

# **BSc (Hons) Artificial Intelligence and Data Science**

**Module: CM1601 Programming Fundamentals**

**Coursework 2 Report**

**RGU Student ID : 2506755**

**IIT Student ID : 20240281**

**Student Name : W.D. Nisitha Nimsara**

# Contents

2.Flow charts.....	3
2.1. Add Function .....	3
2.3. Update Function.....	5
2.4. Dealer Selection and Items Functions.....	6
3.Introduction to functions with code .....	7
3.1. MainController.java .....	7
3.2. AddPageController.java.....	13
3.3. DeletePageController.java .....	18
3.4. UpdatePageController.java .....	23
3.5.ViewinventoryPageController.java.....	31
3.6. SelectdealersPageController .....	36
3.7. DealerItemsPageController.....	41
4.Test plan and test cases .....	46
4.1 Test cases .....	46
4.2 J-units Tests .....	50
5.Robustness and the maintainability .....	51
5.1. Robustness .....	51
5.2. Maintainability.....	51
6.Assumptions.....	51
7.Conclusion .....	52
8.Reference list .....	52
9.Appendices.....	53
9.1. Item.java.....	53
9.2. Dealer.java .....	56
9.3 DealerItem.java .....	56

## 2.Flow charts

### 2.1. Add Function

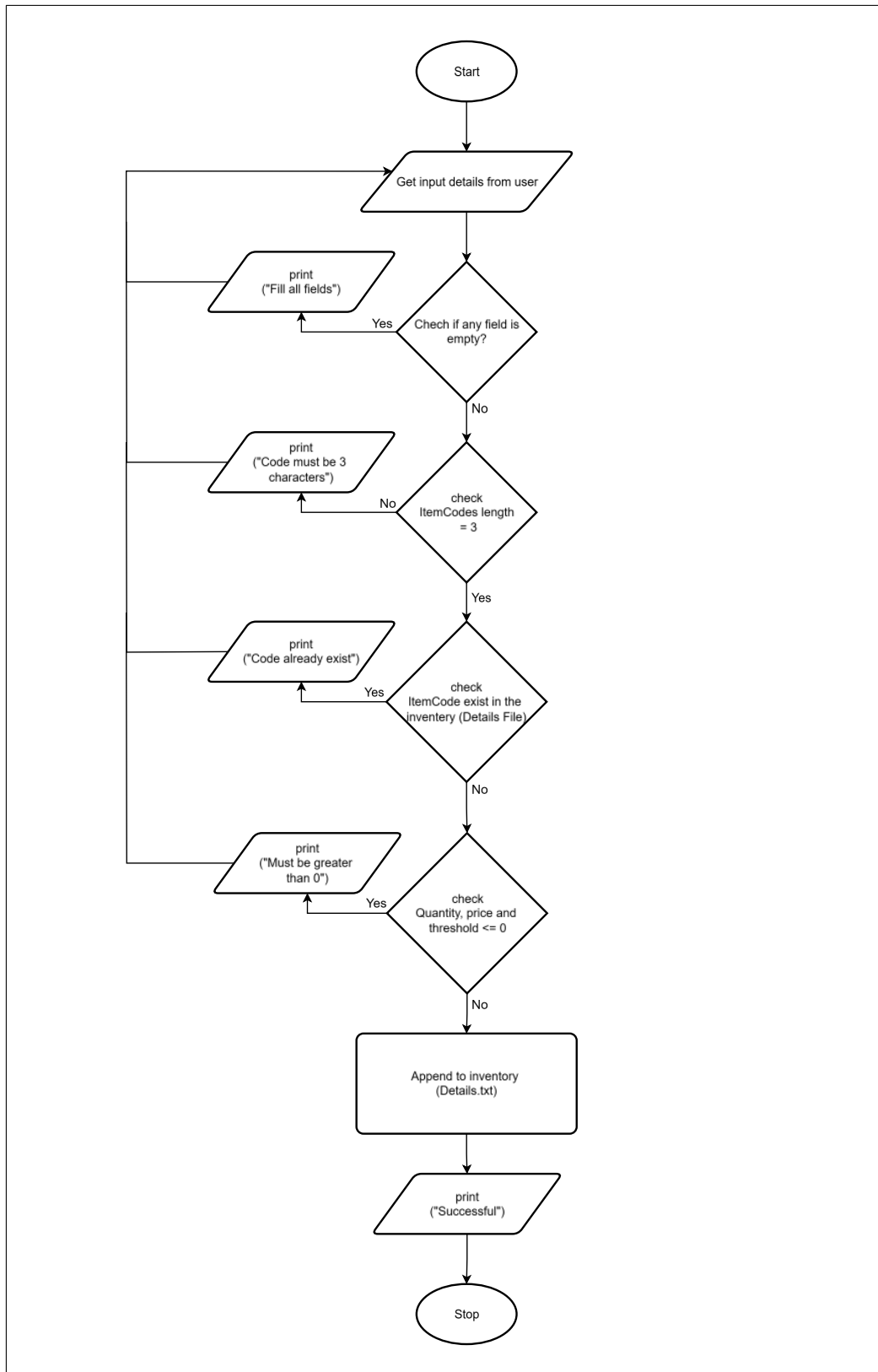


Figure 1 Add function flow-chart

## 2.2. Delete Function

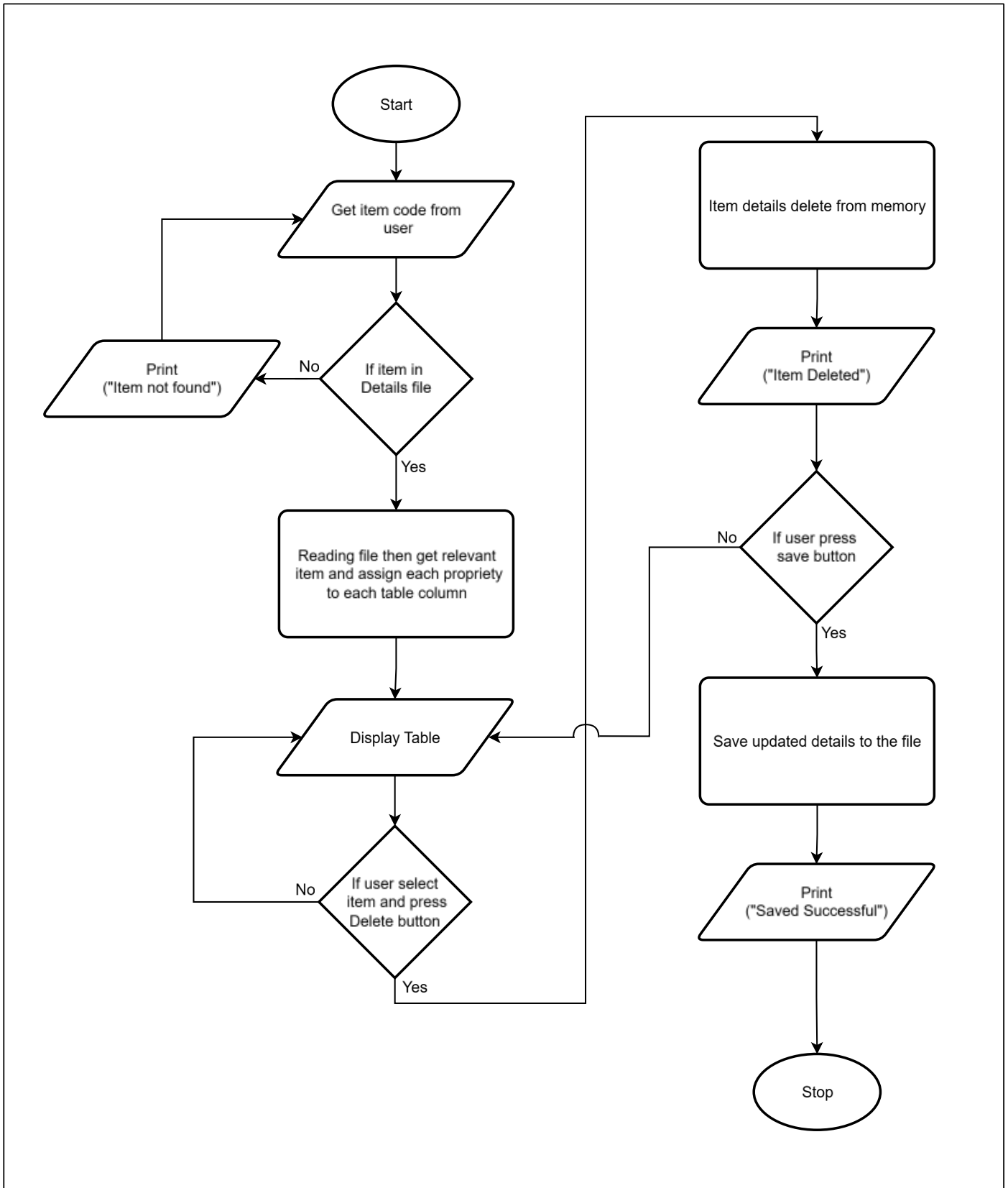


Figure 2 Delete function flow-chart

## 2.3. Update Function

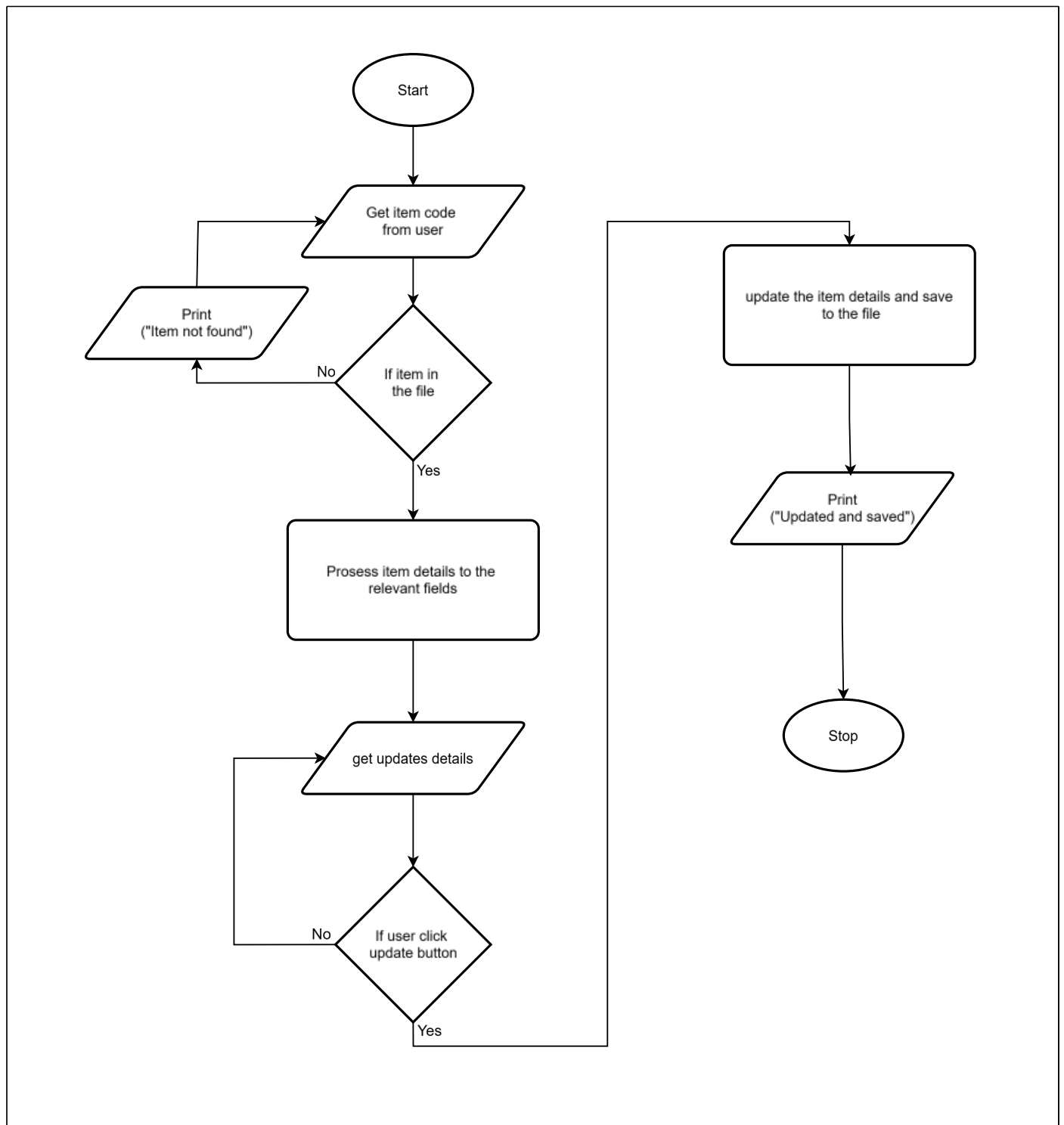


Figure 3 Update function flow-chart

## 2.4. Dealer Selection and Items Functions

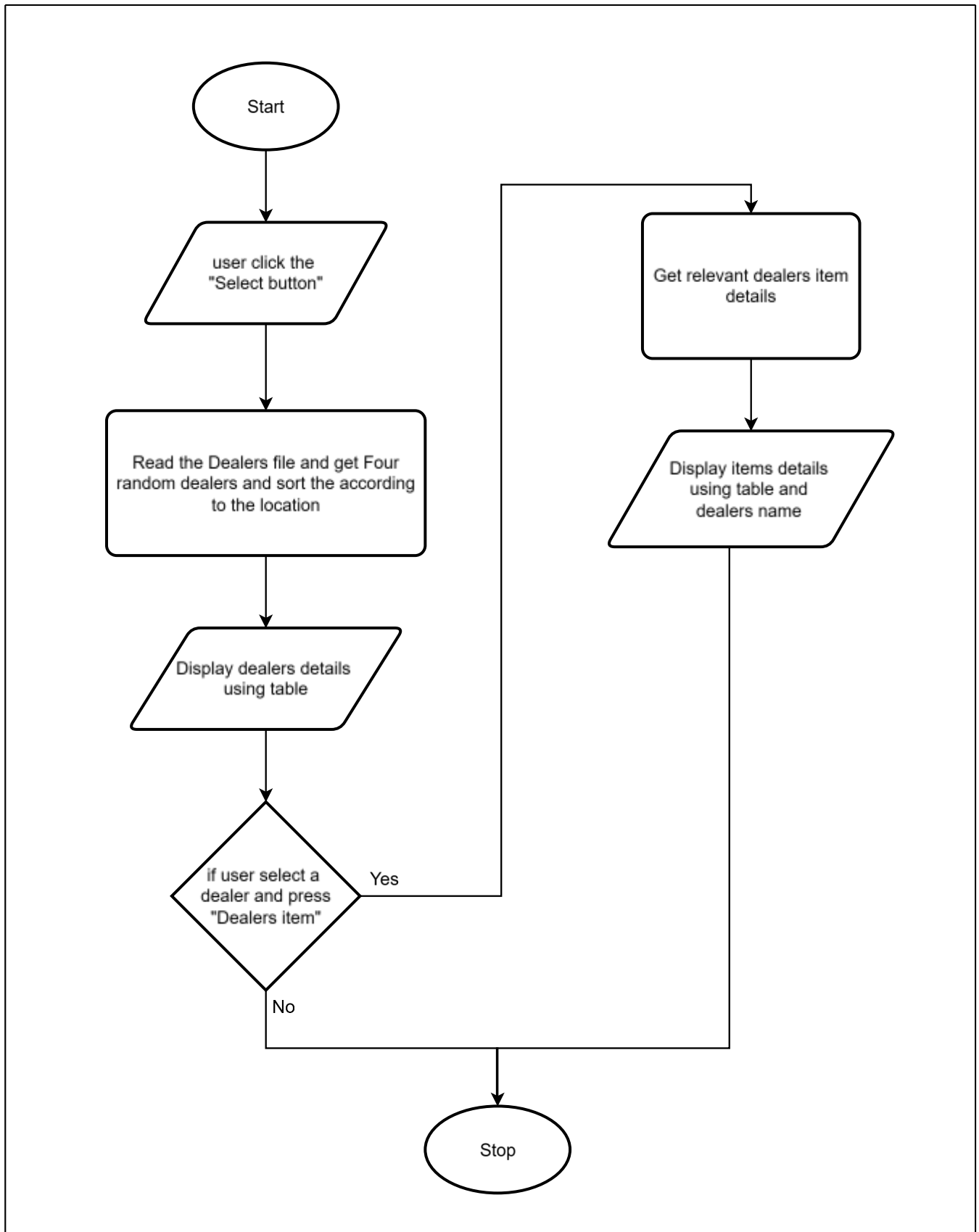


Figure 4 Dealers selection & items flow-chat

### 3.Introduction to functions with code

#### 3.1. MainController.java

The MainController class works as the central controller for the JavaFX-based main dashboard of the inventory management system developed for John's Supermarket. It provides two key functionalities: 1- displaying a table of low-stock items by reading and filtering data from a text file, and 2- handling navigation to other functional pages such as Add, Delete, Update, Inventory View, Dealer Selection, and Dealer Item View.

```
package org.example.johnsupermarket;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.stage.Stage;
import org.example.johnsupermarket.Models.Item;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.math.BigDecimal;
import java.net.URL;
import java.time.LocalDate;
import java.util.ResourceBundle;

public class MainController implements Initializable {
```

@FXML

```
private Stage stage;  
private Scene scene;
```

//this will close the window(Exit function)

```
public void handleExit(ActionEvent event) throws IOException {  
    stage = (Stage) ((Node) event.getSource()).getScene().getWindow();  
    stage.close();  
}
```

//this will navigate to Add function

```
public void GotoAddPage(ActionEvent event) throws IOException {  
    Parent root = FXMLLoader.load(getClass().getResource("addPage-view.fxml"));  
    stage = (Stage) ((Node) event.getSource()).getScene().getWindow();  
    scene = new Scene(root);  
    stage.setTitle("Add Page");  
    stage.setScene(scene);  
    stage.setResizable(false);  
    stage.show();  
}
```

//this will navigate to Delete function

```
public void GotoDIDPage(ActionEvent event) throws IOException {  
    Parent root = FXMLLoader.load(getClass().getResource("deletePage-view.fxml"));  
    stage = (Stage) ((Node) event.getSource()).getScene().getWindow();  
    scene = new Scene(root);  
    stage.setTitle("Delete Page");  
    stage.setScene(scene);  
    stage.setResizable(false);  
    stage.show();  
}
```

//this will navigate to update function

```
public void GotoUpdatePage(ActionEvent event) throws IOException {  
    Parent root = FXMLLoader.load(getClass().getResource("updatePage-view.fxml"));  
    stage = (Stage) ((Node) event.getSource()).getScene().getWindow();  
    scene = new Scene(root);
```



```
stage.setTitle("Update Page");
stage.setScene(scene);
stage.setResizable(false);
stage.show();
}
```

//this will navigate to inventory page

```
public void GotoInventory(ActionEvent event) throws IOException {
    FXMLLoader loader = new FXMLLoader(getClass().getResource("viewINVENPage-view.fxml"));
    Parent root = loader.load();

    ViewinventoryPageController controller = loader.getController();
    controller.refreshInventory();
}
```

```
stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
scene = new Scene(root);
stage.setTitle("Inventory Page");
stage.setScene(scene);
stage.show();
}
```

//this will navigate to Dealer select page

```
public void GotoSelectDealersPage(ActionEvent event) throws IOException {
    Parent root = FXMLLoader.load(getClass().getResource("randomDealersPage-view.fxml"));
    stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
    scene = new Scene(root);
    stage.setTitle("Dealers Page");
    stage.setScene(scene);
    stage.setResizable(false);
    stage.show();
}
```

//this will navigate to dealer's item page

```
public void GotoDealerItemsPage(ActionEvent event) throws IOException {
    Parent root = FXMLLoader.load(getClass().getResource("dealerItemsPage-view.fxml"));
    stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
}
```

```

scene = new Scene(root);
stage.setTitle("Dealer's Items Page");
stage.setScene(scene);
stage.setResizable(false);
stage.show();
}

```

//links with FXML

@FXML

```
private TableView<Item> lowStockTable;
```

```
private ObservableList<Item> itemList = FXCollections.observableArrayList();
```

@FXML

```
private TableColumn<Item, String> colCode;
```

@FXML

```
private TableColumn<Item, String> colName;
```

@FXML

```
private TableColumn<Item, String> colBrand;
```

@FXML

```
private TableColumn<Item, Double> colPrice;
```

@FXML

```
private TableColumn<Item, Integer> colQty;
```

@FXML

```
private TableColumn<Item, String> colCategory;
```

@FXML

```
private TableColumn<Item, String> colDate;
```

@FXML

```
private TableColumn<Item, Integer> colThreshold;
```

//load table columns

@Override

```

public void initialize(URL location, ResourceBundle resources){
    colCode.setCellValueFactory(new PropertyValueFactory<>("code"));
    colName.setCellValueFactory(new PropertyValueFactory<>("name"));
    colBrand.setCellValueFactory(new PropertyValueFactory<>("brand"));
    colPrice.setCellValueFactory(new PropertyValueFactory<>("price"));
    colQty.setCellValueFactory(new PropertyValueFactory<>("quantity"));
}

```

```

colCategory.setCellValueFactory(new PropertyValueFactory<>("category"));
colDate.setCellValueFactory(new PropertyValueFactory<>("date"));
colThreshold.setCellValueFactory(new PropertyValueFactory<>("threshold"));
loadLowstockTableData();

}

//method that uses to load low stock table
private void loadLowstockTableData() {
    //this will clear the table data (if exist)
    itemList.clear();
    //this will open and read the inventory file
    try (BufferedReader br = new BufferedReader(new
FileReader("src/main/resources/Inventory/details.txt"))) {
        //declaration
        String line;
        //check the nullness
        while ((line = br.readLine()) != null) {
            //creating String array using ","
            String[] data = line.split(",");
            //check array length (because if the array dosen't have 9 parts means its invalid)
            if (data.length == 9) {
                //value assigning
                BigDecimal price = new BigDecimal(data[3]);
                int quantity = Integer.parseInt(data[4]);
                int threshold = Integer.parseInt(data[7]);
                //check quantity is less than the threshold
                if (quantity < threshold) {
                    //if "yes" creat ITEM obj
                    Item item = new Item(
                        data[0],
                        data[1],
                        data[2],
                        price,
                        quantity,
                        data[5],

```

```

        LocalDate.parse(data[6]),
        threshold,
        data[8]
    );
    //add to the itemlist array
    itemList.add(item);
}
}
}
//Sorting (bubble sort) according to the category
for (int i = 0; i < itemList.size() - 1; i++) {
    for (int j = 0; j < itemList.size() - i - 1; j++) {
        Item item1 = itemList.get(j);
        Item item2 = itemList.get(j + 1);

        int categoryCompare = item1.getCategory().compareToIgnoreCase(item2.getCategory());

        if (categoryCompare > 0 || (categoryCompare == 0 &&
            item1.getCode().compareToIgnoreCase(item2.getCode()) > 0)) {
            itemList.set(j, item2);
            itemList.set(j + 1, item1);
        }
    }
}
//then pass to lowStock table
lowStockTable.setItems(itemList);
} catch (Exception e) {
    e.printStackTrace();
}
}
}

```

### 3.2. AddPageController.java

The AddPageController class manages the add item functionality of John's Supermarket. It allows users to input item details and then validate those and add(append) to the inventory file, which acts as the system's persistent storage.

```
package org.example.johnsupermarket;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.DatePicker;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.paint.Color;
import javafx.stage.FileChooser;
import javafx.stage.Stage;

import java.io.*;
import java.math.BigDecimal;
import java.time.LocalDate;

public class AddPageController {

    //links between addPage-view.fxml and controller
    @FXML private TextField AddItemCode;
    @FXML private TextField AddItemName;
    @FXML private TextField AddItemBrand;
    @FXML private TextField AddItemPrice;
    @FXML private TextField AddItemQuantity;
    @FXML private TextField AddItemCategory;
    @FXML private TextField AddItemThreshold;
```

```
@FXML private DatePicker AddItemPurchDate;  
@FXML private Button browseImageButton;  
@FXML private Label OutputPanel;  
@FXML private TextField AddItemImagePath;
```

```
private Stage stage;  
private Scene scene;
```

```
//runs after the FXML is loaded
```

```
public void initialize() {  
    browseImageButton.setOnAction(e -> browseForImage());  
}
```

```
//clear all input fields
```

```
@FXML
```

```
private void clearFields() {  
    AddItemCode.clear();  
    AddItemName.clear();  
    AddItemBrand.clear();  
    AddItemPrice.clear();  
    AddItemQuantity.clear();  
    AddItemCategory.clear();  
    AddItemThreshold.clear();  
    AddItemPurchDate.setValue(null);  
    AddItemImagePath.clear();  
}
```

```
//browse button for image selector
```

```
private void browseForImage() {  
    FileChooser fileChooser = new FileChooser();  
    fileChooser.setTitle("Select Item Image");  
    fileChooser.getExtensionFilters().addAll(  
        new FileChooser.ExtensionFilter("Image Files", "*.png", "*.jpg", "*.jpeg", "*.gif", "*.bmp"),  
        new FileChooser.ExtensionFilter("All Files", "*.*")  
    );  
    File selectedFile = fileChooser.showOpenDialog(null);
```

```

//if selected image is not available in the resource/image folder, create a copy
if (selectedFile != null) {
    try {
        String fileName = selectedFile.getName();
        File destDir = new File("src/main/resources/images/");
        if (!destDir.exists()) destDir.mkdirs();

        File destFile = new File(destDir, fileName);
        java.nio.file.Files.copy(selectedFile.toPath(), destFile.toPath(),
            java.nio.file.StandardCopyOption.REPLACE_EXISTING);

        AddItemImagePath.setText(fileName);
    } catch (IOException e) {
        OutputPanel.setText("Failed to copy image.");
    }
}
}

```

//Main save button logic

@FXML

```

public void onSaveButtonClick() {
    try {
        //assigning inputs that came from fxml file
        String code = AddItemCode.getText().trim();
        String name = AddItemName.getText().trim();
        String brand = AddItemBrand.getText().trim();
        BigDecimal price = new BigDecimal(AddItemPrice.getText().trim());
        int quantity = Integer.parseInt(AddItemQuantity.getText().trim());
        String category = AddItemCategory.getText().trim();
        LocalDate date = AddItemPurchDate.getValue();
        String imagePath = AddItemImagePath.getText();
        int threshold = Integer.parseInt(AddItemThreshold.getText().trim());

        //this will check that all the input fields are filed?
        if (code.isEmpty() || name.isEmpty() || brand.isEmpty() || category.isEmpty() ||
            AddItemPrice.getText().trim().isEmpty() || AddItemQuantity.getText().trim().isEmpty() ||

```

```

        date == null || AddItemThreshold.getText().trim().isEmpty() || imagePath.isEmpty()) {
    OutputPanel.setText("Please fill all required fields.");
    return;
}
//this will check the item code's validation
if (code.length() != 3) {
    OutputPanel.setText("Invalid code. Must be 3 characters.");
    return;
}
//this will avoid the duplicate item codes
File file = new File("src/main/resources/Inventory/details.txt");
if (file.exists()) {
    BufferedReader reader = new BufferedReader(new FileReader(file));
    String line;
    while ((line = reader.readLine()) != null) {
        String[] parts = line.split(",", -1);
        if (parts.length > 0 && parts[0].equals(code)) {
            OutputPanel.setText("Item code already exists.");
            OutputPanel.setTextFill(Color.RED);
            reader.close();
            return;
        }
    }
    reader.close();
}
//this will avoid negative numbers and 0
if (price.compareTo(BigDecimal.ZERO) <= 0) {
    OutputPanel.setText("Price must be greater than 0.");
    return;
}
//this will avoid negative numbers and 0
if (quantity <= 0) {
    OutputPanel.setText("Quantity must be greater than 0.");
    return;
}
if (threshold <= 0) {

```



```

        OutputPanel.setText("Threshold must be greater than 0.");
        return;
    }
    //append item to file
    BufferedWriter writer = new BufferedWriter(new
FileWriter("src/main/resources/Inventory/details.txt", true));
        writer.write(String.join(", ", escape(code), escape(name), escape(brand), price.toString(),
            String.valueOf(quantity), escape(category), date.toString(), String.valueOf(threshold),
escape(imagePath)));
        writer.newLine();
        writer.close();
    //send to user a message
    OutputPanel.setText("Item added successfully.");
    OutputPanel.setTextFill(Color.GREEN);
    clearFields();

    } catch (NumberFormatException e) {
        OutputPanel.setText("Invalid numeric input.");
    } catch (Exception e) {
        OutputPanel.setText("Error: " + e.getMessage());
    }
}
//helper method to escape commas
private String escape(String input) {
    return input.replace(",", "\\,");
}
//navigate to home page
public void GotoHome(ActionEvent event) throws IOException {
    Parent root = FXMLLoader.load(getClass().getResource("main-view.fxml"));
    stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
    scene = new Scene(root);
    stage.setTitle("Main Page");
    stage.setScene(scene);
    stage.show();
}
//navigate to inventory page

```

```

public void GotoInventory(ActionEvent event) throws IOException {
    FXMLLoader loader = new FXMLLoader(getClass().getResource("viewINVENPage-view.fxml"));
    Parent root = loader.load();

    ViewinventoryPageController controller = loader.getController();
    controller.refreshInventory();

    stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
    scene = new Scene(root);
    stage.setTitle("Inventory Page");
    stage.setScene(scene);
    stage.show();
}

//close the app
public void handleExit(ActionEvent event) {
    stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
    stage.close();
}
}

```

### 3.3. DeletePageController.java

The DeletePageController class manages the delete items of John's Supermarket system. It provides a searchable interface for finding items by their code, displaying their details in a table, and allowing users to remove selected items from the dataset and stored in details file.

```

package org.example.johnsupermarket;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Label;

```

```

import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.TextField;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
import org.example.johnsupermarket.Models.Item;

import java.io.*;
import java.math.BigDecimal;
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

public class DeletePageController {

    //links between deletePage-view.fxml and the controller
    @FXML private TableView<Item> itemTable;
    @FXML private TableColumn<Item, String> codeCol, nameCol, brandCol, categoryCol, imageCol;
    @FXML private TableColumn<Item, BigDecimal> priceCol;
    @FXML private TableColumn<Item, Integer> quantityCol, thresholdCol;
    @FXML private TableColumn<Item, LocalDate> dateCol;
    @FXML private Label OutputPanel;
    @FXML private TextField itemCodeInput;

    private Stage stage;
    private Scene scene;

    //list for hold items
    private List<Item> allItems = new ArrayList<>();
    //search for item by code(method)
    @FXML
    private void handleSearchItem() {
        String code = itemCodeInput.getText().trim();
        //if item code panel empty

```

```

if (code.isEmpty()) {
    OutputPanel.setText("Enter item code to search.");
    OutputPanel.setTextFill(Color.RED);
    return;
}
//find the relevant details of given item code
List<Item> found = allItems.stream()
    .filter(item -> item.getCode().equalsIgnoreCase(code))
    .collect(Collectors.toList());
if (found.isEmpty()) {
    OutputPanel.setText("Item not found.");
    OutputPanel.setTextFill(Color.RED);
    itemTable.getItems().clear();
} else {
    itemTable.getItems().setAll(found);
    OutputPanel.setText("Item found.");
    OutputPanel.setTextFill(Color.GREEN);
}
}

//load items data to the itemlist
private List<Item> loadItemsFromFile(String path) throws IOException {
    List<Item> list = new ArrayList<>();
    try (BufferedReader br = new BufferedReader(new FileReader(path))) {
        String line;
        int lineNumber = 0;

        while ((line = br.readLine()) != null) {
            lineNumber++;
            if (line.trim().isEmpty()) continue;
            String[] parts = line.split(",", -1);
            if (parts.length != 9) {
                OutputPanel.setText("Line " + lineNumber + ": Wrong number of fields.");
                continue;
            }
            //assigning each value to the item by order
            try {

```

```

        String code = parts[0].trim();
        String name = parts[1].trim();
        String brand = parts[2].trim();
        BigDecimal price = new BigDecimal(parts[3].trim());
        int quantity = Integer.parseInt(parts[4].trim());
        String category = parts[5].trim();
        LocalDate date = LocalDate.parse(parts[6].trim());
        int threshold = Integer.parseInt(parts[7].trim());
        String imagePath = parts[8].trim();
        //make it as require order
        list.add(new Item(code, name, brand, price, quantity, category, date, threshold, imagePath));
    } catch (Exception e) {
        OutputPanel.setText("Error at line " + lineNumber + ": " + e.getMessage());
    }
}
}
return list;
}

//set table columns to values
public void initialize() {
    codeCol.setCellValueFactory(new PropertyValueFactory<>("code"));
    nameCol.setCellValueFactory(new PropertyValueFactory<>("name"));
    brandCol.setCellValueFactory(new PropertyValueFactory<>("brand"));
    priceCol.setCellValueFactory(new PropertyValueFactory<>("price"));
    quantityCol.setCellValueFactory(new PropertyValueFactory<>("quantity"));
    categoryCol.setCellValueFactory(new PropertyValueFactory<>("category"));
    dateCol.setCellValueFactory(new PropertyValueFactory<>("date"));
    thresholdCol.setCellValueFactory(new PropertyValueFactory<>("threshold"));
    imageCol.setCellValueFactory(new PropertyValueFactory<>("imagePath"));

    try {
        //getting details from details file
        allItems = loadItemsFromFile("src/main/resources/Inventory/details.txt");
    } catch (IOException e) {
        OutputPanel.setText("Error loading file.");
        OutputPanel.setTextFill(Color.RED);
    }
}

```

```

    }
}

//delete item from table
@FXML
private void handleDeleteItem() {
    Item selected = itemTable.getSelectionModel().getSelectedItem();
    if (selected == null) {
        OutputPanel.setText("Select item in table to delete.");
        OutputPanel.setTextFill(Color.RED);
        return;
    }

    allItems.removeIf(item -> item.getCode().equals(selected.getCode()));
    itemTable.getItems().clear();
    OutputPanel.setText("Item deleted.");
    OutputPanel.setTextFill(Color.ORANGE);
}

//save all items back to file
@FXML
private void handleSaveItems() {
    try (BufferedWriter bw = new BufferedWriter(new
FileWriter("src/main/resources/Inventory/details.txt"))) {
        for (Item item : allItems) {
            String line = String.join(" ",
                item.getCode(),
                item.getName(),
                item.getBrand(),
                item.getPrice().toString(),
                String.valueOf(item.getQuantity()),
                item.getCategory(),
                item.getDate().toString(),
                String.valueOf(item.getThreshold()),
                item.getImagePath()
            );

```

```

        bw.write(line);
        bw.newLine();
    }
    OutputPanel.setText("Item deleted and Saved successfully.");
    OutputPanel.setTextFill(Color.GREEN);
} catch (IOException e) {
    OutputPanel.setText("Error saving file.");
    OutputPanel.setTextFill(Color.RED);
}
}

//navigate to home page
public void GotoHome(ActionEvent event) throws IOException {
    Parent root = FXMLLoader.load(getClass().getResource("main-view.fxml"));
    stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
    scene = new Scene(root);
    stage.setTitle("Main Page");
    stage.setScene(scene);
    stage.show();
}

//close the app
public void handleExit(ActionEvent event) throws IOException {
    stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
    stage.close();
}
}

```

### 3.4. UpdatePageController.java

The UpdatePageController class handles the updating functionality of John's Supermarket. It provides an interactive interface for loading item data based on item code then it facilitates modifying relevant fields and update those changes back into the details file.

```

package org.example.johnsupermarket;
import javafx.event.ActionEvent;

```

```
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.DatePicker;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.paint.Color;
import javafx.stage.FileChooser;
import javafx.stage.Stage;
import org.example.johnsupermarket.Models.Item;
```

```
import java.io.*;
import java.math.BigDecimal;
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;
```

```
public class UpdatePageController {
    @FXML private Stage stage;
    private Scene scene;

    //links between updatePage-view.fxml and controller
    @FXML private TextField codeInput;
    @FXML private TextField nameInput;
    @FXML private TextField brandInput;
    @FXML private TextField priceInput;
    @FXML private TextField quantityInput;
    @FXML private TextField categoryInput;
    @FXML private TextField thresholdInput;
    @FXML private TextField imagePathInput;
    @FXML private DatePicker dateInput;
    @FXML private Label OutputPanel;

    //declaration
    private List<Item> allItems = new ArrayList<>();
```



```

private Item selectedItem = null;
//load when call the function
@FXML
public void initialize() {
    loadItemsFromFile();
}
//-----get items from file
private void loadItemsFromFile() {
    allItems.clear();
    //open new file obj
    File file = new File("src/main/resources/Inventory/details.txt");
    if (!file.exists()) {
        System.out.println("No inventory file found.");
        return;
    }

    try (BufferedReader br = new BufferedReader(new FileReader(file))) {
        String line;
        //read line by line
        while ((line = br.readLine()) != null) {
            String[] parts = line.split(",");
            //make the item object
            if (parts.length == 9) {
                Item item = new Item(
                    parts[0], parts[1], parts[2],
                    new BigDecimal(parts[3]),
                    Integer.parseInt(parts[4]),
                    parts[5],
                    LocalDate.parse(parts[6]),
                    Integer.parseInt(parts[7]),
                    parts[8]
                );
                //add to the list(all)
                allItems.add(item);
            }
        }
    }
}

```

```

    } catch (IOException e) {
        System.out.println("Error reading inventory file: " + e.getMessage());
    }
}

//search button method
@FXML
private void handleSearchItem() {
    //getting item code
    String code = codeInput.getText().trim();
    if (code.isEmpty()) {
        OutputPanel.setText("Enter item code.");
        OutputPanel.setTextFill(Color.RED);
        return;
    }
    //find the item details relate to the item code
    selectedItem = allItems.stream()
        .filter(item -> item.getCode().equalsIgnoreCase(code))
        .findFirst()
        .orElse(null);
    //null handle
    if (selectedItem == null) {
        OutputPanel.setText("Item not found.");
        OutputPanel.setTextFill(Color.RED);
        clearInputFields();
        return;
    }
    //load the item details to related fields(in fxml)
    nameInput.setText(selectedItem.getName());
    brandInput.setText(selectedItem.getBrand());
    priceInput.setText(selectedItem.getPrice().toString());
    quantityInput.setText(String.valueOf(selectedItem.getQuantity()));
    categoryInput.setText(selectedItem.getCategory());
    dateInput.setValue(selectedItem.getDate());
    thresholdInput.setText(String.valueOf(selectedItem.getThreshold()));
    imagePathInput.setText(selectedItem.getImagePath());
    //for user to view

```

```

OutputPanel.setText("Item loaded.");
OutputPanel.setTextFill(Color.GREEN);
}
//update and save method
@FXML
private void handleUpdateSave() {
    if (selectedItem == null) {
        OutputPanel.setText("No item selected to update.");
        OutputPanel.setTextFill(Color.RED);
        return;
    }
    //get updated/not details from each field
    String name = nameInput.getText().trim();
    String brand = brandInput.getText().trim();
    String priceText = priceInput.getText().trim();
    String quantityText = quantityInput.getText().trim();
    String category = categoryInput.getText().trim();
    LocalDate date = dateInput.getValue();
    String thresholdText = thresholdInput.getText().trim();
    String imagePath = imagePathInput.getText().trim();
    //check the field's completeness
    if (name.isEmpty() || brand.isEmpty() || priceText.isEmpty() || quantityText.isEmpty()
        || category.isEmpty() || date == null || thresholdText.isEmpty() || imagePath.isEmpty()) {
        OutputPanel.setText("All fields must be filled.");
        OutputPanel.setTextFill(Color.RED);
        return;
    }
    BigDecimal price;
    int quantity;
    int threshold;
    try {
        price = new BigDecimal(priceText);
        quantity = Integer.parseInt(quantityText);
        threshold = Integer.parseInt(thresholdText);
    } catch (NumberFormatException e) {
        OutputPanel.setText("Invalid number input.");
    }
}

```

```

        OutputPanel.setTextFill(Color.RED);
        return;
    }
    //make the item object
    selectedItem = new Item(
        selectedItem.getCode(), name, brand, price, quantity, category, date, threshold, imagePath
    );
    for (int i = 0; i < allItems.size(); i++) {
        if (allItems.get(i).getCode().equalsIgnoreCase(selectedItem.getCode())) {
            allItems.set(i, selectedItem);
            break;
        }
    }
    //update(write) in the details file
    try (BufferedWriter bw = new BufferedWriter(new
    FileWriter("src/main/resources/Inventory/details.txt"))) {
        for (Item item : allItems) {
            String line = String.join(" ",
                item.getCode(), item.getName(), item.getBrand(),
                item.getPrice().toString(),
                String.valueOf(item.getQuantity()), item.getCategory(),
                item.getDate().toString(), String.valueOf(item.getThreshold()),
                item.getImagePath()
            );
            bw.write(line);
            bw.newLine();
        }
        OutputPanel.setText("Item updated and saved.");
        OutputPanel.setTextFill(Color.GREEN);
    } catch (IOException e) {
        OutputPanel.setText("Error saving file.");
        OutputPanel.setTextFill(Color.RED);
    }
}
//browse button method
@FXML

```

```

private void handleBrowseImage() {
    //select image from computer
    FileChooser fileChooser = new FileChooser();
    fileChooser.setTitle("Select Item Image");
    fileChooser.getExtensionFilters().addAll(
        new FileChooser.ExtensionFilter("Image Files", "*.png", "*.jpg", "*.jpeg", "*.gif", "*.bmp"),
        new FileChooser.ExtensionFilter("All Files", "*.*")
    );

    File selectedFile = fileChooser.showOpenDialog(null);
    if (selectedFile != null) {
        try {
            String fileName = selectedFile.getName();
            File destDir = new File("src/main/resources/images/");
            //create a copy (If not exist in the images)
            if (!destDir.exists()) destDir.mkdirs();
            File destFile = new File(destDir, fileName);
            java.nio.file.Files.copy(selectedFile.toPath(), destFile.toPath(),
java.nio.file.StandardCopyOption.REPLACE_EXISTING);
            //pass to imgeth(on fxml)
            imagePathInput.setText(fileName);
            OutputPanel.setText("Image selected: " + fileName);
            OutputPanel.setTextFill(Color.GREEN);
        } catch (IOException e) {
            OutputPanel.setText("Image copy failed.");
            OutputPanel.setTextFill(Color.RED);
        }
    }
}

//clear field method
private void clearInputFields() {
    nameInput.clear();
    brandInput.clear();
    priceInput.clear();
    quantityInput.clear();
    categoryInput.clear();
}

```

```

        dateInput.setValue(null);
        thresholdInput.clear();
        imagePathInput.clear();
    }

    //exit the app
    public void handleExit(ActionEvent event) throws IOException {
        stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
        stage.close();
    }

    //navigate to inventory
    public void GotoInventory(ActionEvent event) throws IOException {
        FXMLLoader loader = new FXMLLoader(getClass().getResource("viewINVENPage-view.fxml"));
        Parent root = loader.load();

        ViewinventoryPageController controller = loader.getController();
        controller.refreshInventory();

        stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
        scene = new Scene(root);
        stage.setTitle("Inventory Page");
        stage.setScene(scene);
        stage.show();
    }

    //navigate to the home
    public void GotoHome(ActionEvent event) throws IOException {
        Parent root = FXMLLoader.load(getClass().getResource("main-view.fxml"));
        stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
        scene = new Scene(root);
        stage.setTitle("Home Page");
        stage.setScene(scene);
        stage.show();
    }
}

```

### 3.5.ViewinventoryPageController.java

The ViewinventoryPageController class is responsible for displaying inventory table in john's supermarket. It allows users to visually browse stock data (sort by category and each category by item code), view product images, calculate overall inventory item count and total inventory value.

```
package org.example.johnsupermarket;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.TableCell;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.TextField;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.stage.Stage;
import org.example.johnsupermarket.Models.Item;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.math.BigDecimal;
import java.math.RoundingMode;
import java.time.LocalDate;
```

```
public class ViewinventoryPageController {
    @FXML
```

```

private Stage stage;

private Scene scene;

//link between viewINVENPage-view.fxml and the controller
@FXML private TableView<Item> inventoryTable;
@FXML private TableColumn<Item, String> colCode;
@FXML private TableColumn<Item, String> colName;
@FXML private TableColumn<Item, String> colBrand;
@FXML private TableColumn<Item, Double> colPrice;
@FXML private TableColumn<Item, Integer> colQty;
@FXML private TableColumn<Item, String> colCategory;
@FXML private TableColumn<Item, String> colDate;
@FXML private TableColumn<Item, Integer> colThreshold;
@FXML private TableColumn<Item, String> colImage;
@FXML private TextField TotalItemID;
@FXML private TextField TotalValueID;

private final ObservableList<Item> itemList = FXCollections.observableArrayList();

//this will call the inside methods to run when function called
@FXML
public void initialize() {
    setupColumns();
    refreshInventory();
}

//set columns
private void setupColumns() {
    colCode.setCellValueFactory(new PropertyValueFactory<>("code"));
    colName.setCellValueFactory(new PropertyValueFactory<>("name"));
    colBrand.setCellValueFactory(new PropertyValueFactory<>("brand"));
    colPrice.setCellValueFactory(new PropertyValueFactory<>("price"));
    colQty.setCellValueFactory(new PropertyValueFactory<>("quantity"));
    colCategory.setCellValueFactory(new PropertyValueFactory<>("category"));
    colDate.setCellValueFactory(new PropertyValueFactory<>("date"));
    colThreshold.setCellValueFactory(new PropertyValueFactory<>("threshold"));
    colImage.setCellValueFactory(new PropertyValueFactory<>("imagePath"));

```



```

//image column
collImage.setCellFactory(column -> new TableCell<Item, String>() {
    private final ImageView imageView = new ImageView();
    {
        imageView.setFitHeight(50);
        imageView.setFitWidth(50);
        imageView.setPreserveRatio(true);
    }
    @Override
    protected void updateItem(String imagePath, boolean empty) {
        super.updateItem(imagePath, empty);
        if (empty || imagePath == null || imagePath.trim().isEmpty()) {
            setGraphic(null);
        } else {
            try {
                Image image = new Image(getClass().getResourceAsStream("/images/" + imagePath));
                imageView.setImage(image);
                setGraphic(imageView);
            } catch (Exception e) {
                setGraphic(null);
            }
        }
    }
});
}

//load data from the file
private void loadInventoryData() {
    itemList.clear();
    BigDecimal totalValue = BigDecimal.ZERO;
    int totalItems = 0;
    //read the file
    try (BufferedReader br = new BufferedReader(new
FileReader("src/main/resources/Inventory/details.txt"))) {
        String line;
        //line by line reading
        while ((line = br.readLine()) != null) {

```

```

String[] data = line.split(",");
Item item = null;
//if valid item,
if (data.length == 9) {
    item = new Item(
        data[0], // code
        data[1], // name
        data[2], // brand
        new BigDecimal(data[3]), // price
        Integer.parseInt(data[4]), // quantity
        data[5], // category
        LocalDate.parse(data[6]), // date
        Integer.parseInt(data[7]), // threshold
        data[8] // image
    );
    //add to item arraylist
    itemList.add(item);
}
if (item != null) {
    totalItems++;

    BigDecimal quantityBD = BigDecimal.valueOf(item.getQuantity());
    BigDecimal itemTotal = item.getPrice().multiply(quantityBD);
    totalValue = totalValue.add(itemTotal);
}
}

//sort
for (int i = 0; i < itemList.size() - 1; i++) {
    for (int j = 0; j < itemList.size() - i - 1; j++) {
        Item item1 = itemList.get(j);
        Item item2 = itemList.get(j + 1);

        int categoryCompare = item1.getCategory().compareToIgnoreCase(item2.getCategory());
        if (categoryCompare > 0) {
            //-----swap if item1 category is after item2 category
            itemList.set(j, item2);

```

```

        itemList.set(j + 1, item1);
    } else if (categoryCompare == 0) {
        //-----if categories are same, sort by code
        if (item1.getCode().compareToIgnoreCase(item2.getCode()) > 0) {
            itemList.set(j, item2);
            itemList.set(j + 1, item1);
        }
    }
}

//set values
TotalItemID.setText(String.valueOf(totalItems));
TotalValueID.setText(totalValue.setScale(2, RoundingMode.HALF_UP).toString());
} catch (Exception e) {
    e.printStackTrace();
}
}

//refresh method
public void refreshInventory() {
    loadInventoryData();
    inventoryTable.setItems(itemList);
    inventoryTable.refresh();
}

//close app
public void handleExit(ActionEvent event) throws IOException {
    stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
    stage.close();}

//navigate to add function
public void GotoAddPage(ActionEvent event) throws IOException {
    Parent root = FXMLLoader.load(getClass().getResource("addPage-view.fxml"));
    stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
    scene = new Scene(root);
    stage.setTitle("Add Page");
    stage.setScene(scene);
    stage.setResizable(false);
    stage.show();}

```

```

//navigate to the home
public void GotoHome(ActionEvent event) throws IOException {
    Parent root = FXMLLoader.load(getClass().getResource("main-view.fxml"));
    stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
    scene = new Scene(root);
    stage.setTitle("Main Page");
    stage.setScene(scene);
    stage.show();
}
}

```

### 3.6. SelectdealersPageController

The SelectdealersPageController class manages the Dealer Selection of the inventory management system of John's Supermarket. Its main role is to randomly select 4 dealers from a file, display them in the Table, and it's also enables navigate further to view each dealer's listed items.

```

package org.example.johnsupermarket;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
import org.example.johnsupermarket.Models.Dealer;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;

```

```

import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.stream.Collectors;

public class SelectdealersPageController {
    //links between randomDealersPage-view.fxml and the controller
    @FXML private TableView<Dealer> dealerTable;
    @FXML private TableColumn<Dealer, String> nameCol;
    @FXML private TableColumn<Dealer, String> contactCol;
    @FXML private TableColumn<Dealer, String> locationCol;
    @FXML private Label OutputPanel;

    private Stage stage;
    private Scene scene;

    //when function calls, automatically table column binds to relevant properties of dealer(class)
    @FXML
    public void initialize() {
        nameCol.setCellValueFactory(new PropertyValueFactory<>("name"));
        contactCol.setCellValueFactory(new PropertyValueFactory<>("contact"));
        locationCol.setCellValueFactory(new PropertyValueFactory<>("location"));
    }

    //selecting dealers' method
    @FXML
    private void handleClickHereButton() {
        try {
            //read dealers' file
            List<Dealer> allDealers = readDealersFromFile("src/main/resources/Dealers/DealerDetails.txt");
            if (allDealers.isEmpty()) {
                showError("No dealers found in the file.");
                return;
            }
            //to randomly pick

```

```

Collections.shuffle(allDealers);
//add to the array that has a limit of 4
List<Dealer> selectedDealers = allDealers.stream().limit(4).collect(Collectors.toList());
//sorting by location(Method calling)
bubbleSortDealersByLocation(selectedDealers);
//sent to the dealers' table
dealerTable.getItems().setAll(selectedDealers);
OutputPanel.setText("4 Dealers are randomly selected.");
OutputPanel.setTextFill(Color.GREEN);
} catch (FileNotFoundException e) {
    OutputPanel.setText("File not found.");
    OutputPanel.setTextFill(Color.RED);
} catch (IOException e) {
    OutputPanel.setText("Error reading file.");
    OutputPanel.setTextFill(Color.RED);
} catch (Exception e) {
    OutputPanel.setText("unexpected error.");
    OutputPanel.setTextFill(Color.RED);
}
}

//get dealer details only from the file(not items of them)
private List<Dealer> readDealersFromFile(String filePath) throws IOException {
    List<Dealer> dealerList = new ArrayList<>();
    //reading file
    try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {
        String line;
        while ((line = br.readLine()) != null) {
            if (line.trim().isEmpty()) continue;
            //separating using ","
            String[] dealerParts = line.split(",");
            //dealers' details only have 3 properties so,
            if (dealerParts.length < 3) continue;
            //assign them
            String name = dealerParts[0].trim();
            String contact = dealerParts[1].trim();

```

```

        String location = dealerParts[2].trim();
        //add them to the dealer arraylist
        dealerList.add(new Dealer(name, contact, location));
        // Skip items
        while ((line = br.readLine()) != null && !line.trim().isEmpty()) {
            }
        }
    }
    //return dealer list to the "handleClickHereButton()"
    return dealerList;
}

//sorting method according to the location
private void bubbleSortDealersByLocation(List<Dealer> dealers) {
    int n = dealers.size();
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            String loc1 = dealers.get(j).getLocation().toLowerCase();
            String loc2 = dealers.get(j + 1).getLocation().toLowerCase();

            if (loc1.compareTo(loc2) > 0) {
                Collections.swap(dealers, j, j + 1);
            }
        }
    }
}

//error showing
private void showError(String message) {
    OutputPanel.setText(message);
    OutputPanel.setTextFill(Color.RED);
}

//app close
public void handleExit(ActionEvent event) throws IOException {
    stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
    stage.close();
}

```

//navigate to dealer's item page with selected dealer name

```
public void GotoDealerItemsPage(ActionEvent event) throws IOException {
    Dealer selectedDealer = dealerTable.getSelectionModel().getSelectedItem();
    if (selectedDealer == null) {
        showError("Please select a dealer first.");
        return;
    }
    FXMLLoader loader = new FXMLLoader(getClass().getResource("dealerItemsPage-view.fxml"));
    Parent root = loader.load();

    DealerItemsPageController controller = loader.getController();
    controller.setSelectedDealer(selectedDealer.getName());

    stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
    scene = new Scene(root);
    stage.setTitle("Dealer's Items Page");
    stage.setScene(scene);
    stage.setResizable(false);
    stage.show();
}

//navigate to the home
public void GotoHome(ActionEvent event) throws IOException {
    Parent root = FXMLLoader.load(getClass().getResource("main-view.fxml"));
    stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
    scene = new Scene(root);
    stage.setTitle("Main Page");
    stage.setScene(scene);
    stage.show();
}
}
```



### 3.7. DealerItemsPageController

The DealerItemsPageController class manages the **Dealer Items Page** of **John's Supermarket**. It enables users to search using dealer's name and display the items that the dealer supplies. This controller uses flat file (DealerDetails.txt), reads item data dynamically, and displays it the table.

```
package org.example.johnsupermarket;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.TextField;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
import org.example.johnsupermarket.Models.DealerItem;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class DealerItemsPageController {
    //links between dealerItemPage-view.fxml and the controller
    @FXML private TableView<DealerItem> itemTable;
    @FXML private TableColumn<DealerItem, String> itemNameCol;
    @FXML private TableColumn<DealerItem, String> brandCol;
    @FXML private TableColumn<DealerItem, Double> priceCol;
```

```

@FXML private TableColumn<DealerItem, Integer> quantityCol;
@FXML private TableColumn<DealerItem, String> categoryCol;
@FXML private Label OutputPanel;
@FXML private TextField dealerNameInput;

private String selectedDealer;
private Stage stage;
private Scene scene;

//load table columns with properties of dealerItem(class)
@FXML
public void initialize() {
    itemNameCol.setCellValueFactory(new PropertyValueFactory<>("name"));
    brandCol.setCellValueFactory(new PropertyValueFactory<>("brand"));
    priceCol.setCellValueFactory(new PropertyValueFactory<>("price"));
    quantityCol.setCellValueFactory(new PropertyValueFactory<>("quantity"));
    categoryCol.setCellValueFactory(new PropertyValueFactory<>("category"));
}

//this is navigating from another page (preload dealer names) and calling "handleSearchItems()" function
at the begging
public void setSelectedDealer(String dealerName) {
    this.selectedDealer = dealerName;
    dealerNameInput.setText(dealerName);
    handleSearchItems();
}

//main method: triggered when clicking search button
@FXML
private void handleSearchItems() {
    //get dealer's name
    String inputName = dealerNameInput.getText().trim();
    if (inputName.isEmpty()) {
        OutputPanel.setText("Please enter a dealer name.");
        OutputPanel.setTextFill(Color.RED);
        return;
    }

    //get dealer details the pass it to the table

```

```

try {
    //get dealer details
    List<DealerItem> items = findItemsForDealer("dealers.txt", inputName);
    //employ handling
    if (items.isEmpty()) {
        OutputPanel.setText("No items found for dealer: " + inputName);
        OutputPanel.setTextFill(Color.RED);
    } else {
        //pass top the table
        itemTable.getItems().setAll(items);
        //pass to outputPanel
        OutputPanel.setText("Items loaded for: " + inputName);
        OutputPanel.setTextFill(Color.GREEN);
    }
} catch (IOException e) {
    OutputPanel.setText("Error reading file.");
    OutputPanel.setTextFill(Color.RED);
}
}

//file reading method
private List<DealerItem> findItemsForDealer(String filePath, String dealerName) throws IOException {
    List<DealerItem> itemList = new ArrayList<>();
    //read the file
    try (BufferedReader br = new BufferedReader(new
FileReader("src/main/resources/Dealers/DealerDetails.txt"))) {
        String line;
        boolean dealerFound = false;
        //read line by line
        while ((line = br.readLine()) != null) {
            if (line.trim().isEmpty()) continue;
            //dividing by ","
            String[] dealerParts = line.split(",");
            //skip dealer details
            if (dealerParts.length > 3) continue;

            String currentDealerName = dealerParts[0].trim();

```

```

//check dealer in the file?
if (currentDealerName.equalsIgnoreCase(dealerName)) {
    dealerFound = true;

    //If yes, load their items (until empty)
    while ((line = br.readLine()) != null && !line.trim().isEmpty()) {
        String[] itemParts = line.split(",");

        if (itemParts.length >= 5) {
            String name = itemParts[0].trim();
            String brand = itemParts[1].trim();
            double price = Double.parseDouble(itemParts[2].trim());
            int quantity = Integer.parseInt(itemParts[3].trim());
            String category = itemParts[4].trim();

            //add to the itemList array
            itemList.add(new DealerItem(name, brand, price, quantity, category));
        }
    }
    break;
} else {
    //skip lines until next dealer
    while ((line = br.readLine()) != null && !line.trim().isEmpty()) {}
}

if (!dealerFound) {
    OutputPanel.setText("Dealer not found: " + dealerName);
    OutputPanel.setTextFill(Color.RED);
}

//return itemList to "handleSearchItems()"
return itemList;
}

//Navigate to Select Dealers page
public void GotoSelectDealersPage(ActionEvent event) throws IOException {
    Parent root = FXMLLoader.load(getClass().getResource("randomDealersPage-view.fxml"));
    stage = (Stage) ((Node) event.getSource()).getScene().getWindow();

```

```

    scene = new Scene(root);
    stage.setTitle("Dealers Page");
    stage.setScene(scene);
    stage.setResizable(false);
    stage.show();
}

//Navigate to the home page
public void GotoHome(ActionEvent event) throws IOException {
    Parent root = FXMLLoader.load(getClass().getResource("main-view.fxml"));
    stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
    scene = new Scene(root);
    stage.setTitle("Main Page");
    stage.setScene(scene);
    stage.show();
}

//exit from app
public void handleExit(ActionEvent event) throws IOException {
    stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
    stage.close();
}
}

```

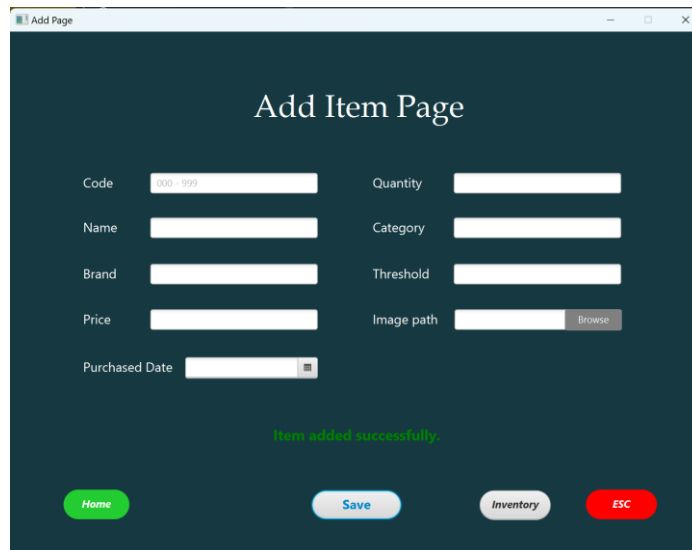
## 4. Test plan and test cases

### 4.1 Test cases

#### 01. Add New Inventory Item

Inputs: Code: 022, Name: Juice, Brand: Minute Maid, Price: 120.50, Qty: 30, Category: Drinks, Date: 2025-07-20, Threshold: 10, Image: juice.jpeg

Expected output: Item saved to file. Confirmation message shown. Fields cleared.



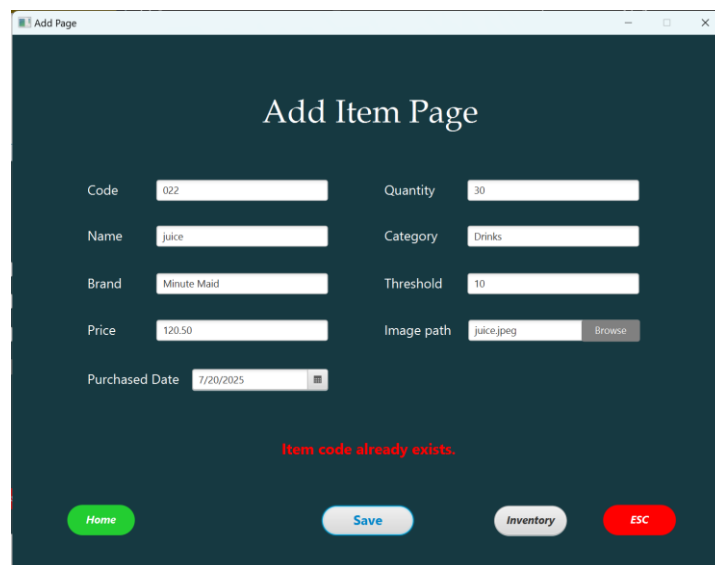
The screenshot shows a web application window titled 'Add Page' with a dark blue background. The main heading is 'Add Item Page'. Below it, there are several input fields: 'Code' (with a placeholder '000 - 999'), 'Quantity', 'Name', 'Category', 'Brand', 'Threshold', 'Price', 'Image path' (with a 'Browse' button), and 'Purchased Date' (with a calendar icon). A green message 'Item added successfully.' is displayed in the center. At the bottom, there are four buttons: 'Home' (green), 'Save' (blue), 'Inventory' (grey), and 'ESC' (red).

Figure 5 Test case -01

#### 02. Add Duplicate Item Code

Inputs: : Code: 022, Name: Juice, Brand: Minute Maid, Price: 120.50, Qty: 30, Category: Drinks, Date: 2025-07-20, Threshold: 10, Image: juice.jpeg (Already exist)

Expected output; Message shown: "Item code already exists." Item not added.



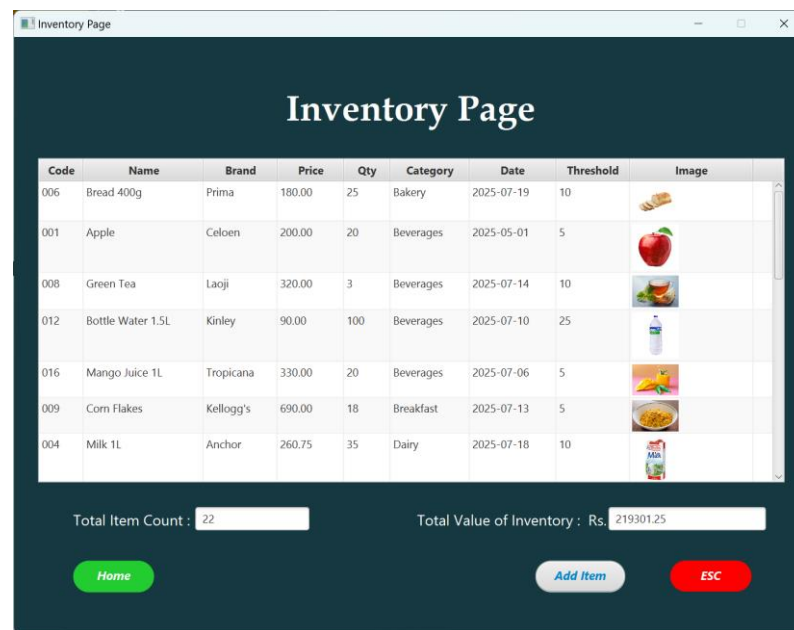
The screenshot shows the same 'Add Item Page' form as Figure 5, but with the following values entered: Code: 022, Quantity: 30, Name: juice, Category: Drinks, Brand: Minute Maid, Threshold: 10, Price: 120.50, Image path: juice.jpeg, and Purchased Date: 7/20/2025. A red message 'Item code already exists.' is displayed in the center. The buttons at the bottom are the same as in Figure 5.

Figure 6 Test case -02








### 03. View Inventory Items

Inputs: Open View Inventory page

Expected output: Table shows all items from file, sorted by category and then by code.



The screenshot shows a web application window titled "Inventory Page". It features a table with the following columns: Code, Name, Brand, Price, Qty, Category, Date, Threshold, and Image. The table contains 8 rows of data. Below the table, there are two summary fields: "Total Item Count : 22" and "Total Value of Inventory : Rs. 219301.25". At the bottom, there are three buttons: "Home" (green), "Add Item" (blue), and "ESC" (red).

Code	Name	Brand	Price	Qty	Category	Date	Threshold	Image
006	Bread 400g	Prima	180.00	25	Bakery	2025-07-19	10	
001	Apple	Celoen	200.00	20	Beverages	2025-05-01	5	
008	Green Tea	Laoji	320.00	3	Beverages	2025-07-14	10	
012	Bottle Water 1.5L	Kinley	90.00	100	Beverages	2025-07-10	25	
016	Mango Juice 1L	Tropicana	330.00	20	Beverages	2025-07-06	5	
009	Corn Flakes	Kellogg's	690.00	18	Breakfast	2025-07-13	5	
004	Milk 1L	Anchor	260.75	35	Dairy	2025-07-18	10	

Total Item Count : 22      Total Value of Inventory : Rs. 219301.25

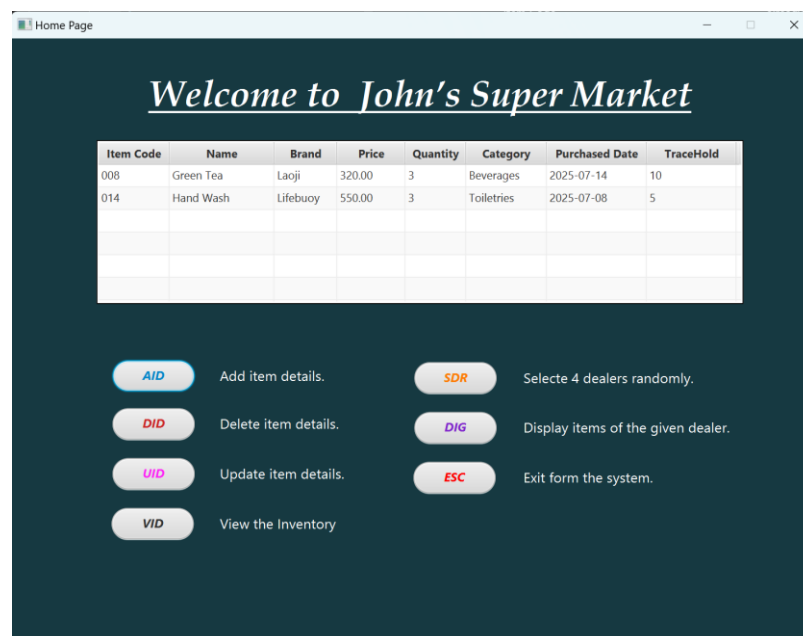
Home Add Item ESC

Figure 7 Test case -03

### 04. Low-Stock Items at Startup

Inputs: Inventory contains item(lifebuoy) with qty 3 and threshold 5

Expected output: That item is displayed in a separate low-stock table on startup.



The screenshot shows a web application window titled "Home Page". It features a welcome message "Welcome to John's Super Market". Below the message is a table with the following columns: Item Code, Name, Brand, Price, Quantity, Category, Purchased Date, and TraceHold. The table contains 2 rows of data. Below the table, there are several buttons with labels and descriptions: AID (Add item details), SDR (Selecte 4 dealers randomly), DID (Delete item details), DIG (Display items of the given dealer), UID (Update item details), ESC (Exit form the system), and VID (View the Inventory).

Item Code	Name	Brand	Price	Quantity	Category	Purchased Date	TraceHold
008	Green Tea	Laoji	320.00	3	Beverages	2025-07-14	10
014	Hand Wash	Lifebuoy	550.00	3	Toiletries	2025-07-08	5

AID Add item details. SDR Selecte 4 dealers randomly.

DID Delete item details. DIG Display items of the given dealer.

UID Update item details. ESC Exit form the system.

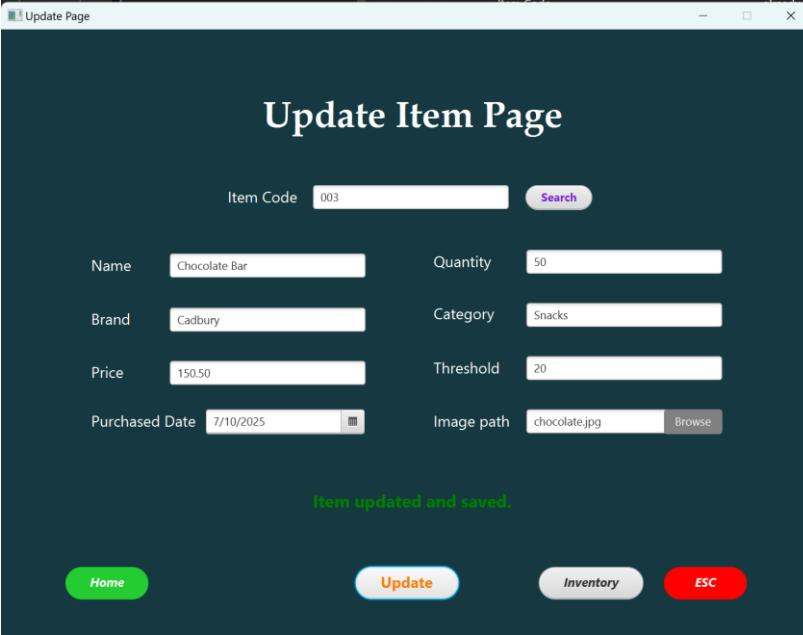
VID View the Inventory

Figure 8 Test case -04

#### 05. Update Existing Inventory Item

Inputs: Update 003 quantity to 50

Expected output: Item updated in file, confirmation shown, UI refreshed.



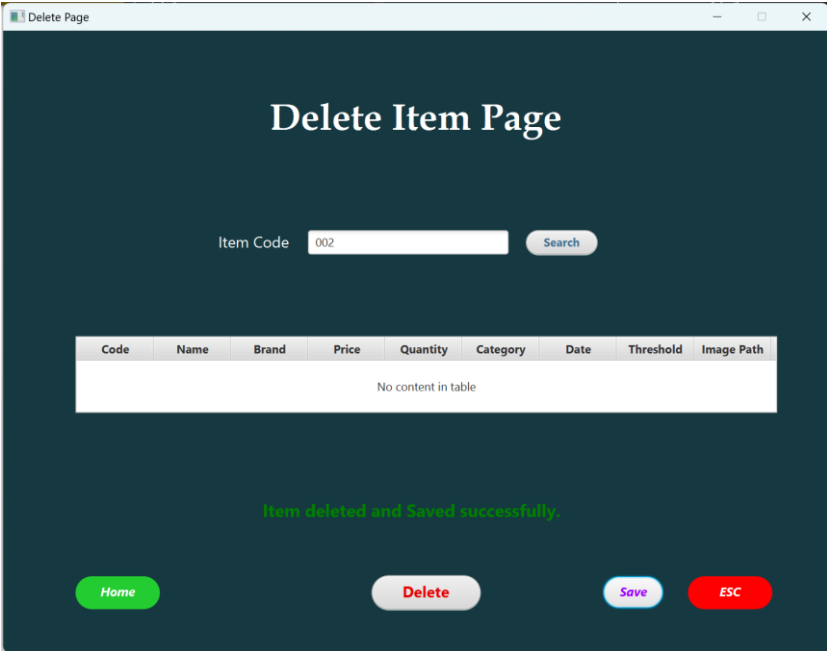
The screenshot shows a web application window titled "Update Page". The main heading is "Update Item Page". Below the heading, there is a search bar with "Item Code" and "003" entered, and a "Search" button. The form contains several input fields: "Name" (Chocolate Bar), "Quantity" (50), "Brand" (Cadbury), "Category" (Snacks), "Price" (150.50), "Threshold" (20), "Purchased Date" (7/10/2025), and "Image path" (chocolate.jpg). A green message "Item updated and saved." is displayed in the center. At the bottom, there are four buttons: "Home" (green), "Update" (orange), "Inventory" (grey), and "ESC" (red).

Figure 9 Test case -05

#### 06. Delete Existing Inventory Item

Inputs: Delete item 002 from inventory

Expected output: Item removed from file. Confirmation message shown. Table updates automatically.



The screenshot shows a web application window titled "Delete Page". The main heading is "Delete Item Page". Below the heading, there is a search bar with "Item Code" and "002" entered, and a "Search" button. Below the search bar, there is a table with the following columns: Code, Name, Brand, Price, Quantity, Category, Date, Threshold, and Image Path. The table is empty, and a message "No content in table" is displayed. A green message "Item deleted and Saved successfully" is displayed in the center. At the bottom, there are four buttons: "Home" (green), "Delete" (orange), "Save" (blue), and "ESC" (red).

Figure 10 Test case -06



#### 07. Select 4 Random Dealers

Inputs: Click "Select Dealers" button

Expected output: 4 random dealers selected and displayed in table.

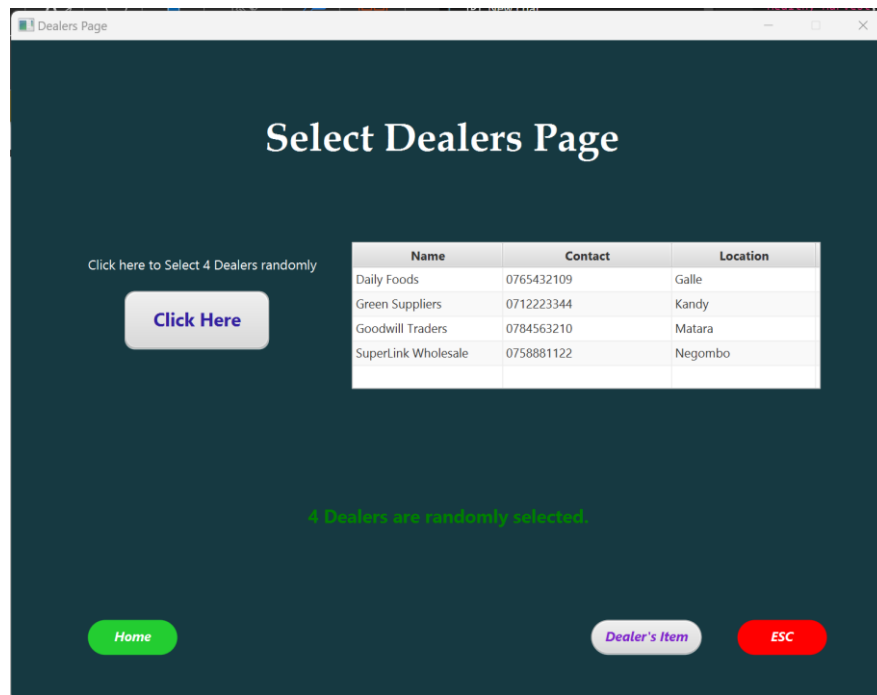


Figure 11 Test case -07

#### 08. Sort Dealers by Location

Inputs: Select 4 dealers with different locations

Expected output: Dealers displayed in ascending order by location.

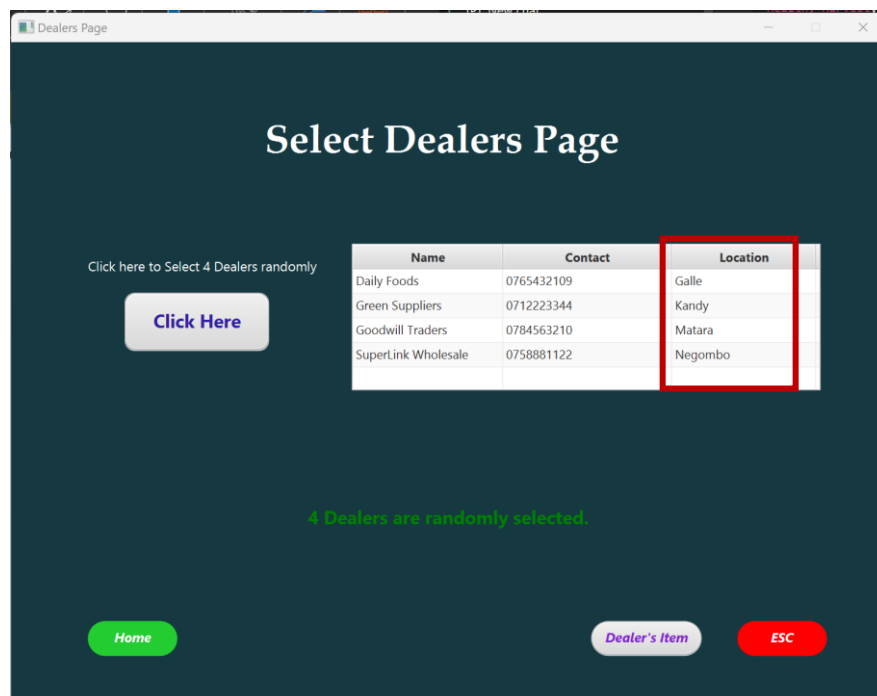


Figure 12 Test case -08

## 09. Display Dealer's Items

Inputs: Select dealer: Daily Foods

Expected output: All items under Daily Foods are displayed in the item table with correct fields and formatting.

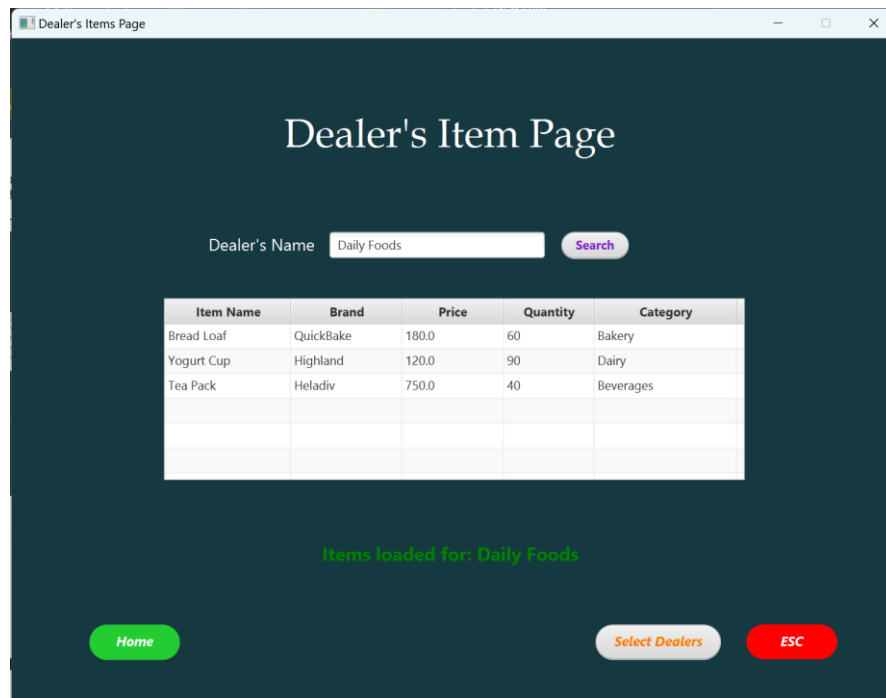


Figure 13 Test case -09

## 4.2 J-units Tests

This J-unit test focus on testing the getters functionality,

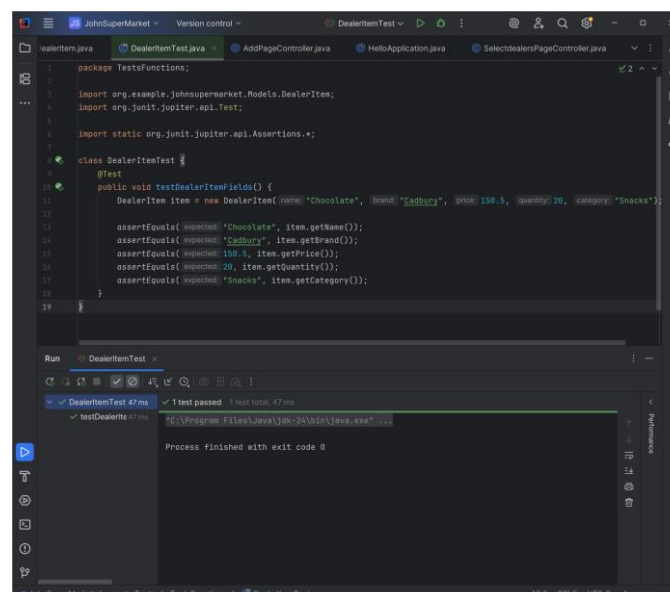


Figure 14 J-unit test

## 5. Robustness and the maintainability

### 5.1. Robustness

The actions towards improving dependability and error resistant-ness of the system includes, adding input validation to confirm that the user has filled all required fields (item details, dealer name, etc.). To ensure that the program does not crash because of errors like missing files or files with incorrect data, added “try and catch blocks” for error handling function. And incorporated error messages which explain the errors in detail to the user so the user can understand what the issues are. Then added preconditions that will guarantee that the data will be in the required format so that unforeseen problems will not occur during processing.

### 5.2. Maintainability

To make the code easier to update, understand, and expand in the future, divide the code into multiple controller classes, each function responsible for part of the system (like adding items, viewing inventory, or selecting dealers). then used helper methods for repeating tasks like reading from files, to avoid repeating the same code in multiple places. Code uses clear variable names, comments, and consistent formatting to make it easier to read and work on later. The code structure also allows future improvements (like adding a database or more features) without needing to rewrite everything again.

## 6. Assumptions

- Assuming All data is stored in text files (details.txt, DealerDetails.txt) using a comma-separated format.
- Each line represents one record.
- All required files and folders (e.g., /resources/images/) exist in the correct path before runtime.
- The program will not create missing files on its own.
- Each item code has 3 numeric characters (e.g. 000-999).
- Users are expected to enter complete and valid data (e.g., numbers for price/quantity).
- No deep validation is performed beyond basic checks.
- Dealers are listed in DealerDetails.txt, followed by their items and each dealer separated by blank line.
- The system is designed for a single user at a time.
- When choosing a photo for an item choose from “/resources/images/” folder, otherwise photo will not display in the inventory until you restart the system.

## 7. Conclusion

In this coursework, I developed a JavaFX-based inventory and dealer management system for John's Super Market. The application includes all required features such as adding, updating, deleting, and viewing inventory items; displaying low-stock items at startup; randomly selecting and sorting dealers; and showing their related items. All data are managed using structured text files, and custom algorithms were used to sorting as required.

For good coding practices, included meaningful method names, modular code structure, and proper input validations. The system was designed using object-oriented programming principles like encapsulation and abstraction, and make sure the code follows robustness and maintainability by handling exceptions and separating logic into manageable methods and classes.

## 8. Reference list

Bro Code (2025) Java Full Course for free (2025). 2 January. Available at:

<https://youtu.be/xTtL8E4LzTQ?si=oKe01Uf2kcGslXGG> (Accessed: 06 July 2025).

Code With Bachi (2021) JavaFX sinhala - Setting up development environment in inteliJ and first program (Video 01). 18 August. . Available at: [https://youtu.be/PNxjYeW\\_KyQ?si=fl8cT7lhzf9ryS7V](https://youtu.be/PNxjYeW_KyQ?si=fl8cT7lhzf9ryS7V) (Accessed: 15 July 2025).

Code With Bachi (2021) JavaFX sinhala - Switch between scenes and open in new stage - VIDEO 2. 23 August. . Available at: <https://youtu.be/zUFIFuwd-Dc?si=EGfdl6806oI1B2pg> (Accessed: 16 July 2025).

Kunal Kushwaha(2021) OOP 1 | Introduction & Concepts - Classes, Objects, Constructors, Keywords. 5 November. Available at: <https://youtu.be/BSVKUk58K6U?si=2sgy9btB-KIPfUrP> (Accessed: 10 July 2025).

Kunal Kushwaha(2021) OOP 2 | Packages, Static, Singleton Class, In-built Methods. 5 November. Available at: [https://youtu.be/\\_Ya6CN13t8k?si=cbmMIJNn3Fo9qEwN](https://youtu.be/_Ya6CN13t8k?si=cbmMIJNn3Fo9qEwN) (Accessed: 11 July 2025).

Kunal Kushwaha(2021) OOP 3 | Principles - Inheritance, Polymorphism, Encapsulation, Abstraction. 8 November. Available at: <https://youtu.be/46T2wD3IuhM?si=od3pD4QcxvHQ53-5> (Accessed: 11 July 2025).

Kunal Kushwaha(2021) OOP 4 | Access Control, In-built Packages, Object Class. 10 November. Available at: <https://youtu.be/W145DXs8fFg?si=X-O5IjbjahPlmX-P> (Accessed: 12 July 2025).

## 9. Appendices

### 9.1. Item.java

```
package org.example.johnsupermarket.Models;
import java.math.BigDecimal;
import java.time.LocalDate;
public class Item {
    private String code;
    private String name;
    private String brand;
    private BigDecimal price;
    private int quantity;
    private String category;
    private LocalDate date;
    private int Threshold;
    private String imagePath;

    public Item(String code, String name, String brand, BigDecimal price, int quantity,
                String category, LocalDate date, int Threshold, String imagePath) {
        this.code = code;
        this.name = name;
        this.brand = brand;
        this.price = price;
        this.quantity = quantity;
        this.category = category;
        this.date = date;
        this.Threshold = Threshold;
        this.imagePath = imagePath;
    }

    public Item(String name, String brand, BigDecimal price, int quantity, String category) {
        this.name = name;
        this.brand = brand;
        this.price = price;
        this.quantity = quantity;
        this.category = category;
```

```

}
//-----Getters
public String getCode() {
    return code;
}
public String getName() {
    return name;
}
public String getBrand() {
    return brand;
}
public BigDecimal getPrice() {
    return price;
}
public int getQuantity() {
    return quantity;
}
public String getCategory() {
    return category;
}
public LocalDate getDate() {
    return date;
}
public String getImagePath() {
    return imagePath;
}
public int getThreshold() {
    return Threshold;
}
//-----Setters
public void setCode(String code) {this.code = code;}
public void setName(String name) {this.name = name;}
public void setBrand(String brand) {
    this.brand = brand;
}
public void setPrice(BigDecimal price) {

```

```

        this.price = price;
    }
    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }
    public void setCategory(String category) {
        this.category = category;
    }
    public void setDate(LocalDate date) {
        this.date = date;
    }
    public void setThreshold(int Threshold) {this.Threshold = Threshold;}
    public void setImagePath(String imagePath) {this.imagePath = imagePath;}
    @Override
    public String toString() {
        return code + "," + name + "," + brand + "," + price + "," + quantity + "," + category + "," + date + "," +
Threshold + "," + imagePath;
    }
}

```

## 9.2. Dealer.java

```
package org.example.johnsupermarket.Models;
import javafx.beans.property.SimpleStringProperty;

public class Dealer {
    private final SimpleStringProperty name;
    private final SimpleStringProperty contact;
    private final SimpleStringProperty location;

    public Dealer(String name, String contact, String location) {
        this.name = new SimpleStringProperty(name);
        this.contact = new SimpleStringProperty(contact);
        this.location = new SimpleStringProperty(location);
    }

    public String getName() { return name.get(); }
    public String getContact() { return contact.get(); }
    public String getLocation() { return location.get(); }

    public void setName(String value) { name.set(value); }
    public void setContact(String value) { contact.set(value); }
    public void setLocation(String value) { location.set(value); }
}
```

## 9.3 DealerItem.java

```
package org.example.johnsupermarket.Models;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.beans.property.SimpleIntegerProperty;
import javafx.beans.property.SimpleStringProperty;

public class DealerItem {
    private final SimpleStringProperty name;
    private final SimpleStringProperty brand;
```



```
private final SimpleDoubleProperty price;  
private final SimpleIntegerProperty quantity;  
private final SimpleStringProperty category;
```

```
public DealerItem(String name, String brand, double price, int quantity, String category) {  
    this.name = new SimpleStringProperty(name);  
    this.brand = new SimpleStringProperty(brand);  
    this.price = new SimpleDoubleProperty(price);  
    this.quantity = new SimpleIntegerProperty(quantity);  
    this.category = new SimpleStringProperty(category);  
}
```

```
public String getName() { return name.get(); }  
public String getBrand() { return brand.get(); }  
public double getPrice() { return price.get(); }  
public int getQuantity() { return quantity.get(); }  
public String getCategory() { return category.get(); }  
}
```