Criterion C: Development

Techniques used:

1. Object oriented programming
2. JavaFX platform
3. Aggregation
4. Algorithmic thinking
5. Multi-purpose scenes and global parameters
6. Two-Dimensional Arrays
7. Linked lists
8. Exception handling using JavaFX LocalDate library
9. Reading, writing .csv files
10. PDFjet external library for Java
11. JavaFX LineChart library
12. JavaFX Alert library, notification system
13. CSS interface design

1. Object Oriented Programming

This technique allows the project to resemble real-life relationships inside code. Since the project revolves around the shop's products, it is perfect to use Object Oriented Programming to build an app to create transactions of products and manage them. The project has 16 classes in total, which can be seen in Appendix 4.

```
public class Product {
    private String name;
    private String barcode;
    private double price;
    private int quantity;
    private String expdate;
    private String dateadded;

    public Product(String name, double price, int quantity, String barcode, String expdate, String dateadded) {...8 lines }
    public Product(String name, double price, int quantity, String barcode, String expdate) {...8 lines }
    public Product(){...2 lines }

    public static String setDate(){...7 lines }
    public void settempDate(String date){...3 lines }
    public  String getDateadded(){...3 lines }
    public String getBarcode() {...3 lines }
    public void setBarcode(String barcode) {...3 lines }
    public String getName() {...3 lines }
    public void setName(String name) {...3 lines }
    public double getPrice() {...3 lines }
    public void setPrice(double price) {...3 lines }
    public int getQuantity() {...3 lines }
    public void setQuantity(int quantity) {...3 lines }
    public String getExpdate() {...3 lines }
    public void setExpdate(String expdate) {...3 lines }

    @Override
    public String toString() {...3 lines }
}
```

Figure 1. Product class, with its instance variables, constructors, and methods.

2. JavaFX platform

JavaFX is an efficient open-source implementation of Java platform that allows for next-generation Graphical User Interface design. The increased customization methods, as well as new design functions, enhance the user experience. Owing to an intuitive SceneBuilder program that allows for Drag & Drop application development on FXML files containing the layout of a scene. This framework is used in every GUI window and f.ex. allows for an implementation of LineChart library, which is discussed later. It is important for the project as it allows for completing the success criteria of a simple graphical user interface and showing an appropriate price diagram.

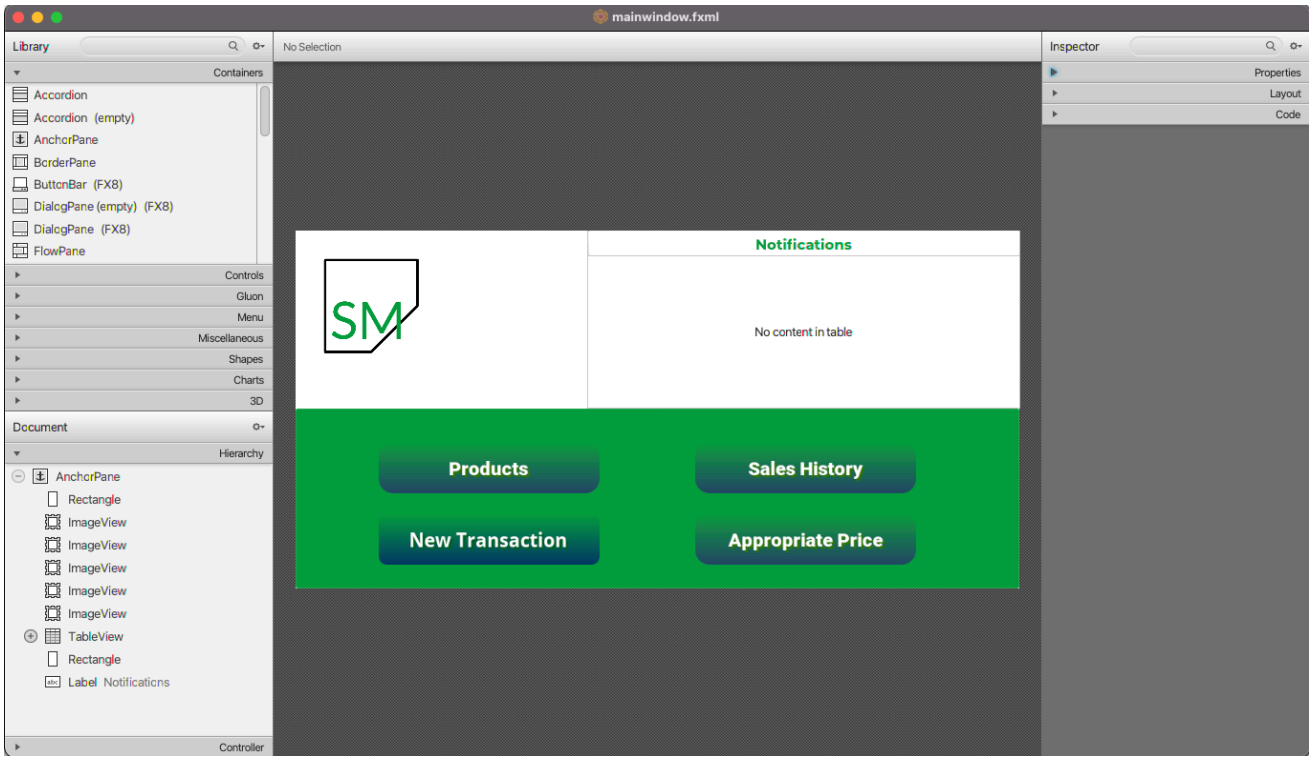

Figure 2. SceneBuilder development interface of the mainwindow.fxml file.

3. Aggregation

This important OOP technique was used in this project to share the components from one class to another. In the application, class TransactionItem has class Product to use its methods on

TransactionItem operations. This allows for efficient and understandable code structure, which most importantly reuses the existing code rather than writing the same code again.

```
public class TransactionItem {

    private Product product;
    private int itemq;

    public TransactionItem(Product product, int itemq) {...4 lines }

    public Product getProduct() {...3 lines }

    public void setProduct(Product product) {...3 lines }

    public int getItemq() {...3 lines }

    public void setItemq(int itemq) {...3 lines }

    public String getBarcode() {...3 lines }

    public String getName() {...3 lines }

    public double getPrice() {...3 lines }

    public int getQuantity() {...3 lines }

    public String getExpdate() {...3 lines }
    public String getDateadded() {...3 lines }
    public void setDateadded(String date) {...3 lines }

}
```

Figure 3. TransactionItem class using aggregation by an instance variable product composed of class Product.

```
@FXML
private void finish() throws IOException {
    for (Iterator<TransactionItem> iterator = transaction.getItems().iterator(); iterator.hasNext();) {
        TransactionItem next = iterator.next();
        for (Iterator<Product> iterator1 = Products.iterator(); iterator1.hasNext();) {
            Product next1 = iterator1.next();
            if(next.getProduct().equals(next1)){
                if(next1.getQuantity()-next.getItemq()<0){
                 Alert alert=new Alert(Alert.AlertType.ERROR);
                 alert.setContentText("You can't substract quantity equal or smaller to zero.");
                 alert.show();
                   return;
                }

                Product p=next.getProduct();
                String product = p.getName() + "," + p.getPrice() + "," + p.getQuantity() + "," +
                                 p.getBarcode() + "," + p.getExpdate() + "," + Product.setDate() + "," + next.getItemq();
                App.AddLineInSalesHistory("saleshistory.csv", product);
                TransactionItem temp=new TransactionItem(p,next.getItemq());
                temp.setDateadded(Product.setDate());
                App.SalesProducts.add(0,temp);
                next1.setQuantity(next1.getQuantity()-next.getItemq());
                    break;
                }
            }
        }
        transaction = new Transaction();
        App.WriteFile(Products, "products.csv");
        App.setRoot("mainwindow");
}
```

Figure 4. finish() method from ChangePOSController class using variables from Transaction and Products class.

4. Algorithmic thinking

In order to solve any problems in a fast and efficient way, algorithmic thinking was used. For instance, in the Aprice2Controller class, method sortData(double[][] a) is responsible for making sure that the data points for demand and supply curves are sorted in ascending order by quantity. By using a bubble sort algorithm, the appropriate graph diagram will correctly display entered information.

```
@FXML
private void next() throws IOException{

    if(c!=0){
        data[c-1][0]=Double.valueOf(price.getText());
        data[c-1][1]=Double.valueOf(quantity.getText());
        c--;
        price.setText("");
        quantity.setText("");
        if(c==0){
            next.setVisible(false);
            finish.setVisible(true);
        }
    }

}
@FXML
private void switchToAprice3() throws IOException {...4 lines }

public void sortData(double[][] a){
    for (int row = 0; row < a.length-1; row++) {
            if(a[row][1]>a[row+1][1]){
                double temp=a[row][1];
                double temp2=a[row][0];
                a[row][1]=a[row+1][1];
                a[row][0]=a[row+1][0];
                a[row+1][1]=temp;
                a[row+1][0]=temp2;
            }
    }
}
```

Figure 5. next() and sortData(double[][] a) methods in Aprice2Controller class using algorithmic thinking.

5. Multi-purpose scenes and global parameters

In order to reduce the size of the program, as well as make the code more efficient, multi-purpose scenes and global parameters were used. Rather than creating 4 FXML files (scenes) for both demand and supply curve generators, the use of a global parameter allows reusing two existing scenes once again to show the appropriate price diagram. Following global parameters are used: passProduct (to edit and remove a product), datapoints (to share amount of points to be entered), lastScene (to correctly direct from the popup window), isDemand (to properly assign entered data to a list of demand or supply dataset, see Figure 6.). This is one of the ingenious ideas that increase the code's efficiency and clarity.

```
public class Aprice3Controller {
    @FXML LineChart<Number,Number> graph;

    public void initialize() {
        graph.setTitle("Appropriate price diagram");
        graph.setAnimated(false);
        XYChart.Series second=new XYChart.Series();
        first.setName("Demand Curve");
        second.setName("Supply Curve");
        if(isDemand){
        for (int i = 0; i < data.length; i++) {
          first.getData().add(new XYChart.Data(String.valueOf(data[i][1]),data[i][0]));
        }
        }
        else if(isDemand==false){

         for (int i = 0; i < data.length; i++) {
          second.getData().add(new XYChart.Data(String.valueOf(data[i][1]),data[i][0]));
        }
            System.out.println(first.toString());
         graph.getData().add(second);
        }
        graph.getData().add(first);
    }
```

Figure 6. An Aprice3Controller multi-purpose scene controller initialize() method, which uses an isDemand global parameter to set values to an appropriate list based on its Boolean value determined in Aprice3Controller.

6. Two-Dimensional Arrays

In order to effectively organize and collect data, a 2D Array was used. Such a matrix of values was the perfect way to store values of X and Y values for supply and demand curves to estimate an appropriate price from data.

```
public class Aprice2Controller {
    @FXML
    private TextField price;
    @FXML
    private TextField quantity;
    @FXML
    private ImageView next;
    @FXML
    private ImageView finish;
    public static double[][] data;
    public int c;

    public void initialize(){
        data=new double[datapoints][2];
        c=datapoints;
    }
```

Figure 7. Initialize of the 2D Array in Aprice2Controller class, as a variable array data.

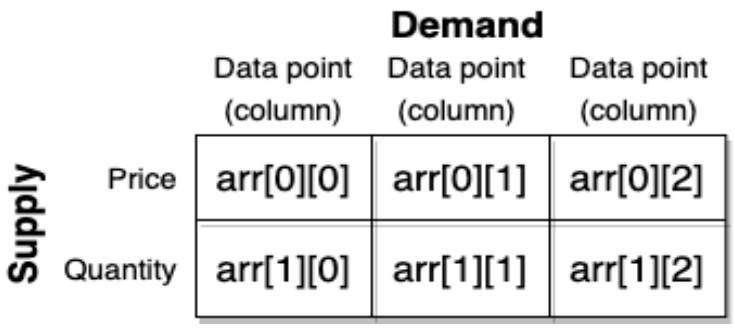

Figure 8. An example of a table showing the 2D Array set of values for 3 data points.

7. Linked lists

In order to store products which amount may vary in size, a dynamic data structure of linked lists implemented by LinkedList Java library was used. Most operations are thus performed on a linked list of products, but also on a TransactionItem linked list. Compared to an array, adding additional elements is easier. Most importantly, such a data structure is easy to scale, if the client's shop will further grow.

```
public class Transaction {

    private LinkedList<TransactionItem> items;
    private double totalp;
    private Date date;
    private double discount;

    public Transaction(LinkedList<TransactionItem> items) {
        this.items = items;
    }

    Transaction() {
        items = new LinkedList<>();
        discount = 0;
    }
```

Figure 9. A part of Transaction class, with its instance variables and constructors, consisting of a LinkedList items.

```
@FXML
private void finish() throws IOException {
    for (Iterator<TransactionItem> iterator = transaction.getItems().iterator(); iterator.hasNext();) {
        TransactionItem next = iterator.next();
        for (Iterator<Product> iterator1 = Products.iterator(); iterator1.hasNext();) {
            Product next1 = iterator1.next();
            if(next.getProduct().equals(next1)){
                if(next1.getQuantity()-next.getItemq()<0){
                 Alert alert=new Alert(Alert.AlertType.ERROR);
                 alert.setContentText("You can't substract quantity equal or smaller to zero.");
                 alert.show();
                   return;
                }

            Product p=next.getProduct();
            String product = p.getName() + "," + p.getPrice() + "," + p.getQuantity() + "," +
                             p.getBarcode() + "," + p.getExpdate() + "," + Product.setDate() + "," + next.getItemq();
            App.AddLineInSalesHistory("saleshistory.csv", product);
            TransactionItem temp=new TransactionItem(p,next.getItemq());
            temp.setDateadded(Product.setDate());
            App.SalesProducts.add(0,temp);
            next1.setQuantity(next1.getQuantity()-next.getItemq());
                break;
            }
        }
    }
    transaction = new Transaction();
    App.WriteFile(Products, "products.csv");
    App.setRoot("mainwindow");
}
```

Figure 10. finish() method from ChangePOSController class, where a TransactionItem temp is added to the SalesProducts LinkedList.

8. Exception handling using JavaFX LocalDate library

The expiration dates of the products are saved using a LocalDate JavaFX library, which allows for sorting and operating on dates. Moreover, this library is used to parse entered data into readable date format, and it can simultaneously check whether the user has entered it correctly. An exception handling is used to inform the user about incorrectly entered data and stop the method from further action. This is an innovative idea to solve a simple problem that would otherwise require an extended manual code writing.

```
public boolean expcheck(TextField x) {
    if(x.getText().isEmpty()==true){
        x.setText("00/00/0000");
        return true;
    }
    try{
    DateTimeFormatter formatter= DateTimeFormatter.ofPattern("MM/dd/yyyy");
    LocalDate date=LocalDate.parse(x.getText(), formatter);
    }
    catch(Exception e){
        System.out.println(e);
        return false;
    }
    return true;
}
```

Figure 11. An expcheck(TextField x) method from EditProductController and AddProductController classes that handles an exception of incorrectly entered date format.

9. Reading, writing .csv files

My client wants his program to run on his desktop work computer, and to store the data on its hard drive. Therefore, storing data with the use of .csv files is the best option suiting his requirements. Its simple implementation and efficiency in size allow for storage of a great number of products offline. The project stores products in products.csv file, and it chronologically stores the last sold products in saleshistory.csv file. The file reading and writing methods were created together with my classmates, with my modifications to suit my project. AddLine and AddLineInSalesHistory methods were created to write a file with one line of a product more, rather than a whole set of a products list.

```
public static void ReadFile(LinkedList<Product> Products, String file) throws FileNotFoundException, IOException {...14 lines }
public static void AddLine(LinkedList<Product> Products, String file_name, String product) throws FileNotFoundException, IOException {...8 lines }
public static void WriteFile(LinkedList<Product> Products, String file_name) throws FileNotFoundException, IOException {...10 lines }
public static void ReadSalesHistory(LinkedList<TransactionItem> list, String file) throws FileNotFoundException, IOException {...24 lines }
public static void AddLineInSalesHistory(String file_name, String product) throws FileNotFoundException, IOException {...8 lines }
```

Figure 12. Reading and writing .csv methods in App class, which modify files products.csv and saleshistory.csv.

```
public static void WriteFile(LinkedList<Product> Products, String file_name) throws FileNotFoundException, IOException {
    File file = new File(file_name);
    FileWriter writer = new FileWriter(file, false);
    for (int size = 0; size < Products.size(); size++) {
        writer.write(Products.get(size).getName() + "," + Products.get(size).getPrice() + "," + Products.get(size).getQuantity()
        + "," + Products.get(size).getBarcode() + "," + Products.get(size).getExpdate() + "," + Products.get(size).getDateadded());
        writer.write("\n");
    }
    writer.flush();
    writer.close();
}

public static void ReadSalesHistory(LinkedList<TransactionItem> list, String file) throws FileNotFoundException, IOException {
    FileReader fr = new FileReader(file);
    BufferedReader bfr = new BufferedReader(fr);
    String line = "";
    String splitBy = ",";
    ArrayList<String[]> data = new ArrayList<>();
    while ((line = bfr.readLine()) != null) {
        try {
            String[] product = line.split(splitBy);
            data.add(product);
        } catch (Exception e) {
            System.out.println(e);
        }
    }
    for (int i = data.size() - 1; i >= 0; i--) {
        String[] product = data.get(i);
        try {
        TransactionItem x = new TransactionItem(new Product(product[0], Double.parseDouble(product[1]),
                Integer.parseInt(product[2]), product[3], product[4], product[5]), Integer.parseInt(product[6]));
            list.add(x);
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

Figure 13. Writing WriteFile(LinkedList<Product> products, String file_name) method and ReadSalesHistory(LinkedList<TransactionItem> list, String file) reverse-reading method for saleshistory.csv file in App class.

10. PDFjet external library for Java

PDFjet is a programmer's library that allows for creating PDF format documents from a desktop-based application. My client will be able to print a weekly or monthly sales report, which is loaded onto a PDF file named a Sales Report. The external library has been obtained under an open source edition license, where use is permitted under retained copyright notice

visible on the document.[1]

```
public void pdf(String s) throws FileNotFoundException, Exception{
    File bill=new File("Sales Report.pdf");
    FileOutputStream billout=new FileOutputStream(bill);
    PDF pdf=new PDF(billout);
    Page page = new Page(pdf, A4.PORTRAIT);
    Font f1=new Font(pdf, CoreFont.HELVETICA_BOLD);
    Font f2=new Font(pdf, CoreFont.HELVETICA);

    Table table=new Table();
    List<List<Cell>> tableData=new ArrayList<List<Cell>>();
    List<Cell> tableRow=new ArrayList<Cell>();
    TextColumn column = new TextColumn();
    column.setWidth(200);
    TextLine text=new TextLine(f1, "Store Manager's Sales Report");
    Paragraph p1=new Paragraph();
    p1.setAlignment(Align.JUSTIFY);
    p1.add(text);
    column.addParagraph(p1);
    column.setLocation(210f, 50f);
    column.drawOn(page);

    TextColumn total = new TextColumn();
    total.setWidth(200);
    TextLine totalText=new TextLine(f1);
    if(s.equals("week"))
        totalText.setText("Total revenue: "+String.valueOf(totalweek));
    else if(s.equals("month"))
        totalText.setText("Total revenue: "+String.valueOf(totalmonth));

    Paragraph p2=new Paragraph();
    p2.setAlignment(Align.JUSTIFY);
    p2.add(totalText);
    total.addParagraph(p2);
    total.setLocation(210f, 70f);
    total.drawOn(page);

    Cell cell=new Cell(f1,"Product");
    tableRow.add(cell);
    cell=new Cell(f1,"Price");
    tableRow.add(cell);
    cell=new Cell(f1,"Quantity Sold");
    tableRow.add(cell);
    cell=new Cell(f1,"Product Revenue");
    tableRow.add(cell);
    tableData.add(tableRow);

    if(s.equals("week"))
        for (int i = 0; i < weeklytable.size(); i++) {
        Cell product=new Cell(f2, weeklytable.get(i).getName());
        Cell quantity=new Cell(f2,String.valueOf(weeklytable.get(i).getItemq()));
        Cell price=new Cell(f2,String.valueOf(weeklytable.get(i).getPrice()));
        Cell tr=new Cell(f2,String.valueOf(weeklytable.get(i).getPrice()*weeklytable.get(i).getItemq()));
        tableRow=new ArrayList<Cell>();
        tableRow.add(product);
        tableRow.add(price);
        tableRow.add(quantity);
        tableRow.add(tr);
        tableData.add(tableRow);
        }
    if(s.equals("month")){
        for (int i = 0; i < monthlytable.size(); i++) {
        Cell product=new Cell(f2, monthlytable.get(i).getName());
        Cell quantity=new Cell(f2,String.valueOf(monthlytable.get(i).getItemq()));
        Cell price=new Cell(f2,String.valueOf(monthlytable.get(i).getPrice()));
        Cell tr=new Cell(f2,String.valueOf(monthlytable.get(i).getPrice()*monthlytable.get(i).getItemq()));
        tableRow=new ArrayList<Cell>();
        tableRow.add(product);
        tableRow.add(quantity);
        tableRow.add(price);
        tableRow.add(tr);
        tableData.add(tableRow);
        }
```

```
    table.setData(tableData);
    table.setPosition(50f, 100f);
    table.setColumnWidth(0, 100f);
    table.setColumnWidth(1, 100f);
    table.setColumnWidth(2, 140f);
    table.setColumnWidth(3, 150f);
    while (true){
        table.drawOn(page);
        if (!table.hasMoreData()){
            table.resetRenderedPagesCount();
            break;
        }
        page=new Page(pdf, A4.PORTRAIT);
    }
            pdf.setTitle("Store Manager's Sales Report");
            pdf.setSubject("Store Manager");
            pdf.setAuthor("Store Manager");
    pdf.complete();
    billout.close();

    System.out.println("Saved to "+ bill.getAbsolutePath());
            Alert a=new Alert(AlertType.INFORMATION);
            a.setContentText("Saved to"+ bill.getAbsolutePath());
            a.show();
}
```

Figure 14. pdf(String s) method in SalesHistoryController that creates a PDF file with sales report.

[1] PDFjet Open Source Edition License, https://pdfjet.com/os/download.html. Accessed 30th Dec, 2021.

Store Manager's Sales Report

Total revenue: 28.15

Product	Quantity Sold	Price	Product Revenue
Tissues	22	0.75	16.5
Bag	8	0.5	4.0
Still water	14	1.7	23.8
Energy Drink	4	3.0	12.0

Figure 15. A sample of a Sales Report PDF file (Sales Report.pdf)

11. JavaFX LineChart library

My client explicitly wanted a tool that would help him in determining an appropriate price for his products. To make that decision, the JavaFX LineChart library was used to draw and create a supply and demand graph showing the equilibrium price. My client can enter multiple data points and see what price and quantity suits him the best. This approach is better than a table with data or a simple explicit answer, as it gives my client a clearer and more detailed look at the appropriate price mechanism.

Figure 16. The aprice3.fxml window with added supply and demand curve on the graph, illustrating the consumer behavior according to the product's price.

```
public class Aprice3Controller {
    @FXML LineChart<Number,Number> graph;

    public void initialize() {
        graph.setTitle("Appropriate price diagram");
        graph.setAnimated(false);
        XYChart.Series second=new XYChart.Series();
        first.setName("Demand Curve");
        second.setName("Supply Curve");
        if(isDemand){
        for (int i = 0; i < data.length; i++) {
          first.getData().add(new XYChart.Data(String.valueOf(data[i][1]),data[i][0]));
        }
        }
        else if(isDemand==false){
         for (int i = 0; i < data.length; i++) {
          second.getData().add(new XYChart.Data(String.valueOf(data[i][1]),data[i][0]));
        }
        graph.getData().add(second);
        }

        graph.getData().add(first);
    }
```

Figure 17. The initialize() method, which enters datapoints in data[][] onto the graph, and initializes (designs) it.

12. JavaFX Alert library, notification system

As humans are prone to committing mistakes, an Alert system was used to notify the client if some actions result in unwanted results. Moreover, a notification table was designed to notify the client about incoming expiration dates and low stock of products. A JavaFX Alert library will warn the user by a pop-up notification, and a notification system table is going to display important information in a simple, clear way.

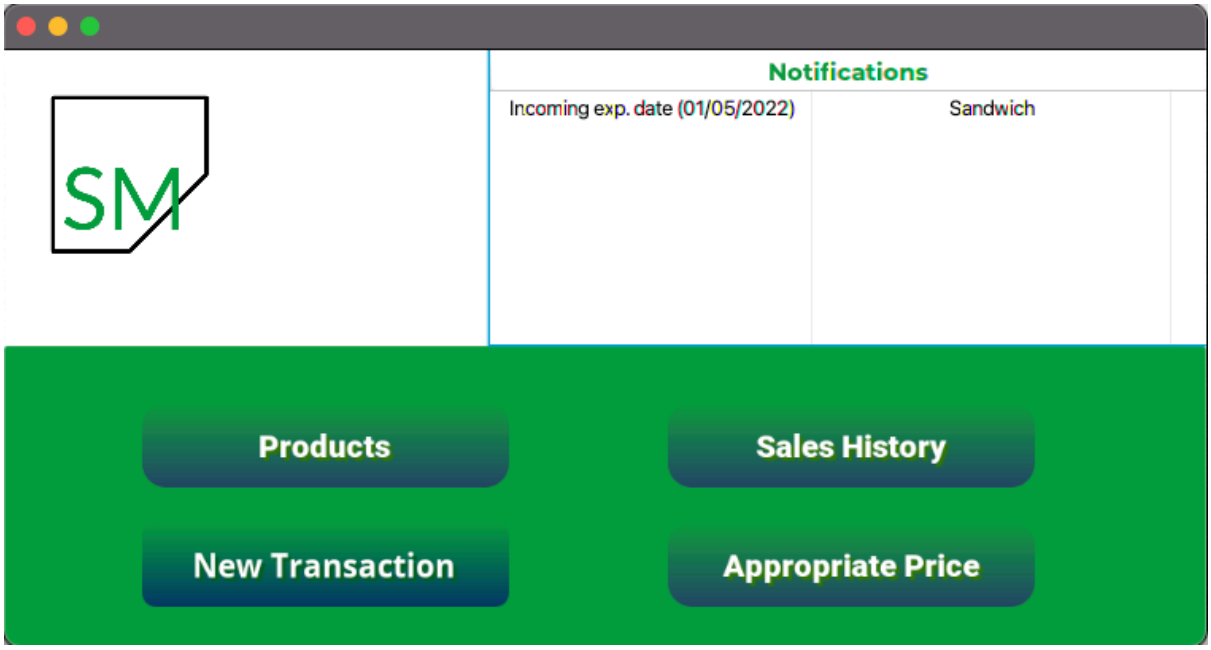

Figure 18. A notification table with incoming expiration dates and low stock visible on the main application's window, mainwindow.fxml.

```
public void populatetable() {
    StackPane placeHolder = new StackPane();
    placeHolder.setStyle("-fx-background-color: transparent;");
    table.setPlaceholder(placeHolder);
    issue.setReorderable(false);
    name.setReorderable(false);
    issue.setStyle( "-fx-alignment: CENTER;");
    name.setStyle( "-fx-alignment: CENTER;");
    ObservableList<MainWindowNotification> list = FXCollections.observableArrayList();
    issue.setCellValueFactory(new PropertyValueFactory<>("issue"));
    name.setCellValueFactory(new PropertyValueFactory<>("name"));
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("MM/dd/yyyy");
    for (int i = 0; i < Products.size(); i++) {
        if(Products.get(i).getQuantity()<=10){
            MainWindowNotification product= new MainWindowNotification("Low quantity ("+Products.get(i).getQuantity()+")",Products.get(i).getName());
            list.add(product);
        }
        if(!"00/00/0000".equals(Products.get(i).getExpdate())){
            String sDate=Products.get(i).getExpdate();
            LocalDate date=LocalDate.parse(sDate, formatter);

        if(expdate(date)){
           MainWindowNotification product= new MainWindowNotification("Incoming exp. date ("+Products.get(i).getExpdate()+")",Products.get(i).getName());
           list.add(product);
        }
    }
    }
    table.setItems(list);
}
```

Figure 19. populatetable() method in MainWindow Controller class, which designs and populates the notification table on the main window.

13. CSS interface design

JavaFX platform allows for designing certain parts of the application using the CSS, Cascading Style Sheets language used for describing an element in a markup language. As it was important that all the information in the application is clear and easy to read, this syntax was used to design the notification table visible on the main window. In result, this technique allowed for

easy, unique design that was impossible with using the SceneBuilder program described in point **2.**

```
1
2   .table-row-cell {
3       -fx-background-color: transparent;
4   }
5
6   .table-row-cell .text {
7          -fx-fill: black ;
8   }
9
10  .table-row-cell:up
11  {
12      -fx-background-color: #007054;
13  }
14
15  .table-row-cell:down {
16      -fx-background-color: #A30029;
17  }
18
19
20
```

Figure 20. The application.css file responsible for designing the notification table.

```
public void setupTable() {
    LinkedList<TransactionItem> tableview=new LinkedList<TransactionItem>(SalesProducts);
    ObservableList<TransactionItem> list=FXCollections.observableArrayList(tableview);
    product.setReorderable(false);
    price.setReorderable(false);
    quantity.setReorderable(false);
    barcode.setReorderable(false);
    name.setSortable(false);
    date.setSortable(false);
    product.setStyle("-fx-alignment: CENTER;");
    name.setStyle("-fx-alignment: CENTER;");
    price.setStyle("-fx-alignment: CENTER;");
    quantity.setStyle("-fx-alignment: CENTER;");
    barcode.setStyle("-fx-alignment: CENTER;");
    date.setStyle("-fx-alignment: CENTER;");
    week.setAlignment(Pos.CENTER);
    month.setAlignment(Pos.CENTER);
    product.setCellValueFactory(new PropertyValueFactory<TransactionItem, Product>("product"));
    name.setCellValueFactory(new PropertyValueFactory<Product, String>("name"));
    price.setCellValueFactory(new PropertyValueFactory<Product, Double>("price"));
    quantity.setCellValueFactory(new PropertyValueFactory<TransactionItem, Integer>("itemq"));
    barcode.setCellValueFactory(new PropertyValueFactory<Product, String>("barcode"));
    date.setCellValueFactory(new PropertyValueFactory<Product, String>("dateadded"));
    table.setItems(list);
}
```

Figure 21. setuptable() method in SalesHistoryController class, where a table is designed using CSS code in .setStyle and .setAligment methods.

Word count: 1010