

Criterion C:

Techniques Used:

1. Graphical User Interface
2. User Defined Methods
3. Database Implementation
4. Aggregation
5. Validation Checks
6. Arrays
7. Database Queries
8. File Handling
9. Overriding
10. Exception Handling
11. API Integration

1. Graphical User Interface (GUI)

GUI Libraries

Using the Netbeans IDE, My software is entirely presented through a graphical user interface created in Java using the Swing API. Various libraries were imported in order to provide me with different templates and functions that helped greatly in the design of the GUI. These libraries provided functions such as buttons, combo boxes and tables which allowed for convenient and easy input of data from the user. This helped achieve abstraction in my software as the user was only shown important information whereas all calculations and algorithms were hidden behind the GUI. This fulfills **success criteria 1** by allowing users to interface with the software easily even without having to learn prior how the software works.

```
import javax.swing.JOptionPane;  
import javax.swing.table.DefaultTableModel;  
import net.proteanit.sql.DbUtils;  
import javax.swing.*;
```

Figure 1. Java Swing library Imports used to create the GUI

JFrames and JPanels

In order to make the software navigation more convenient and user-friendly, a panel system was chosen for the GUI. Instead of having multiple JFrames that would each contain an individual class/module, my software would use only a singular JFrame which would instead have navigation buttons that would swap between the different JPanels.

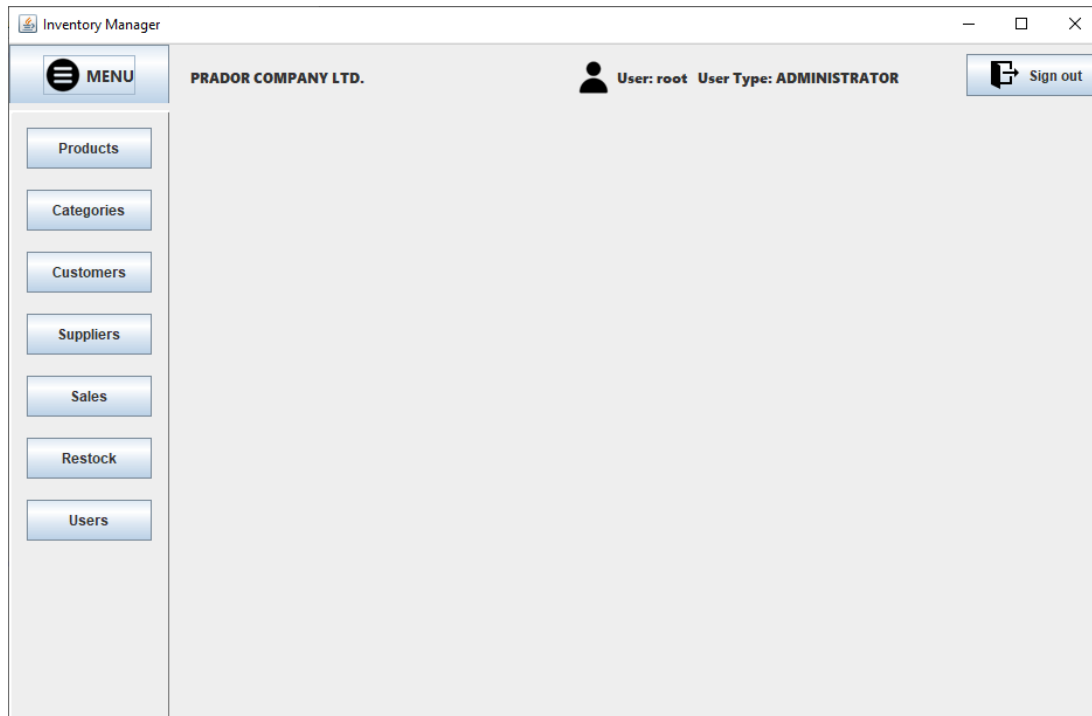


Figure 2. Home Dashboard JFrame

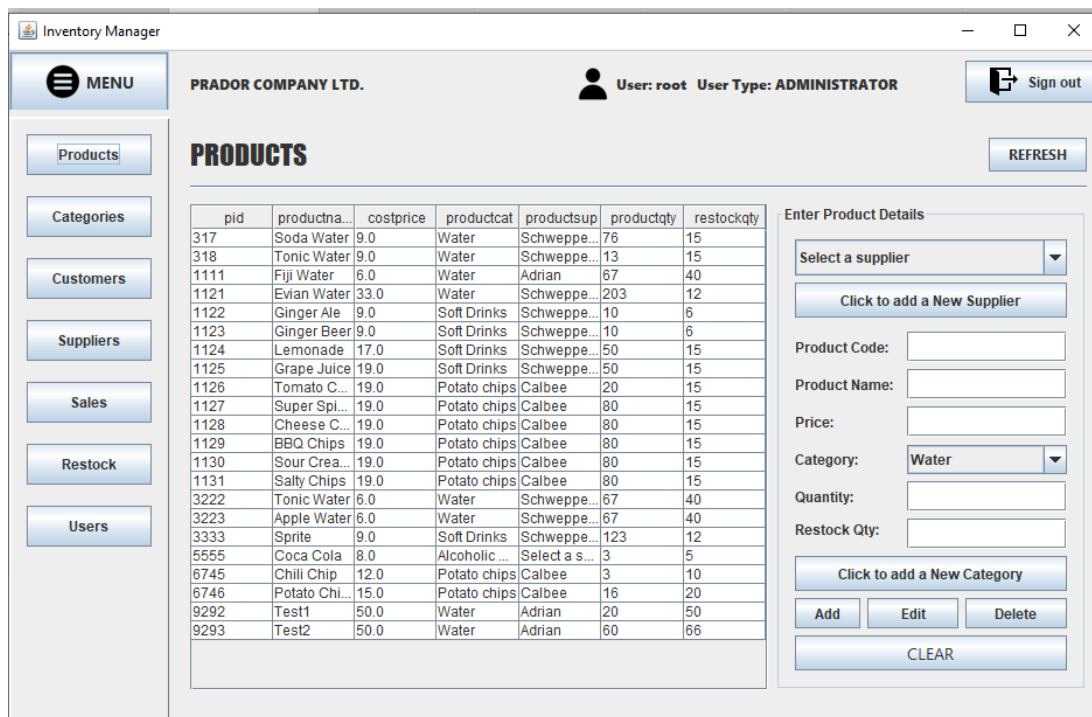


Figure 3. Product JPanel being displayed on Dashboard JFrame. Table is being displayed using the DBUtils library which pulls data from database into Swing JTable

2. User Defined Methods (OOP)

I have created methods that will perform certain tasks such as pulling data from the database and displaying them in the table. Defining methods is highly convenient as it reduces the amount of code needed and can be reused for certain tasks that are repeated many times.

Display Data Method

In order to pull the data from the database and display the tables on the GUI, a method was defined that would connect to the database and pull all data from a specific table. This method was called in the constructor of any classes with tables so that when the user initialized the panel, they could immediately see all the data present. This method was also called by any Add, Delete, Edit and refresh buttons so that the tables could update following a change by the user.

```
// Method to load data into table
public void loadDataSet() {
    try {
        st = Con.createStatement(); //Create Statement
        Rs = st.executeQuery( string: "select * from Products");
        // Select All Columns from Product Table
        productTable.setModel( dataModel: DbUtils.resultSetToTableModel( rs:Rs));
        // Add ResultSet data to Product Table on GUI
    }
    catch(Exception e){
        e.printStackTrace();
    }
}
```

Figure 4. LoadDataSet Method

Get Data Methods

These methods were defined in order to pull data from specific columns in specific tables on the database in order to populate the combo boxes which allow users to easily enter categories or suppliers for products, allowing interdependence between the different tables.

```

// Method to update combo box containing supplier names
public void getCategory() {
    CatCb.removeAllItems();
    //Clear Category ComboBox To Avoid Duplicate Entries
    try {
        st = Con.createStatement();
        String Query = "Select * from categories";
        //Select All Columns from Category Table
        Rs = st.executeQuery( string:Query);
        while (Rs.next()){
            String CatName = Rs.getString( string:"categoryname");
            CatCb.addItem( item:CatName);
            //Add all returned Category names to Category Combo Box
        }
    }
    catch (Exception e){
    }
}

```

Figure 5. getCategory Method is used to populate the category combo box when entering products.

3. Database Implementation

A database schema “inventory” was created using MySQL which would contain the tables that stored all the data used in this program. This was vital in order to achieve **success criteria 5** by allowing the permanent storage of data.

Database Initialisation

```

CREATE TABLE `categories` (
  `idcategories` int NOT NULL,
  `categoryname` varchar(45) NOT NULL,
  PRIMARY KEY (`idcategories`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

CREATE TABLE `customers` (
  `cid` int NOT NULL AUTO_INCREMENT,
  `fullname` varchar(45) NOT NULL,
  `location` varchar(45) NOT NULL,
  `phone` varchar(45) NOT NULL,
  PRIMARY KEY (`cid`)
) ENGINE=InnoDB AUTO_INCREMENT=10000 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

CREATE TABLE `products` (
  `pid` int NOT NULL,
  `productname` varchar(45) NOT NULL,
  `costprice` double NOT NULL,
  `productcat` varchar(45) DEFAULT NULL,
  `productsup` varchar(45) DEFAULT NULL,
  `productqty` int NOT NULL,
  `restockqty` int NOT NULL,
  PRIMARY KEY (`pid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

```

```

CREATE TABLE `salesinfo` (
  `salesid` int NOT NULL AUTO_INCREMENT,
  `date` varchar(45) NOT NULL,
  `productcode` varchar(45) NOT NULL,
  `customercode` varchar(45) NOT NULL,
  `quantity` int NOT NULL,
  `revenue` double NOT NULL,
  PRIMARY KEY (`salesid`)
) ENGINE=InnoDB AUTO_INCREMENT=2032 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

CREATE TABLE `suppliers` (
  `sid` int NOT NULL AUTO_INCREMENT,
  `fullname` varchar(45) NOT NULL,
  `location` varchar(45) NOT NULL,
  `mobile` varchar(10) NOT NULL,
  `email` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`sid`)
) ENGINE=InnoDB AUTO_INCREMENT=43244 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

CREATE TABLE `users` (
  `id` int NOT NULL,
  `name` varchar(45) NOT NULL,
  `location` varchar(45) NOT NULL,
  `phone` varchar(10) NOT NULL,
  `username` varchar(20) NOT NULL,
  `password` varchar(200) NOT NULL,
  `usertype` varchar(45) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

```

Figure 6. The SQL Script used to create the tables in the database. As seen, all tables have a primary key

Java Database Connectivity Driver (JDBC)

In order to connect to and communicate with the database, JDBC API would be used which is an interface between java programs and database management systems. The JDBC driver was installed as a library to allow my software to interact with the JDBC API and in turn with the database.

```

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;

```

Figure 7. JDBC Imports.

```

Connection Con = null;
PreparedStatement pst = null;
ResultSet Rs=null;
Statement st =null;

```

Figure 8. Initialization of Connection, Statement and ResultSet Objects

```

private void Connection(){ //Method To Connect to Database
    try{
        Class.forName( className:"com.mysql.jdbc.Driver"); //Get The JDBC Driver
        Con = DriverManager.getConnection( url: "jdbc:mysql://localhost:3306/inventory", user: "root", password: "root");
        //Connect to Database using credentials
    }catch(ClassNotFoundException e){ //Catch Error for missing Driver
        JOptionPane.showMessageDialog( parentComponent: this, message: "Driver not Found");
    }catch(SQLException e){ //Catch Error for database connection
        JOptionPane.showMessageDialog( parentComponent: this, message: "Failed to Connect to Database");
    }
}

```

Figure 9. Connection Method which is used to connect to the database.

4. Aggregation

An instance of the usage of aggregation in this software lies in the database in order to represent a part-whole relationship between two or more tables. For instance in the relationship between categories and products

```

CREATE TABLE `categories` (
  `idcategories` int NOT NULL,
  `categoryname` varchar(45) NOT NULL,
  PRIMARY KEY (`idcategories`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

CREATE TABLE `products` (
  `pid` int NOT NULL,
  `productname` varchar(45) NOT NULL,
  `costprice` double NOT NULL,
  `productcat` varchar(45) DEFAULT NULL,
  `productsup` varchar(45) DEFAULT NULL,
  `productqty` int NOT NULL,
  `restockqty` int NOT NULL,
  PRIMARY KEY (`pid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

```

Figure 10. The SQL statement of the two tables Categories and Products show that a category can have multiple products and each product belongs to one category. This is a part-whole relationship, which can be represented using aggregation. The “productcat” column in the “products” table is a foreign key that references the “categoryname” column in the “categories” table, indicating that each product belongs to one category.

5. Validation Checks

```
if(codeText.getText().isEmpty() && nameText.getText().isEmpty()) {  
    JOptionPane.showMessageDialog(parentComponent: this, message: "Missing Info, Please enter again");  
}  
else{
```

Figure 11. Validation Check for Input of Products

In order to validate the data being input by the user, If Else statements were used in conjunction with error boxes on the GUI to inform the user if any information was missing. This achieved **success criteria 13**

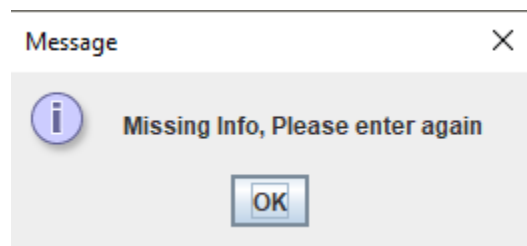


Figure 12. Error Message output on GUI

6. Arrays

As part of the validation process for creating a new account password, each character is tokenized into an index into an array.

```
private boolean validatePass(String pass) {  
    boolean hasNum = false;  
    boolean hasCap = false;  
    char[] characters = pass.toCharArray();  
  
    if (pass.length() < 8) {  
        return false;  
    }  
  
    for (int i = 0; i < pass.length(); i++) {  
        if (Character.isUpperCase(characters[i])) {  
            hasCap = true;  
        }  
  
        if (Character.isDigit(characters[i])) {  
            hasNum = true;  
        }  
    }  
  
    return hasNum && hasCap;  
}
```

Figure 13. The Validatepass() method is called when creating a new account which takes the String input of the password and tokenizes each character in the string into an individual index

in the array char[]. Using linear search with a for loop, the characters are checked with the appropriate validation conditions which then set two boolean values to true. If both boolean values are true, the input password is validated.

7. Database Queries

Simple Queries

Simple queries were extensively used in order to allow for adding, editing and deleting entries/ data from the database. Such simple queries were used to display data from the database without any clauses or specifications.

```
// Method to load data into table
public void loadDataSet() {
    try {
        st = Con.createStatement(); //Create Statement
        Rs = st.executeQuery( string: "select * from Products");
        // Select All Columns from Product Table
        productTable.setModel( dataModel: DbUtils.resultSetToTableModel( rs:Rs));
        // Add ResultSet data to Product Table on GUI
    }
    catch(Exception e){
        e.printStackTrace();
    }
}
```

Figure 14. Simple Query used to pull data from all columns in Products Table

pid	productna...	costprice	productcat	productsup	productqty	restockqty
317	Soda Water	9.0	Water	Schweppes	76	15
318	Tonic Water	9.0	Water	Schweppes	13	15
1111	Fiji Water	6.0	Water	Adrian	67	40
1121	Evian Water	33.0	Water	Schweppes	203	12
1122	Ginger Ale	9.0	Soft Drinks	Schweppes	10	6
1123	Ginger Beer	9.0	Soft Drinks	Schweppes	10	6
1124	Lemonade	17.0	Soft Drinks	Schweppes	50	15
1125	Grape Juice	19.0	Soft Drinks	Schweppes	50	15
1126	Tomato C...	19.0	Potato chips	Calbee	20	15
1127	Super Spic...	19.0	Potato chips	Calbee	80	15
1128	Cheese C...	19.0	Potato chips	Calbee	80	15
1129	BBQ Chips	19.0	Potato chips	Calbee	80	15
1130	Sour Crea...	19.0	Potato chips	Calbee	80	15
1131	Salty Chips	19.0	Potato chips	Calbee	80	15
3222	Tonic Water	6.0	Water	Schweppes	67	40
3223	Apple Water	6.0	Water	Schweppes	67	40
3273	Perrier Wa...	9.0	Water	Schweppes	23	10
3279	Perrier Wa...	9.0	Water	Schweppes	23	10
3333	Sprite	9.0	Soft Drinks	Schweppes	123	12
5555	Coca Cola	8.0	Alcoholic B...	Select a s...	3	5
6745	Chili Chip	12.0	Potato chips	Calbee	3	10
6746	Potato Chi...	15.0	Potato chips	Calbee	16	20
9292	Test1	50.0	Water	Adrian	20	50
9293	Test2	50.0	Water	Adrian	60	66

Figure 15. Product Data pulled from Database and displayed on GUI table.

Complex Queries

Complex queries were used to retrieve certain data from the database such as goods that fell under restocking quantity or to search for specific data given a primary key or foreign key.

```
pst = Con.prepareStatement( string:"select * from products where productsup=? AND productqty < restockqty");
//Query to Select All products from Supplier X and the Restock Quantity is greater than the Quantity on Hand
```

Figure 16. Complex Query used in the Auto Restocking function which searches for products from a specific supplier and where the product quantity is less than the restock quantity

Parameterized Queries

Parameterized queries, also known as prepared statements, are a way to execute SQL queries with parameters that are supplied at runtime, instead of embedding the values directly into the SQL statement. This allows users to input data through the program during runtime using input boxes and text fields etc. By using parameterized queries, we can ensure that any input parameters are properly validated prior to being entered into the database. These queries were used for Add and Edit functions.

```
if(codeText.getText().isEmpty() && nameText.getText().isEmpty()){
    JOptionPane.showMessageDialog(parentComponent: this, message: "Missing Info, Please enter again");
    //Validation Presence Check
}

else{
    try{
        PreparedStatement Save = Con.prepareStatement("insert into Products Values(?, ?, ?, ?, ?, ?, ?)");
        //Parameterized Query to Add into all Columns in Products Table
        Save.setInt(1, Integer.valueOf(codeText.getText()));
        Save.setString(2, nameText.getText());
        Save.setString(3, costText.getText());
        Save.setString(4, CatCb.getSelectedItem().toString());
        Save.setString(5, suppCombo.getSelectedItem().toString());
        Save.setString(6, qtytext.getText());
        Save.setString(7, restockqty.getText());
        //Fetch All Data from textfields and input boxes

        int row = Save.executeUpdate();
        //Execute query
        JOptionPane.showMessageDialog(parentComponent: this, message: "Item Added");
        //Success Message
        loadDataSet();
        //Refresh Table on GUI to Display new entry
    }
}
```

Figure 17. Add Product method which uses a parameterized query in order to take input from textfields and enter into corresponding columns

Result Grid							
		Filter Rows:		Edit:		Export/Import:	
						Wrap Cell Content:	
	pid	productname	costprice	productcat	productsup	productqty	restockqty
▶	317	Soda Water	9	Water	Schweppes	76	15
	318	Tonic Water	9	Water	Schweppes	13	15
	1111	Fiji Water	6	Water	Adrian	67	40
	1121	Evian Water	33	Water	Schweppes	203	12
	1122	Ginger Ale	9	Soft Drinks	Schweppes	10	6
	1123	Ginger Beer	9	Soft Drinks	Schweppes	10	6
	1124	Lemonade	17	Soft Drinks	Schweppes	50	15
	1125	Grape Juice	19	Soft Drinks	Schweppes	50	15
	1126	Tomato Chips	19	Potato chips	Calbee	20	15
	1127	Super Spicy Chips	19	Potato chips	Calbee	80	15
	1128	Cheese Chips	19	Potato chips	Calbee	80	15
	1129	BBQ Chips	19	Potato chips	Calbee	80	15
	1130	Sour Cream Chips	19	Potato chips	Calbee	80	15
	1131	Salty Chips	19	Potato chips	Calbee	80	15
	3222	Tonic Water	6	Water	Schweppes	67	40
	3223	Apple Water	6	Water	Schweppes	67	40
	3273	Perrier Water	9	Water	Schweppes	23	10
	3279	Perrier Water	9	Water	Schweppes	23	10
	3333	Sprite	9	Soft Drinks	Schweppes	123	12
	5555	Coca Cola	8	Alcoholic Be...	Select a su...	3	5
	6745	Chili Chip	12	Potato chips	Calbee	3	10
	6746	Potato Chips	15	Potato chips	Calbee	16	20
	9292	Test1	50	Water	Adrian	20	50
	9293	Test2	50	Water	Adrian	60	66
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 18. Data being viewed on the backend database demonstrating the addition of products in their corresponding column as defined in the parameterized column.

8. File Handling

Using the iText library, a PDF document was created as a sales report for the software. A pdf document was created on the user's local system and then written for formatting. After this, data from the database was queried and written into the corresponding table on the PDF.

```
import com.itextpdf.text.Document;
import com.itextpdf.text.Font;
import com.itextpdf.text.Paragraph;
import com.itextpdf.text.Phrase;
import com.itextpdf.text.pdf.PdfPCell;
import com.itextpdf.text.pdf.PdfPTable;
import com.itextpdf.text.pdf.PdfWriter;
```

Figure 19. Itext library imports used to create and write a PDF report.

```

if (jDateChooser1.getDate() == null){
    JOptionPane.showMessageDialog(parentComponent: null, message: "Please Select A date To Generate Report For");
    //Validation Check For Date Input
}
else{
    Date date = jDateChooser1.getDate();
    String strDate = DateFormat.getDateInstance().format(date);

    try {
        pst = Con.prepareStatement( string:"select * from salesinfo where date=?");
        pst.setString( i:1, string:strDate);
        Rs= pst.executeQuery();
        //Parameterized Query to select all Sales from specific Date

        String filename = "/Users/adrian/csia/SalesReport.pdf";
        //File Directory of PDF and Name of PDF
        Document doc = new Document();

        PdfWriter.getInstance( document: doc, new FileOutputStream( name: filename));
        doc.open();

        Font f = new Font( family:Font.FontFamily.HELVETICA, size:20, style:Font.BOLD);
        Font f_head = new Font( family:Font.FontFamily.HELVETICA, size:12, style:Font.BOLD);
        Font f_cell = new Font( family:Font.FontFamily.HELVETICA, size:12);

        PdfPTable table = new PdfPTable( numColumns: 5);
        PdfPCell head = new PdfPCell(new Phrase( string:"Sale ID", font:f_head));
        table.addCell( cell:head);
        head = new PdfPCell(new Phrase( string:"Date", font:f_head));
        table.addCell( cell:head);
        head = new PdfPCell(new Phrase( string:"Product Code", font:f_head));
        table.addCell( cell:head);
        head = new PdfPCell(new Phrase( string:"Customer Code", font:f_head));
        table.addCell( cell:head);
        head = new PdfPCell(new Phrase( string:"Revenue", font:f_head));
        table.addCell( cell:head);
        table.setHeaderRows( headerRows: 1);
    }
}

```

```

Paragraph p = new Paragraph("Your Sales Report for " + strDate, font:f);
p.setSpacingAfter( spacing:36f);
doc.add( element:p);
//Adding Content to PDF Document such as table for sales data and Footer

while (Rs.next()) {
    PdfPCell cell = new PdfPCell(new Phrase( string:Rs.getString( string:"salesid"), font:f_cell));
    table.addCell(cell);

    cell = new PdfPCell(new Phrase( string:Rs.getString( string:"date"), font:f_cell));
    table.addCell(cell);

    cell = new PdfPCell(new Phrase( string:Rs.getString( string:"productcode"), font:f_cell));
    table.addCell(cell);

    cell = new PdfPCell(new Phrase( string:Rs.getString( string:"customercode"), font:f_cell));
    table.addCell(cell);

    cell = new PdfPCell(new Phrase( string:Rs.getString( string:"revenue"), font:f_cell));
    table.addCell(cell);
}
//Inputting Sales Data from Database into PDF
calculatetotal();
//Calculate Total Revenue Method
table.setSpacingAfter( spacing:72f);
doc.add( element: table);

Paragraph cost = new Paragraph("Total Revenue: " + total + " HKD", font:f);
p.setSpacingAfter( spacing:36f);
doc.add( element: cost);

doc.close();
//Close Document
JOptionPane.showMessageDialog( parentComponent: this, message: "PDF has been created!");
}
catch (Exception e) {
    JOptionPane.showMessageDialog( parentComponent: this, message: e);
}
}

```

Figure 20. Commented Code for the creation and writing of the sales report PDF

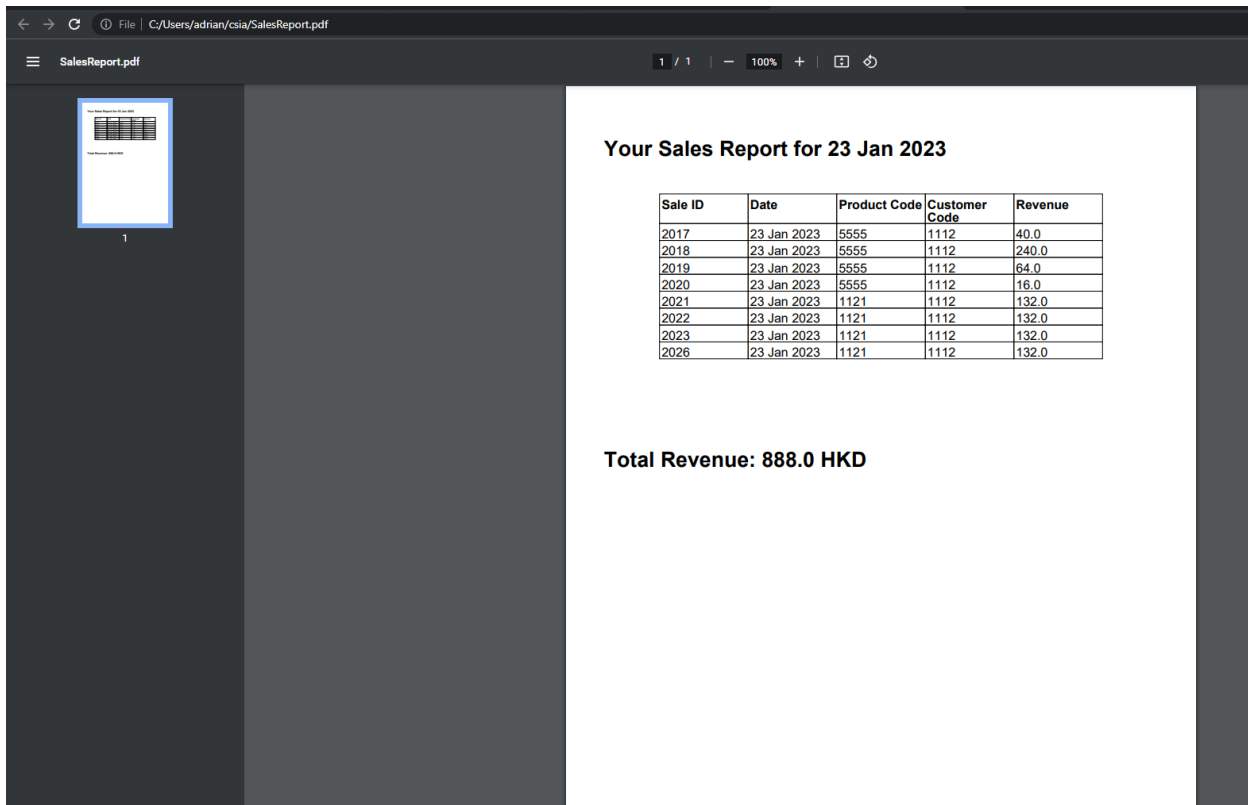


Figure 21. Sales Report PDF displaying the goods sold to which customer and the total revenue of the day.

9. Overriding

The `getPasswordAuthentication()` method is a part of the `java.net.Authenticator` class in Java, which provides a way to authenticate HTTP connections. In order to use the `getPasswordAuthentication()` method, I was required to subclass the `Authenticator` class and override the method. This is because the default implementation of `getPasswordAuthentication()` provided by the library always returns null, indicating that no authentication is required.

```
Session session = Session.getDefaultInstance(props:properties, new Authenticator() {
    @Override
    protected PasswordAuthentication getPasswordAuthentication() {
        return new PasswordAuthentication(userName:username, password);
        //Email Password Authentication Method Is overwritten by New Method
    }
});
```

Figure 22. Usage of `@Override` indicates that the initial method is replaced with my own method.

10. Exception Handling

Using Try Catch clauses, proper error messages could be displayed for instance connection exceptions which are exceptions that occur when attempting to connect to a database, such as invalid login credentials or a connection timeout. These exceptions allow the software to avoid crashing even if there is an undesired or unexpected output and allow me to see where there was an issue.

```
private void Connection(){ //Method To Connect to Database
    try{
        Class.forName( className:"com.mysql.jdbc.Driver"); //Get The JDBC Driver
        Con = DriverManager.getConnection( url:"jdbc:mysql://localhost:3306/inventory", user:"root", password:"root");
        //Connect to Database using credentials
    }catch(ClassNotFoundException e){ //Catch Error for missing Driver
        JOptionPane.showMessageDialog( parentComponent:this, message:"Driver not Found");
    }catch(SQLException e){ //Catch Error for database connection
        JOptionPane.showMessageDialog( parentComponent:this, message:"Failed to Connect to Database");
    }
}
```

Figure 23. Two Catch cases in order to differentiate between different errors that may occur during connection to the database.

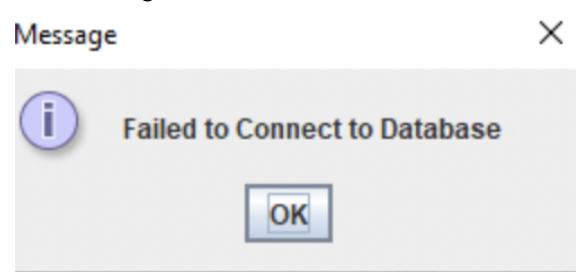


Figure 24. Resulting Output message if the inputted database password is incorrect.

11. API Integration

As a requirement stated by the client and part of **success criteria 12**, an emailing function would be required that would send restock requests to suppliers automatically. To achieve this, the javax.mail package was used which is a part of the JavaMail API. The following libraries were used.

```
import javax.mail.Authenticator;
import javax.mail.Message;
import javax.mail.Message.RecipientType;
import javax.mail.Multipart;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeBodyPart;
import javax.mail.internet.MimeMessage;
import javax.mail.internet.MimeMultipart;
```

Figure 25. Libraries used to implement the JavaMail API

Before the sender could login with their own email address, the mail server properties would need to be specified. Per the client's request, the Gmail mail server was used.

```
Properties properties = new Properties();
properties.put(key:"mail.smtp.auth", value:"true");
//Specifies whether authentication is required by SMTP server, in this case True
properties.put(key:"mail.smtp.starttls.enable", value:"true");
//Specifies whether or not to use STARTTLS encryption
properties.put(key:"mail.smtp.host", value:"smtp.gmail.com");
//Hostname of the SMTP server used to send outgoing emails
properties.put(key:"mail.smtp.port", value:"587");
//Port number used by SMTP server
```

Figure 26. Commented Code initializing Mail Server Properties of Gmail

```
Message message = new MimeMessage(session);

message.setSentDate(date: cur);
message.setSubject("Restock Order for " + cur);
message.setContent(o: "Sent from Inventory Management System.", string: "text/plain");
message.setFrom(new InternetAddress(address: username));
message.setRecipient(type: RecipientType.TO, new InternetAddress(address: receipient));

Multipart mp = new MimeMultipart();
MimeBodyPart body = new MimeBodyPart();

File list = new File(pathname: "/Users/adrian/csia/RestockReport.pdf");
DataSource ds = new FileDataSource(file: list);
body.setDataHandler(new DataHandler(ds));
body.setFileName(filename: list.getName());

mp.addBodyPart(part: body);

message.setContent(multipart: mp);
Transport.send(msg: message);
JOptionPane.showMessageDialog(parentComponent: this, "Restock List has been sent to "+receipient);
}
```

Figure 27. Email contents, recipient and PDF attachment are added to the email.

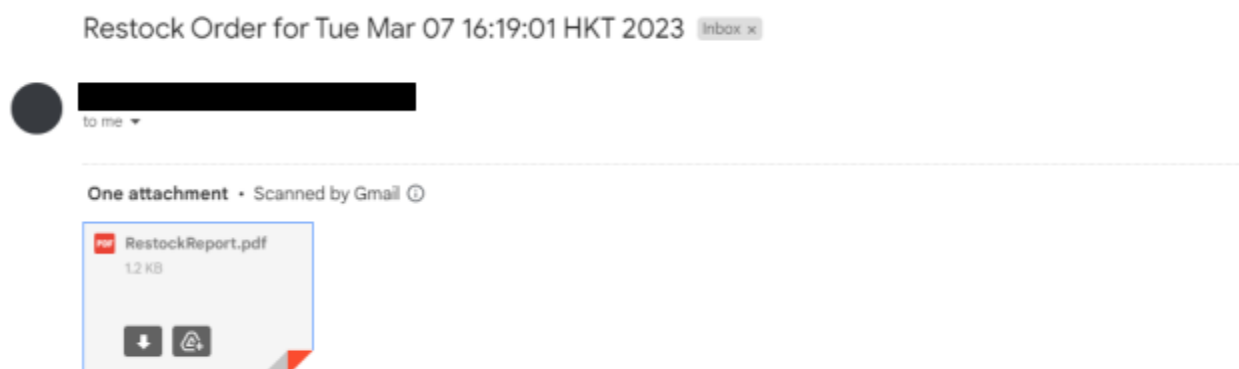


Figure 28. Sample of Email sent to the recipient containing the product to restock and quantity. Email also contains a subject, displaying the date and time of request.


Word Count: 922

References

“Java GUI Tutorial - Make a GUI in 13 Minutes.” *YouTube*, YouTube, 6 Feb. 2020, www.youtube.com/watch?v=5o3fMLPY7qY&t=4s&ab_channel=AlexLee.

“#28 Java Swing Tutorial | Jpanel.” *YouTube*, YouTube, 9 Aug. 2020, www.youtube.com/watch?v=GCmLoEgUUSg&ab_channel=MukulSainiSkills.

“InventoryManagementSystem” *GitHub*, 3 Sep 2021 <https://github.com/AsjadIqbal/InventoryManagementSystem>

“Java Panels .

” *YouTube*, YouTube, 12 Aug. 2020, www.youtube.com/watch?v=dvzAug-YDpM&ab_channel=BroCode.

“Why We Use Class.forName(‘Oracle.jdbc.driver.OracleDriver’) While Connecting to a Database?” *Stack Overflow*, stackoverflow.com/questions/20078586/why-we-use-class-forname-oracle-jdbc-driver-oracledriver-while-connecting-to.

“Java and Mysql - How to Insert Update Delete and Display Data in JTABLE [Part-1] [with Source Code].” *YouTube*, YouTube, 19 Feb. 2017, www.youtube.com/watch?v=eybhdJFhQyE&ab_channel=RunCodes.

“Java and Mysql - How to Insert Update Delete and Display Data in Jtable?[Part-2] [with Source Code].” *YouTube*, YouTube, 19 Feb. 2017, www.youtube.com/watch?v=NgRSW7ZOjLs&ab_channel=RunCodes.

Adi, et al. “Getting Error in Java Mail Api Passwordauthentication Method.” *Stack Overflow*, stackoverflow.com/questions/19973456/getting-error-in-java-mail-api-passwordauthentication-method.

“Sending Email Using the Gmail API and Java.” *YouTube*, YouTube, 5 Nov. 2022, www.youtube.com/watch?v=xtZI23hxetw&t=885s&ab_channel=SebastianDaschner.

“Chapter 7 Examples of Common Queries.” *MySQL*, dev.mysql.com/doc/mysql-tutorial-excerpt/8.0/en/examples.html.

“Java - Generate PDF Using Java ITEXTPDF, Mysql Database Dynamically.” *YouTube*, YouTube, 11 Mar. 2018, www.youtube.com/watch?v=Zg7IS5sPN0M&ab_channel=jinujawadm.

