

INVENTORY MANAGEMENT SYSTEM

Version No.: 1.0

Date: 05/05/2023

Project Name: Inventory Management System

Presented by

Guruteja Kanderi — A20526883

Revision History

Version No.	Date	Prepared by / Modified by	Significant Changes
1.0	05/05/2023	Guruteja Kanderi	Initial project proposal.

WHO CAN REFER

This document can be referred by the following persons as a part of ITMD510 Object Oriented Application Development

- ✓ James Papademas
- ✓ Guruteja Kanderi

1. INTRODUCTION

Inventory Management System The Inventory Management System is a user-friendly JavaFX application that simplifies inventory management. It is designed with the Model-View-Controller (MVC) architecture, making it modular and easy to understand. The system provides flexibility for the admin to add, update, and delete users, and allows users to view updated product information and place orders. Additionally, the admin has full control over the inventory, with the ability, read, update, and delete products as needed. With these features, the Inventory Management System streamlines the inventory management process and improves overall efficiency.

Cynosure of INVENTORY MANAGEMENT SYSTEM

- ✓ Login and logout for each user.
- ✓ Update profile made easy as it fetches the records from the database in the text fields.
- ✓ Handled maximum exceptional handling and recorded all the exceptions in the database table to improve this application.
- ✓ Admin can view and update customer's details.
- ✓ Customer can order the products added by admin.

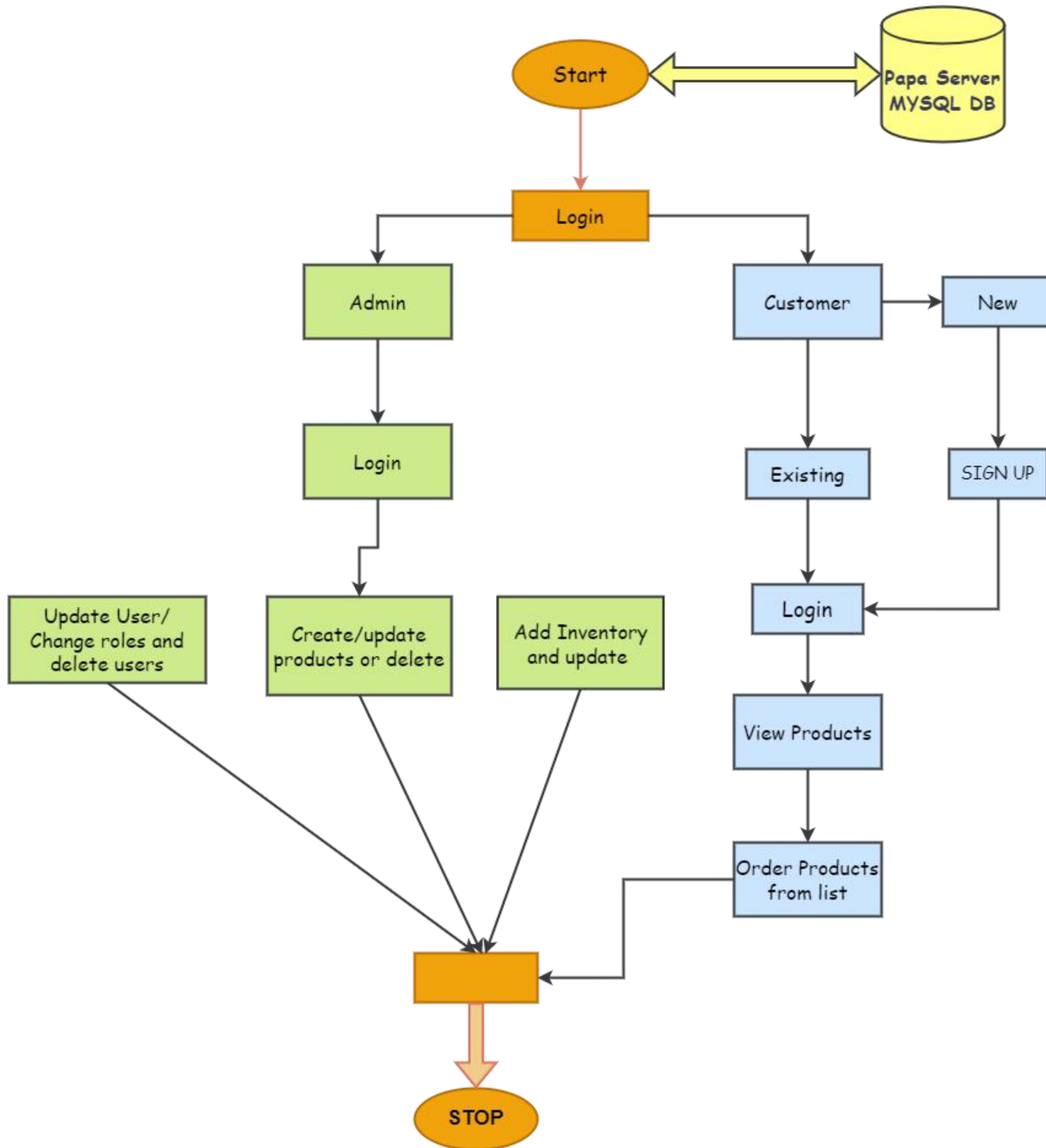
1.2. EXPECTED OUTCOMES

- ✓ Build a common platform for inventory management.
- ✓ Provide hassle-free application for the customers.
- ✓ Admin can view, add, and update the customer details.
- ✓ Admin can view, add, and update the product items.
- ✓ Admin can view and update the inventory items.
- ✓ Customers can create an account, using the sign up option and order products easily.

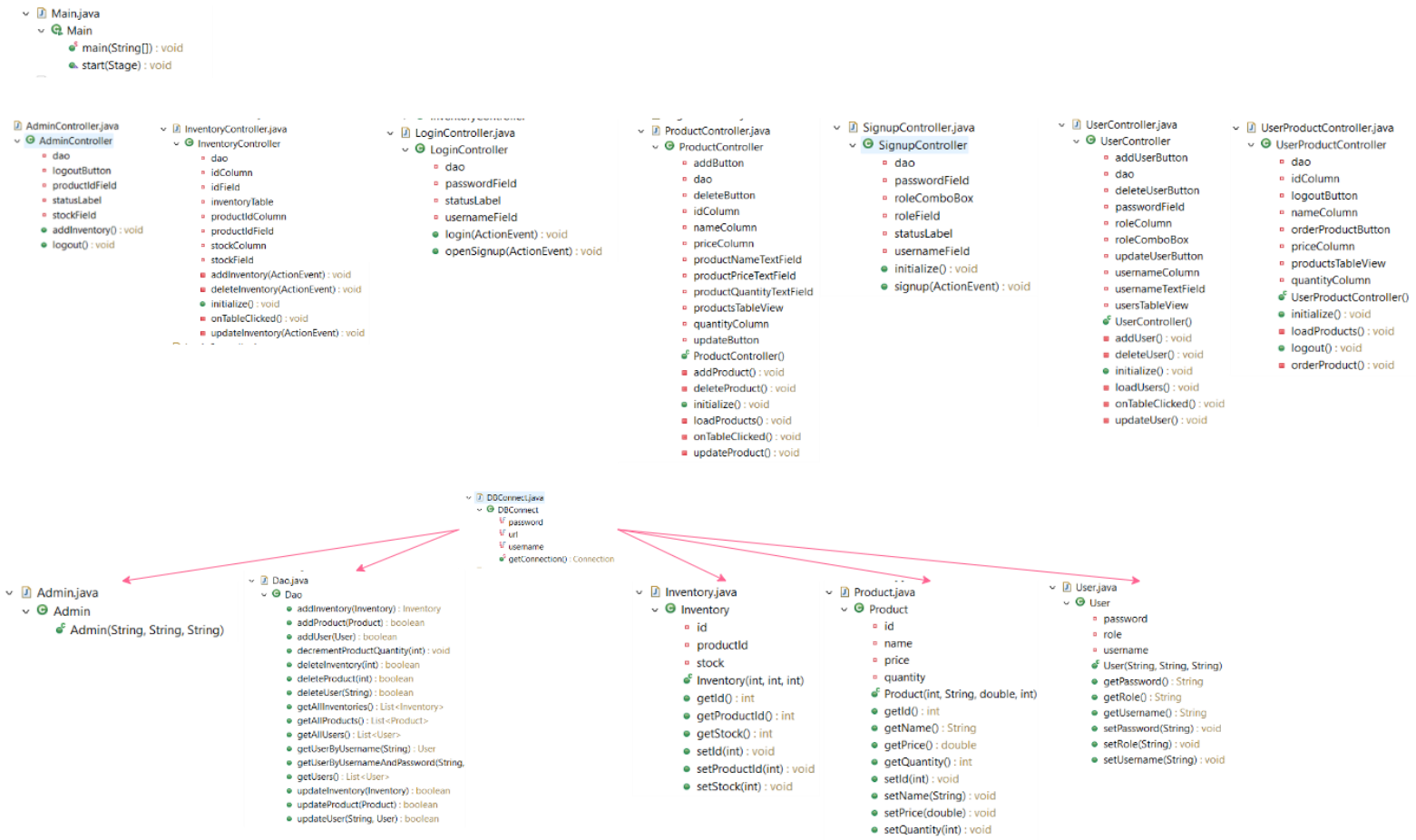
2. TECHNICAL DETAILS

2.1. APPLICATION ARCHITECTURE

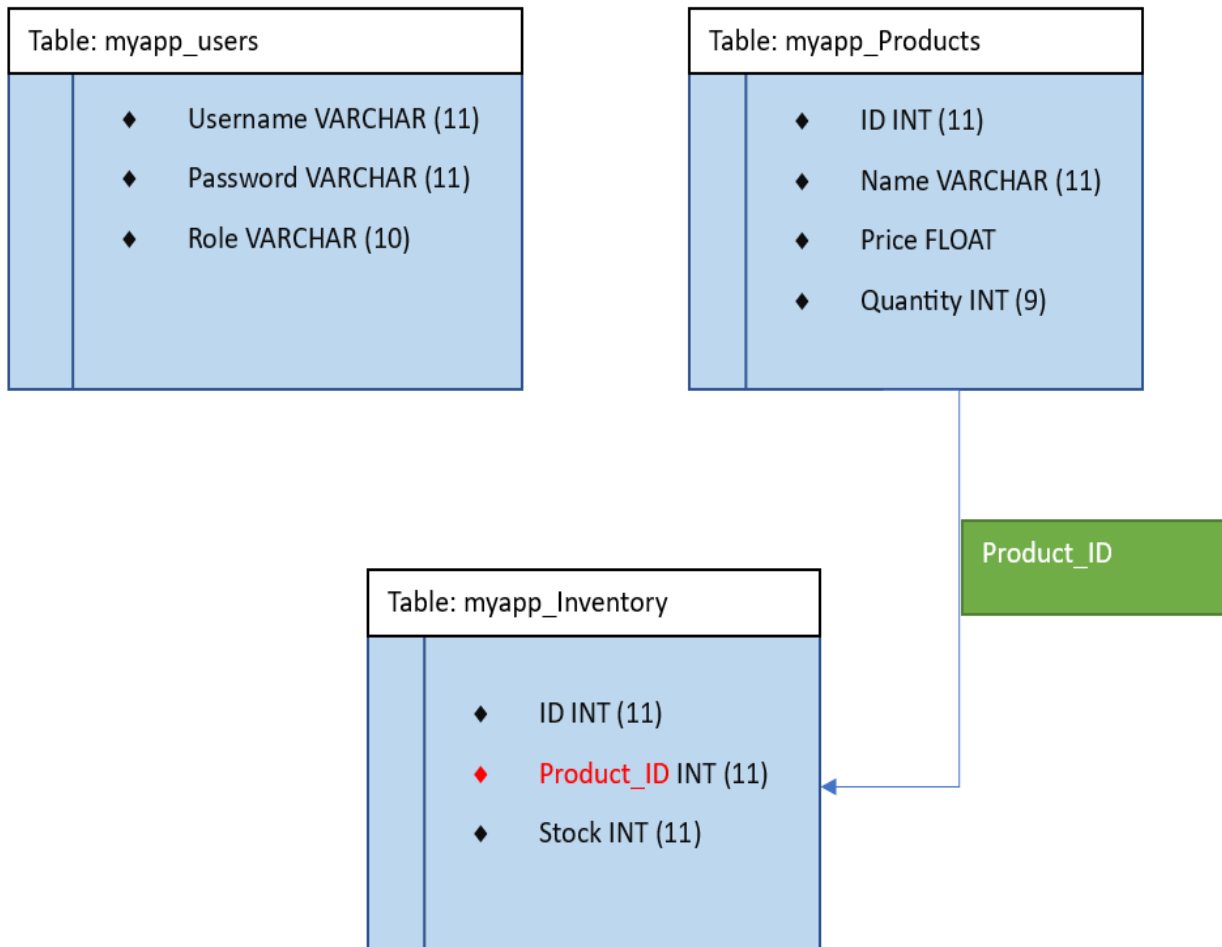
Below is the overview of the application and the navigation inside the application depending upon the category



2.2. UML



2.3. ERD



2.4. TECHNOLOGY USED

2.4.1. Front End

JavaFX: Software platform for creating and delivering desktop applications.

2.4.2. Back End

MySQL Developer: Integrated development environment for working with SQL in MySQL databases.

2.4.3. Technical Requirements

Below is the technology that was required to build this project

1. Java
2. JavaFX

3. MySQL Database

2.4.4. Domain Knowledge Requirements Inventory Management Systems

2.4.5. Tools Used

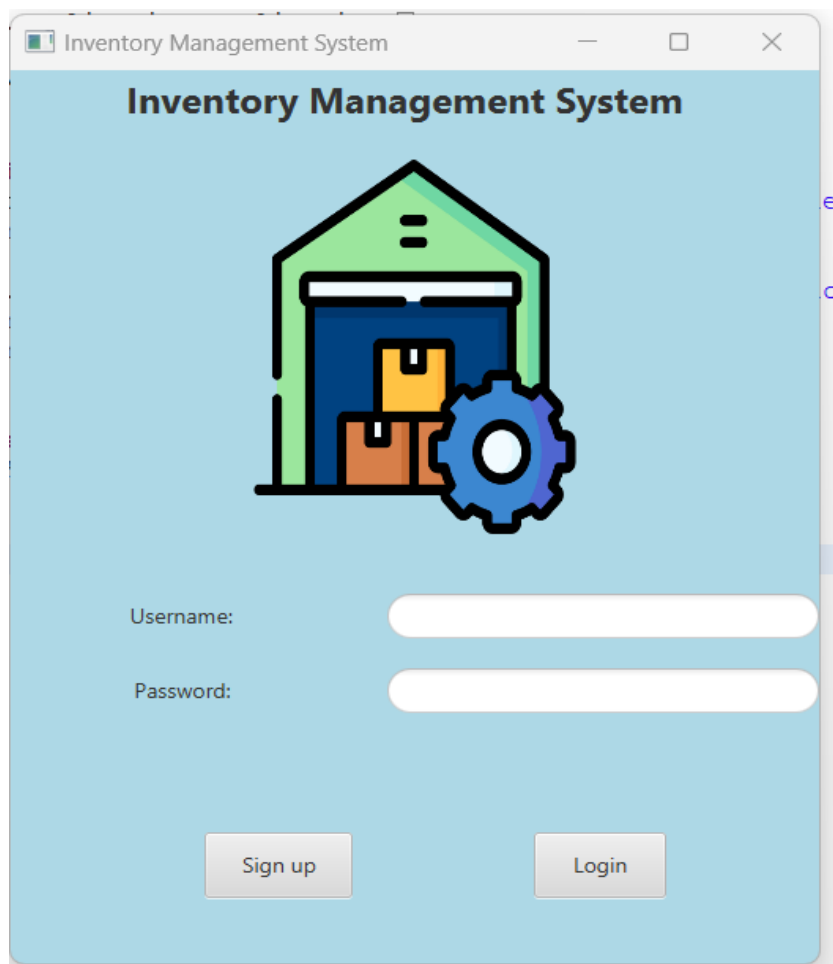
1. Java Eclipse
2. MYSQL Developer

3.HOME PAGE

3.1. Home page

Home Page is common for admin and customer. The page has Admin Login and Customer Login screens, along with that Sign up option is provided for new users.

3.2. Screenshot



4. LOGIN

4.1. Login Screen

4.1.1. Admin Login

When the administrator logs in he/she must enter their User Name in the User Name text box, and Password in the Password field.

If the username or password value is not entered and left as blank or if the values don't match with the admin username and password defined in the Java code, error message will be displayed on the bottom of the screen stating that the credentials are invalid.

Once the required fields are filled with valid entries, the admin will be navigated to a dashboard of admin operations.

S. No	Email-ID	Password	Description
1	admin	admin123	Successful Login
2	admin	admin1234	Credentials are invalid

4.1.2. Customer Login

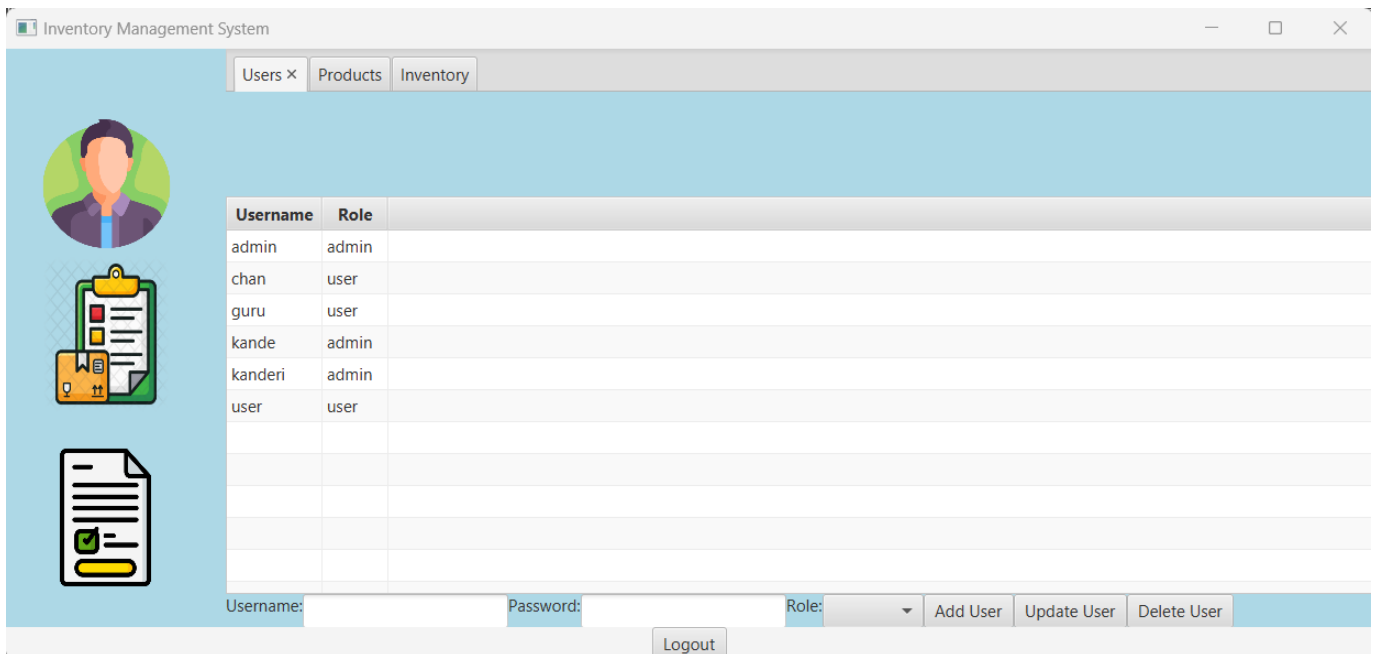
To access their account, the customer must input their Username and Password on the login page. The login fields are mandatory and must be filled in correctly to proceed. If the entered values do not match the credentials stored in the database, an error message will be displayed.

If the username or password value is not entered and left as blank or if the values don't match with credentials stored in the database, an error message will be displayed on the bottom of the screen stating that the credentials are invalid.

Once the required fields are filled with valid entries, the customer will be navigated to a customer dashboard.

S. No	Email-ID	Password	Description
1	chan	chan	Successful Login
2	chan	Chan123	Username or password is Invalid!

4.1.2.1. Admin Login page entry



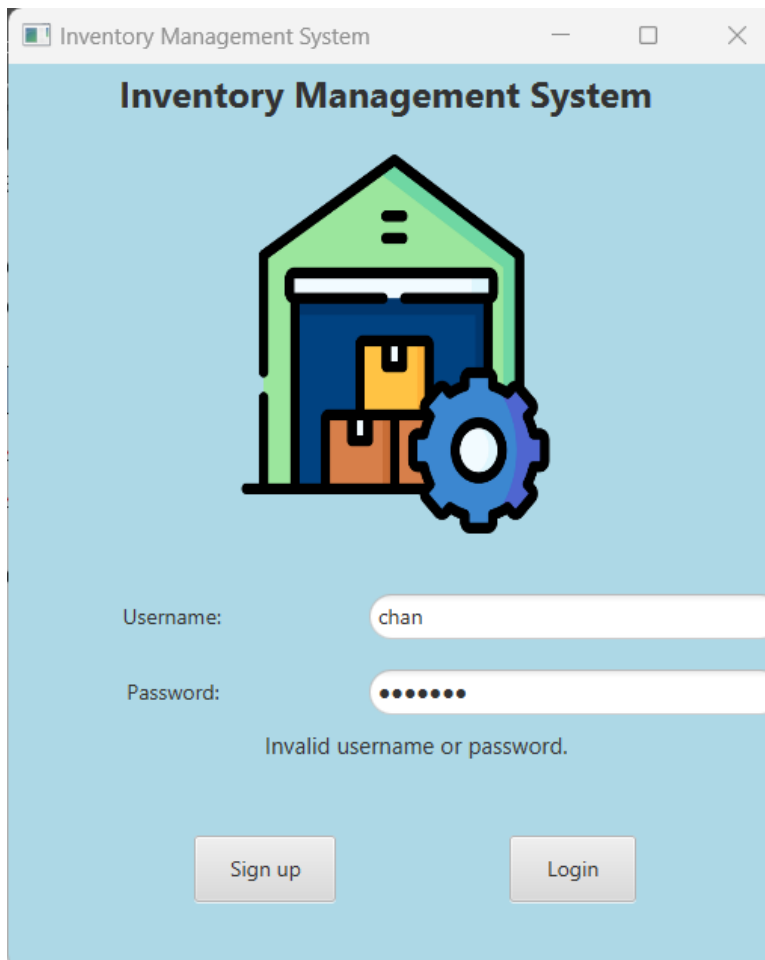
Inventory Management System

Users x Products Inventory

Username	Role
admin	admin
chan	user
guru	user
kande	admin
kanderi	admin
user	user

Username: Password: Role: Add User Update User Delete User Logout

4.1.2.2. Invalid credentials scenario



Inventory Management System

Inventory Management System

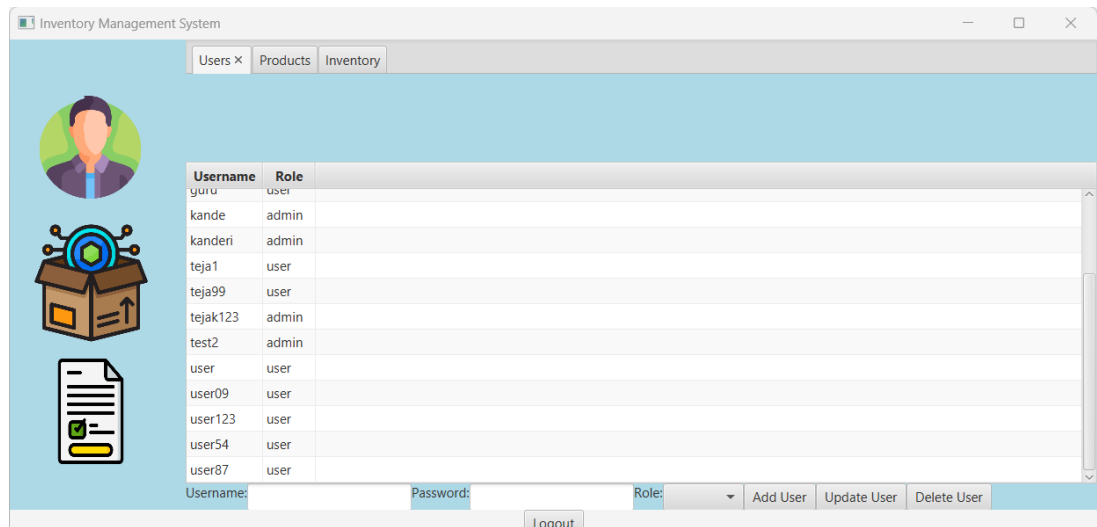
Username: chan

Password:

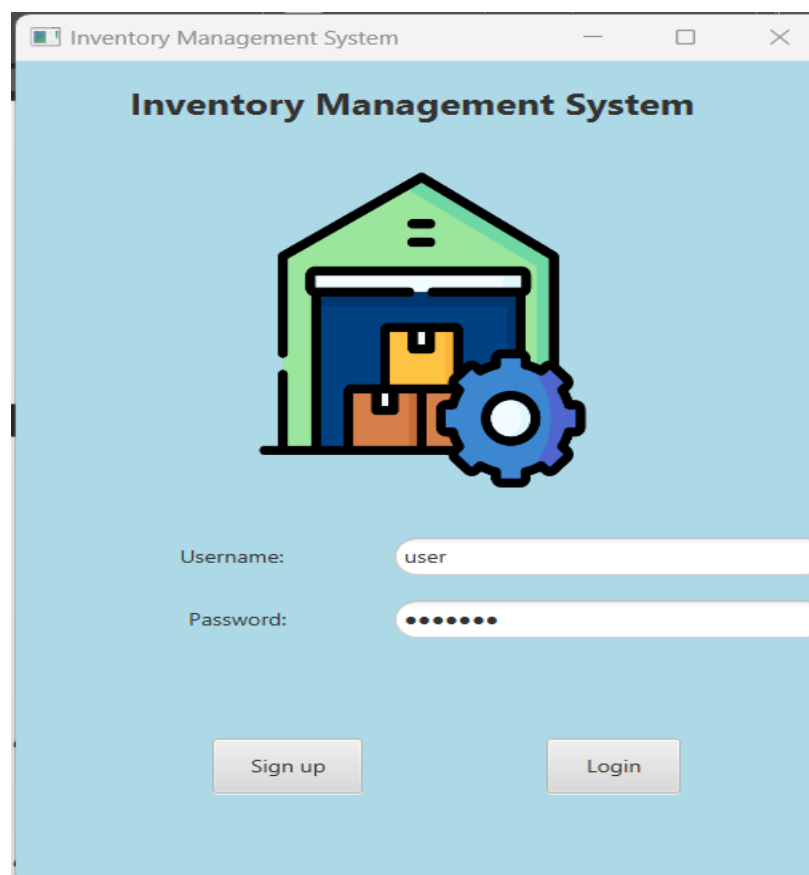
Invalid username or password.

Sign up Login

4.1.2.3. Successful login will navigate the administrator to the admin dashboard.



4.1.2.4. Customer login entry page



4.1.2.5. Upon entering valid credentials, the user will be navigated to the customer dashboard.


Inventory Management System			
ID	Name	Price	Quantity
1	Headphones	200.0	49
2	Bags	20.99	4
4	laptop	20.16	2
6	Shoes	10.9	8
7	Cables	5.2	100
8	Computer	2.5	1
9	Mobile Cases	250.0	1
10	Pen Drives	30.5	7
12	Mobiles	100.0	16
Order Product Logout			

5. CUSTOMER SIGN UP

5.1. Sign up screen

Inventory Management System

Sign Up



Username:

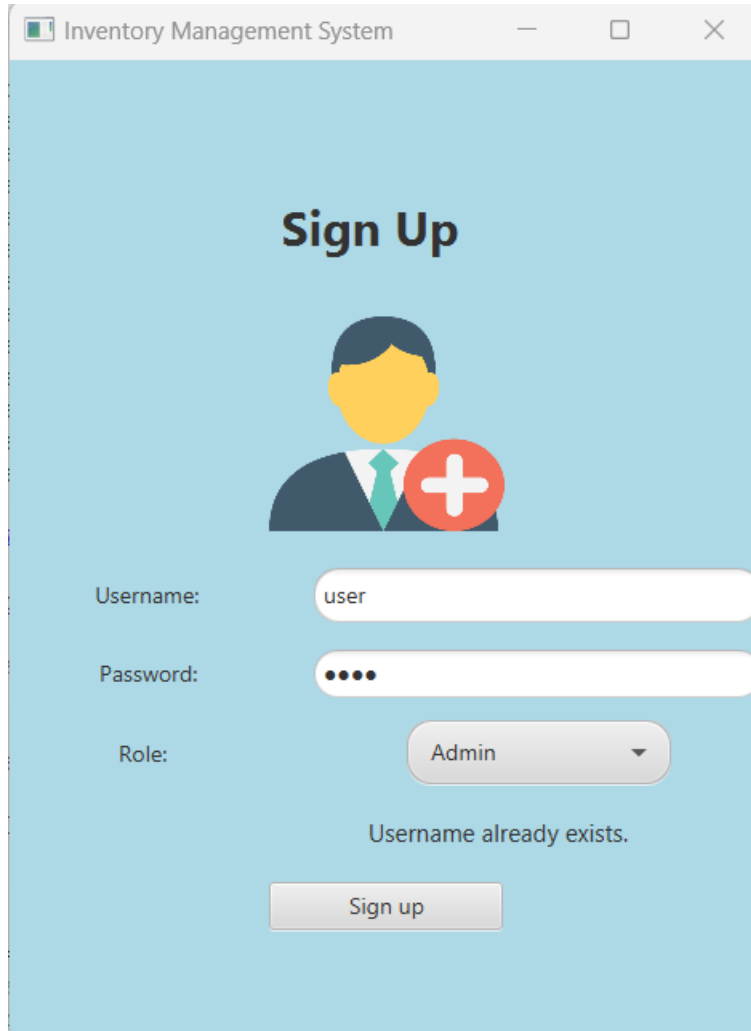
Password:

Role:

User

Sign up

5.2.1. Existing user error prompt:

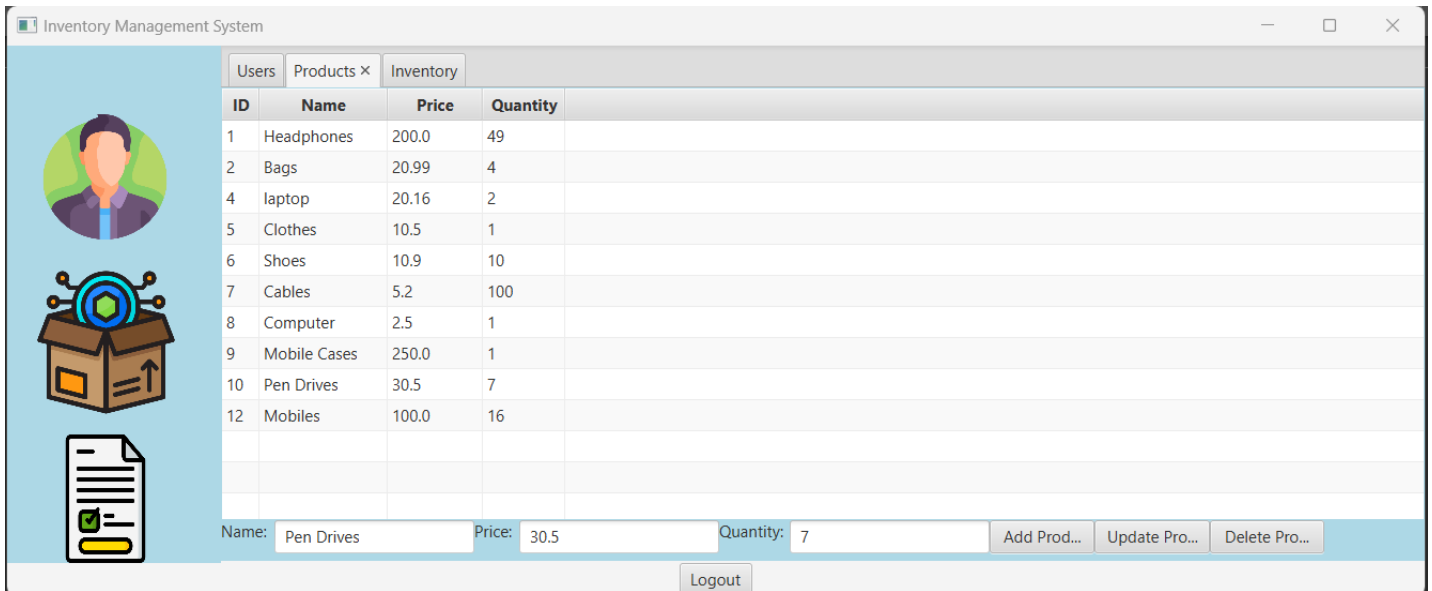


The screenshot shows a web browser window titled "Inventory Management System". The page has a light blue background and is titled "Sign Up" in large black text. Below the title is a stylized icon of a person with a red circle containing a white plus sign. The form contains three input fields: "Username:" with the value "user", "Password:" with four dots, and "Role:" with a dropdown menu showing "Admin". Below the form, the message "Username already exists." is displayed in red. At the bottom is a "Sign up" button.

6. PRODUCTS PAGE

6.1. Products Page:

This page displays all the products that can be updated by the administrator. Which will appear to the users/customers in the application.



The screenshot shows the 'Inventory' tab of the Inventory Management System. On the left is a sidebar with a user profile icon, a box with a gear icon, and a document icon. The main area contains a table with columns: ID, Name, Price, and Quantity. Below the table are input fields for Name, Price, and Quantity, followed by buttons for 'Add Prod...', 'Update Pro...', 'Delete Pro...', and 'Logout'.

ID	Name	Price	Quantity
1	Headphones	200.0	49
2	Bags	20.99	4
4	laptop	20.16	2
5	Clothes	10.5	1
6	Shoes	10.9	10
7	Cables	5.2	100
8	Computer	2.5	1
9	Mobile Cases	250.0	1
10	Pen Drives	30.5	7
12	Mobiles	100.0	16

Name: Pen Drives Price: 30.5 Quantity: 7

Buttons: Add Prod... Update Pro... Delete Pro... Logout

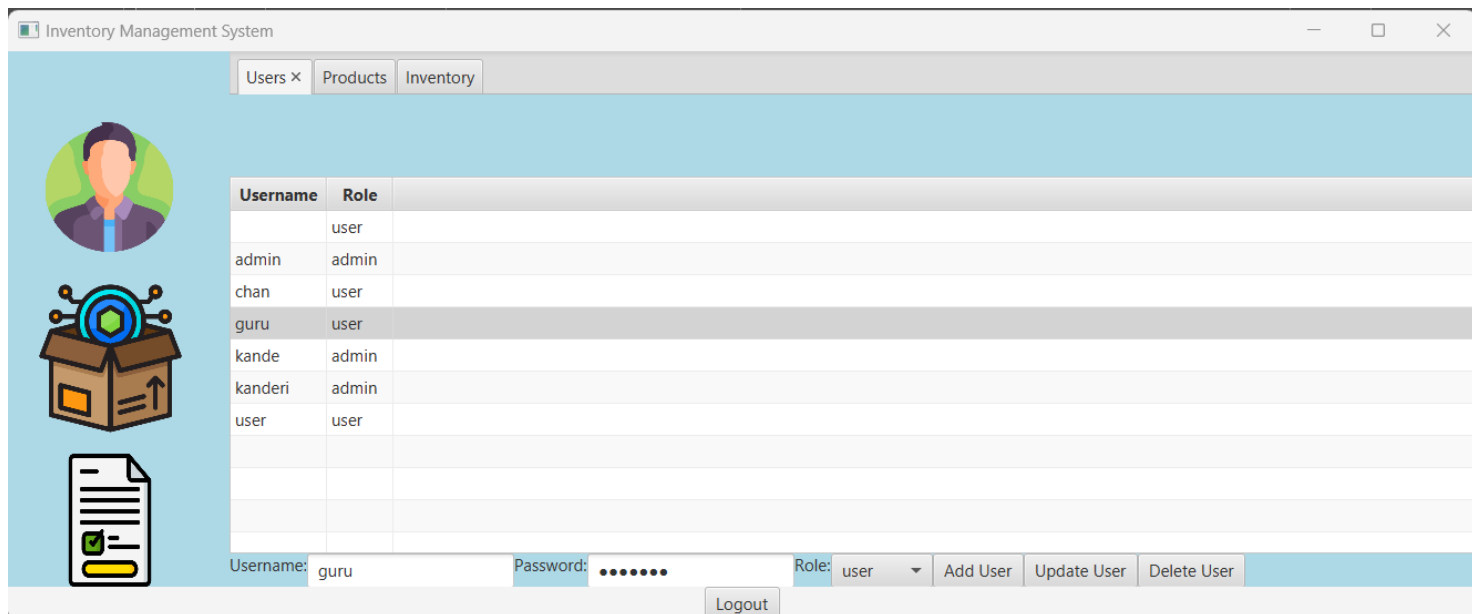
7. ADMIN OPERATIONS

7.1. Admin dashboard

The Admin dashboard displays options and description on the operations possible for the admin

7.2. Admin Users Operation:

The admin is capable of creating new users, updating the existing users and deleting the users whenever required.



The screenshot shows the 'Users' tab of the Inventory Management System. On the left is a sidebar with a user profile icon, a box with a gear icon, and a document icon. The main area contains a table with columns: Username and Role. Below the table are input fields for Username, Password, and Role, followed by buttons for 'Add User', 'Update User', 'Delete User', and 'Logout'.

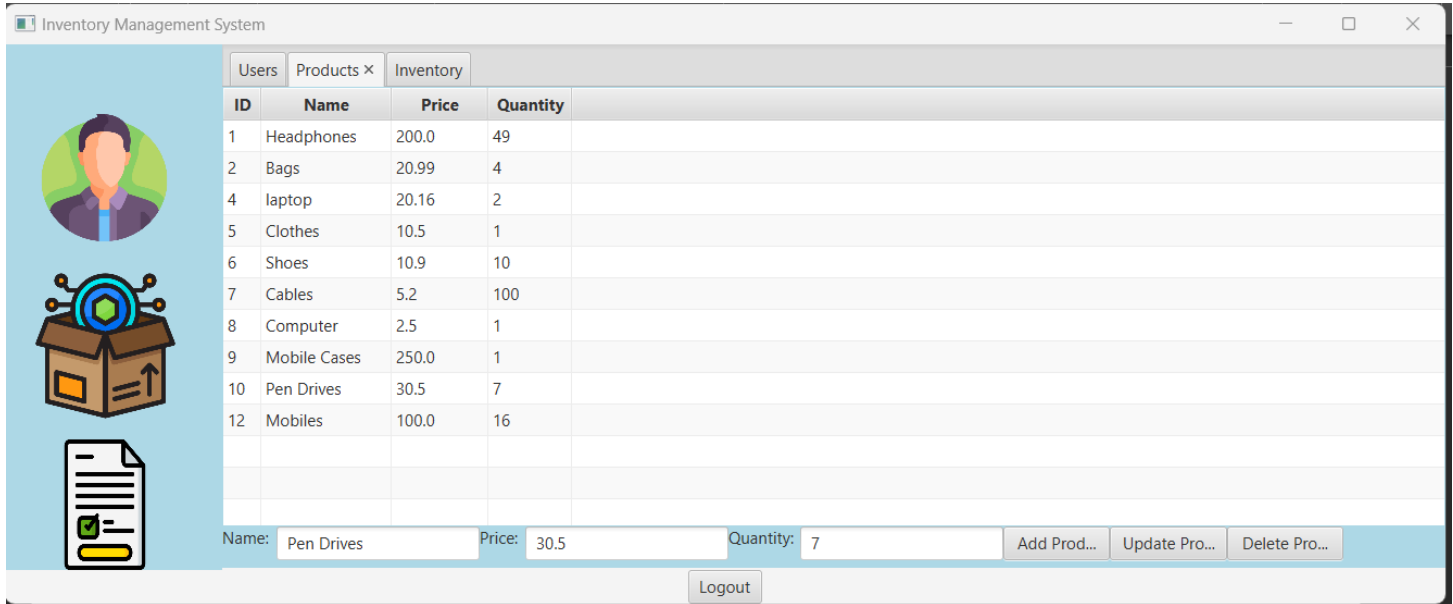
Username	Role
	user
admin	admin
chan	user
guru	user
kande	admin
kanderi	admin
user	user

Username: guru Password: Role: user

Buttons: Add User Update User Delete User Logout

7.4 Admin Products Operation:

The admin is solely responsible for creating the new products in the application and capable of updating the exiting products. Along with that user can delete the products whenever required.



The screenshot shows the 'Inventory Management System' window with the 'Products' tab selected. The interface includes a sidebar with a user profile icon, a box icon, and a document icon. The main area displays a table of products with columns: ID, Name, Price, and Quantity. Below the table, there are input fields for Name, Price, and Quantity, and buttons for 'Add Prod...', 'Update Pro...', 'Delete Pro...', and 'Logout'.

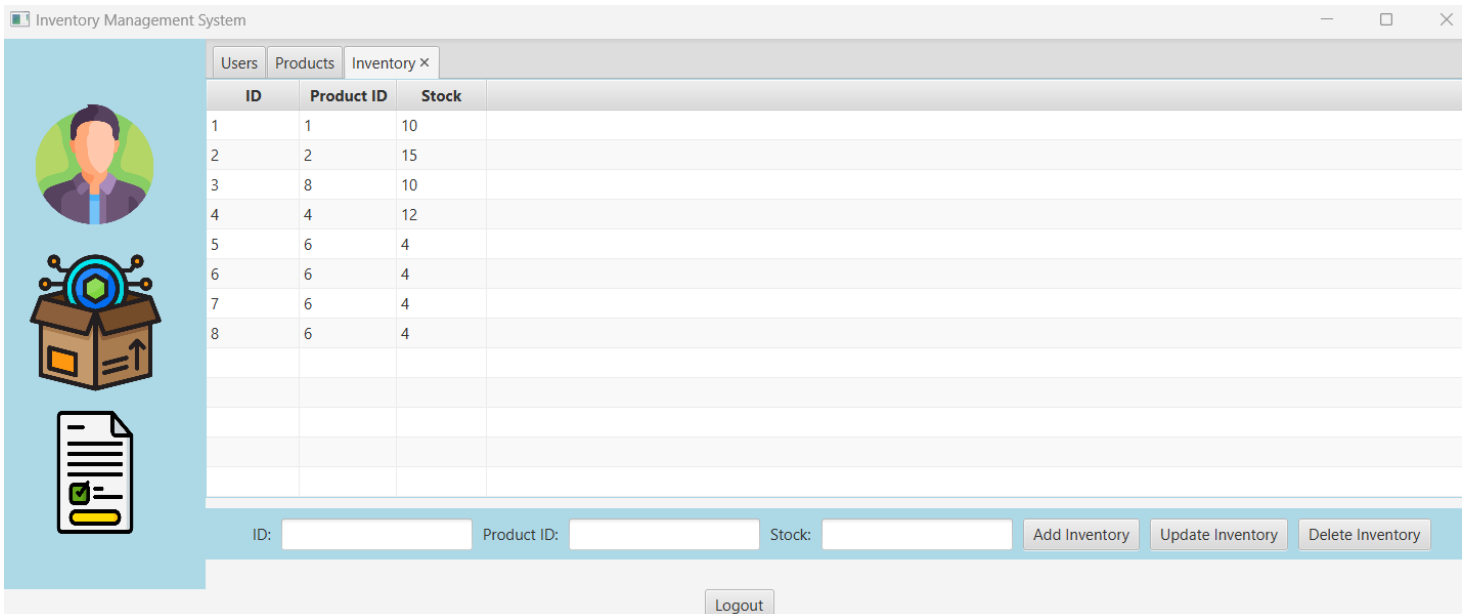
ID	Name	Price	Quantity
1	Headphones	200.0	49
2	Bags	20.99	4
4	laptop	20.16	2
5	Clothes	10.5	1
6	Shoes	10.9	10
7	Cables	5.2	100
8	Computer	2.5	1
9	Mobile Cases	250.0	1
10	Pen Drives	30.5	7
12	Mobiles	100.0	16

Name: Pen Drives Price: 30.5 Quantity: 7

Buttons: Add Prod... Update Pro... Delete Pro... Logout

7.5 Admin Inventory Operation:

Here the Product ID in the Products tab and Inventory is the same. And with the help of Product ID the admin can update the Inventory as per the stock availability.



The screenshot shows the 'Inventory Management System' window with the 'Inventory' tab selected. The interface includes a sidebar with a user profile icon, a box icon, and a document icon. The main area displays a table of inventory with columns: ID, Product ID, and Stock. Below the table, there are input fields for ID, Product ID, and Stock, and buttons for 'Add Inventory', 'Update Inventory', 'Delete Inventory', and 'Logout'.

ID	Product ID	Stock
1	1	10
2	2	15
3	8	10
4	4	12
5	6	4
6	6	4
7	6	4
8	6	4

ID: Product ID: Stock:

Buttons: Add Inventory Update Inventory Delete Inventory Logout

Add: The admin can add products, inventory, users real time.

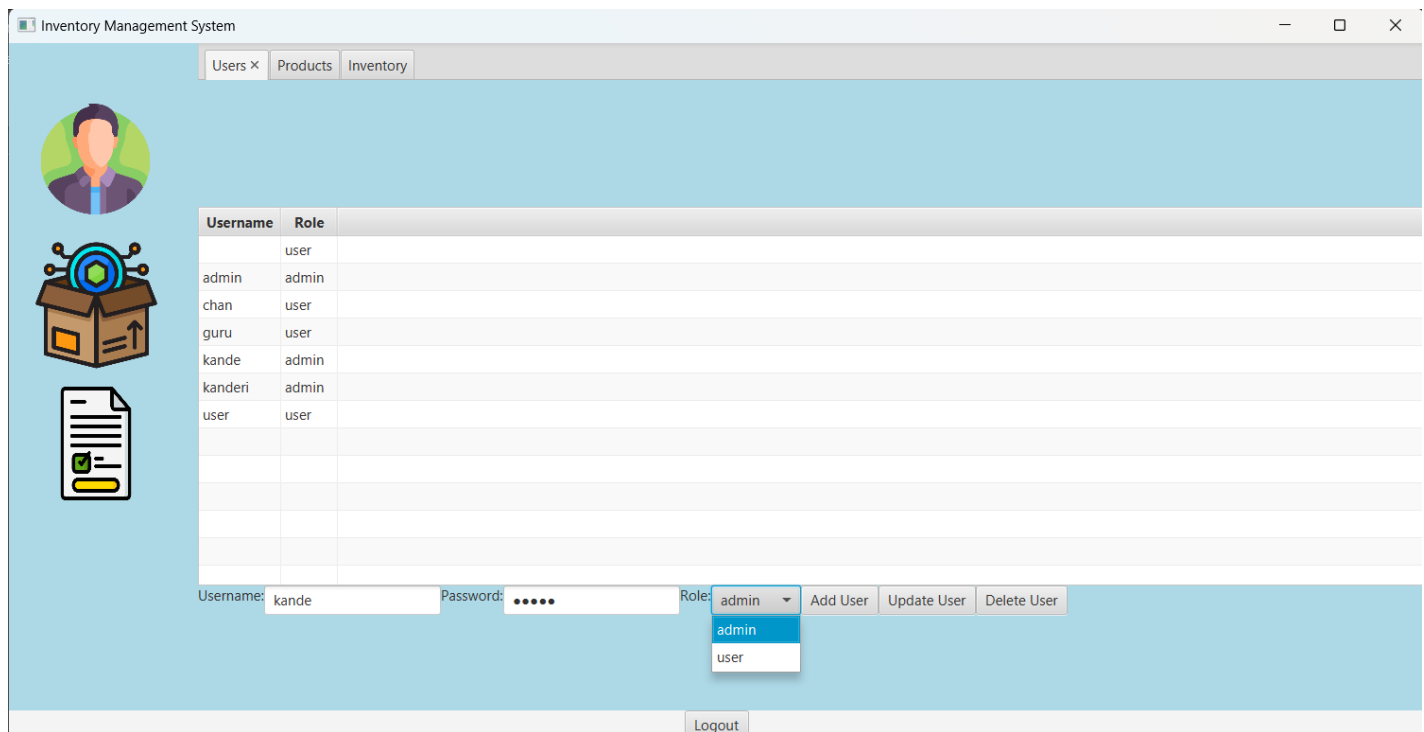
Update: The admin clicks on a row on the table. The row records will render in the text fields. The admin can edit the text fields and click on the update button to commit those changes to database and the table view.

Delete: The admin clicks on a row on the table. The row records will render in the text fields. The admin can click on the delete button to delete the record from database and the table view.

Logout: Logout button will log the admin out of the application and navigate back to login screen.

Changing the role of existing user: The admin can change the role of existing user from user level to admin and vice -versa.

Screenshot:



8. ORDER PRODUCT

- 8.1.** The customer can order products from the list present in the application dashboard.
- 8.2.** Screenshots: The product list before the user orders the product from the list.

Inventory Management System				
ID	Name	Price	Quantity	
1	Headphones	200.0	49	
2	Bags	20.99	4	
4	laptop	20.16	2	
6	Shoes	10.9	8	
7	Cables	5.2	100	
8	Computer	2.5	1	
9	Mobile Cases	250.0	1	
10	Pen Drives	30.5	7	
12	Mobiles	100.0	16	
Order Product		Logout		

The product list got updated from 1 to 0.

Inventory Management System				
ID	Name	Price	Quantity	
1	Headphones	200.0	49	
2	Bags	20.99	4	
4	laptop	20.16	2	
6	Shoes	10.9	8	
7	Cables	5.2	100	
8	Computer	2.5	0	
9	Mobile Cases	250.0	1	
10	Pen Drives	30.5	7	
12	Mobiles	100.0	16	
Order Product		Logout		

9. CUSTOMER DETAIL OPERATIONS

9.1. Functionalities:

The customer can see the products that are updated by admin and order products from the application, he can see the prices and available quantity of the products and place order by selecting the interested product.

9.1.1. The customer has options to navigate back to dashboard or logout of the application.

Inventory Management System				
ID	Name	Price	Quantity	
1	Headphones	200.0	49	
2	Bags	20.99	4	
4	laptop	20.16	2	
6	Shoes	10.9	8	
7	Cables	5.2	100	
8	Computer	2.5	0	
9	Mobile Cases	250.0	1	
10	Pen Drives	30.5	7	
12	Mobiles	100.0	16	
Order Product		Logout		

10. DATABASE TABLES

1. myapp_users
2. myapp_products
3. myapp_inventory

11. EXCEPTION HANDLING

One of the most important aspects is error trapping and error handling which is incorporated in the Inventory Management System. Maximum Exception Handling has been covered.

12. CONCLUSION:

The inventory management system is a well-designed solution that provides comprehensive management of inventory, products, and users. Its ability to perform CRUD operations, such as create, read, update, and delete, further enhances its functionality.

The user-friendly interface and separate admin and user views make it easy for users of all skill levels to navigate and use the system. Additionally, automatic inventory level updates upon placing an order and the ability to add, modify, and delete users and products can streamline business operations and reduce costs.

Overall, the system's features and functionalities, including the CRUD operations, make it a valuable asset for managing users, products, and inventory.

CODE:

Package→appalication

Main.java

```
package application;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

// This is a class named Main that extends the Application class in the JavaFX library.

public class Main extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {

        Parent root = FXMLLoader.load(getClass().getResource("/views/LoginView.fxml"));
        // The title of the login screen is set.
        primaryStage.setTitle("Inventory Management System");

        Scene scene = new Scene(root);

        scene.getStylesheets().add(getClass().getResource("/application/application.css").
toExternalForm());
        // The scene is set as the primary stage's scene, and the stage is displayed.
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public static void main(String[] args) {
        // The launch() method is called to start the JavaFX application.
        launch(args);
    }
}
```

Controllers:

AdminController

```
package controllers;

import java.io.IOException;

import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import models.Dao;
import models.Inventory;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class AdminController {
    // Injecting UI elements defined in the corresponding FXML file using @FXML
    // annotation
    @FXML
    private TextField productIdField, stockField;
    @FXML
    private Label statusLabel;
    @FXML
    private Button logoutButton;

    // Creating a Dao object to interact with the database
    private Dao dao = new Dao();

    public void addInventory() {
        // Retrieving the product ID and stock values from the respective text fields
        int productId = Integer.parseInt(productIdField.getText());
        int stock = Integer.parseInt(stockField.getText());

        // Creating an Inventory object with the retrieved values
        Inventory inventory = new Inventory(0, productId, stock);

        // Calling the Dao class to add the inventory object to the system database
        Inventory addedInventory = dao.addInventory(inventory);

        // Updating the UI with the status of the inventory addition process
        if (addedInventory != null) {
            statusLabel.setText("Inventory added successfully.");
        } else {
            statusLabel.setText("Error adding inventory.");
        }
    }

    public void logout() {
        try {
            // Loading the LoginView.fxml file to switch back to the login view
            FXMLLoader loader = new
FXMLLoader(getClass().getResource("/views/LoginView.fxml"));
            Parent root = loader.load();

            // Getting the current stage and setting its scene to the login view scene
            Stage stage = (Stage) logoutButton.getScene().getWindow();
            stage.setScene(new Scene(root));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```

        stage.show();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}
}

```

InventoryController

```

package controllers;

import java.util.List;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.TextField;
import javafx.scene.control.cell.PropertyValueFactory;
import models.Dao;
import models.Inventory;

public class InventoryController {
    // FXML elements
    @FXML
    private TableView<Inventory> inventoryTable;
    @FXML
    private TableColumn<Inventory, Integer> idColumn;
    @FXML
    private TableColumn<Inventory, Integer> productIdColumn;
    @FXML
    private TableColumn<Inventory, Integer> stockColumn;
    @FXML
    private TextField idField;
    @FXML
    private TextField productIdField;
    @FXML
    private TextField stockField;

    // Data access object
    private Dao dao = new Dao();

    // Initialize table columns and data
    public void initialize() {
        idColumn.setCellValueFactory(new PropertyValueFactory<>("id"));
        productIdColumn.setCellValueFactory(new PropertyValueFactory<>("productId"));
        stockColumn.setCellValueFactory(new PropertyValueFactory<>("stock"));

        // Get all inventories from database and display them in the table
        ObservableList<Inventory> inventoryList = FXCollections.observableArrayList();
        List<Inventory> inventories = dao.getAllInventories();
        if (inventories != null) {
            inventoryList.addAll(inventories);
        }
        inventoryTable.setItems(inventoryList);
    }

    // Add new inventory to the database and table

```

```

@FXML
private void addInventory(ActionEvent event) {
    int productId = Integer.parseInt(productIdField.getText());
    int stock = Integer.parseInt(stockField.getText());

    Inventory newInventory = new Inventory(0, productId, stock);
    Inventory addedInventory = dao.addInventory(newInventory);
    inventoryTable.getItems().add(addedInventory);
}

// Update selected inventory in the database and table
@FXML
private void updateInventory(ActionEvent event) {
    int id = Integer.parseInt(idField.getText());
    int productId = Integer.parseInt(productIdField.getText());
    int stock = Integer.parseInt(stockField.getText());

    Inventory updatedInventory = new Inventory(id, productId, stock);
    dao.updateInventory(updatedInventory);

    inventoryTable.getItems().set(inventoryTable.getSelectionModel().getSelectedIndex(),
    updatedInventory);
}

// Delete selected inventory from the database and table
@FXML
private void deleteInventory(ActionEvent event) {
    Inventory selectedItem = inventoryTable.getSelectionModel().getSelectedItem();
    if (selectedItem != null) {
        dao.deleteInventory(selectedItem.getId());
        inventoryTable.getItems().remove(selectedItem);
    }
}

// Update fields when a row in the table is clicked
@FXML
private void onTableClicked() {
    Inventory selectedItem = inventoryTable.getSelectionModel().getSelectedItem();
    if (selectedItem != null) {
        idField.setText(String.valueOf(selectedItem.getId()));
        productIdField.setText(String.valueOf(selectedItem.getProductId()));
        stockField.setText(String.valueOf(selectedItem.getStock()));
    }
}
}

```

LoginController

```

package controllers;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.PasswordField;
import javafx.scene.control.TextField;
import javafx.stage.Stage;

```

```
import models.Dao;
import models.User;

import java.io.IOException;

public class LoginController {
    // Injecting the username field from the FXML file
    @FXML
    private TextField usernameField;
    // Injecting the password field from the FXML file
    @FXML
    private PasswordField passwordField;
    // Injecting the status label from the FXML file
    @FXML
    private Label statusLabel;

    // Creating an instance of the DAO class
    private Dao dao = new Dao();

    // The method to handle the login button click
    public void login(ActionEvent event) throws IOException {
        // Getting the username and password from the fields
        String username = usernameField.getText();
        String password = passwordField.getText();

        // Fetching the user with the entered username from the database
        User user = dao.getUserByUsername(username);

        // Checking if the user exists and the password is correct
        if (user != null && user.getPassword().equals(password)) {
            // If the user exists and the password is correct, set the status label to
            "Login successful"
            statusLabel.setText("Login successful.");

            // Creating a new parent object for the next view, based on the user's role
            Parent nextViewParent;
            if ("admin".equalsIgnoreCase(user.getRole())) {
                // If the user is an admin, load the admin view
                nextViewParent =
FXMLLoader.load(getClass().getResource("/views/AdminView.fxml"));
            } else {
                // If the user is not an admin, load the user product view
                nextViewParent =
FXMLLoader.load(getClass().getResource("/views/UserProductView.fxml"));
            }

            // Creating a new scene object for the next view, based on the parent object
            Scene nextViewScene = new Scene(nextViewParent);

            Stage window = (Stage) ((Node) event.getSource()).getScene().getWindow();
            window.setScene(nextViewScene);
            window.show();
        } else {
            // If the user does not exist or the password is incorrect, set the status
            label to "Invalid username or password"
            statusLabel.setText("Invalid username or password.");
        }
    }

    // The method to handle the signup button click
    public void openSignup(ActionEvent event) throws IOException {
```

```
// Loading the signup view and creating a new scene object for it
Parent signupViewParent =
FXMLLoader.load(getClass().getResource("/views/SignupView.fxml"));
Scene signupViewScene = new Scene(signupViewParent);

// Getting the window object from the event source and setting the scene to the
signup view scene
Stage window = (Stage) ((Node) event.getSource()).getScene().getWindow();
window.setScene(signupViewScene);
window.show();
}
}
```

ProductController

```
package controllers;

import models.Product;
import models.Dao;
import javafx.fxml.FXML;
import javafx.scene.control.*;
import javafx.collections.FXCollections;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.TextField;
import javafx.scene.control.cell.PropertyValueFactory;

public class ProductController {

    @FXML
    private TextField productNameTextField;

    @FXML
    private TextField productPriceTextField;

    @FXML
    private TableColumn<Product, Integer> idColumn;

    @FXML
    private TableColumn<Product, String> nameColumn;

    @FXML
    private TableColumn<Product, Float> priceColumn;

    @FXML
    private TableColumn<Product, Integer> quantityColumn;

    @FXML
    private TextField productQuantityTextField;

    @FXML
    private Button addButton;

    @FXML
    private Button updateButton;

    @FXML
    private Button deleteButton;

    @FXML
```

```

private TableView<Product> productsTableView;

private Dao dao;

public ProductController() {
    dao = new Dao();
}

@FXML
public void initialize() {
    // Set up the columns of the TableView to be populated with product data
    idColumn.setCellValueFactory(new PropertyValueFactory<>("id"));
    nameColumn.setCellValueFactory(new PropertyValueFactory<>("name"));
    priceColumn.setCellValueFactory(new PropertyValueFactory<>("price"));
    quantityColumn.setCellValueFactory(new PropertyValueFactory<>("quantity"));

    // Populate the TableView with data from the database
    loadProducts();

    // Set up event handlers for button clicks
    addButton.setOnAction(event -> addProduct());
    updateButton.setOnAction(event -> updateProduct());
    deleteButton.setOnAction(event -> deleteProduct());
}

private void loadProducts() {
    // Load all products from the database and add them to the TableView
    productsTableView.setItems(FXCollections.observableArrayList(dao.getAllProducts()));
}

@FXML
private void addProduct() {
    // Get product data from text fields and create a new Product object
    String name = productNameTextField.getText();
    double price = Double.parseDouble(productPriceTextField.getText());
    int quantity = Integer.parseInt(productQuantityTextField.getText());
    Product product = new Product(0, name, price, quantity);

    // Add the new product to the database and refresh the TableView
    dao.addProduct(product);
    loadProducts();
}

@FXML
private void updateProduct() {
    // Get the selected product from the TableView and update its data
    Product selectedProduct =
productsTableView.getSelectionModel().getSelectedItem();
    if (selectedProduct != null) {
        String name = productNameTextField.getText();
        double price = Double.parseDouble(productPriceTextField.getText());
        int quantity = Integer.parseInt(productQuantityTextField.getText());
        Product updatedProduct = new Product(selectedProduct.getId(), name, price,
quantity);

        // Update the product in the database and refresh the TableView
        dao.updateProduct(updatedProduct);
        loadProducts();
    }
}

```



```

@FXML
private void deleteProduct() {
    // Get the selected product from the TableView and delete it from the database
    Product selectedProduct =
productsTableView.getSelectionModel().getSelectedItem();
    if (selectedProduct != null) {
        dao.deleteProduct(selectedProduct.getId());
        loadProducts();
    }
}

@FXML
private void onTableClicked() {
    // When a row in the TableView is clicked, populate the text fields with the
corresponding product data
    Product selectedItem = productsTableView.getSelectionModel().getSelectedItem();
    if (selectedItem != null) {
        productNameTextField.setText(selectedItem.getName());
        productPriceTextField.setText(String.valueOf(selectedItem.getPrice()));

productQuantityTextField.setText(String.valueOf(selectedItem.getQuantity()));
    }
}
}

```

SignupController

```

package controllers;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.control.ComboBox;
import javafx.scene.control.Label;
import javafx.scene.control.PasswordField;
import javafx.scene.control.TextField;
import models.Dao;
import models.User;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import java.io.IOException;

public class SignupController {
    @FXML
    private TextField usernameField;
    @FXML
    private PasswordField passwordField;

    // A ComboBox for selecting the user role (either "Admin" or "User")
    @FXML
    private ComboBox<String> roleComboBox;

    // A TextField for manually entering the user role (in case the ComboBox value is
not used)
    @FXML
    private TextField roleField;
}

```

```
// A label for displaying the status of the signup process (success/failure
messages)
@FXML
private Label statusLabel;

// A reference to the Data Access Object (DAO) used to interact with the database
private Dao dao = new Dao();

@FXML
public void initialize() {
    // Initialize the options for the ComboBox with the two possible roles ("Admin"
or "User")
    ObservableList<String> roleOptions = FXCollections.observableArrayList("Admin",
"User");
    roleComboBox.setItems(roleOptions);

    // Set the default value of the ComboBox to "User"
    roleComboBox.getSelectionModel().select("User");
}

public void signup(ActionEvent event) throws IOException {
    // Get the username, password, and role entered by the user
    String username = usernameField.getText();
    String password = passwordField.getText();
    String role = roleComboBox.getValue();

    // Check if a user with the same username already exists in the database
    User existingUser = dao.getUserByUsername(username);
    if (existingUser != null) {
        // If a user with the same username exists, display an error message
        statusLabel.setText("Username already exists.");
    } else {

        User newUser = new User(username, password, role);
        if (dao.addUser(newUser)) {
            // If the user was successfully added to the database, display a success
message and redirect to the login view
            statusLabel.setText("User added successfully! Redirecting to login...");

            Parent loginViewParent =
FXMLLoader.load(getClass().getResource("/views/LoginView.fxml"));
            Scene loginViewScene = new Scene(loginViewParent);
            Stage window = (Stage) ((Node)
event.getSource()).getScene().getWindow();
            window.setScene(loginViewScene);
            window.show();
        } else {
            // If there was an error adding the user to the database, display an
error message
            statusLabel.setText("Error: Could not add user.");
        }
    }
}
}
```

UserController.java

```
package controllers;

import javafx.fxml.FXML;
```

```
import javafx.scene.control.*;
import javafx.scene.control.cell.PropertyValueFactory;
import models.Dao;
import models.User;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;

public class UserController {

    @FXML
    private TextField usernameTextField;

    @FXML
    private TableColumn<User, String> usernameColumn;

    //    @FXML
    //    private TableColumn<User, String> passwordColumn;

    @FXML
    private TableColumn<User, String> roleColumn;

    @FXML
    private PasswordField passwordField;

    @FXML
    private ComboBox<String> roleComboBox;

    @FXML
    private Button addUserButton;

    @FXML
    private Button updateUserButton;

    @FXML
    private Button deleteUserButton;

    @FXML
    private TableView<User> usersTableView;

    private Dao dao;

    public UserController() {
        dao = new Dao();
    }

    @FXML
    public void initialize() {
        // Set the cell value factories for the table columns to the corresponding user
attributes
        usernameColumn.setCellValueFactory(new PropertyValueFactory<>("username"));
        // passwordColumn.setCellValueFactory(new PropertyValueFactory<>("password"));
        roleColumn.setCellValueFactory(new PropertyValueFactory<>("role"));

        // Populate the role ComboBox
        ObservableList<String> roleList = FXCollections.observableArrayList("admin",
"user");
        roleComboBox.setItems(roleList);
        // roleComboBox.getItems().addAll("Admin", "User");

        // Load the existing users into the table view and set up the buttons to call
the corresponding methods
        loadUsers();
    }
}
```

```

        addUserButton.setOnAction(event -> addUser());
        updateUserButton.setOnAction(event -> updateUser());
        deleteUserButton.setOnAction(event -> deleteUser());
    }

    private void loadUsers() {
        // Load the users from the DAO and set them into the table view
        usersTableView.setItems(FXCollections.observableArrayList(dao.getUsers()));
    }

    @FXML
    private void addUser() {
        // Get the values from the text fields and ComboBox and create a new user with
them
        String username = usernameTextField.getText();
        String password = passwordField.getText();
        String role = roleComboBox.getSelectionModel().getSelectedItem();
        User user = new User(username, password, role);

        // Add the user to the DAO and reload the table view to reflect the changes
        dao.addUser(user);
        loadUsers();
    }

    @FXML
    private void updateUser() {
        // Get the selected user from the table view and the values from the text fields
and ComboBox
        User selectedUser = usersTableView.getSelectionModel().getSelectedItem();
        if (selectedUser != null) {
            String username = usernameTextField.getText();
            String password = passwordField.getText();
            String role = roleComboBox.getSelectionModel().getSelectedItem();
            User updatedUser = new User(username, password, role);

            // Update the user in the DAO and reload the table view to reflect the
changes
            dao.updateUser(selectedUser.getUsername(), updatedUser); // Pass the old
username as an additional parameter
            loadUsers();
        }
    }

    @FXML
    private void deleteUser() {
        // Get the selected user from the table view and delete it from the DAO
        User selectedUser = usersTableView.getSelectionModel().getSelectedItem();
        if (selectedUser != null) {
            dao.deleteUser(selectedUser.getUsername());
            loadUsers();
        }
    }

    @FXML
    private void onTableClicked() {
        User selectedItem = usersTableView.getSelectionModel().getSelectedItem();
        if (selectedItem != null) {
            usernameTextField.setText(selectedItem.getUsername());
            passwordField.setText(selectedItem.getPassword());
            roleComboBox.setValue(selectedItem.getRole());
        }
    }

```

```
}
```

```
@FXML
```

```
private void onTableClicked() {
    User selectedItem = usersTableView.getSelectionModel().getSelectedItem();
    if (selectedItem != null) {
        usernameTextField.setText(selectedItem.getUsername());
        passwordField.setText(selectedItem.getPassword());
        roleComboBox.setValue(selectedItem.getRole());
    }
}
```

```
}
```

Package→Models:

Admin.java

```
package models;
3
public class Admin extends User {

    public Admin(String username, String password, String role) {
        super(username, password, role);
    }

    // Additional methods specific to the admin can be added here
}
```

Dao.java

```
package models;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;

public class Dao {

    // User-related CRUD operations
    public User getUserByUsernameAndPassword(String username, String password) {
        User user = null;
        String query = "SELECT * FROM myapp_users WHERE username = ? AND password = ?";

        try (Connection connection = DBConnect.getConnection();
            PreparedStatement preparedStatement =
connection.prepareStatement(query)) {

            preparedStatement.setString(1, username);
            preparedStatement.setString(2, password);
            ResultSet resultSet = preparedStatement.executeQuery();

            if (resultSet.next()) {
                String role = resultSet.getString("role");
                user = new User(username, password, role);
            }
        } catch (SQLException e) {
```

```
e.printStackTrace();
}

return user;
}

public List<User> getUsers() {
    List<User> users = new ArrayList<>();
    String query = "SELECT * FROM myapp_users";

    try (Connection connection = DBConnect.getConnection();
        Statement statement = connection.createStatement();
        ResultSet resultSet = statement.executeQuery(query)) {

        while (resultSet.next()) {
            String username = resultSet.getString("username");
            String password = resultSet.getString("password");
            String role = resultSet.getString("role");

            users.add(new User(username, password, role));
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return users;
}

public boolean updateUser(String oldUsername, User user) {
    String query = "UPDATE myapp_users SET username = ?, password = ?, role = ?
WHERE username = ?";
    try (Connection connection = DBConnect.getConnection();
        PreparedStatement preparedStatement =
connection.prepareStatement(query)) {

        preparedStatement.setString(1, user.getUsername());
        preparedStatement.setString(2, user.getPassword());
        preparedStatement.setString(3, user.getRole());
        preparedStatement.setString(4, oldUsername);
        int result = preparedStatement.executeUpdate();

        return result > 0;
    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
}

public boolean deleteUser(String username) {
    String query = "DELETE FROM myapp_users WHERE username = ?";
    try (Connection connection = DBConnect.getConnection();
        PreparedStatement preparedStatement =
connection.prepareStatement(query)) {

        preparedStatement.setString(1, username);
        int result = preparedStatement.executeUpdate();

        return result > 0;
    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
}
```

```

    }

    public List<User> getAllUsers() {
        List<User> users = new ArrayList<>();
        String query = "SELECT * FROM myapp_users";

        try (Connection connection = DBConnect.getConnection();
            Statement statement = connection.createStatement();
            ResultSet resultSet = statement.executeQuery(query)) {

            while (resultSet.next()) {
                String username = resultSet.getString("username");
                String password = resultSet.getString("password");
                String role = resultSet.getString("role");

                users.add(new User(username, password, role));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return users;
    }

    public boolean addUser(User user) {
        String query = "INSERT INTO myapp_users (username, password, role) VALUES (?, ?, ?)";

        try (Connection connection = DBConnect.getConnection();
            PreparedStatement preparedStatement =
connection.prepareStatement(query)) {

            preparedStatement.setString(1, user.getUsername());
            preparedStatement.setString(2, user.getPassword());
            preparedStatement.setString(3, user.getRole());
            int result = preparedStatement.executeUpdate();

            return result > 0;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }

    public User getUserByUsername(String username) {
        User user = null;
        String query = "SELECT * FROM myapp_users WHERE username = ?";

        try (Connection connection = DBConnect.getConnection();
            PreparedStatement preparedStatement =
connection.prepareStatement(query)) {

            preparedStatement.setString(1, username);
            ResultSet resultSet = preparedStatement.executeQuery();

            if (resultSet.next()) {
                String password = resultSet.getString("password");
                String role = resultSet.getString("role");
                user = new User(username, password, role);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

```

```

        return user;
    }

    // Product-related CRUD operations
    public List<Product> getAllProducts() {
        List<Product> products = new ArrayList<>();
        String query = "SELECT * FROM myapp_products";

        try (Connection connection = DBConnect.getConnection();
            Statement statement = connection.createStatement();
            ResultSet resultSet = statement.executeQuery(query)) {

            while (resultSet.next()) {
                int id = resultSet.getInt("id");
                String name = resultSet.getString("name");
                double price = resultSet.getDouble("price");
                int quantity = resultSet.getInt("quantity");

                products.add(new Product(id, name, price, quantity));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return products;
    }

    public boolean addProduct(Product product) {
        String query = "INSERT INTO myapp_products (name, price, quantity) VALUES (?, ?, ?)";

        try (Connection connection = DBConnect.getConnection();
            PreparedStatement preparedStatement =
connection.prepareStatement(query)) {

            preparedStatement.setString(1, product.getName());
            preparedStatement.setDouble(2, product.getPrice());
            preparedStatement.setInt(3, product.getQuantity());
            int result = preparedStatement.executeUpdate();

            return result > 0;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }

    public boolean updateProduct(Product product) {
        String query = "UPDATE myapp_products SET name = ?, price = ?, quantity = ?
WHERE id = ?";

        try (Connection connection = DBConnect.getConnection();
            PreparedStatement preparedStatement =
connection.prepareStatement(query)) {

            preparedStatement.setString(1, product.getName());
            preparedStatement.setDouble(2, product.getPrice());
            preparedStatement.setInt(3, product.getQuantity());
            preparedStatement.setInt(4, product.getId());
            int result = preparedStatement.executeUpdate();

            return result > 0;
        }
    }

```



```

    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
}

public boolean deleteProduct(int productId) {
    String query = "DELETE FROM myapp_products WHERE id = ?";
    try (Connection connection = DBConnect.getConnection();
        PreparedStatement preparedStatement =
connection.prepareStatement(query)) {

        preparedStatement.setInt(1, productId);
        int result = preparedStatement.executeUpdate();

        return result > 0;
    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
}

public void decrementProductQuantity(int productId) {
    String query = "UPDATE myapp_products SET quantity = quantity - 1 WHERE id = ?
AND quantity > 0";

    try (Connection connection = DBConnect.getConnection();
        PreparedStatement preparedStatement =
connection.prepareStatement(query)) {

        preparedStatement.setInt(1, productId);
        preparedStatement.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

// Inventory-related CRUD operations
public List<Inventory> getAllInventories() {
    List<Inventory> inventories = new ArrayList<>();
    String query = "SELECT * FROM myapp_inventory";

    try (Connection connection = DBConnect.getConnection();
        Statement statement = connection.createStatement();
        ResultSet resultSet = statement.executeQuery(query)) {

        while (resultSet.next()) {
            int id = resultSet.getInt("id");
            int productId = resultSet.getInt("product_id");
            int stock = resultSet.getInt("stock");

            Inventory inventory = new Inventory(id, productId, stock);
            inventories.add(inventory);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return inventories;
}

```

```

public Inventory addInventory(Inventory inventory) {
    String query = "INSERT INTO myapp_inventory (product_id, stock) VALUES (?, ?)";
    ResultSet generatedKeys = null;
    try (Connection connection = DBConnect.getConnection();
        PreparedStatement preparedStatement = connection.prepareStatement(query,
Statement.RETURN_GENERATED_KEYS)) {

        preparedStatement.setInt(1, inventory.getProductId());
        preparedStatement.setInt(2, inventory.getStock());
        int result = preparedStatement.executeUpdate();

        if (result > 0) {
            generatedKeys = preparedStatement.getGeneratedKeys();
            if (generatedKeys.next()) {
                int id = generatedKeys.getInt(1);
                return new Inventory(id, inventory.getProductId(),
inventory.getStock());
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        if (generatedKeys != null) {
            try {
                generatedKeys.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
    return null;
}

public boolean updateInventory(Inventory inventory) {
    String query = "UPDATE myapp_inventory SET product_id = ?, stock = ? WHERE id =
?";
    try (Connection connection = DBConnect.getConnection();
        PreparedStatement preparedStatement =
connection.prepareStatement(query)) {

        preparedStatement.setInt(1, inventory.getProductId());
        preparedStatement.setInt(2, inventory.getStock());
        preparedStatement.setInt(3, inventory.getId());
        int result = preparedStatement.executeUpdate();

        return result > 0;
    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
}

public boolean deleteInventory(int inventoryId) {
    String query = "DELETE FROM myapp_inventory WHERE id = ?";
    try (Connection connection = DBConnect.getConnection();
        PreparedStatement preparedStatement =
connection.prepareStatement(query)) {

        preparedStatement.setInt(1, inventoryId);
        int result = preparedStatement.executeUpdate();

```

```
        return result > 0;
    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
}
```

DBConnect.java

```
package models;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DBConnect {

    private static final String url =
"jdbc:mysql://www.papademas.net:3307/510fp?autoReconnect=true&useSSL=false";
    private static final String username = "fp510";
    private static final String password = "510";

    public static Connection getConnection() {
        Connection connection = null;

        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            connection = DriverManager.getConnection(url, username, password);
        } catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        }

        return connection;
    }
}
```

Inventory.java

```
package models;

public class Inventory {
    private int id; // unique identifier for the inventory
    private int productId; // identifier for the product this inventory represents
    private int stock; // number of items in stock

    public Inventory(int id, int productId, int stock) {
        this.id = id;
        this.productId = productId;
        this.stock = stock;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public int getProductId() {
        return productId;
    }
}
```

```
public void setProductId(int productId) {
    this.productId = productId;
}

public int getStock() {
    return stock;
}

public void setStock(int stock) {
    this.stock = stock;
}
}
```

Product.java

```
package models;

public class Product {
    private int id;
    private String name;
    private double price;
    private int quantity;

    // Constructor method that initializes the product's id, name, price, and quantity.
    public Product(int id, String name, double price, int quantity) {
        this.id = id;
        this.name = name;
        this.price = price;
        this.quantity = quantity;
    }

    // Public method that returns the product's id.
    public int getId() {
        return id;
    }

    // Public method that sets the product's id.
    public void setId(int id) {
        this.id = id;
    }

    // Public method that returns the product's name.
    public String getName() {
        return name;
    }

    // Public method that sets the product's name.
    public void setName(String name) {
        this.name = name;
    }

    // Public method that returns the product's price.
    public double getPrice() {
        return price;
    }

    // Public method that sets the product's price.
    public void setPrice(double price) {
        this.price = price;
    }
}
```

```
// Public method that returns the product's quantity.
public int getQuantity() {
    return quantity;
}

// Public method that sets the product's quantity.
public void setQuantity(int quantity) {
    this.quantity = quantity;
}
}
}
```

User.java

```
package models;

public class User {
    // Define private fields to store user data
    private String username;
    private String password;
    private String role;

    // Create a constructor to initialize user data
    public User(String username, String password, String role) {
        this.username = username;
        this.password = password;
        this.role = role;
    }

    // Create getter method for user role
    public String getRole() {
        return role;
    }

    // Create setter method for user role
    public void setRole(String role) {
        this.role = role;
    }

    // Create getter method for username
    public String getUsername() {
        return username;
    }

    // Create setter method for username
    public void setUsername(String username) {
        this.username = username;
    }

    // Create getter method for password
    public String getPassword() {
        return password;
    }

    // Create setter method for password
    public void setPassword(String password) {
        this.password = password;
    }
}
```

