SDSU | San Diego State University

CS250 Introduction to Software Systems
Professor Miranda Parker

# Clothing Store Point of Sales System

## [Verification Test Plan]

*Created by:*

*Liam Carter*

*Bryce Jarboe*

*Andrea Ruvalcaba*

*(Group 2)*

## Introduction

This document will describe, in detail, individual test plans for different components of the *Clothing Store Point of Sales Software System*. Entries will be organized by a scope category (*Unit, Functional, and System*) and contain the following information: Function Name, Function Parameters, 2 Test Cases (each with a description of the test itself as well as an Expected Output), and Significance to the software system's functionality and overall operational capabilities.

*Note that return types are not listed seeing as all methods being tested are *void*, deeming them not applicable.

Refer to the *Software Design Specification* for an updated UML Diagram.

## Table of Contents

# Unit Tests

## Log-Out

**Function Name:**

LogOut

**Parameters:**

*None*

**Test Case 1:**

Use logIn to log into the system. Then use logOut.

**Expected Output:**

The user can no longer access the system functions except for the LogIn function. If the user can still interact with anything other than the logIn function the test is a failure.

**Significance:**

This function is important because only employees and admin should have access to this system. If anyone is able to access the system when an employee leaves their station, it could result in theft of items or improper use of the system.

**Test Case 2:**

Halfway through the payment process use the logOut function. Then log back in again to confirm the transaction is canceled.

**Expected Output:**

The user can no longer access the system functions except for the LogIn function. If the user can still interact with anything other than the logIn function the test is a failure. Also, the transaction should be canceled and upon logging back into the system, it should not appear again.

**Significance:**

The reason the transaction should be canceled is the customer does not want to pay for items twice. Also if any gift cards or coupons had been applied to the purchase before the payment, they need to be refunded too.

## Add-Item

**Function Name:**

addItem

**Parameters:**

Clothing& item

**Test Case 1:**

Call addItem with an arbitrary clothing item address. Then call viewInventory and verify the contents for the newly added item, validating the item's information.

**Expected Output:**

The item is viewable in the inventory, preserving all original data passed through the function.

**Test Case 2:**

Verify the inventory contents by calling viewInventory. Then call addItem with an invalid address. Call viewInventory again and verify the contents.

**Expected Output:**

An exception should be thrown and prevent any update to the inventory. Its contents should be unchanged since its previously checked state.

**Significance:**

The addItem function is essential to the operation of the software system as a whole. This is the primary method of adding malleable data to the inventory.

# Functional Tests

## View-Inventory

**Function Name:**

viewInventory

**Parameters:**

*None*

**Test Case 1:**

Use the addItem function to add an item to the inventory. Then, use viewInventory to check the inventory.

**Expected Output:**

Item was successfully added to the inventory and can be seen in the inventory.

**Test Case 2:**

Use the removeItem function to remove an item from the inventory. Then, use viewInventory to check the inventory.

**Expected Output:**

Item was successfully removed from the inventory and is no longer seen in the inventory.

**Significance**:

The viewInventory function needs to properly work in order for the inventory in stock to be 100% accurate. If an item is no longer in stock but still appears in the inventory, a customer could purchase an item that doesn't actually 'exist'. If viewInventory isn't working properly and items aren't being added, then this limits the customers inventory to choose to purchase from.

## Change-Date-Added

**Function Name:**

changeDateAdded

**Parameters:**

*None*

**Test Case 1:**

Go into viewInventory and select an item in the inventory. Select the changeDateAdded function and change the date added to the correct date.

**Expected Output:**

The date added for the item that was selected was properly changed to the user's input.

**Test Case 2:**

Add an item to inventory using the addItem function. The dateAdded should be equal to that day. Use the removeItem function to remove the item from inventory. Re-add the item to inventory and ensure dateAdded still shows that day.

**Expected Output:**

dateAdded will be equal to the current day.

**Significance**:

The dateAdded function needs to properly work because it helps the store know when a particular item was added to the inventory and to be able to know when some items may be too old and out of date to sell anymore or out of season.

## System Tests

### Print-Receipt

**Function Name:**

printReceipt

**Parameters:**

*None*

**Test Case 1:**

Login via the UI. Utilize the check-out's designated scanner on the test item (holding a price of $0). Then, through the payment terminal interface, proceed as instructed on screen and pay for the test item. Verify the contents of any receipts printed at the end of the transaction process.

**Expected Output:**

The procedure above should successfully yield calls for the following functions in order: logIn(), findItem(), viewInventory(), displayTotal(), displayItem(), pay(), printReceipt(), autoLogOut(). The verifiable output of this entire process is a receipt with the correct data printed, preserving the item's name, ID number, and price within the individual item transaction bulk of the print.

**Test Case 2:**

Login via the UI. Manually create test item entries (over 20) and add them (each with a $0 price) to the inventory. Utilize the manual item search and check out all test items. Then, through the payment terminal interface, proceed as instructed on screen and pay for the test items. Verify the contents of any receipts printed at the end of the transaction process.

**Expected Output:**

The procedure above should successfully yield calls for the following functions in order: logIn(), addItem(), findItem(IDNum), viewInventory(), displayTotal(), displayItem(), pay(), printReceipt(), autoLogOut(). The verifiable output of this entire process is a receipt with the correct data printed, preserving all items' names, ID numbers, and prices individually within the transaction bulk of the print.

**Significance:**

These tests rely on the cohesive functionality of the system as a whole, calling methods from each class to yield an output with verifiable data. Should any sector of the software system fail, the process will be hindered and no verifiable output will be generated.

## Item-Exists

**Function Name:**

itemExists

**Parameters:**

*None*

**Test Case 1:**

Login via the UI. Scan an item that belongs to the store's database. Check that the boolean variable storeItem is true. Double check that all information that is displayed by displayItem() matches the item that was scanned.

**Expected Output:**

The function itemExists should update the boolean variable storeItem to true since the item being scanned is chosen to be from the store. We then check the value of storeItem to confirm this info. We also double check that the correct item has been received from the inventory by using displayItem() to display the characteristics of the item and checking them against the item in question.

**Test Case 2:**

Login to the UI. Scan an item that is not an item from the store. Check that the boolean variable itemExists is false. Check to see if displayItem() displays an error message or not.

**Expected Output:**

The function itemExists() should update the boolean variable storeItem to be false because the item scanned will purposefully not be from the store. We expect there to be an error message when displayItem() is called because the item is not a store good.

**Significance**:

These tests are important for the system because it allows the users to know immediately if the item scanned is property of the store or not. While it is possible to check if the item is in the system by using other methods, this can alert the users to check the tag and make sure it is the correct one for the item at hand.

## Authorship

*Andrea Ruvalcaba*

Functional Test - viewInventory

Functional Test - changeDateAdded

*Bryce Jarboe*

Unit Test - addItem

System Test - printReceipt

*Liam Carter*

Updated UML Diagram

Unit Test - LogOut

System Test - itemExists