. There are no capabilities that can be tested in this location.

2.2. Login Page

1. Both Username and Password are correct

Input:

`Username:` ComplexProcessors

`Password:` HelloMPSS

Response:

`Login attempt is successful and the software dashboard is displayed.`

2. Username is correct but the password is incorrect

Input:

`Username:` ComplexProcessors

`Password:` Hello_MPSS

Response:

`A suitable message is presented in a new popup window since the attempt to log in is unsuccessful.`

3. Username is incorrect but the password is correct

Input:

`Username:` Complex_Processors

`Password:` HelloMPSS

Response:

`A suitable message is presented in a new popup window since the attempt to log in is unsuccessful.`

4. Both Username and Password are incorrect

Input:

`Username:` Complex_Processors

`Password:` Hello_MPSS

Response:

`A suitable message is presented in a new popup window since the attempt to log in is unsuccessful.`

2.3. Dashboard

Working of all tabs/buttons:

Input:

One by one, select `Add an Item, Remove an Item, Report Sale, View Inventory, End Day, View Graph, and Back from the menus.`

Response:

`Each click should take you to the correct window.`

2.4. Add an Item

Pre-Condition:

There are the following items in the inventory:

- `Item Type -` Headlights
  `Manufacturer Name` - GeoMechanic
  `Vehicle Type` - SUV
- `Item Type -` Horns
  `Manufacturer Name` - Bosch
  `Vehicle Type` - Sedan
- `Item Type` - Batteries
  `Manufacturer Name` - Amaron
  `Vehicle Type` - Wagon

2.4.1. Working of all drop-down menus:

2.4.1.1. **Input:** Drop-down menu for Item Types

`OUTPUT / RESPONSE:`

- Headlights
- Horns
- Batteries

2.4.1.2. **Input:** Drop-down menu for Manufacturer Names

`OUTPUT / RESPONSE:`

- GeoMechanic
- Bosch
- Amaron

2.4.1.3. **Input:** Drop-down menu for Vehicle Types

`OUTPUT / RESPONSE:`

- SUV
- Sedan
- Wagon

2.4.2. Working of all text fields:

2.4.2.1. **Input:** `Item Type -` Headlights
`Manufacturer Name` - GeoMechanic
`Vehicle Type` - SUV

`OUTPUT / RESPONSE:`

`While Typing Input is should be visible in respective text boxes.`

2.4.3. Working of all input buttons:

2.4.3.1. **Input:** Selection of add button followed by show and back button.

`OUTPUT / RESPONSE:`

`Each button after clicking should redirect to an appropriate window/message.`

2.4.4. Adding data with valid entries:

Input:

`Item Type` - Headlights
`Manufacturer Name` - GeoMechanic
`Vehicle Type` - SUV
`Price` - 2000
`Quantity` - 8

OUTPUT / RESPONSE:

As all the inputs are given in the correct format, the item should be added to the inventory and visible after using drop-down for view inventory.

2.4.5. Adding data with invalid entries:

Input:

`Item Type` - He@dli#hts
`Manufacturer Name` - GeoMech@nic
`Vehicle Type` - $UV
`Price` - Rs. 2000
`Quantity` - Eight

OUTPUT / RESPONSE:

As the inputs given are in incorrect format, an error message should pop up stating the error and occurred in which type (in this case for all types).

2.5. Remove an Item

Pre-condition:

The following items are already in our inventory:

- `Item Type` - Headlights
  `Manufacturer Name` - GeoMechanic
  `Vehicle Type` - SUV
- `Item Type` - Horns
  `Manufacturer Name` - Bosch
  `Vehicle Type` - Sedan
- `Item Type` - Headlights
  `Manufacturer Name` - Amaron
  `Vehicle Type` - Wagon

1. Input:

Examine the options for `Item Type, Manufacturer Name, and Vehicle Type` in the lists.

OUTPUT / RESPONSE:

The following option(s) will be available in the `Item Type` menu:

- Headlights
- Horns

Suppose we select `Headlights`. Now we need to select the `Manufacturer Name` from the menu. The menu will have the following option(s):

- GeoMechanic
- Amaron

Now let us select `GeoMechanic`. The vehicle type menu will now have the following option(s):

- SUV

2. Working of buttons:

Input:

Click on the buttons - Remove and back

OUTPUT / RESPONSE:

`Each click should take the software to the appropriate window and/or show an appropriate message.`

3. All selections i.e. Item Type, Manufacturer Name, and Vehicle Type are valid

Input:

The selection is performed similarly to that described in the input part of this section.

`Item Type` - Headlights

`Manufacturer Name` - GeoMechanic

`Vehicle Type` - SUV

Then the Remove button is clicked.

OUTPUT / RESPONSE:

`This item has to be taken out of the inventory. The View Inventory option on the Dashboard may be used to confirm this.`

2.6. Report a Sale

There are the following items in the inventory:

- `Item Type` - Headlights
  `Manufacturer Name` - GeoMechanic
  `Vehicle Type` - SUV
- `Item Type` - Horns
  `Manufacturer Name` - Bosch
  `Vehicle Type` - Sedan
- `Item Type` - Headlights
  `Manufacturer Name` - Amaron
  `Vehicle Type` - Wagon

1. Working of all menus

Look at the options available in the drop-down menus for `Item Type, Manufacturer Name, and Vehicle Type.`

OUTPUT / RESPONSE:

The `Item Type` drop-down menu will have the following option(s):

- Headlights
- Horns

Suppose now we select `Horns,` The `Vehicle Type` drop-down menu will now have the following option(s):

- Sedan

2. Working of all text fields

Input:

`Quantity - 7`

OUTPUT / RESPONSE:

The typed text should be visible in the text box.

3. Working of all the buttons

Input:

Click on the buttons: Report Sale and Back

OUTPUT / RESPONSE:

Upon clicking the software should redirect towards the appropriate window and/or display the required message.

4. All the fields chosen and the data entered are valid:

Input:

Item Type - Headlights

Manufacturer Name - GeoMechanic

Vehicle Type - SUV

Quantity - 7

OUTPUT / RESPONSE:

The transaction should be performed, a success message shown, and the amount of the item in the inventory should be changed correctly.

5. Quantity entered is non-numeric

Input:

Item Type - Headlights

Manufacturer Name - GeoMechanic

Vehicle Type - SUV

Quantity - Seven

OUTPUT / RESPONSE:

An error message should be displayed saying that the Quantity entered is not a number.

6. Quantity entered is 0

Input:

Item Type - Headlights

Manufacturer Name - GeoMechanic

Vehicle Type - SUV

Quantity - 0

OUTPUT / RESPONSE:

An error message should be displayed saying that the Quantity entered is zero.

7. Quantity entered is negative

Input:

Item Type - Headlights

Manufacturer Name - GeoMechanic

Vehicle Type - SUV

Quantity - -5

OUTPUT / RESPONSE:

An error message should be displayed saying that the Quantity entered is negative.

8. Quantity entered is greater than the quantity in the inventory

Input:

Item Type - Headlights

Manufacturer Name - GeoMechanic

Vehicle Type - SUV
Quantity - 12
OUTPUT / RESPONSE:
An error message should be displayed saying that the quantity entered is greater than the quantity of the item in the inventory.

2.7. View Inventory

1. Working of the scrollable list of items

Input:

Navigate down the list using the mouse wheel.

OUTPUT / RESPONSE:

A list of all the items in the inventory should be displayed in the form of a scrollable table/list. The list should show all the details for each item, like Item Type, Manufacturer Name, Vehicle Type, Quantity, Price, and In Stock or Not In Stock.

2. Working of all buttons

Input:

Click on the Back button.

OUTPUT / RESPONSE:

The click should redirect to the appropriate window.

2.8. Day-End Tasks

1. Computation of a number of items to be ordered at the end of a day

Input:

First, add a new item:
Item Type - Headlights
Manufacturer Name - GeoMechanic
Manufacturer Address - Andheri West, Mumbai, Maharashtra
Vehicle Type - SUV
Price - 7000
Initial Quantity - 3

On Day 1, Report Sale:
Quantity - 1

On Day 2, Report Sale:
Quantity - 11

OUTPUT / RESPONSE:
At the end of Day 1:
Quantity to be Ordered - 13
At the end of Day 2:
Quantity to be Ordered - 16

2. Working of the generated order list, which will be a scrollable list

Input:

Click on the appropriate option to generate the order list.

OUTPUT / RESPONSE:
A list of all the items to be ordered is generated. The list shows the unique ID, Item Type, Manufacturer Name, Manufacturer Address, Vehicle Type, and Quantity to be ordered for each item.

3. Working of all buttons

Input:

Click on the View Daily Revenue and Back.

OUTPUT / RESPONSE:
Each click should redirect to an appropriate window.

2.9. Graph for Daily Sales for a Month

1. View the graph before the first month has ended

Input:

Select the View Graph option.

OUTPUT / RESPONSE:
A message should be displayed saying that the first month has not yet been completed.

2. View the graph on the day a month has ended

Input:

Select the View Graph option.

OUTPUT / RESPONSE:
A graph showing the daily sales for the month which has just been completed should be displayed.

3. View the graph in the middle of a month

Input:

Select the View Graph option.

OUTPUT / RESPONSE:
A graph showing the daily sales for the previous completed month should be displayed.

3. Test Cases for Backend and Database Management

3.1. Test the `Owner Class`

3.1.1. Test the `getName()` function

1. Retrieve and verify the name of the owner

Input:

Call the `getName()` function.

OUTPUT / RESPONSE:
name = GeoMechanic

3.1.2. Test the `setName(name)` function

1. Set the name of the owner to a valid string

Input:

name = GeoMechanic

OUTPUT / RESPONSE:
No output as such, just verify that the name has changed.

2. Set the name of the owner to an invalid string

Input:

name = Ge0Mec#@nic

OUTPUT / RESPONSE:

An error message should be displayed.

3.1.3. Test the `getUserName()` function

1. Retrieve and verify the username of the owner

Input:

Call the `getUsername()` function.

OUTPUT / RESPONSE:

username = Complex_Processors

3.1.4. Test the `setUserName(username)` function

1. Set the username of the owner to a valid string

Input:

username = Useradmin

OUTPUT / RESPONSE:

No output as such, just verify that the username has changed.

2. Set the username of the owner to an invalid string

Input:

username = User*@#admin

OUTPUT / RESPONSE:

An error message should be displayed.

3.1.5. Test the `AuthenticateLogin(username, password)` function

1. Both the username and password passed are the same as the actual username and password of the owner

Input:

username = Useradmin

password = password123

OUTPUT / RESPONSE:

True

2. Username is correct but password is incorrect

Input:

username = useradmin

password = Password456

OUTPUT / RESPONSE:

False

3. Username is incorrect but password is correct

Input:

username = useradmin

password = password123

OUTPUT / RESPONSE:

False

4. Both username and password are incorrect

Input:

username = useradmin

password = Password456

OUTPUT / RESPONSE:

False

3.2. Testing the `Item class`

3.2.1. Test the `Constructor Item(type, price, quantity, manufacturerID, vehicleType, startDate)`

The constructor is called only after ensuring that all the parameters passed are valid.

So, there is no need to test the constructor here.

PRE - CONDITION for section 3.2.2 to 3.2.8:
Consider that no `Item` object has been created till now and suppose we create the following objects on the date 2021-04-03 (yyyy-mm-dd).
Create an `Item` object with the following attributes:

```
type = Battery
price = 6700
quantity = 8
manufacturerID = 1
vehicleType = PickupTruck
```

Now, create another object with the following attributes:

```
type = Mirror
price = 1700
quantity = 12
manufacturerID = 2
vehicleType = Minivan
```

3.2.2. Test the `getUID()` function

1. Retrieve and verify the uID of an Item object.

Input:

Call the function on the first and second object one by one.

OUTPUT / RESPONSE:

```
uID = 1 (for the first object)
uID = 2 (for the second object)
```

3.2.3. Testing the `getType()` function

1. Retrieve and verify the type of an Item object.

Input:

Call the function on the first object.

OUTPUT / RESPONSE:

```
type = Battery
```

3.2.4. Test the `getPrice()` function

1. Retrieve and verify the price of an Item object.

Input:

Call the function on the first object.

OUTPUT / RESPONSE:

```
price = 1700
```

3.2.5. Test the `getQuantity()` function

1. Retrieve and verify the quantity of an Item object.

Input:

Call the function on the first object.

OUTPUT / RESPONSE:

```
quantity = 8
```

3.2.6. Test the `getManufacturerID()` function

1. Retrieve and verify the manufacturerID of an Item object.

Input:

Call the function on the first object.

OUTPUT / RESPONSE:

```
manufacturerID = 1
```

3.2.7. Test the `getVehicleType()` function

1. Retrieve and verify the vehicleType of an Item object.

Input:

Call the function on the first object.

OUTPUT / RESPONSE:

```
vehicleType = PickupTruck
```

3.2.8. Test the `getStartDate()` function

1. Retrieve and verify the startDate of an Item object.

Input:

Call the function on the first object.

OUTPUT / RESPONSE:

```
startDate = 2021-04-03
```

3.2.9. Testing the `save()` function

1. Insert a new item to the database when it is empty

PRE CONDITION:

No item is present in the database.

Input:

Create a new object by calling the constructor with the following attributes, the constructor itself calls the `save()` method:

```
type = Battery
price = 6700
quantity = 8
manufacturerID = 1
vehicleType = PickupTruck
```

OUTPUT / RESPONSE:

```
The item should be added to the database. It can be verified by
querying the database to search for the item.
```

2. Insert a new item to the database when it is not empty

PRE CONDITION:

Ensure that one or more items are already present in the database.

Input:

Now, create a new object with the following attributes, the constructor itself calls the `save() method`:

```
type = Battery
price = 6700
quantity = 8
manufacturerID = 1
vehicleType = PickupTruck
```

OUTPUT / RESPONSE:

```
The item should be added to the database. It can be verified by
```

querying the database to search for the item.

3.2.10. Test the `delete()` function

1. Delete a item when multiple items are present in the inventory database

PRE CONDITION:

The following items are present in the database:

- `type = Battery`
  `price = 6700`
  `quantity = 8`
  `manufacturerID = 1`
  `vehicleType = PickupTruck`
- `type = Mirror`
  `price = 1700`
  `quantity = 12`
  `manufacturerID = 2`
  `vehicleType = Minivan`

Input:

Call the `delete()` method for the item with `uID 2`.

OUTPUT / RESPONSE:

The item should be deleted from the database. It can be verified by querying the database to see if it is present or not.

2. Delete a item when only a single item is present in the inventory database

PRE CONDITION:

Only a single item should be present in the database:

- `uID = 1`
  `type = Battery`
  `price = 6700`
  `quantity = 8`
  `manufacturerID = 1`
  `vehicleType = PickupTruck`

Input:

Call the `delete()` method for the item with `uID 1`.

OUTPUT / RESPONSE:

The item should be deleted from the database, and so the database will not have any more items.

3.2.11. Test the `updateSale(numSold)` function

1. Update the quantity of an item being sold

PRE CONDITION:

Suppose the state of an item initially is as follows:

```
uID = 1
type = Battery
price = 6700
quantity = 8
manufacturerID = 1
vehicleType = PickupTruck
```

Input:

Call the `updateSale()` method for this item with `numSold = 1`.

OUTPUT / RESPONSE:

Now, in the new state, the item object will have quantity = 7.

3.3. Testing the `Manufacturer class`

3.3.1. Testing the Constructor `Manufacturer(name, 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.

PRE CONDITION: for 3.3.2 to 3.3.5:

Suppose a Manufacturer has been added with the following attributes:

```
uID = 1
name = CEAT
address = Andheri West, Mumbai, Maharashtra
itemCount = 2
```

3.3.2. Test the `getUID()` function

1. Retrieve and verify the uID of a Manufacturer object.

Input:

Call the function on the Manufacturer object.

OUTPUT / RESPONSE:

```
uID = 1
```

3.3.3. Test the `getName()` function

1. Retrieve and verify the name of a Manufacturer object.

Input:

Call the function on the Manufacturer object.

OUTPUT / RESPONSE:

```
name = CEAT
```

3.3.4. Test the `getAddress()` function

1. Retrieve and verify the address of a Manufacturer object.

Input:

Call the function on the Manufacturer object.

OUTPUT / RESPONSE:

```
address = Andheri West, Mumbai, Maharashtra
```

3.3.5. Test the `getItemCount()` function

1. Retrieve and verify the itemCount of a Manufacturer object.

Input:

Call the function on the Manufacturer object.

OUTPUT / RESPONSE:

```
itemCount = 2
```

3.3.6. Test the `save()` function

1. Insert a new manufacturer to the database when it is empty

PRE CONDITION:

No manufacturer is yet present in the database.

Input:

Create a new `Manufacturer object` by calling the constructor with the following attributes, the constructor itself calls the `save()` method:

```
name = CEAT
address = Andheri West, Mumbai, Maharashtra
itemCount = 2
```

OUTPUT / RESPONSE:

```
The manufacturer should be added to the database. It can be
verified by querying the database to search for the manufacturer.
```

2. Insert a new manufacturer to the database when it is not empty

PRE CONDITION:

Ensure that one or more manufacturers are already present in the database.

Input:

Now, create a new `Manufacturer object` with the following attributes, the constructor itself calls the `save()` method:

```
name = TATA
address = Andheri West, Mumbai, Maharashtra
itemCount = 4
```

OUTPUT / RESPONSE:

```
The manufacturer should be added to the database. It can be
verified by querying the database to search for the manufacturer.
```

3.3.7. Test the delete() function

1. Delete a manufacturer when multiple manufacturers are present in the inventory database

PRE CONDITION:

The following manufacturers are present in the database:

- `uID = 1`
  `name = CEAT`
  `address = Andheri West, Mumbai, Maharashtra`
  `itemCount = 2`
- `uID = 2`
  `name = Tata`
  `address = Andheri West, Mumbai, Maharashtra`
  `itemCount = 4`

Input:

Call the `delete()` method for the manufacturer with `uID 2`.

OUTPUT / RESPONSE:

```
The manufacturer should be deleted from the database. It can be
verified by querying the database to see if it is present or not.
```

2. Delete a manufacturer when only a single manufacturer is present in the inventory database

PRE CONDITION:

Only a single item should be present in the database:

- `uID = 1`
  `name = CEAT`
  `address = Andheri West, Mumbai, Maharashtra`
  `itemCount = 2`

Input:

Call the `delete()` method for the manufacturer with `uID 1.`

OUTPUT / RESPONSE:

```
The manufacturer should be deleted from the database, and so now
the database will not contain any manufacturers.
```

3.4. Testing the `Inventory class`

3.4.1. Test the `retrieveData()` function

1. The inventory database is empty

Input:

Call the `retrieveData()` method.

OUTPUT / RESPONSE:

`The HashMaps searchMap, itemsList and manufacturersList should be empty.`

2. The inventory database is not empty

PRE CONDITION:

List of items present in the database

- `uID = 1`
  `type = Battery`
  `price = 6700`
  `quantity = 8`
  `manufacturerID = 1`
  `vehicleType = PickupTruck`
- `uID = 2`
  `type = Mirror`
  `price = 1700`
  `quantity = 12`
  `manufacturerID = 2`
  `vehicleType = Minivan`

List of manufacturer(s) present in the database:

- `uID = 1`
  `name = Tata`
  `address = Andheri West, Mumbai, Maharashtra`
  `itemCount = 2`

Input:

Call the retrieveData() method.

OUTPUT / RESPONSE:

```
searchMap.get("Battery").get(1).get("PickupTruck").uID = 1
searchMap.get("Battery").get(1).get("PickupTruck").type = Battery
searchMap.get("Battery").get(1).get("PickupTruck").price = 6700
searchMap.get("Battery").get(1).get("PickupTruck").quantity = 8
searchMap.get("Battery").get(1).get("PickupTruck").manufacturerID
= 1
searchMap.get("Battery").get(1).get("PickupTruck").vehicleType =
PickupTruck

itemsList.get(2).uID = 2
itemsList.get(2).type = Mirror
itemsList.get(2).price = 1700
itemsList.get(2).quantity = 12
itemsList.get(2).manufacturerID = 2
itemsList.get(2).vehicleType = Minivan

manufacturersList.get(1).uid = 1
manufacturersList.get(1).name = Tata
manufacturersList.get(1).address = Andheri West, Mumbai,
Maharashtra
```

```
manufacturersList.get(1).itemCount = 2
```

3.4.2. Test the `removeItem(itemID)` function

1. Remove an item when multiple items are present in the inventory database

PRE CONDITION:

List of items present in the inventory:

- `uID = 1`
  `type = Battery`
  `price = 6700`
  `quantity = 8`
  `manufacturerID = 1`
  `vehicleType = PickupTruck`
- `uID = 2`
  `type = Mirror`
  `price = 1700`
  `quantity = 12`
  `manufacturerID = 2`
  `vehicleType = Minivan`

Input:

Call the `removeItem()` method with `itemID = 2`.

OUTPUT / RESPONSE:

```
The item should be removed from the inventory.
```

New list of items in the inventory:

- `uID = 1`
  `type = Battery`
  `price = 6700`
  `quantity = 8`
  `manufacturerID = 1`
  `vehicleType = PickupTruck`

2. Remove an item when only a single item is present in the inventory

PRE CONDITION:

Only one item is present in the inventory:

- `uID = 1`
  `type = Battery`
  `price = 6700`
  `quantity = 8`
  `manufacturerID = 1`
  `vehicleType = PickupTruck`

Input:

Call the removeItem() method with itemID = 1.

OUTPUT / RESPONSE:

```
The item should be removed from the inventory. Now the list of
items in the inventory becomes empty.
```

3. The manufacturer of the item being deleted makes some other item(s) too.

PRE CONDITION:

List of items present in the inventory:

- `uID = 1`
  `type = Battery`
  `price = 6700`
  `quantity = 8`

```
  manufacturerID = 1
  vehicleType = PickupTruck
• uID = 2
  type = Mirror
  price = 1700
  quantity = 12
  manufacturerID = 2
  vehicleType = Minivan
```

List of manufacturer(s):

```
• uID = 1
  name = Tata
  address = Andheri West, Mumbai, Maharashtra
  itemCount = 2
```

Input:

Call the removeItem() method with itemID = 2.

OUTPUT / RESPONSE:

```
The item should be removed from the inventory.
```

New list of items in the inventory:

```
• uID = 1
  type = Battery
  price = 6700
  quantity = 8
  manufacturerID = 1
  vehicleType = PickupTruck
```

New list of manufacturer(s):

```
• uID = 1
  name = Tata
  address = Andheri West, Mumbai, Maharashtra
  itemCount = 1
```

4. The manufacturer of the item being deleted does not make any other item.

PRE CONDITION:

List of items present in the inventory:

```
• uID = 1
  type = Battery
  price = 6700
  quantity = 8
  manufacturerID = 1
  vehicleType = PickupTruck
• uID = 2
  type = Mirror
  price = 1700
  quantity = 12
  manufacturerID = 2
  vehicleType = Minivan
```

List of manufacturer(s):

```
• uID = 1
  name = Tata
  address = Andheri West, Mumbai, Maharashtra
  itemCount = 1
• uID = 2
```

```
  name = Michelin
  address = Andheri West, Mumbai, Maharashtra
  itemCount = 1
```

Input:

Call the `removeItem()` method with `itemID = 2`.

OUTPUT / RESPONSE:

```
The item should be removed from the inventory. New list of items
in the inventory : • uID = 1
type = Battery
price = 6700
quantity = 8
manufacturerID = 1
vehicleType = PickupTruck

New list of manufacturer(s):
• uID = 1
  name = Tata
  address = Andheri West, Mumbai, Maharashtra
  itemCount = 1
```