Design document

for

Stock, Exchange and Coupons (SEC)

Application

Laavanya Nagarajan

Contents

[1. Preamble 3](#_Toc468655249)

[1.1 Project Overview 3](#_Toc468655250)

[2. Interface Specifications 4](#_Toc468655251)

[1.2 Classes 4](#_Toc468655252)

[1.2.1 Convention 4](#_Toc468655253)

[1.2.2 MySQL databases and tables 4](#_Toc468655254)

[1.2.3 UI 8](#_Toc468655255)

[1.3 TestNG Tests 11](#_Toc468655256)

# Preamble

## Project Overview

This project “SEC” that stands for Stock, Expenses and Coupons is a minimal UI for maintaining my kitchen stock, expenses and the couponing system. “Stock” is any item that requires short-term and frequent purchases such as fruits, vegetables, toiletries. “Expenses”, is the expense incurred on these items. “Coupons” are the relevant coupons that are available with me. Though the three of them are different in their own sense, they are brought under one interface to make it easier for access, but kept distinctively separate as different pages for each.

* 1. Software Overview

The software is divided as 3 modules: ‘Stock’, ‘Expenses’ and ‘Coupons’, back-end database in MySQL to store the stocking, expense and couponing data and a GUI to let me update the required information on available stock, expense made or coupons available.

The ‘Stock’ maintenance is such that, the user (I) can add a new category of a stock item or update an existent stock item. The updation is the closing balance on each day, whenever there is a change. The closing balance may be higher or lower than what was available previously. So, both increase and decrease in stock are updated during the closing balance.

* 1. Purpose

The purpose of this document is to document the design of the ‘Stock’ module. This should also serve a reference for the design and implementations of ‘Expenses’ and ‘Coupons’. This document from here on talks only about the stock, except for the GUI which needs to mention about having tabs for the ‘Expenses’ and the ‘Coupons’ modules.

* 1. Requirements

1. Java 1.8 – Preferably version greater than Java Tiger (Java 5).
2. MySQL 5.6 or above.
   1. Document Outline

This document outlines the design decisions and specifications made for the development of the stock maintenance that will hereafter be referred to as ‘Stock’. The database tables and classes, the UI layout, the unit tests class for the module are described in the forthcoming sections.

# Interface Specifications

## Classes

Package: sec

### Convention

1. The stock item names are in general plural such as “Tomatoes” instead of “Tomato”.
2. The stock items are referred to in general as “item”.

### MySQL databases and tables

Database name: Stock.

The tables are as follows:

1. Stock
   1. Tables
      1. StockMaster – to contain and map each item to an id with its name.
         1. Fields
            1. ItemId – primary key, int, unsigned, auto\_increment, not null.
            2. ItemName – varchar(45), unique
            3. ItemCategory – varchar(45).
      2. ItemAddTransaction – required fields when a new item is added or an existent item stock is updated. Updation of an existent item can be more.
         1. Fields
            1. ItemId – int, not null. ItemQtyAdded – float
            2. ItemAddedOn – date
            3. ItemId and ItemAddedOn combined to form the primary key. This is for the daily closing balance.
            4. Foreign key: ID\_I on ItemId(corresponds to the ItemId on StockMaster).
      3. ItemStock – Updation table that will be updated with the current stock availability whenever there is a stock is added or exhausted.
         * 1. ItemId – int, unsigned, primary key. (The ItemId matching the one in StockMaster table).
           2. AvailableQty – float.F
           3. LastUpdated – date.
           4. Foreign key: ID on ItemId (corresponds to the ItemId on StockMaster).

#### Database Classes

1. Class Common
   1. Description: Defines all the constants required for the project such as database credentials, UI dimensions, UI Strings.
2. Class SecDBManager
   1. Class Methods
      1. Constructor.
         1. Description: Connects to the database.
      2. Private void setStockMaster()
         1. Description: Inserts data into StockMaster table.
      3. Private void setAddTransaction()
         1. Description: Inserts data into ItemAddTransaction table with current day’s date.
      4. Private void setAddTransaction().
         1. Description: Overloaded method. Makes an insert to the ItemAddTransaction for the provided date, given the month, day and year.
      5. Private void setAddTransaction().
         1. Description: Overloaded method. Makes and insert to the ItemAddTransaction for the provided date, given the date in sql date format. I was not sure while designing, if the date might need to use the sql date or need to pass the date as month, date and year. Hence this overloaded form.
      6. Private void setItemStock().
         1. Description: Inserts a row into the ItemStock table with the current day’s date.
      7. Private void setItemStock().
         1. Description: Overloaded method. Inserts a row into the ItemStock table for the date provided as day, month and year individually.
      8. Private void setItemStock().
         1. Description: Overloaded method. Inserts a row into the ItemStock table for the date provided as a date object itself.
      9. Private updateItemAddTransaction()
         1. Description: Makes an update to the ItemAddTransaction table for the item provided on a specific date with the input quantity.
      10. Private void upateItemStock().
          1. Description: Updates the table ItemStock with the provided quantity for the input date and item.
      11. Private void deleteItemStock() – FOR FUTURE REQUIREMENT.
          1. Description: Deletes the item from ItemStock table.
      12. Private void deleteItemFromTransaction() – FOR FUTURE REQUIREMENT.
          1. Description: Deletes the item from ItemAddTransaction.
      13. Private void deleteStockMaster() – FOR FUTURE REQUIREMENT.
          1. Description: Deletes the item from StockMaster.
      14. Private Date getLastDateUpdated().
          1. Description: Retrieves the lastUpdatedDate from ItemAddTransaction table.
      15. Public ResultSet getStockMasterData().
          1. Description – retrieves all the rows from StockMaster table.
      16. Public getStockFromItemId().
          1. Description – retrieves the name of the item given the itemId.
      17. Public int getStockIdFromName().
          1. Description – retrieves the stockId from itemName.
      18. Public String getItemCategory().
          1. Description – retrieves the category of the item, given its itemName.
      19. Public String getItemCategory().
          1. Description – overloaded method. Retrievs the category of the item, given the itemId.
      20. Private double getItemStock()
          1. Description – retrieves the availableqty from ItemStock, given its itemId.
      21. Public ArrayList<String> getAvailableItems()
          1. Description – retrieves the items that are still in stock, ie., those that have availableqty > 0.
      22. Public ResultSet getStockMasterTableFromName().
          1. Description – get row data from StockMaster given its name.
      23. Public ResultSet getStockItemsTableFromName().
          1. Description – get the row data from StockItems given an itemName.
      24. Public String[] getItemsFromCategory().
          1. Description – retrieves the list of items available on the input category.
      25. Public double getQtyFromTransactTable().
          1. Description – gets the quantity from ItemAddTransaction on a given date and itemId.
      26. Public void doAddTransaction().
          1. Description – Inserts data into tables on a new item addition.
      27. Public void doUpdateTransaction().
          1. Description – Does data update on relevant tables during a stock update.
      28. Public void doDeleteTransaction() – FOR FUTURE REQUIREMENT.
          1. Description – Deletes data from relevant tables on an item delete.

### UI

#### UI Layout

The UI layout is designed as follows:

1. The main frame has 3 tabs, Stock, Expenses and Coupons.
2. The ‘Stock’ tab contains a splitpane.
3. The left pane contains user selectable tree for ‘View Stock’ to show report on all available stock, ‘Add new item’, to add a new item to the stock and ‘Update Stock’.
4. The right pane shows report/form corresponding to the option selected on the left pane.

The following diagrams show the UI layout starting from the top-level frame.

Figure 1: Top level window

COUPONS

EXPENSES

STOCK

The top level window consists of 3 tabs. The Stock tab is selected by default. Each tab has a split panel, having a tree of selectable options on the left and the corresponding UI on its right.

The ‘Stock’ tab has the following left tree selection items.

Figure 2:The ' Stock' tab

Stock

View Stock Report

Add New Item

Update Stock

The following figures show the panels display on selection of each node respectively.

Figure 3: View Stock Report panel

|  |  |  |
| --- | --- | --- |
| Item | Quantity | Updated On |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Figure 4: Add New Item

Add New Item

Item Category <Multi-list selection>

Item Name <Text Box>

Item Quantity <Text Box>

Select Date <Date Selector>

<Add Stock Button>

Figure 5: Update Stock

Update Stock

Select Category <Multi-list selection>

Select Item <Text Box>

Quantity <Text Box>

Date <Date Selector>

<Update Stock button>

#### UI Classes

1. Class MainWindow
   1. Class Methods:
      1. initUI()
         1. // Layout – BorderLayout
         2. // Create a tabbed pane.
         3. // Initialize StockUI, ExpensesUI and CouponsUI to create specific tabs.
2. Class StockUI
   1. Class Variables
   2. Class Methods
      1. initStockTab()
         1. // Layout – BorderLayout
         2. // Create a split pane.
      2. showStockLeftTree()
         1. // Show left tree selection.
      3. showStockRightPanel()
         1. // Display the corresponding report table, add new stock panel, update stock panel corresponding to the selection on the left selection tree.
   3. Listener Classes
      1. SelectionListener implements TreeSelectionListener

## TestNG Tests

A unit test class for SecDBManager class.

1. The beforeMethod that involves inserting a record into stockmaster would throw and exception since the category field is unique. In order to check this one, a new data needs to be provded. But the tests would succeed without making any changes to the data.
2. public void doAddTransactionTest().
3. Public void doAddTransactionItemStockIdTest().
4. Public void doAddTransactionItemTransactionTest().
5. Public void updateItemStockTest().
6. Public void updateItemAddTransactionTableTest().