GROUP ASSIGNMENT

TECHNOLOGY PARK MALAYSIA

CT046-3-2-SDM

OBJECT ORIENTED JAVA PROGRAMMING

HAND OUT DATE: 1 DECEMBER 2023

HAND IN DATE: 23 FEBRUARY 2024

INTAKE CODE: APD2F2311CS(DA) / APU2F2311CS(DA) / APU2F2311CS

TEAM MEMBERS	**TP NUMBER**
KERTANAA A/P KUMAR	TP072664
HOOUSHIANEE HARHANGI	TP066080
ABDUL MUHAIMIN AMAN	TP069510
AREEBAH IRFAN MOHAMMED	TP072275

Table of Contents

1.0 INTRODUCTION

In this project, we will create a window-based system to manage sales orders, invoicing, and reports for Yoyo-Furniture, a furniture firm. The system will be constructed in an object-oriented manner, with numerous objects representing real-world things like as sales orders, invoices, reports, and customers. The system will enable users to execute a variety of tasks, such as processing sales orders, producing invoices, generating reports, managing user accounts, updating and deleting, generating invoices, listing and modifying sales orders and so on.

With a user-friendly interface, the GUI-driven system will enable people to engage with it. The system will be built to handle a number of situations, such as processing sale orders, creating invoices using the data from the sales orders, producing reports that shed light on the sales success of the business, and maintaining user accounts.

We will use polymorphism, inheritance, and encapsulation—aspects of object-oriented programming—to create the system. Because of its modular architecture, the system will be simple to scale and maintain. Additionally, the system will be built with security in mind, incorporating user identification and authorization protocols.

An description of the system's architecture, implementation specifics, and samples of code showcasing its object-oriented capabilities are all included in this report. In addition, we will include images of the user interface and a thorough explanation of all the features that the system will support.

2.0 PROPOSED SYSTEM

A window-based system that mimics the workflow of handling sale orders, invoicing, and reports is the proposed system for Yoyo-Furniture. An object-oriented design methodology will be used to create the system, with different objects standing in for actual real-world things including users, reports, invoices, and sale orders. Users with varying levels of access, including officers, administrators, and salespeople, will be able to handle sale orders and reports through the system.

The officer will be able to use features including creating reports, monitoring the status of sold products, processing sale orders when a transaction is approved or closed, and maintaining personal accounts. In addition to searching and editing sale orders, the officer will be able to submit sales for production, issue sale invoices, and generate reports like the work-done and approved/closed sale reports.

The administrator will be able to perform tasks such as generating reports and handling employee and personal accounts. In addition to creating reports like the work done report and the approved/closed sale report, the administrator will have the ability to manage the profiles of officers and salesperson

The salesperson will be able to manage personal profiles, sale order quotations, and view a list of all personal sales orders. The salesperson will be able to generate, alter, remove, and search sale order quotations, as well as list of all disapproved and authorized personal sales orders.

We shall list the different properties required to describe the entities (sale order, invoice, report, user,) in the solution architecture. Additionally, we will list and include the techniques required for every entity, including report generation, modification, removal, and search. The system will verify data integrity and look for duplicate entries.

To get at the object-oriented design principles, we will logically assume certain functional needs. For permission and authentication reasons, for instance, we will assume that every user has a distinct username and password. Additionally, we will presume that every sale order has a distinct ID in order to avoid duplication and guarantee precise tracking.

3.0 SCOPE

1. For a furniture firm, design and construct a window-based system to manage selling orders, invoicing, and reports.

2. To represent real-world elements like sale orders, invoices, reports, and users, utilize an object-oriented approach.

3. Assign distinct access privileges to officers, administrators, and salespeople.

4. Put into practice features like report generation, modifying, removal, and searching.

4.0 OBJECTIVES

1. Make the management of sale orders, invoicing, and reports easier to use using an intuitive UI.

2. Use object-oriented design concepts to create representations of real-world objects.

3. Implementing access permissions and authentication will help to ensure data security and integrity.

4. Execute features including report generation, editing, deletion, and searching.
   To make sure the system is bug-free and satisfies functional requirements, test it.

5.0 LIMITATIONS

1. The system won't be linked to an actual database or production environment; instead, it will be a simulation.
2. In a real-world scenario, the system might not account for every event that could occur.
3. Because the system is a simulation rather than an application that is ready for production, it can have constraints with regard to performance, scalability, and security.
4. It's probable that not every edge case and error handling situation is covered by the system.
5. It's possible that the system cannot be adjusted or configured to satisfy certain company needs
6. It's possible that the system hasn't been thoroughly tested and has errors or problems that weren't discovered during development.
7. It's possible that not all hardware setups or operating systems will work with the system.
8. Users with impairments might not be able to access the system completely.
9. It's possible that the system isn't completely adjusted for varying screen sizes and resolutions.
10. It's possible that the system does not adhere to all rules and industry norms.
11. It's possible that the system can't manage a lot of data or many users at once.
12. There's a chance that the system won't work with other apps or platforms.
13. It's possible that the system can't manage processes or complicated business rules.
14. It's possible that the system cannot deliver analytics or data in real time.
15. Requirements for translation or internationalization could be beyond the system's capabilities.

6.0 SYSTEM DESIGN

6.1 USE CASE DIAGRAM (SYSTEM ACTIVITY) – Kertanaa A/P Kumaar

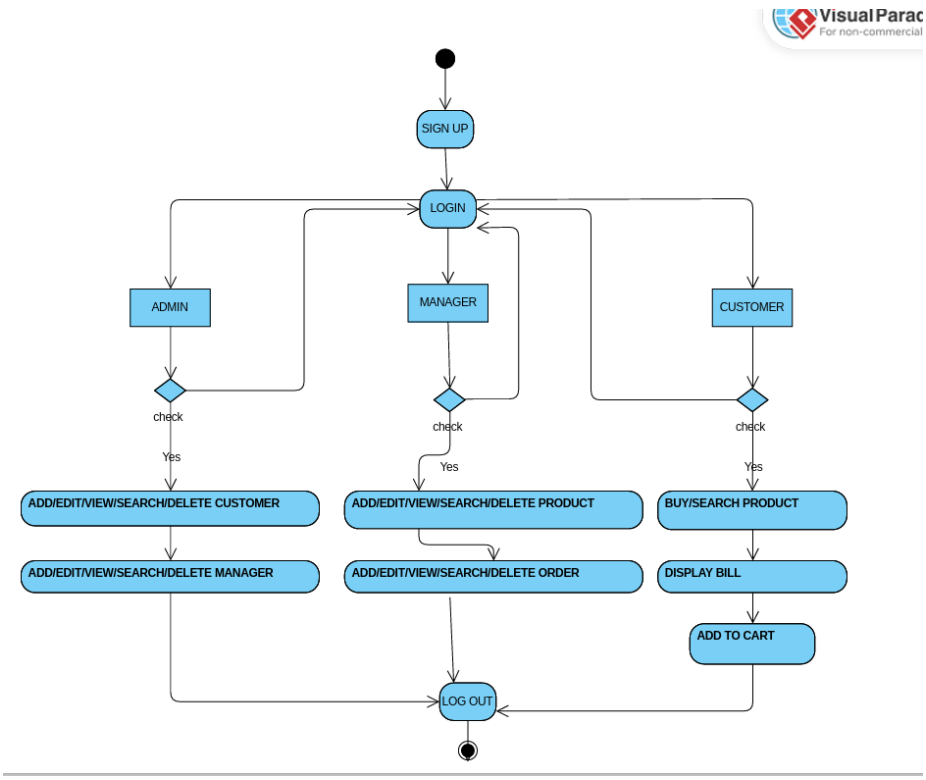


Figure 1 shows the Activity Diagram

System users has to login to the system using valid username and password. Depending on their categories so that they will be able to open their respective pages.

6.2 USE CASE DIAGRAM (CUSTOMER)

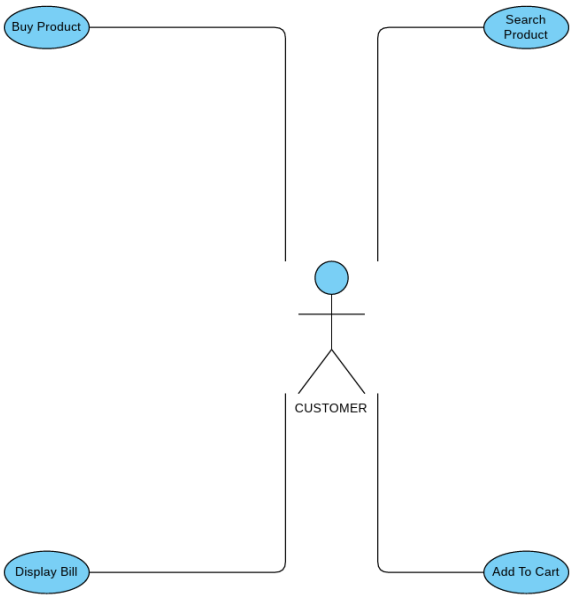


Figure 2 shows the Use Case Diagram of Customer

According to the use case diagram, the Customer role's interactions are mostly focused on product browsing, ordering, and shipping tracking. Customers may go through the products that are offered, choose the things that they want to buy, and then place their purchases. They may also track shipments to keep an eye on the delivery process and examine the status of their orders.

6.3 USE CASE DIAGRAM (MANAGER)

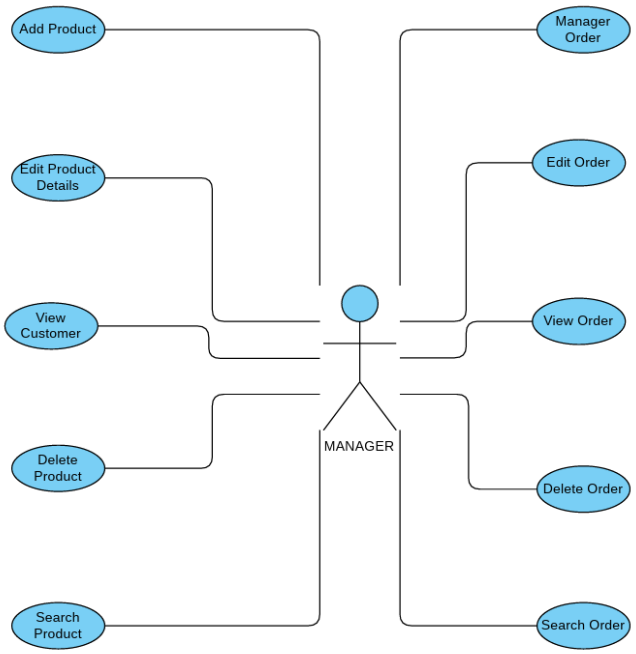


Figure 3 shows the Use Case Diagram of Manager

The Manager job holds more power and is largely responsible for approving or rejecting sales orders. After reviewing outstanding sales orders, managers decide whether to approve or reject them based on several factors, including availability, cost, and client preferences. Furthermore, managers have access to sales reports, which give them insightful knowledge about systemic patterns and overall sales success.

6.4 USE CASE DIAGRAM (ADMIN)

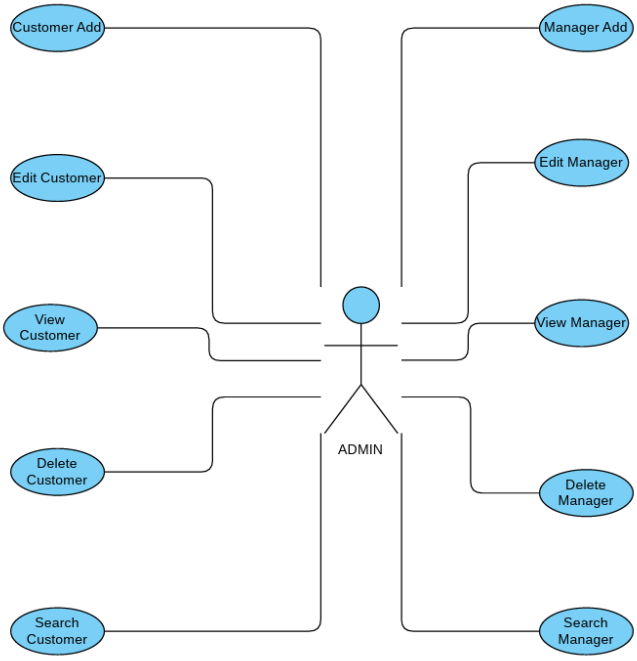
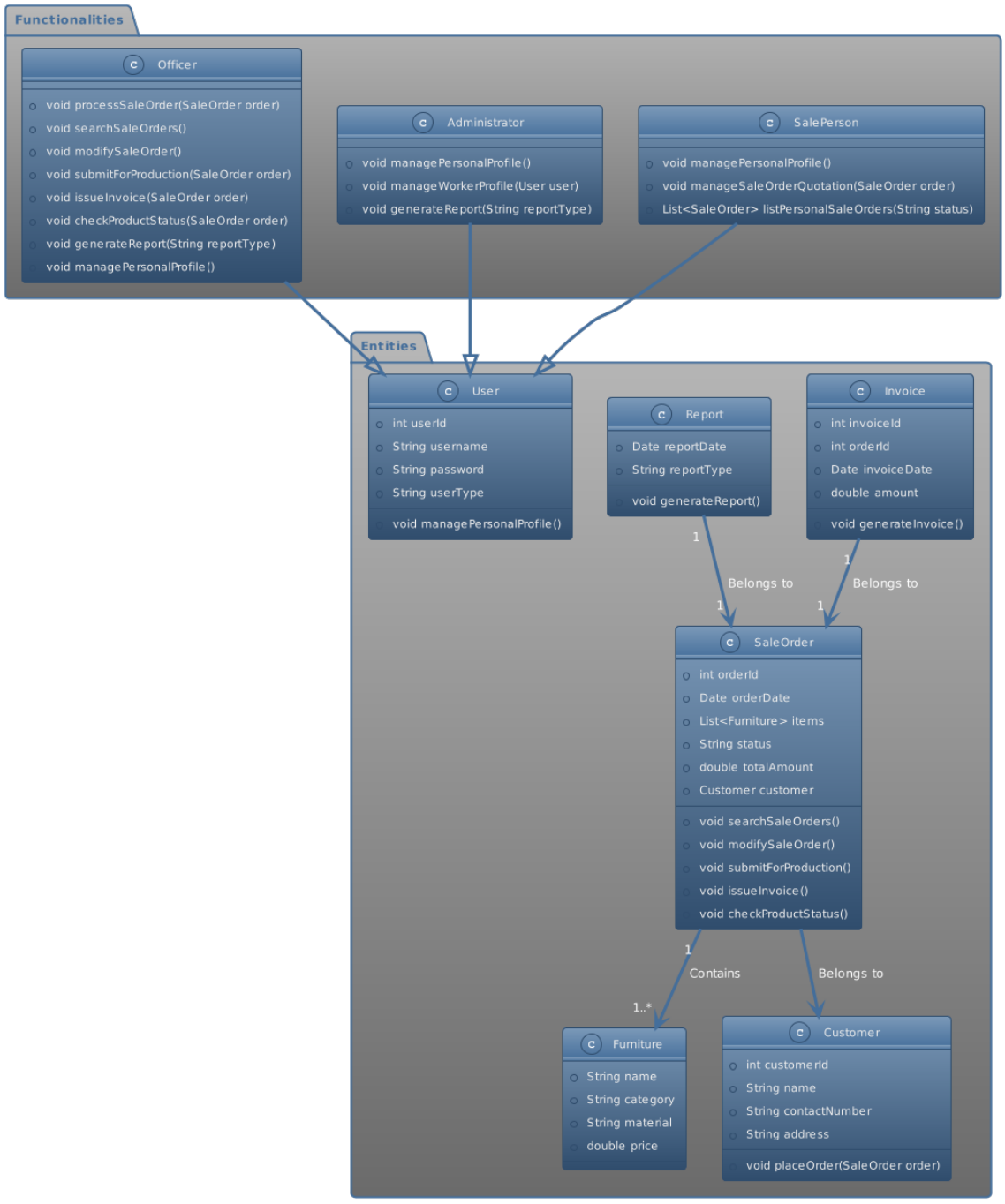


Figure 4 shows the Use Case Diagram of Admin

In contrast, the admin function oversees overseeing and managing the system. Administrators have the power to add, edit, and remove employees from the system, such as officers and salespeople, from their profiles as needed. Moreover, administrators have access to reports and sales data, which gives them the ability to assess system performance and decide after gaining knowledge.

6.5 CLASS DIAGRAM (SYSTEM ACTIVITY) – Abdul Muhaimin Aman



6.5.1 CLASS DIAGRAM DESCRIPTION

Entity/Functionality	Attributes	Brief Description
Officer	- userID - username - password - userType (officer) 	Represents an officer with access to specific functionalities, such as processing sale orders, generating reports, and managing personal profiles.
Administrator	- userID - username - password - userType (administrator) 	Represents an administrator with access to managing personal profiles, worker profiles, and generating reports.
Customer	- customerID - name - contactNumber - address 	Represents a customer, with attributes for customer ID, name, contact number, and address.
Furniture	- name - category - material - price 	Represents a furniture product, with attributes for name, category, material, and price.
Report	- reportDate - reportType 	Represents a report, with attributes for report date and report type.
Sales Invoice	- invoiceID - invoiceDate - amount - order 	Represents a sales invoice, with attributes for invoice ID, invoice date, amount, and order.

Entity/Functionality	**Attributes**	**Brief Description**
processSaleOrder(SaleOrder order)	- order 	A functionality for officers to process a sale order upon sale approval or closed sale.
generateReport(String reportType)	- reportType 	A functionality for officers and administrators to generate reports based on report type.
managePersonalProfile()	- user 	A functionality for users to manage their personal profile.
placeOrder(SaleOrder order)	- order - customer 	A functionality for customers to place an order by providing the order and customer details.

7.0 CONCEPTS USED

7.1 ABDUL MUHAIMIN AMAN

7.1.0 ENCAPSULATION

```
1  package oodjaman;
2
3  import javax.imageio.ImageIO;
10
11 public class oodjaman2 extends JFrame {
12
13     private AdministratorProfile administratorProfile;
14
15     private JTextField emailField;
16     private JTextField phoneNumberField;
17     private JTextField addressField;
18     private JButton saveButton;
19     private JButton goBackButton;
20
21     public oodjaman2() {
22         administratorProfile = AdministratorProfileManager.readProfile();
23
24         setTitle("Administrator Profile");
25         setSize(400, 300);
26         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
27
28         createUI();
29     }
```

Encapsulation is implemented in the oodjaman2 class using private methods (createUI(), saveChanges(), createStyledButton(String text)), private inner class (BackgroundPanel), and private fields (administratorProfile, emailField, phoneNumberField, addressField, saveButton, goBackButton). These aspects are protected from external classes by being marked as private, which limits access to the class itself and hides the internal state and implementation details. By enabling limited access to the class's functionality and encouraging information concealment, this encapsulation helps to create a more modular and manageable construction.

7.1.1 POLYMORPHISM

```
54
55          JTable reportTable = new JTable(tableModel) {
56              @Override
57              public Component prepareRenderer(TableCellRenderer renderer, int row, int column) {
58                  Component comp = super.prepareRenderer(renderer, row, column);
59
60                  // Handle selection state
61                  if (isRowSelected(row)) {
62                      comp.setBackground(new Color(70, 130, 180)); // Light blue for selected row
63                  } else {
64                      comp.setBackground(row % 2 == 0 ? Color.WHITE : new Color(255, 255, 255)); // Alterna
65                  }
66
67                  return comp;
68              }
69          };
70
```

This code snippet overrides the prepareRenderer function to adjust table cell rendering based on selection status and row index. This is an example of polymorphism, as the overridden method offers a customized implementation inside the context of ClosedSaleReport.

7.1.2 INHERITANCE

```
    // Custom JPanel with background image
    private class BackgroundPanel extends JPanel {
        private Image backgroundImage;

        public BackgroundPanel(String imagePath) {
            try {
                backgroundImage = ImageIO.read(new File(imagePath));
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        @Override
        protected void paintComponent(Graphics g) {
            super.paintComponent(g);
            if (backgroundImage != null) {
                g.drawImage(backgroundImage, 0, 0, getWidth(), getHeight(), this);
            }
        }
    }


    // Create a blue gradient background for JOptionPane
    private JPanel createBlueGradientBackground() {
        JPanel panel = new JPanel() {
            @Override
            protected void paintComponent(Graphics g) {
                super.paintComponent(g);
                Graphics2D g2d = (Graphics2D) g;
                int w = getWidth();
                int h = getHeight();
                Color color1 = new Color(70, 130, 180); // Light blue
                Color color2 = new Color(30, 60, 90);   // Dark blue
                GradientPaint gp = new GradientPaint(0, 0, color1, 0, h, color2);
                g2d.setPaint(gp);
                g2d.fillRect(0, 0, w, h);
            }
        };
        return panel;
    }
}
```

Java inheritance is demonstrated by the BackgroundPanel class and the createBlueGradientBackground function, which both extend or generate a subclass of the JPanel class. The custom panels offers extra or altered functionality while inheriting the JPanel class's attributes and behaviours.

7.2 KERTANAA A\P KUMAAR

7.2.0 ENCAPSULATION

```
public class Manage_Personal_Profile_Page extends JFrame {

    private static final long serialVersionUID = 1L;
    private JPanel contentPane;
    private JTextField textField_Name;
    private JTextField textField_Position;
    private JTextField textField_Email;
    private JTextField textField_PhoneNumber;
    private JTextField textField_Address;
```

From the code shown above, all the variables have been declared with a private access method which promotes encapsulation that hide all the data from being modified by non-relevant objects, as opposed to structured programming in that data will act as global data without any privilege access method to control the access. Furthermore, for above variables to be accessed, mutator and access method must be implemented.

7.2.1 INHERITANCE:

```
public Manage_Personal_Profile_Page() {
    setTitle("Manage Personal Profile Page");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 501, 393);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    setContentPane(contentPane);
    contentPane.setLayout(null);

    JLabel lblNewLabel_Name = new JLabel("Name:");
    lblNewLabel_Name.setFont(new Font("TIMES NEW ROMAN", Font.BOLD, 14));
    lblNewLabel_Name.setBounds(145, 14, 87, 24);
    contentPane.add(lblNewLabel_Name);

    textField_Name = new JTextField();
    textField_Name.setBounds(233, 18, 117, 20);
    contentPane.add(textField_Name);
    textField_Name.setColumns(10);

    JLabel lblNewLabel_Position = new JLabel("Position:");
    lblNewLabel_Position.setFont(new Font("TIMES NEW ROMAN", Font.BOLD, 14));
    lblNewLabel_Position.setBounds(145, 49, 87, 24);
    contentPane.add(lblNewLabel_Position);

    textField_Position = new JTextField();
    textField_Position.setBounds(233, 53, 117, 20);
    contentPane.add(textField_Position);
    textField_Position.setColumns(10);

    JLabel lblNewLabel_Email = new JLabel("Email:");
    lblNewLabel_Email.setFont(new Font("TIMES NEW ROMAN", Font.BOLD, 14));
    lblNewLabel_Email.setBounds(145, 84, 87, 24);
    contentPane.add(lblNewLabel_Email);

    textField_Email = new JTextField();
    textField_Email.setBounds(233, 88, 200, 20);
    contentPane.add(textField_Email);

    JLabel lblNewLabel_PhoneNumber = new JLabel("Phone Number:");
    lblNewLabel_PhoneNumber.setFont(new Font("TIMES NEW ROMAN", Font.BOLD, 14));
    lblNewLabel_PhoneNumber.setBounds(85, 119, 136, 24);
    contentPane.add(lblNewLabel_PhoneNumber);

    textField_PhoneNumber = new JTextField();
    textField_PhoneNumber.setBounds(233, 123, 117, 20);
    contentPane.add(textField_PhoneNumber);
    textField_PhoneNumber.setColumns(10);

    JLabel lblNewLabel_Address = new JLabel("Address:");
    lblNewLabel_Address.setFont(new Font("TIMES NEW ROMAN", Font.BOLD, 14));
    lblNewLabel_Address.setBounds(145, 154, 87, 24);
    contentPane.add(lblNewLabel_Address);

    textField_Address = new JTextField();
    textField_Address.setBounds(233, 158, 200, 20);
    contentPane.add(textField_Address);
    textField_Address.setColumns(10);

    // Save Button
    JButton btnSave = new JButton("Save");
    btnSave.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            saveProfile();
            // Add code to display success message
            JLabel successLabel = new JLabel("Profile saved successfully!");
            successLabel.setFont(new Font("TIMES NEW ROMAN", Font.BOLD, 14));
            successLabel.setBounds(180, 250, 200, 30);
            contentPane.add(successLabel);
            contentPane.revalidate();
            contentPane.repaint();
        }
    });
    btnSave.setBounds(180, 200, 100, 30);
    contentPane.add(btnSave);
}
```

The code supplied uses inheritance, where the `JFrame} class is extended by the `Manage_Personal_Profile_Page` class. {Manage_Personal_Profile_Page} may become a graphical window by inheriting all of `JFrame}'s methods and properties thanks to this inheritance. Several Swing components, such as JLabel, JTextField, and JButton, are added to the window's content pane inside the Manage_Personal_Profile_Page constructor. These components inherit the JFrame methods for component placement and addition. Furthermore, event handling is implemented, whereby the `JButton} class, which is derived from `JFrame}, provides an action listener to control the click event of the "Save" button.

7.3 HOUSHIANEE HARHANGI

7.3.0 INHERITANCE

```
public Administrator_Account(String username) {
    initComponents();
    user = username;
}
```

The inherited methods of JFrame are responsible for displaying, sizing, and closing windows. This is a prerequisite for all GUI software.

It is inherited from JFrame the capability to add GUI components such as icons, panels, and labels. Thus, modular user interface construction is possible through the addition of components to the frame in order to form a unified structure.
By specifying a username in the constructor, a logical user-centric interface is demonstrated. This design decision demonstrates an enhanced comprehension of the usability principles of object-oriented programming (OOP) software. By customizing the user experience, the username exemplifies how inheritance can significantly enhance functionality.

7.3.1 ENCAPSULATION

```
private String user;
```

Access to private user variables is restricted to the Administrator_Account class. Encapsulation serves as a safeguard against unauthorized access to user data. Possibly additional methods (not fully illustrated in the provided excerpts), including getters, setters, and business logic methods, as well as the class's constructor, regulate the access and modification of this data.

Encapsulating data in this manner regulates the administration of application states, thereby enhancing data integrity and security.

7.3.2 ABSTRACTION

private void initComponents() {

...

}

The initComponents function simplifies the configuration and initialization of GUI components. This method exemplifies the concept of abstraction by concealing from the application logic the complex GUI setup elements, such as generating components, designating their properties, and adding them to the frame. Users of the Administrator_Account class are not required to comprehend GUI assembly or component initialization in order to utilize the class.

7.3.3 POLYMORPHISM

private void initComponents() {

...

}

The Java.awt file.The polymorphic management of component subclasses, such as JLabel and JButton, is possible with Swing container classes.

Polymorphism permits adaptable administration of GUI components. Enhancing scalability and adaptability, it integrates the interactions of the Administrator_GenerateReport class with various component types. Although they may vary, the process of incorporating various jPanel1 components remains consistent.

For report generation, the initComponents method facilitates the process of initializing and configuring GUI components.

8.0 IMPLEMENTATION

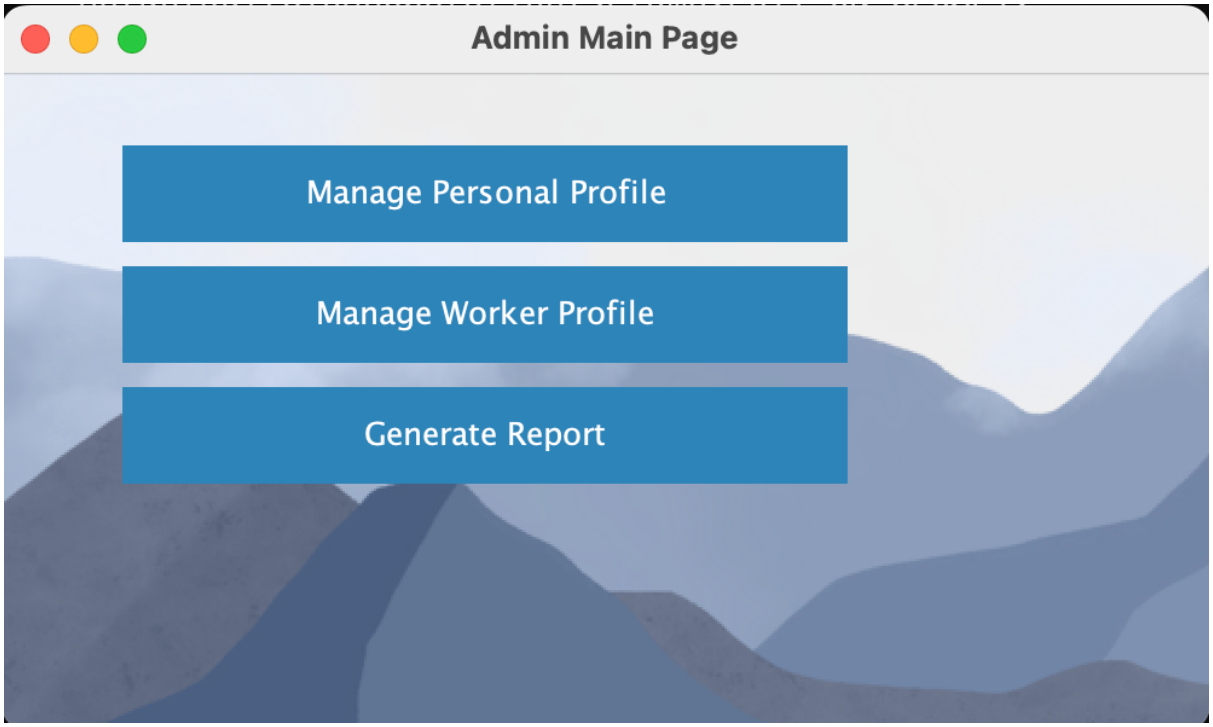
8.1 ADMINISTRATOR- ABDUL MUHAIMIN AMAN

CODE:

```
        generateReportButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // Open the SalesReportMain frame
                SwingUtilities.invokeLater(() -> {
                    new SalesReportMain().setVisible(true);
                });
            }
        });

        // Set layout manager to null for the content pane
        setLayout(null);

        // Set bounds for the buttons and background image
        managePersonalProfileButton.setBounds(50, 30, 300, 40);
        manageWorkerProfileButton.setBounds(50, 80, 300, 40);
        generateReportButton.setBounds(50, 130, 300, 40);
        backgroundImageLabel.setBounds(0, 0, getWidth(), getHeight());

        // Add buttons to the content pane
        add(managePersonalProfileButton);
        add(manageWorkerProfileButton);
        add(generateReportButton);
    }

    private JButton createStyledButton(String text) {
        JButton button = new JButton(text);
        button.setBackground(new Color(70, 130, 180)); // Light blue
        button.setForeground(Color.WHITE);
        button.setFocusPainted(false);
        button.setBorderPainted(false);
        button.setOpaque(true);
        return button;
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            MainAdminFrame mainAdminFrame = new MainAdminFrame();
            mainAdminFrame.setVisible(true);
        });
    }
}
```

```
package oodjaman;

import javax.swing.*;

public class MainAdminFrame extends JFrame {

    private oodjaman2 adminProfileFrame;
    private WorkerProfileGUI salespersonProfileFrame;
    private WorkerProfileGUI officerProfileFrame;

    public MainAdminFrame() {
        setTitle("Admin Main Page");
        setSize(500, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);

        // Create a JLabel to hold the background image
        JLabel backgroundImageLabel = new JLabel();
        setContentPane(backgroundImageLabel);

        // Load the background image
        ImageIcon backgroundImage = new ImageIcon("/Users/aman/eclipse-workspace/oodjaman/src/oodjaman/pngtree-big-mountain-bo
        backgroundImageLabel.setIcon(backgroundImage);

        JButton managePersonalProfileButton = createStyledButton("Manage Personal Profile");
        JButton manageWorkerProfileButton = createStyledButton("Manage Worker Profile");
        JButton generateReportButton = createStyledButton("Generate Report");

        managePersonalProfileButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // Open the Administrator Profile frame
                SwingUtilities.invokeLater(() -> {
                    if (adminProfileFrame == null) {
                        adminProfileFrame = new oodjaman2();
                    }
                    adminProfileFrame.setVisible(true);
                });
            }
        });

        manageWorkerProfileButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // Prompt user to choose worker type
                UserTypeSelectionFrame userTypeSelectionFrame = new UserTypeSelectionFrame();
                userTypeSelectionFrame.setVisible(true);
            }
        });

        generateReportButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
```

GUI:



DESCRIPTION:

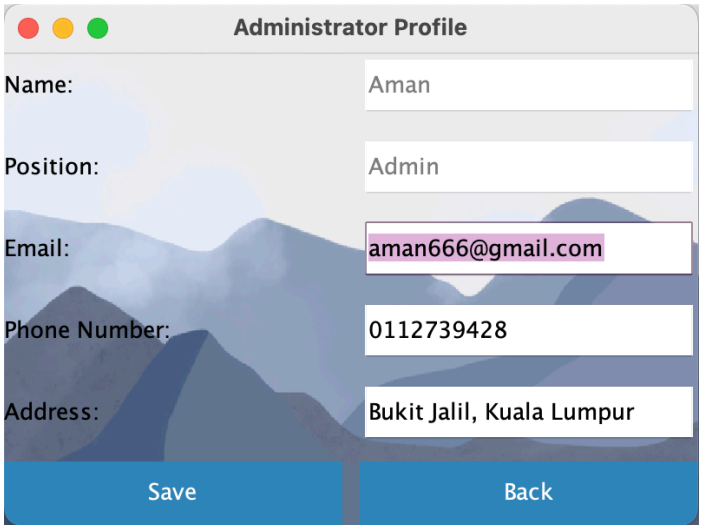
A Java Swing JFrame that expands the capabilities of (GUI) is the MainAdminFrame class. The application's entry point is the public static void main function. The frame's position, size, closure operation, and title are all set inside the constructor. It uses an ImageIcon to load the background picture and configures styled buttons with action listeners. The content pane is set using the setContentPane function, and layout management is turned off by using setLayout(null), which permits component placement by hand. Lastly, the MainAdminFrame object is created and made visible by the main function, which then starts the GUI.

CODE:

```
package oodjaman;

import javax.imageio.ImageIO;

public class oodjaman2 extends JFrame {

    private AdministratorProfile administratorProfile;

    private JTextField emailField;
    private JTextField phoneNumberField;
    private JTextField addressField;
    private JButton saveButton;
    private JButton goBackButton;

    public oodjaman2() {
        administratorProfile = AdministratorProfileManager.readProfile();

        setTitle("Administrator Profile");
        setSize(400, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        createUI();
    }

    private void createUI() {
        JPanel mainPanel = new BackgroundPanel("/Users/aman/eclipse-workspace/oodjaman/src/oodjaman/pngtree-big-mountain-bord
        mainPanel.setLayout(new GridLayout(6, 2, 10, 10));

        mainPanel.add(new JLabel("Name:"));
        mainPanel.add(new JTextField(administratorProfile.getName())).setEnabled(false);
        mainPanel.add(new JLabel("Position:"));
        mainPanel.add(new JTextField(administratorProfile.getPosition())).setEnabled(false);
        mainPanel.add(new JLabel("Email:"));
        emailField = new JTextField(administratorProfile.getEmail());
        mainPanel.add(emailField);
        mainPanel.add(new JLabel("Phone Number:"));
        phoneNumberField = new JTextField(administratorProfile.getPhoneNumber());
        mainPanel.add(phoneNumberField);
        mainPanel.add(new JLabel("Address:"));
        addressField = new JTextField(administratorProfile.getAddress());
        mainPanel.add(addressField);

        saveButton = createStyledButton("Save");
        saveButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                saveChanges();
            }
        });


        // Create "Go Back" button
        goBackButton = createStyledButton("Back");
        goBackButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                dispose(); // Close the current frame
            }
        });

        mainPanel.add(saveButton);
        mainPanel.add(goBackButton);
        setLocationRelativeTo(null);

        getContentPane().add(mainPanel, BorderLayout.CENTER);
    }

    private void saveChanges() {
        administratorProfile.setEmail(emailField.getText());
        administratorProfile.setPhoneNumber(phoneNumberField.getText());
        administratorProfile.setAddress(addressField.getText());

        AdministratorProfileManager.writeProfile(administratorProfile);

        // Create a custom JDialog with a blue gradient background
        JDialog dialog = new JDialog(this, "Success", true);
        dialog.setContentPane(createBlueGradientBackground());
        dialog.setSize(300, 100);
        dialog.setLocationRelativeTo(this);

        // Add a label to the dialog
        JLabel label = new JLabel("Changes saved successfully.");
        label.setForeground(Color.WHITE);
        dialog.add(label, BorderLayout.CENTER);

        // Set a timer to close the dialog after 2 seconds (2000 milliseconds)
        Timer timer = new Timer(2000, new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                dialog.dispose();
            }
        });
        timer.setRepeats(false); // Only run the timer once
        timer.start();

        dialog.setVisible(true);
    }

    private JButton createStyledButton(String text) {
        JButton button = new JButton(text);
```

```
03
04      private JButton createStyledButton(String text) {
05          JButton button = new JButton(text);
06          button.setBackground(new Color(70, 130, 180)); // Light blue
07          button.setForeground(Color.WHITE);
08          button.setFocusPainted(false);
09          button.setBorderPainted(false);
10          button.setOpaque(true);
11          return button;
12      }
13
14      public static void main(String[] args) {
15          SwingUtilities.invokeLater(new Runnable() {
16              @Override
17              public void run() {
18                  new oodjaman2().setVisible(true);
19              }
20          });
21      }
22
23      // Custom JPanel with background image
24      private class BackgroundPanel extends JPanel {
25          private Image backgroundImage;
26
27          public BackgroundPanel(String imagePath) {
28              try {
29                  backgroundImage = ImageIO.read(new File(imagePath));
30              } catch (IOException e) {
31                  e.printStackTrace();
32              }
33          }
34
35          @Override
36          protected void paintComponent(Graphics g) {
37              super.paintComponent(g);
38              if (backgroundImage != null) {
39                  g.drawImage(backgroundImage, 0, 0, getWidth(), getHeight(), this);
40              }
41          }
42      }
43      private JPanel createBlueGradientBackground() {
44          JPanel panel = new JPanel() {
45              @Override
46              protected void paintComponent(Graphics g) {
47                  super.paintComponent(g);
48                  Graphics2D g2d = (Graphics2D) g;
49                  int w = getWidth();
50                  int h = getHeight();
51                  Color color1 = new Color(70, 130, 180); // Light blue
52                  Color color2 = new Color(30, 60, 90);   // Dark blue
```

GUI:



DESCRIPTION:

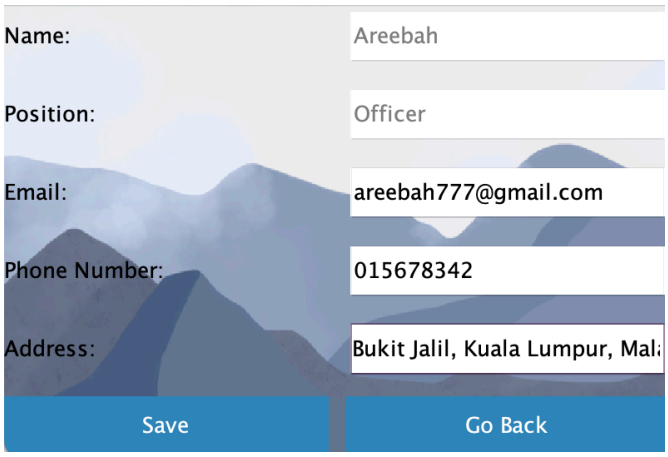
A Swing `JFrame} that manages the Administrator Profile is the `oodjaman2} class. It has areas where you may examine and edit details. The `saveChanges` method modifies the profile, opens a feedback box with a backdrop gradient of blue, and closes on its own after two seconds. The user interface (UI) makes use of Swing components, and the buttons ({saveButton} and `goBackButton{}) have been given a unique light blue style with white text to make them visually appealing.

CODE:

```java
package oodjaman;

import javax.imageio.ImageIO;

public class WorkerProfileGUI extends JFrame {

    private WorkerProfile workerProfile;
    private JButton goBackButton;

    private JTextField emailField;
    private JTextField phoneNumberField;
    private JTextField addressField;

    private String workerType; // "SALESPERSON" or "OFFICER"

    public WorkerProfileGUI(String workerType, String filePath) {
        this.workerType = workerType;
        workerProfile = WorkerProfileManager.readWorkerProfile(workerType, filePath);

        setTitle(workerType + " Profile");
        setSize(400, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
        createUI();
    }

    private void createUI() {
        JPanel mainPanel = new BackgroundPanel("/Users/aman/eclipse-workspace/oodjaman/src/oodjaman/pngtree-big-mountain-bord
        mainPanel.setLayout(new GridLayout(6, 2, 10, 10));

        mainPanel.add(new JLabel("Name:"));
        mainPanel.add(new JTextField(workerProfile.getName())).setEnabled(false);
        mainPanel.add(new JLabel("Position:"));
        mainPanel.add(new JTextField(workerProfile.getPosition())).setEnabled(false);
        mainPanel.add(new JLabel("Email:"));
        emailField = new JTextField(workerProfile.getEmail());
        mainPanel.add(emailField);
        mainPanel.add(new JLabel("Phone Number:"));
        phoneNumberField = new JTextField(workerProfile.getPhoneNumber());
        mainPanel.add(phoneNumberField);
        mainPanel.add(new JLabel("Address:"));
        addressField = new JTextField(workerProfile.getAddress());
        mainPanel.add(addressField);

        JButton saveButton = createStyledButton("Save");
        saveButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                saveChanges();
            }
        });

        // Create "Go Back" button
        goBackButton = createStyledButton("Go Back");
        goBackButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                dispose(); // Close the current frame
            }
        });

        mainPanel.add(saveButton);
        mainPanel.add(goBackButton);

        getContentPane().add(mainPanel, BorderLayout.CENTER);
    }

    private void saveChanges() {
        workerProfile.setEmail(emailField.getText());
        workerProfile.setPhoneNumber(phoneNumberField.getText());
        workerProfile.setAddress(addressField.getText());

        WorkerProfileManager.writeWorkerProfile(workerProfile, workerType);

        // Create a custom JDialog with a blue gradient background
        JDialog dialog = new JDialog(this, "Success", true);
        dialog.setContentPane(createBlueGradientBackground());
        dialog.setSize(300, 100);
        dialog.setLocationRelativeTo(this);

        // Add a label to the dialog
        JLabel label = new JLabel("Changes saved successfully.");
        label.setForeground(Color.WHITE);
        dialog.add(label, BorderLayout.CENTER);

        // Set a timer to close the dialog after 2 seconds (2000 milliseconds)
        Timer timer = new Timer(2000, new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                dialog.dispose();
            }
        });
        timer.setRepeats(false); // Only run the timer once
        timer.start();

        dialog.setVisible(true);
    }


    private JButton createStyledButton(String text) {
        JButton button = new JButton(text);
        button.setBackground(new Color(70, 130, 180)); // Light blue
        button.setForeground(Color.WHITE);
```

```
106    private JButton createStyledButton(String text) {
107        JButton button = new JButton(text);
108        button.setBackground(new Color(70, 130, 180)); // Light blue
109        button.setForeground(Color.WHITE);
110        button.setFocusPainted(false);
111        button.setBorderPainted(false);
112        button.setOpaque(true);
113        return button;
114    }
115
116    public static void main(String[] args) {
117        SwingUtilities.invokeLater(new Runnable() {
118            @Override
119            public void run() {
120                // Open UserTypeSelectionFrame to choose worker type
121                UserTypeSelectionFrame userTypeSelectionFrame = new UserTypeSelectionFrame();
122                userTypeSelectionFrame.setVisible(true);
123            }
124        });
125    }
126
127
128
129
130    // Custom JPanel with background image
131    private class BackgroundPanel extends JPanel {
132        private Image backgroundImage;
133
134        public BackgroundPanel(String imagePath) {
135            try {
136                backgroundImage = ImageIO.read(new File(imagePath));
137            } catch (IOException e) {
138                e.printStackTrace();
139            }
140        }
141
142        @Override
143        protected void paintComponent(Graphics g) {
144            super.paintComponent(g);
145            if (backgroundImage != null) {
146                g.drawImage(backgroundImage, 0, 0, getWidth(), getHeight(), this);
147            }
148        }
149    }
150
151    // Create a blue gradient background for JOptionPane
152    private JPanel createBlueGradientBackground() {
153        JPanel panel = new JPanel() {
154            @Override
155            protected void paintComponent(Graphics g) {
156                super.paintComponent(g);
```

GUI:



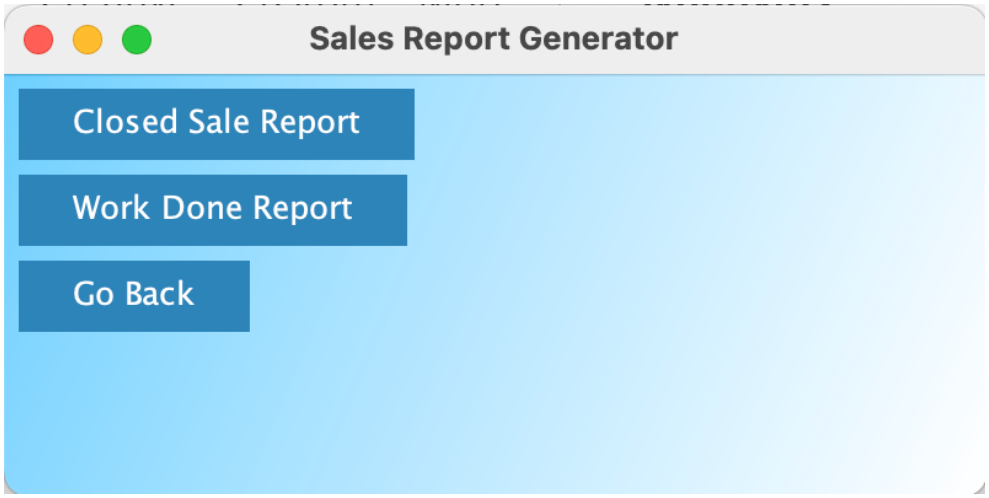
DESCRIPTION:

The WorkerProfileGUI class uses Swing to provide a graphical user interface for controlling worker profiles. It extends JFrame, uses numerous Swing components (such as JPanel and JButton), and handles events via ActionListener. It has layout control, file I/O for worker profiles, and a customizable BackgroundPanel for background pictures. The GUI allows users to change and save worker information while receiving visible feedback via a success dialog.

CODE:

```
package oodjaman;

import javax.swing.*;

public class SalesReportMain extends JFrame {

    private JButton goBackButton;

    public SalesReportMain() {
        setTitle("Sales Report Generator");
        setSize(400, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);

        // Set up gradient background
        setContentPane(new JPanel() {
            @Override
            protected void paintComponent(Graphics g) {
                Graphics2D g2d = (Graphics2D) g;
                Color color1 = new Color(135, 206, 250); // Light blue
                Color color2 = new Color(255, 255, 255); // White
                GradientPaint gp = new GradientPaint(0, 0, color1, getWidth(), getHeight(), color2);
                g2d.setPaint(gp);
                g2d.fillRect(0, 0, getWidth(), getHeight());
            }
        });

        JButton closedSaleReportButton = createStyledButton("Closed Sale Report");
        JButton workDoneReportButton = createStyledButton("Work Done Report");
        goBackButton = createStyledButton("Go Back");

        closedSaleReportButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // Open Closed Sale Report frame
                SwingUtilities.invokeLater(() -> {
                    new ClosedSaleReport("/Users/aman/Downloads/zip (1)/SalesQuotation.txt").setVisible(true);
                    dispose(); // Close the main frame
                });
            }
        });

        workDoneReportButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // Open Work Done Report frame
                SwingUtilities.invokeLater(() -> {
                    new WorkDoneReport("/Users/aman/Downloads/zip (1)/SalesQuotation.txt").setVisible(true);
                    dispose(); // Close the main frame
                });
            }
        });


        goBackButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                SwingUtilities.invokeLater(() -> new MainAdminFrame().setVisible(true));
                dispose(); // Close the current frame
            }
        });

        // Layout setup

        GroupLayout layout = new GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setAutoCreateGaps(true);
        layout.setAutoCreateContainerGaps(true);

        layout.setHorizontalGroup(layout.createSequentialGroup()
                .addGroup(layout.createParallelGroup(GroupLayout.Alignment.LEADING)
                        .addComponent(closedSaleReportButton)
                        .addComponent(workDoneReportButton)
                        .addComponent(goBackButton))
        );

        layout.setVerticalGroup(layout.createSequentialGroup()
                .addComponent(closedSaleReportButton)
                .addComponent(workDoneReportButton)
                .addComponent(goBackButton)
        );
    }

    private JButton createStyledButton(String text) {
        JButton button = new JButton(text);
        button.setBackground(new Color(70, 130, 180)); // Light blue
        button.setForeground(Color.WHITE);
        button.setFocusPainted(false);
        button.setBorderPainted(false);
        button.setOpaque(true);
        return button;
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> new SalesReportMain().setVisible(true));
    }
}
```

GUI:



DESCRIPTION:

The 'SalesReportMain' class manages its layout using GroupLayout, and the 'createStyledButton' function is used to make styled buttons. The buttons ("Closed Sale Report," "Work Done Report," and "Go Back") have matching listeners, which use SwingUtilities for thread safety. When clicked, the "Closed Sale Report" and "Work Done Report" buttons open other frames ('ClosedSaleReport' and 'WorkDoneReport', respectively), while the "Go Back" button returns to the 'MainAdminFrame'. The design attempts to provide a visually appealing and responsive sales report generation interface.

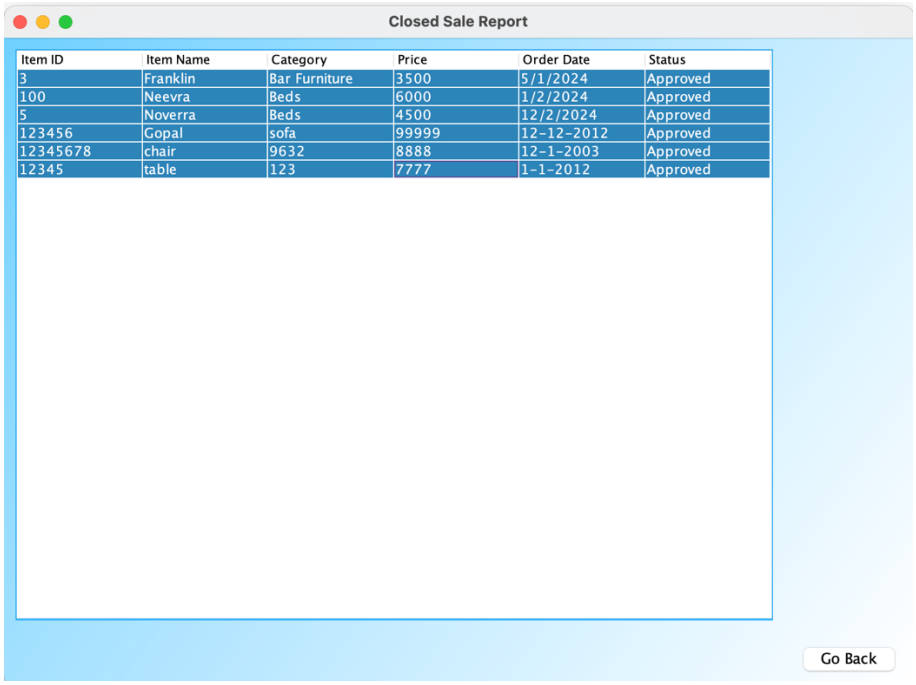
CODE:

```
package oodjaman;

import javax.swing.*;

public class ClosedSaleReport extends JFrame {

    private JButton goBackButton;

    public ClosedSaleReport(String filePath) {
        setTitle("Closed Sale Report");
        setSize(800, 600);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


        // Set up gradient background
        setContentPane(new JPanel() {
            @Override
            protected void paintComponent(Graphics g) {
                Graphics2D g2d = (Graphics2D) g;
                Color color1 = new Color(135, 206, 250); // Light orange
                Color color2 = new Color(255, 255, 255); // Coral
                GradientPaint gp = new GradientPaint(0, 0, color1, getWidth(), getHeight(), color2);
                g2d.setPaint(gp);
                g2d.fillRect(0, 0, getWidth(), getHeight());
            }
        });

        List<String[]> approvedSales = getApprovedSales(filePath);

        // Create table model
        String[] columnNames = {"Item ID", "Item Name", "Category", "Price", "Order Date", "Status"};
        DefaultTableModel tableModel = new DefaultTableModel(columnNames, 0) {
            @Override
            public Class<?> getColumnClass(int column) {
                return getValueAt(0, column).getClass();
            }
        };

        // Add approved sales to table model
        for (String[] sale : approvedSales) {
            tableModel.addRow(sale);
        }

        JTable reportTable = new JTable(tableModel) {
            @Override
            public Component prepareRenderer(TableCellRenderer renderer, int row, int column) {
                Component comp = super.prepareRenderer(renderer, row, column);

                // Handle selection state
                if (isRowSelected(row)) {
                    comp.setBackground(new Color(70, 130, 180)); // Light blue for selected row
```

```
        reportTable.setFillsViewportHeight(false);
        JScrollPane scrollPane = new JScrollPane(reportTable);

        // Create "Go Back" button
        goBackButton = new JButton("Go Back");
        goBackButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                dispose(); // Close the current frame
                SwingUtilities.invokeLater(() -> new SalesReportMain().setVisible(true));
            }
        });

        // Use GridBagLayout to center the table and add the "Go Back" button
        getContentPane().setLayout(new GridBagLayout());
        GridBagConstraints constraints = new GridBagConstraints();
        constraints.gridx = 0;
        constraints.gridy = 0;
        constraints.weightx = 1.0;
        constraints.weighty = 1.0;
        constraints.fill = GridBagConstraints.BOTH;
        constraints.insets = new Insets(10, 10, 10, 10);
        getContentPane().add(scrollPane, constraints);

        constraints.gridx = 1;
        constraints.gridy = 1;
        constraints.weightx = 0;
        constraints.weighty = 0;
        constraints.fill = GridBagConstraints.NONE;
        constraints.anchor = GridBagConstraints.PAGE_END;
        constraints.insets = new Insets(10, 10, 10, 10);
        getContentPane().add(goBackButton, constraints);

        setLocationRelativeTo(null); // Center the frame on the screen
        setVisible(true);
    }

    private List<String[]> getApprovedSales(String filePath) {
        List<String[]> approvedSales = new ArrayList<>();

        try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {
            String line;

            while ((line = reader.readLine()) != null) {
                String[] parts = line.split("\t"); // Assuming tab-separated values

                if (parts.length >= 6) {
                    String status = parts[5].trim().toLowerCase();

                    // Check if the sale is approved
```

```
    private List<String[]> getApprovedSales(String filePath) {
        List<String[]> approvedSales = new ArrayList<>();

        try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {
            String line;

            while ((line = reader.readLine()) != null) {
                String[] parts = line.split("\t"); // Assuming tab-separated values

                if (parts.length >= 6) {
                    String status = parts[5].trim().toLowerCase();

                    // Check if the sale is approved
                    if (status.equals("approved")) {
                        approvedSales.add(parts);
                    }
                }
            }
        } catch (IOException e) {
            e.printStackTrace(); // Handle the exception appropriately
        }

        return approvedSales;
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            new ClosedSaleReport("/Users/aman/Downloads/zip (1)/SalesQuotation.txt");
        });
    }
}
```

GUI:



DESCRIPTION:

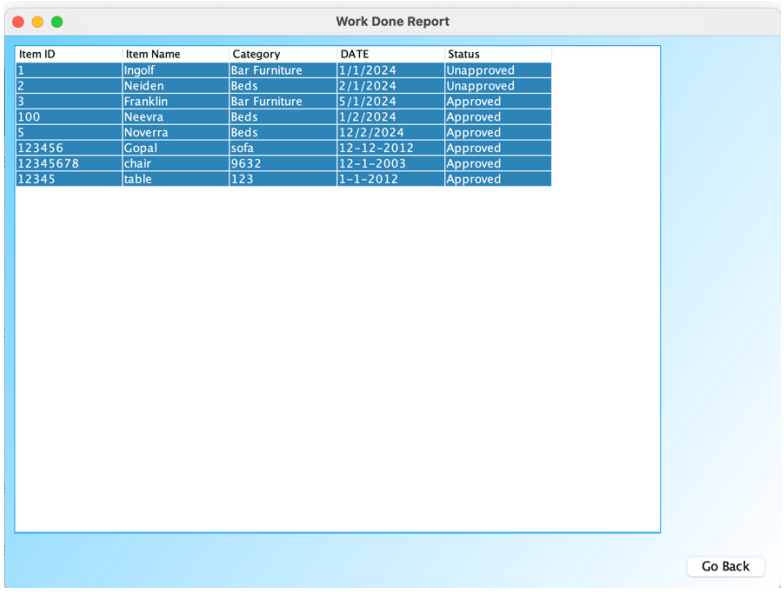
The ClosedSaleReport class uses Swing for its GUI, which includes a JTable displaying authorized sales data and a gradient backdrop. By using SwingUtilities.By managing GUI activities on the Event Dispatch Thread (EDT), invokeLater in the main function guarantees thread safety. The class changes the JTable dynamically after reading sales data from a file and removing authorized sales. The "Go Back" button returns to the sales report's main frame with ease. The program uses a GridBagLayout to keep its layout tidy and aesthetically pleasing. In order to mitigate any concurrent concerns, it is imperative that invokeLater be used to ensure that Swing components are generated and changed on the EDT.

CODE:

```java
package oodjaman;

import javax.swing.*;

public class WorkDoneReport extends JFrame {

    private JButton goBackButton;

    public WorkDoneReport(String filePath) {
        setTitle("Work Done Report");
        setSize(800, 600);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Set up gradient background
        setContentPane(new JPanel() {
            @Override
            protected void paintComponent(Graphics g) {
                Graphics2D g2d = (Graphics2D) g;
                Color color1 = new Color(135, 206, 250); // Light blue
                Color color2 = new Color(255, 255, 255); // White
                GradientPaint gp = new GradientPaint(0, 0, color1, getWidth(), getHeight(), color2);
                g2d.setPaint(gp);
                g2d.fillRect(0, 0, getWidth(), getHeight());
            }
        });

        List<String[]> salesData = getSalesData(filePath);

        // Create table model
        String[] columnNames = {"Item ID", "Item Name", "Category", "Price", "DATE", "Status"};
        DefaultTableModel tableModel = new DefaultTableModel(columnNames, 0) {
            @Override
            public Class<?> getColumnClass(int column) {
                return getValueAt(0, column).getClass();
            }
        };

        // Add sales data to table model
        for (String[] sale : salesData) {
            tableModel.addRow(sale);
        }

        JTable reportTable = new JTable(tableModel) {
            @Override
            public Component prepareRenderer(TableCellRenderer renderer, int row, int column) {
                Component comp = super.prepareRenderer(renderer, row, column);

                // Handle selection state
                if (isRowSelected(row)) {
                    comp.setBackground(new Color(70, 130, 180)); // Light blue for selected row
                } else {

        reportTable.setFillsViewportHeight(false);
        JScrollPane scrollPane = new JScrollPane(reportTable);

        // Create "Go Back" button
        goBackButton = new JButton("Go Back");
        goBackButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                dispose(); // Close the current frame
                SwingUtilities.invokeLater(() -> new SalesReportMain().setVisible(true));
            }
        });

        // Use GridBagLayout to center the table and add the "Go Back" button
        getContentPane().setLayout(new GridBagLayout());
        GridBagConstraints constraints = new GridBagConstraints();
        constraints.gridx = 0;
        constraints.gridy = 0;
        constraints.weightx = 1.0;
        constraints.weighty = 1.0;
        constraints.fill = GridBagConstraints.BOTH;
        constraints.insets = new Insets(10, 10, 10, 10);
        getContentPane().add(scrollPane, constraints);

        constraints.gridx = 1;
        constraints.gridy = 1;
        constraints.weightx = 0;
        constraints.weighty = 0;
        constraints.fill = GridBagConstraints.NONE;
        constraints.anchor = GridBagConstraints.PAGE_END;
        constraints.insets = new Insets(10, 10, 10, 10);
        getContentPane().add(goBackButton, constraints);

        setLocationRelativeTo(null); // Center the frame on the screen
        setVisible(true);
    }

    private List<String[]> getSalesData(String filePath) {
        List<String[]> salesData = new ArrayList<>();

        try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {
            String line;
            boolean isFirstLine = true;

            while ((line = reader.readLine()) != null) {
```

```
        setVisible(true);
    }

    private List<String[]> getSalesData(String filePath) {
        List<String[]> salesData = new ArrayList<>();

        try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {
            String line;
            boolean isFirstLine = true;

            while ((line = reader.readLine()) != null) {
                if (isFirstLine) {
                    isFirstLine = false;
                    continue; // Skip the first line (column names)
                }

                String[] parts = line.split("\t"); // Assuming tab-separated values

                if (parts.length >= 6) {
                    // Include every column
                    String[] saleDetails = Arrays.copyOfRange(parts, 0, 6);
                    salesData.add(saleDetails);
                }
            }
        } catch (IOException e) {
            e.printStackTrace(); // Handle the exception appropriately
        }

        return salesData;
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            new WorkDoneReport("/Users/aman/Downloads/zip (1)/SalesQuotation.txt");
        });
    }
}
```

GUI:



DESCRIPTION:

Swing is used in the WorkDoneReport class to provide a modern GUI with a gradient backdrop and a dynamic JTable for sales data. Using SwingUtilities.Thread safety is ensured with invokeLater. The "Go Back" button seamlessly returns to the main sales report, and the parsing algorithm filters and shows sales information. A balanced visual layout is orchestrated by the GridBagLayout.

8.2 SALESPERSON - KERTANAA A/P KUMAAR

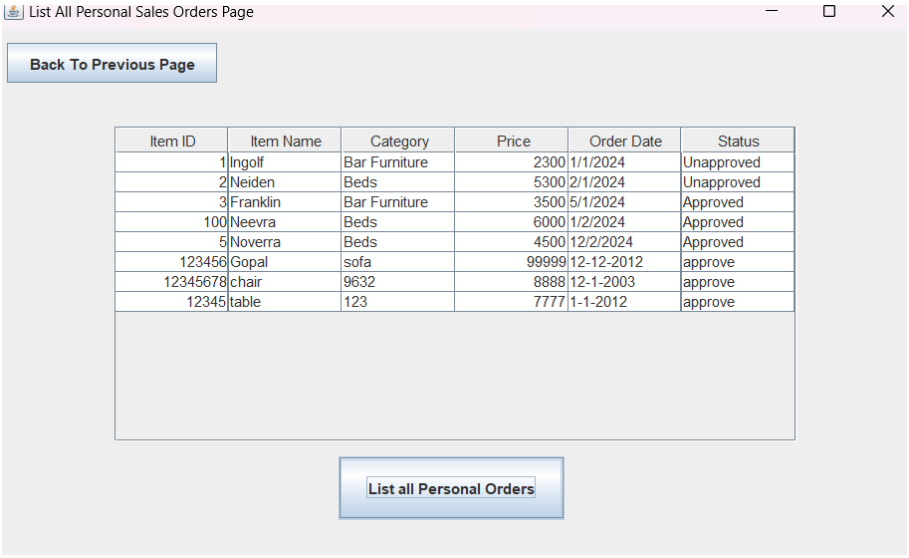
CODE:

```
package group_Assignment_Java;

import java.awt.EventQueue;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JTable;
import javax.swing.JButton;
import javax.swing.table.DefaultTableModel;
import javax.swing.JScrollPane;
import java.awt.event.ActionListener;
import java.io.BufferedReader;
import java.io.FileReader;
import java.awt.event.ActionEvent;

public class List_All_Personal_Sales_Orders_Page extends JFrame {

    private JPanel contentPane;
    private JTable table;
    private static final String sales_quotation_txt_file ="C:\\Users\\kerta\\OneDrive\\Documents\\OODJ\\SalesQuotation.txt";
    private JButton btn_BacktoPreviousPage;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    List_All_Personal_Sales_Orders_Page frame = new List_All_Personal_Sales_Orders_Page();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the frame.
     */
    public List_All_Personal_Sales_Orders_Page() {
```

```
    public List_All_Personal_Sales_Orders_Page() {
        setTitle("List All Personal Sales Orders Page");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 756, 485);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));

        setContentPane(contentPane);
        contentPane.setLayout(null);

        JScrollPane scrollPane = new JScrollPane();
        scrollPane.setBounds(96, 78, 549, 253);
        contentPane.add(scrollPane);

        table = new JTable();
        scrollPane.setViewportView(table);
        table.setModel(new DefaultTableModel(
            new Object[][] {
            },
            new String[] {
                "Item ID", "Item Name", "Category", "Price", "Order Date", "Status"
            }
        ) {
            Class[] columnTypes = new Class[] {
                Integer.class, String.class, String.class, Integer.class, String.class, String.class
            };
            public Class getColumnClass(int columnIndex) {
                return columnTypes[columnIndex];
            }
        });

        JButton btnNewButton = new JButton("List all Personal Orders");
        btnNewButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                 updateTableFromFile();
            }
        });
        btnNewButton.setBounds(276, 343, 182, 51);
        contentPane.add(btnNewButton);

        btn_BacktoPreviousPage = new JButton("Back To Previous Page");
        btn_BacktoPreviousPage.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
```

```
                // Use tab as the delimiter
                String delimiter = "\t";

                // Split the line into columns using the delimiter
                String[] columns = line.split(delimiter);

                // Add a new row to the table model
                model.addRow(columns);
            }
        } catch (Exception ex) {
            ex.printStackTrace();
            // Handle the exception (e.g., show an error message)
        }
    }


}
```

GUI:



Item ID	Item Name	Category	Price	Order Date	Status
1	Ingolf	Bar Furniture	2300	1/1/2024	Unapproved
2	Neiden	Beds	5300	2/1/2024	Unapproved
3	Franklin	Bar Furniture	3500	5/1/2024	Approved
100	Neevra	Beds	6000	1/2/2024	Approved
5	Noverra	Beds	4500	12/2/2024	Approved
123456	Gopal	sofa	99999	12-12-2012	approve
12345678	chair	9632	8888	12-1-2003	approve
12345	table	123	7777	1-1-2012	approve

The figure above shows the listing of all the orders. It contains of Item ID, Item Name, Category, Price, Order Date and Status.

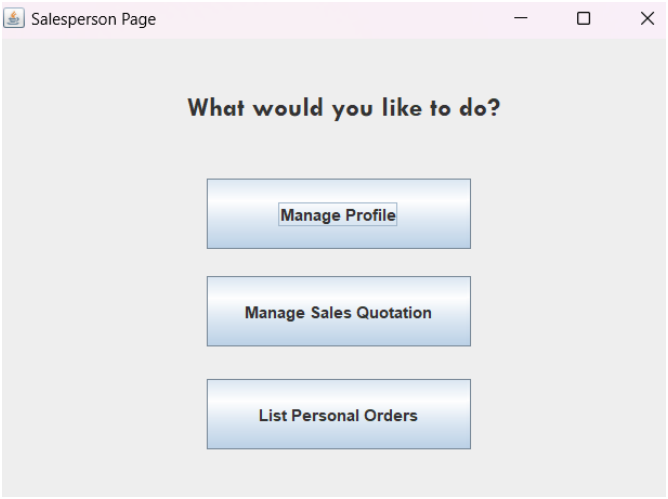
CODE:

```
package group_Assignment_Java;

import java.awt.EventQueue;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JLabel;
import javax.swing.JButton;
import java.awt.Font;
import javax.swing.SwingConstants;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class Salesperson_Page extends JFrame {

    /**
     *
     */
    private static final long serialVersionUID = 1L;
    private JPanel contentPane;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Salesperson_Page frame = new Salesperson_Page();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the frame.
     */
    public Salesperson_Page() {
        setTitle("Salesperson Page");
```

```
public Salesperson_Page() {
    setTitle("Salesperson Page");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 522, 392);
    setLocationRelativeTo(null);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));

    setContentPane(contentPane);
    contentPane.setLayout(null);

    JLabel lblNewLabel = new JLabel("What would you like to do?");
    lblNewLabel.setHorizontalAlignment(SwingConstants.CENTER);
    lblNewLabel.setFont(new Font("Tw Cen MT", Font.BOLD, 20));
    lblNewLabel.setBounds(48, 35, 420, 30);
    contentPane.add(lblNewLabel);

    JButton btn_Manage_Personal_Profile = new JButton("Manage Profile");
    btn_Manage_Personal_Profile.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            Manage_Personal_Profile_Page profilePage = new Manage_Personal_Profile_Page();
            profilePage.setVisible(true);
        }
    });
    btn_Manage_Personal_Profile.setBounds(158, 104, 195, 52);
    contentPane.add(btn_Manage_Personal_Profile);

    JButton btn_Manage_Sales_Quotation = new JButton("Manage Sales Quotation");
    btn_Manage_Sales_Quotation.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            Manage_Sales_Order_Quotation_Page salesQuotationPage = new Manage_Sales_Order_Quotation_Page();
            salesQuotationPage.setVisible(true);
        }
    });
    btn_Manage_Sales_Quotation.setBounds(158, 176, 195, 52);
    contentPane.add(btn_Manage_Sales_Quotation);

    JButton btn_List_All_Personal_Orders = new JButton("List Personal Orders");
    btn_List_All_Personal_Orders.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            List_All_Personal_Sales_Orders_Page personalSalesOrdersPage = new List_All_Personal_Sales_Orders_Page();
            personalSalesOrdersPage.setVisible(true);
```

```
        });
        btn_Manage_Sales_Quotation.setBounds(158, 176, 195, 52);
        contentPane.add(btn_Manage_Sales_Quotation);

        JButton btn_List_All_Personal_Orders = new JButton("List Personal Orders");
        btn_List_All_Personal_Orders.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                List_All_Personal_Sales_Orders_Page personalSalesOrdersPage = new List_All_Personal_Sales_Orders_Page();
                personalSalesOrdersPage.setVisible(true);
            }
        });
        btn_List_All_Personal_Orders.setBounds(158, 252, 195, 52);
        contentPane.add(btn_List_All_Personal_Orders);
    }
}
```

GUI:



The figure above shows the main page of a salesperson. The page contains of Manage Profile, Manage Sales Quotation and Listings of Personal Orders.

CODE:

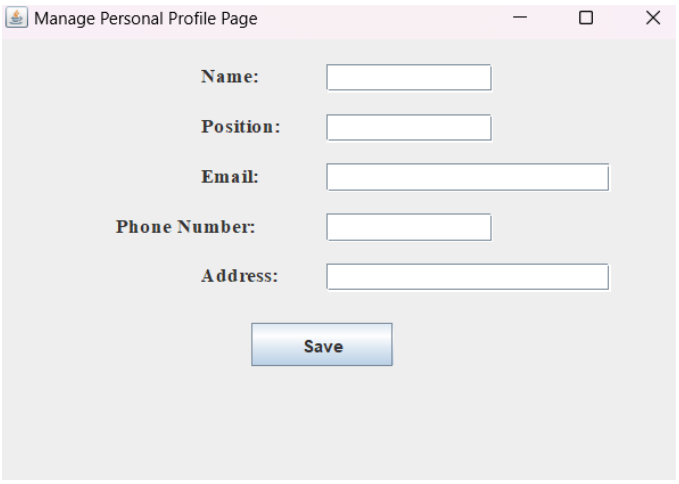
```
import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JLabel;
import java.awt.Font;
import javax.swing.JTextField;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

public class Manage_Personal_Profile_Page extends JFrame {

    private static final long serialVersionUID = 1L;
    private JPanel contentPane;
    private JTextField textField_Name;
    private JTextField textField_Position;
    private JTextField textField_Email;
    private JTextField textField_PhoneNumber;
    private JTextField textField_Address;

    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Manage_Personal_Profile_Page frame = new Manage_Personal_Profile_Page();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    public Manage_Personal_Profile_Page() {
        setTitle("Manage Personal Profile Page");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 501, 393);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
```

```
        setContentPane(contentPane);
        contentPane.setLayout(null);

        JLabel lblNewLabel_Name = new JLabel("Name:");
        lblNewLabel_Name.setFont(new Font("TIMES NEW ROMAN", Font.BOLD, 14));
        lblNewLabel_Name.setBounds(145, 14, 87, 24);
        contentPane.add(lblNewLabel_Name);

        textField_Name = new JTextField();
        textField_Name.setBounds(233, 18, 117, 20);
        contentPane.add(textField_Name);
        textField_Name.setColumns(10);

        JLabel lblNewLabel_Position = new JLabel("Position:");
        lblNewLabel_Position.setFont(new Font("TIMES NEW ROMAN", Font.BOLD, 14));
        lblNewLabel_Position.setBounds(145, 49, 87, 24);
        contentPane.add(lblNewLabel_Position);

        textField_Position = new JTextField();
        textField_Position.setBounds(233, 53, 117, 20);
        contentPane.add(textField_Position);
        textField_Position.setColumns(10);

        JLabel lblNewLabel_Email = new JLabel("Email:");
        lblNewLabel_Email.setFont(new Font("TIMES NEW ROMAN", Font.BOLD, 14));
        lblNewLabel_Email.setBounds(145, 84, 87, 24);
        contentPane.add(lblNewLabel_Email);

        textField_Email = new JTextField();
        textField_Email.setBounds(233, 88, 200, 20);
        contentPane.add(textField_Email);
        textField_Email.setColumns(10);

        JLabel lblNewLabel_PhoneNumber = new JLabel("Phone Number:");
        lblNewLabel_PhoneNumber.setFont(new Font("TIMES NEW ROMAN", Font.BOLD, 14));
        lblNewLabel_PhoneNumber.setBounds(85, 119, 136, 24);
        contentPane.add(lblNewLabel_PhoneNumber);

        textField_PhoneNumber = new JTextField();
        textField_PhoneNumber.setBounds(233, 123, 117, 20);
        contentPane.add(textField_PhoneNumber);
        textField_PhoneNumber.setColumns(10);
```

```
    setContentPane(contentPane);
    contentPane.setLayout(null);

    JLabel lblNewLabel_Name = new JLabel("Name:");
    lblNewLabel_Name.setFont(new Font("TIMES NEW ROMAN", Font.BOLD, 14));
    lblNewLabel_Name.setBounds(145, 14, 87, 24);
    contentPane.add(lblNewLabel_Name);

    textField_Name = new JTextField();
    textField_Name.setBounds(233, 18, 117, 20);
    contentPane.add(textField_Name);
    textField_Name.setColumns(10);

    JLabel lblNewLabel_Position = new JLabel("Position:");
    lblNewLabel_Position.setFont(new Font("TIMES NEW ROMAN", Font.BOLD, 14));
    lblNewLabel_Position.setBounds(145, 49, 87, 24);
    contentPane.add(lblNewLabel_Position);

    textField_Position = new JTextField();
    textField_Position.setBounds(233, 53, 117, 20);
    contentPane.add(textField_Position);
    textField_Position.setColumns(10);

    JLabel lblNewLabel_Email = new JLabel("Email:");
    lblNewLabel_Email.setFont(new Font("TIMES NEW ROMAN", Font.BOLD, 14));
    lblNewLabel_Email.setBounds(145, 84, 87, 24);
    contentPane.add(lblNewLabel_Email);

    textField_Email = new JTextField();
    textField_Email.setBounds(233, 88, 200, 20);
    contentPane.add(textField_Email);
    textField_Email.setColumns(10);

    JLabel lblNewLabel_PhoneNumber = new JLabel("Phone Number:");
    lblNewLabel_PhoneNumber.setFont(new Font("TIMES NEW ROMAN", Font.BOLD, 14));
    lblNewLabel_PhoneNumber.setBounds(85, 119, 136, 24);
    contentPane.add(lblNewLabel_PhoneNumber);

    textField_PhoneNumber = new JTextField();
    textField_PhoneNumber.setBounds(233, 123, 117, 20);
    contentPane.add(textField_PhoneNumber);
    textField_PhoneNumber.setColumns(10);

       JLabel lblNewLabel_Address = new JLabel("Address:");
       lblNewLabel_Address.setFont(new Font("TIMES NEW ROMAN", Font.BOLD, 14));
       lblNewLabel_Address.setBounds(145, 154, 87, 24);
       contentPane.add(lblNewLabel_Address);

       textField_Address = new JTextField();
       textField_Address.setBounds(233, 158, 200, 20);
       contentPane.add(textField_Address);
       textField_Address.setColumns(10);

       // Save Button
       JButton btnSave = new JButton("Save");
       btnSave.addActionListener(new ActionListener() {
           public void actionPerformed(ActionEvent e) {
               saveProfile();
               // Add code to display success message
               JLabel successLabel = new JLabel("Profile saved successfully!");
               successLabel.setFont(new Font("TIMES NEW ROMAN", Font.BOLD, 14));
               successLabel.setBounds(180, 250, 200, 30);
               contentPane.add(successLabel);
               contentPane.revalidate();
               contentPane.repaint();
           }
       });
       btnSave.setBounds(180, 200, 100, 30);
       contentPane.add(btnSave);
   }

   // Method to save profile data
   private void saveProfile() {
       String name = textField_Name.getText();
       String position = textField_Position.getText();
       String email = textField_Email.getText();
       String phoneNumber = textField_PhoneNumber.getText();
       String address = textField_Address.getText();

       // Write profile data to a text file
       try (BufferedWriter writer = new BufferedWriter(new FileWriter("profile.txt"))) {
           writer.write("Name: " + name);
           writer.newLine();
           writer.write("Position: " + position);
           writer.newLine();
```

```
    // Method to save profile data
    private void saveProfile() {
        String name = textField_Name.getText();
        String position = textField_Position.getText();
        String email = textField_Email.getText();
        String phoneNumber = textField_PhoneNumber.getText();
        String address = textField_Address.getText();

        // Write profile data to a text file
        try (BufferedWriter writer = new BufferedWriter(new FileWriter("profile.txt"))) {
            writer.write("Name: " + name);
            writer.newLine();
            writer.write("Position: " + position);
            writer.newLine();
            writer.write("Email: " + email);
            writer.newLine();
            writer.write("Phone Number: " + phoneNumber);
            writer.newLine();
            writer.write("Address: " + address);
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

GUI:



The figure above shows the Personal Profile page. The page includes Name, Position, Email, Phone Number and Address.

CODE:

```
package group_Assignment_Java;

import java.awt.EventQueue;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JTable;
import javax.swing.table.DefaultTableModel;
import javax.swing.JScrollPane;
import javax.swing.JLabel;
import javax.swing.JOptionPane;

import java.awt.Font;
import javax.swing.JTextField;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.HashSet;
import java.util.Set;
import java.awt.event.ActionEvent;

public class Manage_Sales_Order_Quotation_Page extends JFrame {

    /**
     *
     */
    private static final long serialVersionUID = 1L;
    private JPanel contentPane;
    private JTable table;
    private JTextField textField_ItemID;
    private JTextField textField_ItemName;
    private JTextField textField_Category;
    private JTextField textField_OrderDate;
    private JTextField textField_Status;
    private JTextField textField_Price;
    private static final String sales_quotation_txt_file = "C:\\Users\\kerta\\OneDrive\\Documents\\OODJ\\SalesQuotation1.txt";

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Manage_Sales_Order_Quotation_Page frame = new Manage_Sales_Order_Quotation_Page();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the frame.
     */
    public Manage_Sales_Order_Quotation_Page() {
        setTitle("Manage Sales Order Quotation Page");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 907, 548);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));

        setContentPane(contentPane);
        contentPane.setLayout(null);

        JScrollPane scrollPane = new JScrollPane();
        scrollPane.setBounds(38, 11, 818, 294);
        contentPane.add(scrollPane);

        table = new JTable();
        scrollPane.setViewportView(table);
        loadDataFromFile();

        table.setModel(new DefaultTableModel(
            new Object[][] {
            },
            new String[] {
                "Item ID", "Item Name", "Category", "Price", "Order Date", "Status"
            }
```

```
) {
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    Class[] columnTypes = new Class[] {
        Integer.class, String.class, String.class, Integer.class, String.class, String.class
    };
    public Class getColumnClass(int columnIndex) {
        return columnTypes[columnIndex];
    }
});

JLabel lblNewLabel = new JLabel("Item ID:");
lblNewLabel.setFont(new Font("Tw Cen MT", Font.BOLD, 17));
lblNewLabel.setBounds(26, 337, 108, 14);
contentPane.add(lblNewLabel);

JLabel lblItemName = new JLabel("Item Name:");
lblItemName.setFont(new Font("Tw Cen MT", Font.BOLD, 17));
lblItemName.setBounds(26, 374, 108, 24);
contentPane.add(lblItemName);

JLabel lblCategory = new JLabel("Category:");
lblCategory.setFont(new Font("Tw Cen MT", Font.BOLD, 17));
lblCategory.setBounds(26, 412, 108, 17);
contentPane.add(lblCategory);

JLabel lblPrice = new JLabel("Price:");
lblPrice.setFont(new Font("Tw Cen MT", Font.BOLD, 17));
lblPrice.setBounds(295, 337, 108, 14);
contentPane.add(lblPrice);

JLabel lblOrderDate = new JLabel("Order Date:");
lblOrderDate.setFont(new Font("Tw Cen MT", Font.BOLD, 17));
lblOrderDate.setBounds(295, 377, 108, 19);
contentPane.add(lblOrderDate);

JLabel lblNewLabel_1 = new JLabel("Status:");
lblNewLabel_1.setFont(new Font("Tw Cen MT", Font.BOLD, 17));
lblNewLabel_1.setBounds(295, 415, 108, 14);
contentPane.add(lblNewLabel_1);
```

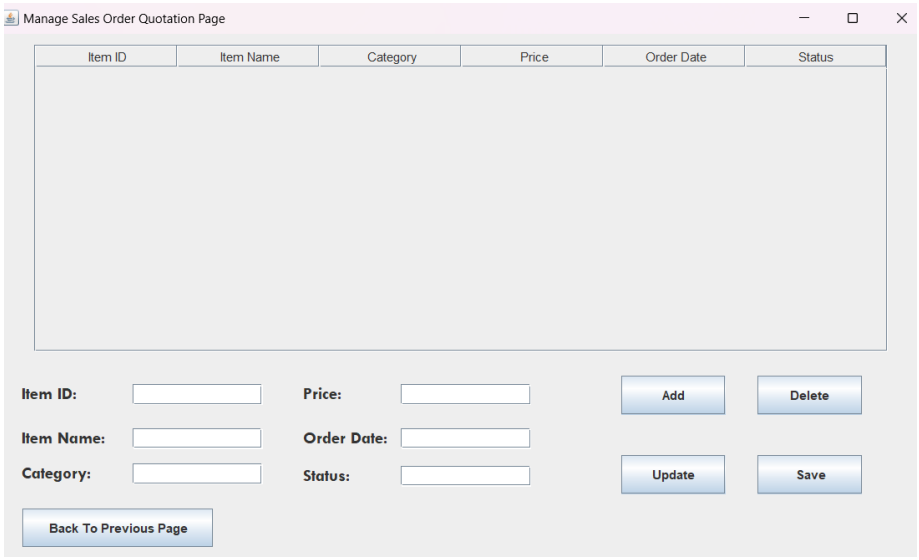
```
textField_ItemID = new JTextField();
textField_ItemID.setBounds(132, 336, 125, 20);
contentPane.add(textField_ItemID);
textField_ItemID.setColumns(10);

textField_ItemName = new JTextField();
textField_ItemName.setColumns(10);
textField_ItemName.setBounds(132, 378, 125, 20);
contentPane.add(textField_ItemName);

textField_Category = new JTextField();
textField_Category.setColumns(10);
textField_Category.setBounds(132, 412, 125, 20);
contentPane.add(textField_Category);

textField_OrderDate = new JTextField();
textField_OrderDate.setColumns(10);
textField_OrderDate.setBounds(389, 378, 125, 20);
contentPane.add(textField_OrderDate);

textField_Status = new JTextField();
textField_Status.setColumns(10);
textField_Status.setBounds(389, 414, 125, 20);
contentPane.add(textField_Status);

textField_Price = new JTextField();
textField_Price.setColumns(10);
textField_Price.setBounds(389, 336, 125, 20);
contentPane.add(textField_Price);
```

```
JButton btn_Add = new JButton("Add");
btn_Add.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        if(validateFields()) {
            int itemID = Integer.parseInt(textField_ItemID.getText());
            String itemName = textField_ItemName.getText();
            String category = textField_Category.getText();
            int price = Integer.parseInt(textField_Price.getText());
            String orderDate = textField_OrderDate.getText();
            String status = textField_Status.getText();

            DefaultTableModel model = (DefaultTableModel) table.getModel();
            model.addRow(new Object[]{itemID, itemName, category, price, orderDate, status});
            textField_ItemID.setText("");
            textField_ItemName.setText("");
            textField_Category.setText("");
            textField_Price.setText("");
            textField_OrderDate.setText("");
            textField_Status.setText("");
        }
        else {
            JOptionPane.showMessageDialog(null, "Please fill in all fields", "Validation Error", JOptionPane.ERROR_MESSAGE);
        }

    }
});
btn_Add.setBounds(600, 328, 100, 37);
contentPane.add(btn_Add);

JButton btn_Delete = new JButton("Delete");
btn_Delete.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
         // Get the selected row index
        int selectedRowIndex = table.getSelectedRow();

        // Check if a row is selected
        if (selectedRowIndex != -1) {
            // Get the model of the table
            DefaultTableModel model = (DefaultTableModel) table.getModel();

            // Remove the selected row
            model.removeRow(selectedRowIndex);
    }


        }
    }
        }
    );
    btn_Delete.setBounds(731, 328, 100, 37);
    contentPane.add(btn_Delete);

    JButton btn_Update = new JButton("Update");
    btn_Update.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            // Get the selected row index
            int selectedRowIndex = table.getSelectedRow();

            // Check if a row is selected
            if (selectedRowIndex != -1) {
                // Get the model of the table
                DefaultTableModel model = (DefaultTableModel) table.getModel();

                // Update the selected row with new values
                for (int columnIndex = 0; columnIndex < model.getColumnCount(); columnIndex++) {
                    String newValue = getTextFieldValue(columnIndex);
                    if (!newValue.isEmpty()) {
                        model.setValueAt(newValue, selectedRowIndex, columnIndex);
                    }
                }

                // Clear the text fields after the update
                clearTextFields();
            }
        }
```

```
            // Helper method to get the value from the corresponding text field
            private String getTextFieldValue(int columnIndex) {
                switch (columnIndex) {
                    case 0:
                        return textField_ItemID.getText();
                    case 1:
                        return textField_ItemName.getText();
                    case 2:
                        return textField_Category.getText();
                    case 3:
                        return textField_Price.getText();
                    case 4:
                        return textField_OrderDate.getText();
                    case 5:
                        return textField_Status.getText();
                    default:
                        return ""; // Handle additional columns if needed
                }
            }

            // Helper method to clear all text fields
            private void clearTextFields() {
                textField_ItemID.setText("");
                textField_ItemName.setText("");
                textField_Category.setText("");
                textField_Price.setText("");
                textField_OrderDate.setText("");
                textField_Status.setText("");
            }
        });

        btn_Update.setBounds(600, 404, 100, 37);
        contentPane.add(btn_Update);

        JButton btn_Save = new JButton("Save");
        btn_Save.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                saveDataToFile();

            }


        });
        btn_Save.setBounds(731, 404, 100, 37);
        contentPane.add(btn_Save);

        JButton btn_BacktoPreviousPage = new JButton("Back To Previous Page");
        btn_BacktoPreviousPage.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                Salesperson_Page salespersonPage = new Salesperson_Page();
                salespersonPage.setVisible(true);
                dispose();
            }
        });
        btn_BacktoPreviousPage.setBounds(27, 455, 182, 37);
        contentPane.add(btn_BacktoPreviousPage);
    }

    private void saveDataToFile() {
        try {
            DefaultTableModel model = (DefaultTableModel) table.getModel();
            int rowCount = model.getRowCount();

            BufferedWriter writer = new BufferedWriter(new FileWriter(sales_quotation_txt_file, true));

            // Write column headers
            if (salesQuotationFileIsEmpty()) {
                for (int i = 0; i < model.getColumnCount(); i++) {
                    writer.write(model.getColumnName(i));
                    if (i < model.getColumnCount() - 1) {
                        writer.write("\t");
                    } else {
                        writer.write("\n");
                    }
                }
            }
```

```
            // Write data
            if (rowCount > 0) {
                int lastRow = rowCount - 1;
                StringBuilder lastRowString = new StringBuilder();
                for (int col = 0; col < model.getColumnCount(); col++) {
                    lastRowString.append(model.getValueAt(lastRow, col).toString());
                    if (col < model.getColumnCount() - 1) {
                        lastRowString.append("\t");
                    } else {
                        lastRowString.append("\n");
                    }
                }

            Set<String> existingRows = readExistingRows();
            if (!existingRows.contains(lastRowString.toString())) {
                    // Write data only if it doesn't already exist
                    writer.write(lastRowString.toString());
                }
            }

            writer.close();
            JOptionPane.showMessageDialog(null, "Data successfully added to " + "txt.file");
        } catch (IOException ex) {
            ex.printStackTrace();
            JOptionPane.showMessageDialog(null, "Error saving data to file: " + ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
        }
    }
    private void loadDataFromFile() {
        try (BufferedReader reader = new BufferedReader(new FileReader(sales_quotation_txt_file))) {
            DefaultTableModel model = (DefaultTableModel) table.getModel();
            String line;

            // Read column headers
            if ((line = reader.readLine()) != null) {
                String[] columnHeaders = line.split("\t");
                model.setColumnIdentifiers(columnHeaders);
            }

            // Read data
            while ((line = reader.readLine()) != null) {
                String[] rowData = line.split("\t");



              // Read data
              while ((line = reader.readLine()) != null) {
                  String[] rowData = line.split("\t");
                  model.addRow(rowData);
              }

          } catch (IOException ex) {
              ex.printStackTrace();
              JOptionPane.showMessageDialog(null, "Error loading data from file: " + ex.getMessage(), "Error",
                      JOptionPane.ERROR_MESSAGE);
          }
      }

      private boolean validateFields() {
          // Validate that all fields are not empty
          return !textField_ItemID.getText().isEmpty()
                  && !textField_ItemName.getText().isEmpty()
                  && !textField_Category.getText().isEmpty()
                  && !textField_Price.getText().isEmpty()
                  && !textField_OrderDate.getText().isEmpty()
                  && !textField_Status.getText().isEmpty();
      }

      private boolean salesQuotationFileIsEmpty() throws IOException {
          BufferedReader reader = new BufferedReader(new FileReader(sales_quotation_txt_file));
          boolean isEmpty = reader.readLine() == null; // Check if the first line is null, indicating an empty file
          reader.close();
          return isEmpty;
      }

      private Set<String> readExistingRows() throws IOException {
          Set<String> existingRows = new HashSet<>();
          BufferedReader reader = new BufferedReader(new FileReader(sales_quotation_txt_file));
          String line;
          while ((line = reader.readLine()) != null) {
              existingRows.add(line);
          }
          reader.close();
          return existingRows;
      }
  }
```

GUI:



The figure above shows the page to manage sales order quotation. The page above shows the field to put in the details.

8.3 OFFICER – AREEBAH IRFAN MOHAMMED

CODE:

Officer class:

```
21  public String getUsername() {
22      return username;
23  }
24
25  // Method to manage personal profile
26  public void manageProfile(String username, String password) {
27      this.username = username;
28      this.password = password;
29      System.out.println("Profile updated successfully!");
30  }
31
32  // Method to process sale order upon sale approval/ closed sale
33  public void processSaleOrder(SaleOrder saleOrder) {
34      if (!saleOrders.containsKey(saleOrder.getOrderID())) {
35          saleOrders.put(saleOrder.getOrderID(), saleOrder);
36          System.out.println("Sale order processed successfully!"
37      } else {
38          System.out.println("Sale order already processed!");
39      }
40  }
41
42  // Method to search sale orders
43  public SaleOrder searchSaleOrder(String orderID) {
44      return saleOrders.get(orderID);
45  }
```

```
1  package group_Assignment_Java;
2
3  import java.util.HashMap;
4  import java.util.List;
5  import java.util.Map;
6
7  //Officer class representing a user with access to system functionalities
8  @SuppressWarnings({ "unused" })
9  class Officer {
10  private String username;
11  @SuppressWarnings("unused")
12  private String password;
13  private Map<String, SaleOrder> saleOrders;
14
15  public Officer(String username, String password) {
16      this.username = username;
17      this.password = password;
18      this.saleOrders = new HashMap<>();
19  }
20
21  public String getUsername() {
22      return username;
23  }
24
25  // Method to manage personal profile
26  public void manageProfile(String username, String password) {
27      this.username = username;
28      this.password = password;
```

```
47  // Method to create a new sale order
48  public void createSaleOrder(SaleOrder saleOrder) {
49      saleOrders.put(saleOrder.getOrderID(), saleOrder);
50      System.out.println("Sale order created successfully!");
51  }
52
53  // Method to modify a sale order
54  public void modifySaleOrder(SaleOrder saleOrder, List<Product> newProducts) {
55      saleOrder.getProducts1().clear();
56      List<Product> products = (List<Product>) saleOrder.getProducts();
57      List<Product> products2 = products;
58      List<Product> products1 = products2;
59      List<Product> list = products1;
60      boolean all = list.addAll(newProducts);
61      System.out.println("Sale order modified successfully!");
62  }
63
64  // Method to remove a sale order
65  public void removeSaleOrder(SaleOrder saleOrder) {
66      saleOrders.remove(saleOrder.getOrderID());
67      System.out.println("Sale order removed successfully!");
68  }
```

```java
// Method to submit sales for production and issue sale invoice
public void submitForProduction(SaleOrder saleOrder) {
    saleOrder.approveSale();
    saleOrder.closeSale();
    System.out.println("Sale submitted for production and invoice issued successfully!");
}

// Method to check sale product status
public String checkProductStatus(SaleOrder saleOrder) {
    // Logic to check product status
    return "Product status: In Progress";
}

// Method to generate reports
public Report generateReport(String reportType) {
    String reportContent = "";
    // Logicto generate report content based on report type
    switch (reportType) {
        case "workDone":
            reportContent = "Work done report content...";
            break;
        case "approvedClosedSale":
            reportContent = "Approved/closed sale report content...";
            break;
        default:
            reportContent = "Invalid report type!";
    }
    return new Report(reportType, reportContent);
```

The `Officer` class in the provided code serves as a representation of a user within a system, specifically focusing on their role in managing sale orders and generating reports. The class uses encapsulation to store and manipulate relevant data, such as the officer's username and password, as private instance variables. The `saleOrders` variable, implemented as a `Map`, allows efficient storage and retrieval of sale orders based on their unique order IDs. The class provides several methods to interact with sale orders. The `processSaleOrder` method adds a sale order to the `saleOrders` map if it doesn't already exist, and the `createSaleOrder` method adds a new sale order to the map. Conversely, the `removeSaleOrder` method removes a sale order from the map. The `modifySaleOrder` method enables the modification of a sale order by replacing its existing list of products with a new list.Other methods focus on different aspects of sale order management. The `searchSaleOrder` method allows searching for a specific sale order based on its order ID. The `submitForProduction` method initiates the process of approving and closing a sale order, while the `checkProductStatus` method is intended to provide information about the status of the products within a sale order. Additionally, the `Officer` class includes functionality to manage personal profiles. The `manageProfile` method allows an officer to update their username and password. Lastly, the `generateReport` method generates reports based on a specified report type. The method uses a `switch` statement to determine the content of the report based on the provided type. Currently, two report types, "workDone" and "approvedClosedSale," are supported, each with its corresponding report content. If an invalid report type is provided, a default message is assigned.

Product class:

```java
package group_Assignment_Java;


//Product class representing furniture items
public class Product {
 private int id;
 private String name;
 private String category;
 private double price;

 public Product(int id, String name, String category, double price) {
     this.id = id;
     this.name = name;
     this.category = category;
     this.price = price;
 }

 public int getId() {
     return id;
 }

 public String getName() {
     return name;
 }

 public String getCategory() {
     return category;
 }

 public double getPrice() {
     return price;
 }
}

```

The provided code defines a `Product` class representing furniture items. This class encapsulates the properties of a product, including its ID, name, category, and price. The class has a constructor that takes these properties as parameters and assigns them to the corresponding instance variables. Additionally, the class provides getter methods for each property, allowing other classes to retrieve the values of the product's ID, name, category, and price. Overall, the `Product` class serves as a blueprint for creating instances of furniture products and accessing their information when needed.

Invoice class:

```java
package group_Assignment_Java;

//Invoice class representing an invoice for a sale order
class Invoice {
 private String invoiceID;
 private SaleOrder saleOrder;

 public Invoice(String invoiceID, SaleOrder saleOrder) {
     this.invoiceID = invoiceID;
     this.saleOrder = saleOrder;
 }

 public String getInvoiceID() {
     return invoiceID;
 }

 public SaleOrder getSaleOrder() {
     return saleOrder;
 }
}

```

The provided code introduces the `Invoice` class, representing an invoice for a sale order. This class encapsulates the properties of an invoice, including the invoice ID and the corresponding sale order. The `Invoice` class has a constructor that takes the invoice ID and the associated sale order as parameters and initializes the respective instance variables. It also provides getter methods to retrieve the invoice ID and the sale order. The `Invoice` class serves as a blueprint for creating invoice objects that store information about a specific sale order, enabling easy access to the invoice details when needed.

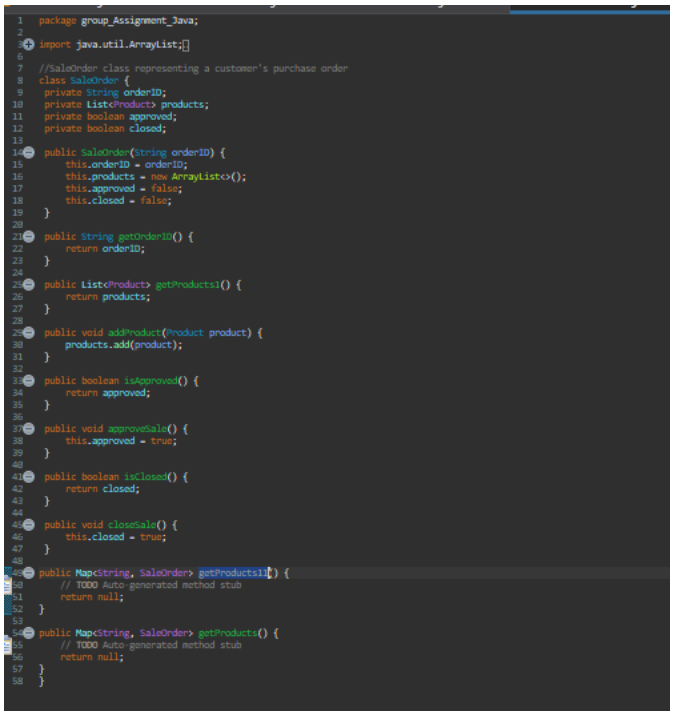
Report class:

```java
package group_Assignment_Java;

//Report class representing various types of reports
class Report {
 private String type;
 private String content;

 public Report(String type, String content) {
     this.type = type;
     this.content = content;
 }

 public String getType() {
     return type;
 }

 public String getContent() {
     return content;
 }
}

```

The provided code introduces the `Report` class, which represents various types of reports. This class encapsulates two properties: the type of the report and its corresponding content. The `Report` class includes a constructor that takes the report type and content as parameters and initializes the respective instance variables. It also provides getter methods to retrieve the

report type and content. The `Report` class serves as a blueprint for creating report objects with specific types and content, allowing easy access to these details when generating or working with reports in a system.

SaleOrder class:



```
package group_Assignment_Java;

import java.util.ArrayList;

//SaleOrder class representing a customer's purchase order
class SaleOrder {
 private String orderID;
 private List<Product> products;
 private boolean approved;
 private boolean closed;

 public SaleOrder(String orderID) {
     this.orderID = orderID;
     this.products = new ArrayList<>();
     this.approved = false;
     this.closed = false;
 }

 public String getOrderID() {
     return orderID;
 }

 public List<Product> getProducts1() {
     return products;
 }

 public void addProduct(Product product) {
     products.add(product);
 }

 public boolean isApproved() {
     return approved;
 }

 public void approveSale() {
     this.approved = true;
 }

 public boolean isClosed() {
     return closed;
 }

 public void closeSale() {
     this.closed = true;
 }

public Map<String, SaleOrder> getProducts1() {
    // TODO Auto-generated method stub
    return null;
}

public Map<String, SaleOrder> getProducts() {
    // TODO Auto-generated method stub
    return null;
}
}
```

The provided code presents the `SaleOrder` class, representing a customer's purchase order. This class encapsulates the properties of a sale order, including the order ID, a list of products, and the status of approval and closure. The `SaleOrder` class features a constructor that takes the order ID as a parameter and initializes the instance variables. It also provides getter methods to retrieve the order ID, the list of products, and the status of approval and closure. Additionally, there are methods to add a product to the order, approve the sale, and close the sale. The `SaleOrder` class serves as a blueprint for creating sale order objects, allowing easy access to the order details and enabling operations such as adding products, approving the sale, and closing the sale within a system.

YoyoFurnitureGUI1 class:

```java
package group_Assignment_Java;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JFrame;

// GUI class for Yoyo Furniture System
public class YoyoFurnitureGUI1 extends JFrame {

    /**
     *
     */
    private static final long serialVersionUID = 1L;
    // GUI components
    private JButton manageProfileButton;
    private JButton processSaleOrderButton;
    private JButton searchSaleOrderButton;
    private JButton createSaleOrderButton;
    private JButton modifySaleOrderButton;
    private JButton removeSaleOrderButton;
    private JButton submitForProductionButton;
    private JButton checkProductStatusButton;
    private JButton generateReportButton;
    private YoyoFurnitureGUI1 frame;

    public void YoyoFurnitureGUI() {
        // Set up the frame
        setTitle("Yoyo Furniture System");
        setSize(400, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new GridLayout(3, 3));

        // Initialize GUI components
        manageProfileButton = new JButton();
        processSaleOrderButton = new JButton();
        searchSaleOrderButton = new JButton();
        createSaleOrderButton = new JButton();
        modifySaleOrderButton = new JButton();
        removeSaleOrderButton = new JButton();
        submitForProductionButton = new JButton();
        checkProductStatusButton = new JButton();
        generateReportButton = new JButton();
```

The provided code introduces the `YoyoFurnitureGUI1` class, which represents a graphical user interface (GUI) for the Yoyo Furniture System. This class extends the `JFrame` class from the Swing library, enabling the creation of a window-based interface. The GUI consists of several buttons, each representing a specific functionality within the furniture system. These buttons include options such as managing profiles, processing sale orders, searching sale orders, creating sale orders, modifying sale orders, removing sale orders, submitting orders for production, checking product status, and generating reports. Each button is associated with an action listener, implemented as an anonymous inner class. When a button is clicked, the corresponding action is triggered, and the associated code block is executed. This allows for the customization of functionality and the implementation of specific logic for each button. The GUI components are added to the frame using the `add()` method,

8.4 AUTHORIZATION AND AUTHENTICATION – HOUSHIANEE HARHANGI

Administrator_Account

```
package javaapplication2;

/**
 *
 * @author Cody
 */
public class AdministratorPage extends javax.swing.JFrame {
    String username;
    /**
     * Creates new form Trainer
     */
    public AdministratorPage() {
        initComponents();
    }

    AdministratorPage(String text){
        this();
        username = text;
        jLabel1.setText("Welcome Back, " + username);
    }

    @SuppressWarnings("unchecked")
    Generated Code

    private void logOutActionPerformed(java.awt.event.ActionEvent evt) {
        Login_done l = new Login_done(2);
        l.show();
        dispose();
    }

    private void editClassActionPerformed(java.awt.event.ActionEvent evt) {
        Administrator_Workers ec = new Administrator_Workers(username);
        ec.show();
        dispose();
    }

    private void generateReportActionPerformed(java.awt.event.ActionEvent evt) {
```

javaapplication2.AdministratorPage

```
    private void generateReportActionPerformed(java.awt.event.ActionEvent evt) {
        Administrator_GenerateReport ogr = new Administrator_GenerateReport(username);
        ogr.show();
        dispose();
    }

    private void accountActionPerformed(java.awt.event.ActionEvent evt) {
        Administrator_Account oa = new Administrator_Account(username);
        oa.show();
        dispose();
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        Look and feel setting code (optional)
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new AdministratorPage().setVisible(true);
            }
        });
```

javaapplication2.AdministratorPage

```
191                 }
192             });
193         }
194
195         // Variables declaration - do not modify
196         private javax.swing.JButton account;
197         private javax.swing.JButton editClass;
198         private javax.swing.JButton generateReport;
199         private javax.swing.JLabel jLabel1;
200         private javax.swing.JPanel jPanel1;
201         private javax.swing.JButton logOut;
202         // End of variables declaration
203     }
204
```

GUI:

It appears that the class constructor receives the username variable, which enables the jLabel1 component to render a customised welcome message. Following "Welcome Back" is the alias on this label.

When the Workers button (editClass) is clicked, the editClassActionPerformed method is called, which closes the graphical user interface and loads the Administrator_Workers page with the user's username.

The generateReport action is initiated when the generateReport button (generateReport) is clicked. This action terminates the current window and launches Administrator_GenerateReport with the specified username in order to generate the report.

To manage the administrator's account settings, this button invokes accountActionPerformed to open the Administrator_Account window and close the present one.

The action listener for the logout icon (logOut) terminates the current window and initiates the Login_done process to log the user out.

This method launches the application, configures the graphical user interface, and renders the AdministratorPage.

Landing_done

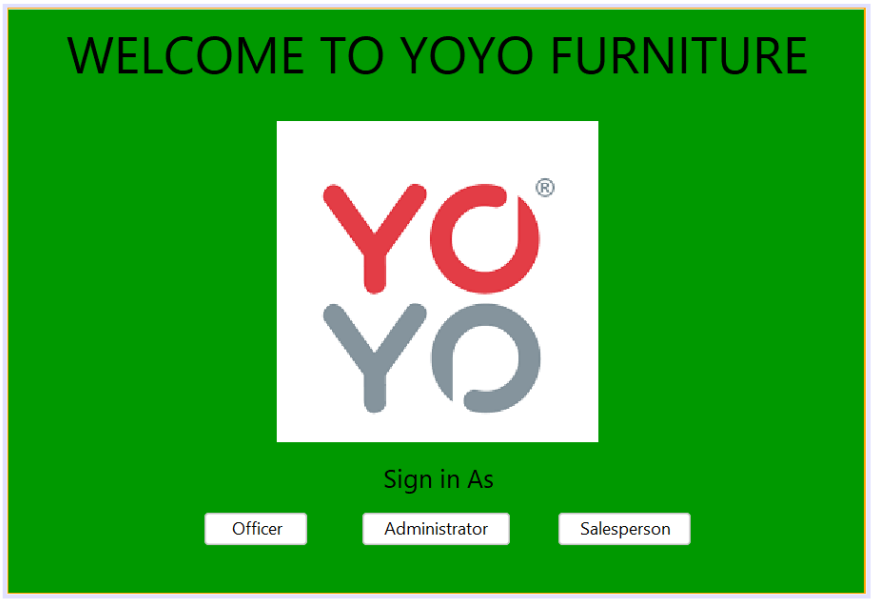
CODE;

```
package javaapplication2;

/**
 *
 * @author Cody
 */
public class Landing_done extends javax.swing.JFrame {

    /**
     * Creates new form Login
     */
    public Landing_done() {
        initComponents();
    }
    void signIn(int i){
        Login_done l = new Login_done(i);
        l.show();
        dispose();
    }
    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    Generated Code

    private void officerSignInActionPerformed(java.awt.event.ActionEvent evt) {
        signIn(1);
    }

    private void administratorSignInActionPerformed(java.awt.event.ActionEvent evt) {
        signIn(2);
    }

    private void salespersonSignInActionPerformed(java.awt.event.ActionEvent evt) {
```

javaapplication2.Landing_done

```
128         signIn(2);
129     }
130
131     private void salespersonSignInActionPerformed(java.awt.event.ActionEvent evt) {
132         signIn(3);
133     }
134
135     /**
136      * @param args the command line arguments
137      */
138     public static void main(String args[]) {
139         /* Set the Nimbus look and feel */
140         Look and feel setting code (optional)
161         //</editor-fold>
162         //</editor-fold>
163         //</editor-fold>
164
165         /* Create and display the form */
166         java.awt.EventQueue.invokeLater(new Runnable() {
167             public void run() {
168                 new Landing_done().setVisible(true);
169             }
170         });
171     }
172
173     // Variables declaration - do not modify
174     private javax.swing.JButton administratorSignIn;
175     private javax.swing.JLabel jLabel1;
176     private javax.swing.JLabel jLabel2;
177     private javax.swing.JLabel jLabel3;
178     private javax.swing.JPanel jPanel1;
179     private javax.swing.JButton officerSignIn;
180     private javax.swing.JButton salespersonSignIn;
181     // End of variables declaration
182 }
183
```

GUI:



GUI windows, including Landing_done, extend JFrame. Without displaying their contents, the initComponents() method initialises components. It appears that the signIn() method generates a new Login_done window populated with an integer value that corresponds to the user role (e.g., officer, administrator, merchant).

The officerSalesperson, administratorSignInActionPerformed, and SignInActionPerformedDifferent integers are passed to signIn(int) via the SignInActionPerformed method in order to denote unique user categories.

Main establishes the aesthetic and functional characteristics of the application and launches Landing_done. Using this technique, the GUI application is launched.

Login_done

CODE:

```
package javaapplication2;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JOptionPane;

/**
 *
 * @author Cody
 */
public class Login_done extends javax.swing.JFrame {
    File f = new File("");
    int condition;
    /**
     * Creates new form StaffLogin
     */
    public Login_done(int par) {
        initComponents();
        switch (par) {
            case 1:
                f = new File("officer.txt");
                condition = 1;
                title.setText("Officer Login");
                break;
            case 2:
                f = new File("administrator.txt");
                register.setVisible(false);
                condition = 2;
                title.setText("Administrator Login");
                break;
            case 3:
                f = new File("salesperson.txt");
```

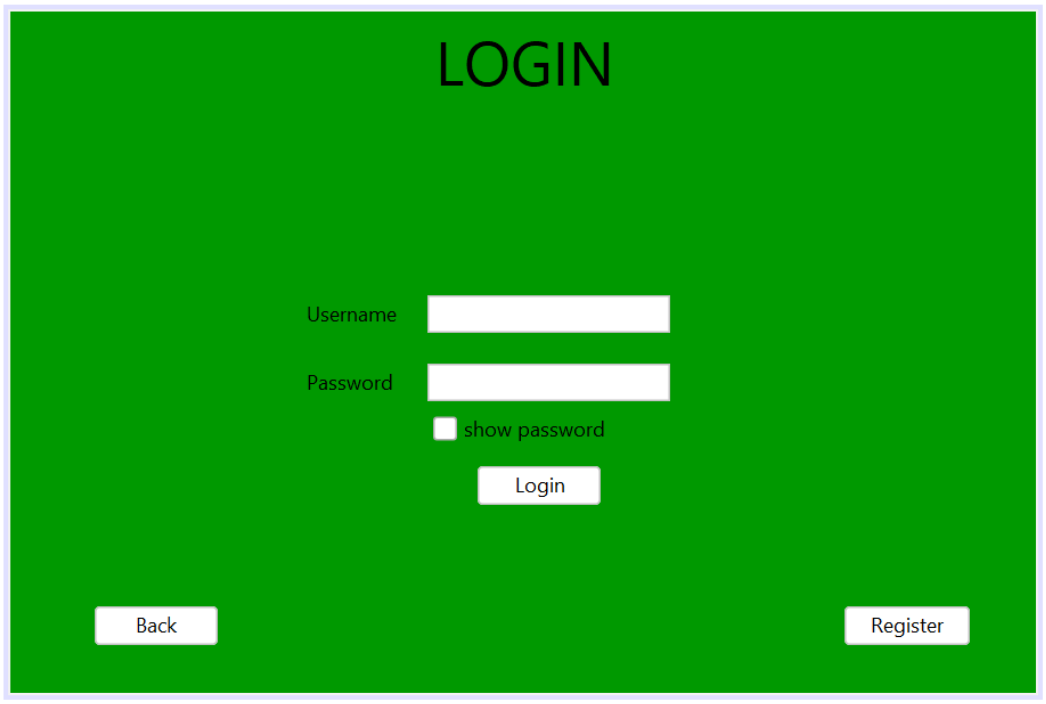
```
            case 3:
                f = new File("salesperson.txt");
                register.setVisible(false);
                condition = 3;
                title.setText("Salesperson Login");
                break;
            default:
                JOptionPane.showMessageDialog(null, "Invalid Input.");
                break;
        }
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    Generated Code

    private void showActionPerformed(java.awt.event.ActionEvent evt) {
        if(show.isSelected()){
            password.setEchoChar((char)0);
        }else{
            password.setEchoChar('*');
        }
    }

    private void loginActionPerformed(java.awt.event.ActionEvent evt) {
        boolean status = true;
        try (BufferedReader br = new BufferedReader(new FileReader(f))) {
            String line;

            while ((line = br.readLine()) != null) {
                String[] fields = line.split(",");

                if (fields.length > 0) {
                    String fileUsername = fields[0].trim();
                    String filePassword = fields[1].trim();

                    if (fileUsername.equals(username.getText()) && filePassword.equals(password.getText())) {
```

```
                if (fileUsername.equals(username.getText()) && filePassword.equals(password.getText())) {
                    JOptionPane.showMessageDialog(null, "Successfully Log In");
                    switch (condition){
                        case 1:
                            OfficerPage op = new OfficerPage(username.getText());
                            op.show();
                            dispose();
                            status = false;
                            break;
                        case 2:
                            AdministratorPage ap = new AdministratorPage(username.getText());
                            ap.show();
                            dispose();
                            status = false;
                            break;
                        case 3:
                            SalespersonPage sp = new SalespersonPage(username.getText());
                            sp.show();
                            dispose();
                            status = false;
                            break;
                    }
                }
            }
        }
        if(status){
            JOptionPane.showMessageDialog(null, "Invalid Credentials, please try again");
        }
    } catch (FileNotFoundException ex) {
        Logger.getLogger(Register.class.getName()).log(Level.SEVERE, null, ex);
    } catch (IOException ex) {
        Logger.getLogger(Register.class.getName()).log(Level.SEVERE, null, ex);
    }

}

private void backActionPerformed(java.awt.event.ActionEvent evt) {
    Landing_done landing = new Landing_done();
    landing.show();
    dispose();
```

```
    dispose();
}

private void registerActionPerformed(java.awt.event.ActionEvent evt) {
    Register register = new Register();
    register.show();
    dispose();
}

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
    /* Set the Nimbus look and feel */
    Look and feel setting code (optional)
    //</editor-fold>
    //</editor-fold>
    //</editor-fold>
    //</editor-fold>
    //</editor-fold>
    //</editor-fold>
    //</editor-fold>
    //</editor-fold>
    //</editor-fold>
    //</editor-fold>
    //</editor-fold>
    //</editor-fold>
    //</editor-fold>
    //</editor-fold>
    //</editor-fold>

    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new Login_done(1).setVisible(true);
        }
    });
}

// Variables declaration - do not modify
private javax.swing.JButton back;
```

```
292     }
293
294     // Variables declaration - do not modify
295     private javax.swing.JButton back;
296     private javax.swing.JLabel jLabel1;
297     private javax.swing.JLabel jLabel2;
298     private javax.swing.JPanel jPanel1;
299     private javax.swing.JButton login;
300     private javax.swing.JPasswordField password;
301     private javax.swing.JButton register;
302     private javax.swing.JCheckBox show;
303     private javax.swing.JLabel title;
304     private javax.swing.JTextField username;
305     // End of variables declaration
306 }
307
```

GUI:



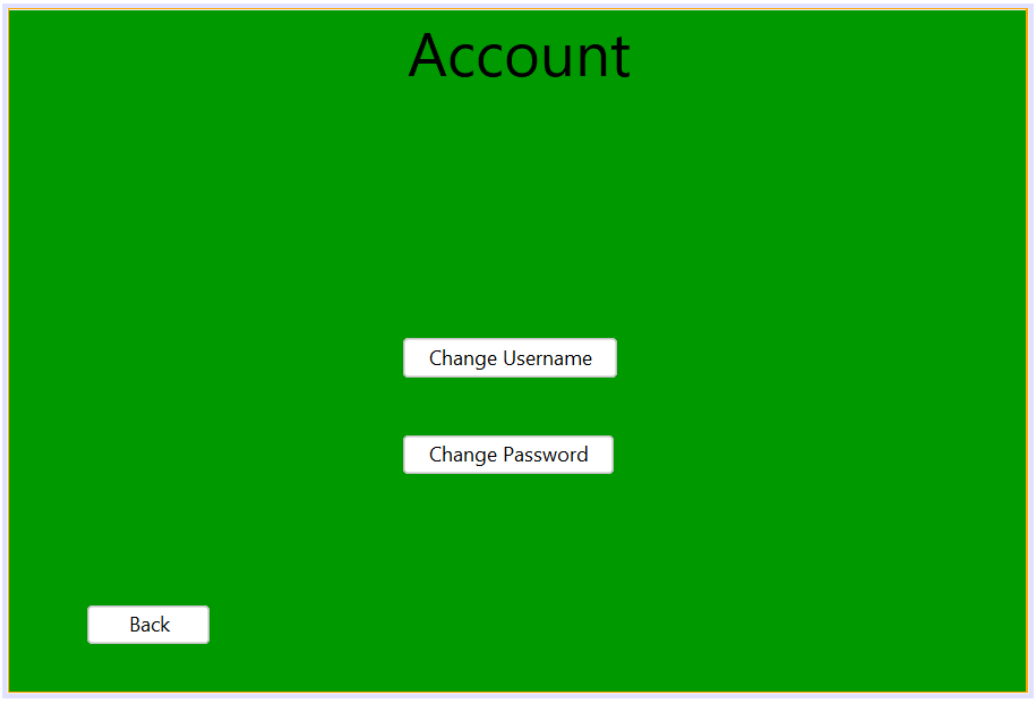
ACCOUNT_LOGIN

CODE:

```
package javaapplication2;

/**
 *
 * @author Cody
 */
public class Officer_Account extends javax.swing.JFrame {
    String username;
    /**
     * Creates new form Officer_Account
     */
    public Officer_Account(String user) {
        initComponents();
        username = user;
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    Generated Code

    private void changeUsernameActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void changePasswordActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void backActionPerformed(java.awt.event.ActionEvent evt) {
        OfficerPage op = new OfficerPage(username);
        op.show();
        dispose();
    }
javaapplication2.Officer_Account
```

```
        dispose();
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        Look and feel setting code (optional)

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            private String user;
            public void run() {
                new Officer_Account(user).setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JButton back;
    private javax.swing.JButton changePassword;
    private javax.swing.JButton changeUsername;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JPanel jPanel1;
    // End of variables declaration
}
```

GUI:



Page of Login GUI: To log in, enter your username and password. By analysing text files for user credentials, the Login_done class authorises access to user pages according to the user's role (e.g., officer, administrator, salesperson).

Landing Page GUI: Users are redirected to their role-specific landing page upon logging in. Users can view sales history, submit sales, verify product status, generate reports, manage accounts, or log out via the icons that appear on this page.

9.0 ADDITIONAL FEATURES

9.1 ABDUL MUHAIMIN AMAN

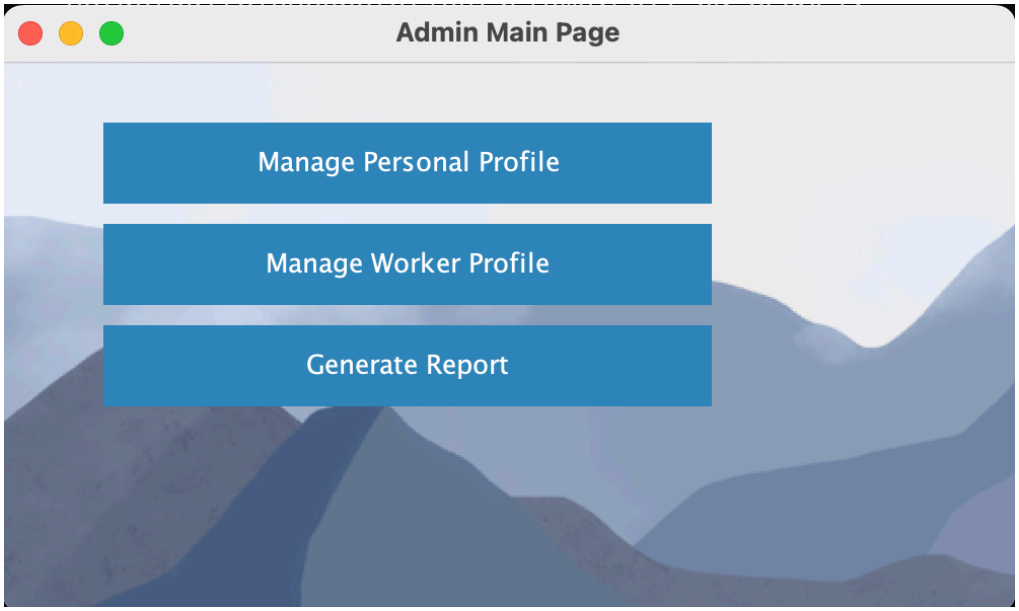
9.1.0 BACKGROUND IMAGE

CODE:

```
    // Create a JLabel to hold the background image
    JLabel backgroundImageLabel = new JLabel();
    setContentPane(backgroundImageLabel);

    // Load the background image
    ImageIcon backgroundImage = new ImageIcon("/Users/aman/eclipse-workspace/oodjaman/src/oodjaman/pngtree-big-mountain-bo
    backgroundImageLabel.setIcon(backgroundImage);
```

```
// Set bounds for the buttons and background image
managePersonalProfileButton.setBounds(50, 30, 300, 40);
manageWorkerProfileButton.setBounds(50, 80, 300, 40);
generateReportButton.setBounds(50, 130, 300, 40);
backgroundImageLabel.setBounds(0, 0, getWidth(), getHeight());
```

GUI:



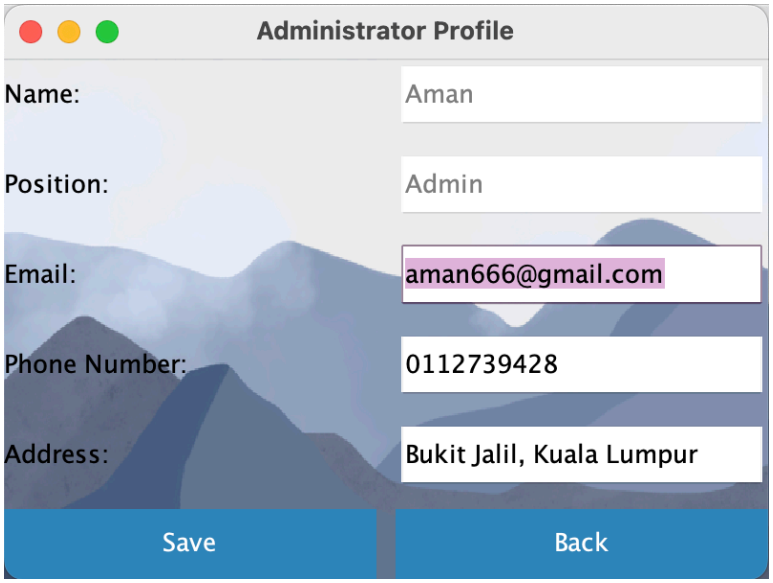
9.1.1 STYLED BUTTONS

CODE:

```
        saveButton = createStyledButton("Save");
        saveButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                saveChanges();
            }
        });

        // Create "Go Back" button
        goBackButton = createStyledButton("Back");
        goBackButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                dispose(); // Close the current frame
            }
        });
```

```
    private JButton createStyledButton(String text) {
        JButton button = new JButton(text);
        button.setBackground(new Color(70, 130, 180)); // Light blue
        button.setForeground(Color.WHITE);
        button.setFocusPainted(false);
        button.setBorderPainted(false);
        button.setOpaque(true);
        return button;
    }
```

GUI:

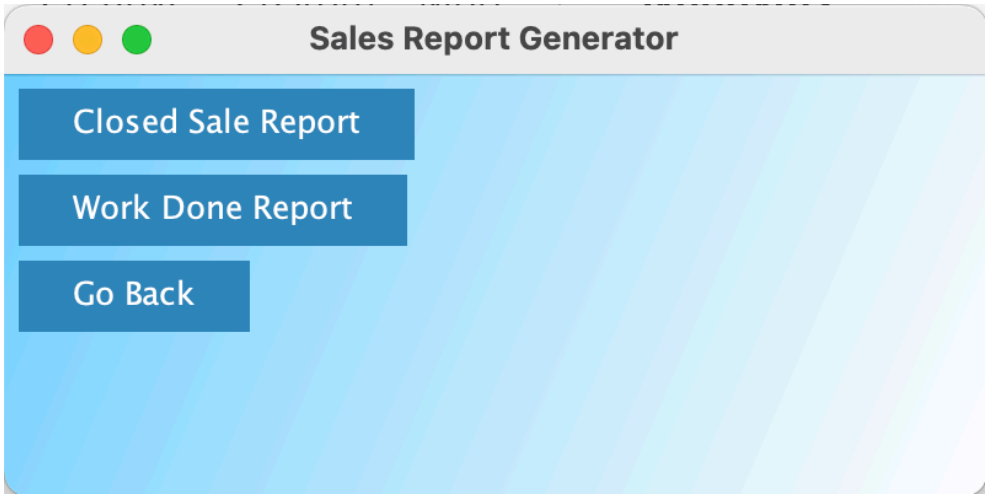


9.1.2 GRADIENT COLOUR BACKGROUND

CODE:

```
// Set up gradient background
setContentPane(new JPanel() {
    @Override
    protected void paintComponent(Graphics g) {
        Graphics2D g2d = (Graphics2D) g;
        Color color1 = new Color(135, 206, 250); // Light blue
        Color color2 = new Color(255, 255, 255); // White
        GradientPaint gp = new GradientPaint(0, 0, color1, getWidth(), getHeight(), color2);
        g2d.setPaint(gp);
        g2d.fillRect(0, 0, getWidth(), getHeight());
    }
});
```

GUI:

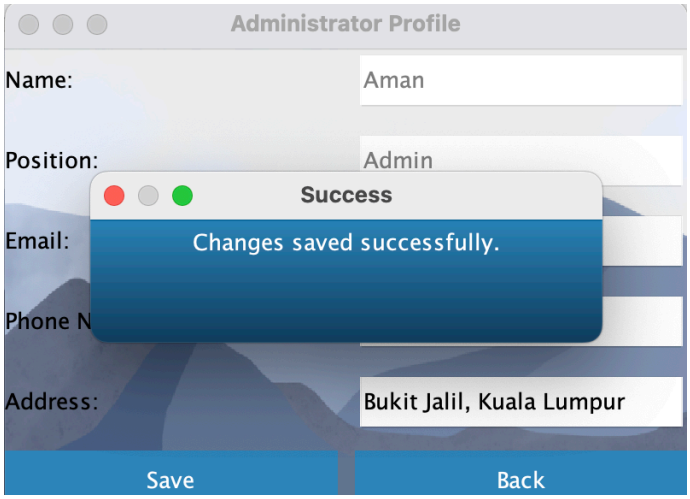


9.1.3 TIMER FOR DIALOG BOX

CODE:

```
        // Set a timer to close the dialog after 2 seconds (2000 milliseconds)
        Timer timer = new Timer(2000, new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                dialog.dispose();
            }
        });
        timer.setRepeats(false); // Only run the timer once
        timer.start();

        dialog.setVisible(true);
    }
```

GUI:

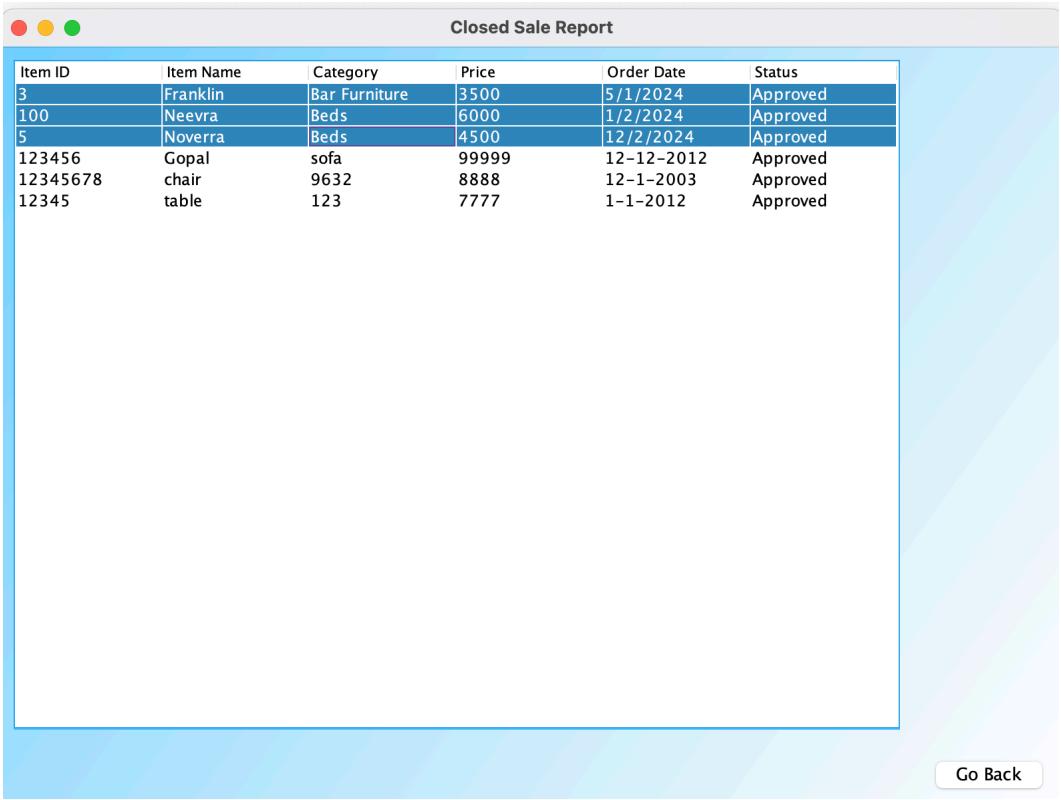


9.1.3 ALTERNATING ROW COLOURS

CODE:

```
            // Handle selection state
            if (isRowSelected(row)) {
                comp.setBackground(new Color(70, 130, 180)); // Light blue for selected row
            } else {
                comp.setBackground(row % 2 == 0 ? Color.WHITE : new Color(255, 255, 255)); // Alternating row colors
            }

            return comp;
        }
    };
```

GUI:

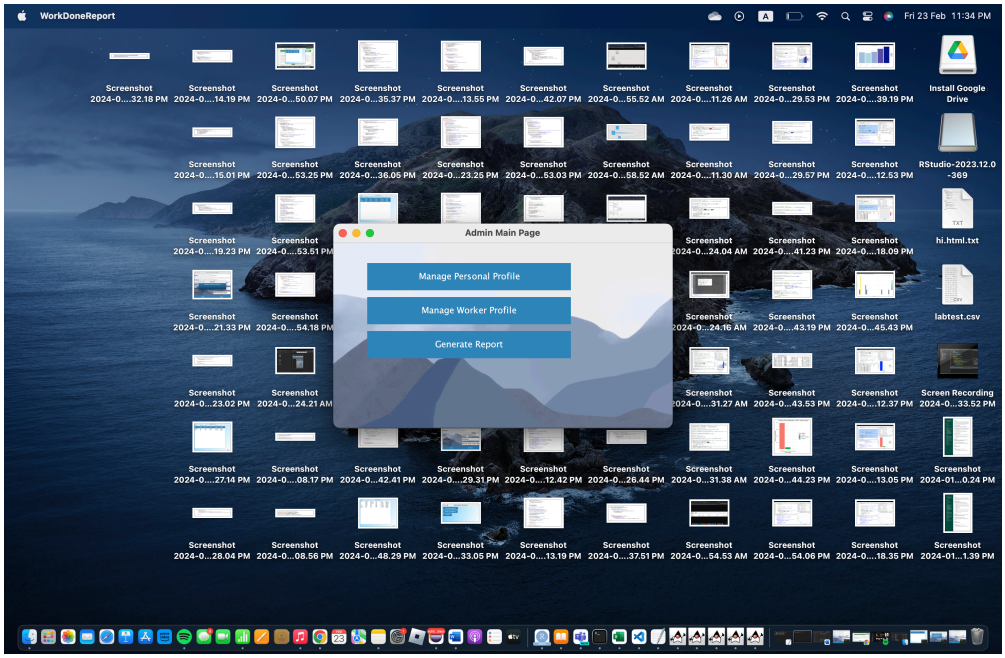


9.1.4 CENTERING FRAME TO SCREEN

CODE:

```
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
```

GUI:



10.0 CONCLUSION

To sum up, the Yoyo-Furniture Sales Management System provides a comprehensive approach to addressing the difficulties encountered by the furniture enterprise in overseeing its sales activities. The system offers a user-friendly interface and strong functionality for simplifying sale order processing, invoice production, report generation, and product administration by using contemporary technology and object-oriented design concepts. The system reduces possible security concerns by providing officers, administrators, and salespeople with safe access control using authentication and authorization processes. All things considered, the Yoyo-Furniture Sales Management System is expected to improve productivity, accuracy, and efficiency in duties linked to sales, which will eventually help the firm succeed and expand in the highly competitive industry.

11.0 WORKLOAD MATRIX

NAME	TP NUMBER	TASKS
ABDUL MUHAIMIN AMAN	TP069510	• Administrator: · Manage personal profile · Manage worker profile (saleperson and officer) · Generate report (Type: work done report, approved/closed sale report) • Introduction • Table of content • Objective • Scope • Limitations • Class Diagram • Concepts used • Implementation • Additional features • Compiling Documentation • Workload Matrix
KERTANAA A/P KUMAR	TP072664	• Salesperson: · Manage personal profile · Manage sale order quotation (create/modify/remove/search) · List all personal sale orders (unapproved/approved) • Use case diagram • Concepts used • Implementation • Documentation

		• Conclusion • Workload Matrix
HOOUSHIANEE HARHANGI	TP066080	• GUI interface • Concepts used • Authentication and Authorisation • Implementation
AREEBAH IRFAN MOHAMMED	TP072275	• Officer: · Manage personal profile · Process sale order upon sale approval/ closed sale · Submit sales for production · Check sale product status for work done or in progress status · Generate report (Type: work done report, approved/closed sale report • Implementation