

PROJECT REPORT

To fulfill the assignment for the Object Oriented Visual Programming course

Lecturer/Course Instructor: Williem



Compiled by:

Johana Veronica Setiawan (001202400206)

Keira Nevrada Lay (001202400170)

Yosmia Khaira Nisa (001202400108)

PRESIDENT UNIVERSITY

FACULTY OF COMPUTING

CIKARANG UTARA

2025

TABLE OF CONTENTS

TABLE OF CONTENTS.....	2
PART A.....	3
A.1. OVERVIEW.....	3
A.2. BUSINESS FUNCTION.....	8
A.3. DATA REQUIREMENTS.....	10
PART B.....	12
B.1. START POINT.....	12
B.2. LOGIN PAGE.....	12
B.3. SIGN UP PAGE.....	13
B.4. MANAGE ORDER (USER PAGE).....	14
B.5. ADMIN MENU PAGE.....	20
B.6. USER CONTROL PANEL PAGE (ADMIN).....	21
B.7. CATEGORY PAGE (ADMIN).....	22
B.8. PRODUCT PAGE (ADMIN).....	24
B.9. VIEW ORDER PAGE (ADMIN).....	27
PART C.....	28
C.1. DATABASE.....	28
C.2. USER TABLE.....	28
C.3. CATEGORY TABLE.....	28
C.4. PRODUCT TABLE.....	29
C.5. PURCHASE TABLE.....	29
PART D.....	30

Link Github: https://github.com/keinvl/oovp_project.git

PART A

INTRODUCTION

A.1. OVERVIEW

The Inventory Management System is a desktop-based visual application developed using Java Swing for its graphical user interface and MySQL as the backend database. With its straightforward and accessible interface, this system streamlines inventory tasks, featuring a clean interface that minimizes the requirement for multiple windows.

The app employs JPanel elements to update what is shown in the main window instead. This method optimizes user satisfaction by ensuring that everything is organized, making it effortless to switch between tasks like managing products, logging in, or viewing transaction histories.

Utilizing essential object-oriented concepts, the system allows users to perform create, read, update, and delete operations on data stored in a MySQL database. It delivers impressive features for users and managers, such as instant updates, well-organized data views, and secure data processing techniques. The user-friendly and streamlined layout facilitates rapid interaction with the system, making it an excellent choice for real-world inventory and retail usage.

Upon their initial login, users will be directed to the login page to verify their details. The login system verifies whether the email and password provided by users correspond with the information saved in the database. Once the user login successfully, a session is initiated to ensure the user's identity is maintained.

The screenshot shows a web browser window titled "LOGIN". On the left, there is a large olive-green rectangle with the text "WELCOME" in white, followed by "Inventory Management System" in a smaller font. At the bottom of this rectangle, it says "© 2025 IT4 OQVP PROJECT". On the right, the word "LOGIN" is displayed in bold. Below it are two input fields labeled "Email" and "Password". A "Login" button is positioned below the password field. At the bottom right, there is a link "I don't have an account" followed by a "Sign Up" button.

Image A.1.1: Log-in Page

For users who do not have an account, the system provides a “Sign Up” feature. Users are asked to complete a registration form by entering essential information such as a unique name, email address, and a secure password with confirmation. If the email account is already in use, the system will notify the user that the email is taken. When the registration is completed, the user will proceed to the login page.

The screenshot shows a web browser window titled "Sign Up". The word "SIGN UP" is centered at the top. Below it are three input fields labeled "Full name", "Email", and "Password". A "Sign Up" button is located below the password field. At the bottom, there is a link "I've an account" followed by a "Login" button.

Image A.1.2: Register Page

In this instance, the roles designated as "Admin" and "User" are employed. Upon successful authentication with the "Admin" role, the user will be redirected to the

AdminMain page, which will immediately display four menus: "User" which will present the ManageUser page, "Category" which will present the AdminCategory page, "Product" which will present the AdminProduct page, and "Order" which will present the ViewOrder page. This menu also includes an "Log Out" option, which redirects the user to the login page.

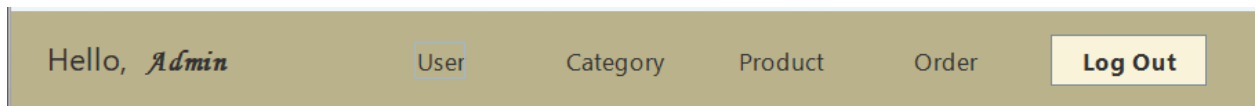


Image A.1.3: Admin Menu Page

Upon accessing the "User" menu, the Admin Control Panel will be displayed. In this section, the administrator is able to execute the CRUD operations on user accounts. Administrators are able to initiate the addition of new accounts on this page, in addition to the modification of data pertaining to name, email, password, and role. Additionally, the administrator has the capability to remove users as necessary.

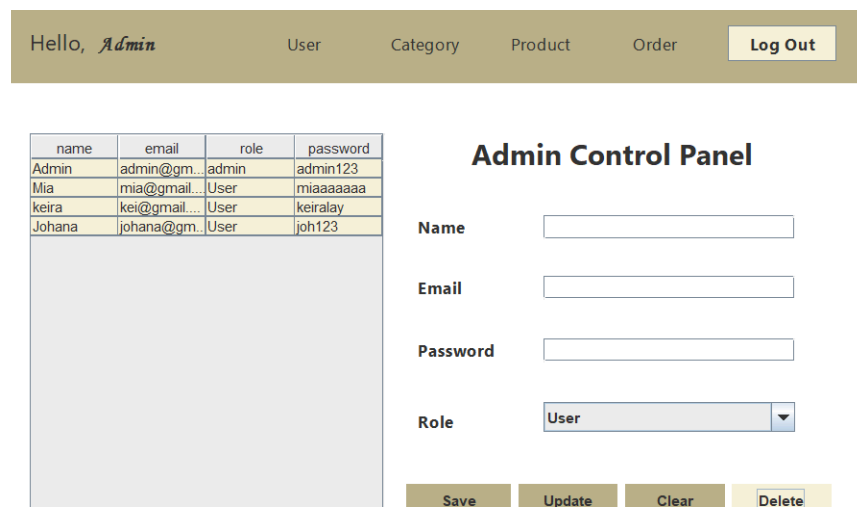
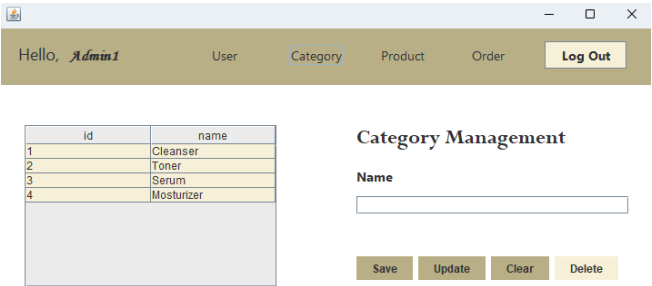


Image A.1.4: Admin Control Panel Page

When accessing the Category, the administrator is provided with a user-friendly form designed for creating new product categories. This feature is essential for managing vast inventories, ensuring that numerous products are arranged with labels that are clear and easy to interpret. The administrator can execute CRUD process in the category management.



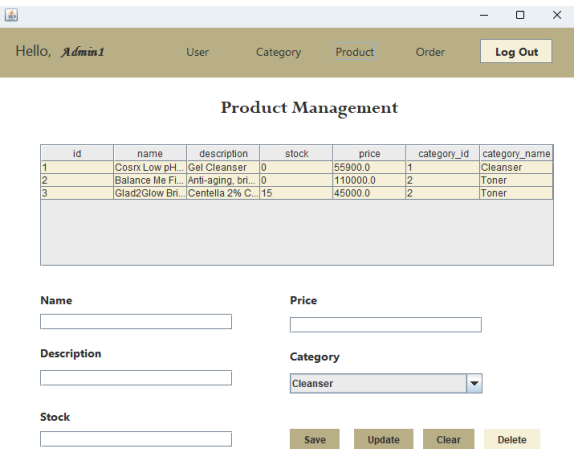
id	name
1	Cleanser
2	Toner
3	Serum
4	Moisturizer

Category Management

Name

Image A.1.5: Admin Category Page

Product management enables administrators to monitor and oversee the core inventory items listed in the system. Within this module, administrators have the ability to categorize products appropriately, establish pricing, monitor inventory levels, and ensure that the product details are accurate and current. This interface simplifies the process of adding and updating products, ensuring the business operates efficiently by providing users with up to date inventory information.



Product Management

id	name	description	stock	price	category_id	category_name
1	Coarn Low pH	Gel Cleanser	0	55900.0	1	Cleanser
2	Balance Me FL	Anti-aging, br...	0	110000.0	2	Toner
3	Glad2Glow Br	Centella 2% C...	15	45000.0	2	Toner

Name

Price

Description

Category

Stock

Image A.1.6: Admin Product Page

In the final administrative menu, entitled "Order," the administrator is able to view a record of the purchases made by users. While the administrator lacks direct insight into the specific user who purchased the product, it is possible to access a comprehensive record of transactions involving the release of goods, including the precise amount and total price. This transaction history facilitates effective management of product inventory.



id	product_name	quantity	price	description	subtotal
1	Cosrx Low pH Goo...	3	55900.0	Gel Cleanser	167700.0
2	Balance Me Fine T...	10	110000.0	Anti-aging, brighte...	1100000.0
3	Cosrx Low pH Goo...	9	55900.0	Gel Cleanser	503100.0
4	Balance Me Fine T...	10	110000.0	Anti-aging, brighte...	1100000.0

Delete

Image A.1.7: Admin View Order Page

This form is made simple and effective to make it easier for users in transactions. After the user logs in, this form will appear, and it will directly connect to the database and retrieve product data for display in the product table. In this form, it allows users to view product details, add products to the order cart and decide which products to buy, before the order is saved in the database.

Order

Products

ID	Name	Price	Stock	Description	Category ID	Category ...
1	smartwatch	10000.0	9	pretty, cool	1	elect
2	headphone	2000.0	10	loud	1	elect

Selected Prodeuct:

Product Name

Product Price

Product Description

Order Quantity

Add to Cart

Cart

Product Id	Name	Quantity	Price	Description	Sub Total

Total Amount: 00000

Save Order Details

Reset

Close

Form A.1.8 (Order Page)

A.2. BUSINESS FUNCTION

The inventory management system is designed for two types of users: standard users, who are customers, and administrators, who are referred to as admins. Depending on their role, every user is granted access to specific sections of the inventory system that assist them in performing their duties. These categories encompass the different roles available in the business sector.

The system should allows users to:

- Registration

Allows users to create an account and access user pages for shopping cart, and product checkout.

- Login

Allows registered users to authenticate and access the system, specifically the ManageOrder page where they can review and place orders.

- Product Catalog

Allows users to find all product categories along with the product name, a short description, stock availability, and price. Users can also click on a product and enter the quantity they want to purchase.

- Shopping Cart

Allows users to save products before purchasing, as well as adding or removing products based on their desire.

- Save Order

Once the cart is finalized, users can submit (save) the order to the database, officially registering their transaction in the system.

- Reset

Allows users to reset form inputs or clear the cart during order placement for ease of re-selection.

- Manage User

Allow admin to add more manage users account, admin can create new account, read the users data that already exists, update users account, and delete users account if needed.

- Create Category

Allow admin to create more categories, update category name, clear category and also delete the category.

- Create Product

Allow admin to create more products, update products (price, stock, etc), clear products and also delete the product.

- View Order

Allow admin to view users' orders from the ManageOrder page. View Order Page shows history of orders, also can delete history of orders if they needed.

- Log Out

Allows admin to log out their account whenever they want, and revert them to the login page.

A.3. DATA REQUIREMENTS

1. User Register Information (email, name, and password)
2. User Login Information (email, password)
3. Product Information (product name, short description, stock, price)
4. Shopping Cart Information (product name, product price, quantity of product, total price)
5. Manage User Information (email, name, and password) & CRUD
6. Manage Category Information (category name) & CRUD
7. Manage Product Information (product name, short description, stock, price) & CRUD
8. View Order Information (product name, product price, quantity of product, total price) & delete

PART B

SYSTEM ARCHITECTURE AND LOGICAL DESIGN

B.1. START POINT

In the folder `inventory_OOVP_FinalProject`, there is a file named `LoginAndSignUp.java`. When you run this project, it will directly navigate to the login page.

```
public class LoginAndSignUp {  
  
    public static void main(String[] args) {  
  
        Login loginFrame = new Login();  
        loginFrame.setVisible(true);  
        loginFrame.pack();  
        loginFrame.setLocationRelativeTo(null);  
    }  
}
```

B.2. LOGIN PAGE

On the login page, we use "role" to distinguish between users and admins. When logging in, if the role is "admin", the user will be directed to the AdminMain page. If the role is "user", the user will be directed to the ManageOrder page, which can also be referred to as the user page. During login, the system will match the entered data with the data in the database. If the data matches, the user will be able to access the system. However, if the data does not match, a notification saying "Incorrect Password or Email" and "Error" will appear.



```
if (notFound == 1 && Password.equals(passDb)) {  
    if ("admin".equalsIgnoreCase(role)) {  
        AdminMain adminFrame = new AdminMain();  
        adminFrame.setUser(fname);  
        adminFrame.setVisible(true);  
        adminFrame.pack();  
        adminFrame.setLocationRelativeTo(null);  
    } else {  
        ManageOrder manageOrderFrame = new ManageOrder();  
        manageOrderFrame.setVisible(true);  
        manageOrderFrame.pack();  
        manageOrderFrame.setLocationRelativeTo(null);  
    }  
    this.dispose();  
} else {  
    JOptionPane.showMessageDialog(new JFrame(), "Incorrect email or password"  
        JOptionPane.ERROR_MESSAGE);  
    password.setText("");  
}
```

There is also a sign up button for users who do not have an account yet. When this button is clicked, the user will be directed to the Sign Up page.

```
private void jButton2ActionPerformed(java.awt.event  
  
    SignUp SignUpFrame = new SignUp();  
    SignUpFrame.setVisible(true);  
    SignUpFrame.pack();  
    SignUpFrame.setLocationRelativeTo(null);  
    this.dispose();  
}
```

B.3. SIGN UP PAGE

On the sign up page, there is a form that must be filled out to register an account. Users who want to register are required to enter their name, email, and password. The email field must be filled with an address ending in "@gmail.com" and the email must not have been registered before, as the system will check the email against the database. If the email is already registered, a notification will appear stating that the account already exists. When it successfully creates an account that means the data already exists in the user table.

```
name = fname.getText();  
email = emailAddress.getText();  
Password = pass.getText();  
  
String emailRegex = "^[^@\\s]+@[^@\\s]+\\.([^@\\s]+(\\.[^@\\s]+)?)$";  
if (!email.matches(emailRegex)) {  
    JOptionPane.showMessageDialog(new JFrame(), "Invalid email format!", "Error", JOptionPane.ERROR_MESSAGE);  
    return;  
}  
  
String checkQuery = "SELECT * FROM user WHERE email = '" + email + "'";  
ResultSet rs = st.executeQuery(checkQuery);  
if (rs.next()) {  
    JOptionPane.showMessageDialog(new JFrame(), "Email already registered!", "Error", JOptionPane.ERROR_MESSAGE);  
    return;  
}  
  
query = "INSERT INTO user(name, email, password)" +  
        "VALUES ('" + name + "', '" + email + "', '" + Password + "')";  
  
st.execute(query);  
fname.setText("");  
emailAddress.setText("");  
pass.setText("");  
showMessageDialog(null, "New account has been created successfully!");  
  
} catch (Exception e) {  
    System.out.println("Error!" + e.getMessage());  
}
```

B.4. MANAGE ORDER (USER PAGE)

Order

Products

ID	Name	Price	Stock	Description	Category ID	Category ...
1	smartwatch	10000.0	9	pretty, cool	1	elect
2	headphone	2000.0	10	loud	1	elect

Selected Prodeuct:

Product Name

Product Price

Product Description

Order Quantity

Add to Cart

Cart

Product Id	Name	Quantity	Price	Description	Sub Total
------------	------	----------	-------	-------------	-----------

Total Amount: 00000

Save Order Details

Reset

Close

Image B.1.1: The Final Design for Order Form

This class has the title Order in its form which is designed to facilitate users in the process of ordering products in an e-commerce application. It allows users to view available products, add selected products to the shopping cart and save order details to the database.

```
public class ManageOrder extends javax.swing.JFrame {  
  
    private int productPk = 0;  
    private int finalTotalPrice = 0;  
    private String orderId = "";  
    private Connection con;  
  
    public ManageOrder() {  
        initComponents();  
        setLocationRelativeTo(null);  
        connect();  
    }  
}
```

The user interface design was created using java swing components, which provide a simple and intuitive way for users to interact with the application. The main components used are JLabel, JTextField, JTable, and JButton which are organized using GroupLayout.

```
private javax.swing.JButton btnAddToCart;
private javax.swing.JButton btnBackToLogin;
private javax.swing.JButton btnReset;
private javax.swing.JButton btnSaveOrderDetails;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel10;
private javax.swing.JLabel jLabel11;
private javax.swing.JLabel jLabel12;
private javax.swing.JLabel jLabel13;
private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel8;
private javax.swing.JLabel jLabel9;
private javax.swing.JScrollPane jScrollPane2;
private javax.swing.JScrollPane jScrollPane3;
private javax.swing.JLabel lblFinalTotalPrice;
private javax.swing.JTable tableCart;
private javax.swing.JTable tableProduct;
private javax.swing.JTextField txtOrderQuantity;
private javax.swing.JTextField txtProductDescription;
private javax.swing.JTextField txtProductName;
private javax.swing.JTextField txtProductPrice;
```

We use the `connect` method to establish a connection to the database, allowing the application to execute SQL queries to retrieve product data and store booking details. The connection parameters (URL, username, and password) are specified in the code, ensuring that the application can access the database.

```
private void connect() {
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        String url = "jdbc:mysql://localhost:3306/oovpproject";
        String username = "root";
        String password = "";
        con = DriverManager.getConnection(url, username, password);
    } catch (Exception e) {
        JOptionPane.showMessageDialog(null, "Connection failed: " + e.getMessage());
    }
}
```

Then, how can the application retrieve data from the database? The application can retrieve data from the database using an SQL query that joins the `product` and `category`

tables. In the result it displays the 'tableproduct' JTable on the form. the user can select the product by clicking to view the details which are then filled automatically in the corresponding text fields. Which is where the text fields can only be viewed and leaves the 'quantity' to be edited so that the user can put the desired product in the shopping cart.

```
private void formComponentShown(java.awt.event.ComponentEvent evt) {
    // TODO add your handling code here:
    txtProductName.setEditable(false);
    txtProductPrice.setEditable(false);
    txtProductDescription.setEditable(false);

    DefaultTableModel productModel = (DefaultTableModel) tableProduct.getModel ();

    try{
        Statement st = con.createStatement();
        ResultSet rs = st.executeQuery(
            "SELECT product.id, product.name, product.price, product.stock, product.description, " +
            "category.id AS category_id, category.name AS category_name " +
            "FROM product " +
            "INNER JOIN category ON product.category_id = category.id " +
            "WHERE product.stock > 0"
        );
        while (rs.next()) {
            productModel.addRow(new Object[]{
                rs.getString("id"),
                rs.getString("name"),
                rs.getString("price"),
                rs.getString("stock"),
                rs.getString("description"),
                rs.getString("category_id"),
                rs.getString("category_name")
            });
        }
    }
    catch(Exception e){
        JOptionPane.showMessageDialog(null, e);
    }
}
```

Implementation of Create, Read and Delete Process	
	<p>The read process is implemented to fetch product data form the database and display it to the user when the form is shown. This enables users to view available products with their details before ordering.</p> <p>On form display ('formComponentShown'), a SQL query is executed to retrieve all products with stock greater than zero, joined with their categories. The result set populates the 'tableProduct' JTable. Then users can select from the list for further actions.</p>

*Selecting a product populates its details in text fields :


```
private void tableProductMouseClicked(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    int index = tableProduct.getSelectedRow();
    TableModel model = tableProduct.getModel();
    String id = model.getValueAt(index, 0).toString();
    productPk = Integer.parseInt(id);

    String productName = model.getValueAt(index, 1).toString();
    txtProductName.setText(productName);

    String productPrice = model.getValueAt(index, 2).toString();
    txtProductPrice.setText(productPrice);

    String productDescription = model.getValueAt(index, 4).toString();
    txtProductDescription.setText(productDescription);
}
```

The user can only add the selected product to the shopping cart after filling in the quantity on the form. without leaving it blank, or an error will appear and the order will not go into the user's shopping cart. But before it is put into the shopping cart, the application system checks the stock of the product first to avoid users ordering more products than the number of products available. The cart displays a JTable `tableCart`, and users can delete items from the cart if needed.

*Adding selected product to cart with quantity check and stock validation:

```
private void btnAddToCartActionPerformed(java.awt.event.ActionEvent evt) {
    String noOfUnits = txtOrderQuantity.getText();

    if (noOfUnits.isEmpty() || Integer.parseInt(noOfUnits) <= 0) {
        JOptionPane.showMessageDialog(null, "Quantity must be greater than 0 and cannot be empty.");
        return;
    }

    String productName = txtProductName.getText();
    String productDescription = txtProductDescription.getText();
    String productPrice = txtProductPrice.getText();

    try {
        if (productPrice != null && !productPrice.isEmpty() && !noOfUnits.isEmpty()) {
            double price = Double.parseDouble(productPrice);
            double units = Double.parseDouble(noOfUnits);
            double totalPrice = price * units;

            int checkStock = 0;
            int checkProductAlreadyExistInCart = 0;

            try {
                Statement st = con.createStatement();
                ResultSet rs = st.executeQuery("select * from product where id = " + productPk);
                while (rs.next()) {
                    if (rs.getInt("stock") >= units) {
                        checkStock = 1;
                    } else {
                        JOptionPane.showMessageDialog(null, "Product is out of stock. Only " +
                            rs.getInt("stock") + " left");
                    }
                }
            }
        }
    }
}
```

```

    } catch (Exception e) {
        JOptionPane.showMessageDialog(null, e);
    }

    if (checkStock == 1) {
        DefaultTableModel model = (DefaultTableModel) tableCart.getModel();
        if (tableCart.getRowCount() != 0) {
            for (int i = 0; i < tableCart.getRowCount(); i++) {
                if (Integer.parseInt(model.getValueAt(i, 0).toString()) == productPk) {
                    checkProductAlreadyExistInCart = 1;
                    JOptionPane.showMessageDialog(null, "Product already exist in cart");
                }
            }
        }

        if (checkProductAlreadyExistInCart == 0) {
            model.addRow(new Object[] { productPk, productName, noOfUnits, productPrice,
                productDescription, totalPrice });
            finalTotalPrice += totalPrice;
            lblFinalTotalPrice.setText(String.valueOf(finalTotalPrice));
            JOptionPane.showMessageDialog(null, "Added Successfully");
        }

        clearProductFields();
    } else {
        JOptionPane.showMessageDialog(null, "No of quantity and product field is required");
    }
} catch (NumberFormatException e) {
    JOptionPane.showMessageDialog(null, "Invalid price or quantity format");
}
}

```

*Removing product from cart updates total price accordingly :

```

private void tableCartMouseClicked(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    int index = tableCart.getSelectedRow();
    int a = JOptionPane.showConfirmDialog(null, "Do you want to remove this product",
        "select", JOptionPane.YES_NO_OPTION);

    if (a == 0) {
        TableModel model = tableCart.getModel();
        double subTotal = Double.parseDouble(model.getValueAt(index, 5).toString());
        finalTotalPrice -= subTotal;
        lblFinalTotalPrice.setText(String.valueOf(finalTotalPrice));

        DefaultTableModel cartModel = (DefaultTableModel) tableCart.getModel();
        cartModel.removeRow(index);
    }
}

```

Implementation of Create, Read and **Delete** Process

A delete process that allows users to remove products from the cart before completing an order, this is done by clicking on an item in the cart table and confirming its removal.

When the user clicks an entry in `tableCart`, a confirmation dialog asks if they want to remove the item. If the user confirms, the item is removed from the cart table. And the total price (`finalTotalPrice`) is adjusted accordingly.

When the user clicks the “Save Order Details” button, the application saves the order details to the `purchases` table in the database. It also updates the stock of the products

purchased. The application handles potential SQL exceptions and provides feedback to the user regarding the success or failure of the operation.

```
private void btnSaveOrderDetailsActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    DefaultTableModel model = (DefaultTableModel) tableCart.getModel();  
    int rowCount = model.getRowCount();  
  
    if (rowCount == 0) {  
        JOptionPane.showMessageDialog(this, "Cart is still empty.");  
        return;  
    }  
  
    try {  
        Connection conn = DriverManager.getConnection(  
            "jdbc:mysql://localhost:3306/oovpproject", "root", ""  
        );  
  
        String sql = "INSERT INTO purchase (productId, name, quantity, "  
            + "price, description, subtotal) " +  
            "VALUES (?, ?, ?, ?, ?, ?)";  
        PreparedStatement stmt = conn.prepareStatement(sql);  
  
        for (int i = 0; i < rowCount; i++) {  
            int productId = Integer.parseInt(model.getValueAt(i, 0).toString());  
            String product = model.getValueAt(i, 1).toString();  
            String quantityStr = model.getValueAt(i, 2).toString();  
            String priceStr = model.getValueAt(i, 3).toString();  
            String description = model.getValueAt(i, 4).toString();  
            String subtotalStr = model.getValueAt(i, 5).toString();  
  
            double quantity = Double.parseDouble(quantityStr);  
            double price = Double.parseDouble(priceStr);  
            double subtotal = Double.parseDouble(subtotalStr);  
  
            stmt.setInt(1, productId);  
            stmt.setString(2, product);  
            stmt.setDouble(3, quantity);  
            stmt.setDouble(4, price);  
            stmt.setString(5, description);  
            stmt.setDouble(6, subtotal);  
  
            stmt.executeUpdate();  
  
            String updateStockSql = "UPDATE product SET stock = stock - ? "  
                + "WHERE id = ?";  
            PreparedStatement updateStockStmt = conn.prepareStatement(updateStockSql);  
            updateStockStmt.setDouble(1, quantity);  
            updateStockStmt.setInt(2, productId);  
            updateStockStmt.executeUpdate();  
            updateStockStmt.close();  
        }  
  
        JOptionPane.showMessageDialog(this, "The order was successfully saved!");  
  
        model.setRowCount(0);  
        finalTotalPrice = 0;  
        lblFinalTotalPrice.setText("00000");  
  
        stmt.close();  
        conn.close();  
    } catch (SQLException e) {  
        e.printStackTrace();  
        JOptionPane.showMessageDialog(this, "An error occurred while saving "  
            + "to the database.");  
    }  
}
```

Implementation of Create , Read and Delete	
	<p>This code also shows the Create process through the feature of adding new order details into the database when the user confirms the purchase. This is implemented in the method that handles the "Save Order Details" button click.</p> <p>This process iterates over each item in the shopping cart table. In this method, each item inserts a new record into the purchase table. After inserting, it updates the product stock to reduce the amount purchased. and provides feedback to the user if the operation succeeds or fails.</p>

The application includes error handling mechanisms to ensure a smooth user experience. For example, it checks for empty fields, invalid input formats, and stock availability before allowing actions like adding to the cart or saving orders. Appropriate messages are displayed to inform the users of any issues.

```

if (noOfUnits.isEmpty() || Integer.parseInt(noOfUnits) <= 0) {
    JOptionPane.showMessageDialog(null, "Quantity must be greater than 0 and cannot be empty.");
    return;
}

```

B.5. ADMIN MENU PAGE

When logging in with the "admin" role, the user will be directed straight to the admin menu. There, the admin can click on the User menu to go directly to the ManageUser page, the Category menu to go to the AdminCategory page, the Product menu to go to the AdminProduct page, and the Order menu to go to the ViewOrder page. On the far right, there is also a log out button which, when clicked, will redirect the user back to the login page.

```

private void btnCategoryActionPerformed(java.awt.event.ActionEvent evt) {
    AdminCategory categoryPanel = new AdminCategory();
    categoryPanel.setSize(contentPanel.getSize());
    categoryPanel.setVisible(true);

    contentPanel.removeAll();
    contentPanel.add(categoryPanel);
    contentPanel.revalidate();
    contentPanel.repaint();
}

private void btnUserActionPerformed(java.awt.event.ActionEvent evt) {
    ManageUser userPanel = new ManageUser();
    userPanel.setSize(contentPanel.getSize());
    userPanel.setVisible(true);

    contentPanel.removeAll();
    contentPanel.add(userPanel);
    contentPanel.revalidate();
    contentPanel.repaint();
}

private void btnProductActionPerformed(java.awt.event.ActionEvent evt) {
    AdminProduct productPanel = new AdminProduct();
    productPanel.setSize(contentPanel.getSize()); // Agar ukuran penuh
    productPanel.setVisible(true);

    contentPanel.removeAll();
    contentPanel.add(productPanel);
    contentPanel.revalidate();
    contentPanel.repaint();
}

private void btnOrderActionPerformed(java.awt.event.ActionEvent evt) {
    ViewOrder orderPanel = new ViewOrder();
    orderPanel.setSize(contentPanel.getSize()); // Agar ukuran penuh
    orderPanel.setVisible(true);

    contentPanel.removeAll();
    contentPanel.add(orderPanel);
    contentPanel.revalidate();
    contentPanel.repaint();
}

private void btnLogoutActionPerformed(java.awt.event.ActionEvent evt) {
    int a = JOptionPane.showConfirmDialog(null, "Do you want to logout", "Select", JOptionPane.YES_NO_OPTION);
    if (a == 0) {
        setVisible(false);
        new Login().setVisible(true);
    }
}

```

B.6. USER CONTROL PANEL PAGE (ADMIN)

On the Manage User page, there is a table displaying user data, including name, email, and password, which is retrieved directly from the database. This page provides CRUD (Create, Read, Update, Delete) functionalities, represented by the Save, Update, and

```

private void displayUsers() {
    try {
        connect();
        String sql = "SELECT name, email, role, password FROM user";
        prep = conn.prepareStatement(sql);
        res = prep.executeQuery();
        userTable.setModel(DbUtils.resultSetToTableModel(res));
    } catch (Exception e) {
        JOptionPane.showMessageDialog(this, "Error " + e.getMessage());
    }
}

```

Delete buttons. Admins accessing this page can add new accounts, update existing account information, or delete accounts when necessary.

The Read feature is implemented through a table view that displays all user data stored in the database. Whenever the admin performs actions such as saving, updating, or deleting data, the system will automatically update the information in the database.

```
private void addbtnActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        connect();
        String sql = "INSERT INTO user (name, email, role, password) VALUES (?, ?, ?, ?)";
        prep = conn.prepareStatement(sql);
        prep.setString(1, jname.getText());
        prep.setString(2, jemail.getText());
        prep.setString(3, jrole.getSelectedItem().toString());
        prep.setString(4, jpass.getText());
        prep.executeUpdate();
        displayUsers();
        clear();
        JOptionPane.showMessageDialog(this, "Data successfully added");
    } catch (Exception e) {
        JOptionPane.showMessageDialog(this, "Failed to add : " + e.getMessage());
    }
}
```

```
private void updatebtnActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        connect();
        String sql = "UPDATE user SET name=?, role=?, password=? WHERE email=?";
        prep = conn.prepareStatement(sql);
        prep.setString(1, jname.getText());
        prep.setString(2, jrole.getSelectedItem().toString());
        prep.setString(3, jpass.getText());
        prep.setString(4, jemail.getText());
        prep.executeUpdate();
        displayUsers();
        clear();
        JOptionPane.showMessageDialog(this, "Data successfully updated");
    }
}
```

```
private void deletebtnActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        connect();
        String sql = "DELETE FROM user WHERE email=?";
        prep = conn.prepareStatement(sql);
        prep.setString(1, jemail.getText());
        prep.executeUpdate();
        displayUsers();
        clear();
        JOptionPane.showMessageDialog(this, "Data successfully deleted");
    } catch (Exception e) {
        JOptionPane.showMessageDialog(this, "Failed to delete: " + e.getMessage());
    }
}
```

```
private void userTableMouseClicked(java.awt.event.MouseEvent evt) {
    int row = userTable.getSelectedRow();
    String email = userTable.getModel().getValueAt(row, 1).toString();

    try {
        connect();
        String sql = "SELECT * FROM user WHERE email = ?";
        prep = conn.prepareStatement(sql);
        prep.setString(1, email);
        res = prep.executeQuery();
        if (res.next()) {
            jname.setText(res.getString("name"));
            jemail.setText(res.getString("email"));
            jrole.setSelectedItem(res.getString("role"));
            jpass.setText(res.getString("password"));
        }
    }
}
```

B.7. CATEGORY PAGE (ADMIN)

On the AdminCategory page, the admin can also perform CRUD operations, such as adding a new category, viewing category data through a table populated from the category table, updating the category name, and deleting a category if necessary. The concept of the CRUD buttons on this page is the same as the one used on the ManageUser page, utilizing the Save, Update, and Delete buttons.

a. Create

The create function allows the admin to insert a new category name into the database. It checks whether the input is empty or already exists before inserting the data.

```

private void btnSaveActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        String name = textName.getText();

        if (name.isEmpty()) {
            JOptionPane.showMessageDialog(null, "Name cannot be empty!");
            return;
        }

        if (isCategoryExists(name)) {
            JOptionPane.showMessageDialog(null, "Category name already exists!");
            return;
        }

        String sql = "INSERT INTO category (name) VALUES (?)";
        prep = conn.prepareStatement(sql);
        prep.setString(1, name);
        prep.executeUpdate();
        JOptionPane.showMessageDialog(null, "Successfully saved!");
        display();
        clear();
    } catch (Exception e) {
        JOptionPane.showMessageDialog(null, "Unable to save" + e.getMessage());
    }
}

```

b. Read

The read function displays all categories in a table. When a row is clicked, the name is shown in the input field for editing or deletion.

```

private void tableCategoryMouseClicked(java.awt.event.MouseEvent evt) {
    int row = tableCategory.getSelectedRow();
    textName.setText(tableCategory.getValueAt(row, 1).toString());
}

```

c. Update

The update function modifies the selected category name. It ensures that the new name is not empty and not a duplicate before updating.

```

private void btnUpdateActionPerformed(java.awt.event.ActionEvent evt) {
    int row = tableCategory.getSelectedRow();
    if (row >= 0) {
        int id = Integer.parseInt(tableCategory.getValueAt(row, 0).toString());
        String name = textName.getText();

        if (name.isEmpty()) {
            JOptionPane.showMessageDialog(null, "Name cannot be empty!");
            return;
        }

        if (isCategoryExistsForUpdate(name, id)) {
            JOptionPane.showMessageDialog(null, "Category name already exists!");
            return;
        }

        try {
            String sql = "UPDATE category SET name = ? WHERE id = ?";
            prep = conn.prepareStatement(sql);
            prep.setString(1, name);
            prep.setInt(2, id);
            prep.executeUpdate();
            JOptionPane.showMessageDialog(null, "Successfully update!");
            display();
            clear();
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, "Unable to update" + e.getMessage());
        }
    }
}

```

d. Delete

The delete function removes the selected category from the database after confirmation from the admin.

```
private void btnDeleteActionPerformed(java.awt.event.ActionEvent evt) {  
    int row = tableCategory.getSelectedRow();  
    if (row >= 0) {  
        int id = Integer.parseInt(tableCategory.getValueAt(row, 0).toString());  
  
        try {  
            String sql = "DELETE FROM category WHERE id = ?";  
            prep = conn.prepareStatement(sql);  
            prep.setInt(1, id);  
            prep.executeUpdate();  
            JOptionPane.showMessageDialog(null, "Successfully deleted!");  
            display();  
            clear();  
        } catch (Exception e) {  
            JOptionPane.showMessageDialog(null, "Unable to delete" + e.getMessage());  
        }  
    }  
}
```

B.8. PRODUCT PAGE (ADMIN)

On the Product page, the admin can perform CRUD operations on product data adding new products, viewing the product list through a table, updating product information, and deleting products if needed. Each product is linked to a specific category, so when adding or updating a product, the admin can select the appropriate category from a predefined list.

```
private void display() {  
    try {  
        String sql = "SELECT p.id, p.name, p.description, p.stock, p.price, p.category_id, c.name AS category_name " +  
            "FROM product p JOIN category c ON p.category_id = c.id";  
        prep = conn.prepareStatement(sql);  
        res = prep.executeQuery();  
        tableProduct.setModel(DbUtils.resultSetToTableModel(res));  
    } catch (Exception e) {  
        JOptionPane.showMessageDialog(null, e);  
    }  
}
```

The displayed product data includes the product name, stock, description, price, and its associated category. This feature ensures that each product is properly categorized and

helps the admin manage product data more efficiently based on their categories. The Save, Update, and Delete buttons on this page follow the same concept as those on the ManageUser and AdminCategory pages.

a. Create

The create function allows the admin to insert a new product, including name, description stock, price into the database and select a category. It checks whether the input is empty or already exists before inserting the data.

```
private void btnSaveActionPerformed(java.awt.event.ActionEvent evt) {  
    String name = textName.getText();  
    String desc = textDesc.getText();  
    String stockStr = textStock.getText().trim();  
    String priceStr = textPrice.getText().trim();  
  
    if (name.isEmpty() || stockStr.isEmpty() || priceStr.isEmpty()) {  
        JOptionPane.showMessageDialog(null, "Please fill in all fields.");  
        return;  
    }  
  
    try {  
        int stock = Integer.parseInt(stockStr);  
        double price = Double.parseDouble(priceStr);  
  
        if (stock <= 0) {  
            JOptionPane.showMessageDialog(null, "Stock must be greater than 0.");  
            return;  
        }  
  
        if (price <= 0) {  
            JOptionPane.showMessageDialog(null, "Price must be greater than 0.");  
            return;  
        }  
    }  
  
    String categoryName = comboCategory.getSelectedItem().toString();  
  
    String sqlCategoryId = "SELECT id FROM category WHERE name = ?";  
    prep = conn.prepareStatement(sqlCategoryId);  
  
    prep.setString(1, categoryName);  
    res = prep.executeQuery();  
    res.next();  
    int categoryId = res.getInt("id");  
  
    String checkSql = "SELECT COUNT(*) FROM product WHERE name = ?";  
    prep = conn.prepareStatement(checkSql);  
    prep.setString(1, name);  
    res = prep.executeQuery();  
    res.next();  
    int count = res.getInt(1);  
  
    if (count > 0) {  
        JOptionPane.showMessageDialog(null, "A product with this name already exists.");  
        return;  
    }  
  
    String sql = "INSERT INTO product (name, description, stock, price, category_id) VALUES (?, ?, ?, ?, ?)";  
    prep = conn.prepareStatement(sql);  
    prep.setString(1, name);  
    prep.setString(2, desc);  
    prep.setInt(3, stock);  
    prep.setDouble(4, price);  
    prep.setInt(5, categoryId);  
    prep.executeUpdate();  
  
    JOptionPane.showMessageDialog(null, "Successfully saved the product!");  
    display();  
    clear();  
}
```

b. Read

The read function displays all products and their information in a table. When a row is clicked, the name is shown in the input field for editing or deletion.

```
private void tableProductMouseClicked(java.awt.event.MouseEvent evt) {  
    int row = tableProduct.getSelectedRow();  
    textName.setText(tableProduct.getValueAt(row, 1).toString());  
    textDesc.setText(tableProduct.getValueAt(row, 2).toString());  
    textStock.setText(tableProduct.getValueAt(row, 3).toString());  
    textPrice.setText(tableProduct.getValueAt(row, 4).toString());  
    comboCategory.setSelectedItem(tableProduct.getValueAt(row, 6).toString());  
}
```

c. Update

The update function modifies the selected product. It ensures that the new information is not empty and not a duplicate before updating.

```
private void btnUpdateActionPerformed(java.awt.event.ActionEvent evt) {  
    int row = tableProduct.getSelectedRow();  
    if (row == -1) {  
        JOptionPane.showMessageDialog(this, "Please select a product to update.");  
        return;  
    }  
  
    try {  
        String productId = tableProduct.getValueAt(row, 0).toString();  
        String name = textName.getText();  
        String desc = textDesc.getText();  
        int stock = Integer.parseInt(textStock.getText());  
        double price = Double.parseDouble(textPrice.getText());  
        String categoryName = comboCategory.getSelectedItem().toString();  
  
        String checkSql = "SELECT COUNT(*) FROM product WHERE name = ? AND id != ?";  
        prep = conn.prepareStatement(checkSql);  
        prep.setString(1, name);  
        prep.setString(2, productId);  
        res = prep.executeQuery();  
        res.next();  
        int count = res.getInt(1);  
  
        if (count > 0) {  
            JOptionPane.showMessageDialog(this, "A product with this name already exists.");  
            return;  
        }  
  
        String sqlCategoryId = "SELECT id FROM category WHERE name = ?";  
  
        prep = conn.prepareStatement(sqlCategoryId);  
        prep.setString(1, categoryName);  
        res = prep.executeQuery();  
        res.next();  
        int categoryId = res.getInt("id");  
  
        String sql = "UPDATE product SET name = ?, description = ?, stock = ?, price = ?, category_id = ? WHERE id = ?";  
        prep = conn.prepareStatement(sql);  
        prep.setString(1, name);  
        prep.setString(2, desc);  
        prep.setInt(3, stock);  
        prep.setDouble(4, price);  
        prep.setInt(5, categoryId);  
        prep.setString(6, productId);  
        prep.executeUpdate();  
  
        JOptionPane.showMessageDialog(this, "Product updated successfully.");  
        display();  
        clear();  
    } catch (Exception e) {  
        JOptionPane.showMessageDialog(null, "Unable to save the product: " + e.getMessage());  
    }  
}
```

d. Delete

The delete function removes the selected product from the database after confirmation from the admin.

```
private void btnDeleteActionPerformed(java.awt.event.ActionEvent evt) {  
  
    int row = tableProduct.getSelectedRow();  
    if (row == -1) {  
        JOptionPane.showMessageDialog(this, "Please select a product to delete.");  
        return;  
    }  
  
    try {  
        String productId = tableProduct.getValueAt(row, 0).toString();  
        String sql = "DELETE FROM product WHERE id = ?";  
        prep = conn.prepareStatement(sql);  
        prep.setString(1, productId);  
        prep.executeUpdate();  
  
        JOptionPane.showMessageDialog(this, "Product deleted successfully.");  
        display();  
        clear();  
    } catch (Exception e) {  
        JOptionPane.showMessageDialog(null, "Unable to delete the product: " + e.getMessage());  
    }  
}
```

B.9. VIEW ORDER PAGE (ADMIN)

On the View Order page, the admin can view a list of transaction history or user orders. The displayed data includes User ID, Product ID, Product Name, Quantity, Unit Price, Total Price, and a Description of the ordered product.

```
private void display() {  
    try {  
        String sql = "SELECT id, name AS product_name, quantity, price, description, subtotal FROM purchase";  
        prep = conn.prepareStatement(sql);  
        res = prep.executeQuery();  
        tableOrder.setModel(DbUtils.resultSetToTableModel(res));  
    } catch (Exception e) {  
        JOptionPane.showMessageDialog(null, e);  
    }  
}
```

This page serves as a transaction history view that allows the admin to monitor the details of each incoming order. While the admin cannot add or update order data, a Delete button is available if certain entries need to be removed, such as due to input errors or user cancellation requests. The main purpose of this page is to serve as a monitoring center for completed transactions.

PART C

DATABASE IMPLEMENTATION

C.1. DATABASE

```
CREATE DATABASE oovpproject;
```

C.2. USER TABLE

```
CREATE TABLE user (  
    id INT NOT NULL AUTO_INCREMENT,  
    name VARCHAR(20) NOT NULL,  
    email VARCHAR(20) NOT NULL,  
    password VARCHAR(20) NOT NULL,  
    role VARCHAR(10) NOT NULL  
    DEFAULT 'user',  
    PRIMARY KEY (id)  
);
```

C.3. CATEGORY TABLE

```
CREATE TABLE category (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    name VARCHAR(100) NOT NULL  
);
```

C.4. PRODUCT TABLE

```
CREATE TABLE product (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    name VARCHAR(100) NOT NULL,  
    description TEXT,  
    stock INT,  
    price DOUBLE,  
    category_id INT,  
    FOREIGN KEY (category_id) REFERENCES category(id)  
);
```

C.5. PURCHASE TABLE

```
CREATE TABLE purchase (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    productId INT,  
    name VARCHAR(100) NOT NULL,  
    quantity INT,  
    price DOUBLE,  
    description TEXT,  
    subtotal DOUBLE,  
    FOREIGN KEY (productId) REFERENCES product(id)  
);
```

PART D

CONCLUSION

Inventory Management System is a project that we successfully completed using Java Swing as the interface and MySQL as the database, a simple and effective system designed to facilitate the management of products, purchase transactions, and user data in a shop or business. During development, we applied object-oriented programming principles to ensure the code is more structured and easy to understand. We also created a system that distinguishes user roles as "admin" and "user" to provide access according to their respective needs, making it more secure and efficient.

With the main features such as registration, login, product catalog, shopping cart, and purchase history that have been well implemented for both types of users. In particular, admins have different panels to manage user data, categories, products, and view transaction history. The interface is designed to be easy to use and not confusing, even for new users.

Through this project, we not only learned how to build an application technically, but also how to design a workflow and interface that is convenient for users. This project gave us real-life experience in developing an app that leaves room for improvement, which will be useful in building our skills in the future.