**Project Overview**

* **Project Name:** CSE380-Project
* **Title:** SMS based Remote Server Monitoring System

**Objectives**

* To develop a system that allows remote monitoring of server metrics via SMS.
* To enable server management through a user interface that includes login functionality, server management, and user management.
* To provide administrators with the ability to view and manage server status, as well as to approve restart requests.

**Design Concepts**

* The project is structured in a Java-based MVC architecture.
* The **Main** class initializes the GUI and sets up the main frame for the application.
* The **Admin** and **User** classes extend an **AbstractUser** class, encapsulating shared functionality and allowing for role-based access control.
* The **Server** class contains methods for simulating server monitoring and managing alert subscriptions.
* The **LoginPanel** class provides a user interface for authentication.
* The **SMSHandler** class simulates the sending of SMS messages to notify about server status.
* The **UserDatabase** class acts as a simple authentication system with hardcoded users.

**Applications**

* The system can be used by IT departments to monitor server health remotely.
* It can serve as a tool for system administrators to manage servers and users within an organization.
* The alert system can notify administrators of critical server metrics, allowing for proactive maintenance.

**Additional Files for Detailed Review**

* UI classes (**AdminDashboard**, **LoginPanel**, **ManageServers**, **ManageUsers**, **ServerManagement**, **UserDashboard**) that likely contain the user interface logic.
* Utility classes (**FileHandler**, **SMSHandler**, **UserDatabase**) that provide supporting functionality such as file operations and user management.
* Wireframes (**smartSMS\_1\_2023.pdf**, **smartSMS\_2\_2023.pdf**) that outline the design and layout of the user interface.

**Executive Summary**

This project aims to develop a Java-based application that enables the monitoring and management of server metrics remotely via SMS notifications. It is designed to assist system administrators in overseeing server health and performance without the need for constant manual supervision.

**Introduction**

With the increasing reliance on servers for business operations, uninterrupted service and quick response to issues are paramount. This project introduces a solution that allows administrators to receive alerts and manage servers through a user-friendly interface.

**Project Objectives**

* Develop a GUI application for server management.
* Implement SMS notifications for server alerts.
* Provide role-based access control for system administrators and users.

**Methodology**

The project follows an object-oriented approach using Java, with a focus on modularity and reusability of code. The MVC (Model-View-Controller) design pattern is employed to separate concerns and enhance maintainability.

**System Design**

**Architecture**

The system is built on a Java Swing framework for the GUI, with backend logic to simulate server monitoring and SMS notifications.

**Components**

* **Main Application:** Initializes the GUI and sets the stage for user interaction.
* **User Authentication:** Manages login sessions and user credentials.
* **Server Management:** Allows administrators to monitor and manage server metrics.
* **Alert System:** Sends SMS alerts when server metrics exceed predefined thresholds.

**Implementation**

**User Interface**

The interface is designed to be intuitive, with separate dashboards for admins and users, login panels, and server management screens.

**Server Simulation**

Server metrics such as CPU usage, memory usage, and network latency are simulated for demonstration purposes.

**SMS Handling**

A mock SMS handler is implemented to demonstrate the alert system's functionality.

**Testing**

The application includes basic unit tests to ensure the reliability of key functionalities such as user authentication, server monitoring, and SMS notifications.

**Results and Discussion**

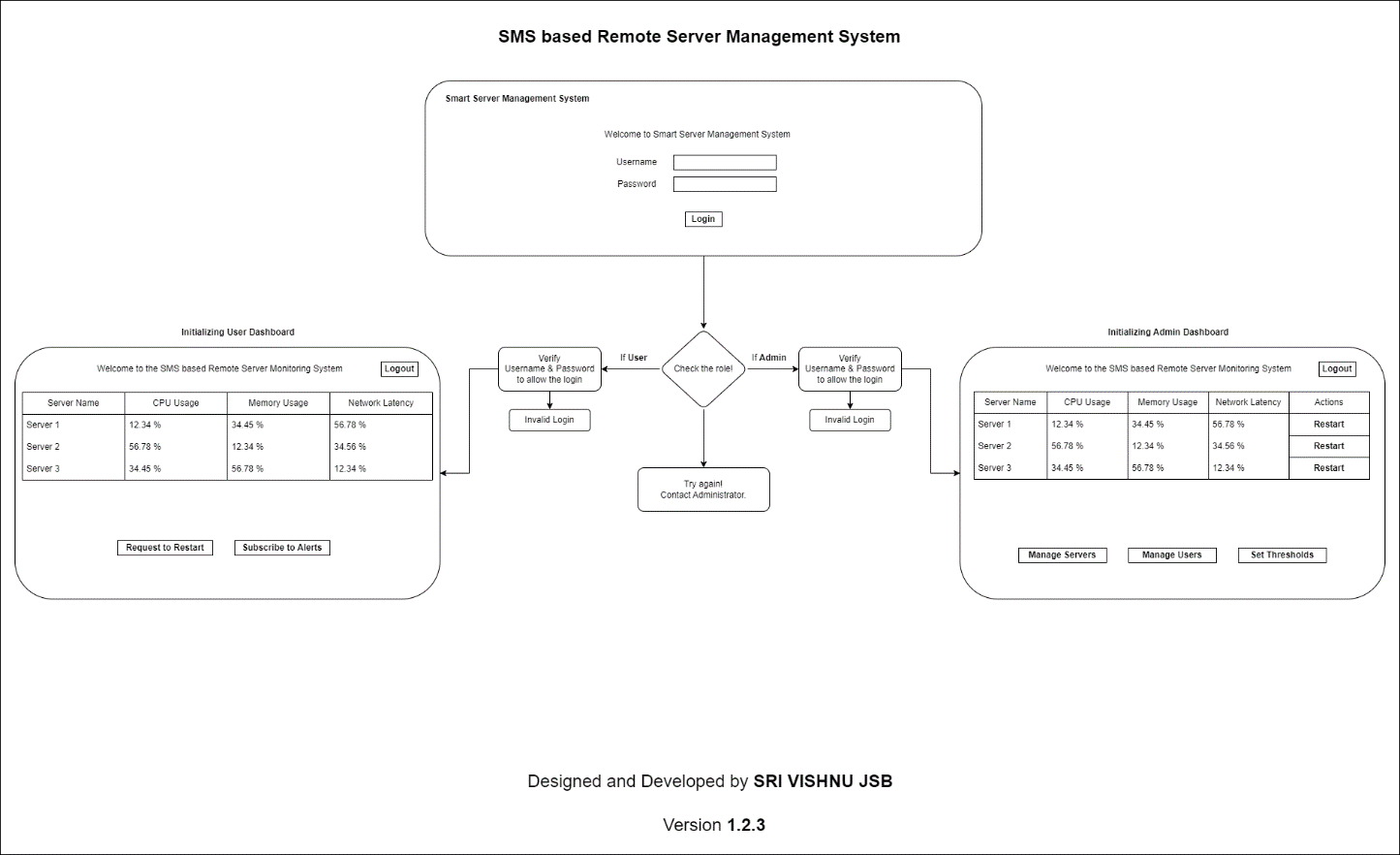
The system successfully demonstrates the core functionalities expected of a remote server monitoring system. However, the current implementation is a prototype, and further development is required for real-world application.

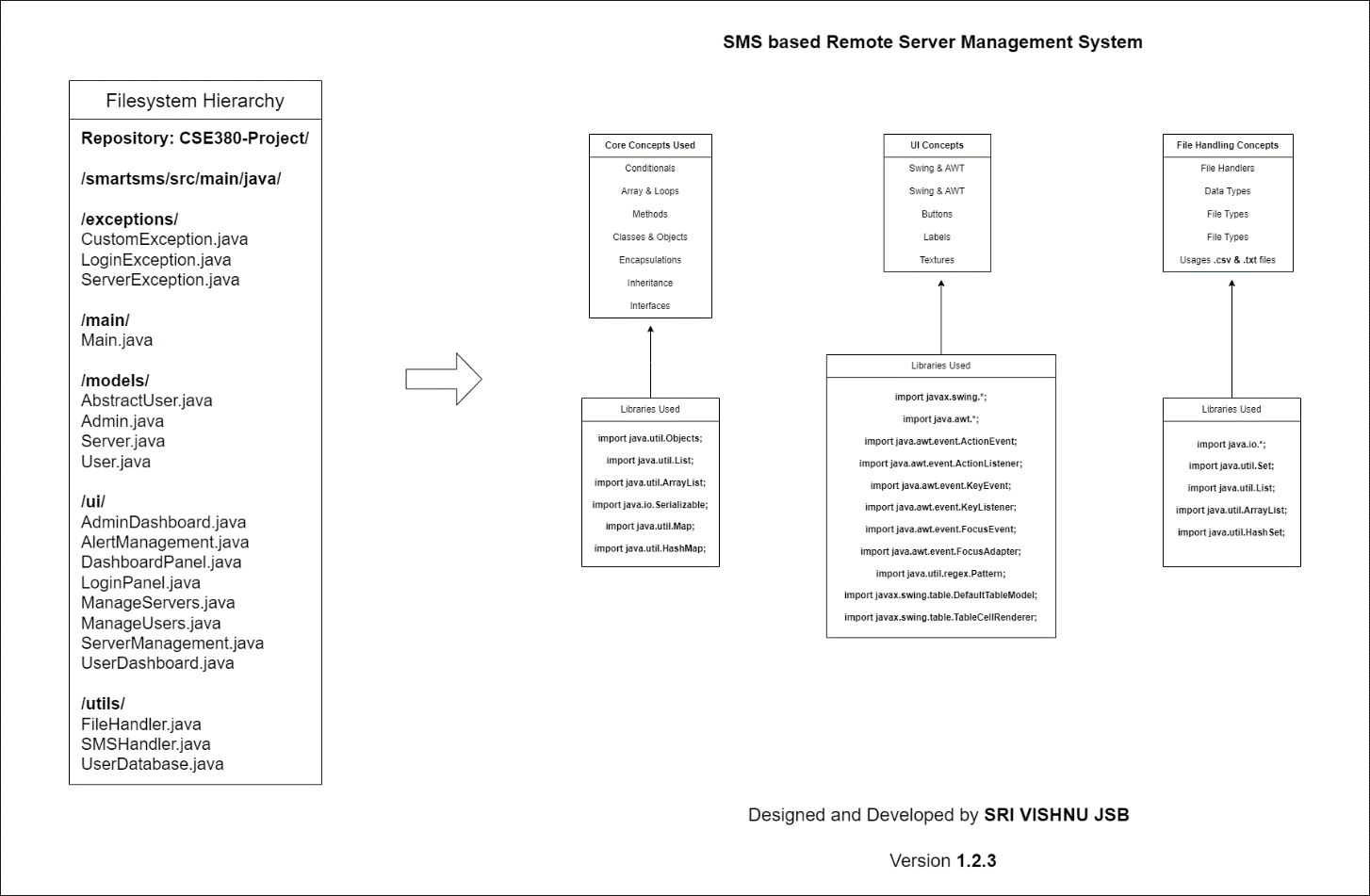
**Conclusion**

The SMS-based Remote Server Monitoring System shows promise in providing a robust tool for server administration. Future work will focus on integrating real server metrics, enhancing security, and improving the user experience.

**Appendices**

* **Source Code:** A link to the GitHub repository.
* **Wireframes:** Visual designs of the user interface.
* **User Manual:** Instructions on how to use the application.
* **Developer Documentation:** Detailed documentation for future developers.





**Source Code of Server Monitoring System**

Main.java

package main.java.main;

import main.java.ui.LoginPanel;

import javax.swing.\*;

public class Main {

    public static void main(String[] args) {

        SwingUtilities.invokeLater(new Runnable() {

            public void run() {

                createAndShowGUI();

            }

        });

    }

    public static void createAndShowGUI() {

        JFrame frame = new JFrame("Smart Server Management System");

        frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

        frame.setSize(1200, 600);

        LoginPanel loginPanel = new LoginPanel();

        frame.add(loginPanel);

        frame.setVisible(true);

    }

}

AdminDashboard.java

package main.java.ui;

import main.java.models.Server;

import main.java.utils.FileHandler;

import main.java.models.AbstractUser;

import main.java.utils.UserDatabase;

import javax.swing.\*;

import java.awt.\*;

import java.util.ArrayList;

import java.util.List;

import javax.swing.table.DefaultTableModel;

public class AdminDashboard {

    private ArrayList<Server> servers;

    private List<AbstractUser> users;

    private GridBagConstraints gbc;

    private JPanel parentPanel;  // Add this line

    private DashboardPanel dashboardPanel;  // Add this line to declare the field

    private DefaultTableModel tableModel;  // Add this line to declare the field

    // Declare serverDropdown as a class member

    private JComboBox<String> serverDropdown = new JComboBox<>();

    public AdminDashboard(ArrayList<Server> servers, List<AbstractUser> users, GridBagConstraints gbc, JPanel parentPanel, DashboardPanel dashboardPanel, DefaultTableModel tableModel) {

        //System.out.println("Number of servers in AdminDashboard: " + servers.size());  // Debug line

        this.servers = servers;

        this.users = users;

        this.gbc = gbc;

        this.parentPanel = parentPanel;  // Initialize the JPanel

        this.dashboardPanel = dashboardPanel;  // Initialize the DashboardPanel field

        this.tableModel = tableModel;  // Initialize the DefaultTableModel field

        //System.out.println("AdminDashboard Server object: "+servers.size());

    }

    public void initializeAdminDashboard(JPanel panel) {

        System.out.println("Initializing Admin Dashboard");  // Debugging line

        JPanel adminPanel = new JPanel(new FlowLayout());  // Create a new JPanel with FlowLayout

        JButton manageServersButton = new JButton("Manage Servers");

        manageServersButton.addActionListener(e -> openManageServersPanel());

        JButton manageUsersButton = new JButton("Manage Users");

        manageUsersButton.addActionListener(e -> openManageUsersPanel());

        JButton setThresholdsButton = new JButton("Set Thresholds");

        setThresholdsButton.addActionListener(e -> setThresholdsForServers());

        adminPanel.add(manageServersButton);  // Add the button to the JPanel

        adminPanel.add(manageUsersButton);    // Add the button to the JPanel

        adminPanel.add(setThresholdsButton);  // Add the button to the JPanel

        gbc.gridx = 0;

        gbc.gridy = 4;  // Adjust as needed

        gbc.weighty = 0.1;

        gbc.fill = GridBagConstraints.HORIZONTAL;

        parentPanel.add(adminPanel, gbc);  // Use parentPanel to add components

    }

    public void openManageServersPanel() {

        JFrame manageServersFrame = new JFrame("Manage Servers");

        // Create a DefaultTableModel object

        String[] columnNames = {"Server Name", "CPU Limit", "Memory Limit", "Network Limit"};

        DefaultTableModel tableModel = new DefaultTableModel(columnNames, 0);

        // Create a ServerManagement object with the tableModel

        ServerManagement serverManagement = new ServerManagement(servers,tableModel);

        //ManageServers manageServersPanel = new ManageServers(new ArrayList<>(this.servers), serverManagement,this);

        ManageServers manageServersPanel = new ManageServers(this.servers, serverManagement, this);

        manageServersFrame.add(manageServersPanel);

        manageServersFrame.setSize(1000, 400);

        manageServersFrame.setVisible(true);

        manageServersFrame.setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

        manageServersFrame.addWindowListener(new java.awt.event.WindowAdapter() {

            @Override

            public void windowClosed(java.awt.event.WindowEvent windowEvent) {

                refreshTable();  // Refresh the table when the ManageServers window is closed

            }

        });

    }

    public void openManageUsersPanel() {

        List<AbstractUser> allUsers = new ArrayList<>();

        // Load users from UserDatabase

        if (UserDatabase.users != null) {

            allUsers.addAll(UserDatabase.users);

        }

        // Load users from users.txt

        //List<AbstractUser> fileUsers = FileHandler.readUsersFromTextFile("users.txt"); // Assuming you have a method that reads users from a text file

        List<AbstractUser> fileUsers = FileHandler.readUsersFromCSVFile("users.csv"); // Assuming you have a method that reads users from a text file

        if (fileUsers != null) {

            allUsers.addAll(fileUsers);

        }

        JFrame manageUsersFrame = new JFrame("Manage Users");

        ManageUsers manageUsersPanel = new ManageUsers(allUsers);  // No need for explicit cast now

        manageUsersFrame.add(manageUsersPanel);

        manageUsersFrame.setSize(1000, 400);

        manageUsersFrame.setVisible(true);

        manageUsersFrame.setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

    }

    public void setThresholdsForServers() {

        // Initialize components

        JComboBox<String> serverDropdown = new JComboBox<>();

        serverDropdown.setPreferredSize(new Dimension(150, 25));  // Width: 150, Height: 25

        JTextField cpuField = new JTextField(5);

        JTextField memoryField = new JTextField(5);

        JTextField networkField = new JTextField(5);

        JCheckBox applyToAllCheckbox = new JCheckBox("Apply to All Servers");

        // Populate server dropdown

        if (servers.isEmpty()) {

            serverDropdown.addItem("No server found");

            applyToAllCheckbox.setEnabled(false);

        } else {

            for (Server server : servers) {

                serverDropdown.addItem(server.getName());

            }

        }

        // Create panel and add components

        JPanel panel = new JPanel();

        panel.add(new JLabel("Select Server:"));

        panel.add(serverDropdown);

        panel.add(new JLabel("CPU Limit:"));

        panel.add(cpuField);

        panel.add(new JLabel("Memory Limit:"));

        panel.add(memoryField);

        panel.add(new JLabel("Network Limit:"));

        panel.add(networkField);

        panel.add(applyToAllCheckbox);

        // Show dialog

        int result = JOptionPane.showConfirmDialog(null, panel, "Set Thresholds", JOptionPane.OK\_CANCEL\_OPTION);

        // Handle dialog result

        try {

            if (result == JOptionPane.OK\_OPTION) {

                String selectedServerName = (String) serverDropdown.getSelectedItem();

                int cpuThreshold = Integer.parseInt(cpuField.getText());

                int memoryThreshold = Integer.parseInt(memoryField.getText());

                int networkThreshold = Integer.parseInt(networkField.getText());

                if (applyToAllCheckbox.isSelected()) {

                    for (Server server : servers) {

                        server.setCpuThreshold(cpuThreshold);

                        server.setMemoryThreshold(memoryThreshold);

                        server.setNetworkThreshold(networkThreshold);

                    }

                } else {

                    for (Server server : servers) {

                        if (server.getName().equals(selectedServerName)) {

                            server.setCpuThreshold(cpuThreshold);

                            server.setMemoryThreshold(memoryThreshold);

                            server.setNetworkThreshold(networkThreshold);

                            break;

                        }

                    }

                }

            }

        } catch (NumberFormatException e) {

            JOptionPane.showMessageDialog(null, "Please enter valid integer values for thresholds.");

        } catch (Exception e) {

            JOptionPane.showMessageDialog(null, "An error occurred: " + e.getMessage());

        }

    }

    public void refreshTable() {

        dashboardPanel.refreshServers();

    }

}

AlertManagement.java

package main.java.ui;

import main.java.models.Server;

import javax.swing.\*;

import java.util.ArrayList;

public class AlertManagement {

    private ArrayList<Server> servers;

    private JComboBox<String> serverDropdown;

    private JComboBox<String> alertTypeDropdown;

    public AlertManagement(ArrayList<Server> servers) {

        if (servers == null) {

            throw new IllegalArgumentException("Servers list cannot be null");

        }

        this.servers = servers;

        this.serverDropdown = new JComboBox<>();

        this.alertTypeDropdown = new JComboBox<>();

        updateDropdowns();

    }

    public void subscribeToAlerts() {

        JPanel panel = createSubscriptionPanel();

        int result = showSubscriptionDialog(panel);

        if (result == JOptionPane.OK\_OPTION) {

            handleSubscription();

        }

    }

    public JPanel createSubscriptionPanel() {

        this.serverDropdown = createServerDropdown();  // Use 'this' to refer to the class-level variable

        this.alertTypeDropdown = createAlertTypeDropdown();  // Use 'this' to refer to the class-level variable

        JPanel panel = new JPanel();

        panel.add(new JLabel("Select Server:"));

        panel.add(this.serverDropdown);  // Use 'this' to refer to the class-level variable

        panel.add(new JLabel("Select Alert Type:"));

        panel.add(this.alertTypeDropdown);  // Use 'this' to refer to the class-level variable

        return panel;

    }

    public JComboBox<String> createServerDropdown() {

        JComboBox<String> serverDropdown = new JComboBox<>();

        serverDropdown.addItem("All Servers");

        for (Server server : servers) {

            serverDropdown.addItem(server.getName());

        }

        return serverDropdown;

    }

    public JComboBox<String> createAlertTypeDropdown() {

        String[] alertTypes = {"All Alerts", "CPU Usage", "Memory Usage", "Network Latency"};

        JComboBox<String> dropdown = new JComboBox<>(alertTypes);

        return dropdown;

    }

    public int showSubscriptionDialog(JPanel panel) {

        return JOptionPane.showConfirmDialog(null, panel, "Subscribe to Alerts", JOptionPane.OK\_CANCEL\_OPTION);

    }

    public void handleSubscription() {

        if (serverDropdown == null || alertTypeDropdown == null) {

            // Log an error or show a message to the user

            System.err.println("Dropdowns are not initialized.");

            return;

        }

        String selectedServerName = (String) serverDropdown.getSelectedItem();

        String selectedAlertType = (String) alertTypeDropdown.getSelectedItem();

        if (selectedServerName == null || selectedAlertType == null) {

            // Log an error or show a message to the user

            System.err.println("No selection made in one or both dropdowns.");

            return;

        }

        if ("All Servers".equals(selectedServerName)) {

            for (Server server : servers) {

                if ("All Alerts".equals(selectedAlertType)) {

                    server.subscribeToAlert("CPU Usage");

                    server.subscribeToAlert("Memory Usage");

                    server.subscribeToAlert("Network Latency");

                } else {

                    server.subscribeToAlert(selectedAlertType);

                }

            }

        } else {

            for (Server server : servers) {

                if (server.getName().equals(selectedServerName)) {

                    if ("All Alerts".equals(selectedAlertType)) {

                        server.subscribeToAlert("CPU Usage");

                        server.subscribeToAlert("Memory Usage");

                        server.subscribeToAlert("Network Latency");

                    } else {

                        server.subscribeToAlert(selectedAlertType);

                    }

                    break;

                }

            }

        }

    }

    public void subscribeAllServersToAllAlerts() {

        for (Server server : servers) {

            server.subscribeToAlert("CPU Usage");

            server.subscribeToAlert("Memory Usage");

            server.subscribeToAlert("Network Latency");

        }

    }

    public void subscribeSpecificServerToAllAlerts(String serverName) {

        for (Server server : servers) {

            if (server.getName().equals(serverName)) {

                server.subscribeToAlert("CPU Usage");

                server.subscribeToAlert("Memory Usage");

                server.subscribeToAlert("Network Latency");

                break;

            }

        }

    }

    public void subscribeAllServers(String alertType) {

        for (Server server : servers) {

            server.subscribeToAlert(alertType);

        }

    }

    public void subscribeSpecificServer(String serverName, String alertType) {

        for (Server server : servers) {

            if (server.getName().equals(serverName)) {

                server.subscribeToAlert(alertType);

                break;

            }

        }

    }

    public void updateDropdowns() {

        // Null checks

        if (serverDropdown == null || alertTypeDropdown == null || servers == null) {

            System.err.println("Dropdowns or servers list are not initialized.");

            return;

        }

        // Clear existing items

        serverDropdown.removeAllItems();

        alertTypeDropdown.removeAllItems();

        // Add 'All Servers' and 'All Alerts' options

        serverDropdown.addItem("All Servers");

        alertTypeDropdown.addItem("All Alerts");

        // Populate serverDropdown with server names

        for (Server server : servers) {

            serverDropdown.addItem(server.getName());

        }

        // Populate alertTypeDropdown with alert types

        alertTypeDropdown.addItem("CPU Alert");

        alertTypeDropdown.addItem("Memory Alert");

        // Add more alert types as needed

    }

    public JComboBox<String> getServerDropdown() {

        return this.serverDropdown;

    }

    public JComboBox<String> getAlertTypeDropdown() {

        return this.alertTypeDropdown;

    }

}

DashboardPanel

package main.java.ui;

import main.java.main.Main;

import main.java.models.AbstractUser;

import main.java.models.Server;

import javax.swing.\*;

import javax.swing.table.DefaultTableModel;

import java.awt.\*;

import java.util.ArrayList;

public class DashboardPanel extends JPanel {

    private JLabel welcomeLabel;

    private JTable serverTable;

    private JPanel parentPanel;

    private DefaultTableModel tableModel;

    private ArrayList<Server> servers;

    private AbstractUser currentUser;

    private ArrayList<AbstractUser> users;

    private GridBagConstraints gbc = new GridBagConstraints();

    private ServerManagement serverManagement;

    public DashboardPanel(AbstractUser currentUser, JPanel parentPanel) {

        setLayout(new GridBagLayout());

        this.currentUser = currentUser;

        this.parentPanel = parentPanel;

        // Initialize tableModel

        String[] columnNames;

        if ("admin".equals(currentUser.getRole())) {

            columnNames = new String[]{"Server Name", "CPU Usage", "Memory Usage", "Network Latency", "Actions"};

        } else {

            columnNames = new String[]{"Server Name", "CPU Usage", "Memory Usage", "Network Latency"};

        }

        // Initialize tableModel and servers

        this.tableModel = new DefaultTableModel(columnNames, 0);

        this.servers = new ArrayList<>();

        // Initialize ServerManagement with the tableModel and servers

        if (this.tableModel != null) {

            this.serverManagement = new ServerManagement(this.servers, this.tableModel);  // Make sure to initialize here

            if (this.serverManagement != null) {

                this.servers = serverManagement.initializeServers();  // Assuming initializeServers is a method in ServerManagement

            } else {

                System.err.println("serverManagement is null. Aborting.");

                return;

            }

        } else {

            System.err.println("tableModel is null. Cannot initialize serverManagement.");

            return;

        }

        //this.servers = initializeServers();

        if (this.servers == null) {

            System.err.println("servers is null. Aborting.");

            return;  // Exit if servers is null

        }

        // Initialize UI Components

        initializeUI();

        // Initialize ServerManagement with the tableModel and servers

        this.serverManagement = new ServerManagement(this.servers, this.tableModel);

        //System.out.println("Dashboard Server object: "+servers.size());

    }

    private void initializeUI() {

        // Welcome Label

        welcomeLabel = new JLabel("Welcome to the SMS based Remote Server Monitoring System!");

        welcomeLabel.setHorizontalAlignment(SwingConstants.CENTER);

        gbc.gridx = 0;

        gbc.gridy = 0;

        gbc.gridwidth = 2;

        gbc.weightx = 1.0;

        gbc.weighty = 0.1;

        gbc.insets = new Insets(10, 10, 10, 10);

        add(welcomeLabel, gbc);

        // Initialize Table

        initializeTable();

        // Initialize Timer

        initializeTimer();

        // Initialize Dashboard

        initializeDashboard();

        // Logout Button

        initializeLogoutButton();

        // Populate the table initially

        serverManagement.simulateServerMonitoring(servers);

    }

    private void initializeTable() {

        serverTable = new JTable(tableModel);

        serverTable.setPreferredScrollableViewportSize(new Dimension(500, 70));  // Set the preferred size

        gbc.gridx = 0;

        gbc.gridy = 1;

        gbc.weighty = 0.9;

        gbc.fill = GridBagConstraints.BOTH;

        if ("admin".equals(currentUser.getRole())) {

            serverTable.getColumn("Actions").setCellEditor(serverManagement.new SimpleButtonEditor(serverManagement));

            serverTable.getColumn("Actions").setCellRenderer(new ServerManagement.SimpleButtonRenderer());

        }

        JScrollPane scrollPane = new JScrollPane(serverTable);

        scrollPane.setPreferredSize(new Dimension(500, 200));  // Optional: set JScrollPane size

        add(scrollPane, gbc);

    }

    private void initializeTimer() {

        Timer timer = new Timer(5000, e -> {

            //System.out.println("Timer is called");  // Debug print

            tableModel.setRowCount(0);

            serverManagement.simulateServerMonitoring(servers);

        });

        timer.start();

    }

    private void initializeDashboard() {

        if (currentUser != null) {

            if ("admin".equals(currentUser.getRole())) {

                DefaultTableModel tableModel = new DefaultTableModel();  // Initialize this as needed

                AdminDashboard adminDashboard = new AdminDashboard(servers, users, gbc, this, this,tableModel);

                adminDashboard.initializeAdminDashboard(this);

            } else {

                UserDashboard userDashboard = new UserDashboard(servers, users, gbc, this, this);

                userDashboard.initializeUserDashboard(this);

            }

        }

    }

    private void initializeLogoutButton() {

        JButton logoutButton = new JButton("Logout");

        logoutButton.addActionListener(e -> {

            for (Frame frame : Frame.getFrames()) {

                frame.dispose();

            }

            Main.createAndShowGUI();

        });

        add(logoutButton);

    }

    public void refreshServers() {

        this.servers = serverManagement.initializeServers(); // Re-initialize servers

        tableModel.setRowCount(0); // Clear the table

        serverManagement.simulateServerMonitoring(servers);

    }

}

LoginPanel.java

package main.java.ui;

import main.java.models.AbstractUser;

import main.java.utils.UserDatabase;

import main.java.utils.FileHandler;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.KeyEvent;

import java.awt.event.KeyListener;

import java.util.List;

public class LoginPanel extends JPanel {

    private JLabel welcomeLabel;

    private JLabel userLabel;

    private JTextField userText;

    private JLabel passwordLabel;

    private JPasswordField passwordText;

    private JButton loginButton;

    public LoginPanel() {

        setLayout(new GridBagLayout());  // Use GridBagLayout

        GridBagConstraints gbc = new GridBagConstraints();

        welcomeLabel = new JLabel("Welcome to the Smart Server Management System");

        gbc.gridx = 0;

        gbc.gridy = 0;

        gbc.gridwidth = 2;

        gbc.insets = new Insets(10, 10, 10, 10);

        add(welcomeLabel, gbc);

        userLabel = new JLabel("Username");

        gbc.gridx = 0;

        gbc.gridy = 1;

        gbc.gridwidth = 1;

        add(userLabel, gbc);

        userText = new JTextField(20);

        gbc.gridx = 1;

        gbc.gridy = 1;

        add(userText, gbc);

        passwordLabel = new JLabel("Password");

        gbc.gridx = 0;

        gbc.gridy = 2;

        add(passwordLabel, gbc);

        passwordText = new JPasswordField(20);

        gbc.gridx = 1;

        gbc.gridy = 2;

        add(passwordText, gbc);

        loginButton = new JButton("Login");

        gbc.gridx = 0;

        gbc.gridy = 3;

        gbc.gridwidth = 2;

        add(loginButton, gbc);

        loginButton.addActionListener(new ActionListener() {

            @Override

            public void actionPerformed(ActionEvent e) {

                handleLogin();

            }

        });

        // Add key listener to passwordText

        passwordText.addKeyListener(new KeyListener() {

            @Override

            public void keyTyped(KeyEvent e) {

            }

            @Override

            public void keyPressed(KeyEvent e) {

                if (e.getKeyCode() == KeyEvent.VK\_ENTER) {

                    handleLogin();

                }

            }

            @Override

            public void keyReleased(KeyEvent e) {

            }

        });

    }

    private void handleLogin() {

        String username = userText.getText();

        String password = new String(passwordText.getPassword());

        // First, try to authenticate using UserDatabase

        AbstractUser currentUser = UserDatabase.authenticate(username, password);

        // If authentication fails, try to authenticate using users.txt

        if (currentUser == null) {

            //List<AbstractUser> usersFromFile = FileHandler.readUsersFromTextFile("users.txt");

            List<AbstractUser> usersFromFile = FileHandler.readUsersFromCSVFile("users.csv");

            for (AbstractUser user : usersFromFile) {

                if (user.getUsername().equals(username) && user.getPassword().equals(password)) {

                    currentUser = user;

                    break;

                }

            }

        }

        // Check if authentication was successful

        if (currentUser != null) {

            JOptionPane.showMessageDialog(this, "Login successful!", "Success", JOptionPane.INFORMATION\_MESSAGE);

            // Switch to the dashboard

            JPanel parentPanel = new JPanel();  // Or some existing JPanel instance

            JFrame mainFrame = (JFrame) SwingUtilities.getWindowAncestor(this);

            mainFrame.getContentPane().removeAll();

            mainFrame.add(new DashboardPanel(currentUser,parentPanel));  // Pass currentUser here

            mainFrame.revalidate();

            mainFrame.repaint();

        } else {

            JOptionPane.showMessageDialog(this, "Invalid credentials.", "Error", JOptionPane.ERROR\_MESSAGE);

        }

    }

}

ManageServers.java

package main.java.ui;

import main.java.models.Server;

import main.java.utils.FileHandler;

import javax.swing.\*;

import javax.swing.table.DefaultTableModel;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.util.ArrayList;

public class ManageServers extends JPanel {

    private JTable serverTable;

    private DefaultTableModel tableModel;

    private ArrayList<Server> servers;

    private ServerManagement serverManagement;  // Added this line

    private AdminDashboard adminDashboard;  // Declare the field

    public ManageServers(ArrayList<Server> servers, ServerManagement serverManagement, AdminDashboard adminDashboard) {  // Added ServerManagement parameter

        this.servers = servers;

        this.serverManagement = serverManagement;  // Initialize the ServerManagement instance

        this.adminDashboard = adminDashboard;  // Initialize the AdminDashboard instance

        setLayout(new BorderLayout());

        String[] columnNames = {"Server Name", "CPU Limit", "Memory Limit", "Network Limit"};

        this.tableModel = new DefaultTableModel(columnNames, 0);

        for (Server server : this.servers) {

            Object[] rowData = {

                    server.getName(),

                    server.getCpuThreshold(),

                    server.getMemoryThreshold(),

                    server.getNetworkThreshold()

            };

            this.tableModel.addRow(rowData);

        }

        serverTable = new JTable(this.tableModel);

        JButton addButton = new JButton("Add");

        JButton editButton = new JButton("Edit");

        JButton removeButton = new JButton("Remove");

        addButton.addActionListener(new AddServerAction());

        editButton.addActionListener(new EditServerAction());

        removeButton.addActionListener(new RemoveServerAction());

        JPanel buttonPanel = new JPanel();

        buttonPanel.add(addButton);

        buttonPanel.add(editButton);

        buttonPanel.add(removeButton);

        add(new JScrollPane(serverTable), BorderLayout.CENTER);

        add(buttonPanel, BorderLayout.SOUTH);

    }

    private class AddServerAction implements ActionListener {

        @Override

        public void actionPerformed(ActionEvent e) {

            String name = JOptionPane.showInputDialog("Enter Server Name:");

            if (name == null) return; // User cancelled, exit the method

            String cpuLimitStr = JOptionPane.showInputDialog("Enter CPU Limit:");

            if (cpuLimitStr == null) return; // User cancelled, exit the method

            String memoryLimitStr = JOptionPane.showInputDialog("Enter Memory Limit:");

            if (memoryLimitStr == null) return; // User cancelled, exit the method

            String networkLimitStr = JOptionPane.showInputDialog("Enter Network Limit:");

            if (networkLimitStr == null) return; // User cancelled, exit the method

            try {

                int cpuLimit = Integer.parseInt(cpuLimitStr);

                int memoryLimit = Integer.parseInt(memoryLimitStr);

                int networkLimit = Integer.parseInt(networkLimitStr);

                //Server newServer = new Server(name, cpuLimit, memoryLimit, networkLimit);

                //servers.add(newServer);

                Object[] rowData = {name, cpuLimit, memoryLimit, networkLimit};

                tableModel.addRow(rowData);

                // Use ServerManagement's addServer method

                serverManagement.addServer(name, cpuLimit, memoryLimit, networkLimit);

                FileHandler.writeServersToCSVFile("servers.csv",servers);  // Save to CSV

                populateServerTable();  // <-- Add this line

                // Refresh the AdminDashboard table

                adminDashboard.refreshTable();

            } catch (NumberFormatException ex) {

                JOptionPane.showMessageDialog(null, "Invalid input. Please enter numerical values for limits.");

            }

        }

    }

    private class EditServerAction implements ActionListener {

        private String validateInput(String prompt, String defaultValue) {

            String input = JOptionPane.showInputDialog(prompt, defaultValue);

            if (input == null || input.isEmpty()) {

                JOptionPane.showMessageDialog(null, "Input cannot be empty. Please try again.");

                return null; // or throw an exception, or use a default value

            }

            return input;

        }

        @Override

        public void actionPerformed(ActionEvent e) {

            try {

                int selectedRow = serverTable.getSelectedRow();

                if (selectedRow >= 0 && selectedRow < servers.size()) {

                    Server serverToEdit = servers.get(selectedRow);

                    String newName = validateInput("Enter new Server Name:", serverToEdit.getName());

                    String newCpuLimitStr = validateInput("Enter new CPU Limit:", String.valueOf(serverToEdit.getCpuThreshold()));

                    String newMemoryLimitStr = validateInput("Enter new Memory Limit:", String.valueOf(serverToEdit.getMemoryThreshold()));

                    String newNetworkLimitStr = validateInput("Enter new Network Limit:", String.valueOf(serverToEdit.getNetworkThreshold()));

                    int newCpuLimit = Integer.parseInt(newCpuLimitStr);

                    int newMemoryLimit = Integer.parseInt(newMemoryLimitStr);

                    int newNetworkLimit = Integer.parseInt(newNetworkLimitStr);

                    serverToEdit.setName(newName);

                    serverToEdit.setCpuThreshold(newCpuLimit);  // Assuming you have a setter for this

                    serverToEdit.setMemoryThreshold(newMemoryLimit);  // Assuming you have a setter for this

                    serverToEdit.setNetworkThreshold(newNetworkLimit);  // Assuming you have a setter for this

                    FileHandler.writeServersToCSVFile("servers.csv", servers);  // Save to CSV

                    populateServerTable();  // Refresh the table

                    adminDashboard.refreshTable();

                } else {

                    JOptionPane.showMessageDialog(null, "Please select a server to edit.");

                }

            } catch (NumberFormatException ex) {

                JOptionPane.showMessageDialog(null, "Invalid input. Please enter numerical values for limits.");

            } catch (Exception ex) {

                JOptionPane.showMessageDialog(null, "An unexpected error occurred. Please try again.");

            }

        }

    }

    // New method to populate the server table

    private void populateServerTable() {

        tableModel.setRowCount(0);

        for (Server server : this.servers) {

            Object[] rowData = {

                    server.getName(),

                    server.getCpuThreshold(),

                    server.getMemoryThreshold(),

                    server.getNetworkThreshold()

            };

            tableModel.addRow(rowData);

        }

    }

    private class RemoveServerAction implements ActionListener {

        @Override

        public void actionPerformed(ActionEvent e) {

            int selectedRow = serverTable.getSelectedRow();

            if (selectedRow >= 0 && selectedRow < servers.size()) {

                servers.remove(selectedRow);

                // Update the CSV file

                FileHandler.writeServersToCSVFile("servers.csv", servers);  // Save to CSV

                populateServerTable();  // <-- Add this line

                adminDashboard.refreshTable();

                //System.out.println("Servers after deletion: " + servers);  // Debugging line

            } else {

                JOptionPane.showMessageDialog(null, "Please select a server to remove.");

            }

        }

    }

}

ManageUsers.java

package main.java.ui;

import main.java.models.AbstractUser;

import main.java.models.Admin;

import main.java.models.User;

import main.java.utils.FileHandler;

import javax.swing.\*;

import javax.swing.table.DefaultTableModel;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.util.ArrayList;

import java.util.List;

import java.util.regex.Pattern;

public class ManageUsers extends JPanel {

    private JTable userTable;

    private DefaultTableModel tableModel;

    private ArrayList<AbstractUser> users;

    public ManageUsers(List<? extends AbstractUser> users) {

        this.users = new ArrayList<>(FileHandler.readUsersFromCSVFile("users.csv"));

        setLayout(new BorderLayout());

        String[] columnNames = {"Username", "Role", "Phone Number"};

        tableModel = new DefaultTableModel(columnNames, 0);

        populateTable();

        userTable = new JTable(tableModel);

        JButton addButton = new JButton("Add");

        JButton editButton = new JButton("Edit");

        JButton removeButton = new JButton("Remove");

        addButton.addActionListener(new AddUserAction());

        editButton.addActionListener(new EditUserAction());

        removeButton.addActionListener(new RemoveUserAction());

        JPanel buttonPanel = new JPanel();

        buttonPanel.add(addButton);

        buttonPanel.add(editButton);

        buttonPanel.add(removeButton);

        add(new JScrollPane(userTable), BorderLayout.CENTER);

        add(buttonPanel, BorderLayout.SOUTH);

    }

    private void populateTable() {

        tableModel.setRowCount(0);

        for (AbstractUser user : this.users) {

            Object[] rowData = {user.getUsername(), user.getRole(), user.getPhoneNumber()};

            tableModel.addRow(rowData);

        }

    }

    private String validateInput(String prompt, String defaultValue) {

        String input;

        do {

            input = JOptionPane.showInputDialog(null, prompt, defaultValue);

            if (input == null) {

                return null;  // User clicked "X" or "Close"

            }

            if (!Pattern.matches("\\s\*", input)) {

                return input;

            } else {

                JOptionPane.showMessageDialog(null, "Should not be blank or contain only whitespace.");

            }

        } while (true);

    }

    private class AddUserAction implements ActionListener {

        @Override

        public void actionPerformed(ActionEvent e) {

            String username = validateInput("Enter Username:", "");

            String password = validateInput("Enter Password:", "");

            String role = validateInput("Enter Role (admin/user):", "");

            String phoneNumber = validateInput("Enter Phone Number:", "");

            // Check for null values

            if (username == null || password == null || role == null || phoneNumber == null) {

                JOptionPane.showMessageDialog(null, "Operation cancelled or incomplete data.");

                return;

            }

            // Check for duplicate username

            boolean isDuplicate = users.stream().anyMatch(user -> user.getUsername().equals(username));

            if (isDuplicate) {

                JOptionPane.showMessageDialog(null, "Username already exists.");

                return; // Exit the method if username is duplicate

            }

            AbstractUser newUser;

            if ("admin".equals(role)) {

                newUser = new Admin(username, password, role, phoneNumber, null);

            } else {

                newUser = new User(username, password, role, phoneNumber, null);

            }

            addUser(newUser);

            populateTable();

        }

    }

    private class EditUserAction implements ActionListener {

        @Override

        public void actionPerformed(ActionEvent e) {

            int selectedRow = userTable.getSelectedRow();

            if (selectedRow >= 0 && selectedRow < users.size()) {

                AbstractUser user = users.get(selectedRow);

                String newUsername = validateInput("Enter new Username:",user.getUsername());

                String newPassword = validateInput("Enter new Password:", user.getPassword());

                String newRole = validateInput("Enter new Role (admin/user):", user.getRole());

                String newPhoneNumber = validateInput("Enter new Phone Number:", user.getPhoneNumber());

                user.setUsername(newUsername);

                user.setPassword(newPassword);

                user.setRole(newRole);

                user.setPhoneNumber(newPhoneNumber);

                populateTable();

                // Update the CSV file

                FileHandler.writeUsersToCSVFile("users.csv", users);

            } else {

                JOptionPane.showMessageDialog(null, "Please select a user to edit.");

            }

        }

    }

    private class RemoveUserAction implements ActionListener {

        @Override

        public void actionPerformed(ActionEvent e) {

            int selectedRow = userTable.getSelectedRow();

            if (selectedRow >= 0 && selectedRow < users.size()) {

                users.remove(selectedRow);

                populateTable();

                // Update the CSV file

                FileHandler.writeUsersToCSVFile("users.csv", users);

            } else {

                JOptionPane.showMessageDialog(null, "Please select a user to remove.");

            }

        }

    }

    private void addUser(AbstractUser newUser) {

        boolean isDuplicate = false;

        for (AbstractUser user : users) {

            if (user.getUsername().equals(newUser.getUsername())) {

                isDuplicate = true;

                break;

            }

        }

        if (!isDuplicate) {

            users.add(newUser);

            Object[] rowData = {newUser.getUsername(), newUser.getRole(), newUser.getPhoneNumber()};

            tableModel.addRow(rowData);

            FileHandler.appendUserToCSVFile("users.csv", newUser);

        } else {

            JOptionPane.showMessageDialog(null, "Username already exists.");

        }

    }

}

ServerManagement.java

package main.java.ui;

import main.java.models.Server;

import main.java.utils.FileHandler;

import main.java.utils.SMSHandler;

import javax.swing.\*;

import javax.swing.table.DefaultTableModel;

import javax.swing.table.TableCellRenderer;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.util.ArrayList;

public class ServerManagement {

    private ArrayList<Server> servers;

    private DefaultTableModel tableModel;

    public ServerManagement(ArrayList<Server> servers, DefaultTableModel tableModel) {

        this.servers = servers;

        this.tableModel = tableModel;

        //System.out.println("Server object 1: "+servers.size());

        if (tableModel == null) {

            System.err.println("tableModel is null. Aborting simulation.");

            return;

        }

        //this.servers = initializeServers();  // <-- Add this line here

        //System.out.println("Server object 2: "+servers.size());

    }

    public void simulateServerMonitoring(ArrayList<Server> servers) {

        this.servers = servers;

        if (servers == null || tableModel == null) {

            System.err.println("Either servers or tableModel is null. Aborting simulation.");

            return;

        }

        tableModel.setRowCount(0);

        for (Server server : servers) {

            server.simulateMonitoring();

            // Create a new JButton for each row

            //JButton restartButton = new JButton("Restart");

            //restartButton.setActionCommand("Restart\_" + server.getName());  // Set action command

            Object[] rowData = {

                    server.getName(),

                    String.format("%.2f", server.getCpuUsage()),

                    String.format("%.2f", server.getMemoryUsage()),

                    String.format("%.2f", server.getNetworkLatency()),

                    "Restart"  // Add a "Restart" action here

            };

            tableModel.addRow(rowData);

            FileHandler.writeLog("server\_metrics.txt",

                    String.format("%s: CPU=%.2f, Memory=%.2f, Latency=%.2f",

                            server.getName(),

                            server.getCpuUsage(),

                            server.getMemoryUsage(),

                            server.getNetworkLatency())

            );

        }

        checkServerThresholds();

        tableModel.fireTableDataChanged();

    }

    public void restartServer(Server server) {

        server.restart();

    }

    public void checkServerThresholds() {

        for (Server server : servers) {

            String message = server.checkThresholds();

            if (!message.isEmpty()) {

                SMSHandler.sendSMS("123-456-7890", message);

                FileHandler.writeLog("server\_alerts.txt", server.getName() + ": " + message);

            }

        }

    }

    public void addServer(String name, int cpuLimit, int memoryLimit, int networkLimit) {

        Server newServer = new Server(name, cpuLimit, memoryLimit, networkLimit);

        servers.add(newServer);

        Object[] rowData = {name, cpuLimit, memoryLimit, networkLimit};

        tableModel.addRow(rowData);

        tableModel.fireTableDataChanged();

    }

    public void editServer(int selectedRow, String name, int cpuLimit, int memoryLimit, int networkLimit) {

        if (selectedRow >= 0 && selectedRow < servers.size()) {

            Server server = servers.get(selectedRow);

            server.setName(name);

            tableModel.setValueAt(name, selectedRow, 0);

            tableModel.fireTableDataChanged();

        } else {

            JOptionPane.showMessageDialog(null, "Invalid row selected for editing.");

        }

    }

    public static class SimpleButtonRenderer extends JButton implements TableCellRenderer {

        public Component getTableCellRendererComponent(JTable table, Object value, boolean isSelected, boolean hasFocus, int row, int column) {

            setText("Restart");

            return this;

        }

    }

    // Updated inner class to non-static and added a reference to the ServerManagement instance

    public class SimpleButtonEditor extends DefaultCellEditor implements ActionListener {

        private JButton button;

        private int row;

        private JTable table;

        private ServerManagement serverManagementInstance;  // Add this line

        public SimpleButtonEditor(ServerManagement serverManagementInstance) {

            super(new JCheckBox());

            this.serverManagementInstance = serverManagementInstance;  // Initialize it

            button = new JButton("Restart");

        }

        @Override

        public Component getTableCellEditorComponent(JTable table, Object value, boolean isSelected, int row, int column) {

            this.row = row;

            this.table = table;

            for (ActionListener al : button.getActionListeners()) {

                button.removeActionListener(al);

            }

            button.addActionListener(this);

            return button;

        }

        @Override

        public void actionPerformed(ActionEvent e) {

            //System.out.println("Restart button clicked for row: " + row);

            Server server = serverManagementInstance.servers.get(row);  // Access it via instance

            serverManagementInstance.restartServer(server);  // Access it via instance

            // Update the table row with the new server metrics

            tableModel.setValueAt(String.format("%.2f", server.getCpuUsage()), row, 1);

            tableModel.setValueAt(String.format("%.2f", server.getMemoryUsage()), row, 2);

            tableModel.setValueAt(String.format("%.2f", server.getNetworkLatency()), row, 3);

            tableModel.fireTableRowsUpdated(row, row);  // Notify that the data for the specific row has changed

            fireEditingStopped();

        }

    }

    public ArrayList<Server> initializeServers() {

        return new ArrayList<>(FileHandler.readServersFromCSVFile("servers.csv"));

    }

}

UserManagement.java

package main.java.ui;

import main.java.models.Server;

import main.java.models.AbstractUser;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.FocusEvent;

import java.awt.event.FocusAdapter;

import java.util.ArrayList;

import javax.swing.table.DefaultTableModel;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class UserDashboard extends JPanel {

    private JLabel welcomeLabel;

    private JTable serverTable;

    private DefaultTableModel tableModel;

    private AbstractUser currentUser;

    private GridBagConstraints gbc = new GridBagConstraints();

    private AlertManagement alertManagement;

    private ArrayList<Server> servers;

    private ArrayList<AbstractUser> users;

    private JPanel parentPanel;

    private DashboardPanel dashboardPanel;

    private Server selectedServer = null;

    public UserDashboard(ArrayList<Server> servers, ArrayList<AbstractUser> users, GridBagConstraints gbc, JPanel parentPanel, DashboardPanel dashboardPanel) {

        if(servers == null) {

            throw new IllegalArgumentException("Servers list cannot be null");

        }

        setLayout(new GridBagLayout());

        this.servers = servers;

        this.users = users;

        this.gbc = gbc;

        this.parentPanel = parentPanel;

        this.dashboardPanel = dashboardPanel;

        String[] columnNames = {"Server Name", "CPU Usage", "Memory Usage", "Network Latency"};

        tableModel = new DefaultTableModel(columnNames, 0);

        serverTable = new JTable(tableModel);

        serverTable.setSelectionMode(ListSelectionModel.SINGLE\_SELECTION);

        serverTable.setFocusable(true);

        serverTable.setRowSelectionAllowed(true);

        for (Server server : servers) {

            Object[] rowData = {server.getName(), server.getCpuUsage(), server.getMemoryUsage(), server.getNetworkLatency()};

            tableModel.addRow(rowData);

        }

        JScrollPane scrollPane = new JScrollPane(serverTable);

        gbc.gridx = 0;

        gbc.gridy = 0;

        gbc.weighty = 0.1;

        gbc.fill = GridBagConstraints.HORIZONTAL;

        this.add(scrollPane, gbc);

        // Debug: Check if table model is populated correctly

        //System.out.println("Debug: Table model row count: " + tableModel.getRowCount());

        // Focus Debugging

        serverTable.addFocusListener(new FocusAdapter() {

            public void focusGained(FocusEvent e) {

                System.out.println("Table gained focus");

            }

            public void focusLost(FocusEvent e) {

                System.out.println("Table lost focus");

            }

        });

    }

    public void initializeUserDashboard(DashboardPanel dashboardPanel) {

        System.out.println("Initializing User Dashboard");

        // Debug: Check if table model is populated correctly now

        //System.out.println("Debug: Table model row count: " + tableModel.getRowCount());

        this.dashboardPanel = dashboardPanel;

        JPanel userPanel = new JPanel(new FlowLayout());

        this.alertManagement = new AlertManagement(servers);

        alertManagement.updateDropdowns();

        // Initialize the Request Restart Button with the new ActionListener

        initializeRequestRestartButton(userPanel);

        initializeSubscribeAlertsButton(userPanel);

        addDropdownsToPanel(gbc);

        gbc.gridx = 0;

        gbc.gridy = 4;

        gbc.weighty = 0.1;

        gbc.fill = GridBagConstraints.HORIZONTAL;

        parentPanel.add(userPanel, gbc);

        //System.out.println("Table Row Count: " + serverTable.getRowCount());

    }

    // Inside initializeRequestRestartButton method

    public void initializeRequestRestartButton(JPanel userPanel) {

        JButton requestRestartButton = new JButton("Request Server Restart");

        userPanel.add(requestRestartButton);

        requestRestartButton.addActionListener(new ActionListener() {

            @Override

            public void actionPerformed(ActionEvent e) {

                JDialog serverSelectDialog = new JDialog();

                serverSelectDialog.setTitle("Select Server to Restart");

                serverSelectDialog.setLayout(new GridBagLayout());

                GridBagConstraints gbc = new GridBagConstraints();

                JComboBox<Object> serverDropdown = new JComboBox<>();

                serverDropdown.addItem("All Servers");

                for (Server server : servers) {

                    serverDropdown.addItem(server);

                }

                JButton submitButton = new JButton("Submit");

                submitButton.addActionListener(new ActionListener() {

                    @Override

                    public void actionPerformed(ActionEvent e) {

                        Object selected = serverDropdown.getSelectedItem();

                        if ("All Servers".equals(selected)) {

                            for (Server server : servers) {

                                requestServerRestart(server);

                            }

                        } else {

                            Server selectedServer = (Server) selected;

                            if (selectedServer != null) {

                                requestServerRestart(selectedServer);

                            }

                        }

                        serverSelectDialog.dispose();

                    }

                });

                gbc.insets = new Insets(10, 10, 10, 10);  // Padding

                gbc.gridx = 0;

                gbc.gridy = 0;

                gbc.anchor = GridBagConstraints.WEST;

                serverSelectDialog.add(new JLabel("Select Server:"), gbc);

                gbc.gridx = 1;

                serverSelectDialog.add(serverDropdown, gbc);

                gbc.gridx = 2;

                gbc.gridy = 0;  // Changed from 1 to 2 to add more space

                gbc.gridwidth = 1;

                gbc.anchor = GridBagConstraints.EAST;

                serverSelectDialog.add(submitButton, gbc);

                serverSelectDialog.setPreferredSize(new Dimension(400, 150));  // Set the preferred size

                serverSelectDialog.pack();

                serverSelectDialog.setLocationRelativeTo(null);  // Center the dialog on the screen

                serverSelectDialog.setVisible(true);

            }

        });

    }

    public void initializeSubscribeAlertsButton(JPanel userPanel) {

        JButton subscribeAlertsButton = new JButton("Subscribe to Alerts");

        subscribeAlertsButton.addActionListener(e -> alertManagement.subscribeToAlerts());

        userPanel.add(subscribeAlertsButton);

    }

    private void addDropdownsToPanel(GridBagConstraints gbc) {

        add(alertManagement.getServerDropdown(), gbc);

        add(alertManagement.getAlertTypeDropdown(), gbc);

    }

    private void requestServerRestart() {

        Server serverToRestart = servers.get(0);

        requestServerRestart(serverToRestart);

    }

    private void requestServerRestart(Server serverToRestart) {

        int dialogResult = JOptionPane.showConfirmDialog(this,

                "Do you want to submit restart for " + serverToRestart.getName() + "?",

                "Confirm Request",

                JOptionPane.YES\_NO\_OPTION);

        if (dialogResult == JOptionPane.YES\_OPTION) {

            //serverToRestart.restart();

            sendRestartRequestToAdmin(serverToRestart);

        }

    }

    private void sendRestartRequestToAdmin(Server server) {

        System.out.println("Restart request for " + server.getName() + " sent to admin.");

    }

}

FileHandler.java

package main.java.utils;

import main.java.models.AbstractUser;

import main.java.models.Admin;

import main.java.models.User;

import main.java.models.Server;

import java.io.\*;

import java.util.ArrayList;

import java.util.HashSet;

import java.util.List;

import java.util.Set;

public class FileHandler {

    // Save Serializable data to a file

    public static void saveToFile(String filename, Serializable data) throws IOException {

        try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(filename))) {

            oos.writeObject(data);

        }

    }

    // Read Serializable data from a file

    public static Object readFromFile(String filename) throws IOException, ClassNotFoundException {

        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(filename))) {

            return ois.readObject();

        }

    }

    // Write log data to a file

    public static void writeLog(String fileName, String content) {

        try (PrintWriter out = new PrintWriter(new BufferedWriter(new FileWriter(fileName, true)))) {

            out.println(content);

        } catch (IOException e) {

            handleException("Error writing log to file", e);

        }

    }

    // Write users to a text file

    public static void writeUsersToTextFile(String filename, List<AbstractUser> users) throws IOException {

        try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename))) {

            for (AbstractUser user : users) {

                writer.write(formatUser(user));

                writer.newLine();

            }

        }

    }

    // Read users from a text file

    public static List<AbstractUser> readUsersFromTextFile(String filename) {

        return readUsersFromFile(filename, "text");

    }

    // Write users to a CSV file

    public static void writeUsersToCSVFile(String filename, List<AbstractUser> users) {

        Set<AbstractUser> uniqueUsers = new HashSet<>(users); // Create a set of unique users

        try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename))) {

            for (AbstractUser user : uniqueUsers) {

                writer.write(formatUser(user));

                writer.newLine();

            }

        } catch (IOException e) {

            handleException("Error writing users to file", e);

        }

    }

    // Read users from a CSV file

    public static List<AbstractUser> readUsersFromCSVFile(String filename) {

        return readUsersFromFile(filename, "CSV");

    }

    private static String formatUser(AbstractUser user) {

        return user.getUsername() + "," + user.getPassword() + "," + user.getRole() + "," + user.getPhoneNumber();

    }

    private static List<AbstractUser> readUsersFromFile(String filename, String fileType) {

        List<AbstractUser> users = new ArrayList<>();

        File file = new File(filename);

        // Check if the file exists; if not, create it

        if (!file.exists()) {

            handleException(filename + " not found. Creating an empty file.", null);

            try {

                file.createNewFile();

            } catch (IOException e) {

                handleException("Error creating new file " + filename, e);

            }

            return users; // Return an empty list as the file was not there

        }

        // Read the file

        try (BufferedReader reader = new BufferedReader(new FileReader(file))) {

            String line;

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

                addUserFromLine(line, users);

            }

        } catch (IOException e) {

            handleException("Error reading users from " + fileType + " file", e);

        }

        return users;

    }

    private static void addUserFromLine(String line, List<AbstractUser> users) {

        String[] parts = line.split(",");

        if (parts.length >= 4) {  // Make sure you have at least 4 parts

            String username = parts[0];

            String password = parts[1];

            String role = parts[2];

            String phoneNumber = parts[3];  // Make sure this is not null or empty

            AbstractUser user = createUser(username, password, role, phoneNumber);

            if (user != null) {

                users.add(user);

            }

        }

    }

    private static AbstractUser createUser(String username, String password, String role, String phoneNumber) {

        if ("admin".equals(role)) {

            return new Admin(username, password, role, phoneNumber, null);

        } else {

            return new User(username, password, role, phoneNumber, null);

        }

    }

    public static void handleException(String message, Exception e) {

        if (message != null) {

            System.err.println(message);

        }

        if (e != null) {

            e.printStackTrace();

        }

    }

    // Append a new user to a CSV file

    public static void appendUserToCSVFile(String filename, AbstractUser newUser) {

        List<AbstractUser> existingUsers = readUsersFromCSVFile(filename);

        // Check for duplicate users based on username

        for (AbstractUser user : existingUsers) {

            if (user.getUsername().equals(newUser.getUsername())) {

                handleException("User already exists", null);

                return;

            }

        }

        // Append the new user to the file

        try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename, true))) {

            // Check if this is the first line in the file

            if (existingUsers.size() > 0) {

                writer.newLine();

            }

            writer.write(formatUser(newUser));

        } catch (IOException e) {

            handleException("Error appending user to file", e);

        }

    }

    // Write servers to a CSV file

    public static void writeServersToCSVFile(String filename, ArrayList<Server> servers) {

        try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename))) {

            // Write the header

            //writer.write("Server Name,CPU Limit,Memory Limit,Network Limit");

            writer.newLine();

            // Write the server data

            for (Server server : servers) {

                writer.write(server.getName() + "," + server.getCpuThreshold() + "," + server.getMemoryThreshold() + "," + server.getNetworkThreshold());

                writer.newLine();

            }

        } catch (IOException e) {

            handleException("Error writing servers to CSV file", e);

        }

    }

    // Read servers from a CSV file

    public static List<Server> readServersFromCSVFile(String filename) {

        List<Server> servers = new ArrayList<>();

        File file = new File(filename);

        if (!file.exists()) {

            handleException(filename + " not found. Creating an empty file.", null);

            try {

                file.createNewFile();

            } catch (IOException e) {

                handleException("Error creating new file " + filename, e);

            }

            return servers;

        }

        try (BufferedReader reader = new BufferedReader(new FileReader(file))) {

            String line;

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

                addServerFromLine(line, servers);

            }

        } catch (IOException e) {

            handleException("Error reading servers from CSV file", e);

        }

        return servers;

    }

    private static String formatServer(Server server) {

        return server.getName() + "," + server.getCpuThreshold() + "," + server.getMemoryThreshold() + "," + server.getNetworkThreshold();

    }

    private static void addServerFromLine(String line, List<Server> servers) {

        String[] parts = line.split(",");

        if (parts.length >= 4) {

            String name = parts[0];

            int cpuThreshold = Integer.parseInt(parts[1]);

            int memoryThreshold = Integer.parseInt(parts[2]);

            int networkThreshold = Integer.parseInt(parts[3]);

            Server server = new Server(name, cpuThreshold, memoryThreshold, networkThreshold);

            servers.add(server);

        }

    }

}

SMSHandler.java

package main.java.utils;

public class SMSHandler {

    public static void sendSMS(String phoneNumber, String message) {

        // Simulate sending an SMS

        System.out.println("SMS sent to " + phoneNumber + ": " + message);

    }

}

UserDatabase.java

package main.java.utils;

import main.java.models.AbstractUser;

import main.java.models.Admin;

import main.java.models.User;

import java.util.ArrayList;

import java.util.List;

public class UserDatabase {

    public static List<AbstractUser> users = new ArrayList<>();

    static {

        users.add(new Admin("admin", "123", "admin", "123-4567-890" , null));

        users.add(new User("user", "u123", "user", "098-7654-321" ,null));

    }

    public static AbstractUser authenticate(String username, String password) {

        for (AbstractUser user : users) {

            if (user.getUsername().equals(username) && user.getPassword().equals(password)) {

                return user;

            }

        }

        return null;

    }

}

AbstractUser.java

package main.java.models;

import java.io.Serializable;

import java.util.Objects;

public abstract class AbstractUser implements Serializable {

    private String username;

    private String password;

    private String role;

    private String phoneNumber;

    // Parameterized constructor

    public AbstractUser(String username, String password, String role, String phoneNumber) {

        this.username = username;

        this.password = password;

        this.role = role;

        this.phoneNumber = phoneNumber;

    }

    // Getters and Setters

    public String getUsername() {

        return username;

    }

    public void setUsername(String username) {

        this.username = username;

    }

    public String getPassword() {

        return password;

    }

    public void setPassword(String password) {

        this.password = password;

    }

    public String getRole() {

        return role;

    }

    public void setRole(String role) {

        this.role = role;

    }

    public String getPhoneNumber() {return phoneNumber;}

    public void setPhoneNumber(String phoneNumber) {this.phoneNumber = phoneNumber;}

    // Abstract methods for login and logout

    public abstract void login();

    public abstract void logout();

    @Override

    public boolean equals(Object o) {

        if (this == o) return true;

        if (o == null || getClass() != o.getClass()) return false;

        AbstractUser that = (AbstractUser) o;

        return username.equals(that.username);

    }

    @Override

    public int hashCode() {

        return Objects.hash(username);

    }

}

Admin.java

package main.java.models;

import java.util.List;

import java.io.Serializable;

public class Admin extends AbstractUser implements Serializable {

    private List<Server> managedServers;

    public Admin(String username, String password, String role, String phoneNumber, List<Server> managedServers) {

        super(username, password, role, phoneNumber);

        this.managedServers = managedServers;

    }

    // Getters and Setters for managedServers

    public List<Server> getManagedServers() {

        return managedServers;

    }

    public void setManagedServers(List<Server> managedServers) {

        this.managedServers = managedServers;

    }

    @Override

    public void login() {

        System.out.println("Admin " + getUsername() + " logged in.");

    }

    @Override

    public void logout() {

        System.out.println("Admin " + getUsername() + " logged out.");

    }

    public void viewAllServerStatus() {

        for (Server server : managedServers) {

            System.out.println("Server: " + server.getName() + ", CPU: " + server.getCpuUsage() + ", Memory: " + server.getMemoryUsage() + ", Latency: " + server.getNetworkLatency());

        }

    }

    public void approveRestartRequest(Server server) {

        server.restart();

        System.out.println("Restart request approved for server: " + server.getName());

    }

    public void addServer(Server server) {

        if ("admin".equals(getRole())) {

            // Logic to add a server

            managedServers.add(server);

            System.out.println("Server added: " + server.getName());

        } else {

            System.out.println("Insufficient privileges to add server.");

        }

    }

    public void removeServer(Server server) {

        if ("admin".equals(getRole())) {

            // Logic to remove a server

            managedServers.remove(server);

            System.out.println("Server removed: " + server.getName());

        } else {

            System.out.println("Insufficient privileges to remove server.");

        }

    }

}

Server.java

package main.java.models;

import main.java.exceptions.ServerException;

import main.java.utils.FileHandler;

import java.util.HashMap;

import java.util.Map;

public class Server {

    private String name;

    private double cpuUsage;

    private double memoryUsage;

    private double networkLatency;

    private int cpuThreshold;

    private int memoryThreshold;

    private int networkThreshold;

    // A map to hold subscriptions for different alert types

    private Map<String, Boolean> alertSubscriptions;

    public Server(String name, int cpuThreshold, int memoryThreshold, int networkThreshold) {

        this.name = name;

        this.cpuUsage = cpuUsage;

        this.memoryUsage = memoryUsage;

        this.networkLatency = networkLatency;

        // Initialize the thresholds

        this.cpuThreshold = cpuThreshold;

        this.memoryThreshold = memoryThreshold;

        this.networkThreshold = networkThreshold;

        // Initialize the alert subscriptions map

        alertSubscriptions = new HashMap<>();

        alertSubscriptions.put("CPU Usage", false);

        alertSubscriptions.put("Memory Usage", false);

        alertSubscriptions.put("Network Latency", false);

    }

    // Added getter methods for the thresholds

    public int getCpuThreshold() {

        return cpuThreshold;

    }

    public int getMemoryThreshold() {

        return memoryThreshold;

    }

    public int getNetworkThreshold() {

        return networkThreshold;

    }

    // Add these setter methods for the thresholds

    public void setCpuThreshold(int cpuThreshold) {

        this.cpuThreshold = cpuThreshold;

    }

    public void setMemoryThreshold(int memoryThreshold) {

        this.memoryThreshold = memoryThreshold;

    }

    public void setNetworkThreshold(int networkThreshold) {

        this.networkThreshold = networkThreshold;

    }

    public void simulateMonitoring() {

        try {

            // existing code

            // Simulate CPU usage

            this.cpuUsage = Math.random() \* 100;

            // Simulate Memory usage

            this.memoryUsage = Math.random() \* 100;

            // Simulate Network latency

            this.networkLatency = Math.random() \* 100;

        } catch(RuntimeException e) {

            throw new ServerException("Error during server monitoring: " + e.getMessage());

        }

    }

    // Getter methods for the metrics

    public double getCpuUsage() {

        return cpuUsage;

    }

    public double getMemoryUsage() {

        return memoryUsage;

    }

    public double getNetworkLatency() {

        return networkLatency;

    }

    public String checkThresholds() {

        //StringBuilder message = new StringBuilder("SMS sent to 123-456-7890: ");

        StringBuilder message = new StringBuilder();

        boolean hasWarning = false;

        // Use class variables for thresholds

        if (cpuUsage > cpuThreshold) {

            message.append("Warning: Server " + this.name + " CPU usage is high. ");

            hasWarning = true;

        }

        if (memoryUsage > memoryThreshold) {

            message.append("Warning: Server " + this.name + " Memory usage is high. ");

            hasWarning = true;

        }

        if (networkLatency > networkThreshold) {

            message.append("Warning: Server " + this.name + " Network latency is high. ");

            hasWarning = true;

        }

        return hasWarning ? message.toString() : "";

    }

    public String getName() {

        return this.name;

    }

    public void setName(String name) {

        this.name = name;

    }

    public void setCpuUsage(int cpuUsage) {

        this.cpuUsage = cpuUsage;

    }

    public void setMemoryUsage(int memoryUsage) {

        this.memoryUsage = memoryUsage;

    }

    public void setNetworkLatency(int networkLatency) {

        this.networkLatency = networkLatency;

    }

    public void restart() {

        //System.out.println("Server is restarting...");  // Debugging line

        this.cpuUsage = 0.0;

        this.memoryUsage = 0.0;

        this.networkLatency = 0.0;

        // Log the restart action

        //System.out.println("Server restarted. Metrics set to 0.0");

        System.out.println("Server " + this.getName() + " metrics after restart: CPU=" + this.cpuUsage + ", Memory=" + this.memoryUsage + ", Latency=" + this.networkLatency);

        FileHandler.writeLog("server\_actions.txt", "Restarted " + this.getName());

    }

    public void subscribeToAlert(String alertType) {

        if ("All Alerts".equals(alertType)) {

            for (String key : alertSubscriptions.keySet()) {

                alertSubscriptions.put(key, true);

            }

            System.out.println("Subscribed to all alerts for server " + name);

        } else if (alertSubscriptions.containsKey(alertType)) {

            alertSubscriptions.put(alertType, true);

            System.out.println("Subscribed to " + alertType + " alerts for server " + name);

        } else {

            System.out.println("Invalid alert type: " + alertType);

        }

    }

    // Method to check if the server is subscribed to a particular alert

    public boolean isSubscribedToAlert(String alertType) {

        return alertSubscriptions.getOrDefault(alertType, false);

    }

    @Override

    public String toString() {

        return this.getName();  // Assuming you have a getName() method that returns the server's name

    }

}

User.java

package main.java.models;

import java.util.List;

import java.io.Serializable;

public class User extends AbstractUser implements Serializable {

    private List<Server> serverList;

    public User(String username, String password, String role, String phoneNumber, List<Server> serverList) {

        super(username, password, role, phoneNumber);

        this.serverList = serverList;

    }

    // Getters and Setters for serverList

    public List<Server> getServerList() {

        return serverList;

    }

    public void setServerList(List<Server> serverList) {

        this.serverList = serverList;

    }

    @Override

    public void login() {

        System.out.println("User " + getUsername() + " logged in.");

    }

    @Override

    public void logout() {

        System.out.println("User " + getUsername() + " logged out.");

    }

    public void viewServerStatus() {

        if (hasPrivilege("view")) {

            for (Server server : serverList) {

                System.out.println("Server: " + server.getName() + ", CPU: " + server.getCpuUsage() + ", Memory: " + server.getMemoryUsage() + ", Latency: " + server.getNetworkLatency());

            }

        } else {

            System.out.println("Insufficient privileges to view server status.");

        }

    }

    public void requestServerRestart(Server server) {

        if (hasPrivilege("restart")) {

            System.out.println("Restart request sent for server: " + server.getName());

        } else {

            System.out.println("Insufficient privileges to request server restart.");

        }

    }

    private boolean hasPrivilege(String action) {

        // You can extend this method to include more complex role-based logic

        switch (action) {

            case "view":

                return "user".equals(getRole()) || "admin".equals(getRole());

            case "restart":

                return "admin".equals(getRole());

            default:

                return false;

        }

    }

}

CustomException.java

package main.java.exceptions;

public class CustomException extends Exception {

    public CustomException(String message) {

        super(message);

    }

}

LoginException.java

package main.java.exceptions;

public class LoginException extends CustomException {

    public LoginException(String message) {

        super(message);

    }

}

ServerException.java

package main.java.exceptions;

public class ServerException extends RuntimeException {

    public ServerException(String message) {

        super(message);

    }

}

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~XXXXXXXXXXXXXX~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~