ActionDAO.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package ecsapplication.DataAccess;
import ecsapplication.Model.Action;
import ecsapplication.Model.Employee;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;



public class ActionDAO {
    // Create action
    public boolean createAction(Action action) {
        String sql = "INSERT INTO action (action_type, employee_id, equipment_id, action_date, action_name) VALUES (?, ?, ?, ?, ?)";


        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS)) {


            stmt.setString(1, action.getActionType());
            stmt.setInt(2, action.getEmployee() != null ? action.getEmployee().getEmployeeId() : null);
```

```
            stmt.setInt(3, action.getEquipment() != null ?
action.getEquipment().getEquipmentId() : null);
            stmt.setTimestamp(4, action.getActionDate() != null ?
Timestamp.valueOf(action.getActionDate()) : null);
            stmt.setString(5, action.getActionName());


            int affectedRows = stmt.executeUpdate();
            if (affectedRows == 0) {
                return false;
            }


            try (ResultSet generatedKeys = stmt.getGeneratedKeys()) {
                if (generatedKeys.next()) {
                    action.setActionId(generatedKeys.getInt(1));
                    return true;
                } else {
                    return false;
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }

    // Get action by ID
    public Action getActionById(int actionId) {
```

```
    String sql = "SELECT * FROM action WHERE action_id = ?";

    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, actionId);

        try (ResultSet rs = stmt.executeQuery()) {
            if (rs.next()) {
                return extractActionFromResultSet(rs);
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return null;
}

// Get all actions
public List<Action> getAllActions() {
    List<Action> actions = new ArrayList<>();
    String sql = "SELECT * FROM action";

    try (Connection conn = DatabaseConnection.getConnection();
         Statement stmt = conn.createStatement();
         ResultSet rs = stmt.executeQuery(sql)) {
```

```
            while (rs.next()) {
                actions.add(extractActionFromResultSet(rs));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return actions;
    }

    // Update action
    public boolean updateAction(Action action) {
        String sql = "UPDATE action SET action_name = ? WHERE action_id
= ?";

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setString(1, action.getActionName());
            stmt.setInt(2, action.getActionId());

            int affectedRows = stmt.executeUpdate();
            return affectedRows > 0;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
```

```
        }
    }

    // Delete action
    public boolean deleteAction(int actionId) {
        String sql = "DELETE FROM action WHERE action_id = ?";

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setInt(1, actionId);

            int affectedRows = stmt.executeUpdate();
            return affectedRows > 0;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }

    // Helper method to extract Action from ResultSet
    private Action extractActionFromResultSet(ResultSet rs) throws
SQLException {
        Action action = new Action();
        action.setActionId(rs.getInt("action_id"));
        action.setActionName(rs.getString("action_name"));
        return action;
```

```
    }
}
```

AuditingDAO.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-
default.txt to change this license
```

```
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package ecsapplication.DataAccess;
import ecsapplication.Model.Action;
import ecsapplication.Model.Auditing;
import ecsapplication.Model.Employee;
import java.sql.*;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;

public class AuditingDAO {
    private EmployeeDAO employeeDAO;
    private ActionDAO actionDAO;

    public AuditingDAO() {
        // Empty constructor
    }

    // Lazily initialize the DAOs when needed
    private EmployeeDAO getEmployeeDAO() {
        if (employeeDAO == null) {
            employeeDAO = DBManager.getInstance().getDAO(EmployeeDAO.class);
        }
        return employeeDAO;
```

```
    }

    private ActionDAO getActionDAO() {
        if (actionDAO == null) {
            actionDAO = DBManager.getInstance().getDAO(ActionDAO.class);
        }
        return actionDAO;
    }

    // Create audit log
    public boolean createAuditLog(Auditing audit) {
        String sql = "INSERT INTO auditing (timestamp, details, employee_id,
action_id) VALUES (?, ?, ?, ?)";

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql,
Statement.RETURN_GENERATED_KEYS)) {

            if (audit.getTimestamp() != null) {
                stmt.setTimestamp(1, Timestamp.valueOf(audit.getTimestamp()));
            } else {
                stmt.setTimestamp(1, Timestamp.valueOf(LocalDateTime.now()));
            }

            stmt.setString(2, audit.getDetails());
            stmt.setInt(3, audit.getEmployee().getEmployeeId());
            stmt.setInt(4, audit.getAction().getActionId());
```

```
        int affectedRows = stmt.executeUpdate();
        if (affectedRows == 0) {
            return false;
        }

        try (ResultSet generatedKeys = stmt.getGeneratedKeys()) {
            if (generatedKeys.next()) {
                audit.setLogId(generatedKeys.getInt(1));
                return true;
            } else {
                return false;
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
}

// Get audit log by ID
public Auditing getAuditLogById(int logId) {
    String sql = "SELECT * FROM auditing WHERE log_id = ?";

    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {
```

```
        stmt.setInt(1, logId);

        try (ResultSet rs = stmt.executeQuery()) {
            if (rs.next()) {
                return extractAuditLogFromResultSet(rs);
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return null;
}

// Get all audit logs
public List<Auditing> getAllAuditLogs() {
    List<Auditing> auditLogs = new ArrayList<>();
    String sql = "SELECT * FROM auditing ORDER BY timestamp DESC";

    try (Connection conn = DatabaseConnection.getConnection();
         Statement stmt = conn.createStatement();
         ResultSet rs = stmt.executeQuery(sql)) {

        while (rs.next()) {
            auditLogs.add(extractAuditLogFromResultSet(rs));
        }
    } catch (SQLException e) {
```

```
            e.printStackTrace();
        }

        return auditLogs;
    }

    // Get audit logs by employee
    public List<Auditing> getAuditLogsByEmployee(int employeeId) {
        List<Auditing> auditLogs = new ArrayList<>();
        String sql = "SELECT * FROM auditing WHERE employee_id = ? ORDER
BY timestamp DESC";

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setInt(1, employeeId);

            try (ResultSet rs = stmt.executeQuery()) {
                while (rs.next()) {
                    auditLogs.add(extractAuditLogFromResultSet(rs));
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return auditLogs;
```

```
    }

    // Get audit logs by action
    public List<Auditing> getAuditLogsByAction(int actionId) {
        List<Auditing> auditLogs = new ArrayList<>();
        String sql = "SELECT * FROM auditing WHERE action_id = ? ORDER BY
timestamp DESC";

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setInt(1, actionId);

            try (ResultSet rs = stmt.executeQuery()) {
                while (rs.next()) {
                    auditLogs.add(extractAuditLogFromResultSet(rs));
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return auditLogs;
    }

    // Get audit logs by date range
```

```
    public List<Auditing> getAuditLogsByDateRange(LocalDateTime
startDate, LocalDateTime endDate) {
        List<Auditing> auditLogs = new ArrayList<>();
        String sql = "SELECT * FROM auditing WHERE timestamp BETWEEN ?
AND ? ORDER BY timestamp DESC";

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setTimestamp(1, Timestamp.valueOf(startDate));
            stmt.setTimestamp(2, Timestamp.valueOf(endDate));

            try (ResultSet rs = stmt.executeQuery()) {
                while (rs.next()) {
                    auditLogs.add(extractAuditLogFromResultSet(rs));
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return auditLogs;
    }

    // Update audit log
    public boolean updateAuditLog(Auditing audit) {
        String sql = "UPDATE auditing SET timestamp = ?, details = ?,
employee_id = ?, action_id = ? WHERE log_id = ?";
```

```
    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setTimestamp(1, Timestamp.valueOf(audit.getTimestamp()));
        stmt.setString(2, audit.getDetails());
        stmt.setInt(3, audit.getEmployee().getEmployeeId());
        stmt.setInt(4, audit.getAction().getActionId());
        stmt.setInt(5, audit.getLogId());

        int affectedRows = stmt.executeUpdate();
        return affectedRows > 0;
    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
}

// Delete audit log
public boolean deleteAuditLog(int logId) {
    String sql = "DELETE FROM auditing WHERE log_id = ?";

    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, logId);
```

```
            int affectedRows = stmt.executeUpdate();
            return affectedRows > 0;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }

    // Helper method to extract Auditing from ResultSet
    private Auditing extractAuditLogFromResultSet(ResultSet rs) throws
SQLException {
        Auditing audit = new Auditing();
        audit.setLogId(rs.getInt("log_id"));
        audit.setTimestamp(rs.getTimestamp("timestamp").toLocalDateTime());
        audit.setDetails(rs.getString("details"));

        Employee employee =
getEmployeeDAO().getEmployeeById(rs.getInt("employee_id"));
        audit.setEmployee(employee);

        Action action = getActionDAO().getActionById(rs.getInt("action_id"));
        audit.setAction(action);

        return audit;
    }
}
```

DBManager.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package ecsapplication.DataAccess;
```

```
import java.sql.Connection;
import java.sql.SQLException;
import java.util.HashMap;
import java.util.Map;

/**
 * Manages database connections and provides access to DAO classes
 */
public class DBManager {
    private static DBManager instance;
    private Map<Class<?>, Object> daoMap = new HashMap<>();

    // Private constructor to enforce singleton pattern
    private DBManager() {
        // Initialize all DAO classes
try {
    // Initialize all DAO classes
    System.out.println("Initializing ActionDAO");
    daoMap.put(ActionDAO.class, new ActionDAO());

    System.out.println("Initializing AuditingDAO");
    daoMap.put(AuditingDAO.class, new AuditingDAO());

    System.out.println("Initializing EmployeeDAO");
    daoMap.put(EmployeeDAO.class, new EmployeeDAO());

    System.out.println("Initializing EmployeeRoleDAO");
```

```
    daoMap.put(EmployeeRoleDAO.class, new EmployeeRoleDAO());

    System.out.println("Initializing EquipmentDAO");
    daoMap.put(EquipmentDAO.class, new EquipmentDAO());

    System.out.println("Initializing EquipmentAvailabilityDAO");
    daoMap.put(EquipmentAvailabilityDAO.class, new
EquipmentAvailabilityDAO());

    System.out.println("Initializing EquipmentStatusDAO");
    daoMap.put(EquipmentStatusDAO.class, new EquipmentStatusDAO());

    System.out.println("Initializing LocationDAO");
    daoMap.put(LocationDAO.class, new LocationDAO());

    System.out.println("Initializing NotificationLogDAO");
    daoMap.put(NotificationLogDAO.class, new NotificationLogDAO());

    System.out.println("Initializing NotificationTypeDAO");
    daoMap.put(NotificationTypeDAO.class, new NotificationTypeDAO());

    System.out.println("Initializing SkillDAO");
    daoMap.put(SkillDAO.class, new SkillDAO());

    System.out.println("Initializing SupervisorDAO");
    daoMap.put(SupervisorDAO.class, new SupervisorDAO());
```

```
    System.out.println("Initializing TransactionDAO");
    daoMap.put(TransactionDAO.class, new TransactionDAO());

    System.out.println("Initializing ValueLossDAO");
    daoMap.put(ValueLossDAO.class, new ValueLossDAOImpl(this));

    System.out.println("All DAOs initialized successfully");
} catch (Exception e) {
    System.err.println("Error initializing DBManager: " + e.getMessage());
    e.printStackTrace();

    // Initialize an empty map to prevent null pointer exceptions
    daoMap = new HashMap<>();
    }
    }
    // Get singleton instance
    public static synchronized DBManager getInstance() {
        if (instance == null) {
            instance = new DBManager();
        }
        return instance;
    }

    // Get database connection
    public Connection getConnection() throws SQLException {
        return DatabaseConnection.getConnection();
    }
```

```
    // Get DAO by class
    @SuppressWarnings("unchecked")
    public <T> T getDAO(Class<T> daoClass) {
        return (T) daoMap.get(daoClass);
    }

    // Testing database connection
    public boolean testConnection() {
        try (Connection conn = getConnection()) {
            return conn != null && !conn.isClosed();
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }
}
```

DatabaseConnection.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package ecsapplication.DataAccess;
import java.sql.Connection;
import java.sql.DriverManager;
```

```
import java.sql.SQLException;


public class DatabaseConnection {
    static final String URL = "jdbc:mysql://localhost:3306/ecs_project_db";
    static final String USER = "root";
    public static final String PASSWORD = "Password"; // Change this to your
actual database password


    public static Connection getConnection() throws SQLException {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            return DriverManager.getConnection(URL, USER, PASSWORD);
        } catch (ClassNotFoundException e) {
            throw new SQLException("Database driver not found", e);
        }
    }


    public static boolean testConnection() {
        try {
            Connection conn = getConnection();
            boolean isConnected = conn != null && !conn.isClosed();
            if (isConnected) {
                System.out.println("Database connection successful!");
            } else {
                System.out.println("Failed to establish database connection.");
            }
            if (conn != null) conn.close();
```

```
            return isConnected;
        } catch (SQLException e) {
            System.err.println("Database connection error: " + e.getMessage());
            e.printStackTrace();
            return false;
        }
    }
}
```

DatabaseConnectionTester.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package ecsapplication.DataAccess;

import java.sql.Connection;
import java.sql.DatabaseMetaData;
```

```
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

/**
 * Utility class to test database connectivity
 */
public class DatabaseConnectionTester {

    /**
     * Main method to test database connection
     * @param args command line arguments (not used)
     */
    public static void main(String[] args) {
        System.out.println("Database Connection Test");
        System.out.println("========================");

        // Test basic connection
        testBasicConnection();

        // If basic connection succeeds, test table existence and counts
        if (testBasicConnection()) {
            testTables();
        }
    }

    /**
```

```
     * Tests basic database connectivity
     * @return true if connection succeeds, false otherwise
     */
    public static boolean testBasicConnection() {
        System.out.println("\nTesting Basic Connection...");
        try {
            // Get connection details from DatabaseConnection class
            System.out.println("Connection URL: " + DatabaseConnection.URL);
            System.out.println("Username: " + DatabaseConnection.USER);
            System.out.println("Password: " +
(DatabaseConnection.PASSWORD.isEmpty() ? "[empty]" : "[provided]"));

            // Try to establish connection
            Connection conn = DatabaseConnection.getConnection();

            if (conn != null && !conn.isClosed()) {
                System.out.println("□ Connection successful!");

                // Get database metadata
                DatabaseMetaData metaData = conn.getMetaData();
                System.out.println("Database Product: " +
metaData.getDatabaseProductName());
                System.out.println("Database Version: " +
metaData.getDatabaseProductVersion());
                System.out.println("Driver Name: " + metaData.getDriverName());
                System.out.println("Driver Version: " +
metaData.getDriverVersion());
```

```
            // Close connection
            conn.close();
            System.out.println("Connection closed properly.");
            return true;
        } else {
            System.out.println("□ Connection failed! Connection is null or
closed.");
            return false;
        }
    } catch (SQLException e) {
        System.out.println("□ Connection error: " + e.getMessage());
        e.printStackTrace();
        return false;
    } catch (Exception e) {
        System.out.println("□ Unexpected error: " + e.getMessage());
        e.printStackTrace();
        return false;
    }
}

/**
 * Tests database tables existence and record counts
 */
public static void testTables() {
    System.out.println("\nTesting Tables...");

    // List of tables to check
```

```
        String[] tables = {
            "action",
            "auditing",
            "employee",
            "employeerole",
            "equipment",
            "equipmentavailability",
            "equipmentstatus",
            "location",
            "notificationlog",
            "notificationtype",
            "skills",
            "supervisor",
            "transaction"
        };

        try (Connection conn = DatabaseConnection.getConnection()) {
            // Get database metadata
            DatabaseMetaData metaData = conn.getMetaData();

            // Loop through each table and check existence
            for (String table : tables) {
                ResultSet rs = metaData.getTables(null, null, table, new String[]
{"TABLE"});

                if (rs.next()) {
                    System.out.print("□ Table '" + table + "' exists. ");
```

```
                // Count records in the table
                try (Statement stmt = conn.createStatement()) {
                    ResultSet countRs = stmt.executeQuery("SELECT COUNT(*)
FROM " + table);
                    if (countRs.next()) {
                        int count = countRs.getInt(1);
                        System.out.println("Record count: " + count);
                    }
                } catch (SQLException e) {
                    System.out.println("Error counting records: " +
e.getMessage());
                }
            } else {
                System.out.println("□ Table '" + table + "' does not exist!");
            }

            rs.close();
        }

    } catch (SQLException e) {
        System.out.println("□ Error testing tables: " + e.getMessage());
        e.printStackTrace();
    }
  }
}
```

DatabaseInitializer.java

```
package ecsapplication.DataAccess;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
```

```
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;

public class DatabaseInitializer {
    private static final String DB_NAME = "ecs_project_db";
    private static final String MYSQL_URL = "jdbc:mysql://localhost:3306/";
    private static final String USER = "root";
    private static final String PASSWORD = "Password"; // Use the same password as DatabaseConnection

    public static void initializeDatabase() {
        try {
            if (!databaseExists()) {
                createDatabase();
            }
            initializeTables();
        } catch (SQLException e) {
            System.err.println("Error initializing database: " + e.getMessage());
            e.printStackTrace();
        }
    }

    private static boolean databaseExists() throws SQLException {
        try (Connection conn = DriverManager.getConnection(MYSQL_URL, USER, PASSWORD)) {
            ResultSet resultSet = conn.getMetaData().getCatalogs();
```

```
            while (resultSet.next()) {
                if (DB_NAME.equals(resultSet.getString(1))) {
                    return true;
                }
            }
        }
        return false;
    }

    private static void createDatabase() throws SQLException {
        try (Connection conn = DriverManager.getConnection(MYSQL_URL,
USER, PASSWORD);
             Statement stmt = conn.createStatement()) {
            stmt.executeUpdate("CREATE DATABASE IF NOT EXISTS " +
DB_NAME);
            System.out.println("Database created successfully");
        }
    }

    private static void initializeTables() throws SQLException {
        // Read SQL script from file
        List<String> sqlStatements = readSQLScript();

        // Execute each SQL statement
        try (Connection conn = DatabaseConnection.getConnection();
             Statement stmt = conn.createStatement()) {
```

```
            for (String sql : sqlStatements) {
                if (!sql.trim().isEmpty()) {
                    try {
                        stmt.executeUpdate(sql);
                    } catch (SQLException e) {
                        // Log the error but continue with other statements
                        System.err.println("Error executing SQL statement: " +
e.getMessage());
                        System.err.println("Statement: " + sql);
                    }
                }
            }
            System.out.println("Database tables initialized successfully");
        }
    }

    private static List<String> readSQLScript() {
        List<String> sqlStatements = new ArrayList<>();
        StringBuilder currentStatement = new StringBuilder();

        try (InputStream is =
DatabaseInitializer.class.getResourceAsStream("/ecsapplication/Resources/
EMS_Database.sql");
             BufferedReader reader = new BufferedReader(new
InputStreamReader(is))) {

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

```
            // Skip comments and empty lines
            if (line.trim().startsWith("--") || line.trim().isEmpty()) {
                continue;
            }

            currentStatement.append(line);

            // If the line ends with a semicolon, it's the end of a statement
            if (line.trim().endsWith(";")) {
                sqlStatements.add(currentStatement.toString());
                currentStatement = new StringBuilder();
            } else {
                currentStatement.append(" ");
            }
        }
    } catch (IOException e) {
        System.err.println("Error reading SQL script: " + e.getMessage());
        e.printStackTrace();
    }

    return sqlStatements;
  }
}
```

EmployeeDAO.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package ecsapplication.DataAccess;
import ecsapplication.Model.Employee;
import ecsapplication.Model.EmployeeRole;
import ecsapplication.Model.Skill;
import java.sql.*;
```

```
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;

public class EmployeeDAO {
    private EmployeeRoleDAO roleDAO = new EmployeeRoleDAO();
    private SkillDAO skillDAO = new SkillDAO();

    // Create employee
    public boolean createEmployee(Employee employee) {
        String sql = "INSERT INTO employee (first_name, last_name, hire_date,
role_id, skill_id, email) VALUES (?, ?, ?, ?, ?, ?)";

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql,
Statement.RETURN_GENERATED_KEYS)) {

            stmt.setString(1, employee.getFirstName());
            stmt.setString(2, employee.getLastName());
            stmt.setDate(3, Date.valueOf(employee.getHireDate()));
            stmt.setInt(4, employee.getRole().getRoleId());
            stmt.setInt(5, employee.getSkill().getSkillId());
            stmt.setString(6, employee.getEmail());

            int affectedRows = stmt.executeUpdate();
            if (affectedRows == 0) {
                return false;
```

```
        }

        try (ResultSet generatedKeys = stmt.getGeneratedKeys()) {
            if (generatedKeys.next()) {
                employee.setEmployeeId(generatedKeys.getInt(1));
                return true;
            } else {
                return false;
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
}

// Get employee by ID
public Employee getEmployeeById(int employeeId) {
    String sql = "SELECT * FROM employee WHERE employee_id = ?";

    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, employeeId);

        try (ResultSet rs = stmt.executeQuery()) {
            if (rs.next()) {
```

```
                return extractEmployeeFromResultSet(rs);
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return null;
}

// Get all employees
public List<Employee> getAllEmployees() {
    List<Employee> employees = new ArrayList<>();
    String sql = "SELECT * FROM employee";

    try (Connection conn = DatabaseConnection.getConnection();
         Statement stmt = conn.createStatement();
         ResultSet rs = stmt.executeQuery(sql)) {

        while (rs.next()) {
            employees.add(extractEmployeeFromResultSet(rs));
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return employees;
```

```
}

// Get employees by role
public List<Employee> getEmployeesByRole(int roleId) {
    List<Employee> employees = new ArrayList<>();
    String sql = "SELECT * FROM employee WHERE role_id = ?";

    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, roleId);

        try (ResultSet rs = stmt.executeQuery()) {
            while (rs.next()) {
                employees.add(extractEmployeeFromResultSet(rs));
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return employees;
}

// Update employee
public boolean updateEmployee(Employee employee) {
```

```
        String sql = "UPDATE employee SET first_name = ?, last_name = ?,
hire_date = ?, role_id = ?, skill_id = ?, email = ? WHERE employee_id = ?";

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setString(1, employee.getFirstName());
            stmt.setString(2, employee.getLastName());
            stmt.setDate(3, Date.valueOf(employee.getHireDate()));
            stmt.setInt(4, employee.getRole().getRoleId());
            stmt.setInt(5, employee.getSkill().getSkillId());
            stmt.setString(6, employee.getEmail());
            stmt.setInt(7, employee.getEmployeeId());

            int affectedRows = stmt.executeUpdate();
            return affectedRows > 0;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }

    // Delete employee
    public boolean deleteEmployee(int employeeId) {
        String sql = "DELETE FROM employee WHERE employee_id = ?";

        try (Connection conn = DatabaseConnection.getConnection();
```

```
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setInt(1, employeeId);

            int affectedRows = stmt.executeUpdate();
            return affectedRows > 0;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }

    // Get employee by ID (String version)
    public Employee getById(String employeeId) {
        try {
            return getEmployeeById(Integer.parseInt(employeeId));
        } catch (NumberFormatException e) {
            e.printStackTrace();
            return null;
        }
    }

    // Helper method to extract Employee from ResultSet
    private Employee extractEmployeeFromResultSet(ResultSet rs) throws
SQLException {
        Employee employee = new Employee();
        employee.setEmployeeId(rs.getInt("employee_id"));
```

```
        employee.setFirstName(rs.getString("first_name"));
        employee.setLastName(rs.getString("last_name"));
        employee.setHireDate(rs.getDate("hire_date").toLocalDate());

        EmployeeRole role = roleDAO.getRoleById(rs.getInt("role_id"));
        employee.setRole(role);

        Skill skill = skillDAO.getSkillById(rs.getInt("skill_id"));
        employee.setSkill(skill);

        employee.setEmail(rs.getString("email"));

        return employee;
    }
}
```

EmployeeRoleDAO.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package ecsapplication.DataAccess;
import ecsapplication.Model.EmployeeRole;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class EmployeeRoleDAO {
    // Create role
    public boolean createRole(EmployeeRole role) {
        String sql = "INSERT INTO employeerole (role_name) VALUES (?)";
```

```
        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql,
Statement.RETURN_GENERATED_KEYS)) {

            stmt.setString(1, role.getRoleName());

            int affectedRows = stmt.executeUpdate();
            if (affectedRows == 0) {
                return false;
            }

            try (ResultSet generatedKeys = stmt.getGeneratedKeys()) {
                if (generatedKeys.next()) {
                    role.setRoleId(generatedKeys.getInt(1));
                    return true;
                } else {
                    return false;
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }

    // Get role by ID
    public EmployeeRole getRoleById(int roleId) {
```

```
    String sql = "SELECT * FROM employeerole WHERE role_id = ?";

    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, roleId);

        try (ResultSet rs = stmt.executeQuery()) {
            if (rs.next()) {
                return extractRoleFromResultSet(rs);
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return null;
}

// Get all roles
public List<EmployeeRole> getAllRoles() {
    List<EmployeeRole> roles = new ArrayList<>();
    String sql = "SELECT * FROM employeerole";

    try (Connection conn = DatabaseConnection.getConnection();
         Statement stmt = conn.createStatement();
         ResultSet rs = stmt.executeQuery(sql)) {
```

```
            while (rs.next()) {
                roles.add(extractRoleFromResultSet(rs));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return roles;
    }

    // Update role
    public boolean updateRole(EmployeeRole role) {
        String sql = "UPDATE employeerole SET role_name = ? WHERE role_id =
?";

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setString(1, role.getRoleName());
            stmt.setInt(2, role.getRoleId());

            int affectedRows = stmt.executeUpdate();
            return affectedRows > 0;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
```

```
    }
}

// Delete role
public boolean deleteRole(int roleId) {
    String sql = "DELETE FROM employeerole WHERE role_id = ?";

    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, roleId);

        int affectedRows = stmt.executeUpdate();
        return affectedRows > 0;
    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
}

// Get role by name
public EmployeeRole getRoleByName(String roleName) {
    String sql = "SELECT * FROM employeerole WHERE role_name = ?";

    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {
```

```
            stmt.setString(1, roleName);

            try (ResultSet rs = stmt.executeQuery()) {
                if (rs.next()) {
                    EmployeeRole role = new EmployeeRole();
                    role.setRoleId(rs.getInt("role_id"));
                    role.setRoleName(rs.getString("role_name"));
                    return role;
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return null;
    }

    // Helper method to extract EmployeeRole from ResultSet
    private EmployeeRole extractRoleFromResultSet(ResultSet rs) throws
SQLException {
        EmployeeRole role = new EmployeeRole();
        role.setRoleId(rs.getInt("role_id"));
        role.setRoleName(rs.getString("role_name"));
        return role;
    }
}
```

EmployeeService.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package ecsapplication.DataAccess;
import ecsapplication.Model.Action;
import ecsapplication.Model.Auditing;
import ecsapplication.Model.Employee;
import ecsapplication.Model.EmployeeRole;
import ecsapplication.Model.Equipment;
import ecsapplication.Model.Skill;
import ecsapplication.Model.Supervisor;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.List;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.sql.Connection;
```

```
/**
 * Service class for employee management operations
 */
public class EmployeeService {
    private EmployeeDAO employeeDAO;
    private EmployeeRoleDAO roleDAO;
    private SkillDAO skillDAO;
    private AuditingDAO auditingDAO;
    private ActionDAO actionDAO;
    private SupervisorDAO supervisorDAO;
    private Connection connection;

    public EmployeeService() {
        try {
            DBManager dbManager = DBManager.getInstance();
            this.employeeDAO = dbManager.getDAO(EmployeeDAO.class);
            this.roleDAO = dbManager.getDAO(EmployeeRoleDAO.class);
            this.skillDAO = dbManager.getDAO(SkillDAO.class);
            this.auditingDAO = dbManager.getDAO(AuditingDAO.class);
            this.actionDAO = dbManager.getDAO(ActionDAO.class);
            this.supervisorDAO = dbManager.getDAO(SupervisorDAO.class);
            this.connection = dbManager.getConnection();
        } catch (SQLException e) {
            e.printStackTrace();
            throw new RuntimeException("Failed to initialize EmployeeService: "
+ e.getMessage());
```

```
        }
    }

    /**
     * Get employee by ID
     */
    public Employee getEmployeeById(int employeeId) {
        return employeeDAO.getEmployeeById(employeeId);
    }

    /**
     * Create a new employee
     */
    public boolean createEmployee(Employee newEmployee, Employee
actingEmployee) {
        boolean created = employeeDAO.createEmployee(newEmployee);

        if (!created) {
            return false;
        }

        // Create audit log
        Action action = actionDAO.getActionById(1); // 1 = CreateUser

        Auditing audit = new Auditing();
        audit.setTimestamp(LocalDateTime.now());
```

```
        audit.setDetails("Created new employee account for ID " +
newEmployee.getEmployeeId() +
                         " (" + newEmployee.getFirstName() + " " +
newEmployee.getLastName() + ")");
        audit.setEmployee(actingEmployee);
        audit.setAction(action);


        auditingDAO.createAuditLog(audit);


        return true;
    }


    /**
     * Update employee role
     */
    public boolean updateEmployeeRole(int employeeId, int roleId, Employee
actingEmployee) {
        Employee employee = employeeDAO.getEmployeeById(employeeId);


        if (employee == null) {
            return false;
        }


        EmployeeRole newRole = roleDAO.getRoleById(roleId);


        if (newRole == null) {
            return false;
        }
```

```
        // Store old role for audit log
        String oldRoleName = employee.getRole().getRoleName();

        // Update employee role
        employee.setRole(newRole);
        boolean updated = employeeDAO.updateEmployee(employee);

        if (!updated) {
            return false;
        }

        // Create audit log
        Action action = actionDAO.getActionById(3); // 3 = AssignRole

        Auditing audit = new Auditing();
        audit.setTimestamp(LocalDateTime.now());
        audit.setDetails("Changed role for employee ID " +
employee.getEmployeeId() +
                        " from " + oldRoleName + " to " + newRole.getRoleName());
        audit.setEmployee(actingEmployee);
        audit.setAction(action);

        auditingDAO.createAuditLog(audit);

        return true;
    }
```

```
    /**
     * Update employee skill level
     */
    public boolean updateEmployeeSkill(int employeeId, int skillId, Employee
actingEmployee) {
        Employee employee = employeeDAO.getEmployeeById(employeeId);

        if (employee == null) {
            return false;
        }

        Skill newSkill = skillDAO.getSkillById(skillId);

        if (newSkill == null) {
            return false;
        }

        // Store old skill for audit log
        String oldSkillLevel = employee.getSkill().getSkillLevel();

        // Update employee skill
        employee.setSkill(newSkill);
        boolean updated = employeeDAO.updateEmployee(employee);

        if (!updated) {
            return false;
```

```
        }

        // Create audit log
        Action action = actionDAO.getActionById(2); // 2 = AssignSkillLevel

        Auditing audit = new Auditing();
        audit.setTimestamp(LocalDateTime.now());
        audit.setDetails("Updated skill level for employee ID " +
employee.getEmployeeId() +
                        " from " + oldSkillLevel + " to " + newSkill.getSkillLevel());
        audit.setEmployee(actingEmployee);
        audit.setAction(action);

        auditingDAO.createAuditLog(audit);

        return true;
    }

    /**
     * Remove employee from system
     */
    public boolean removeEmployee(int employeeId, Employee
actingEmployee) {
        try {
            Employee employee = employeeDAO.getEmployeeById(employeeId);

            if (employee == null) {
```

```
            return false;
        }

        // Delete the employee
        boolean deleted = employeeDAO.deleteEmployee(employeeId);

        if (deleted) {
            // Create audit log
            Action action = actionDAO.getActionById(4); // 4 = RemoveUser

            Auditing audit = new Auditing();
            audit.setTimestamp(LocalDateTime.now());
            audit.setDetails("Removed employee account for ID " +
employeeId +
                    " (" + employee.getFirstName() + " " +
employee.getLastName() + ")");
            audit.setEmployee(actingEmployee);
            audit.setAction(action);

            try {
                auditingDAO.createAuditLog(audit);
            } catch (Exception e) {
                // If audit log fails, we still want to return true since the
employee was deleted
                e.printStackTrace();
            }
        }
```

```
            return deleted;
        } catch (Exception e) {
            // Log the error but don't throw it
            e.printStackTrace();
            return false;
        }
    }

    /**
     * Assign employee as supervisor
     */
    public boolean assignSupervisor(int employeeId, Employee
actingEmployee) {
        Employee employee = employeeDAO.getEmployeeById(employeeId);

        if (employee == null) {
            return false;
        }

        // Check if already a supervisor
        Supervisor existingSupervisor =
supervisorDAO.getSupervisorByEmployeeId(employeeId);

        if (existingSupervisor != null) {
            return false; // Already a supervisor
        }
```

```
        // Check if employee has Supervisor role
        if (employee.getRole().getRoleId() != 1) { // 1 = Supervisor
            // Update employee role first
            EmployeeRole supervisorRole = roleDAO.getRoleById(1);
            employee.setRole(supervisorRole);
            employeeDAO.updateEmployee(employee);
        }

        // Create supervisor record
        Supervisor supervisor = new Supervisor();
        supervisor.setEmployee(employee);

        boolean created = supervisorDAO.createSupervisor(supervisor);

        if (!created) {
            return false;
        }

        // Create audit log
        Action action = actionDAO.getActionById(3); // 3 = AssignRole (using
this for supervisor assignment)

        Auditing audit = new Auditing();
        audit.setTimestamp(LocalDateTime.now());
        audit.setDetails("Assigned Supervisor status to employee ID " +
employeeId +
                         " (" + employee.getFirstName() + " " +
employee.getLastName() + ")");
```

```
        audit.setEmployee(actingEmployee);
        audit.setAction(action);

        auditingDAO.createAuditLog(audit);

        return true;
    }
}
```

EquipmentAvailabilityDAO.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package ecsapplication.DataAccess;
import ecsapplication.Model.EquipmentAvailability;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;


public class EquipmentAvailabilityDAO {
    // Create availability
    public boolean createAvailability(EquipmentAvailability availability) {
        String sql = "INSERT INTO equipmentavailability (availability_name) VALUES (?)";


        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS)) {


            stmt.setString(1, availability.getAvailabilityName());
```

```
            int affectedRows = stmt.executeUpdate();
            if (affectedRows == 0) {
                return false;
            }

            try (ResultSet generatedKeys = stmt.getGeneratedKeys()) {
                if (generatedKeys.next()) {
                    availability.setAvailabilityId(generatedKeys.getInt(1));
                    return true;
                } else {
                    return false;
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }

    // Get availability by ID
    public EquipmentAvailability getAvailabilityById(int availabilityId) {
        String sql = "SELECT * FROM equipmentavailability WHERE
availability_id = ?";

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {
```

```
        stmt.setInt(1, availabilityId);

        try (ResultSet rs = stmt.executeQuery()) {
            if (rs.next()) {
                return extractAvailabilityFromResultSet(rs);
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return null;
}

// Get all availabilities
public List<EquipmentAvailability> getAllAvailabilities() {
    List<EquipmentAvailability> availabilities = new ArrayList<>();
    String sql = "SELECT * FROM equipmentavailability";

    try (Connection conn = DatabaseConnection.getConnection();
         Statement stmt = conn.createStatement();
         ResultSet rs = stmt.executeQuery(sql)) {

        while (rs.next()) {
            availabilities.add(extractAvailabilityFromResultSet(rs));
        }
```

```
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return availabilities;
    }

    // Update availability
    public boolean updateAvailability(EquipmentAvailability availability) {
        String sql = "UPDATE equipmentavailability SET availability_name = ?
WHERE availability_id = ?";

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setString(1, availability.getAvailabilityName());
            stmt.setInt(2, availability.getAvailabilityId());

            int affectedRows = stmt.executeUpdate();
            return affectedRows > 0;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }

    // Delete availability
```

```
    public boolean deleteAvailability(int availabilityId) {
        String sql = "DELETE FROM equipmentavailability WHERE availability_id
= ?";

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setInt(1, availabilityId);

            int affectedRows = stmt.executeUpdate();
            return affectedRows > 0;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }

    // Helper method to extract EquipmentAvailability from ResultSet
    private EquipmentAvailability extractAvailabilityFromResultSet(ResultSet
rs) throws SQLException {
        EquipmentAvailability availability = new EquipmentAvailability();
        availability.setAvailabilityId(rs.getInt("availability_id"));
        availability.setAvailabilityName(rs.getString("availability_name"));
        return availability;
    }
}
```

EquipmentDAO.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package ecsapplication.DataAccess;
import ecsapplication.Model.Employee;
import ecsapplication.Model.Equipment;
import ecsapplication.Model.EquipmentAvailability;
import ecsapplication.Model.EquipmentStatus;
import ecsapplication.Model.Location;
import ecsapplication.Model.Skill;
import java.sql.*;
import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.List;


public class EquipmentDAO {
    private SkillDAO skillDAO = new SkillDAO();
    private EquipmentAvailabilityDAO availabilityDAO = new EquipmentAvailabilityDAO();
    private EquipmentStatusDAO statusDAO = new EquipmentStatusDAO();
    private EmployeeDAO employeeDAO = new EmployeeDAO();
    private LocationDAO locationDAO = new LocationDAO();
```

```
    // Create equipment
    public boolean createEquipment(Equipment equipment) {
        String sql = "INSERT INTO equipment (equipment_code,
equipment_type, value, skill_id, " +
                "availability_id, status_id, employee_id, location_id) " +
                "VALUES (?, ?, ?, ?, ?, ?, ?, ?)";

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql,
Statement.RETURN_GENERATED_KEYS)) {

            stmt.setInt(1, equipment.getEquipmentCode());
            stmt.setString(2, equipment.getEquipmentType());
            stmt.setBigDecimal(3, equipment.getValue());
            stmt.setInt(4, equipment.getSkill().getSkillId());
            stmt.setInt(5, equipment.getAvailability().getAvailabilityId());
            stmt.setInt(6, equipment.getStatus().getStatusId());

            if (equipment.getEmployee() != null) {
                stmt.setInt(7, equipment.getEmployee().getEmployeeId());
            } else {
                stmt.setNull(7, Types.INTEGER);
            }

            if (equipment.getLocation() != null) {
                stmt.setInt(8, equipment.getLocation().getLocationId());
```

```
        } else {
            stmt.setNull(8, Types.INTEGER);
        }

        int affectedRows = stmt.executeUpdate();
        if (affectedRows == 0) {
            return false;
        }

        try (ResultSet generatedKeys = stmt.getGeneratedKeys()) {
            if (generatedKeys.next()) {
                equipment.setEquipmentId(generatedKeys.getInt(1));
                return true;
            } else {
                return false;
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
}

// Get equipment by ID
public Equipment getEquipmentById(int equipmentId) {
    String sql = "SELECT * FROM equipment WHERE equipment_id = ?";
```

```
    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, equipmentId);

        try (ResultSet rs = stmt.executeQuery()) {
            if (rs.next()) {
                return extractEquipmentFromResultSet(rs);
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return null;
}

// Get all equipment
public List<Equipment> getAllEquipment() {
    List<Equipment> equipmentList = new ArrayList<>();
    String sql = "SELECT * FROM equipment";

    try (Connection conn = DatabaseConnection.getConnection();
         Statement stmt = conn.createStatement();
         ResultSet rs = stmt.executeQuery(sql)) {

        while (rs.next()) {
```

```
            equipmentList.add(extractEquipmentFromResultSet(rs));
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return equipmentList;
}

// Get equipment by type
public List<Equipment> getEquipmentByType(String type) {
    List<Equipment> equipmentList = new ArrayList<>();
    String sql = "SELECT * FROM equipment WHERE equipment_type = ?";

    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setString(1, type);

        try (ResultSet rs = stmt.executeQuery()) {
            while (rs.next()) {
                equipmentList.add(extractEquipmentFromResultSet(rs));
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
```

```
    return equipmentList;
}

// Get equipment by status
public List<Equipment> getEquipmentByStatus(int statusId) {
    List<Equipment> equipmentList = new ArrayList<>();
    String sql = "SELECT * FROM equipment WHERE status_id = ?";

    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, statusId);

        try (ResultSet rs = stmt.executeQuery()) {
            while (rs.next()) {
                equipmentList.add(extractEquipmentFromResultSet(rs));
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return equipmentList;
}

// Get equipment by availability
```

```
public List<Equipment> getEquipmentByAvailability(int availabilityId) {
    List<Equipment> equipmentList = new ArrayList<>();
    String sql = "SELECT * FROM equipment WHERE availability_id = ?";

    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, availabilityId);

        try (ResultSet rs = stmt.executeQuery()) {
            while (rs.next()) {
                equipmentList.add(extractEquipmentFromResultSet(rs));
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return equipmentList;
}

// Get equipment by employee
public List<Equipment> getEquipmentByEmployee(int employeeId) {
    List<Equipment> equipmentList = new ArrayList<>();
    String sql = "SELECT * FROM equipment WHERE employee_id = ?";

    try (Connection conn = DatabaseConnection.getConnection();
```

```
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, employeeId);

        try (ResultSet rs = stmt.executeQuery()) {
            while (rs.next()) {
                equipmentList.add(extractEquipmentFromResultSet(rs));
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return equipmentList;
}

// Get equipment by location
public List<Equipment> getEquipmentByLocation(int locationId) {
    List<Equipment> equipmentList = new ArrayList<>();
    String sql = "SELECT * FROM equipment WHERE location_id = ?";

    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, locationId);

        try (ResultSet rs = stmt.executeQuery()) {
```

```
            while (rs.next()) {
                equipmentList.add(extractEquipmentFromResultSet(rs));
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return equipmentList;
}

// Get equipment by ID (String version)
public Equipment getById(String equipmentId) {
    try {
        return getEquipmentById(Integer.parseInt(equipmentId));
    } catch (NumberFormatException e) {
        e.printStackTrace();
        return null;
    }
}

// Update equipment
public boolean updateEquipment(Equipment equipment) {
    String sql = "UPDATE equipment SET equipment_code = ?,
equipment_type = ?, value = ?, " +
                 "skill_id = ?, availability_id = ?, status_id = ?, employee_id = ?,
location_id = ? " +
```

```
            "WHERE equipment_id = ?";

try (Connection conn = DatabaseConnection.getConnection();
     PreparedStatement stmt = conn.prepareStatement(sql)) {

    stmt.setInt(1, equipment.getEquipmentCode());
    stmt.setString(2, equipment.getEquipmentType());
    stmt.setBigDecimal(3, equipment.getValue());
    stmt.setInt(4, equipment.getSkill().getSkillId());
    stmt.setInt(5, equipment.getAvailability().getAvailabilityId());
    stmt.setInt(6, equipment.getStatus().getStatusId());

    if (equipment.getEmployee() != null) {
        stmt.setInt(7, equipment.getEmployee().getEmployeeId());
    } else {
        stmt.setNull(7, Types.INTEGER);
    }

    if (equipment.getLocation() != null) {
        stmt.setInt(8, equipment.getLocation().getLocationId());
    } else {
        stmt.setNull(8, Types.INTEGER);
    }

    stmt.setInt(9, equipment.getEquipmentId());

    int affectedRows = stmt.executeUpdate();
```

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

// Delete equipment
public boolean deleteEquipment(int equipmentId) {
    String sql = "DELETE FROM equipment WHERE equipment_id = ?";

    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, equipmentId);

        int affectedRows = stmt.executeUpdate();
        return affectedRows > 0;
    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
}

// Check out equipment
public boolean checkOutEquipment(int equipmentId, int employeeId) {
```

```
        String sql = "UPDATE equipment SET availability_id = ?, employee_id
= ? WHERE equipment_id = ?";

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            // 2 = CheckedOut status
            stmt.setInt(1, 2);
            stmt.setInt(2, employeeId);
            stmt.setInt(3, equipmentId);

            int affectedRows = stmt.executeUpdate();
            return affectedRows > 0;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }

    // Check in equipment
    public boolean checkInEquipment(int equipmentId) {
        String sql = "UPDATE equipment SET availability_id = ?, employee_id =
NULL WHERE equipment_id = ?";

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {
```

```
            // 1 = Available status
            stmt.setInt(1, 1);
            stmt.setInt(2, equipmentId);

            int affectedRows = stmt.executeUpdate();
            return affectedRows > 0;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }

    // Update equipment status
    public boolean update(Equipment equipment) {
        String sql = "UPDATE equipment SET status_id = ? WHERE
equipment_id = ?";

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setInt(1, equipment.getStatus().getStatusId());
            stmt.setInt(2, equipment.getEquipmentId());

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

```
        }
    }

    // Get filtered equipment with optimized query
    public List<Equipment> getFilteredEquipment(String type, Integer
locationId, Integer statusId, Integer availabilityId) {
        List<Equipment> equipmentList = new ArrayList<>();
        StringBuilder sql = new StringBuilder(
            "SELECT e.*, " +
            "s.status_id, s.status_name, " +
            "a.availability_id, a.availability_name, " +
            "emp.employee_id, emp.first_name, emp.last_name, " +
            "l.location_id, l.location_name, " +
            "sk.skill_id, sk.skill_level " +
            "FROM equipment e " +
            "LEFT JOIN equipmentstatus s ON e.status_id = s.status_id " +
            "LEFT JOIN equipmentavailability a ON e.availability_id =
a.availability_id " +
            "LEFT JOIN employee emp ON e.employee_id = emp.employee_id " +
            "LEFT JOIN location l ON e.location_id = l.location_id " +
            "LEFT JOIN skills sk ON e.skill_id = sk.skill_id " +
            "WHERE 1=1"
        );

        List<Object> params = new ArrayList<>();

        // Add filter conditions
```

```
if (type != null && !type.isEmpty()) {
    sql.append(" AND e.equipment_type = ?");
    params.add(type);
}

if (locationId != null) {
    sql.append(" AND e.location_id = ?");
    params.add(locationId);
}

if (statusId != null) {
    sql.append(" AND e.status_id = ?");
    params.add(statusId);
}

if (availabilityId != null) {
    sql.append(" AND e.availability_id = ?");
    params.add(availabilityId);
}

System.out.println("Executing SQL: " + sql.toString());
System.out.println("Parameters: " + params);

try (Connection conn = DatabaseConnection.getConnection();
     PreparedStatement stmt = conn.prepareStatement(sql.toString())) {

    // Set parameters
```

```
        for (int i = 0; i < params.size(); i++) {
            stmt.setObject(i + 1, params.get(i));
        }

        try (ResultSet rs = stmt.executeQuery()) {
            while (rs.next()) {
                Equipment equipment = new Equipment();
                equipment.setEquipmentId(rs.getInt("equipment_id"));
                equipment.setEquipmentCode(rs.getInt("equipment_code"));
                equipment.setEquipmentType(rs.getString("equipment_type"));
                equipment.setValue(rs.getBigDecimal("value"));

                // Set status
                if (rs.getObject("status_id") != null) {
                    EquipmentStatus status = new EquipmentStatus();
                    status.setStatusId(rs.getInt("status_id"));
                    status.setStatusName(rs.getString("status_name"));
                    equipment.setStatus(status);
                }

                // Set availability
                if (rs.getObject("availability_id") != null) {
                    EquipmentAvailability availability = new
EquipmentAvailability();
                    availability.setAvailabilityId(rs.getInt("availability_id"));

availability.setAvailabilityName(rs.getString("availability_name"));
```

```
    equipment.setAvailability(availability);
}

// Set employee if exists
if (rs.getObject("employee_id") != null) {
    Employee employee = new Employee();
    employee.setEmployeeId(rs.getInt("employee_id"));
    employee.setFirstName(rs.getString("first_name"));
    employee.setLastName(rs.getString("last_name"));
    equipment.setEmployee(employee);
}

// Set location if exists
if (rs.getObject("location_id") != null) {
    Location location = new Location();
    location.setLocationId(rs.getInt("location_id"));
    location.setLocationName(rs.getString("location_name"));
    equipment.setLocation(location);
}

// Set skill if exists
if (rs.getObject("skill_id") != null) {
    Skill skill = new Skill();
    skill.setSkillId(rs.getInt("skill_id"));
    skill.setSkillLevel(rs.getString("skill_level"));
    equipment.setSkill(skill);
}
```

```
                equipmentList.add(equipment);
            }
        }

        System.out.println("Query returned " + equipmentList.size() + "
results");

    } catch (SQLException e) {
        System.err.println("SQL Error: " + e.getMessage());
        e.printStackTrace();
    }

    return equipmentList;
}

// Helper method to extract Equipment from ResultSet
private Equipment extractEquipmentFromResultSet(ResultSet rs) throws
SQLException {
    Equipment equipment = new Equipment();
    equipment.setEquipmentId(rs.getInt("equipment_id"));
    equipment.setEquipmentCode(rs.getInt("equipment_code"));
    equipment.setEquipmentType(rs.getString("equipment_type"));
    equipment.setValue(rs.getBigDecimal("value"));

    Skill skill = skillDAO.getSkillById(rs.getInt("skill_id"));
    equipment.setSkill(skill);
```

```
        EquipmentAvailability availability =
availabilityDAO.getAvailabilityById(rs.getInt("availability_id"));
        equipment.setAvailability(availability);


        EquipmentStatus status =
statusDAO.getStatusById(rs.getInt("status_id"));
        equipment.setStatus(status);


        int employeeId = rs.getInt("employee_id");
        if (!rs.wasNull()) {
            Employee employee = employeeDAO.getEmployeeById(employeeId);
            equipment.setEmployee(employee);
        }


        int locationId = rs.getInt("location_id");
        if (!rs.wasNull()) {
            Location location = locationDAO.getLocationById(locationId);
            equipment.setLocation(location);
        }


        return equipment;
    }
}
```

EquipmentService.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
```

```
package ecsapplication.DataAccess;
import ecsapplication.Model.Action;
import ecsapplication.Model.Auditing;
import ecsapplication.Model.Employee;
import ecsapplication.Model.Equipment;
import ecsapplication.Model.EquipmentStatus;
import ecsapplication.Model.Location;
import ecsapplication.Model.NotificationLog;
import ecsapplication.Model.NotificationType;
import ecsapplication.Model.Transaction;
import java.math.BigDecimal;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.List;

/**
 * Service class for equipment management operations
 */
public class EquipmentService {
    private EquipmentDAO equipmentDAO;
    private TransactionDAO transactionDAO;
    private EmployeeDAO employeeDAO;
    private AuditingDAO auditingDAO;
    private NotificationLogDAO notificationLogDAO;
    private NotificationTypeDAO notificationTypeDAO;
    private ActionDAO actionDAO;
```

```
    public EquipmentService() {
        DBManager dbManager = DBManager.getInstance();
        this.equipmentDAO = dbManager.getDAO(EquipmentDAO.class);
        this.transactionDAO = dbManager.getDAO(TransactionDAO.class);
        this.employeeDAO = dbManager.getDAO(EmployeeDAO.class);
        this.auditingDAO = dbManager.getDAO(AuditingDAO.class);
        this.notificationLogDAO =
dbManager.getDAO(NotificationLogDAO.class);
        this.notificationTypeDAO =
dbManager.getDAO(NotificationTypeDAO.class);
        this.actionDAO = dbManager.getDAO(ActionDAO.class);
    }


    /**
     * Check out equipment to an employee
     */
    public boolean checkOutEquipment(int equipmentId, int employeeId,
LocalDate dueDate, Employee actingEmployee) {
        // Get equipment and employee
        Equipment equipment =
equipmentDAO.getEquipmentById(equipmentId);
        Employee employee = employeeDAO.getEmployeeById(employeeId);


        if (equipment == null || employee == null) {
            return false;
        }


        // Check if equipment is available
```

```
        if (equipment.getAvailability().getAvailabilityId() != 1) { // 1 = Available
            return false;
        }

        // Check if employee has the required skill
        if (employee.getSkill().getSkillId() < equipment.getSkill().getSkillId()) {
            return false;
        }

        // Update equipment status
        boolean equipmentUpdated =
equipmentDAO.checkOutEquipment(equipmentId, employeeId);

        if (!equipmentUpdated) {
            return false;
        }

        // Create transaction
        Transaction transaction = new Transaction();
        transaction.setTransactionDate(LocalDateTime.now());
        transaction.setDueDate(dueDate);
        transaction.setEmployee(employee);
        transaction.setEquipment(equipment);

        boolean transactionCreated =
transactionDAO.createTransaction(transaction);
```

```
        if (!transactionCreated) {
            // Rollback equipment status
            equipmentDAO.checkInEquipment(equipmentId);
            return false;
        }

        // Create notification
        NotificationType notificationType =
notificationTypeDAO.getNotificationTypeById(1); // 1 = InitialWarning

        NotificationLog notification = new NotificationLog();
        notification.setMessage("Equipment " + equipment.getEquipmentType()
+ " (ID: " + equipment.getEquipmentId() +
                                ") due on " + dueDate.toString() + ". Please return on
time.");
        notification.setTimestamp(LocalDateTime.now());
        notification.setRead(false);
        notification.setType(notificationType);
        notification.setEmployee(employee);

        notificationLogDAO.createNotificationLog(notification);

        // Create audit log
        Action action = actionDAO.getActionById(11); // 11 = ChangeAvailability

        Auditing audit = new Auditing();
        audit.setTimestamp(LocalDateTime.now());
```

```
        audit.setDetails("Checked out " + equipment.getEquipmentType() + "
(ID: " + equipment.getEquipmentId() +
                        ") to employee " + employee.getFirstName() + " " +
employee.getLastName() +
                        " (ID: " + employee.getEmployeeId() + ")");
        audit.setEmployee(actingEmployee);
        audit.setAction(action);


        auditingDAO.createAuditLog(audit);


        return true;
    }


    /**
     * Return equipment from an employee
     */
    public boolean returnEquipment(int transactionId, Employee
actingEmployee) {
        // Get transaction
        Transaction transaction =
transactionDAO.getTransactionById(transactionId);


        if (transaction == null || transaction.getReturnDate() != null) {
            return false;
        }


        // Update transaction with return date
```

```
        boolean returned = transactionDAO.recordReturn(transactionId,
LocalDate.now());

        if (!returned) {
            return false;
        }

        // Create audit log
        Action action = actionDAO.getActionById(11); // 11 = ChangeAvailability

        Auditing audit = new Auditing();
        audit.setTimestamp(LocalDateTime.now());
        audit.setDetails("Returned " +
transaction.getEquipment().getEquipmentType() +
                         " (ID: " + transaction.getEquipment().getEquipmentId() +
                         ") from employee " +
transaction.getEmployee().getFirstName() +
                         " " + transaction.getEmployee().getLastName() +
                         " (ID: " + transaction.getEmployee().getEmployeeId() + ")");
        audit.setEmployee(actingEmployee);
        audit.setAction(action);

        auditingDAO.createAuditLog(audit);

        return true;
    }

    /**
```

```
     * Generate overdue equipment report
     */
    public List<Transaction> generateOverdueReport() {
        return transactionDAO.getOverdueTransactions();
    }

    /**
     * Change equipment status
     */
    public boolean changeEquipmentStatus(int equipmentId, int statusId,
Employee actingEmployee) {
        Equipment equipment =
equipmentDAO.getEquipmentById(equipmentId);

        if (equipment == null) {
            return false;
        }

        // Get new status
        EquipmentStatusDAO statusDAO =
DBManager.getInstance().getDAO(EquipmentStatusDAO.class);
        EquipmentStatus newStatus = statusDAO.getStatusById(statusId);

        if (newStatus == null) {
            return false;
        }

        // Update equipment status
```

```
        equipment.setStatus(newStatus);
        boolean updated = equipmentDAO.updateEquipment(equipment);

        if (!updated) {
            return false;
        }

        // Create audit log
        Action action = actionDAO.getActionById(10); // 10 = ChangeStatus

        Auditing audit = new Auditing();
        audit.setTimestamp(LocalDateTime.now());
        audit.setDetails("Changed status of " + equipment.getEquipmentType()
+
                         " (ID: " + equipment.getEquipmentId() + ") to " +
newStatus.getStatusName());
        audit.setEmployee(actingEmployee);
        audit.setAction(action);

        auditingDAO.createAuditLog(audit);

        return true;
    }

    /**
     * Change equipment location
     */
```

```
    public boolean changeEquipmentLocation(int equipmentId, int locationId,
Employee actingEmployee) {
        Equipment equipment =
equipmentDAO.getEquipmentById(equipmentId);

        if (equipment == null) {
            return false;
        }

        // Get new location
        LocationDAO locationDAO =
DBManager.getInstance().getDAO(LocationDAO.class);
        Location newLocation = locationDAO.getLocationById(locationId);

        if (newLocation == null) {
            return false;
        }

        // Update equipment location
        equipment.setLocation(newLocation);
        boolean updated = equipmentDAO.updateEquipment(equipment);

        if (!updated) {
            return false;
        }

        // Create audit log
        Action action = actionDAO.getActionById(9); // 9 = ChangeLocation
```

```
        Auditing audit = new Auditing();
        audit.setTimestamp(LocalDateTime.now());
        audit.setDetails("Changed location of " +
equipment.getEquipmentType() +
                        " (ID: " + equipment.getEquipmentId() + ") to " +
newLocation.getLocationName());
        audit.setEmployee(actingEmployee);
        audit.setAction(action);


        auditingDAO.createAuditLog(audit);


        return true;
    }
}
```

EquipmentStatusDAO.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package ecsapplication.DataAccess;
import ecsapplication.Model.EquipmentStatus;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;


public class EquipmentStatusDAO {
    // Create status
    public boolean createStatus(EquipmentStatus status) {
        String sql = "INSERT INTO equipmentstatus (status_name) VALUES (?)";


        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS)) {


            stmt.setString(1, status.getStatusName());


            int affectedRows = stmt.executeUpdate();
            if (affectedRows == 0) {
                return false;
            }
```

```
        try (ResultSet generatedKeys = stmt.getGeneratedKeys()) {
            if (generatedKeys.next()) {
                status.setStatusId(generatedKeys.getInt(1));
                return true;
            } else {
                return false;
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
}

// Get status by ID
public EquipmentStatus getStatusById(int statusId) {
    String sql = "SELECT * FROM equipmentstatus WHERE status_id = ?";

    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, statusId);

        try (ResultSet rs = stmt.executeQuery()) {
            if (rs.next()) {
                return extractStatusFromResultSet(rs);
            }
```

```
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return null;
}

// Get all statuses
public List<EquipmentStatus> getAllStatuses() {
    List<EquipmentStatus> statuses = new ArrayList<>();
    String sql = "SELECT * FROM equipmentstatus";

    try (Connection conn = DatabaseConnection.getConnection();
         Statement stmt = conn.createStatement();
         ResultSet rs = stmt.executeQuery(sql)) {

        while (rs.next()) {
            statuses.add(extractStatusFromResultSet(rs));
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return statuses;
}
```

```
    // Update status
    public boolean updateStatus(EquipmentStatus status) {
        String sql = "UPDATE equipmentstatus SET status_name = ? WHERE
status_id = ?";

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setString(1, status.getStatusName());
            stmt.setInt(2, status.getStatusId());

            int affectedRows = stmt.executeUpdate();
            return affectedRows > 0;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }

    // Delete status
    public boolean deleteStatus(int statusId) {
        String sql = "DELETE FROM equipmentstatus WHERE status_id = ?";

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setInt(1, statusId);
```

```
            int affectedRows = stmt.executeUpdate();
            return affectedRows > 0;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }

    // Get status by name
    public EquipmentStatus getStatusByName(String statusName) {
        String sql = "SELECT * FROM equipmentstatus WHERE status_name
= ?";

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setString(1, statusName);

            try (ResultSet rs = stmt.executeQuery()) {
                if (rs.next()) {
                    return extractStatusFromResultSet(rs);
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
```

```
        return null;
    }

    // Helper method to extract EquipmentStatus from ResultSet
    private EquipmentStatus extractStatusFromResultSet(ResultSet rs) throws SQLException {
        EquipmentStatus status = new EquipmentStatus();
        status.setStatusId(rs.getInt("status_id"));
        status.setStatusName(rs.getString("status_name"));
        return status;
    }
}
```

LocationDAO.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
```

```
 */
package ecsapplication.DataAccess;
import ecsapplication.Model.Location;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class LocationDAO {
    // Create location
    public boolean createLocation(Location location) {
        String sql = "INSERT INTO location (location_name) VALUES (?)";

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql,
Statement.RETURN_GENERATED_KEYS)) {

            stmt.setString(1, location.getLocationName());

            int affectedRows = stmt.executeUpdate();
            if (affectedRows == 0) {
                return false;
            }

            try (ResultSet generatedKeys = stmt.getGeneratedKeys()) {
                if (generatedKeys.next()) {
                    location.setLocationId(generatedKeys.getInt(1));
                    return true;
```

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

// Get location by ID
public Location getLocationById(int locationId) {
    String sql = "SELECT * FROM location WHERE location_id = ?";

    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, locationId);

        try (ResultSet rs = stmt.executeQuery()) {
            if (rs.next()) {
                return extractLocationFromResultSet(rs);
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
```

```
        return null;
    }

    // Get all locations
    public List<Location> getAllLocations() {
        List<Location> locations = new ArrayList<>();
        String sql = "SELECT * FROM location";

        try (Connection conn = DatabaseConnection.getConnection();
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(sql)) {

            while (rs.next()) {
                locations.add(extractLocationFromResultSet(rs));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return locations;
    }

    // Update location
    public boolean updateLocation(Location location) {
        String sql = "UPDATE location SET location_name = ? WHERE location_id
= ?";
```

```
    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setString(1, location.getLocationName());
        stmt.setInt(2, location.getLocationId());

        int affectedRows = stmt.executeUpdate();
        return affectedRows > 0;
    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
}

// Delete location
public boolean deleteLocation(int locationId) {
    String sql = "DELETE FROM location WHERE location_id = ?";

    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, locationId);

        int affectedRows = stmt.executeUpdate();
        return affectedRows > 0;
    } catch (SQLException e) {
```

```
            e.printStackTrace();
            return false;
        }
    }

    // Helper method to extract Location from ResultSet
    private Location extractLocationFromResultSet(ResultSet rs) throws
SQLException {
        Location location = new Location();
        location.setLocationId(rs.getInt("location_id"));
        location.setLocationName(rs.getString("location_name"));
        return location;
    }
}
```

NotificationLogDAO.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-
default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit
this template
 */
package ecsapplication.DataAccess;
import ecsapplication.Model.Employee;
```

```
import ecsapplication.Model.NotificationLog;
import ecsapplication.Model.NotificationType;
import java.sql.*;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;

public class NotificationLogDAO {
    private NotificationTypeDAO typeDAO = new NotificationTypeDAO();
    private EmployeeDAO employeeDAO = new EmployeeDAO();

    // Create notification log
    public boolean createNotificationLog(NotificationLog log) {
        String sql = "INSERT INTO notificationlog (message, timestamp, is_read,
type_id, employee_id) " +
                     "VALUES (?, ?, ?, ?, ?)";

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql,
Statement.RETURN_GENERATED_KEYS)) {

            stmt.setString(1, log.getMessage());

            if (log.getTimestamp() != null) {
                stmt.setTimestamp(2, Timestamp.valueOf(log.getTimestamp()));
            } else {
                stmt.setTimestamp(2, Timestamp.valueOf(LocalDateTime.now()));
```

```
        }

        stmt.setBoolean(3, log.isRead());
        stmt.setInt(4, log.getType().getTypeId());
        stmt.setInt(5, log.getEmployee().getEmployeeId());

        int affectedRows = stmt.executeUpdate();
        if (affectedRows == 0) {
            return false;
        }

        try (ResultSet generatedKeys = stmt.getGeneratedKeys()) {
            if (generatedKeys.next()) {
                log.setNotificationId(generatedKeys.getInt(1));
                return true;
            } else {
                return false;
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
}

// Get notification log by ID
public NotificationLog getNotificationLogById(int notificationId) {
```

```
    String sql = "SELECT * FROM notificationlog WHERE notification_id = ?";

    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, notificationId);

        try (ResultSet rs = stmt.executeQuery()) {
            if (rs.next()) {
                return extractNotificationLogFromResultSet(rs);
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return null;
}

// Get all notification logs
public List<NotificationLog> getAllNotificationLogs() {
    List<NotificationLog> logs = new ArrayList<>();
    String sql = "SELECT * FROM notificationlog";

    try (Connection conn = DatabaseConnection.getConnection();
         Statement stmt = conn.createStatement();
         ResultSet rs = stmt.executeQuery(sql)) {
```

```
            while (rs.next()) {
                logs.add(extractNotificationLogFromResultSet(rs));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return logs;
    }

    // Get notification logs by employee
    public List<NotificationLog> getNotificationLogsByEmployee(int
employeeId) {
        List<NotificationLog> logs = new ArrayList<>();
        String sql = "SELECT * FROM notificationlog WHERE employee_id = ?";

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setInt(1, employeeId);

            try (ResultSet rs = stmt.executeQuery()) {
                while (rs.next()) {
                    logs.add(extractNotificationLogFromResultSet(rs));
                }
            }
```

```
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return logs;
    }

    // Get unread notification logs by employee
    public List<NotificationLog> getUnreadNotificationLogsByEmployee(int
employeeId) {
        List<NotificationLog> logs = new ArrayList<>();
        String sql = "SELECT * FROM notificationlog WHERE employee_id = ?
AND is_read = 0";

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setInt(1, employeeId);

            try (ResultSet rs = stmt.executeQuery()) {
                while (rs.next()) {
                    logs.add(extractNotificationLogFromResultSet(rs));
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
```

```
        return logs;
    }

    // Mark notification as read
    public boolean markNotificationAsRead(int notificationId) {
        String sql = "UPDATE notificationlog SET is_read = 1 WHERE
notification_id = ?";

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setInt(1, notificationId);

            int affectedRows = stmt.executeUpdate();
            return affectedRows > 0;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }

    // Update notification log
    public boolean updateNotificationLog(NotificationLog log) {
        String sql = "UPDATE notificationlog SET message = ?, timestamp = ?,
is_read = ?, " +
                     "type_id = ?, employee_id = ? WHERE notification_id = ?";
```

```
    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setString(1, log.getMessage());
        stmt.setTimestamp(2, Timestamp.valueOf(log.getTimestamp()));
        stmt.setBoolean(3, log.isRead());
        stmt.setInt(4, log.getType().getTypeId());
        stmt.setInt(5, log.getEmployee().getEmployeeId());
        stmt.setInt(6, log.getNotificationId());

        int affectedRows = stmt.executeUpdate();
        return affectedRows > 0;
    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
}

// Delete notification log
public boolean deleteNotificationLog(int notificationId) {
    String sql = "DELETE FROM notificationlog WHERE notification_id = ?";

    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, notificationId);
```

```
            int affectedRows = stmt.executeUpdate();
            return affectedRows > 0;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }

    // Helper method to extract NotificationLog from ResultSet
    private NotificationLog extractNotificationLogFromResultSet(ResultSet rs)
throws SQLException {
        NotificationLog log = new NotificationLog();
        log.setNotificationId(rs.getInt("notification_id"));
        log.setMessage(rs.getString("message"));
        log.setTimestamp(rs.getTimestamp("timestamp").toLocalDateTime());
        log.setRead(rs.getBoolean("is_read"));

        NotificationType type =
typeDAO.getNotificationTypeById(rs.getInt("type_id"));
        log.setType(type);

        Employee employee =
employeeDAO.getEmployeeById(rs.getInt("employee_id"));
        log.setEmployee(employee);

        return log;
    }
```

```
}
```

NotificationTypeDAO.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package ecsapplication.DataAccess;
import ecsapplication.Model.NotificationType;
import java.sql.*;
import java.util.ArrayList;
```

```
import java.util.List;


public class NotificationTypeDAO {
    // Create notification type
    public boolean createNotificationType(NotificationType type) {
        String sql = "INSERT INTO notificationtype (type_name) VALUES (?)";


        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql,
Statement.RETURN_GENERATED_KEYS)) {


            stmt.setString(1, type.getTypeName());


            int affectedRows = stmt.executeUpdate();
            if (affectedRows == 0) {
                return false;
            }


            try (ResultSet generatedKeys = stmt.getGeneratedKeys()) {
                if (generatedKeys.next()) {
                    type.setTypeId(generatedKeys.getInt(1));
                    return true;
                } else {
                    return false;
                }
            }
        } catch (SQLException e) {
```

```
        e.printStackTrace();
        return false;
    }
}

// Get notification type by ID
public NotificationType getNotificationTypeById(int typeId) {
    String sql = "SELECT * FROM notificationtype WHERE type_id = ?";

    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, typeId);

        try (ResultSet rs = stmt.executeQuery()) {
            if (rs.next()) {
                return extractNotificationTypeFromResultSet(rs);
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return null;
}

// Get all notification types
```

```
    public List<NotificationType> getAllNotificationTypes() {
        List<NotificationType> types = new ArrayList<>();
        String sql = "SELECT * FROM notificationtype";

        try (Connection conn = DatabaseConnection.getConnection();
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(sql)) {

            while (rs.next()) {
                types.add(extractNotificationTypeFromResultSet(rs));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return types;
    }

    // Update notification type
    public boolean updateNotificationType(NotificationType type) {
        String sql = "UPDATE notificationtype SET type_name = ? WHERE
type_id = ?";

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setString(1, type.getTypeName());
```

```
            stmt.setInt(2, type.getTypeId());

            int affectedRows = stmt.executeUpdate();
            return affectedRows > 0;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }

    // Delete notification type
    public boolean deleteNotificationType(int typeId) {
        String sql = "DELETE FROM notificationtype WHERE type_id = ?";

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setInt(1, typeId);

            int affectedRows = stmt.executeUpdate();
            return affectedRows > 0;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }
```

```
    // Helper method to extract NotificationType from ResultSet
    private NotificationType extractNotificationTypeFromResultSet(ResultSet rs) throws SQLException {
        NotificationType type = new NotificationType();
        type.setTypeId(rs.getInt("type_id"));
        type.setTypeName(rs.getString("type_name"));
        return type;
    }
}
```

ReportingService.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package ecsapplication.DataAccess;
import ecsapplication.Model.Action;
import ecsapplication.Model.Auditing;
import ecsapplication.Model.Employee;
import ecsapplication.Model.Equipment;
import ecsapplication.Model.Transaction;
import java.time.LocalDate;
import java.time.LocalDateTime;
```

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * Service class for generating various system reports
 */
public class ReportingService {
    private TransactionDAO transactionDAO;
    private EquipmentDAO equipmentDAO;
    private EmployeeDAO employeeDAO;
    private AuditingDAO auditingDAO;
    private ActionDAO actionDAO;

    public ReportingService() {
        DBManager dbManager = DBManager.getInstance();
        this.transactionDAO = dbManager.getDAO(TransactionDAO.class);
        this.equipmentDAO = dbManager.getDAO(EquipmentDAO.class);
        this.employeeDAO = dbManager.getDAO(EmployeeDAO.class);
        this.auditingDAO = dbManager.getDAO(AuditingDAO.class);
        this.actionDAO = dbManager.getDAO(ActionDAO.class);
    }

    /**
     * Generate equipment usage report
     */
```

```
    public List<Transaction> generateUsageReport(LocalDate startDate,
LocalDate endDate, Employee actingEmployee) {
        List<Transaction> allTransactions =
transactionDAO.getAllTransactions();
        List<Transaction> filteredTransactions = new ArrayList<>();

        LocalDateTime startDateTime = startDate.atStartOfDay();
        LocalDateTime endDateTime = endDate.atTime(23, 59, 59);

        // Filter transactions by date range
        for (Transaction transaction : allTransactions) {
            if (transaction.getTransactionDate().isAfter(startDateTime) &&
                transaction.getTransactionDate().isBefore(endDateTime)) {
                filteredTransactions.add(transaction);
            }
        }

        // Create audit log
        Action action = actionDAO.getActionById(5); // 5 = UsageReport

        Auditing audit = new Auditing();
        audit.setTimestamp(LocalDateTime.now());
        audit.setDetails("Generated equipment usage report from " + startDate
+ " to " + endDate);
        audit.setEmployee(actingEmployee);
        audit.setAction(action);

        auditingDAO.createAuditLog(audit);
```

```
        return filteredTransactions;
    }

    /**
     * Generate overdue equipment report
     */
    public List<Transaction> generateOverdueReport(Employee
actingEmployee) {
        List<Transaction> overdueTransactions =
transactionDAO.getOverdueTransactions();

        // Create audit log
        Action action = actionDAO.getActionById(6); // 6 = OverdueReport

        Auditing audit = new Auditing();
        audit.setTimestamp(LocalDateTime.now());
        audit.setDetails("Generated overdue equipment report as of " +
LocalDate.now());
        audit.setEmployee(actingEmployee);
        audit.setAction(action);

        auditingDAO.createAuditLog(audit);

        return overdueTransactions;
    }

    /**
```

```
     * Generate equipment status report
     */
    public Map<String, List<Equipment>> generateStatusReport(Employee
actingEmployee) {
        List<Equipment> allEquipment = equipmentDAO.getAllEquipment();
        Map<String, List<Equipment>> statusReport = new HashMap<>();

        // Group equipment by status
        for (Equipment equipment : allEquipment) {
            String statusName = equipment.getStatus().getStatusName();

            if (!statusReport.containsKey(statusName)) {
                statusReport.put(statusName, new ArrayList<>());
            }

            statusReport.get(statusName).add(equipment);
        }

        // Create audit log
        Action action = actionDAO.getActionById(8); // 8 = StatusReport

        Auditing audit = new Auditing();
        audit.setTimestamp(LocalDateTime.now());
        audit.setDetails("Generated equipment status report");
        audit.setEmployee(actingEmployee);
        audit.setAction(action);
```

```
        auditingDAO.createAuditLog(audit);


        return statusReport;
    }


    /**
     * Generate activity report for an employee
     */
    public List<Transaction> generateActivityReport(int employeeId,
Employee actingEmployee) {
        List<Transaction> employeeTransactions =
transactionDAO.getTransactionsByEmployee(employeeId);


        // Create audit log
        Action action = actionDAO.getActionById(7); // 7 = ActivityReport


        Employee targetEmployee =
employeeDAO.getEmployeeById(employeeId);
        String employeeName = targetEmployee != null ?
                        targetEmployee.getFirstName() + " " +
targetEmployee.getLastName() :
                        "Unknown";


        Auditing audit = new Auditing();
        audit.setTimestamp(LocalDateTime.now());
        audit.setDetails("Viewed activity log for employee ID " + employeeId +
" (" + employeeName + ")");
        audit.setEmployee(actingEmployee);
```

```
        audit.setAction(action);

        auditingDAO.createAuditLog(audit);

        return employeeTransactions;
    }
}
```

SkillDAO.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package ecsapplication.DataAccess;

import ecsapplication.Model.Skill;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class SkillDAO {
    // Create skill
    public boolean createSkill(Skill skill) {
```

```
        String sql = "INSERT INTO skills (skill_level) VALUES (?)";

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql,
Statement.RETURN_GENERATED_KEYS)) {

            stmt.setString(1, skill.getSkillLevel());

            int affectedRows = stmt.executeUpdate();
            if (affectedRows == 0) {
                return false;
            }

            try (ResultSet generatedKeys = stmt.getGeneratedKeys()) {
                if (generatedKeys.next()) {
                    skill.setSkillId(generatedKeys.getInt(1));
                    return true;
                } else {
                    return false;
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }
```

```
// Get skill by ID
public Skill getSkillById(int skillId) {
    String sql = "SELECT * FROM skills WHERE skill_id = ?";

    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, skillId);

        try (ResultSet rs = stmt.executeQuery()) {
            if (rs.next()) {
                return extractSkillFromResultSet(rs);
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return null;
}

// Get all skills
public List<Skill> getAllSkills() {
    List<Skill> skills = new ArrayList<>();
    String sql = "SELECT * FROM skills";

    try (Connection conn = DatabaseConnection.getConnection();
```

```
         Statement stmt = conn.createStatement();
         ResultSet rs = stmt.executeQuery(sql)) {

        while (rs.next()) {
            skills.add(extractSkillFromResultSet(rs));
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return skills;
}

// Update skill
public boolean updateSkill(Skill skill) {
    String sql = "UPDATE skills SET skill_level = ? WHERE skill_id = ?";

    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setString(1, skill.getSkillLevel());
        stmt.setInt(2, skill.getSkillId());

        int affectedRows = stmt.executeUpdate();
        return affectedRows > 0;
    } catch (SQLException e) {
        e.printStackTrace();
```

```
        return false;
    }
}

// Delete skill
public boolean deleteSkill(int skillId) {
    String sql = "DELETE FROM skills WHERE skill_id = ?";

    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, skillId);

        int affectedRows = stmt.executeUpdate();
        return affectedRows > 0;
    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
}

// Get skill by level
public Skill getSkillByLevel(String skillLevel) {
    String sql = "SELECT * FROM skills WHERE skill_level = ?";

    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {
```

```
            stmt.setString(1, skillLevel);

            try (ResultSet rs = stmt.executeQuery()) {
                if (rs.next()) {
                    Skill skill = new Skill();
                    skill.setSkillId(rs.getInt("skill_id"));
                    skill.setSkillLevel(rs.getString("skill_level"));
                    return skill;
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return null;
    }

    // Helper method to extract Skill from ResultSet
    private Skill extractSkillFromResultSet(ResultSet rs) throws SQLException {
        Skill skill = new Skill();
        skill.setSkillId(rs.getInt("skill_id"));
        skill.setSkillLevel(rs.getString("skill_level"));
        return skill;
    }
}
```

SupervisorDAO.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package ecsapplication.DataAccess;
import ecsapplication.Model.Employee;
import ecsapplication.Model.Supervisor;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class SupervisorDAO {
    private EmployeeDAO employeeDAO = new EmployeeDAO();

    // Create supervisor
    public boolean createSupervisor(Supervisor supervisor) {
        String sql = "INSERT INTO supervisor (employee_id) VALUES (?)";
```

```
        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql,
Statement.RETURN_GENERATED_KEYS)) {

            stmt.setInt(1, supervisor.getEmployee().getEmployeeId());

            int affectedRows = stmt.executeUpdate();
            if (affectedRows == 0) {
                return false;
            }

            try (ResultSet generatedKeys = stmt.getGeneratedKeys()) {
                if (generatedKeys.next()) {
                    supervisor.setSupervisorId(generatedKeys.getInt(1));
                    return true;
                } else {
                    return false;
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }

    // Get supervisor by ID
    public Supervisor getSupervisorById(int supervisorId) {
```

```
    String sql = "SELECT * FROM supervisor WHERE supervisor_id = ?";

    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, supervisorId);

        try (ResultSet rs = stmt.executeQuery()) {
            if (rs.next()) {
                return extractSupervisorFromResultSet(rs);
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return null;
}

// Get supervisor by employee ID
public Supervisor getSupervisorByEmployeeId(int employeeId) {
    String sql = "SELECT * FROM supervisor WHERE employee_id = ?";

    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, employeeId);
```

```
        try (ResultSet rs = stmt.executeQuery()) {
            if (rs.next()) {
                return extractSupervisorFromResultSet(rs);
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return null;
}

// Get all supervisors
public List<Supervisor> getAllSupervisors() {
    List<Supervisor> supervisors = new ArrayList<>();
    String sql = "SELECT * FROM supervisor";

    try (Connection conn = DatabaseConnection.getConnection();
         Statement stmt = conn.createStatement();
         ResultSet rs = stmt.executeQuery(sql)) {

        while (rs.next()) {
            supervisors.add(extractSupervisorFromResultSet(rs));
        }
    } catch (SQLException e) {
        e.printStackTrace();
```

```
        }

        return supervisors;
    }

    // Update supervisor
    public boolean updateSupervisor(Supervisor supervisor) {
        String sql = "UPDATE supervisor SET employee_id = ? WHERE
supervisor_id = ?";

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setInt(1, supervisor.getEmployee().getEmployeeId());
            stmt.setInt(2, supervisor.getSupervisorId());

            int affectedRows = stmt.executeUpdate();
            return affectedRows > 0;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }

    // Delete supervisor
    public boolean deleteSupervisor(int supervisorId) {
        String sql = "DELETE FROM supervisor WHERE supervisor_id = ?";
```

```
        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setInt(1, supervisorId);

            int affectedRows = stmt.executeUpdate();
            return affectedRows > 0;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }

    // Helper method to extract Supervisor from ResultSet
    private Supervisor extractSupervisorFromResultSet(ResultSet rs) throws
SQLException {
        Supervisor supervisor = new Supervisor();
        supervisor.setSupervisorId(rs.getInt("supervisor_id"));

        Employee employee =
employeeDAO.getEmployeeById(rs.getInt("employee_id"));
        supervisor.setEmployee(employee);

        return supervisor;
    }
}
```

TransactionDAO.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package ecsapplication.DataAccess;
import ecsapplication.Model.Employee;
import ecsapplication.Model.Equipment;
import ecsapplication.Model.Transaction;
import java.sql.*;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;

public class TransactionDAO {
    private EmployeeDAO employeeDAO = new EmployeeDAO();
    private EquipmentDAO equipmentDAO = new EquipmentDAO();

    // Create transaction
    public boolean createTransaction(Transaction transaction) {
```

```
        String sql = "INSERT INTO transaction (transaction_date, due_date,
return_date, employee_id, equipment_id) " +
                     "VALUES (?, ?, ?, ?, ?)";


        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql,
Statement.RETURN_GENERATED_KEYS)) {


            if (transaction.getTransactionDate() != null) {
                stmt.setTimestamp(1,
Timestamp.valueOf(transaction.getTransactionDate()));
            } else {
                stmt.setTimestamp(1, Timestamp.valueOf(LocalDateTime.now()));
            }


            stmt.setDate(2, Date.valueOf(transaction.getDueDate()));


            if (transaction.getReturnDate() != null) {
                stmt.setDate(3, Date.valueOf(transaction.getReturnDate()));
            } else {
                stmt.setNull(3, Types.DATE);
            }


            stmt.setInt(4, transaction.getEmployee().getEmployeeId());
            stmt.setInt(5, transaction.getEquipment().getEquipmentId());


            int affectedRows = stmt.executeUpdate();
            if (affectedRows == 0) {
```

```
            return false;
        }

        try (ResultSet generatedKeys = stmt.getGeneratedKeys()) {
            if (generatedKeys.next()) {
                transaction.setTransactionId(generatedKeys.getInt(1));
                return true;
            } else {
                return false;
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
}

// Get transaction by ID
public Transaction getTransactionById(int transactionId) {
    String sql = "SELECT * FROM transaction WHERE transaction_id = ?";

    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, transactionId);

        try (ResultSet rs = stmt.executeQuery()) {
```

```
            if (rs.next()) {
                return extractTransactionFromResultSet(rs);
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return null;
}

// Get all transactions
public List<Transaction> getAllTransactions() {
    List<Transaction> transactions = new ArrayList<>();
    String sql = "SELECT * FROM transaction";

    try (Connection conn = DatabaseConnection.getConnection();
         Statement stmt = conn.createStatement();
         ResultSet rs = stmt.executeQuery(sql)) {

        while (rs.next()) {
            transactions.add(extractTransactionFromResultSet(rs));
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
```

```
    return transactions;
}

// Get transactions by employee
public List<Transaction> getTransactionsByEmployee(int employeeId) {
    List<Transaction> transactions = new ArrayList<>();
    String sql = "SELECT * FROM transaction WHERE employee_id = ?";

    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, employeeId);

        try (ResultSet rs = stmt.executeQuery()) {
            while (rs.next()) {
                transactions.add(extractTransactionFromResultSet(rs));
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return transactions;
}

// Get transactions by equipment
public List<Transaction> getTransactionsByEquipment(int equipmentId) {
```

```
        List<Transaction> transactions = new ArrayList<>();
        String sql = "SELECT * FROM transaction WHERE equipment_id = ?";

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setInt(1, equipmentId);

            try (ResultSet rs = stmt.executeQuery()) {
                while (rs.next()) {
                    transactions.add(extractTransactionFromResultSet(rs));
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return transactions;
    }

    // Get overdue transactions
    public List<Transaction> getOverdueTransactions() {
        List<Transaction> transactions = new ArrayList<>();
        String sql = "SELECT * FROM transaction WHERE return_date IS NULL
AND due_date < CURDATE()";

        try (Connection conn = DatabaseConnection.getConnection();
```

```
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(sql)) {

            while (rs.next()) {
                transactions.add(extractTransactionFromResultSet(rs));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return transactions;
    }

    // Record equipment return
    public boolean recordReturn(int transactionId, LocalDate returnDate) {
        String sql = "UPDATE transaction SET return_date = ? WHERE
transaction_id = ?";

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setDate(1, Date.valueOf(returnDate));
            stmt.setInt(2, transactionId);

            int affectedRows = stmt.executeUpdate();

            if (affectedRows > 0) {
```

```
            // Get the equipment ID
            String getEquipmentSql = "SELECT equipment_id FROM
transaction WHERE transaction_id = ?";
            try (PreparedStatement getEquipStmt =
conn.prepareStatement(getEquipmentSql)) {
                getEquipStmt.setInt(1, transactionId);
                try (ResultSet rs = getEquipStmt.executeQuery()) {
                    if (rs.next()) {
                        int equipmentId = rs.getInt("equipment_id");

                        // Update equipment availability
                        EquipmentDAO equipmentDAO = new EquipmentDAO();
                        return equipmentDAO.checkInEquipment(equipmentId);
                    }
                }
            }
        }

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

// Update transaction
public boolean updateTransaction(Transaction transaction) {
```

```
        String sql = "UPDATE transaction SET transaction_date = ?, due_date
= ?, return_date = ?, " +
                     "employee_id = ?, equipment_id = ? WHERE transaction_id
= ?";


        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {


            stmt.setTimestamp(1,
Timestamp.valueOf(transaction.getTransactionDate()));
            stmt.setDate(2, Date.valueOf(transaction.getDueDate()));


            if (transaction.getReturnDate() != null) {
                stmt.setDate(3, Date.valueOf(transaction.getReturnDate()));
            } else {
                stmt.setNull(3, Types.DATE);
            }


            stmt.setInt(4, transaction.getEmployee().getEmployeeId());
            stmt.setInt(5, transaction.getEquipment().getEquipmentId());
            stmt.setInt(6, transaction.getTransactionId());


            int affectedRows = stmt.executeUpdate();
            return affectedRows > 0;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
```

```
    }

    // Delete transaction
    public boolean deleteTransaction(int transactionId) {
        String sql = "DELETE FROM transaction WHERE transaction_id = ?";

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setInt(1, transactionId);

            int affectedRows = stmt.executeUpdate();
            return affectedRows > 0;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }

    // Get active transactions by employee
    public List<Transaction> getActiveTransactionsByEmployee(int
employeeId) {
        List<Transaction> transactions = new ArrayList<>();
        String sql = "SELECT * FROM transaction WHERE employee_id = ? AND
return_date IS NULL";

        try (Connection conn = DatabaseConnection.getConnection();
```

```
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setInt(1, employeeId);

            try (ResultSet rs = stmt.executeQuery()) {
                while (rs.next()) {
                    transactions.add(extractTransactionFromResultSet(rs));
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return transactions;
    }

    // Get active transaction for equipment
    public Transaction getActiveTransaction(String equipmentId) {
        String sql = "SELECT * FROM transaction WHERE equipment_id = ? AND
return_date IS NULL";

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setString(1, equipmentId);

            try (ResultSet rs = stmt.executeQuery()) {
```

```
            if (rs.next()) {
                return extractTransactionFromResultSet(rs);
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return null;
}

// Get active transaction for equipment
public Transaction getActiveTransactionByEquipment(int equipmentId) {
    String sql = "SELECT * FROM transaction WHERE equipment_id = ? AND
return_date IS NULL";

    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, equipmentId);

        try (ResultSet rs = stmt.executeQuery()) {
            if (rs.next()) {
                return extractTransactionFromResultSet(rs);
            }
        }
    } catch (SQLException e) {
```

```
            e.printStackTrace();
        }

        return null;
    }

    // Update transaction
    public boolean update(Transaction transaction) {
        String sql = "UPDATE transaction SET return_date = ?, status = ?
WHERE transaction_id = ?";

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            if (transaction.getReturnDate() != null) {
                stmt.setDate(1,
java.sql.Date.valueOf(transaction.getReturnDate()));
            } else {
                stmt.setNull(1, Types.DATE);
            }

            stmt.setString(2, "Completed");
            stmt.setInt(3, transaction.getTransactionId());

            return stmt.executeUpdate() > 0;
        } catch (SQLException e) {
            e.printStackTrace();
```

```
            return false;
        }
    }

    // Helper method to extract Transaction from ResultSet
    private Transaction extractTransactionFromResultSet(ResultSet rs) throws
SQLException {
        Transaction transaction = new Transaction();
        transaction.setTransactionId(rs.getInt("transaction_id"));

        Timestamp transactionDate = rs.getTimestamp("transaction_date");
        transaction.setTransactionDate(transactionDate.toLocalDateTime());

        transaction.setDueDate(rs.getDate("due_date").toLocalDate());

        Date returnDate = rs.getDate("return_date");
        if (returnDate != null) {
            transaction.setReturnDate(returnDate.toLocalDate());
        }

        Employee employee =
employeeDAO.getEmployeeById(rs.getInt("employee_id"));
        transaction.setEmployee(employee);

        Equipment equipment =
equipmentDAO.getEquipmentById(rs.getInt("equipment_id"));
        transaction.setEquipment(equipment);
```

```
        return transaction;
    }
}
```

ValueLossDAO.java

```
package ecsapplication.DataAccess;

import java.math.BigDecimal;
```

```
import java.util.Map;


public interface ValueLossDAO {
    void processTerminationEquipmentLoss(int employeeId, String
employeeName);
    BigDecimal getTotalInventoryValue();
    BigDecimal getTotalLostValue();
    Map<String, BigDecimal> getMonthlyLossTrend();
}
```

ValueLossDAOImpl.java

```
package ecsapplication.DataAccess;


import java.math.BigDecimal;
```

```
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.LinkedHashMap;
import java.util.Map;

public class ValueLossDAOImpl implements ValueLossDAO {
    private final DBManager dbManager;

    public ValueLossDAOImpl(DBManager dbManager) {
        this.dbManager = dbManager;
    }

    @Override
    public void processTerminationEquipmentLoss(int employeeId, String
employeeName) {
        try (Connection conn = dbManager.getConnection();
             CallableStatement stmt = conn.prepareCall("{CALL
process_termination_equipment_loss(?, ?)}")) {

            stmt.setInt(1, employeeId);
            stmt.setString(2, employeeName);
            stmt.execute();

        } catch (SQLException e) {
```

```
            throw new RuntimeException("Error processing equipment loss: " +
e.getMessage(), e);
        }
    }

    @Override
    public BigDecimal getTotalInventoryValue() {
        try (Connection conn = dbManager.getConnection();
             PreparedStatement stmt = conn.prepareStatement(
                 "SELECT total_value FROM total_inventory_value ORDER BY
last_updated DESC LIMIT 1")) {

            ResultSet rs = stmt.executeQuery();
            if (rs.next()) {
                return rs.getBigDecimal("total_value");
            }
            return BigDecimal.ZERO;

        } catch (SQLException e) {
            throw new RuntimeException("Error getting total inventory value: " +
e.getMessage(), e);
        }
    }

    @Override
    public BigDecimal getTotalLostValue() {
        try (Connection conn = dbManager.getConnection();
             PreparedStatement stmt = conn.prepareStatement(
```

```
            "SELECT COALESCE(SUM(value_lost), 0) as total_lost_value FROM
value_loss")) {

            ResultSet rs = stmt.executeQuery();
            if (rs.next()) {
                return rs.getBigDecimal("total_lost_value");
            }
            return BigDecimal.ZERO;

        } catch (SQLException e) {
            throw new RuntimeException("Error getting total lost value: " +
e.getMessage(), e);
        }
    }

    @Override
    public Map<String, BigDecimal> getMonthlyLossTrend() {
        Map<String, BigDecimal> monthlyTrend = new LinkedHashMap<>();
        String query = "SELECT DATE_FORMAT(termination_date, '%Y-%m') as
month, SUM(value_lost) as total_loss " +
                       "FROM value_loss " +
                       "GROUP BY DATE_FORMAT(termination_date, '%Y-%m') " +
                       "ORDER BY month DESC " +
                       "LIMIT 12";

        try (Connection conn = dbManager.getConnection();
             PreparedStatement stmt = conn.prepareStatement(query);
             ResultSet rs = stmt.executeQuery()) {
```

```
            while (rs.next()) {
                String month = rs.getString("month");
                BigDecimal totalLoss = rs.getBigDecimal("total_loss");
                monthlyTrend.put(month, totalLoss);
            }
        } catch (SQLException e) {
            throw new RuntimeException("Failed to retrieve monthly loss trend: "
+ e.getMessage(), e);
        }
        return monthlyTrend;
    }
}
```

Action.java

```
package ecsapplication.Model;

import java.time.LocalDateTime;

/*
```

```
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
// Action.java
public class Action {
    private int actionId;
    private String actionName;
    private String actionType;
    private Employee employee;
    private Equipment equipment;
    private LocalDateTime actionDate;

    // Constructors
    public Action() {}

    public Action(int actionId, String actionName) {
        this.actionId = actionId;
        this.actionName = actionName;
    }

    // Getters and Setters
    public int getActionId() {
        return actionId;
    }
```

```
public void setActionId(int actionId) {
    this.actionId = actionId;
}

public String getActionName() {
    return actionName;
}

public void setActionName(String actionName) {
    this.actionName = actionName;
}

public String getActionType() {
    return actionType;
}

public void setActionType(String actionType) {
    this.actionType = actionType;
}

public Employee getEmployee() {
    return employee;
}

public int getEmployeeId() {
    return employee != null ? employee.getEmployeeId() : 0;
}
```

```
public void setEmployee(Employee employee) {
    this.employee = employee;
}

public Equipment getEquipment() {
    return equipment;
}

public void setEquipment(Equipment equipment) {
    this.equipment = equipment;
}

public void setEquipmentId(String equipmentId) {
    if (this.equipment == null) {
        this.equipment = new Equipment();
    }
    this.equipment.setEquipmentId(Integer.parseInt(equipmentId));
}

public LocalDateTime getActionDate() {
    return actionDate;
}

public void setActionDate(LocalDateTime actionDate) {
    this.actionDate = actionDate;
}
```

```
    @Override
    public String toString() {
        return "Action{" +
                "actionId=" + actionId +
                ", actionName='" + actionName + '\'' +
                ", actionType='" + actionType + '\'' +
                ", employee=" + employee +
                ", equipment=" + equipment +
                ", actionDate=" + actionDate +
                '}';
    }
}
```

Auditing.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package ecsapplication.Model;
import java.time.LocalDateTime;
```

```
import java.util.Date;
/**
 *
 * @author Ughon
 */
public class Auditing {
    private int logId;
    private LocalDateTime timestamp;
    private String details;
    private Action action;
    private Employee employee;
    private String actionType;
    private Equipment equipment;
    private Date auditDate;

    // Constructors
    public Auditing() {}

    public Auditing(int logId, LocalDateTime timestamp, String details,
                    Action action, Employee employee) {
        this.logId = logId;
        this.timestamp = timestamp;
        this.details = details;
        this.action = action;
        this.employee = employee;
    }
```

```
// Getters and Setters
public int getLogId() {
    return logId;
}

public void setLogId(int logId) {
    this.logId = logId;
}

public LocalDateTime getTimestamp() {
    return timestamp;
}

public void setTimestamp(LocalDateTime timestamp) {
    this.timestamp = timestamp;
}

public String getDetails() {
    return details;
}

public void setDetails(String details) {
    this.details = details;
}

public Action getAction() {
    return action;
```

```
}

public void setAction(Action action) {
    this.action = action;
}

public Employee getEmployee() {
    return employee;
}

public void setEmployee(Employee employee) {
    this.employee = employee;
}

// Additional setters
public void setAction(String action) {
    this.actionType = action;
}

public void setEmployeeId(int employeeId) {
    if (this.employee == null) {
        this.employee = new Employee();
    }
    this.employee.setEmployeeId(employeeId);
}

public void setEquipmentId(String equipmentId) {
```

```
        if (this.equipment == null) {
            this.equipment = new Equipment();
        }
        this.equipment.setEquipmentId(Integer.parseInt(equipmentId));
    }

    public void setAuditDate(Date auditDate) {
        this.auditDate = auditDate;
    }

    @Override
    public String toString() {
        return "Auditing{" +
                "logId=" + logId +
                ", timestamp=" + timestamp +
                ", details='" + details + '\'' +
                ", action=" + action +
                ", employee=" + employee +
                '}';
    }
}
```

Auditing_1.java

```
package ecsapplication.Model;

import java.time.LocalDateTime;

public class Auditing_1 {
    private int auditId;
    private LocalDateTime timestamp;
    private String details;
    private Action action;
    private Employee employee;
    private Integer equipmentId;  // Can be null
```

```
// Constructors
public Auditing_1() {}

public Auditing_1(int auditId, LocalDateTime timestamp, String details,
            Action action, Employee employee, Integer equipmentId) {
    this.auditId = auditId;
    this.timestamp = timestamp;
    this.details = details;
    this.action = action;
    this.employee = employee;
    this.equipmentId = equipmentId;
}

// Getters and Setters
public int getAuditId() {
    return auditId;
}

public void setAuditId(int auditId) {
    this.auditId = auditId;
}

public LocalDateTime getAuditDate() {
    return timestamp;
}

public void setTimestamp(LocalDateTime timestamp) {
```

```
    this.timestamp = timestamp;
}

public String getDetails() {
    return details;
}

public void setDetails(String details) {
    this.details = details;
}

public Action getAction() {
    return action;
}

public void setAction(Action action) {
    this.action = action;
}

public Employee getEmployee() {
    return employee;
}

public void setEmployee(Employee employee) {
    this.employee = employee;
}
```

```
    public Integer getEquipmentId() {
        return equipmentId;
    }

    public void setEquipmentId(Integer equipmentId) {
        this.equipmentId = equipmentId;
    }

    @Override
    public String toString() {
        return "Auditing{" +
                "auditId=" + auditId +
                ", timestamp=" + timestamp +
                ", details='" + details + '\'' +
                ", action=" + action +
                ", employee=" + employee +
                ", equipmentId=" + equipmentId +
                '}';
    }
}
```

DatabaseConnection.java

```
package ecsapplication.Model;

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

public class DatabaseConnection {
    private static final String URL =
"jdbc:mysql://localhost:3306/ems_database";
    private static final String USER = "root";
    private static final String PASSWORD = "";

    public static Connection getConnection() throws SQLException {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            return DriverManager.getConnection(URL, USER, PASSWORD);
```

```
        } catch (ClassNotFoundException e) {
            throw new SQLException("MySQL JDBC Driver not found", e);
        }
    }
}
```

ECSApplication.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this template
 */
package ecsapplication.Model;
// Add this import
import ecsapplication.DataAccess.DatabaseInitializer;
import ecsapplication.Setup.MySQLInstaller;
import ecsapplication.UserInterface.MainWindow;
import javax.swing.SwingUtilities;
import javax.swing.UIManager;
import javax.swing.JOptionPane;

/**
```

```
 * Main application class for the Equipment Checkout System
 * @author Ughon
 */
public class ECSApplication {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // Check and install MySQL if needed
        if (!MySQLInstaller.checkAndInstallMySQL()) {
            JOptionPane.showMessageDialog(null,
                "MySQL installation is required to run this application.",
                "Installation Required",
                JOptionPane.ERROR_MESSAGE);
            System.exit(1);
        }

        // Initialize the database before launching the GUI
        try {
            DatabaseInitializer.initializeDatabase();
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null,
                "Failed to initialize database: " + e.getMessage(),
                "Database Error",
                JOptionPane.ERROR_MESSAGE);
            System.exit(1);
```

```
        }

        // Launch the GUI on the Event Dispatch Thread
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                try {
                    // Set system look and feel
UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
                } catch (Exception e) {
                    e.printStackTrace();
                }

                // Create and show the main window
                new MainWindow().setVisible(true);
            }
        });
    }
}
```

Employee.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package ecsapplication.Model;
import java.time.LocalDate;

public class Employee {
    private int employeeId;
    private String firstName;
    private String lastName;
    private LocalDate hireDate;
    private EmployeeRole role;
    private Skill skill;
    private String email;
```

```
    // Constructors
    public Employee() {}

    public Employee(int employeeId, String firstName, String lastName,
LocalDate hireDate,
                    EmployeeRole role, Skill skill, String email) {
        this.employeeId = employeeId;
        this.firstName = firstName;
        this.lastName = lastName;
        this.hireDate = hireDate;
        this.role = role;
        this.skill = skill;
        this.email = email;
    }

    // Getters and Setters
    public int getEmployeeId() {
        return employeeId;
    }

    public void setEmployeeId(int employeeId) {
        this.employeeId = employeeId;
    }

    public String getFirstName() {
        return firstName;
```

```
}

public void setFirstName(String firstName) {
    this.firstName = firstName;
}

public String getLastName() {
    return lastName;
}

public void setLastName(String lastName) {
    this.lastName = lastName;
}

public LocalDate getHireDate() {
    return hireDate;
}

public void setHireDate(LocalDate hireDate) {
    this.hireDate = hireDate;
}

public EmployeeRole getRole() {
    return role;
}

public void setRole(EmployeeRole role) {
```

```
    this.role = role;
}

public String getRoleName() {
    return role != null ? role.getRoleName() : "";
}

public Skill getSkill() {
    return skill;
}

public void setSkill(Skill skill) {
    this.skill = skill;
}

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}

public String getFullName() {
    return firstName + " " + lastName;
}
```

```
    // Additional getters
    public String getSkills() {
        return skill != null ? skill.getSkillName() : "";
    }

    @Override
    public String toString() {
        return "Employee{" +
                "employeeId=" + employeeId +
                ", firstName='" + firstName + '\'' +
                ", lastName='" + lastName + '\'' +
                ", hireDate=" + hireDate +
                ", role=" + role +
                ", skill=" + skill +
                ", email='" + email + '\'' +
                '}';
    }
}
```

EmployeeRole.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package ecsapplication.Model;


public class EmployeeRole {
    private int roleId;
    private String roleName;

    // Constructors
    public EmployeeRole() {}

    public EmployeeRole(int roleId, String roleName) {
        this.roleId = roleId;
        this.roleName = roleName;
    }
```

```
    // Getters and Setters
    public int getRoleId() {
        return roleId;
    }

    public void setRoleId(int roleId) {
        this.roleId = roleId;
    }

    public String getRoleName() {
        return roleName;
    }

    public void setRoleName(String roleName) {
        this.roleName = roleName;
    }

    @Override
    public String toString() {
        return "EmployeeRole{" +
                "roleId=" + roleId +
                ", roleName='" + roleName + '\'' +
                '}';
    }
}
```

Equipment.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package ecsapplication.Model;
import java.math.BigDecimal;

public class Equipment {
    private int equipmentId;
    private int equipmentCode;
    private String equipmentType;
    private BigDecimal value;
    private Skill skill;
    private EquipmentAvailability availability;
    private EquipmentStatus status;
    private Employee employee;
    private Location location;

    // Constructors
```

```
    public Equipment() {}


    public Equipment(int equipmentId, int equipmentCode, String
equipmentType, BigDecimal value,
                    Skill skill, EquipmentAvailability availability, EquipmentStatus
status,
                    Employee employee, Location location) {
        this.equipmentId = equipmentId;
        this.equipmentCode = equipmentCode;
        this.equipmentType = equipmentType;
        this.value = value;
        this.skill = skill;
        this.availability = availability;
        this.status = status;
        this.employee = employee;
        this.location = location;
    }

    // Getters and Setters
    public int getEquipmentId() {
        return equipmentId;
    }

    public void setEquipmentId(int equipmentId) {
        this.equipmentId = equipmentId;
    }
```

```
public int getEquipmentCode() {
    return equipmentCode;
}

public void setEquipmentCode(int equipmentCode) {
    this.equipmentCode = equipmentCode;
}

public String getEquipmentType() {
    return equipmentType;
}

public void setEquipmentType(String equipmentType) {
    this.equipmentType = equipmentType;
}

public BigDecimal getValue() {
    return value;
}

public void setValue(BigDecimal value) {
    this.value = value;
}

public Skill getSkill() {
    return skill;
}
```

```
public void setSkill(Skill skill) {
    this.skill = skill;
}

public EquipmentAvailability getAvailability() {
    return availability;
}

public void setAvailability(EquipmentAvailability availability) {
    this.availability = availability;
}

public EquipmentStatus getStatus() {
    return status;
}

public void setStatus(EquipmentStatus status) {
    this.status = status;
}

public Employee getEmployee() {
    return employee;
}

public void setEmployee(Employee employee) {
    this.employee = employee;
```

```
    }

    public Location getLocation() {
        return location;
    }

    public void setLocation(Location location) {
        this.location = location;
    }

    @Override
    public String toString() {
        return "Equipment{" +
                "equipmentId=" + equipmentId +
                ", equipmentCode=" + equipmentCode +
                ", equipmentType='" + equipmentType + '\'' +
                ", value=" + value +
                ", skill=" + skill +
                ", availability=" + availability +
                ", status=" + status +
                ", employee=" + employee +
                ", location=" + location +
                '}';
    }
}
```

EquipmentAvailability.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package ecsapplication.Model;


public class EquipmentAvailability {
    private int availabilityId;
    private String availabilityName;

    // Constructors
    public EquipmentAvailability() {}

    public EquipmentAvailability(int availabilityId, String availabilityName) {
        this.availabilityId = availabilityId;
        this.availabilityName = availabilityName;
    }

    // Getters and Setters
    public int getAvailabilityId() {
        return availabilityId;
    }
```

```
    public void setAvailabilityId(int availabilityId) {
        this.availabilityId = availabilityId;
    }

    public String getAvailabilityName() {
        return availabilityName;
    }

    public void setAvailabilityName(String availabilityName) {
        this.availabilityName = availabilityName;
    }

    @Override
    public String toString() {
        return "EquipmentAvailability{" +
                "availabilityId=" + availabilityId +
                ", availabilityName='" + availabilityName + '\'' +
                '}';
    }
}
```

EquipmentManagementWindow.java

```
package ecsapplication.Model;


import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableRowSorter;
import javax.swing.RowFilter;
import java.awt.*;
import java.awt.event.*;
import java.sql.*;


public class EquipmentManagementWindow extends JFrame {
    private JTable equipmentTable;
    private DefaultTableModel tableModel;
    private JScrollPane scrollPane;
    private TableRowSorter<DefaultTableModel> sorter;
    private JTextField searchField;
    private static final String[] COLUMN_NAMES = {
        "Equipment ID", "Name", "Type", "Status", "Location", "Last
Maintenance", "Next Maintenance"
    };


    public EquipmentManagementWindow() {
        initComponents();
        // Load data after window is visible
        SwingUtilities.invokeLater(this::loadEquipmentData);
```

```
    }

    private void initComponents() {
        setTitle("Equipment Management");
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        setSize(800, 600);
        setLocationRelativeTo(null);

        // Initialize table model
        tableModel = new DefaultTableModel(COLUMN_NAMES, 0) {
            @Override
            public boolean isCellEditable(int row, int column) {
                return false; // Make table non-editable
            }
        };

        // Create and configure table
        equipmentTable = new JTable(tableModel);

equipmentTable.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);

equipmentTable.setAutoResizeMode(JTable.AUTO_RESIZE_ALL_COLUMNS);

        // Add sorting capability
        sorter = new TableRowSorter<>(tableModel);
        equipmentTable.setRowSorter(sorter);
```

```
// Add table to scroll pane
scrollPane = new JScrollPane(equipmentTable);

// Create search panel
JPanel searchPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
searchField = new JTextField(20);
JButton searchButton = new JButton("Search");

searchButton.addActionListener(e -> filterTable());
searchField.addKeyListener(new KeyAdapter() {
    @Override
    public void keyReleased(KeyEvent e) {
        filterTable();
    }
});

searchPanel.add(new JLabel("Search:"));
searchPanel.add(searchField);
searchPanel.add(searchButton);

// Add refresh button
JButton refreshButton = new JButton("Refresh");
refreshButton.addActionListener(e -> loadEquipmentData());
searchPanel.add(refreshButton);

// Add row selection listener
equipmentTable.getSelectionModel().addListSelectionListener(e -> {
```

```
            if (!e.getValueIsAdjusting()) {
                int selectedRow = equipmentTable.getSelectedRow();
                if (selectedRow >= 0) {
                    int modelRow =
equipmentTable.convertRowIndexToModel(selectedRow);
                    // You can add custom handling for row selection here
                }
            }
        });

        // Add components to frame
        getContentPane().setLayout(new BorderLayout());
        getContentPane().add(searchPanel, BorderLayout.NORTH);
        getContentPane().add(scrollPane, BorderLayout.CENTER);
    }

    private void filterTable() {
        String text = searchField.getText();
        if (text.trim().length() == 0) {
            sorter.setRowFilter(null);
        } else {
            sorter.setRowFilter(RowFilter.regexFilter("(?i)" + text));
        }
    }

    private void loadEquipmentData() {
        try {
```

```
        // Clear existing data
        tableModel.setRowCount(0);

        // Create and execute query
        try ( // Get database connection
              Connection conn = DatabaseConnection.getConnection())
        {
           // Create and execute query
           String query = "SELECT * FROM Equipment ORDER BY
equipment_id";
           PreparedStatement pstmt = conn.prepareStatement(query);
           ResultSet rs = pstmt.executeQuery();

           // Populate table with data
           while (rs.next()) {
              Object[] row = {
                 rs.getInt("equipment_id"),
                 rs.getString("name"),
                 rs.getString("type"),
                 rs.getString("status"),
                 rs.getString("location"),
                 rs.getDate("last_maintenance"),
                 rs.getDate("next_maintenance")
              };
              tableModel.addRow(row);
           }
```

```
            // Close resources
            rs.close();
            pstmt.close();
        }

    } catch (SQLException e) {
        JOptionPane.showMessageDialog(this,
            "Error loading equipment data: " + e.getMessage(),
            "Database Error",
            JOptionPane.ERROR_MESSAGE);
        e.printStackTrace();
    }
}

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

EquipmentStatus.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package ecsapplication.Model;

public class EquipmentStatus {
    private int statusId;
    private String statusName;

    // Constructors
    public EquipmentStatus() {}

    public EquipmentStatus(int statusId, String statusName) {
        this.statusId = statusId;
        this.statusName = statusName;
    }

    // Getters and Setters
    public int getStatusId() {
        return statusId;
    }

    public void setStatusId(int statusId) {
        this.statusId = statusId;
```

```
    }

    public String getStatusName() {
        return statusName;
    }

    public void setStatusName(String statusName) {
        this.statusName = statusName;
    }

    @Override
    public String toString() {
        return "EquipmentStatus{" +
                "statusId=" + statusId +
                ", statusName='" + statusName + '\'' +
                '}';
    }
}
```

Location.java

```
/*
```

```
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package ecsapplication.Model;

public class Location {
    private int locationId;
    private String locationName;

    // Constructors
    public Location() {}

    public Location(int locationId, String locationName) {
        this.locationId = locationId;
        this.locationName = locationName;
    }

    // Getters and Setters
    public int getLocationId() {
        return locationId;
    }

    public void setLocationId(int locationId) {
        this.locationId = locationId;
    }
```

```
    public String getLocationName() {
        return locationName;
    }

    public void setLocationName(String locationName) {
        this.locationName = locationName;
    }

    @Override
    public String toString() {
        return "Location{" +
                "locationId=" + locationId +
                ", locationName='" + locationName + '\'' +
                '}';
    }
}
```

NotificationLog.java

```
/*
```

```
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package ecsapplication.Model;
import java.time.LocalDateTime;

public class NotificationLog {
    private int notificationId;
    private String message;
    private LocalDateTime timestamp;
    private boolean isRead;
    private NotificationType type;
    private Employee employee;

    // Constructors
    public NotificationLog() {}

    public NotificationLog(int notificationId, String message, LocalDateTime timestamp,
                           boolean isRead, NotificationType type, Employee employee) {
        this.notificationId = notificationId;
        this.message = message;
        this.timestamp = timestamp;
        this.isRead = isRead;
        this.type = type;
        this.employee = employee;
```

```
}

// Getters and Setters
public int getNotificationId() {
    return notificationId;
}

public void setNotificationId(int notificationId) {
    this.notificationId = notificationId;
}

public String getMessage() {
    return message;
}

public void setMessage(String message) {
    this.message = message;
}

public LocalDateTime getTimestamp() {
    return timestamp;
}

public void setTimestamp(LocalDateTime timestamp) {
    this.timestamp = timestamp;
}
```

```
public boolean isRead() {
    return isRead;
}

public void setRead(boolean read) {
    isRead = read;
}

public NotificationType getType() {
    return type;
}

public void setType(NotificationType type) {
    this.type = type;
}

public Employee getEmployee() {
    return employee;
}

public void setEmployee(Employee employee) {
    this.employee = employee;
}

@Override
public String toString() {
    return "NotificationLog{" +
```

```
                "notificationId=" + notificationId +
                ", message='" + message + '\'' +
                ", timestamp=" + timestamp +
                ", isRead=" + isRead +
                ", type=" + type +
                ", employee=" + employee +
                '}';
    }
}
```

NotificationType.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-
default.txt to change this license
```

```
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package ecsapplication.Model;

public class NotificationType {
    private int typeId;
    private String typeName;

    // Constructors
    public NotificationType() {}

    public NotificationType(int typeId, String typeName) {
        this.typeId = typeId;
        this.typeName = typeName;
    }

    // Getters and Setters
    public int getTypeId() {
        return typeId;
    }

    public void setTypeId(int typeId) {
        this.typeId = typeId;
    }

    public String getTypeName() {
```

```
        return typeName;
    }

    public void setTypeName(String typeName) {
        this.typeName = typeName;
    }

    @Override
    public String toString() {
        return "NotificationType{" +
                "typeId=" + typeId +
                ", typeName='" + typeName + '\'' +
                '}';
    }
}
```

Skill.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-
default.txt to change this license
```

```
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package ecsapplication.Model;

public class Skill {
    private int skillId;
    private String skillLevel;

    // Constructors
    public Skill() {}

    public Skill(int skillId, String skillLevel) {
        this.skillId = skillId;
        this.skillLevel = skillLevel;
    }

    // Getters and Setters
    public int getSkillId() {
        return skillId;
    }

    public void setSkillId(int skillId) {
        this.skillId = skillId;
    }

    public String getSkillLevel() {
```

```
        return skillLevel;
    }

    public void setSkillLevel(String skillLevel) {
        this.skillLevel = skillLevel;
    }

    public String getSkillName() {
        return skillLevel; // Using skillLevel as the skill name
    }

    @Override
    public String toString() {
        return "Skill{" +
                "skillId=" + skillId +
                ", skillLevel='" + skillLevel + '\'' +
                '}';
    }
}
```

Supervisor.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
```

```
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package ecsapplication.Model;

public class Supervisor {
    private int supervisorId;
    private Employee employee;

    // Constructors
    public Supervisor() {}

    public Supervisor(int supervisorId, Employee employee) {
        this.supervisorId = supervisorId;
        this.employee = employee;
    }

    // Getters and Setters
    public int getSupervisorId() {
        return supervisorId;
    }

    public void setSupervisorId(int supervisorId) {
        this.supervisorId = supervisorId;
    }

    public Employee getEmployee() {
```

```
        return employee;
    }

    public void setEmployee(Employee employee) {
        this.employee = employee;
    }

    @Override
    public String toString() {
        return "Supervisor{" +
                "supervisorId=" + supervisorId +
                ", employee=" + employee +
                '}';
    }
}
```

Transaction.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
```

```
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package ecsapplication.Model;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.Date;

public class Transaction {
    private int transactionId;
    private LocalDateTime transactionDate;
    private LocalDate dueDate;
    private LocalDate returnDate;
    private Employee employee;
    private Equipment equipment;

    // Constructors
    public Transaction() {}

    public Transaction(int transactionId, LocalDateTime transactionDate, LocalDate dueDate,
                       LocalDate returnDate, Employee employee, Equipment equipment) {
        this.transactionId = transactionId;
        this.transactionDate = transactionDate;
        this.dueDate = dueDate;
        this.returnDate = returnDate;
        this.employee = employee;
```

```
    this.equipment = equipment;
}

// Getters and Setters
public int getTransactionId() {
    return transactionId;
}

public void setTransactionId(int transactionId) {
    this.transactionId = transactionId;
}

public LocalDateTime getTransactionDate() {
    return transactionDate;
}

public void setTransactionDate(LocalDateTime transactionDate) {
    this.transactionDate = transactionDate;
}

public LocalDate getDueDate() {
    return dueDate;
}

public void setDueDate(LocalDate dueDate) {
    this.dueDate = dueDate;
}
```

```
public LocalDate getReturnDate() {
    return returnDate;
}

public void setReturnDate(LocalDate returnDate) {
    this.returnDate = returnDate;
}

public Employee getEmployee() {
    return employee;
}

public void setEmployee(Employee employee) {
    this.employee = employee;
}

public Equipment getEquipment() {
    return equipment;
}

public void setEquipment(Equipment equipment) {
    this.equipment = equipment;
}

public boolean isOverdue() {
    return returnDate == null && LocalDate.now().isAfter(dueDate);
```

```
    }

    public boolean isReturned() {
        return returnDate != null;
    }

    public String getEquipmentId() {
        return equipment != null ?
String.valueOf(equipment.getEquipmentId()) : null;
    }

    public void setCheckInDate(Date checkInDate) {
        this.returnDate =
checkInDate.toInstant().atZone(java.time.ZoneId.systemDefault()).toLocalDa
te();
    }

    public void setStatus(String status) {
        // Status is tracked through returnDate
        if ("Completed".equals(status)) {
            this.returnDate = LocalDate.now();
        }
    }

    @Override
    public String toString() {
        return "Transaction{" +
                "transactionId=" + transactionId +
```

```
            ", transactionDate=" + transactionDate +
            ", dueDate=" + dueDate +
            ", returnDate=" + returnDate +
            ", employee=" + employee +
            ", equipment=" + equipment +
            '}';
    }
}
```

MySQLInstaller.java

```
package ecsapplication.Setup;

import ecsapplication.DataAccess.DatabaseConnection;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileOutputStream;
```

```
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.URL;
import java.nio.channels.Channels;
import java.nio.channels.ReadableByteChannel;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.List;
import javax.swing.JOptionPane;
import javax.swing.SwingWorker;


public class MySQLInstaller {
    private static final String MYSQL_INSTALLER_URL =
"https://dev.mysql.com/get/Downloads/MySQL-8.0/mysql-installer-
community-8.0.35.0.msi";
    private static final String MYSQL_INSTALLER_PATH =
System.getProperty("java.io.tmpdir") + "mysql-installer.msi";
    private static final String MYSQL_SERVICE_NAME = "MySQL80";


    public static boolean checkAndInstallMySQL() {
        if (isMySQLInstalled()) {
            return true;
        }


        int response = JOptionPane.showConfirmDialog(null,
```

```
            "MySQL is not installed. Would you like to install it now?",
            "MySQL Installation Required",
            JOptionPane.YES_NO_OPTION);

        if (response != JOptionPane.YES_OPTION) {
            return false;
        }

        return installMySQL();
    }

    private static boolean isMySQLInstalled() {
        try {
            // Check if MySQL service exists
            Process process = Runtime.getRuntime().exec("sc query " +
MYSQL_SERVICE_NAME);
            BufferedReader reader = new BufferedReader(new
InputStreamReader(process.getInputStream()));
            String line;
            while ((line = reader.readLine()) != null) {
                if (line.contains("SERVICE_NAME: " + MYSQL_SERVICE_NAME)) {
                    return true;
                }
            }

            // Check if MySQL is in PATH
            Process pathCheck = Runtime.getRuntime().exec("where mysql");
```

```
            return pathCheck.waitFor() == 0;
        } catch (Exception e) {
            return false;
        }
    }

    private static boolean installMySQL() {
        try {
            // Download MySQL installer
            downloadMySQLInstaller();

            // Run the installer silently
            ProcessBuilder builder = new ProcessBuilder(
                "msiexec", "/i", MYSQL_INSTALLER_PATH,
                "/qn", "INSTALLDIR=C:\\Program Files\\MySQL\\MySQL Server 8.0",
                "SERVICENAME=" + MYSQL_SERVICE_NAME,
                "PASSWORD=" + DatabaseConnection.PASSWORD
            );

            Process process = builder.start();
            int exitCode = process.waitFor();

            if (exitCode != 0) {
                JOptionPane.showMessageDialog(null,
                    "Failed to install MySQL. Please install it manually.",
                    "Installation Error",
                    JOptionPane.ERROR_MESSAGE);
```

```
            return false;
        }

        // Configure MySQL
        configureMySQL();

        return true;
    } catch (Exception e) {
        JOptionPane.showMessageDialog(null,
            "Error during MySQL installation: " + e.getMessage(),
            "Installation Error",
            JOptionPane.ERROR_MESSAGE);
        return false;
    }
}

private static void downloadMySQLInstaller() throws IOException {
    URL website = new URL(MYSQL_INSTALLER_URL);
    ReadableByteChannel rbc =
Channels.newChannel(website.openStream());
    FileOutputStream fos = new
FileOutputStream(MYSQL_INSTALLER_PATH);
    fos.getChannel().transferFrom(rbc, 0, Long.MAX_VALUE);
    fos.close();
}

private static void configureMySQL() throws IOException,
InterruptedException {
```

```
        // Create my.ini file with custom configuration
        String myIniPath = "C:\\Program Files\\MySQL\\MySQL Server 8.0\\
my.ini";
        List<String> config = new ArrayList<>();
        config.add("[mysqld]");
        config.add("port=3306");
        config.add("basedir=\"C:\\Program Files\\MySQL\\MySQL Server 8.0\"");
        config.add("datadir=\"C:\\Program Files\\MySQL\\MySQL Server 8.0\\
data\"");
        config.add("default_authentication_plugin=mysql_native_password");
        config.add("default-storage-engine=INNODB");
        config.add("character-set-server=utf8mb4");
        config.add("collation-server=utf8mb4_0900_ai_ci");

        Files.write(Paths.get(myIniPath), config);

        // Restart MySQL service
        Runtime.getRuntime().exec("net stop " +
MYSQL_SERVICE_NAME).waitFor();
        Runtime.getRuntime().exec("net start " +
MYSQL_SERVICE_NAME).waitFor();

        // Set root password
        ProcessBuilder builder = new ProcessBuilder(
            "C:\\Program Files\\MySQL\\MySQL Server 8.0\\bin\\mysqladmin",
            "-u", "root",
            "password", DatabaseConnection.PASSWORD
        );
```

```
        builder.start().waitFor();
    }
}
```

AddEquipmentWindow.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/GUIForms/JFrame.java to edit this template
 */
```

```
package ecsapplication.UserInterface;
import javax.swing.JOptionPane;
import java.sql.*;
import ecsapplication.DataAccess.DatabaseConnection;

/**
 *
 * @author Redux
 */
public class AddEquipmentWindow extends javax.swing.JFrame
{
    private EquipmentManagementWindow parentWindow;

    /**
     * Creates new form AddEquipmentWindow
     */
    public AddEquipmentWindow()
    {
        this(null);
    }

    public AddEquipmentWindow(EquipmentManagementWindow parent)
    {
        initComponents();
        setLocationRelativeTo(null);
        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
        this.parentWindow = parent;
```

```
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents
    private void initComponents()
    {

        lblEquipmentType = new javax.swing.JLabel();
        lblQuantity = new javax.swing.JLabel();
        lblLocation = new javax.swing.JLabel();
        btnSave = new javax.swing.JButton();
        btnCancel = new javax.swing.JButton();
        cmbEquipmentType = new javax.swing.JComboBox<>();
        cmbQuantity = new javax.swing.JComboBox<>();
        cmbLocation = new javax.swing.JComboBox<>();


setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("Add New Equipment");
        setResizable(false);
```

```
lblEquipmentType.setText("Equipment Type:");

lblQuantity.setText("Quantity:");

lblLocation.setText("Location:");

btnSave.setBackground(new java.awt.Color(51, 102, 255));
btnSave.setForeground(new java.awt.Color(255, 255, 255));
btnSave.setText("Save");
btnSave.addActionListener(new java.awt.event.ActionListener()
{
    public void actionPerformed(java.awt.event.ActionEvent evt)
    {
        btnSaveActionPerformed(evt);
    }
});

btnCancel.setBackground(new java.awt.Color(204, 204, 204));
btnCancel.setText("Cancel");
btnCancel.addActionListener(new java.awt.event.ActionListener()
{
    public void actionPerformed(java.awt.event.ActionEvent evt)
    {
        btnCancelActionPerformed(evt);
    }
});
```

```
        cmbEquipmentType.setModel(new
javax.swing.DefaultComboBoxModel<>(new String[] { "Select Equipment
Type...", "Hammer", "Cordless Drill", "Circular Saw", "Ladder (6ft)", "Wrench
Set", "Power Washer", "Forklift", "Pallet Jack", "Pressure Gauge", "Voltage
Tester" }));


        cmbQuantity.setModel(new javax.swing.DefaultComboBoxModel<>(new
String[] { "Select Quantity...", "1", "2", "3", "4", "5" }));


        cmbLocation.setModel(new javax.swing.DefaultComboBoxModel<>(new
String[] { "Select Location...", "North Warehouse", "South Warehouse" }));


        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Ali
gnment.TRAILING)
                    .addGroup(layout.createSequentialGroup()
                        .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                        .addComponent(btnCancel,
javax.swing.GroupLayout.PREFERRED_SIZE, 100,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addGap(20, 20, 20)
                        .addComponent(btnSave,
javax.swing.GroupLayout.PREFERRED_SIZE, 100,
javax.swing.GroupLayout.PREFERRED_SIZE))
```

```
                .addGroup(layout.createSequentialGroup()
                    .addGap(29, 29, 29)
                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                        .addComponent(lblEquipmentType)
                        .addComponent(lblQuantity)
                        .addComponent(lblLocation))
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 59, Short.MAX_VALUE)
                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
                        .addComponent(cmbEquipmentType, 0, 300, Short.MAX_VALUE)
                        .addComponent(cmbQuantity, 0, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addComponent(cmbLocation, 0, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))))
                .addGap(23, 23, 23))
        );
        layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
                .addContainerGap(41, Short.MAX_VALUE)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(lblEquipmentType)
                    .addComponent(cmbEquipmentType, javax.swing.GroupLayout.PREFERRED_SIZE,
```

```
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

            .addGap(30, 30, 30)

            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Ali
gnment.BASELINE)

                .addComponent(lblQuantity)

                .addComponent(cmbQuantity,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

            .addGap(27, 27, 27)

            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Ali
gnment.BASELINE)

                .addComponent(lblLocation)

                .addComponent(cmbLocation,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

            .addGap(80, 80, 80)

            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Ali
gnment.LEADING)

                .addComponent(btnCancel,
javax.swing.GroupLayout.PREFERRED_SIZE, 30,
javax.swing.GroupLayout.PREFERRED_SIZE)

                .addComponent(btnSave,
javax.swing.GroupLayout.PREFERRED_SIZE, 30,
javax.swing.GroupLayout.PREFERRED_SIZE))

            .addGap(36, 36, 36))

        );


        pack();
```

```
    }// </editor-fold>//GEN-END:initComponents


    private void btnSaveActionPerformed(java.awt.event.ActionEvent
evt)//GEN-FIRST:event_btnSaveActionPerformed
    {//GEN-HEADEREND:event_btnSaveActionPerformed
        // Get selected items
        int equipmentIndex = cmbEquipmentType.getSelectedIndex();
        int quantityIndex = cmbQuantity.getSelectedIndex();
        int locationIndex = cmbLocation.getSelectedIndex();

        // Validate that none are still at the default index
        if (equipmentIndex == 0 || quantityIndex == 0 || locationIndex == 0) {
            JOptionPane.showMessageDialog(this,
                "Please make a selection for all fields.",
                "Validation Error",
                JOptionPane.WARNING_MESSAGE);
            return;
        }

        // Get the selected values
        String equipmentType =
cmbEquipmentType.getSelectedItem().toString();
        int quantity =
Integer.parseInt(cmbQuantity.getSelectedItem().toString());
        String location = cmbLocation.getSelectedItem().toString();

        try (Connection conn = DatabaseConnection.getConnection()) {
            // Start transaction
```

```
            conn.setAutoCommit(false);


            try {
                // First get the location_id from the location table
                int locationId;
                try (PreparedStatement locStmt = conn.prepareStatement(
                        "SELECT location_id FROM location WHERE location_name
= ?")) {
                    locStmt.setString(1, location);
                    ResultSet rs = locStmt.executeQuery();
                    if (!rs.next()) {
                        throw new SQLException("Location not found: " + location);
                    }
                    locationId = rs.getInt("location_id");
                }


                // Get the default status_id for 'Available'
                int statusId;
                try (PreparedStatement statStmt = conn.prepareStatement(
                        "SELECT status_id FROM equipmentstatus WHERE
UPPER(status_name) = 'AVAILABLE' OR status_name = 'Available'")) {
                    ResultSet rs = statStmt.executeQuery();
                    if (!rs.next()) {
                        // If no 'Available' status found, get the first status_id from the
table
                        try (PreparedStatement defaultStatStmt =
conn.prepareStatement(
```

```
                    "SELECT status_id, status_name FROM equipmentstatus
LIMIT 1")) {
                ResultSet defaultRs = defaultStatStmt.executeQuery();
                if (!defaultRs.next()) {
                    throw new SQLException("No equipment status found in
the database");
                }
                statusId = defaultRs.getInt("status_id");
                System.out.println("Using default status: " +
defaultRs.getString("status_name"));
            }
        } else {
            statusId = rs.getInt("status_id");
        }
    }

    // Insert equipment records based on quantity
    String insertSql = "INSERT INTO equipment (equipment_type,
equipment_code, skill_id, status_id, location_id, availability_id, value) " +
                    "SELECT ?, ?, 1, ?, ?, 1, etv.value " +
                    "FROM equipment_type_value etv " +
                    "WHERE etv.equipment_type = ?";
    try (PreparedStatement pstmt =
conn.prepareStatement(insertSql)) {
        for (int i = 0; i < quantity; i++) {
            // Debug print
            System.out.println("Attempting to insert: " +
                "Type=" + equipmentType + ", " +
```

```
                    "Code=" + generateEquipmentCode() + ", " +
                    "Status=" + statusId + ", " +
                    "Location=" + locationId);

                pstmt.setString(1, equipmentType);
                pstmt.setInt(2, generateEquipmentCode());
                pstmt.setInt(3, statusId);
                pstmt.setInt(4, locationId);
                pstmt.setString(5, equipmentType);  // For the WHERE clause to get the correct value
                pstmt.executeUpdate();
            }
        }

        // Commit transaction
        conn.commit();

        // Show success message
        JOptionPane.showMessageDialog(this,
            "Equipment added successfully!",
            "Success",
            JOptionPane.INFORMATION_MESSAGE);

        // Reset form
        cmbEquipmentType.setSelectedIndex(0);
        cmbQuantity.setSelectedIndex(0);
        cmbLocation.setSelectedIndex(0);
```

```
            // Refresh parent window if available
            if (parentWindow != null) {
                parentWindow.loadEquipmentData();
            }

            // Close the window
            this.dispose();

        } catch (SQLException ex) {
            // Rollback transaction on error
            conn.rollback();
            throw ex;
        }
    } catch (SQLException ex) {
        JOptionPane.showMessageDialog(this,
            "Error adding equipment: " + ex.getMessage(),
            "Database Error",
            JOptionPane.ERROR_MESSAGE);
        ex.printStackTrace();
    }
}//GEN-LAST:event_btnSaveActionPerformed

private int generateEquipmentCode() {
    // Generate a random 4-digit number between 1000 and 9999
    return 1000 + (int)(Math.random() * 9000);
}
```

```
    private void btnCancelActionPerformed(java.awt.event.ActionEvent
evt)//GEN-FIRST:event_btnCancelActionPerformed

    {//GEN-HEADEREND:event_btnCancelActionPerformed

        this.dispose(); // closes the AddUserWindow

    }//GEN-LAST:event_btnCancelActionPerformed


    /**

     * @param args the command line arguments

     */

    public static void main(String args[])

    {

        /* Set the Nimbus look and feel */

        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting
code (optional) ">

        /* If Nimbus (introduced in Java SE 6) is not available, stay with the
default look and feel.

         * For details see
http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html

         */

        try

        {

            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels())

            {

                if ("Nimbus".equals(info.getName()))

                {

                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
```

```
            break;
        }
    }
} catch (ClassNotFoundException ex)
{
java.util.logging.Logger.getLogger(AddEquipmentWindow.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
} catch (InstantiationException ex)
{
java.util.logging.Logger.getLogger(AddEquipmentWindow.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
} catch (IllegalAccessException ex)
{
java.util.logging.Logger.getLogger(AddEquipmentWindow.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
} catch (javax.swing.UnsupportedLookAndFeelException ex)
{
java.util.logging.Logger.getLogger(AddEquipmentWindow.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
}
//</editor-fold>

/* Create and display the form */
java.awt.EventQueue.invokeLater(new Runnable()
{
```

```
            public void run()
            {
                new AddEquipmentWindow().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JButton btnCancel;
    private javax.swing.JButton btnSave;
    private javax.swing.JComboBox<String> cmbEquipmentType;
    private javax.swing.JComboBox<String> cmbLocation;
    private javax.swing.JComboBox<String> cmbQuantity;
    private javax.swing.JLabel lblEquipmentType;
    private javax.swing.JLabel lblLocation;
    private javax.swing.JLabel lblQuantity;
    // End of variables declaration//GEN-END:variables
}
```

AddUserWindow.java

```
/*
```

```
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/GUIForms/JFrame.java to edit this template
 */
package ecsapplication.UserInterface;

import ecsapplication.DataAccess.DBManager;
import ecsapplication.DataAccess.EmployeeService;
import ecsapplication.DataAccess.EmployeeRoleDAO;
import ecsapplication.DataAccess.SkillDAO;
import ecsapplication.Model.Employee;
import ecsapplication.Model.EmployeeRole;
import ecsapplication.Model.Skill;
import java.time.LocalDate;
import javax.swing.JOptionPane;
import java.util.List;

/**
 *
 * @author Redux
 */
public class AddUserWindow extends javax.swing.JFrame
{
    private EmployeeService employeeService;
    private Employee loggedInEmployee; // Store the logged-in user
    private EmployeeRoleDAO roleDAO;
```

```
private SkillDAO skillDAO;

/**
 * Creates new form AddUserWindow
 */
public AddUserWindow()
{
    initComponents();
    setLocationRelativeTo(null); // center the window
    setDefaultCloseOperation(DISPOSE_ON_CLOSE);

    // Initialize services and DAOs
    employeeService = new EmployeeService();
    roleDAO = DBManager.getInstance().getDAO(EmployeeRoleDAO.class);
    skillDAO = DBManager.getInstance().getDAO(SkillDAO.class);
}

/**
 * Creates new form with logged in employee
 */
public AddUserWindow(Employee loggedInEmployee) {
    this();
    this.loggedInEmployee = loggedInEmployee;
}

/**
 * This method is called from within the constructor to initialize the form.
```

```
     * WARNING: Do NOT modify this code. The content of this method is
always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-
BEGIN:initComponents
    private void initComponents()
    {

        jPanel1 = new javax.swing.JPanel();
        txtFirstName = new javax.swing.JTextField();
        txtLastName = new javax.swing.JTextField();
        lblSkillsLevel = new javax.swing.JLabel();
        lblFirstName = new javax.swing.JLabel();
        lblLastName = new javax.swing.JLabel();
        lblRole = new javax.swing.JLabel();
        btnSave = new javax.swing.JButton();
        btnCancel = new javax.swing.JButton();
        cmbRole = new javax.swing.JComboBox<>();
        cmbSkillLevel = new javax.swing.JComboBox<>();


setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("Add New User");
        setResizable(false);


        jPanel1.setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());
```

```
        jPanel1.add(txtFirstName, new
org.netbeans.lib.awtextra.AbsoluteConstraints(160, 50, 300, -1));
        jPanel1.add(txtLastName, new
org.netbeans.lib.awtextra.AbsoluteConstraints(160, 100, 300, -1));


        lblSkillsLevel.setText("Skill Level:");
        jPanel1.add(lblSkillsLevel, new
org.netbeans.lib.awtextra.AbsoluteConstraints(30, 200, -1, -1));


        lblFirstName.setText("First Name:");
        jPanel1.add(lblFirstName, new
org.netbeans.lib.awtextra.AbsoluteConstraints(30, 50, -1, -1));


        lblLastName.setText("Last Name:");
        jPanel1.add(lblLastName, new
org.netbeans.lib.awtextra.AbsoluteConstraints(30, 100, -1, -1));


        lblRole.setText("Role:");
        jPanel1.add(lblRole, new
org.netbeans.lib.awtextra.AbsoluteConstraints(30, 150, -1, -1));


        btnSave.setBackground(new java.awt.Color(51, 102, 255));
        btnSave.setForeground(new java.awt.Color(255, 255, 255));
        btnSave.setText("Save");
        btnSave.addActionListener(new java.awt.event.ActionListener()
        {
            public void actionPerformed(java.awt.event.ActionEvent evt)
            {
                btnSaveActionPerformed(evt);
```

```
        }
    });
    jPanel1.add(btnSave, new
org.netbeans.lib.awtextra.AbsoluteConstraints(360, 250, 100, 30));


    btnCancel.setBackground(new java.awt.Color(204, 204, 204));
    btnCancel.setText("Cancel");
    btnCancel.addActionListener(new java.awt.event.ActionListener()
    {
        public void actionPerformed(java.awt.event.ActionEvent evt)
        {
            btnCancelActionPerformed(evt);
        }
    });
    jPanel1.add(btnCancel, new
org.netbeans.lib.awtextra.AbsoluteConstraints(240, 250, 100, 30));


    cmbRole.setModel(new javax.swing.DefaultComboBoxModel<>(new
String[] { "Select A Role...", "Supervisor", "DepotStaff", "Maintenance" }));
    jPanel1.add(cmbRole, new
org.netbeans.lib.awtextra.AbsoluteConstraints(160, 150, 300, -1));


    cmbSkillLevel.setModel(new
javax.swing.DefaultComboBoxModel<>(new String[] { "Select A Skill
Level...", "Novice", "Intermediate", "Advanced" }));
    jPanel1.add(cmbSkillLevel, new
org.netbeans.lib.awtextra.AbsoluteConstraints(160, 200, 300, -1));
```

```
        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE,
500, Short.MAX_VALUE)
        );
        layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE,
310, Short.MAX_VALUE)
        );


        pack();
    }// </editor-fold>//GEN-END:initComponents


    private void btnSaveActionPerformed(java.awt.event.ActionEvent
evt)//GEN-FIRST:event_btnSaveActionPerformed
    {//GEN-HEADEREND:event_btnSaveActionPerformed
        // Get values from fields
        String firstName = txtFirstName.getText().trim();
        String lastName = txtLastName.getText().trim();
        int roleIndex = cmbRole.getSelectedIndex();
        int skillLevelIndex = cmbSkillLevel.getSelectedIndex();


        // Validate fields
```

```
        if (firstName.isEmpty() || lastName.isEmpty() || roleIndex == 0 ||
skillLevelIndex == 0) {
            JOptionPane.showMessageDialog(this,
                "Please fill out all fields.",
                "Validation Error",
                JOptionPane.WARNING_MESSAGE);
            return;
        }


        try {
            // Get selected role and skill
            String roleName = cmbRole.getSelectedItem().toString();
            String skillLevel = cmbSkillLevel.getSelectedItem().toString();


            System.out.println("Selected skill level from dropdown: '" + skillLevel
+ "'"); // Debug log
            System.out.println("Available skill levels in database:"); // Debug log
            List<Skill> allSkills = skillDAO.getAllSkills();
            for (Skill s : allSkills) {
                System.out.println("- '" + s.getSkillLevel() + "' (ID: " + s.getSkillId()
+ ")");
            }


            System.out.println("Attempting to find role: " + roleName); // Debug
log
            // Get role ID based on role name
            EmployeeRole role = roleDAO.getRoleByName(roleName);
            if (role == null) {
```

```
                JOptionPane.showMessageDialog(this,
                    "Error: Role '" + roleName + "' not found in database. Please
contact system administrator.",
                    "Database Error",
                    JOptionPane.ERROR_MESSAGE);
                return;
            }
            System.out.println("Found role with ID: " + role.getRoleId()); // Debug
log


            System.out.println("Attempting to find skill level: " + skillLevel); //
Debug log
            // Get skill ID based on skill level
            Skill skill = skillDAO.getSkillByLevel(skillLevel);
            if (skill == null) {
                JOptionPane.showMessageDialog(this,
                    "Error: Skill level '" + skillLevel + "' not found in database.
Please contact system administrator.",
                    "Database Error",
                    JOptionPane.ERROR_MESSAGE);
                return;
            }
            System.out.println("Found skill with ID: " + skill.getSkillId()); // Debug
log


            // Create new employee
            Employee newEmployee = new Employee();
            newEmployee.setFirstName(firstName);
```

```
        newEmployee.setLastName(lastName);
        newEmployee.setHireDate(LocalDate.now());
        newEmployee.setEmail(firstName.toLowerCase() + "." +
lastName.toLowerCase() + "@gbmanufacturing.co");

        // Set role and skill using the retrieved objects
        newEmployee.setRole(role);
        newEmployee.setSkill(skill);

        // Use logged-in employee for audit logging, or default to supervisor if
not available
        Employee actingEmployee = loggedInEmployee != null ?
loggedInEmployee :
            employeeService.getEmployeeById(1); // Default to ID 1
(Supervisor)

        if (actingEmployee == null) {
            JOptionPane.showMessageDialog(this,
                "Error: Could not determine acting employee for audit logging.
Please log in again.",
                "Authentication Error",
                JOptionPane.ERROR_MESSAGE);
            return;
        }

        System.out.println("Attempting to create employee with Role ID: " +
role.getRoleId() +
                           ", Skill ID: " + skill.getSkillId()); // Debug log
```

```
            // Save to database using service
            boolean success = employeeService.createEmployee(newEmployee,
actingEmployee);

            if (success) {
                JOptionPane.showMessageDialog(this,
                    "Employee created successfully!\nEmployee ID: " +
newEmployee.getEmployeeId() + "\n" +
                    "Name: " + firstName + " " + lastName + "\n" +
                    "Role: " + role.getRoleName() + "\n" +
                    "Skill Level: " + skill.getSkillLevel(),
                    "Success",
                    JOptionPane.INFORMATION_MESSAGE);

                // Reset form
                txtFirstName.setText("");
                txtLastName.setText("");
                cmbRole.setSelectedIndex(0);
                cmbSkillLevel.setSelectedIndex(0);

                // Close window
                this.dispose();
            } else {
                JOptionPane.showMessageDialog(this,
                    "Failed to create employee. Database error occurred.\n" +
                    "Please check the console for error details and contact system
administrator.",
                    "Database Error",
```

```
                JOptionPane.ERROR_MESSAGE);
            }
        } catch (Exception e) {
            JOptionPane.showMessageDialog(this,
                "Error creating employee:\n" + e.getMessage() + "\n\n" +
                "Please check the console for error details and contact system administrator.",
                "System Error",
                JOptionPane.ERROR_MESSAGE);
            e.printStackTrace();
        }
    }//GEN-LAST:event_btnSaveActionPerformed


    private void btnCancelActionPerformed(java.awt.event.ActionEvent evt)//GEN-FIRST:event_btnCancelActionPerformed
    {//GEN-HEADEREND:event_btnCancelActionPerformed
        this.dispose(); // closes the AddUserWindow
    }//GEN-LAST:event_btnCancelActionPerformed


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

```
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the
default look and feel.
         * For details see
http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try
        {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels())
            {
                if ("Nimbus".equals(info.getName()))
                {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex)
        {

java.util.logging.Logger.getLogger(AddUserWindow.class.getName()).log(java
.util.logging.Level.SEVERE, null, ex);
        } catch (InstantiationException ex)
        {

java.util.logging.Logger.getLogger(AddUserWindow.class.getName()).log(java
.util.logging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex)
        {
```

```
java.util.logging.Logger.getLogger(AddUserWindow.class.getName()).log(java
.util.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex)
        {

java.util.logging.Logger.getLogger(AddUserWindow.class.getName()).log(java
.util.logging.Level.SEVERE, null, ex);
        }
        //</editor-fold>


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


    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JButton btnCancel;
    private javax.swing.JButton btnSave;
    private javax.swing.JComboBox<String> cmbRole;
    private javax.swing.JComboBox<String> cmbSkillLevel;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JLabel lblFirstName;
```

```
    private javax.swing.JLabel lblLastName;
    private javax.swing.JLabel lblRole;
    private javax.swing.JLabel lblSkillsLevel;
    private javax.swing.JTextField txtFirstName;
    private javax.swing.JTextField txtLastName;
    // End of variables declaration//GEN-END:variables
}
```

CheckInWindow.java

```
/*
```

```
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/GUIForms/JFrame.java to edit this template
 */
package ecsapplication.UserInterface;

import ecsapplication.DataAccess.DBManager;
import ecsapplication.DataAccess.EmployeeDAO;
import ecsapplication.DataAccess.EquipmentDAO;
import ecsapplication.DataAccess.TransactionDAO;
import ecsapplication.Model.Employee;
import ecsapplication.Model.Equipment;
import ecsapplication.Model.Transaction;
import java.sql.SQLException;
import java.util.List;
import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;

/**
 *
 * @author Ughon
 */
public class CheckInWindow extends javax.swing.JFrame
{
    private EmployeeDAO employeeDAO;
    private EquipmentDAO equipmentDAO;
```

```
    private TransactionDAO transactionDAO;
    private Employee currentEmployee;
    private List<Transaction> currentTransactions;
    private DefaultTableModel equipmentTableModel;
    private DefaultTableModel employeeTableModel;

    /**
     * Creates new form NewJFrame
     */
    public CheckInWindow()
    {
        initComponents();

        // Initialize DAOs
        employeeDAO =
DBManager.getInstance().getDAO(EmployeeDAO.class);
        equipmentDAO =
DBManager.getInstance().getDAO(EquipmentDAO.class);
        transactionDAO =
DBManager.getInstance().getDAO(TransactionDAO.class);

        // Set up equipment table model
        equipmentTableModel = (DefaultTableModel)
tblChekedoutEquip.getModel();
        equipmentTableModel.setRowCount(0); // Clear any existing data

        // Set up employee table model
        employeeTableModel = (DefaultTableModel) tblEmployees.getModel();
```

```
employeeTableModel.setRowCount(0); // Clear any existing data

// Load all employees
loadAllEmployees();

// Add action listener to verify button
btnCheckInVerifyEmployee.addActionListener(e -> verifyEmployee());

// Add action listener to table selection
tblChekedoutEquip.getSelectionModel().addListSelectionListener(e -> {
    if (!e.getValueIsAdjusting()) {
        handleEquipmentSelection();
    }
});

// Add action listener to employee table selection
tblEmployees.getSelectionModel().addListSelectionListener(e -> {
    if (!e.getValueIsAdjusting()) {
        handleEmployeeSelection();
    }
});

// Add action listener to submit button
btnCheckInSubmit.addActionListener(e -> submitCheckIn());

// Add action listener to clear button
btnCheckInClear.addActionListener(e -> clearForm());
```

```
    }


    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents
    private void initComponents()
    {


        jPnlCheckIn = new javax.swing.JPanel();
        lblCheckInEquipConditionComment = new javax.swing.JLabel();
        txtCheckInEmpId = new javax.swing.JTextField();
        btnCheckInVerifyEmployee = new java.awt.Button();
        btnCheckInSubmit = new javax.swing.JButton();
        btnCheckInClear = new javax.swing.JButton();
        lblCheckInEquipConditionVerification = new javax.swing.JLabel();
        rdoCheckInYes = new javax.swing.JRadioButton();
        rdoCheckInEquipConditionVerificationN = new javax.swing.JRadioButton();
        lblCheckInEmpId = new javax.swing.JLabel();
        lblCheckInTitle = new javax.swing.JLabel();
        pnlCheckInEmpInfo = new javax.swing.JPanel();
        lblCheckOutEmpName = new javax.swing.JLabel();
```

```
        lblCheckOutEmpDepartment = new javax.swing.JLabel();
        lblCheckOutEmpSkills = new javax.swing.JLabel();
        lblCheckOutEmpRole = new javax.swing.JLabel();
        txtCheckOutEmpName = new javax.swing.JTextField();
        txtCheckOutRole = new javax.swing.JTextField();
        txtCheckOutEmpDepartment1 = new javax.swing.JTextField();
        txtCheckOutSkill2 = new javax.swing.JTextField();
        scrlCheckedInEquipPane = new javax.swing.JScrollPane();
        tblChekedoutEquip = new javax.swing.JTable();
        lblCheckInEquipTable = new javax.swing.JLabel();
        cmbCheckInEquipType = new javax.swing.JComboBox<>();
        tblEmployees = new javax.swing.JTable();


setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);


        jPnlCheckIn.setBorder(javax.swing.BorderFactory.createLineBorder(new
java.awt.Color(0, 0, 0)));
        jPnlCheckIn.setPreferredSize(new java.awt.Dimension(400, 650));
        jPnlCheckIn.setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());


        lblCheckInEquipConditionComment.setText("Condition:");
        jPnlCheckIn.add(lblCheckInEquipConditionComment, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 520, 89, -1));
        jPnlCheckIn.add(txtCheckInEmpId, new
org.netbeans.lib.awtextra.AbsoluteConstraints(10, 84, 158, -1));


        btnCheckInVerifyEmployee.setLabel("Verify");
```

```
        btnCheckInVerifyEmployee.setMinimumSize(new
java.awt.Dimension(60, 20));

        btnCheckInVerifyEmployee.setPreferredSize(new
java.awt.Dimension(45, 20));

        jPnlCheckIn.add(btnCheckInVerifyEmployee, new
org.netbeans.lib.awtextra.AbsoluteConstraints(170, 90, 154, -1));


        btnCheckInSubmit.setText("Submit");

        jPnlCheckIn.add(btnCheckInSubmit, new
org.netbeans.lib.awtextra.AbsoluteConstraints(7, 557, 130, -1));


        btnCheckInClear.setText("Clear");

        jPnlCheckIn.add(btnCheckInClear, new
org.netbeans.lib.awtextra.AbsoluteConstraints(178, 557, 130, -1));


        lblCheckInEquipConditionVerification.setText("Condition Matches
Description?");

        jPnlCheckIn.add(lblCheckInEquipConditionVerification, new
org.netbeans.lib.awtextra.AbsoluteConstraints(10, 480, -1, -1));


        rdoCheckInYes.setText("Yes");

        rdoCheckInYes.addActionListener(new java.awt.event.ActionListener()

        {

            public void actionPerformed(java.awt.event.ActionEvent evt)

            {

                rdoCheckInYesActionPerformed(evt);

            }

        });
```

```
        jPnlCheckIn.add(rdoCheckInYes, new
org.netbeans.lib.awtextra.AbsoluteConstraints(210, 480, 47, -1));


        rdoCheckInEquipConditionVerificationN.setText("No");

        jPnlCheckIn.add(rdoCheckInEquipConditionVerificationN, new
org.netbeans.lib.awtextra.AbsoluteConstraints(270, 480, -1, -1));


        lblCheckInEmpId.setText("Employee ID:");

        jPnlCheckIn.add(lblCheckInEmpId, new
org.netbeans.lib.awtextra.AbsoluteConstraints(10, 60, 100, -1));


        lblCheckInTitle.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N


lblCheckInTitle.setHorizontalAlignment(javax.swing.SwingConstants.CENTER)
;

        lblCheckInTitle.setText("Check In Equipment");

        lblCheckInTitle.setPreferredSize(new java.awt.Dimension(200, 35));

        jPnlCheckIn.add(lblCheckInTitle, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 20, 303, 20));



pnlCheckInEmpInfo.setBorder(javax.swing.BorderFactory.createTitledBorder("
Employee Information"));

        pnlCheckInEmpInfo.setToolTipText("");

        pnlCheckInEmpInfo.setPreferredSize(new java.awt.Dimension(300,
100));

        pnlCheckInEmpInfo.setLayout(new
org.netbeans.lib.awtextra.AbsoluteLayout());


        lblCheckOutEmpName.setText("Frist Name:");
```

```
        pnlCheckInEmpInfo.add(lblCheckOutEmpName, new
org.netbeans.lib.awtextra.AbsoluteConstraints(10, 30, -1, 20));

        lblCheckOutEmpDepartment.setText("Last Name:");

        pnlCheckInEmpInfo.add(lblCheckOutEmpDepartment, new
org.netbeans.lib.awtextra.AbsoluteConstraints(10, 60, -1, 30));

        lblCheckOutEmpSkills.setText("Skills:");

        pnlCheckInEmpInfo.add(lblCheckOutEmpSkills, new
org.netbeans.lib.awtextra.AbsoluteConstraints(10, 90, -1, 30));

        lblCheckOutEmpRole.setText("Role:");

        pnlCheckInEmpInfo.add(lblCheckOutEmpRole, new
org.netbeans.lib.awtextra.AbsoluteConstraints(10, 130, -1, 20));

        txtCheckOutEmpName.setEditable(false);

        pnlCheckInEmpInfo.add(txtCheckOutEmpName, new
org.netbeans.lib.awtextra.AbsoluteConstraints(100, 30, 213, 30));

        txtCheckOutRole.setEditable(false);

        pnlCheckInEmpInfo.add(txtCheckOutRole, new
org.netbeans.lib.awtextra.AbsoluteConstraints(100, 120, 213, 30));

        txtCheckOutEmpDepartment1.setEditable(false);

        pnlCheckInEmpInfo.add(txtCheckOutEmpDepartment1, new
org.netbeans.lib.awtextra.AbsoluteConstraints(100, 60, 213, 30));

        txtCheckOutSkill2.setEditable(false);
```

```
        pnlCheckInEmpInfo.add(txtCheckOutSkill2, new
org.netbeans.lib.awtextra.AbsoluteConstraints(100, 90, 213, 30));


        jPnlCheckIn.add(pnlCheckInEmpInfo, new
org.netbeans.lib.awtextra.AbsoluteConstraints(10, 120, 322, 160));


        tblChekedoutEquip.setModel(new javax.swing.table.DefaultTableModel(
            new Object [][]
            {
                {null, null, null, null},
                {null, null, null, null},
                {null, null, null, null},
                {null, null, null, null}
            },
            new String []
            {
                "ID", "Type", "Status", "Due Date"
            }
        ));
        scrlCheckedInEquipPane.setViewportView(tblChekedoutEquip);


        jPnlCheckIn.add(scrlCheckedInEquipPane, new
org.netbeans.lib.awtextra.AbsoluteConstraints(10, 310, 320, 160));


        lblCheckInEquipTable.setFont(new java.awt.Font("Segoe UI", 1, 14)); //
NOI18N
        lblCheckInEquipTable.setText("Checked Out Equipment");
```

```
        jPnlCheckIn.add(lblCheckInEquipTable, new
org.netbeans.lib.awtextra.AbsoluteConstraints(80, 280, -1, -1));


        cmbCheckInEquipType.setModel(new
javax.swing.DefaultComboBoxModel<>(new String[] { "Hammer", "Cordelss
Drill", "Circular Saw", "Ladder (6ft)", "Wrench Set", "Power Washer", "Forklift",
"Pallet Jack", "Pressure Gauge", "Voltage Tester" }));
        cmbCheckInEquipType.setPreferredSize(new java.awt.Dimension(102,
20));
        jPnlCheckIn.add(cmbCheckInEquipType, new
org.netbeans.lib.awtextra.AbsoluteConstraints(110, 510, 200, 30));


        tblEmployees.setModel(new javax.swing.table.DefaultTableModel(
            new Object [][]
            {
                {null, null, null, null, null},
                {null, null, null, null, null},
                {null, null, null, null, null},
                {null, null, null, null, null}
            },
            new String []
            {
                "ID", "First Name", "Last Name", "Role", "Skill Level"
            }
        ));
        jPnlCheckIn.add(tblEmployees, new
org.netbeans.lib.awtextra.AbsoluteConstraints(10, 310, 320, 160));
```

```
        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 400, Short.MAX_VALUE)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Align
ment.LEADING)
                .addGroup(layout.createSequentialGroup()
                    .addGap(0, 0, Short.MAX_VALUE)
                    .addComponent(jPnlCheckIn,
javax.swing.GroupLayout.PREFERRED_SIZE, 340,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addGap(0, 0, Short.MAX_VALUE)))
        );
        layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 700, Short.MAX_VALUE)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Align
ment.LEADING)
                .addGroup(layout.createSequentialGroup()
                    .addGap(0, 0, Short.MAX_VALUE)
                    .addComponent(jPnlCheckIn,
javax.swing.GroupLayout.PREFERRED_SIZE, 590,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addGap(0, 0, Short.MAX_VALUE)))
        );
```

```
        pack();
    }// </editor-fold>//GEN-END:initComponents


    private void rdoCheckInYesActionPerformed(java.awt.event.ActionEvent
evt)//GEN-FIRST:event_rdoCheckInYesActionPerformed
    {//GEN-HEADEREND:event_rdoCheckInYesActionPerformed
        // TODO add your handling code here:
    }//GEN-LAST:event_rdoCheckInYesActionPerformed


    /**
     * @param args the command line arguments
     */
    public static void main(String args[])
    {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting
code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the
default look and feel.
         * For details see
http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try
        {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels())
            {
                if ("Nimbus".equals(info.getName()))
```

```
            {
                javax.swing.UIManager.setLookAndFeel(info.getClassName());
                break;
            }
        }
    } catch (ClassNotFoundException ex)
    {

java.util.logging.Logger.getLogger(CheckInWindow.class.getName()).log(java.
util.logging.Level.SEVERE, null, ex);
    } catch (InstantiationException ex)
    {

java.util.logging.Logger.getLogger(CheckInWindow.class.getName()).log(java.
util.logging.Level.SEVERE, null, ex);
    } catch (IllegalAccessException ex)
    {

java.util.logging.Logger.getLogger(CheckInWindow.class.getName()).log(java.
util.logging.Level.SEVERE, null, ex);
    } catch (javax.swing.UnsupportedLookAndFeelException ex)
    {

java.util.logging.Logger.getLogger(CheckInWindow.class.getName()).log(java.
util.logging.Level.SEVERE, null, ex);
    }
    //</editor-fold>
    //</editor-fold>
```

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

// Variables declaration - do not modify//GEN-BEGIN:variables
private javax.swing.JButton btnCheckInClear;
private javax.swing.JButton btnCheckInSubmit;
private java.awt.Button btnCheckInVerifyEmployee;
private javax.swing.JComboBox<String> cmbCheckInEquipType;
private javax.swing.JPanel jPnlCheckIn;
private javax.swing.JLabel lblCheckInEmpId;
private javax.swing.JLabel lblCheckInEquipConditionComment;
private javax.swing.JLabel lblCheckInEquipConditionVerification;
private javax.swing.JLabel lblCheckInEquipTable;
private javax.swing.JLabel lblCheckInTitle;
private javax.swing.JLabel lblCheckOutEmpDepartment;
private javax.swing.JLabel lblCheckOutEmpName;
private javax.swing.JLabel lblCheckOutEmpRole;
private javax.swing.JLabel lblCheckOutEmpSkills;
private javax.swing.JPanel pnlCheckInEmpInfo;
private javax.swing.JRadioButton rdoCheckInEquipConditionVerificationN;
```

```
    private javax.swing.JRadioButton rdoCheckInYes;
    private javax.swing.JScrollPane scrlCheckedInEquipPane;
    private javax.swing.JTable tblChekedoutEquip;
    private javax.swing.JTextField txtCheckInEmpId;
    private javax.swing.JTextField txtCheckOutEmpDepartment1;
    private javax.swing.JTextField txtCheckOutEmpName;
    private javax.swing.JTextField txtCheckOutRole;
    private javax.swing.JTextField txtCheckOutSkill2;
    private javax.swing.JTable tblEmployees;
    // End of variables declaration//GEN-END:variables


    private void verifyEmployee() {
        try {
            String empIdText = txtCheckInEmpId.getText().trim();
            if (empIdText.isEmpty()) {
                JOptionPane.showMessageDialog(this, "Please enter an Employee
ID", "Error", JOptionPane.ERROR_MESSAGE);
                return;
            }


            int employeeId = Integer.parseInt(empIdText);
            currentEmployee = employeeDAO.getEmployeeById(employeeId);


            if (currentEmployee == null) {
                JOptionPane.showMessageDialog(this, "Employee not found",
"Error", JOptionPane.ERROR_MESSAGE);
                return;
```

```
            }

            // Populate employee information
            txtCheckOutEmpName.setText(currentEmployee.getFirstName());
txtCheckOutEmpDepartment1.setText(currentEmployee.getLastName());
            txtCheckOutSkill2.setText(currentEmployee.getSkill().getSkillLevel());
            txtCheckOutRole.setText(currentEmployee.getRole().getRoleName());

            // Load checked out equipment
            loadCheckedOutEquipment(employeeId);
        } catch (NumberFormatException e) {
            JOptionPane.showMessageDialog(this, "Invalid Employee ID format",
"Error", JOptionPane.ERROR_MESSAGE);
        }
    }

    private void loadCheckedOutEquipment(int employeeId) {
        currentTransactions =
transactionDAO.getActiveTransactionsByEmployee(employeeId);
        equipmentTableModel.setRowCount(0);
        for (Transaction transaction : currentTransactions) {
            Equipment equipment =
equipmentDAO.getEquipmentById(transaction.getEquipment().getEquipment
Id());
            equipmentTableModel.addRow(new Object[]{
                equipment.getEquipmentId(),
                equipment.getEquipmentType(),
```

```
                equipment.getStatus().getStatusName(),
                transaction.getDueDate()
            });
        }
    }

    private void handleEquipmentSelection() {
        int selectedRow = tblChekedoutEquip.getSelectedRow();
        if (selectedRow >= 0) {
            int equipmentId = (int)
equipmentTableModel.getValueAt(selectedRow, 0);
            Equipment equipment =
equipmentDAO.getEquipmentById(equipmentId);
            // Set the equipment type in the combo box

cmbCheckInEquipType.setSelectedItem(equipment.getEquipmentType());
            // Set the condition radio buttons based on equipment status
            String status = equipment.getStatus().getStatusName();
            rdoCheckInYes.setSelected(status.equals("Good"));
            rdoCheckInEquipConditionVerificationN.setSelected(!
status.equals("Good"));
        }
    }

    private void submitCheckIn() {
        if (currentEmployee == null) {
            JOptionPane.showMessageDialog(this, "Please verify employee first",
"Error", JOptionPane.ERROR_MESSAGE);
```

```
            return;
        }


        int selectedRow = tblChekedoutEquip.getSelectedRow();
        if (selectedRow < 0) {
            JOptionPane.showMessageDialog(this, "Please select equipment to
check in", "Error", JOptionPane.ERROR_MESSAGE);
            return;
        }


        int equipmentId = (int) equipmentTableModel.getValueAt(selectedRow,
0);
        Transaction transaction = currentTransactions.stream()
                .filter(t -> t.getEquipment().getEquipmentId() == equipmentId)
                .findFirst()
                .orElse(null);
        if (transaction == null) {
            JOptionPane.showMessageDialog(this, "Selected equipment not found
in transactions", "Error", JOptionPane.ERROR_MESSAGE);
            return;
        }
        // Update transaction with return date
        transaction.setReturnDate(java.time.LocalDate.now());
        transactionDAO.updateTransaction(transaction);
        // Update equipment status
        Equipment equipment =
equipmentDAO.getEquipmentById(equipmentId);
        if (rdoCheckInEquipConditionVerificationN.isSelected()) {
```

```
            // If condition doesn't match, update status to "Needs Repair"
            equipment.getStatus().setStatusName("Needs Repair");
            equipmentDAO.updateEquipment(equipment);
        }
        JOptionPane.showMessageDialog(this, "Equipment checked in
successfully", "Success", JOptionPane.INFORMATION_MESSAGE);
        clearForm();
    }

    private void clearForm() {
        txtCheckInEmpId.setText("");
        txtCheckOutEmpName.setText("");
        txtCheckOutEmpDepartment1.setText("");
        txtCheckOutSkill2.setText("");
        txtCheckOutRole.setText("");
        equipmentTableModel.setRowCount(0);
        cmbCheckInEquipType.setSelectedIndex(0);
        rdoCheckInYes.setSelected(false);
        rdoCheckInEquipConditionVerificationN.setSelected(false);
        currentEmployee = null;
        currentTransactions = null;
    }

    private void loadAllEmployees() {
        try {
            List<Employee> employees = employeeDAO.getAllEmployees();
            employeeTableModel.setRowCount(0); // Clear existing data
```

```
            for (Employee employee : employees) {
                employeeTableModel.addRow(new Object[]{
                    employee.getEmployeeId(),
                    employee.getFirstName(),
                    employee.getLastName(),
                    employee.getRole().getRoleName(),
                    employee.getSkill().getSkillLevel()
                });
            }
        } catch (Exception e) {
            JOptionPane.showMessageDialog(this, "Error loading employees: " +
e.getMessage(),
                "Error", JOptionPane.ERROR_MESSAGE);
        }
    }

    private void handleEmployeeSelection() {
        int selectedRow = tblEmployees.getSelectedRow();
        if (selectedRow >= 0) {
            int employeeId = (int) employeeTableModel.getValueAt(selectedRow,
0);
            txtCheckInEmpId.setText(String.valueOf(employeeId));
            verifyEmployee();
        }
    }
}
```

CheckInWindowTester.java

```
package ecsapplication.UserInterface;

import ecsapplication.DataAccess.ActionDAO;
import ecsapplication.DataAccess.AuditingDAO;
import ecsapplication.DataAccess.DBManager;
import ecsapplication.DataAccess.EmployeeDAO;
import ecsapplication.DataAccess.EquipmentDAO;
import ecsapplication.DataAccess.EquipmentStatusDAO;
import ecsapplication.DataAccess.LocationDAO;
import ecsapplication.DataAccess.TransactionDAO;
import ecsapplication.Model.Action;
import ecsapplication.Model.Employee;
import ecsapplication.Model.Equipment;
import ecsapplication.Model.Location;
import ecsapplication.Model.EquipmentStatus;
import ecsapplication.Model.Transaction;
```

```
import java.util.List;

/**
 * Test class to verify database connectivity and DAO functionality
 * for the Check-In Window implementation
 */
public class CheckInWindowTester {

    public static void main(String[] args) {
        System.out.println("Testing CheckInWindow Database Functionality");

System.out.println("==========================================
=========");

        try {
            // Test database connection
            boolean connected = DBManager.getInstance().testConnection();
            System.out.println("Database connection successful: " + connected);
            if (!connected) {
                System.err.println("ERROR: Could not connect to database.
Aborting tests.");
                return;
            }

            // Get DAOs
            EmployeeDAO employeeDAO =
DBManager.getInstance().getDAO(EmployeeDAO.class);
```

```
        EquipmentDAO equipmentDAO =
DBManager.getInstance().getDAO(EquipmentDAO.class);
        LocationDAO locationDAO =
DBManager.getInstance().getDAO(LocationDAO.class);
        EquipmentStatusDAO statusDAO =
DBManager.getInstance().getDAO(EquipmentStatusDAO.class);
        ActionDAO actionDAO =
DBManager.getInstance().getDAO(ActionDAO.class);
        TransactionDAO transactionDAO =
DBManager.getInstance().getDAO(TransactionDAO.class);


        // Test employee retrieval
        System.out.println("\nTesting Employee Retrieval:");
        Employee employee = employeeDAO.getEmployeeById(1);
        if (employee != null) {
            System.out.println("✓ Successfully retrieved employee: " +
                    employee.getFirstName() + " " + employee.getLastName() +
                    " (ID: " + employee.getEmployeeId() + ")");
            System.out.println("   Role: " +
employee.getRole().getRoleName());
            System.out.println("   Skill: " + employee.getSkill().getSkillLevel());
        } else {
            System.out.println("✗ Failed to retrieve employee with ID 1");
        }


        // Test equipment retrieval
        System.out.println("\nTesting Equipment Retrieval:");
        Equipment equipment = equipmentDAO.getEquipmentById(1);
        if (equipment != null) {
```

```
            System.out.println("✓ Successfully retrieved equipment: " +
                    equipment.getEquipmentType() + " (ID: " +
equipment.getEquipmentId() + ")");
            System.out.println("   Status: " +
equipment.getStatus().getStatusName());
            System.out.println("   Availability: " +
equipment.getAvailability().getAvailabilityName());
            System.out.println("   Required Skill: " +
equipment.getSkill().getSkillLevel());
        } else {
            System.out.println("✗ Failed to retrieve equipment with ID 1");
        }

        // Test location retrieval
        System.out.println("\nTesting Location Retrieval:");
        List<Location> locations = locationDAO.getAllLocations();
        if (locations != null && !locations.isEmpty()) {
            System.out.println("✓ Successfully retrieved " + locations.size() + "
locations:");
            for (Location location : locations) {
                System.out.println("   ID: " + location.getLocationId() +
                        ", Name: " + location.getLocationName());
            }
        } else {
            System.out.println("✗ Failed to retrieve locations");
        }

        // Test equipment status retrieval
```

```
        System.out.println("\nTesting Equipment Status Retrieval:");
        List<EquipmentStatus> statuses = statusDAO.getAllStatuses();
        if (statuses != null && !statuses.isEmpty()) {
            System.out.println("✓ Successfully retrieved " + statuses.size() + "
equipment statuses:");
            for (EquipmentStatus status : statuses) {
                System.out.println("   ID: " + status.getStatusId() +
                        ", Name: " + status.getStatusName());
            }
        } else {
            System.out.println("✗ Failed to retrieve equipment statuses");
        }

        // Test action retrieval
        System.out.println("\nTesting Action Retrieval:");
        Action action = actionDAO.getActionById(11); // 11 =
ChangeAvailability
        if (action != null) {
            System.out.println("✓ Successfully retrieved action: " +
action.getActionName() +
                    " (ID: " + action.getActionId() + ")");
        } else {
            System.out.println("✗ Failed to retrieve action with ID 11");
        }

        // Find equipment that is checked out
        System.out.println("\nFinding Checked Out Equipment:");
        List<Equipment> allEquipment = equipmentDAO.getAllEquipment();
```

```
        int checkedOutCount = 0;


        for (Equipment equip : allEquipment) {
            if (equip.getAvailability().getAvailabilityId() == 2) { // 2 =
CheckedOut
                checkedOutCount++;
                System.out.println("   Found checked out equipment: " +
                        equip.getEquipmentType() + " (ID: " +
equip.getEquipmentId() + ")");


                // Check for transaction
                List<Transaction> transactions =
transactionDAO.getTransactionsByEquipment(equip.getEquipmentId());
                boolean activeTransactionFound = false;


                for (Transaction trans : transactions) {
                    if (trans.getReturnDate() == null) {
                        activeTransactionFound = true;
                        System.out.println("      Found active transaction (ID: " +
trans.getTransactionId() +
                                ") for employee " +
trans.getEmployee().getFirstName() + " " +
                                trans.getEmployee().getLastName());
                        break;
                    }
                }


                if (!activeTransactionFound) {
```

```
                    System.out.println("      No active transaction found for this
equipment");
                }


                if (checkedOutCount >= 3) {
                    System.out.println("   ...");
                    System.out.println("   (Total checked out: " +
checkedOutCount + ")");
                    break;
                }
            }
        }


        if (checkedOutCount == 0) {
            System.out.println("   No equipment is currently checked out");
        }


        System.out.println("\nAll tests completed. The CheckInWindow
implementation should work correctly.");


    } catch (Exception e) {
        System.err.println("ERROR during test: " + e.getMessage());
        e.printStackTrace();
    }
  }
}
```

CheckOutWindow.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/GUIForms/JFrame.java to edit this template
 */


package ecsapplication.UserInterface;

import ecsapplication.DataAccess.ActionDAO;

import ecsapplication.DataAccess.AuditingDAO;

import ecsapplication.DataAccess.DBManager;
```

```
import ecsapplication.DataAccess.EmployeeDAO;
import ecsapplication.DataAccess.EquipmentStatusDAO;
import ecsapplication.DataAccess.EquipmentDAO;
import ecsapplication.DataAccess.EquipmentAvailabilityDAO;
import ecsapplication.DataAccess.LocationDAO;
import ecsapplication.DataAccess.TransactionDAO;
import ecsapplication.Model.Action;
import ecsapplication.Model.Auditing;
import ecsapplication.Model.Employee;
import ecsapplication.Model.Equipment;
import ecsapplication.Model.EquipmentAvailability;
import ecsapplication.Model.EquipmentStatus;
import ecsapplication.Model.Location;
import ecsapplication.Model.Skill;
import ecsapplication.Model.Transaction;
import java.awt.event.ActionListener;

import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.List;
import javax.swing.ButtonGroup;
import javax.swing.DefaultComboBoxModel;
import javax.swing.JOptionPane;

/**
 *
 * @author Ughon
```

```
 */
public class CheckOutWindow extends javax.swing.JFrame {
    private MainWindow parentWindow;
    private EmployeeDAO employeeDAO;
    private EquipmentDAO equipmentDAO;
    private LocationDAO locationDAO;
    private EquipmentStatusDAO statusDAO;
    private EquipmentAvailabilityDAO availabilityDAO;
    private ActionDAO actionDAO;
    private AuditingDAO auditingDAO;
    private TransactionDAO transactionDAO;

    private Employee currentEmployee;
    private Equipment currentEquipment;
    private Transaction currentTransaction;

    /** Creates new form CheckInWindow */
    public CheckOutWindow() {
    try {
        DBManager dbManager = DBManager.getInstance();
        this.employeeDAO = dbManager.getDAO(EmployeeDAO.class);
        this.equipmentDAO = dbManager.getDAO(EquipmentDAO.class);
        // Additional DAOs initialized...
    } catch (Exception e) {
        // Error handling...
    }
    }
```

```
    /**
     * Creates new form with parent window reference
     */
    public CheckOutWindow(MainWindow parent)
    {
        this.parentWindow = parent;
        initializeDAOs();
        initComponents();
        setupComponents();
        setLocationRelativeTo(null);
        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
    }

    /**
 * Initialize all Data Access Objects
 */
    private void initializeDAOs() {
            try {
        DBManager dbManager = DBManager.getInstance();
        this.employeeDAO = dbManager.getDAO(EmployeeDAO.class);
        this.equipmentDAO = dbManager.getDAO(EquipmentDAO.class);
        this.locationDAO = dbManager.getDAO(LocationDAO.class);
        this.statusDAO = dbManager.getDAO(EquipmentStatusDAO.class);
        this.availabilityDAO =
dbManager.getDAO(EquipmentAvailabilityDAO.class);
        this.actionDAO = dbManager.getDAO(ActionDAO.class);
```

```
            this.auditingDAO = dbManager.getDAO(AuditingDAO.class);
            this.transactionDAO = dbManager.getDAO(TransactionDAO.class);
        } catch (Exception e) {
            System.err.println("Error initializing DAOs: " + e.getMessage());
            e.printStackTrace();
            JOptionPane.showMessageDialog(this,
                "Error connecting to database: " + e.getMessage(),
                "Database Error",
                JOptionPane.ERROR_MESSAGE);
        }

        }

/**
 * Setup components after initialization
 */
private void setupComponents() {
    // Load equipment types into combobox
    loadEquipmentTypesComboBox();

    // Load locations into combobox
    loadLocationsComboBox();

    // Load status options
    loadStatusComboBoxes();

    // Setup button handlers
```

```
    setupButtonHandlers();

    // Setup change listeners for filters
    setupFilterChangeListeners();
}

/**
 * Setup button event handlers
 */
private void setupButtonHandlers() {
    // Verify button handler
    btnCheckOutVerifyEmployee.addActionListener(evt -> verifyEmployee());

    // Submit button handler
    btnCheckOutSubmit.addActionListener(evt -> submitCheckOut());

    // Clear button handler
    btnCheckOutClear.addActionListener(evt -> clearAllFields());
}



/**
 * Setup change listeners for filter components
 */
private void setupFilterChangeListeners() {
    try {
        // Equipment type and location combined listener
```

```
        ActionListener updateAvailabilityInfo = evt -> {
            try {
                String selectedType = (String)
cmbCheckOutEquipType.getSelectedItem();
                String selectedLocation = (String)
lblCheckOutEquipmentLocation.getSelectedItem();


                if (selectedType != null && !selectedType.equals("Select
Equipment Type...") &&
                    selectedLocation != null && !selectedLocation.equals("Select
Location...")) {


                    // Extract location ID from selection
                    int locationId = Integer.parseInt(selectedLocation.split(" ")[0]);


                    // Get all equipment of selected type at specified location
                    List<Equipment> allEquipment =
equipmentDAO.getAllEquipment();
                    int availableCount = 0;
                    Skill requiredSkill = null;
                    StringBuilder availableEquipmentInfo = new StringBuilder();
                    availableEquipmentInfo.append("Available Equipment IDs:\n\n");


                    for (Equipment equip : allEquipment) {
                        // Check if equipment matches type and location
                        if (equip.getEquipmentType().equals(selectedType) &&
                            equip.getLocation() != null &&
                            equip.getLocation().getLocationId() == locationId) {
```

```
                    // Count and collect available equipment
                    if (equip.getAvailability().getAvailabilityId() == 1) { // 1 =
Available
                        availableCount++;
                        availableEquipmentInfo.append("ID:
").append(equip.getEquipmentId())
                                .append(" - Status:
").append(equip.getStatus().getStatusName())
                                .append("\n");
                    }


                    // Get skill requirement (should be the same for all
equipment of this type)
                    if (requiredSkill == null) {
                        requiredSkill = equip.getSkill();
                    }
                }
            }


            // Update availability field
            txtCheckInEquipName.setText("Available: " + availableCount + "
at this location");


            // Update skill level if found
            if (requiredSkill != null) {

txtCheckInEquipSkillLevel.setText(requiredSkill.getSkillLevel());
            }
```

```
                // Show popup with available equipment IDs if any are available
                if (availableCount > 0) {
                    JOptionPane.showMessageDialog(null,
                        availableEquipmentInfo.toString(),
                        "Available Equipment",
                        JOptionPane.INFORMATION_MESSAGE);
                } else {
                    JOptionPane.showMessageDialog(null,
                        "No equipment of this type is currently available at this
location.",
                        "No Available Equipment",
                        JOptionPane.INFORMATION_MESSAGE);
                }
            }
        } catch (Exception e) {
            System.err.println("Error in combo box change listener: " +
e.getMessage());
            e.printStackTrace();
            // Don't show error dialog here, just log the error to avoid
disrupting the UI
        }
    };

    // Add the same listener to both combo boxes - safely
    System.out.println("Adding listeners to combo boxes");
    if (cmbCheckOutEquipType != null) {
        cmbCheckOutEquipType.addActionListener(updateAvailabilityInfo);
```

```
        } else {
            System.err.println("WARNING: cmbCheckOutEquipType is null");
        }

        if (lblCheckOutEquipmentLocation != null) {
lblCheckOutEquipmentLocation.addActionListener(updateAvailabilityInfo);
        } else {
            System.err.println("WARNING: lblCheckOutEquipmentLocation is
null");
        }

        // Status selection changes
        if (cmbCheckOutEquipStatus != null) {
            cmbCheckOutEquipStatus.addActionListener(evt -> {
                try {
                    String selectedStatus = (String)
cmbCheckOutEquipStatus.getSelectedItem();
                    if (selectedStatus != null && txtCheckInEquipCondition != null)
{
                        txtCheckInEquipCondition.setText(selectedStatus);
                    }
                } catch (Exception e) {
                    System.err.println("Error in status selection: " +
e.getMessage());
                }
            });
        }
```

```
        System.out.println("Filter change listeners setup completed");
    } catch (Exception e) {
        System.err.println("ERROR setting up filter change listeners: " +
e.getMessage());
        e.printStackTrace();
    }
}


    /**
     * Load equipment types into combobox
     */
    private void loadEquipmentTypesComboBox() {
        try {
            // Get all distinct equipment types from database
            List<Equipment> allEquipment = equipmentDAO.getAllEquipment();

            // Create a list of unique equipment types
            java.util.Set<String> uniqueTypes = new java.util.HashSet<>();
            for (Equipment equipment : allEquipment) {
                uniqueTypes.add(equipment.getEquipmentType());
            }

            // Create and set the model
            DefaultComboBoxModel<String> model = new
DefaultComboBoxModel<>();
            model.addElement("Select Equipment Type...");
```

```
            for (String type : uniqueTypes) {
                model.addElement(type);
            }

            cmbCheckOutEquipType.setModel(model);

        } catch (Exception e) {
            System.err.println("Error loading equipment types: " +
e.getMessage());
            e.printStackTrace();
        }
    }

    /**
     * Load locations into combobox
     */
    private void loadLocationsComboBox() {
        try {
            // Get all locations from database
            List<Location> locations = locationDAO.getAllLocations();

            // Create and set the model
            DefaultComboBoxModel<String> model = new
DefaultComboBoxModel<>();
            model.addElement("Select Location...");

            for (Location location : locations) {
```

```
            model.addElement(location.getLocationId() + " " +
location.getLocationName());
        }

        lblCheckOutEquipmentLocation.setModel(model);

    } catch (Exception e) {
        System.err.println("Error loading locations: " + e.getMessage());
        e.printStackTrace();
    }
}

/**
 * Load equipment status and condition options
 */
private void loadStatusComboBoxes() {
    try {
        // Get all statuses from database
        List<EquipmentStatus> statuses = statusDAO.getAllStatuses();

        // Create and set the model for condition combobox
        DefaultComboBoxModel<String> conditionModel = new
DefaultComboBoxModel<>();
        for (EquipmentStatus status : statuses) {
            conditionModel.addElement(status.getStatusName());
        }
```

```
        // Only set the model for the remaining status combobox
        cmbCheckOutEquipStatus.setModel(conditionModel);

    } catch (Exception e) {
        System.err.println("Error loading status options: " + e.getMessage());
        e.printStackTrace();
    }
}

    /**
 * Verify employee ID and load employee information
 */
private void verifyEmployee() {
    System.out.println("verifyEmployee called"); // Add this debug line
    try {
        // Get employee ID from text field
        String employeeIdStr = txtCheckOutEmpId.getText().trim();

        if (employeeIdStr.isEmpty()) {
            JOptionPane.showMessageDialog(this,
                "Please enter an Employee ID.",
                "Validation Error",
                JOptionPane.WARNING_MESSAGE);
            return;
        }

        // Parse employee ID as integer
```

```
int employeeId;
try {
    employeeId = Integer.parseInt(employeeIdStr);
} catch (NumberFormatException e) {
    JOptionPane.showMessageDialog(this,
        "Please enter a valid numeric Employee ID.",
        "Validation Error",
        JOptionPane.WARNING_MESSAGE);
    return;
}

// Query employee from database
Employee employee = employeeDAO.getEmployeeById(employeeId);

if (employee == null) {
    JOptionPane.showMessageDialog(this,
        "Employee not found with ID: " + employeeId,
        "Employee Not Found",
        JOptionPane.WARNING_MESSAGE);
    return;
}

// Save current employee
currentEmployee = employee;

// Populate employee info fields
txtCheckInEmpName.setText(employee.getFirstName());
```

```
        txtCheckInEmpDepartment1.setText(employee.getLastName());
        txtCheckskill2.setText(employee.getSkill().getSkillLevel());
        txtCheckIeRole.setText(employee.getRole().getRoleName());

        // Show success message
        JOptionPane.showMessageDialog(this,
            "Employee verified: " + employee.getFirstName() + " " +
employee.getLastName(),
            "Employee Verified",
            JOptionPane.INFORMATION_MESSAGE);

    } catch (Exception e) {
        JOptionPane.showMessageDialog(this,
            "Error verifying employee: " + e.getMessage(),
            "Database Error",
            JOptionPane.ERROR_MESSAGE);
        e.printStackTrace();
    }
}

/**
 * Verify equipment ID and load equipment information
 */
private boolean verifyEquipment() {
    try {
        // Get equipment ID from text field
        String equipmentIdStr = txtCheckOutInputEquipID.getText().trim();
```

```
        if (equipmentIdStr.isEmpty()) {
            JOptionPane.showMessageDialog(this,
                "Please enter an Equipment ID.",
                "Validation Error",
                JOptionPane.WARNING_MESSAGE);
            return false;
        }

        // Parse equipment ID as integer
        int equipmentId;
        try {
            equipmentId = Integer.parseInt(equipmentIdStr);
        } catch (NumberFormatException e) {
            JOptionPane.showMessageDialog(this,
                "Please enter a valid numeric Equipment ID.",
                "Validation Error",
                JOptionPane.WARNING_MESSAGE);
            return false;
        }

        // Query equipment from database
        Equipment equipment =
equipmentDAO.getEquipmentById(equipmentId);

        if (equipment == null) {
            JOptionPane.showMessageDialog(this,
```

```
            "Equipment not found with ID: " + equipmentId,
            "Equipment Not Found",
            JOptionPane.WARNING_MESSAGE);
        return false;
    }

    // Check if equipment is checked out
    if (equipment.getAvailability().getAvailabilityId() != 2) { // 2 =
CheckedOut
        JOptionPane.showMessageDialog(this,
            "This equipment is not currently checked out. Current status: " +
            equipment.getAvailability().getAvailabilityName(),
            "Equipment Status Error",
            JOptionPane.WARNING_MESSAGE);
        return false;
    }

    // Find the transaction for this equipment
    List<Transaction> transactions =
transactionDAO.getTransactionsByEquipment(equipmentId);
    Transaction activeTransaction = null;

    for (Transaction transaction : transactions) {
        if (transaction.getReturnDate() == null) {
            activeTransaction = transaction;
            break;
        }
```

```
        }

        if (activeTransaction == null) {
            JOptionPane.showMessageDialog(this,
                "No active transaction found for this equipment.",
                "Transaction Error",
                JOptionPane.WARNING_MESSAGE);
            return false;
        }

        // Save current equipment and transaction
        currentEquipment = equipment;
        currentTransaction = activeTransaction;

        // Populate equipment info fields

txtCheckInEquipName.setText(equipment.getAvailability().getAvailabilityNam
e());
        txtCheckInEquipSkillLevel.setText(equipment.getSkill().getSkillLevel());

txtCheckInEquipCondition.setText(equipment.getStatus().getStatusName());


        // Set equipment type in combobox
        for (int i = 0; i < cmbCheckOutEquipType.getItemCount(); i++) {
            if
(cmbCheckOutEquipType.getItemAt(i).equals(equipment.getEquipmentType()
)) {
                cmbCheckOutEquipType.setSelectedIndex(i);
```

```
            break;
        }
    }

    // Set status in remaining combobox
    String currentStatus = equipment.getStatus().getStatusName();
    for (int i = 0; i < cmbCheckOutEquipStatus.getItemCount(); i++) {
        if (cmbCheckOutEquipStatus.getItemAt(i).equals(currentStatus)) {
            cmbCheckOutEquipStatus.setSelectedIndex(i);
            break;
        }
    }

    // Set location if available
    if (equipment.getLocation() != null) {
        String locationString = equipment.getLocation().getLocationId() + " "
+ equipment.getLocation().getLocationName();
        for (int i = 0; i < lblCheckOutEquipmentLocation.getItemCount(); i+
+) {
            if
(lblCheckOutEquipmentLocation.getItemAt(i).toString().equals(locationString)
) {
                lblCheckOutEquipmentLocation.setSelectedIndex(i);
                break;
            }
        }
    }
```

```
            return true;

        } catch (Exception e) {
            JOptionPane.showMessageDialog(this,
                "Error verifying equipment: " + e.getMessage(),
                "Database Error",
                JOptionPane.ERROR_MESSAGE);
            e.printStackTrace();
            return false;
        }
    }

    private void submitCheckOut() {
        System.out.println("submitCheckOut started"); // Add this debug line
        try {
            // Verify employee
            if (currentEmployee == null) {
                JOptionPane.showMessageDialog(this,
                    "Please verify an employee first.",
                    "Validation Error",
                    JOptionPane.WARNING_MESSAGE);
                return;
            }

            // Get equipment ID and verify
            String equipmentIdStr = txtCheckOutInputEquipID.getText().trim();
            if (equipmentIdStr.isEmpty()) {
```

```
            JOptionPane.showMessageDialog(this,
                "Please enter an Equipment ID.",
                "Validation Error",
                JOptionPane.WARNING_MESSAGE);
            return;
        }

        // Parse equipment ID
        int equipmentId;
        try {
            equipmentId = Integer.parseInt(equipmentIdStr);
        } catch (NumberFormatException e) {
            JOptionPane.showMessageDialog(this,
                "Please enter a valid numeric Equipment ID.",
                "Validation Error",
                JOptionPane.WARNING_MESSAGE);
            return;
        }

        // Query equipment from database
        Equipment equipment =
equipmentDAO.getEquipmentById(equipmentId);
        if (equipment == null) {
            JOptionPane.showMessageDialog(this,
                "Equipment not found with ID: " + equipmentId,
                "Equipment Not Found",
                JOptionPane.WARNING_MESSAGE);
```

```
            return;
        }

        // Verify equipment is available
        if (equipment.getAvailability().getAvailabilityId() != 1) { // 1 = Available
            JOptionPane.showMessageDialog(this,
                "This equipment is not available for checkout. Current status: " +
                equipment.getAvailability().getAvailabilityName(),
                "Equipment Status Error",
                JOptionPane.WARNING_MESSAGE);
            return;
        }

        // Rest of the method remains the same...
        // Get selected location
        int locationIndex = lblCheckOutEquipmentLocation.getSelectedIndex();
        if (locationIndex == -1) {
            JOptionPane.showMessageDialog(this,
                "Please select a location.",
                "Validation Error",
                JOptionPane.WARNING_MESSAGE);
            return;
        }

        // Get location ID from the selected item
        String locationItem = (String)
lblCheckOutEquipmentLocation.getSelectedItem();
```

```
        int locationId = Integer.parseInt(locationItem.split(" ")[0]);
        Location location = locationDAO.getLocationById(locationId);

// Create new transaction
Transaction transaction = new Transaction();
transaction.setEmployee(currentEmployee);
transaction.setEquipment(equipment);
transaction.setTransactionDate(LocalDateTime.now()); // Changed from setCheckoutDate
transaction.setDueDate(LocalDate.now().plusDays(7)); // Add a default due date of 7 days

// Save transaction to database
boolean transactionCreated = transactionDAO.createTransaction(transaction);
if (!transactionCreated) {
    JOptionPane.showMessageDialog(this,
        "Error creating transaction record.",
        "Database Error",
        JOptionPane.ERROR_MESSAGE);
    return;
}

// Update equipment
// Set availability to Checked Out (ID=2)
EquipmentAvailability checkedOutStatus = availabilityDAO.getAvailabilityById(2);
equipment.setAvailability(checkedOutStatus);
```

```
// Set equipment location
equipment.setLocation(location);

// Set employee reference
equipment.setEmployee(currentEmployee);


        // Update equipment in database
        boolean equipmentUpdated =
equipmentDAO.updateEquipment(equipment);
        if (!equipmentUpdated) {
            JOptionPane.showMessageDialog(this,
                "Error updating equipment status.",
                "Database Error",
                JOptionPane.ERROR_MESSAGE);
            return;
        }


        // Create audit log entry
        Action action = actionDAO.getActionById(10); // 10 = CheckOut


        Auditing audit = new Auditing();
        audit.setTimestamp(LocalDateTime.now());
        audit.setDetails("Checked out " + equipment.getEquipmentType() +
                        " (ID: " + equipment.getEquipmentId() + ") to " +
                        currentEmployee.getFirstName() + " " +
currentEmployee.getLastName() +
```

```
                ". Location: " + location.getLocationName());
        audit.setEmployee(currentEmployee);
        audit.setAction(action);

        boolean auditCreated = auditingDAO.createAuditLog(audit);
        if (!auditCreated) {
            // Log the error but don't stop the process
            System.err.println("Warning: Failed to create audit log entry");
        }

        // Show success message
        JOptionPane.showMessageDialog(this,
            "Equipment successfully checked out to " +
currentEmployee.getFirstName() + " " + currentEmployee.getLastName(),
            "Success",
            JOptionPane.INFORMATION_MESSAGE);

        // Clear form
        clearAllFields();

        // Refresh parent window if needed
        if (parentWindow != null && parentWindow instanceof MainWindow) {
            // Only call refreshData if it exists
            try {

parentWindow.getClass().getMethod("refreshData").invoke(parentWindow);
            } catch (Exception e) {
```

```
                // Method doesn't exist, silently ignore
                System.out.println("Note: refreshData method not found in
MainWindow");
            }
        }

    } catch (Exception e) {
        JOptionPane.showMessageDialog(this,
            "Error processing checkout: " + e.getMessage(),
            "Error",
            JOptionPane.ERROR_MESSAGE);
        e.printStackTrace();
    }
}
/**
 * Clear all input fields and reset form
 */
private void clearAllFields() {
    // Temporarily remove the action listener from the equipment ID field
    ActionListener[] listeners = txtCheckOutInputEquipID.getActionListeners();
    for (ActionListener listener : listeners) {
        txtCheckOutInputEquipID.removeActionListener(listener);
    }

    // Clear text fields
    txtCheckOutEmpId.setText("");
    txtCheckInEmpName.setText("");
```

```
    txtCheckInEmpDepartment1.setText("");
    txtCheckskill2.setText("");
    txtCheckIeRole.setText("");
    txtCheckInEquipName.setText("");
    txtCheckInEquipSkillLevel.setText("");
    txtCheckInEquipCondition.setText("");
    txtCheckOutInputEquipID.setText("");

    // Reset comboboxes
    cmbCheckOutEquipType.setSelectedIndex(0);
    lblCheckOutEquipmentLocation.setSelectedIndex(0);

    // Reset current objects
    currentEmployee = null;
    currentEquipment = null;
    currentTransaction = null;

    // Re-add the action listener to the equipment ID field
    for (ActionListener listener : listeners) {
        txtCheckOutInputEquipID.addActionListener(listener);
    }
}
    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
```

```
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-
BEGIN:initComponents
    private void initComponents()
    {


        jPnlCheckIn = new javax.swing.JPanel();
        lblCheckOutEquip = new javax.swing.JLabel();
        pnlCheckOutEquipInfo = new javax.swing.JPanel();
        lblCheckInEquipName = new javax.swing.JLabel();
        lblCheckInEquipSkill = new javax.swing.JLabel();
        txtCheckInEquipName = new javax.swing.JTextField();
        txtCheckInEquipSkillLevel = new javax.swing.JTextField();
        txtCheckInEquipCondition = new javax.swing.JTextField();
        lblCheckInEquipCondition = new javax.swing.JLabel();
        lblCheckOutEquipConditionComment = new javax.swing.JLabel();
        txtCheckOutEmpId = new javax.swing.JTextField();
        btnCheckOutVerifyEmployee = new java.awt.Button();
        pnlCheckOutEmpInfo = new javax.swing.JPanel();
        lblCheckInEmpName = new javax.swing.JLabel();
        lblCheckInEmpDepartment = new javax.swing.JLabel();
        lblCheckInEmpSkills = new javax.swing.JLabel();
        lblCheckInEmpRole = new javax.swing.JLabel();
        txtCheckInEmpName = new javax.swing.JTextField();
        txtCheckIeRole = new javax.swing.JTextField();
        txtCheckInEmpDepartment1 = new javax.swing.JTextField();
        txtCheckskill2 = new javax.swing.JTextField();
```

```
        btnCheckOutSubmit = new javax.swing.JButton();
        btnCheckOutClear = new javax.swing.JButton();
        lblCheckOutEmpId = new javax.swing.JLabel();
        lblCheckOutTitle = new javax.swing.JLabel();
        lblCheckOutLocation = new javax.swing.JLabel();
        cmbCheckOutEquipType = new javax.swing.JComboBox<>();
        lblCheckOutEquipmentLocation = new javax.swing.JComboBox<>();
        lblCheckOutInputEquipID = new javax.swing.JLabel();
        txtCheckOutInputEquipID = new javax.swing.JTextField();
        cmbCheckOutEquipStatus = new javax.swing.JComboBox<>();
        jLabel1 = new javax.swing.JLabel();
        jTextField1 = new javax.swing.JTextField();


setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("ECS - Check In Window");
        setMinimumSize(new java.awt.Dimension(400, 700));
        setPreferredSize(new java.awt.Dimension(400, 700));
        getContentPane().setLayout(new
org.netbeans.lib.awtextra.AbsoluteLayout());


        jPnlCheckIn.setBorder(javax.swing.BorderFactory.createLineBorder(new
java.awt.Color(0, 0, 0)));
        jPnlCheckIn.setPreferredSize(new java.awt.Dimension(400, 650));
        jPnlCheckIn.setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());


        lblCheckOutEquip.setText("Equipment Type:");
```

```
        jPnlCheckIn.add(lblCheckOutEquip, new
org.netbeans.lib.awtextra.AbsoluteConstraints(10, 300, 120, -1));



pnlCheckOutEquipInfo.setBorder(javax.swing.BorderFactory.createTitledBorde
r("Equipment Information"));

        pnlCheckOutEquipInfo.setToolTipText("");

        pnlCheckOutEquipInfo.setPreferredSize(new java.awt.Dimension(300,
100));

        pnlCheckOutEquipInfo.setLayout(new
org.netbeans.lib.awtextra.AbsoluteLayout());



        lblCheckInEquipName.setText("Availability:");

        pnlCheckOutEquipInfo.add(lblCheckInEquipName, new
org.netbeans.lib.awtextra.AbsoluteConstraints(10, 20, -1, -1));



        lblCheckInEquipSkill.setText("Skills:");

        pnlCheckOutEquipInfo.add(lblCheckInEquipSkill, new
org.netbeans.lib.awtextra.AbsoluteConstraints(10, 50, 73, 30));



        txtCheckInEquipName.setEditable(false);

        pnlCheckOutEquipInfo.add(txtCheckInEquipName, new
org.netbeans.lib.awtextra.AbsoluteConstraints(90, 20, 220, -1));



        txtCheckInEquipSkillLevel.setEditable(false);

        pnlCheckOutEquipInfo.add(txtCheckInEquipSkillLevel, new
org.netbeans.lib.awtextra.AbsoluteConstraints(90, 50, 220, -1));



        txtCheckInEquipCondition.setEditable(false);
```

```
        pnlCheckOutEquipInfo.add(txtCheckInEquipCondition, new
org.netbeans.lib.awtextra.AbsoluteConstraints(92, 158, 235, 20));


        lblCheckInEquipCondition.setText("Condition:");

        pnlCheckOutEquipInfo.add(lblCheckInEquipCondition, new
org.netbeans.lib.awtextra.AbsoluteConstraints(11, 158, -1, -1));


        jPnlCheckIn.add(pnlCheckOutEquipInfo, new
org.netbeans.lib.awtextra.AbsoluteConstraints(10, 350, 320, 90));


        lblCheckOutEquipConditionComment.setText("Condition:");

        jPnlCheckIn.add(lblCheckOutEquipConditionComment, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 480, -1, -1));


        txtCheckOutEmpId.setPreferredSize(new java.awt.Dimension(64, 30));

        txtCheckOutEmpId.addActionListener(new
java.awt.event.ActionListener()

        {

            public void actionPerformed(java.awt.event.ActionEvent evt)

            {

                txtCheckOutEmpIdActionPerformed(evt);

            }

        });

        jPnlCheckIn.add(txtCheckOutEmpId, new
org.netbeans.lib.awtextra.AbsoluteConstraints(7, 70, 158, 30));


        btnCheckOutVerifyEmployee.setLabel("Verify");

        btnCheckOutVerifyEmployee.setMinimumSize(new
java.awt.Dimension(60, 20));
```

```
        btnCheckOutVerifyEmployee.setPreferredSize(new
java.awt.Dimension(45, 20));

        jPnlCheckIn.add(btnCheckOutVerifyEmployee, new
org.netbeans.lib.awtextra.AbsoluteConstraints(175, 80, 154, -1));



pnlCheckOutEmpInfo.setBorder(javax.swing.BorderFactory.createTitledBorder
("Employee Information"));

        pnlCheckOutEmpInfo.setToolTipText("");

        pnlCheckOutEmpInfo.setPreferredSize(new java.awt.Dimension(300,
100));

        pnlCheckOutEmpInfo.setLayout(new
org.netbeans.lib.awtextra.AbsoluteLayout());


        lblCheckInEmpName.setText("Frist Name:");

        pnlCheckOutEmpInfo.add(lblCheckInEmpName, new
org.netbeans.lib.awtextra.AbsoluteConstraints(11, 22, -1, -1));


        lblCheckInEmpDepartment.setText("Last Name:");

        pnlCheckOutEmpInfo.add(lblCheckInEmpDepartment, new
org.netbeans.lib.awtextra.AbsoluteConstraints(10, 60, -1, -1));


        lblCheckInEmpSkills.setText("Skills:");

        pnlCheckOutEmpInfo.add(lblCheckInEmpSkills, new
org.netbeans.lib.awtextra.AbsoluteConstraints(10, 100, -1, -1));


        lblCheckInEmpRole.setText("Role:");

        pnlCheckOutEmpInfo.add(lblCheckInEmpRole, new
org.netbeans.lib.awtextra.AbsoluteConstraints(10, 140, -1, 20));
```

```
        txtCheckInEmpName.setEditable(false);
        txtCheckInEmpName.setPreferredSize(new java.awt.Dimension(64, 30));
        pnlCheckOutEmpInfo.add(txtCheckInEmpName, new
org.netbeans.lib.awtextra.AbsoluteConstraints(98, 22, 213, 30));


        txtCheckIeRole.setEditable(false);
        pnlCheckOutEmpInfo.add(txtCheckIeRole, new
org.netbeans.lib.awtextra.AbsoluteConstraints(100, 140, 213, -1));


        txtCheckInEmpDepartment1.setEditable(false);
        pnlCheckOutEmpInfo.add(txtCheckInEmpDepartment1, new
org.netbeans.lib.awtextra.AbsoluteConstraints(100, 60, 213, 30));


        txtCheckskill2.setEditable(false);
        pnlCheckOutEmpInfo.add(txtCheckskill2, new
org.netbeans.lib.awtextra.AbsoluteConstraints(100, 100, 213, 30));


        jPnlCheckIn.add(pnlCheckOutEmpInfo, new
org.netbeans.lib.awtextra.AbsoluteConstraints(7, 100, 322, 180));


        btnCheckOutSubmit.setText("Submit");
        btnCheckOutSubmit.addActionListener(new
java.awt.event.ActionListener()
        {
            public void actionPerformed(java.awt.event.ActionEvent evt)
            {
                btnCheckOutSubmitActionPerformed(evt);
            }
        });
```

```
        jPnlCheckIn.add(btnCheckOutSubmit, new
org.netbeans.lib.awtextra.AbsoluteConstraints(7, 557, 130, -1));


        btnCheckOutClear.setText("Clear");

        jPnlCheckIn.add(btnCheckOutClear, new
org.netbeans.lib.awtextra.AbsoluteConstraints(168, 557, 130, -1));


        lblCheckOutEmpId.setText("Employee ID:");

        jPnlCheckIn.add(lblCheckOutEmpId, new
org.netbeans.lib.awtextra.AbsoluteConstraints(7, 49, 332, -1));


        lblCheckOutTitle.setFont(new java.awt.Font("Segoe UI", 1, 14)); //
NOI18N

lblCheckOutTitle.setHorizontalAlignment(javax.swing.SwingConstants.CENTE
R);

        lblCheckOutTitle.setText("Check Out Equipment");

lblCheckOutTitle.setVerticalAlignment(javax.swing.SwingConstants.TOP);

        lblCheckOutTitle.setPreferredSize(new java.awt.Dimension(200, 35));

        jPnlCheckIn.add(lblCheckOutTitle, new
org.netbeans.lib.awtextra.AbsoluteConstraints(7, 23, 332, 20));


        lblCheckOutLocation.setText("Location");

        jPnlCheckIn.add(lblCheckOutLocation, new
org.netbeans.lib.awtextra.AbsoluteConstraints(10, 320, -1, -1));


        cmbCheckOutEquipType.setModel(new
javax.swing.DefaultComboBoxModel<>(new String[] { "Hammer", "Cordelss
Drill", "Circular Saw", "Ladder (6ft)", "Wrench Set", "Power Washer", "Forklift",
"Pallet Jack", "Pressure Gauge", "Voltage Tester" }));
```

```
        cmbCheckOutEquipType.setPreferredSize(new java.awt.Dimension(102,
20));

        jPnlCheckIn.add(cmbCheckOutEquipType, new
org.netbeans.lib.awtextra.AbsoluteConstraints(140, 290, 190, 30));


        lblCheckOutEquipmentLocation.setModel(new
javax.swing.DefaultComboBoxModel<>(new String[] { "1 North Warehouse",
"2 South Warehouse " }));

        lblCheckOutEquipmentLocation.setPreferredSize(new
java.awt.Dimension(102, 20));

        lblCheckOutEquipmentLocation.addActionListener(new
java.awt.event.ActionListener()

        {

            public void actionPerformed(java.awt.event.ActionEvent evt)

            {

                lblCheckOutEquipmentLocationActionPerformed(evt);

            }

        });

        jPnlCheckIn.add(lblCheckOutEquipmentLocation, new
org.netbeans.lib.awtextra.AbsoluteConstraints(140, 320, 190, 30));


        lblCheckOutInputEquipID.setText("Input Equipment ID:");

        jPnlCheckIn.add(lblCheckOutInputEquipID, new
org.netbeans.lib.awtextra.AbsoluteConstraints(10, 440, -1, -1));


        txtCheckOutInputEquipID.addActionListener(new
java.awt.event.ActionListener()

        {

            public void actionPerformed(java.awt.event.ActionEvent evt)

            {
```

```
            txtCheckOutInputEquipIDActionPerformed(evt);
        }
    });
    jPnlCheckIn.add(txtCheckOutInputEquipID, new
org.netbeans.lib.awtextra.AbsoluteConstraints(150, 440, 170, -1));

    cmbCheckOutEquipStatus.setModel(new
javax.swing.DefaultComboBoxModel<>(new String[] { "Functional",
"Damaged", "NeedsRepair", "InRepair" }));
    cmbCheckOutEquipStatus.setPreferredSize(new java.awt.Dimension(65,
26));
    jPnlCheckIn.add(cmbCheckOutEquipStatus, new
org.netbeans.lib.awtextra.AbsoluteConstraints(100, 480, 230, 30));

    jLabel1.setText("Due Date:");
    jPnlCheckIn.add(jLabel1, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 520, -1, -1));
    jPnlCheckIn.add(jTextField1, new
org.netbeans.lib.awtextra.AbsoluteConstraints(130, 520, 130, -1));

    getContentPane().add(jPnlCheckIn, new
org.netbeans.lib.awtextra.AbsoluteConstraints(25, 25, 340, 590));

    pack();
}// </editor-fold>//GEN-END:initComponents

private void
lblCheckOutEquipmentLocationActionPerformed(java.awt.event.ActionEvent
evt)//GEN-FIRST:event_lblCheckOutEquipmentLocationActionPerformed
```

```
    {//GEN-
HEADEREND:event_lblCheckOutEquipmentLocationActionPerformed
        // TODO add your handling code here:
    }//GEN-LAST:event_lblCheckOutEquipmentLocationActionPerformed


    private void
txtCheckOutInputEquipIDActionPerformed(java.awt.event.ActionEvent
evt)//GEN-FIRST:event_txtCheckOutInputEquipIDActionPerformed
    {//GEN-HEADEREND:event_txtCheckOutInputEquipIDActionPerformed
            // Automatically verify equipment when user presses Enter
        System.out.println("Equipment ID field action performed"); // Add this
debug line
    verifyEquipment();
    }//GEN-LAST:event_txtCheckOutInputEquipIDActionPerformed


    private void
txtCheckOutEmpIdActionPerformed(java.awt.event.ActionEvent evt)//GEN-
FIRST:event_txtCheckOutEmpIdActionPerformed
    {//GEN-HEADEREND:event_txtCheckOutEmpIdActionPerformed
        // TODO add your handling code here:
    }//GEN-LAST:event_txtCheckOutEmpIdActionPerformed


    private void
btnCheckOutSubmitActionPerformed(java.awt.event.ActionEvent evt)//GEN-
FIRST:event_btnCheckOutSubmitActionPerformed
    {//GEN-HEADEREND:event_btnCheckOutSubmitActionPerformed
        submitCheckOut();
    }//GEN-LAST:event_btnCheckOutSubmitActionPerformed
```

```
    /**
     * @param args the command line arguments
     */
  public static void main(String args[]) {
    // Test database connection first
    boolean connected = DBManager.getInstance().testConnection();
    if (!connected) {
        System.err.println("ERROR: Could not connect to database. Please check database configuration.");
        JOptionPane.showMessageDialog(null,
            "Could not connect to database. Please check database configuration.",
            "Database Connection Error",
            JOptionPane.ERROR_MESSAGE);
        return;
    }


    /* Set the Nimbus look and feel */
    //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
    /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
     * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
     */
    try {
        for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
            if ("Nimbus".equals(info.getName())) {
```

```
                javax.swing.UIManager.setLookAndFeel(info.getClassName());
                break;
            }
        }
    } catch (ClassNotFoundException ex) {
java.util.logging.Logger.getLogger(CheckOutWindow.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
    } catch (InstantiationException ex) {
java.util.logging.Logger.getLogger(CheckOutWindow.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
    } catch (IllegalAccessException ex) {
java.util.logging.Logger.getLogger(CheckOutWindow.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
    } catch (javax.swing.UnsupportedLookAndFeelException ex) {
java.util.logging.Logger.getLogger(CheckOutWindow.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
    }
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
```

//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>

//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>
//</editor-fold>

```
    //</editor-fold>
    //</editor-fold>

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

// Variables declaration - do not modify//GEN-BEGIN:variables
private javax.swing.JButton btnCheckOutClear;
private javax.swing.JButton btnCheckOutSubmit;
private java.awt.Button btnCheckOutVerifyEmployee;
private javax.swing.JComboBox<String> cmbCheckOutEquipStatus;
private javax.swing.JComboBox<String> cmbCheckOutEquipType;
private javax.swing.JLabel jLabel1;
private javax.swing.JPanel jPnlCheckIn;
private javax.swing.JTextField jTextField1;
private javax.swing.JLabel lblCheckInEmpDepartment;
private javax.swing.JLabel lblCheckInEmpName;
private javax.swing.JLabel lblCheckInEmpRole;
private javax.swing.JLabel lblCheckInEmpSkills;
private javax.swing.JLabel lblCheckInEquipCondition;
private javax.swing.JLabel lblCheckInEquipName;
private javax.swing.JLabel lblCheckInEquipSkill;
```

```
    private javax.swing.JLabel lblCheckOutEmpId;
    private javax.swing.JLabel lblCheckOutEquip;
    private javax.swing.JLabel lblCheckOutEquipConditionComment;
    private javax.swing.JComboBox<String> lblCheckOutEquipmentLocation;
    private javax.swing.JLabel lblCheckOutInputEquipID;
    private javax.swing.JLabel lblCheckOutLocation;
    private javax.swing.JLabel lblCheckOutTitle;
    private javax.swing.JPanel pnlCheckOutEmpInfo;
    private javax.swing.JPanel pnlCheckOutEquipInfo;
    private javax.swing.JTextField txtCheckIeRole;
    private javax.swing.JTextField txtCheckInEmpDepartment1;
    private javax.swing.JTextField txtCheckInEmpName;
    private javax.swing.JTextField txtCheckInEquipCondition;
    private javax.swing.JTextField txtCheckInEquipName;
    private javax.swing.JTextField txtCheckInEquipSkillLevel;
    private javax.swing.JTextField txtCheckOutEmpId;
    private javax.swing.JTextField txtCheckOutInputEquipID;
    private javax.swing.JTextField txtCheckskill2;
    // End of variables declaration//GEN-END:variables

}
```

EmployeeTerminationWindow.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/GUIForms/JFrame.java to edit this template
 */
package ecsapplication.UserInterface;
import javax.swing.JOptionPane;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.text.ParseException;
import ecsapplication.DataAccess.EmployeeService;
import ecsapplication.DataAccess.EquipmentDAO;
import ecsapplication.DataAccess.DBManager;
import ecsapplication.DataAccess.TransactionDAO;
import ecsapplication.DataAccess.ValueLossDAO;
import ecsapplication.Model.Employee;
import ecsapplication.Model.Equipment;
import ecsapplication.Model.Transaction;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.List;
import java.math.BigDecimal;

/**
 *
```

```
 * @author Redux
 */
public class EmployeeTerminationWindow extends javax.swing.JFrame
{
    private EmployeeService employeeService;
    private EquipmentDAO equipmentDAO;
    private TransactionDAO transactionDAO;
    private ValueLossDAO valueLossDAO;
    private Employee currentEmployee;
    private Employee loggedInEmployee;

    /**
     * Creates new form EmployeeTerminationWindow
     */
    public EmployeeTerminationWindow()
    {
        this(null); // Call the other constructor with null loggedInEmployee
    }

    public EmployeeTerminationWindow(Employee loggedInEmployee)
    {
        try {
            initComponents();
            setLocationRelativeTo(null);
            setDefaultCloseOperation(DISPOSE_ON_CLOSE);
            lblStatusPending.setVisible(false);
            lblStatusComplete.setVisible(false);
```

```
            lblStatusNone.setVisible(true);


            this.loggedInEmployee = loggedInEmployee;
            this.employeeService = new EmployeeService();
            this.equipmentDAO =
DBManager.getInstance().getDAO(EquipmentDAO.class);
            this.transactionDAO =
DBManager.getInstance().getDAO(TransactionDAO.class);
            this.valueLossDAO =
DBManager.getInstance().getDAO(ValueLossDAO.class);
        } catch (Exception e) {
            e.printStackTrace();
            JOptionPane.showMessageDialog(this,
                "Error initializing window: " + e.getMessage(),
                "Initialization Error",
                JOptionPane.ERROR_MESSAGE);
            dispose(); // Close the window if initialization fails
        }
    }


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

```
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents
    private void initComponents()
    {
        try {
            // Add NetBeans AbsoluteLayout library
            Class.forName("org.netbeans.lib.awtextra.AbsoluteLayout");

            jPanel1 = new javax.swing.JPanel();
            lblEmployeeTerminationProcess = new javax.swing.JLabel();
            lblEnterEmployeeID = new javax.swing.JLabel();
            txtSearchEmployeeID = new javax.swing.JTextField();
            btnSearch = new javax.swing.JButton();
            jScrollPane2 = new javax.swing.JScrollPane();
            txtProcessNote = new javax.swing.JTextArea();
            lblEmployeeInformation = new javax.swing.JLabel();
            lblFirstName = new javax.swing.JLabel();
            lblSkillLevel = new javax.swing.JLabel();
            lblLastName = new javax.swing.JLabel();
            lblLastDay = new javax.swing.JLabel();
            lblRole = new javax.swing.JLabel();
            txtLastName = new javax.swing.JTextField();
            txtFirstName = new javax.swing.JTextField();
            txtSkillLevel = new javax.swing.JTextField();
            txtLastDay = new javax.swing.JTextField();
            txtRole = new javax.swing.JTextField();
            lblTerminationStatus = new javax.swing.JLabel();
```

```
        lblStatusPending = new javax.swing.JLabel();
        lblStatusComplete = new javax.swing.JLabel();
        lblStatusNone = new javax.swing.JLabel();
        jLabel4 = new javax.swing.JLabel();
        jScrollPane1 = new javax.swing.JScrollPane();
        tblEquipmentToBeReturned = new javax.swing.JTable();
        lblTotalItemsText = new javax.swing.JLabel();
        lblTotalItems = new javax.swing.JLabel();
        lblTotalValueText = new javax.swing.JLabel();
        lblTotalValue = new javax.swing.JLabel();
        btnPrintEquipmentList = new javax.swing.JButton();
        btnProcessTermination1 = new javax.swing.JButton();
        lblLastDay1 = new javax.swing.JLabel();
        txtLastDay1 = new javax.swing.JTextField();
        lblEmail = new javax.swing.JLabel();
        txtEmail = new javax.swing.JTextField();


setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("Equipment Checkout System - Admin View");
        setPreferredSize(new java.awt.Dimension(970, 730));
        setResizable(false);

        jPanel1.setPreferredSize(new java.awt.Dimension(950, 750));
        jPanel1.setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());
```

```
        lblEmployeeTerminationProcess.setFont(new java.awt.Font("Segoe
UI", 1, 16)); // NOI18N

        lblEmployeeTerminationProcess.setText("Employee Termination
Process");

        jPanel1.add(lblEmployeeTerminationProcess, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 20, 260, 30));


        lblEnterEmployeeID.setText("Enter Employee ID:");

        jPanel1.add(lblEnterEmployeeID, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 50, 110, 30));

        jPanel1.add(txtSearchEmployeeID, new
org.netbeans.lib.awtextra.AbsoluteConstraints(130, 52, 300, 30));


        btnSearch.setText("Search");

        btnSearch.addActionListener(new java.awt.event.ActionListener() {

            public void actionPerformed(java.awt.event.ActionEvent evt) {

                btnSearchActionPerformed(evt);

            }

        });

        jPanel1.add(btnSearch, new
org.netbeans.lib.awtextra.AbsoluteConstraints(440, 50, 80, 30));



jScrollPane2.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTI
CAL_SCROLLBAR_NEVER);

        jScrollPane2.setWheelScrollingEnabled(false);


        txtProcessNote.setEditable(false);

        txtProcessNote.setBackground(new java.awt.Color(255, 255, 204));
```

```
        txtProcessNote.setColumns(20);

        txtProcessNote.setLineWrap(true);

        txtProcessNote.setRows(5);

        txtProcessNote.setText("Note: This process will:\n1. Remove selected
employee from the system\n2. Remove all outstanding equipment checked
out by this employee from the system.");

        txtProcessNote.setWrapStyleWord(true);

        jScrollPane2.setViewportView(txtProcessNote);


        jPanel1.add(jScrollPane2, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 100, 300, 70));


        lblEmployeeInformation.setFont(new java.awt.Font("Segoe UI", 1,
16)); // NOI18N

        lblEmployeeInformation.setText("Employee Information");

        jPanel1.add(lblEmployeeInformation, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 180, 230, 30));


        lblFirstName.setText("First Name:");

        jPanel1.add(lblFirstName, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 230, -1, -1));


        lblSkillLevel.setText("Skill Level:");

        jPanel1.add(lblSkillLevel, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 290, -1, -1));


        lblLastName.setText("Last Name:");

        jPanel1.add(lblLastName, new
org.netbeans.lib.awtextra.AbsoluteConstraints(290, 230, -1, -1));
```

```
        lblLastDay.setText("Last Day:");
        jPanel1.add(lblLastDay, new
org.netbeans.lib.awtextra.AbsoluteConstraints(580, 290, -1, -1));

        lblRole.setText("Role:");
        jPanel1.add(lblRole, new
org.netbeans.lib.awtextra.AbsoluteConstraints(290, 290, -1, -1));

        txtLastName.setEditable(false);
        txtLastName.setBackground(new java.awt.Color(255, 255, 255));
        txtLastName.addActionListener(new java.awt.event.ActionListener()
        {
            public void actionPerformed(java.awt.event.ActionEvent evt)
            {
                txtLastNameActionPerformed(evt);
            }
        });
        jPanel1.add(txtLastName, new
org.netbeans.lib.awtextra.AbsoluteConstraints(360, 230, 200, -1));

        txtFirstName.setEditable(false);
        txtFirstName.setBackground(new java.awt.Color(255, 255, 255));
        jPanel1.add(txtFirstName, new
org.netbeans.lib.awtextra.AbsoluteConstraints(100, 230, 170, -1));

        txtSkillLevel.setEditable(false);
        txtSkillLevel.setBackground(new java.awt.Color(255, 255, 255));
```

```
        jPanel1.add(txtSkillLevel, new
org.netbeans.lib.awtextra.AbsoluteConstraints(100, 290, 170, -1));


        txtLastDay.addActionListener(new java.awt.event.ActionListener()
        {
            public void actionPerformed(java.awt.event.ActionEvent evt)
            {
                txtLastDayActionPerformed(evt);
            }
        });
        jPanel1.add(txtLastDay, new
org.netbeans.lib.awtextra.AbsoluteConstraints(660, 290, 200, -1));


        txtRole.setEditable(false);
        txtRole.setBackground(new java.awt.Color(255, 255, 255));
        txtRole.addActionListener(new java.awt.event.ActionListener()
        {
            public void actionPerformed(java.awt.event.ActionEvent evt)
            {
                txtRoleActionPerformed(evt);
            }
        });
        jPanel1.add(txtRole, new
org.netbeans.lib.awtextra.AbsoluteConstraints(360, 290, 200, -1));


        lblTerminationStatus.setText("Termination Status:");
        jPanel1.add(lblTerminationStatus, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 360, -1, -1));
```

```
        lblStatusPending.setForeground(new java.awt.Color(255, 0, 51));
        lblStatusPending.setText("Pending - Equipment Not Returned");
        jPanel1.add(lblStatusPending, new
org.netbeans.lib.awtextra.AbsoluteConstraints(140, 360, -1, -1));

        lblStatusComplete.setForeground(new java.awt.Color(0, 204, 0));
        lblStatusComplete.setText("Complete");
        jPanel1.add(lblStatusComplete, new
org.netbeans.lib.awtextra.AbsoluteConstraints(140, 360, -1, -1));

        lblStatusNone.setForeground(new java.awt.Color(102, 102, 102));
        lblStatusNone.setText("N/A");
        jPanel1.add(lblStatusNone, new
org.netbeans.lib.awtextra.AbsoluteConstraints(140, 360, -1, -1));

        jLabel4.setFont(new java.awt.Font("Segoe UI", 1, 16)); // NOI18N
        jLabel4.setText("Equipment To Be Returned");
        jPanel1.add(jLabel4, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 400, -1, -1));

        tblEquipmentToBeReturned.setModel(new
javax.swing.table.DefaultTableModel(
            new Object [][]
            {
                {null, null, null, null, null, null},
                {null, null, null, null, null, null},
                {null, null, null, null, null, null},
```

```
                {null, null, null, null, null, null},
                {null, null, null, null, null, null},
                {null, null, null, null, null, null},
                {null, null, null, null, null, null},
                {null, null, null, null, null, null},
                {null, null, null, null, null, null},
                {null, null, null, null, null, null},
                {null, null, null, null, null, null},
                {null, null, null, null, null, null},
                {null, null, null, null, null, null},
                {null, null, null, null, null, null},
                {null, null, null, null, null, null}
            },
            new String []
            {
                "Equipment ID", "Name", "Checkout Date", "Due Date", "Value",
"Status"
            }
        ));
        jScrollPane1.setViewportView(tblEquipmentToBeReturned);


        jPanel1.add(jScrollPane1, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 430, 910, 180));


        lblTotalItemsText.setText("Total Items:");
        jPanel1.add(lblTotalItemsText, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 630, -1, -1));
```

```
        lblTotalItems.setText("-");

        jPanel1.add(lblTotalItems, new
org.netbeans.lib.awtextra.AbsoluteConstraints(90, 630, 60, -1));


        lblTotalValueText.setText("Total Value:");

        jPanel1.add(lblTotalValueText, new
org.netbeans.lib.awtextra.AbsoluteConstraints(210, 630, -1, -1));


        lblTotalValue.setText("-");

        jPanel1.add(lblTotalValue, new
org.netbeans.lib.awtextra.AbsoluteConstraints(280, 630, -1, -1));


        btnPrintEquipmentList.setText("Print Equipment List");

        jPanel1.add(btnPrintEquipmentList, new
org.netbeans.lib.awtextra.AbsoluteConstraints(770, 630, 150, 40));


        btnProcessTermination1.setText("Process Termination");

        btnProcessTermination1.addActionListener(new
java.awt.event.ActionListener()
        {
            public void actionPerformed(java.awt.event.ActionEvent evt)
            {
                btnProcessTermination1ActionPerformed(evt);
            }
        });

        jPanel1.add(btnProcessTermination1, new
org.netbeans.lib.awtextra.AbsoluteConstraints(610, 630, 150, 40));


        lblLastDay1.setText("Last Day:");
```

```
        jPanel1.add(lblLastDay1, new
org.netbeans.lib.awtextra.AbsoluteConstraints(580, 290, -1, -1));


        txtLastDay1.addActionListener(new java.awt.event.ActionListener()
        {
            public void actionPerformed(java.awt.event.ActionEvent evt)
            {
                txtLastDay1ActionPerformed(evt);
            }
        });
        jPanel1.add(txtLastDay1, new
org.netbeans.lib.awtextra.AbsoluteConstraints(660, 290, 200, -1));


        lblEmail.setText("Email:");
        jPanel1.add(lblEmail, new
org.netbeans.lib.awtextra.AbsoluteConstraints(580, 230, -1, -1));


        txtEmail.setEditable(false);
        txtEmail.setBackground(new java.awt.Color(255, 255, 255));
        txtEmail.addActionListener(new java.awt.event.ActionListener()
        {
            public void actionPerformed(java.awt.event.ActionEvent evt)
            {
                txtEmailActionPerformed(evt);
            }
        });
        jPanel1.add(txtEmail, new
org.netbeans.lib.awtextra.AbsoluteConstraints(660, 230, 200, -1));
```

```
            javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());

            getContentPane().setLayout(layout);

            layout.setHorizontalGroup(


layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

                .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE,
998, Short.MAX_VALUE)

            );

            layout.setVerticalGroup(


layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

                .addComponent(jPanel1,
javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.DEFAULT_SIZE, 762, Short.MAX_VALUE)

            );



            pack();

        } catch (ClassNotFoundException e) {

            throw new RuntimeException("Required NetBeans AbsoluteLayout
library is missing", e);

        }

    }// </editor-fold>//GEN-END:initComponents


    private void txtLastNameActionPerformed(java.awt.event.ActionEvent
evt)//GEN-FIRST:event_txtLastNameActionPerformed

    {//GEN-HEADEREND:event_txtLastNameActionPerformed

        // TODO add your handling code here:
```

```
    }//GEN-LAST:event_txtLastNameActionPerformed


    private void txtLastDayActionPerformed(java.awt.event.ActionEvent
evt)//GEN-FIRST:event_txtLastDayActionPerformed
    {//GEN-HEADEREND:event_txtLastDayActionPerformed
        // TODO add your handling code here:
    }//GEN-LAST:event_txtLastDayActionPerformed


    private void txtRoleActionPerformed(java.awt.event.ActionEvent
evt)//GEN-FIRST:event_txtRoleActionPerformed
    {//GEN-HEADEREND:event_txtRoleActionPerformed
        // TODO add your handling code here:
    }//GEN-LAST:event_txtRoleActionPerformed


    private void
btnProcessTermination1ActionPerformed(java.awt.event.ActionEvent evt) {
        if (currentEmployee == null) {
            JOptionPane.showMessageDialog(this,
                "Please search for an employee first.",
                "Process Error",
                JOptionPane.WARNING_MESSAGE);
            return;
        }

        String lastDay = txtLastDay.getText().trim();

        // Validate format using regex
        if (!lastDay.matches("\\d{4}-\\d{2}-\\d{2}")) {
```

```
            JOptionPane.showMessageDialog(this,
                "Please enter the last day in yyyy-MM-dd format (e.g., 2025-04-
05).",
                "Date Format Error",
                JOptionPane.WARNING_MESSAGE);
            return;
        }

        // Validate the date itself
        try {
            SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
            sdf.setLenient(false); // force strict matching
            Date parsedDate = sdf.parse(lastDay);

            // Get any equipment that will be marked as lost
            List<Equipment> checkedOutEquipment =
equipmentDAO.getEquipmentByEmployee(currentEmployee.getEmployeeId()
);
            BigDecimal totalLostValue = BigDecimal.ZERO;

            boolean equipmentProcessed = true;
            String equipmentError = null;

            if (!checkedOutEquipment.isEmpty()) {
                totalLostValue = checkedOutEquipment.stream()
                    .map(Equipment::getValue)
                    .reduce(BigDecimal.ZERO, BigDecimal::add);
```

```
            // Process equipment loss using the DAO
            try {
                System.out.println("Processing equipment loss for employee: "
+ currentEmployee.getEmployeeId());
                valueLossDAO.processTerminationEquipmentLoss(
                    currentEmployee.getEmployeeId(),
                    currentEmployee.getFirstName() + " " +
currentEmployee.getLastName()
                );
                System.out.println("Equipment loss processing completed
successfully");
            } catch (Exception e) {
                equipmentProcessed = false;
                equipmentError = e.getMessage();
                System.err.println("Error processing equipment loss: " +
e.getMessage());
                e.printStackTrace();
            }
        }

        if (!equipmentProcessed) {
            JOptionPane.showMessageDialog(this,
                "Error processing equipment loss:\n" + equipmentError,
                "Process Error",
                JOptionPane.ERROR_MESSAGE);
            return;
        }
```

```
            // Process termination
            System.out.println("Processing employee termination");
            boolean success =
employeeService.removeEmployee(currentEmployee.getEmployeeId(),
loggedInEmployee);

            if (success) {
                StringBuilder message = new StringBuilder();
                message.append("Employee termination processed successfully.\
n");
                message.append("Employee:
").append(currentEmployee.getFirstName()).append("
").append(currentEmployee.getLastName()).append("\n");
                message.append("Last Day: ").append(lastDay).append("\n");

                if (!checkedOutEquipment.isEmpty()) {
                    message.append("\nEquipment marked as lost:\n");
                    message.append("Total Items:
").append(checkedOutEquipment.size()).append("\n");
                    message.append("Total Value Lost:
$").append(String.format("%.2f", totalLostValue));
                }

                System.out.println("Termination completed successfully");
                JOptionPane.showMessageDialog(this,
                    message.toString(),
                    "Termination Complete",
                    JOptionPane.INFORMATION_MESSAGE);
```

```
                // Clear form
                clearForm();
            } else {
                System.err.println("Employee termination failed");
                JOptionPane.showMessageDialog(this,
                    "Failed to process termination. The equipment was processed but employee removal failed.\n" +
                    "Please contact system administrator.",
                    "Process Error",
                    JOptionPane.ERROR_MESSAGE);
            }

        } catch (ParseException e) {
            System.err.println("Date parsing error: " + e.getMessage());
            JOptionPane.showMessageDialog(this,
                "The entered date is not valid. Please enter a real date in yyyy-MM-dd format.",
                "Invalid Date",
                JOptionPane.WARNING_MESSAGE);
        }
    }

    private void txtLastDay1ActionPerformed(java.awt.event.ActionEvent evt)//GEN-FIRST:event_txtLastDay1ActionPerformed
    {//GEN-HEADEREND:event_txtLastDay1ActionPerformed
        // TODO add your handling code here:
    }//GEN-LAST:event_txtLastDay1ActionPerformed
```

```
    private void txtEmailActionPerformed(java.awt.event.ActionEvent
evt)//GEN-FIRST:event_txtEmailActionPerformed

    {//GEN-HEADEREND:event_txtEmailActionPerformed

        // TODO add your handling code here:

    }//GEN-LAST:event_txtEmailActionPerformed


    private void btnSearchActionPerformed(java.awt.event.ActionEvent evt) {

        try {

            int employeeId =
Integer.parseInt(txtSearchEmployeeID.getText().trim());

            currentEmployee = employeeService.getEmployeeById(employeeId);


            if (currentEmployee == null) {

                JOptionPane.showMessageDialog(this,

                    "Employee not found with ID: " + employeeId,

                    "Search Error",

                    JOptionPane.WARNING_MESSAGE);

                return;

            }


            // Display employee information

            txtFirstName.setText(currentEmployee.getFirstName());

            txtLastName.setText(currentEmployee.getLastName());

            txtEmail.setText(currentEmployee.getEmail());

            txtRole.setText(currentEmployee.getRole().getRoleName());

            txtSkillLevel.setText(currentEmployee.getSkill().getSkillLevel());
```

```
        // Check for equipment
        List<Equipment> checkedOutEquipment =
equipmentDAO.getEquipmentByEmployee(employeeId);
        if (!checkedOutEquipment.isEmpty()) {
            lblStatusPending.setVisible(true);
            lblStatusComplete.setVisible(false);
            lblStatusNone.setVisible(false);

            // Populate equipment table
            javax.swing.table.DefaultTableModel model =
(javax.swing.table.DefaultTableModel)
tblEquipmentToBeReturned.getModel();
            model.setRowCount(0); // Clear existing rows

            BigDecimal totalValue = BigDecimal.ZERO;
            for (Equipment equipment : checkedOutEquipment) {
                // Get the active transaction for this equipment
                Transaction transaction =
transactionDAO.getActiveTransactionByEquipment(equipment.getEquipmentI
d());

                model.addRow(new Object[]{
                    equipment.getEquipmentId(),
                    equipment.getEquipmentType(),
                    transaction != null ? transaction.getTransactionDate() : "N/A",
                    transaction != null ? transaction.getDueDate() : "N/A",
                    equipment.getValue(),
                    equipment.getStatus().getStatusName()
```

```
                });
                totalValue = totalValue.add(equipment.getValue());
            }

            lblTotalItems.setText(String.valueOf(checkedOutEquipment.size()));
            lblTotalValue.setText("$" + String.format("%.2f", totalValue));
        } else {
            lblStatusPending.setVisible(false);
            lblStatusComplete.setVisible(true);
            lblStatusNone.setVisible(false);

            // Clear equipment table
            javax.swing.table.DefaultTableModel model =
(javax.swing.table.DefaultTableModel)
tblEquipmentToBeReturned.getModel();
            model.setRowCount(0);

            lblTotalItems.setText("-");
            lblTotalValue.setText("-");
        }

    } catch (NumberFormatException e) {
        JOptionPane.showMessageDialog(this,
            "Please enter a valid numeric Employee ID.",
            "Input Error",
            JOptionPane.WARNING_MESSAGE);
    }
```

```
    }

    private void clearForm() {
        txtSearchEmployeeID.setText("");
        txtFirstName.setText("");
        txtLastName.setText("");
        txtEmail.setText("");
        txtRole.setText("");
        txtSkillLevel.setText("");
        txtLastDay.setText("");

        javax.swing.table.DefaultTableModel model =
(javax.swing.table.DefaultTableModel)
tblEquipmentToBeReturned.getModel();
        model.setRowCount(0);

        lblTotalItems.setText("-");
        lblTotalValue.setText("-");

        lblStatusPending.setVisible(false);
        lblStatusComplete.setVisible(false);
        lblStatusNone.setVisible(true);

        currentEmployee = null;
    }

    /**
```

```
     * @param args the command line arguments
     */
    public static void main(String args[])
    {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
         * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try
        {
            for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels())
            {
                if ("Nimbus".equals(info.getName()))
                {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex)
        {
java.util.logging.Logger.getLogger(EmployeeTerminationWindow.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (InstantiationException ex)
```

```
        {

java.util.logging.Logger.getLogger(EmployeeTerminationWindow.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex)
        {

java.util.logging.Logger.getLogger(EmployeeTerminationWindow.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex)
        {

java.util.logging.Logger.getLogger(EmployeeTerminationWindow.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        }
        //</editor-fold>

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

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JButton btnPrintEquipmentList;
```

```
private javax.swing.JButton btnProcessTermination1;
private javax.swing.JButton btnSearch;
private javax.swing.JLabel jLabel4;
private javax.swing.JPanel jPanel1;
private javax.swing.JScrollPane jScrollPane1;
private javax.swing.JScrollPane jScrollPane2;
private javax.swing.JLabel lblEmail;
private javax.swing.JLabel lblEmployeeInformation;
private javax.swing.JLabel lblEmployeeTerminationProcess;
private javax.swing.JLabel lblEnterEmployeeID;
private javax.swing.JLabel lblFirstName;
private javax.swing.JLabel lblLastDay;
private javax.swing.JLabel lblLastDay1;
private javax.swing.JLabel lblLastName;
private javax.swing.JLabel lblRole;
private javax.swing.JLabel lblSkillLevel;
private javax.swing.JLabel lblStatusComplete;
private javax.swing.JLabel lblStatusNone;
private javax.swing.JLabel lblStatusPending;
private javax.swing.JLabel lblTerminationStatus;
private javax.swing.JLabel lblTotalItems;
private javax.swing.JLabel lblTotalItemsText;
private javax.swing.JLabel lblTotalValue;
private javax.swing.JLabel lblTotalValueText;
private javax.swing.JTable tblEquipmentToBeReturned;
private javax.swing.JTextField txtEmail;
private javax.swing.JTextField txtFirstName;
```

```
    private javax.swing.JTextField txtLastDay;
    private javax.swing.JTextField txtLastDay1;
    private javax.swing.JTextField txtLastName;
    private javax.swing.JTextArea txtProcessNote;
    private javax.swing.JTextField txtRole;
    private javax.swing.JTextField txtSearchEmployeeID;
    private javax.swing.JTextField txtSkillLevel;
    // End of variables declaration//GEN-END:variables
}
```

EquipmentManagementWindow.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/GUIForms/JFrame.java to edit this template
 */
```

```
package ecsapplication.UserInterface;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.sql.*;
import ecsapplication.DataAccess.DatabaseConnection;

/**
 *
 * @author Redux
 */
public class EquipmentManagementWindow extends javax.swing.JFrame
{
    private DefaultTableModel model;

    /**
     * Creates new form EquipmentManagementWindow
     */
    public EquipmentManagementWindow()
    {
        initComponents();
        setLocationRelativeTo(null);
        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
        setupTable();
        loadEquipmentData();

        // Add search button action listener
```

```
        btnSearchEquipment.addActionListener(evt -> performSearch());
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents
    private void initComponents()
    {

        jPanel1 = new javax.swing.JPanel();
        jPanel2 = new javax.swing.JPanel();
        btnSearchEquipment = new javax.swing.JButton();
        txtSearchEquipment = new javax.swing.JTextField();
        btnAddNewEquipment = new javax.swing.JButton();
        jScrollPane1 = new javax.swing.JScrollPane();
        tblEquipmentList = new javax.swing.JTable();
        lblSearchByEquip = new javax.swing.JLabel();


setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("Equipment Checkout System - Equipment");
        setPreferredSize(new java.awt.Dimension(920, 565));
```

```
        setResizable(false);


        jPanel2.setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());


        btnSearchEquipment.setBackground(new java.awt.Color(51, 102, 255));
        btnSearchEquipment.setForeground(new java.awt.Color(255, 255,
255));
        btnSearchEquipment.setText("Search");
        jPanel2.add(btnSearchEquipment, new
org.netbeans.lib.awtextra.AbsoluteConstraints(760, 20, 120, 30));
        jPanel2.add(txtSearchEquipment, new
org.netbeans.lib.awtextra.AbsoluteConstraints(510, 20, 240, 30));


        btnAddNewEquipment.setBackground(new java.awt.Color(51, 102,
255));
        btnAddNewEquipment.setForeground(new java.awt.Color(255, 255,
255));
        btnAddNewEquipment.setText("Add New Equipment");
        btnAddNewEquipment.addActionListener(new
java.awt.event.ActionListener()
        {
            public void actionPerformed(java.awt.event.ActionEvent evt)
            {
                btnAddNewEquipmentActionPerformed(evt);
            }
        });
        jPanel2.add(btnAddNewEquipment, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 20, 140, 30));
```

```
tblEquipmentList.setModel(new javax.swing.table.DefaultTableModel(
    new Object [][]
    {
        {null, null, null, null, null, null, null},
        {null, null, null, null, null, null, null},
        {null, null, null, null, null, null, null},
        {null, null, null, null, null, null, null},
        {null, null, null, null, null, null, null},
        {null, null, null, null, null, null, null},
        {null, null, null, null, null, null, null},
        {null, null, null, null, null, null, null},
        {null, null, null, null, null, null, null},
        {null, null, null, null, null, null, null},
        {null, null, null, null, null, null, null},
        {null, null, null, null, null, null, null},
        {null, null, null, null, null, null, null},
        {null, null, null, null, null, null, null},
        {null, null, null, null, null, null, null},
        {null, null, null, null, null, null, null},
        {null, null, null, null, null, null, null},
        {null, null, null, null, null, null, null},
        {null, null, null, null, null, null, null},
        {null, null, null, null, null, null, null},
        {null, null, null, null, null, null, null},
        {null, null, null, null, null, null, null},
        {null, null, null, null, null, null, null},
        {null, null, null, null, null, null, null},
```

```
            {null, null, null, null, null, null, null}
        },
        new String []
        {
            "Equipment ID", "Name", "Category", "Required Skill", "Status",
"Checked Out To", "Actions"
        }
    ));
    jScrollPane1.setViewportView(tblEquipmentList);


    jPanel2.add(jScrollPane1, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 80, 860, 410));


    lblSearchByEquip.setText("Search By Type:");
    jPanel2.add(lblSearchByEquip, new
org.netbeans.lib.awtextra.AbsoluteConstraints(370, 30, -1, -1));


    javax.swing.GroupLayout jPanel1Layout = new
javax.swing.GroupLayout(jPanel1);
    jPanel1.setLayout(jPanel1Layout);
    jPanel1Layout.setHorizontalGroup(

jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
        .addComponent(jPanel2, javax.swing.GroupLayout.DEFAULT_SIZE,
920, Short.MAX_VALUE)
    );
    jPanel1Layout.setVerticalGroup(
```

```
jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
            .addComponent(jPanel2, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
        );


        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
        );
        layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
        );


        pack();
    }// </editor-fold>//GEN-END:initComponents


    private void
btnAddNewEquipmentActionPerformed(java.awt.event.ActionEvent
evt)//GEN-FIRST:event_btnAddNewEquipmentActionPerformed
    {//GEN-HEADEREND:event_btnAddNewEquipmentActionPerformed
```

```
        AddEquipmentWindow addWindow = new AddEquipmentWindow(this);
        addWindow.setLocationRelativeTo(null); // center the window
        addWindow.setVisible(true);
    }//GEN-LAST:event_btnAddNewEquipmentActionPerformed

    private void setupTable() {
        // Setup the table model with column names
        model = (DefaultTableModel) tblEquipmentList.getModel();
        model.setColumnCount(7); // Set number of columns
        model.setRowCount(0);  // Clear existing rows

        // Set column headers
        String[] columnNames = {"Equipment ID", "Equipment Type",
"Equipment Code", "Required Skill", "Status", "Checked Out To", "Actions"};
        model.setColumnIdentifiers(columnNames);
    }

    public void loadEquipmentData() {
        model.setRowCount(0); // Clear existing rows

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(
                "SELECT equipment_id, equipment_type, equipment_code, " +
                "skill_level, status_name, " +
                "CONCAT(COALESCE(e.first_name, ''), ' ', COALESCE(e.last_name,
'')) as checked_out_to, " +
                "location_name " +
```

```
            "FROM equipment eq " +
            "LEFT JOIN skills s ON eq.skill_id = s.skill_id " +
            "LEFT JOIN equipmentstatus es ON eq.status_id = es.status_id " +
            "LEFT JOIN employee e ON eq.employee_id = e.employee_id " +
            "LEFT JOIN location l ON eq.location_id = l.location_id " +
            "ORDER BY equipment_id")) {

        ResultSet rs = stmt.executeQuery();

        while (rs.next()) {
            model.addRow(new Object[]{
                rs.getInt("equipment_id"),
                rs.getString("equipment_type"),
                rs.getString("equipment_code"),
                rs.getString("skill_level"),
                rs.getString("status_name"),
                rs.getString("checked_out_to"),
                "View/Edit"  // Action button text
            });
        }

    } catch (SQLException ex) {
        JOptionPane.showMessageDialog(this,
            "Error loading equipment data: " + ex.getMessage(),
            "Database Error",
            JOptionPane.ERROR_MESSAGE);
        ex.printStackTrace();
```

```
        }
    }

    private void performSearch() {
        String searchTerm = txtSearchEquipment.getText().trim();

        if (searchTerm.isEmpty()) {
            loadEquipmentData(); // If search is empty, load all data
            return;
        }

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(
                "SELECT eq.equipment_id, eq.equipment_type,
eq.equipment_code, " +
                "s.skill_level, es.status_name, " +
                "CONCAT(COALESCE(e.first_name, ''), ' ', COALESCE(e.last_name,
'')) as checked_out_to, " +
                "l.location_name " +
                "FROM equipment eq " +
                "LEFT JOIN skills s ON eq.skill_id = s.skill_id " +
                "LEFT JOIN equipmentstatus es ON eq.status_id = es.status_id " +
                "LEFT JOIN employee e ON eq.employee_id = e.employee_id " +
                "LEFT JOIN location l ON eq.location_id = l.location_id " +
                "WHERE eq.equipment_id LIKE ? " +
                "OR LOWER(eq.equipment_type) LIKE LOWER(?) " +
                "OR LOWER(eq.equipment_code) LIKE LOWER(?) " +
```

```
            "OR LOWER(s.skill_level) LIKE LOWER(?) " +
            "OR LOWER(es.status_name) LIKE LOWER(?) " +
            "OR LOWER(l.location_name) LIKE LOWER(?) " +
            "OR LOWER(CONCAT(COALESCE(e.first_name, ''), ' ',
COALESCE(e.last_name, ''))) LIKE LOWER(?) " +
            "ORDER BY eq.equipment_id")) {

        String likePattern = "%" + searchTerm + "%";
        // Set the search pattern for each searchable field
        stmt.setString(1, likePattern);  // equipment_id
        stmt.setString(2, likePattern);  // equipment_type
        stmt.setString(3, likePattern);  // equipment_code
        stmt.setString(4, likePattern);  // skill_level
        stmt.setString(5, likePattern);  // status_name
        stmt.setString(6, likePattern);  // location_name
        stmt.setString(7, likePattern);  // employee name

        ResultSet rs = stmt.executeQuery();
        model.setRowCount(0); // Clear existing rows

        while (rs.next()) {
            model.addRow(new Object[]{
                rs.getInt("equipment_id"),
                rs.getString("equipment_type"),
                rs.getString("equipment_code"),
                rs.getString("skill_level"),
                rs.getString("status_name"),
```

```
                rs.getString("checked_out_to"),
                "View/Edit"
            });
        }

    } catch (SQLException ex) {
        JOptionPane.showMessageDialog(this,
            "Error searching equipment: " + ex.getMessage(),
            "Database Error",
            JOptionPane.ERROR_MESSAGE);
        ex.printStackTrace();
    }
}

/**
 * @param args the command line arguments
 */
public static void main(String args[])
{
    /* Set the Nimbus look and feel */
    //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
    /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
     * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
     */
    try
```

```
        {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels())
            {
                if ("Nimbus".equals(info.getName()))
                {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex)
        {

java.util.logging.Logger.getLogger(EquipmentManagementWindow.class.getN
ame()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (InstantiationException ex)
        {

java.util.logging.Logger.getLogger(EquipmentManagementWindow.class.getN
ame()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex)
        {

java.util.logging.Logger.getLogger(EquipmentManagementWindow.class.getN
ame()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex)
        {

java.util.logging.Logger.getLogger(EquipmentManagementWindow.class.getN
ame()).log(java.util.logging.Level.SEVERE, null, ex);
```

```
        }
        //</editor-fold>

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

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JButton btnAddNewEquipment;
    private javax.swing.JButton btnSearchEquipment;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JLabel lblSearchByEquip;
    private javax.swing.JTable tblEquipmentList;
    private javax.swing.JTextField txtSearchEquipment;
    // End of variables declaration//GEN-END:variables
}
```

EquipmentUsageReportPanel.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/GUIForms/JPanel.java to edit this template
 */
package ecsapplication.UserInterface;
```

```
import ecsapplication.DataAccess.DatabaseConnection;
import ecsapplication.DataAccess.EquipmentDAO;
import ecsapplication.DataAccess.TransactionDAO;
import ecsapplication.Model.Equipment;
import javax.swing.table.DefaultTableModel;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Date;
import javax.swing.JOptionPane;
import java.sql.PreparedStatement;
import java.sql.Timestamp;
import java.lang.StringBuilder;

/**
 *
 * @author Ughon
 */
public class EquipmentUsageReportPanel extends javax.swing.JPanel
{
    private TransactionDAO transactionDAO;
    private EquipmentDAO equipmentDAO;
    private DefaultTableModel tableModel;

    /**
     * Creates new form EquipmentUsageReportq
```

```
     */
    public EquipmentUsageReportPanel()
    {
        initComponents();
        transactionDAO = new TransactionDAO();
        equipmentDAO = new EquipmentDAO();
        tableModel = (DefaultTableModel) tblEquipmentUsage.getModel();
        loadLast100Transactions();
        populateEquipmentTypes();
    }


    private void loadPanelContent(javax.swing.JPanel contentPanel) {
    // Clear existing components
    pnlEquipmentUsageReport.removeAll();


    // Add the title back
    pnlEquipmentUsageReport.add(lbllTitleUsage, new
org.netbeans.lib.awtextra.AbsoluteConstraints(10, 5, -1, -1));


    // Resize panel to fit content
    pnlEquipmentUsageReport.setPreferredSize(new java.awt.Dimension(600,
520));


    // Add the new content panel
    pnlEquipmentUsageReport.add(contentPanel, new
org.netbeans.lib.awtextra.AbsoluteConstraints(0, 40, 600, 480));


    // Refresh the panel
```

```
    pnlEquipmentUsageReport.revalidate();
    pnlEquipmentUsageReport.repaint();
}


    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents
    private void initComponents()
    {


        pnlEquipmentUsageReport = new javax.swing.JPanel();
        lbllTitleUsage = new javax.swing.JLabel();
        lblDateRange = new javax.swing.JLabel();
        lbFromDate = new javax.swing.JLabel();
        lblToDate = new javax.swing.JLabel();
        dateStart = new com.toedter.calendar.JDateChooser();
        dateEnd = new com.toedter.calendar.JDateChooser();
        btnGenerate = new javax.swing.JButton();
        comboEquipmentType = new javax.swing.JComboBox<>();
        lblComboEquipmentType = new javax.swing.JLabel();
        jScrollPane1 = new javax.swing.JScrollPane();
```

```
        tblEquipmentUsage = new javax.swing.JTable();
        btnExportUserReport = new javax.swing.JButton();

        setPreferredSize(new java.awt.Dimension(600, 550));
        setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

        pnlEquipmentUsageReport.setName("ReportFrame"); // NOI18N
        pnlEquipmentUsageReport.setPreferredSize(new
java.awt.Dimension(600, 40));
        pnlEquipmentUsageReport.setLayout(new
org.netbeans.lib.awtextra.AbsoluteLayout());

        lbllTitleUsage.setFont(new java.awt.Font("Arial", 1, 14)); // NOI18N
        lbllTitleUsage.setText("Equipment Usage Report");
        lbllTitleUsage.setPreferredSize(new java.awt.Dimension(300, 30));
        pnlEquipmentUsageReport.add(lbllTitleUsage, new
org.netbeans.lib.awtextra.AbsoluteConstraints(10, 5, -1, -1));

        add(pnlEquipmentUsageReport, new
org.netbeans.lib.awtextra.AbsoluteConstraints(0, 0, -1, -1));

        lblDateRange.setFont(new java.awt.Font("Arial", 0, 14)); // NOI18N
        lblDateRange.setText("Date Range:");
        lblDateRange.setPreferredSize(new java.awt.Dimension(80, 25));
        add(lblDateRange, new
org.netbeans.lib.awtextra.AbsoluteConstraints(10, 50, 100, -1));

        lbFromDate.setText("From");
```

```
        lbFromDate.setPreferredSize(new java.awt.Dimension(40, 25));
        add(lbFromDate, new
org.netbeans.lib.awtextra.AbsoluteConstraints(100, 50, -1, -1));


        lblToDate.setText("To:");
        add(lblToDate, new org.netbeans.lib.awtextra.AbsoluteConstraints(250,
50, -1, -1));


        dateStart.setPreferredSize(new java.awt.Dimension(100, 25));
        add(dateStart, new org.netbeans.lib.awtextra.AbsoluteConstraints(140,
50, -1, -1));


        dateEnd.setPreferredSize(new java.awt.Dimension(100, 25));
        add(dateEnd, new org.netbeans.lib.awtextra.AbsoluteConstraints(280,
50, -1, -1));


        btnGenerate.setText("Generate");
        btnGenerate.setPreferredSize(new java.awt.Dimension(100, 25));
        btnGenerate.addActionListener(new java.awt.event.ActionListener()
        {
            public void actionPerformed(java.awt.event.ActionEvent evt)
            {
                btnGenerateActionPerformed(evt);
            }
        });
        add(btnGenerate, new
org.netbeans.lib.awtextra.AbsoluteConstraints(300, 85, -1, -1));
```

```
        comboEquipmentType.setModel(new
javax.swing.DefaultComboBoxModel<>(new String[] { "check", "database",
"for ", "types" }));

        comboEquipmentType.setPreferredSize(new java.awt.Dimension(150,
25));

        add(comboEquipmentType, new
org.netbeans.lib.awtextra.AbsoluteConstraints(120, 85, -1, -1));


        lblComboEquipmentType.setText("Equipment Type:");

        lblComboEquipmentType.setPreferredSize(new java.awt.Dimension(100,
25));

        add(lblComboEquipmentType, new
org.netbeans.lib.awtextra.AbsoluteConstraints(10, 85, 110, -1));


        jScrollPane1.setPreferredSize(new java.awt.Dimension(580, 350));


        tblEquipmentUsage.setModel(new javax.swing.table.DefaultTableModel(
            new Object [][]
            {
                {null, null, null, null},
                {null, null, null, null},
                {null, null, null, null},
                {null, null, null, null}
            },
            new String []
            {
                "Equipment ID", "Equipment Type", "Usage", "Current Status"
            }
        ));
```

```
        jScrollPane1.setViewportView(tblEquipmentUsage);


        add(jScrollPane1, new
org.netbeans.lib.awtextra.AbsoluteConstraints(35, 150, 500, 250));


        btnExportUserReport.setText("Export");
        btnExportUserReport.setPreferredSize(new java.awt.Dimension(100,
39));
        add(btnExportUserReport, new
org.netbeans.lib.awtextra.AbsoluteConstraints(210, 410, -1, -1));
    }// </editor-fold>//GEN-END:initComponents


    private void btnGenerateActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_btnGenerateActionPerformed
        if (dateStart.getDate() == null || dateEnd.getDate() == null) {
            JOptionPane.showMessageDialog(this,
                "Please select both start and end dates",
                "Date Range Required",
                JOptionPane.WARNING_MESSAGE);
            return;
        }


        String selectedType = (String) comboEquipmentType.getSelectedItem();
        StringBuilder sql = new StringBuilder(
            "SELECT t.*, e.equipment_type, e.equipment_code " +
            "FROM transaction t " +
            "JOIN equipment e ON t.equipment_id = e.equipment_id " +
            "WHERE t.transaction_date BETWEEN ? AND ? ");
```

```
        if (!"All Types".equals(selectedType)) {
            sql.append("AND e.equipment_type = ? ");
        }

        sql.append("ORDER BY t.transaction_date DESC");

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql.toString())) {

            stmt.setTimestamp(1, new
Timestamp(dateStart.getDate().getTime()));
            stmt.setTimestamp(2, new
Timestamp(dateEnd.getDate().getTime()));

            if (!"All Types".equals(selectedType)) {
                stmt.setString(3, selectedType);
            }

            tableModel.setRowCount(0); // Clear existing data

            try (ResultSet rs = stmt.executeQuery()) {
                while (rs.next()) {
                    String equipmentId = rs.getString("equipment_code");
                    String equipmentType = rs.getString("equipment_type");
                    Date transactionDate = rs.getDate("transaction_date");
                    Date returnDate = rs.getDate("return_date");
```

```
                // Calculate usage duration
                String usageDuration = calculateDuration(transactionDate,
returnDate);

                // Get current status
                String currentStatus =
getCurrentStatus(rs.getInt("equipment_id"));

                tableModel.addRow(new Object[]{
                    equipmentId,
                    equipmentType,
                    usageDuration,
                    currentStatus
                });
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
        JOptionPane.showMessageDialog(this,
            "Error filtering transactions: " + e.getMessage(),
            "Database Error",
            JOptionPane.ERROR_MESSAGE);
    }
}//GEN-LAST:event_btnGenerateActionPerformed

private void loadLast100Transactions() {
```

```
        String sql = "SELECT t.*, e.equipment_type, e.equipment_code " +
                     "FROM transaction t " +
                     "JOIN equipment e ON t.equipment_id = e.equipment_id " +
                     "ORDER BY t.transaction_date DESC LIMIT 100";

        try (Connection conn = DatabaseConnection.getConnection();
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(sql)) {

            tableModel.setRowCount(0); // Clear existing data

            while (rs.next()) {
                String equipmentId = rs.getString("equipment_code");
                String equipmentType = rs.getString("equipment_type");
                Date transactionDate = rs.getDate("transaction_date");
                Date returnDate = rs.getDate("return_date");

                // Calculate usage duration
                String usageDuration = calculateDuration(transactionDate,
returnDate);

                // Get current status
                String currentStatus =
getCurrentStatus(rs.getInt("equipment_id"));

                tableModel.addRow(new Object[]{
                    equipmentId,
```

```
                equipmentType,
                usageDuration,
                currentStatus
            });
        }
    } catch (SQLException e) {
        e.printStackTrace();
        JOptionPane.showMessageDialog(this,
            "Error loading transactions: " + e.getMessage(),
            "Database Error",
            JOptionPane.ERROR_MESSAGE);
    }
}

private void populateEquipmentTypes() {
    String sql = "SELECT DISTINCT equipment_type FROM equipment
ORDER BY equipment_type";

    try (Connection conn = DatabaseConnection.getConnection();
         Statement stmt = conn.createStatement();
         ResultSet rs = stmt.executeQuery(sql)) {

        comboEquipmentType.removeAllItems();
        comboEquipmentType.addItem("All Types");

        while (rs.next()) {
            comboEquipmentType.addItem(rs.getString("equipment_type"));
```

```
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    private String calculateDuration(Date startDate, Date returnDate) {
        if (returnDate == null) {
            return "In Use";
        }

        long diffInMillies = returnDate.getTime() - startDate.getTime();
        long diffInHours = diffInMillies / (60 * 60 * 1000);

        if (diffInHours < 24) {
            return diffInHours + " hours";
        } else {
            return (diffInHours / 24) + " days";
        }
    }

    private String getCurrentStatus(int equipmentId) {
        try {
            Equipment equipment =
equipmentDAO.getEquipmentById(equipmentId);
            if (equipment != null) {
                return equipment.getStatus().getStatusName();
```

```
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return "Unknown";
    }

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JButton btnExportUserReport;
    private javax.swing.JButton btnGenerate;
    private javax.swing.JComboBox<String> comboEquipmentType;
    private com.toedter.calendar.JDateChooser dateEnd;
    private com.toedter.calendar.JDateChooser dateStart;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JLabel lbFromDate;
    private javax.swing.JLabel lblComboEquipmentType;
    private javax.swing.JLabel lblDateRange;
    private javax.swing.JLabel lblToDate;
    private javax.swing.JLabel lbllTitleUsage;
    private javax.swing.JPanel pnlEquipmentUsageReport;
    private javax.swing.JTable tblEquipmentUsage;
    // End of variables declaration//GEN-END:variables
}
```

GenerateReportsWindow.java

```
package ecsapplication.UserInterface;
```

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/GUIForms/JFrame.java to edit this template
 */

/**
 *
 * @author Ughon
 */
public class GenerateReportsWindow extends javax.swing.JFrame
{

    /**
     * Creates new form GenerateReportsWindow
     */
    public GenerateReportsWindow()
    {
        initComponents();
        setLocationRelativeTo(null); // center the window
        setDefaultCloseOperation(DISPOSE_ON_CLOSE);

        // Load inventory report by default
        try {
            System.out.println("Loading default inventory report...");
            InventoryReportPanel reportPanel = new InventoryReportPanel();
```

```
            loadReportPanel(reportPanel);
            System.out.println("Default inventory report loaded");
        } catch (Exception e) {
            System.err.println("Error loading default inventory report: " +
e.getMessage());
            e.printStackTrace();
        }
    }


    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-
BEGIN:initComponents
    private void initComponents()
    {


        pnlHeader = new javax.swing.JPanel();
        lblTitleHeader = new javax.swing.JLabel();
        pnlActionBar = new javax.swing.JPanel();
        btnInventoryValueLossReport = new javax.swing.JButton();
        btnInventoryReportFrame = new javax.swing.JButton();
        btnEquipmentUsageReportFrame = new javax.swing.JButton();
        btnUserActivityReport = new javax.swing.JButton();
```

```
        btnOverDueEquipmentFrame1 = new javax.swing.JButton();
        pnlReportFrame = new javax.swing.JPanel();

setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE
);
        setTitle("ECS-Reports");
        setName("ECS Reports"); // NOI18N
        setPreferredSize(new java.awt.Dimension(800, 600));
        getContentPane().setLayout(new
org.netbeans.lib.awtextra.AbsoluteLayout());

        pnlHeader.setBackground(new java.awt.Color(0, 51, 255));
        pnlHeader.setPreferredSize(new java.awt.Dimension(800, 60));
        pnlHeader.setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

        lblTitleHeader.setBackground(new java.awt.Color(0, 51, 255));
        lblTitleHeader.setFont(new java.awt.Font("Arial", 1, 18)); // NOI18N
        lblTitleHeader.setForeground(new java.awt.Color(255, 255, 255));
        lblTitleHeader.setText("ECS Reports");
        lblTitleHeader.setPreferredSize(new java.awt.Dimension(200, 30));
        pnlHeader.add(lblTitleHeader, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 15, -1, -1));

        getContentPane().add(pnlHeader, new
org.netbeans.lib.awtextra.AbsoluteConstraints(0, 0, -1, -1));

        pnlActionBar.setBackground(new java.awt.Color(204, 204, 204));
```

```
        pnlActionBar.setPreferredSize(new java.awt.Dimension(200, 540));

        pnlActionBar.setLayout(new
org.netbeans.lib.awtextra.AbsoluteLayout());



        btnInventoryValueLossReport.setFont(new java.awt.Font("Arial", 0,
12)); // NOI18N

        btnInventoryValueLossReport.setText("Inventory Value/Loss");

        btnInventoryValueLossReport.setPreferredSize(new
java.awt.Dimension(180, 30));

        btnInventoryValueLossReport.addActionListener(new
java.awt.event.ActionListener()

        {

            public void actionPerformed(java.awt.event.ActionEvent evt)

            {

                btnInventoryValueLossReportActionPerformed(evt);

            }

        });

        pnlActionBar.add(btnInventoryValueLossReport, new
org.netbeans.lib.awtextra.AbsoluteConstraints(10, 180, -1, -1));



        btnInventoryReportFrame.setFont(new java.awt.Font("Arial", 0, 12)); //
NOI18N

        btnInventoryReportFrame.setText("Inventory Report");

        btnInventoryReportFrame.setPreferredSize(new
java.awt.Dimension(180, 30));

        btnInventoryReportFrame.addActionListener(new
java.awt.event.ActionListener()

        {

            public void actionPerformed(java.awt.event.ActionEvent evt)
```

```
            {
                btnInventoryReportFrameActionPerformed(evt);
            }
        });
        pnlActionBar.add(btnInventoryReportFrame, new
org.netbeans.lib.awtextra.AbsoluteConstraints(10, 20, -1, -1));

        btnEquipmentUsageReportFrame.setFont(new java.awt.Font("Arial", 0,
12)); // NOI18N
        btnEquipmentUsageReportFrame.setText("Equipment Usage Report");
        btnEquipmentUsageReportFrame.setPreferredSize(new
java.awt.Dimension(180, 30));
        btnEquipmentUsageReportFrame.addActionListener(new
java.awt.event.ActionListener()
        {
            public void actionPerformed(java.awt.event.ActionEvent evt)
            {
                btnEquipmentUsageReportFrameActionPerformed(evt);
            }
        });
        pnlActionBar.add(btnEquipmentUsageReportFrame, new
org.netbeans.lib.awtextra.AbsoluteConstraints(10, 100, -1, -1));

        btnUserActivityReport.setFont(new java.awt.Font("Arial", 0, 12)); //
NOI18N
        btnUserActivityReport.setText("Transaction Report");
        btnUserActivityReport.setPreferredSize(new java.awt.Dimension(180,
30));
```

```
        btnUserActivityReport.addActionListener(new
java.awt.event.ActionListener()
        {
            public void actionPerformed(java.awt.event.ActionEvent evt)
            {
                btnUserActivityReportActionPerformed(evt);
            }
        });
        pnlActionBar.add(btnUserActivityReport, new
org.netbeans.lib.awtextra.AbsoluteConstraints(10, 60, -1, -1));


        btnOverDueEquipmentFrame1.setFont(new java.awt.Font("Arial", 0,
12)); // NOI18N
        btnOverDueEquipmentFrame1.setText("Overdue Equipment Report");
        btnOverDueEquipmentFrame1.setPreferredSize(new
java.awt.Dimension(180, 30));
        btnOverDueEquipmentFrame1.addActionListener(new
java.awt.event.ActionListener()
        {
            public void actionPerformed(java.awt.event.ActionEvent evt)
            {
                btnOverDueEquipmentFrame1ActionPerformed(evt);
            }
        });
        pnlActionBar.add(btnOverDueEquipmentFrame1, new
org.netbeans.lib.awtextra.AbsoluteConstraints(10, 140, -1, -1));


        getContentPane().add(pnlActionBar, new
org.netbeans.lib.awtextra.AbsoluteConstraints(0, 60, -1, -1));
```

```
        pnlReportFrame.setName("pnlReportFrame"); // NOI18N
        pnlReportFrame.setPreferredSize(new java.awt.Dimension(600, 550));
        pnlReportFrame.setLayout(new
org.netbeans.lib.awtextra.AbsoluteLayout());
        getContentPane().add(pnlReportFrame, new
org.netbeans.lib.awtextra.AbsoluteConstraints(200, 60, 600, -1));


        pack();
    }// </editor-fold>//GEN-END:initComponents


    private void
btnEquipmentUsageReportFrameActionPerformed(java.awt.event.ActionEven
t evt)//GEN-FIRST:event_btnEquipmentUsageReportFrameActionPerformed
    {//GEN-
HEADEREND:event_btnEquipmentUsageReportFrameActionPerformed
       try {
        System.out.println("Creating equipment usage panel...");
        EquipmentUsageReportPanel reportPanel = new
EquipmentUsageReportPanel();
        loadReportPanel(reportPanel);
        System.out.println("Equipment usage panel should now be displayed");
    } catch (Exception e) {
        System.err.println("Error displaying equipment usage panel: " +
e.getMessage());
        e.printStackTrace();
    }
    }//GEN-LAST:event_btnEquipmentUsageReportFrameActionPerformed
```

```
    private void
btnInventoryValueLossReportActionPerformed(java.awt.event.ActionEvent
evt)//GEN-FIRST:event_btnInventoryValueLossReportActionPerformed

    {//GEN-HEADEREND:event_btnInventoryValueLossReportActionPerformed

        try {

            System.out.println("DEBUG: Starting to create value analysis report
panel...");

            System.out.println("DEBUG: Instantiating
ValueAnalysisReportPanel...");

            ValueAnalysisReportPanel reportPanel = new
ValueAnalysisReportPanel();

            System.out.println("DEBUG: Panel instantiated successfully");


            System.out.println("DEBUG: Calling loadReportPanel...");

            loadReportPanel(reportPanel);

            System.out.println("DEBUG: loadReportPanel completed");


            System.out.println("DEBUG: Value analysis report panel loaded
successfully");

        } catch (Exception e) {

            System.err.println("ERROR: Failed to display value analysis report
panel");

            System.err.println("ERROR: Exception type: " +
e.getClass().getName());

            System.err.println("ERROR: Message: " + e.getMessage());

            e.printStackTrace();

        }

    }//GEN-LAST:event_btnInventoryValueLossReportActionPerformed
```

```
    private void
btnUserActivityReportActionPerformed(java.awt.event.ActionEvent evt)//GEN-
FIRST:event_btnUserActivityReportActionPerformed

    {//GEN-HEADEREND:event_btnUserActivityReportActionPerformed

        try {

            System.out.println("Creating transaction report panel...");

            TransactionReport reportPanel = new TransactionReport();

            // Special handling for TransactionReport since it's a different type

            pnlReportFrame.removeAll();

            pnlReportFrame.add(reportPanel.getContentPane(), new
org.netbeans.lib.awtextra.AbsoluteConstraints(0, 0, 600, 500));

            pnlReportFrame.revalidate();

            pnlReportFrame.repaint();

            System.out.println("Transaction report panel should now be
displayed");

        } catch (Exception e) {

            System.err.println("Error displaying transaction report panel: " +
e.getMessage());

            e.printStackTrace();

        }

    }//GEN-LAST:event_btnUserActivityReportActionPerformed


    private void
btnInventoryReportFrameActionPerformed(java.awt.event.ActionEvent
evt)//GEN-FIRST:event_btnInventoryReportFrameActionPerformed

    {//GEN-HEADEREND:event_btnInventoryReportFrameActionPerformed

        try {

            System.out.println("Creating inventory panel...");

            InventoryReportPanel reportPanel = new InventoryReportPanel();
```

```
            loadReportPanel(reportPanel);
            System.out.println("Inventory panel should now be displayed");
        } catch (Exception e) {
            System.err.println("Error displaying inventory panel: " +
e.getMessage());
            e.printStackTrace();
        }
    }//GEN-LAST:event_btnInventoryReportFrameActionPerformed


    private void
btnOverDueEquipmentFrame1ActionPerformed(java.awt.event.ActionEvent
evt)//GEN-FIRST:event_btnOverDueEquipmentFrame1ActionPerformed
    {//GEN-HEADEREND:event_btnOverDueEquipmentFrame1ActionPerformed
        // TODO add your handling code here:
    }//GEN-LAST:event_btnOverDueEquipmentFrame1ActionPerformed


    // Helper method to load report panels
    private void loadReportPanel(javax.swing.JPanel panel) {
        pnlReportFrame.removeAll();
        pnlReportFrame.add(panel, new
org.netbeans.lib.awtextra.AbsoluteConstraints(0, 0, 600, 500));
        panel.setVisible(true);
        pnlReportFrame.revalidate();
        pnlReportFrame.repaint();
    }


    /**
     * @param args the command line arguments
```

```
     */
    public static void main(String args[])
    {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
         * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try
        {
            for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels())
            {
                if ("Nimbus".equals(info.getName()))
                {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex)
        {

java.util.logging.Logger.getLogger(GenerateReportsWindow.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (InstantiationException ex)
        {
```

```
java.util.logging.Logger.getLogger(GenerateReportsWindow.class.getName())
.log(java.util.logging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex)
        {

java.util.logging.Logger.getLogger(GenerateReportsWindow.class.getName())
.log(java.util.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex)
        {

java.util.logging.Logger.getLogger(GenerateReportsWindow.class.getName())
.log(java.util.logging.Level.SEVERE, null, ex);
        }
        //</editor-fold>

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

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JButton btnEquipmentUsageReportFrame;
    private javax.swing.JButton btnInventoryReportFrame;
```

```
    private javax.swing.JButton btnInventoryValueLossReport;
    private javax.swing.JButton btnOverDueEquipmentFrame1;
    private javax.swing.JButton btnUserActivityReport;
    private javax.swing.JLabel lblTitleHeader;
    private javax.swing.JPanel pnlActionBar;
    private javax.swing.JPanel pnlHeader;
    private javax.swing.JPanel pnlReportFrame;
    // End of variables declaration//GEN-END:variables

}
```

HubWindow.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/GUIForms/JFrame.java to edit this template
 */
package ecsapplication.UserInterface;

import ecsapplication.Model.Employee;
import javax.swing.JOptionPane;

/**
```

```
 *
 * @author Redux
 */
public class HubWindow extends javax.swing.JFrame
{
    private Employee loggedInEmployee;
    /**
     * Creates new form HubWindow
     */
    public HubWindow()
    {
        initComponents();
        setLocationRelativeTo(null);
    }

        /**
     * Creates new form with logged in employee
     */
    public HubWindow(Employee employee) {
        initComponents();
        setLocationRelativeTo(null);

        this.loggedInEmployee = employee;

        // Set window title to include employee name
        this.setTitle("Equipment Checkout System - " +
employee.getFirstName() + " " + employee.getLastName());
```

```
        // Configure role-based access
        configureAccessControl();
    }

    /**
     * Configure UI elements based on employee role
     */
    private void configureAccessControl() {
        if (loggedInEmployee != null) {
            // Get role ID
            int roleId = loggedInEmployee.getRole().getRoleId();

            // Role ID 1 = Supervisor, Role ID 2 = DepotStaff, Role ID 3 =
Maintenance
            // Only supervisors can access employee termination
            btnTermination.setEnabled(roleId == 1);

            // For demonstration purposes, add a small visual indicator
            if (roleId == 1) {
                lblECSHub.setText("ECS Hub - Supervisor");
            } else if (roleId == 2) {
                lblECSHub.setText("ECS Hub - Depot Staff");
            } else if (roleId == 3) {
                lblECSHub.setText("ECS Hub - Maintenance");
            }
        }
```

```
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents
    private void initComponents()
    {

        jPanel1 = new javax.swing.JPanel();
        lblECSHub = new javax.swing.JLabel();
        btnCheckInOut = new javax.swing.JButton();
        btnUserManagement = new javax.swing.JButton();
        btnEquipmentManagement = new javax.swing.JButton();
        btnTermination = new javax.swing.JButton();
        btnReports = new javax.swing.JButton();


setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("Equipment Checkout System - Hub");
        setResizable(false);

        lblECSHub.setFont(new java.awt.Font("Segoe UI", 1, 36)); // NOI18N
```

```
lblECSHub.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        lblECSHub.setText("ECS Hub");

        btnCheckInOut.setText("Check In / Out");
        btnCheckInOut.addActionListener(new java.awt.event.ActionListener()
        {
            public void actionPerformed(java.awt.event.ActionEvent evt)
            {
                btnCheckInOutActionPerformed(evt);
            }
        });

        btnUserManagement.setText("User Management");
        btnUserManagement.addActionListener(new
java.awt.event.ActionListener()
        {
            public void actionPerformed(java.awt.event.ActionEvent evt)
            {
                btnUserManagementActionPerformed(evt);
            }
        });

        btnEquipmentManagement.setText("Equipment Management");
        btnEquipmentManagement.addActionListener(new
java.awt.event.ActionListener()
        {
            public void actionPerformed(java.awt.event.ActionEvent evt)
```

```
            {
                btnEquipmentManagementActionPerformed(evt);
            }
        });

        btnTermination.setText("Employee Termination");
        btnTermination.addActionListener(new java.awt.event.ActionListener()
        {
            public void actionPerformed(java.awt.event.ActionEvent evt)
            {
                btnTerminationActionPerformed(evt);
            }
        });

        btnReports.setText("Generate Reports");
        btnReports.addActionListener(new java.awt.event.ActionListener()
        {
            public void actionPerformed(java.awt.event.ActionEvent evt)
            {
                btnReportsActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout jPanel1Layout = new
javax.swing.GroupLayout(jPanel1);
        jPanel1.setLayout(jPanel1Layout);
        jPanel1Layout.setHorizontalGroup(
```

```
jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
            .addComponent(lblECSHub, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addGap(77, 77, 77)
                .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLa
yout.Alignment.LEADING)
                    .addComponent(btnUserManagement,
javax.swing.GroupLayout.PREFERRED_SIZE, 197,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(btnCheckInOut,
javax.swing.GroupLayout.PREFERRED_SIZE, 197,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.R
ELATED, 127, Short.MAX_VALUE)
                .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLa
yout.Alignment.LEADING)
                    .addComponent(btnEquipmentManagement,
javax.swing.GroupLayout.PREFERRED_SIZE, 197,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(btnTermination,
javax.swing.GroupLayout.PREFERRED_SIZE, 197,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(77, 77, 77))
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addGap(239, 239, 239)
                .addComponent(btnReports,
javax.swing.GroupLayout.PREFERRED_SIZE, 197,
javax.swing.GroupLayout.PREFERRED_SIZE)
```

```
            .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
        );
        jPanel1Layout.setVerticalGroup(

jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addComponent(lblECSHub,
javax.swing.GroupLayout.PREFERRED_SIZE, 55,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.R
ELATED, 47, Short.MAX_VALUE)
                .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLa
yout.Alignment.BASELINE)
                    .addComponent(btnUserManagement,
javax.swing.GroupLayout.PREFERRED_SIZE, 44,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(btnEquipmentManagement,
javax.swing.GroupLayout.PREFERRED_SIZE, 44,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(42, 42, 42)
                .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLa
yout.Alignment.BASELINE)
                    .addComponent(btnCheckInOut,
javax.swing.GroupLayout.PREFERRED_SIZE, 44,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(btnTermination,
javax.swing.GroupLayout.PREFERRED_SIZE, 44,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(42, 42, 42)
```

```
                .addComponent(btnReports,
javax.swing.GroupLayout.PREFERRED_SIZE, 44,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(32, 32, 32))
        );

        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
        );
        layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
        );

        pack();
    }// </editor-fold>//GEN-END:initComponents

    private void
btnUserManagementActionPerformed(java.awt.event.ActionEvent evt)//GEN-
FIRST:event_btnUserManagementActionPerformed
    {//GEN-HEADEREND:event_btnUserManagementActionPerformed
        UserManagementWindow userWin = new UserManagementWindow();
```

```
        userWin.setLocationRelativeTo(null);
        userWin.setVisible(true);
    }//GEN-LAST:event_btnUserManagementActionPerformed

    private void
btnEquipmentManagementActionPerformed(java.awt.event.ActionEvent
evt)//GEN-FIRST:event_btnEquipmentManagementActionPerformed
    {//GEN-HEADEREND:event_btnEquipmentManagementActionPerformed
        EquipmentManagementWindow equipWin = new
EquipmentManagementWindow();
        equipWin.setLocationRelativeTo(null);
        equipWin.setVisible(true);
    }//GEN-LAST:event_btnEquipmentManagementActionPerformed

    private void btnCheckInOutActionPerformed(java.awt.event.ActionEvent
evt)//GEN-FIRST:event_btnCheckInOutActionPerformed
    {//GEN-HEADEREND:event_btnCheckInOutActionPerformed
        MainWindow mainWin = new MainWindow();
        mainWin.setLocationRelativeTo(null);
        mainWin.setVisible(true);
    }//GEN-LAST:event_btnCheckInOutActionPerformed

    private void btnTerminationActionPerformed(java.awt.event.ActionEvent
evt)//GEN-FIRST:event_btnTerminationActionPerformed
    {//GEN-HEADEREND:event_btnTerminationActionPerformed
        // Check if user is supervisor
        if (loggedInEmployee != null &&
loggedInEmployee.getRole().getRoleId() == 1) {
```

```
            EmployeeTerminationWindow termWin = new
EmployeeTerminationWindow();
            termWin.setLocationRelativeTo(null);
            termWin.setVisible(true);
        } else {
            // Not a supervisor
            JOptionPane.showMessageDialog(
                this,
                "Only supervisors can access the employee termination function.",
                "Access Denied",
                JOptionPane.WARNING_MESSAGE
            );
        }
    }//GEN-LAST:event_btnTerminationActionPerformed


    private void btnReportsActionPerformed(java.awt.event.ActionEvent
evt)//GEN-FIRST:event_btnReportsActionPerformed
    {//GEN-HEADEREND:event_btnReportsActionPerformed
        GenerateReportsWindow reportWin = new GenerateReportsWindow();
        reportWin.setLocationRelativeTo(null);
        reportWin.setVisible(true);
    }//GEN-LAST:event_btnReportsActionPerformed


    /**
     * @param args the command line arguments
     */
    public static void main(String args[])
```

```
    {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
         * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try
        {
            for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels())
            {
                if ("Nimbus".equals(info.getName()))
                {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex)
        {
            java.util.logging.Logger.getLogger(HubWindow.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (InstantiationException ex)
        {
```

```
java.util.logging.Logger.getLogger(HubWindow.class.getName()).log(java.util.
logging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex)
        {

java.util.logging.Logger.getLogger(HubWindow.class.getName()).log(java.util.
logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex)
        {

java.util.logging.Logger.getLogger(HubWindow.class.getName()).log(java.util.
logging.Level.SEVERE, null, ex);
        }
        //</editor-fold>

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

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JButton btnCheckInOut;
    private javax.swing.JButton btnEquipmentManagement;
```

```
    private javax.swing.JButton btnReports;
    private javax.swing.JButton btnTermination;
    private javax.swing.JButton btnUserManagement;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JLabel lblECSHub;
    // End of variables declaration//GEN-END:variables
}
```

InventoryReportPanel.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/GUIForms/JPanel.java to edit this template
 */
```

```
package ecsapplication.UserInterface;

import ecsapplication.DataAccess.DBManager;
import ecsapplication.DataAccess.EquipmentDAO;
import ecsapplication.Model.Equipment;
import java.util.List;
import javax.swing.table.DefaultTableModel;
import javax.swing.*;
import java.awt.*;

/**
 *
 * @author Ughon
 */
public class InventoryReportPanel extends javax.swing.JPanel
{
    private final EquipmentDAO equipmentDAO;
    /**
     * Creates new form InventoryReportPanel
     */
    public InventoryReportPanel()
    {
        initComponents();

        // Initialize DAO
        this.equipmentDAO =
DBManager.getInstance().getDAO(EquipmentDAO.class);
```

```
        // Load equipment data when panel is created
        loadEquipmentData();
    }
    /**
 * Loads equipment data from database into the table
 */
private void loadEquipmentData() {
    try {
        System.out.println("Loading equipment data...");

        // Get all equipment without filters
        List<Equipment> equipmentList = equipmentDAO.getAllEquipment();

        System.out.println("Retrieved " + equipmentList.size() + " equipment
records");

        // Get the table model
        DefaultTableModel model = (DefaultTableModel)
tblEquipmentList.getModel();

        // Clear existing data
        model.setRowCount(0);

        // Add each equipment item to the table
        for (Equipment equipment : equipmentList) {
            Object[] row = new Object[6];
```

```
            row[0] = equipment.getEquipmentId();
            row[1] = equipment.getEquipmentType();
            row[2] = equipment.getSkill() != null ?
equipment.getSkill().getSkillLevel() : "N/A";
            row[3] = equipment.getStatus() != null ?
equipment.getStatus().getStatusName() : "Unknown";
            row[4] = equipment.getEmployee() != null ?
                equipment.getEmployee().getFirstName() + " " +
equipment.getEmployee().getLastName() :
                "None";
            row[5] = "";

            model.addRow(row);
        }

        System.out.println("Equipment data loaded successfully");

    } catch (Exception e) {
        System.err.println("Error loading equipment data: " + e.getMessage());
        e.printStackTrace();
        javax.swing.JOptionPane.showMessageDialog(this,
            "Error loading equipment data: " + e.getMessage(),
            "Data Loading Error",
            javax.swing.JOptionPane.ERROR_MESSAGE);
    }
}

    /**
```

```
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents
    private void initComponents()
    {

        inventoryReportPane = new javax.swing.JScrollPane();
        tblEquipmentList = new javax.swing.JTable();
        lbInventoryReoportlTitle = new javax.swing.JLabel();
        lblSearchFilter = new javax.swing.JLabel();
        btnSearchInventory = new javax.swing.JButton();
        btnExportUserReport = new javax.swing.JButton();
        cmbEquipType = new javax.swing.JComboBox<>();
        cmbEquipLocation = new javax.swing.JComboBox<>();
        lblEquipLocation = new javax.swing.JLabel();
        lblEquipStatus = new javax.swing.JLabel();
        cmbEquipCondition = new javax.swing.JComboBox<>();
        lblEquipAvailability = new javax.swing.JLabel();
        cmbEquipAvailability = new javax.swing.JComboBox<>();
        btnSearchInventory1 = new javax.swing.JButton();

        setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());
```

```
        inventoryReportPane.setPreferredSize(new java.awt.Dimension(300,
150));


        tblEquipmentList.setFont(new java.awt.Font("Segoe UI", 0, 12)); //
NOI18N
        tblEquipmentList.setModel(new javax.swing.table.DefaultTableModel(
            new Object [][]
            {
                {null, null, null, null, null, null},
                {null, null, null, null, null, null},
                {null, null, null, null, null, null},
                {null, null, null, null, null, null},
                {null, null, null, null, null, null},
                {null, null, null, null, null, null},
                {null, null, null, null, null, null},
                {null, null, null, null, null, null},
                {null, null, null, null, null, null},
                {null, null, null, null, null, null},
                {null, null, null, null, null, null},
                {null, null, null, null, null, null},
                {null, null, null, null, null, null},
                {null, null, null, null, null, null},
                {null, null, null, null, null, null},
                {null, null, null, null, null, null},
                {null, null, null, null, null, null},
                {null, null, null, null, null, null},
```

```
                {null, null, null, null, null, null},
                {null, null, null, null, null, null},
                {null, null, null, null, null, null},
                {null, null, null, null, null, null},
                {null, null, null, null, null, null},
                {null, null, null, null, null, null},
                {null, null, null, null, null, null},
                {null, null, null, null, null, null},
                {null, null, null, null, null, null},
                {null, null, null, null, null, null}
            },
            new String []
            {
                "Equipment ID", "Type", "Category", "Status", "Holder", "Actions"
            }
        ));
        inventoryReportPane.setViewportView(tblEquipmentList);


        add(inventoryReportPane, new
org.netbeans.lib.awtextra.AbsoluteConstraints(35, 150, 500, 250));


        lbInventoryReoportlTitle.setFont(new java.awt.Font("Segoe UI", 1,
18)); // NOI18N
        lbInventoryReoportlTitle.setText("Inventory Report");
        add(lbInventoryReoportlTitle, new
org.netbeans.lib.awtextra.AbsoluteConstraints(220, 0, -1, -1));


        lblSearchFilter.setText("Equipment Type:");
```

```
        add(lblSearchFilter, new
org.netbeans.lib.awtextra.AbsoluteConstraints(40, 30, -1, -1));


        btnSearchInventory.setText("Search");

        btnSearchInventory.addActionListener(new
java.awt.event.ActionListener()

        {

            public void actionPerformed(java.awt.event.ActionEvent evt)

            {

                btnSearchInventoryActionPerformed(evt);

            }

        });

        add(btnSearchInventory, new
org.netbeans.lib.awtextra.AbsoluteConstraints(460, 70, -1, -1));


        btnExportUserReport.setText("Export");

        btnExportUserReport.setPreferredSize(new java.awt.Dimension(100,
39));

        add(btnExportUserReport, new
org.netbeans.lib.awtextra.AbsoluteConstraints(210, 410, -1, -1));


        cmbEquipType.setModel(new
javax.swing.DefaultComboBoxModel<>(new String[] { "Hammer", "Cordelss
Drill", "Circular Saw", "Ladder (6ft)", "Wrench Set", "Power Washer", "Forklift",
"Pallet Jack", "Pressure Gauge", "Voltage Tester" }));

        cmbEquipType.setPreferredSize(new java.awt.Dimension(102, 20));

        add(cmbEquipType, new
org.netbeans.lib.awtextra.AbsoluteConstraints(150, 30, 200, -1));
```

```
        cmbEquipLocation.setModel(new
javax.swing.DefaultComboBoxModel<>(new String[] { "1 North Warehouse",
"2 South Warehouse " }));

        cmbEquipLocation.setMinimumSize(new java.awt.Dimension(5, 5));

        cmbEquipLocation.setPreferredSize(new java.awt.Dimension(125, 22));

        cmbEquipLocation.addActionListener(new
java.awt.event.ActionListener()

        {

            public void actionPerformed(java.awt.event.ActionEvent evt)

            {

                cmbEquipLocationActionPerformed(evt);

            }

        });

        add(cmbEquipLocation, new
org.netbeans.lib.awtextra.AbsoluteConstraints(150, 60, 200, -1));



        lblEquipLocation.setText("Location:");

        add(lblEquipLocation, new
org.netbeans.lib.awtextra.AbsoluteConstraints(40, 60, -1, -1));



lblEquipStatus.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);

        lblEquipStatus.setText("Condition:");

        add(lblEquipStatus, new
org.netbeans.lib.awtextra.AbsoluteConstraints(40, 90, 60, -1));



        cmbEquipCondition.setModel(new
javax.swing.DefaultComboBoxModel<>(new String[] { "Functional",
"Damaged", "NeedsRepair", "InRepair" }));

        cmbEquipCondition.setPreferredSize(new java.awt.Dimension(65, 26));
```

```
        add(cmbEquipCondition, new
org.netbeans.lib.awtextra.AbsoluteConstraints(150, 90, 200, 20));



lblEquipAvailability.setHorizontalAlignment(javax.swing.SwingConstants.CEN
TER);

        lblEquipAvailability.setText("Availability:");

        add(lblEquipAvailability, new
org.netbeans.lib.awtextra.AbsoluteConstraints(40, 120, -1, -1));



        cmbEquipAvailability.setModel(new
javax.swing.DefaultComboBoxModel<>(new String[] { "Available",
"CheckedOut", "Unavailable" }));

        cmbEquipAvailability.setPreferredSize(new java.awt.Dimension(65, 26));

        add(cmbEquipAvailability, new
org.netbeans.lib.awtextra.AbsoluteConstraints(150, 120, 200, 20));



        btnSearchInventory1.setText("Clear");

        btnSearchInventory1.addActionListener(new
java.awt.event.ActionListener()

        {

            public void actionPerformed(java.awt.event.ActionEvent evt)

            {

                btnSearchInventory1ActionPerformed(evt);

            }

        });

        add(btnSearchInventory1, new
org.netbeans.lib.awtextra.AbsoluteConstraints(460, 100, -1, -1));

    }// </editor-fold>//GEN-END:initComponents
```

```
    private void
btnSearchInventoryActionPerformed(java.awt.event.ActionEvent evt)//GEN-
FIRST:event_btnSearchInventoryActionPerformed
    {//GEN-HEADEREND:event_btnSearchInventoryActionPerformed
        filterEquipment();
    }//GEN-LAST:event_btnSearchInventoryActionPerformed


    private void
cmbEquipLocationActionPerformed(java.awt.event.ActionEvent evt)//GEN-
FIRST:event_cmbEquipLocationActionPerformed
    {//GEN-HEADEREND:event_cmbEquipLocationActionPerformed
       // TODO add your handling code here:
    }//GEN-LAST:event_cmbEquipLocationActionPerformed


    private void
btnSearchInventory1ActionPerformed(java.awt.event.ActionEvent evt)//GEN-
FIRST:event_btnSearchInventory1ActionPerformed
    {//GEN-HEADEREND:event_btnSearchInventory1ActionPerformed
          // Clear filter fields
    cmbEquipType.setSelectedIndex(0);
    cmbEquipLocation.setSelectedIndex(0);
    cmbEquipCondition.setSelectedIndex(0);
    cmbEquipAvailability.setSelectedIndex(0);

    // Reload all equipment data
    loadEquipmentData();
    }//GEN-LAST:event_btnSearchInventory1ActionPerformed
/**
 * Filter equipment based on selected criteria
```

```
 */
private void filterEquipment() {
    try {
        System.out.println("Starting filter operation...");

        // Get selected values from combo boxes
        String selectedType = cmbEquipType.getSelectedItem().toString();
        String selectedLocation =
cmbEquipLocation.getSelectedItem().toString();
        String selectedCondition =
cmbEquipCondition.getSelectedItem().toString();
        String selectedAvailability =
cmbEquipAvailability.getSelectedItem().toString();

        System.out.println("Selected filters:");
        System.out.println("Type: " + selectedType);
        System.out.println("Location: " + selectedLocation);
        System.out.println("Condition: " + selectedCondition);
        System.out.println("Availability: " + selectedAvailability);

        // Convert selections to appropriate IDs
        String typeFilter = "Hammer".equals(selectedType) ? null :
selectedType;

        Integer locationId = null;
        if (!"1 North Warehouse".equals(selectedLocation)) {
            try {
                locationId = Integer.parseInt(selectedLocation.split(" ")[0]);
```

```
            System.out.println("Parsed location ID: " + locationId);
        } catch (Exception e) {
            System.err.println("Error parsing location ID from: " +
selectedLocation);
        }
    }

    Integer statusId = null;
    if (!"Functional".equals(selectedCondition)) {
        statusId = getStatusIdFromName(selectedCondition);
        System.out.println("Mapped status ID: " + statusId + " for condition: "
+ selectedCondition);
    }

    Integer availabilityId = null;
    if (!"Available".equals(selectedAvailability)) {
        availabilityId = getAvailabilityIdFromName(selectedAvailability);
        System.out.println("Mapped availability ID: " + availabilityId + " for
availability: " + selectedAvailability);
    }

    System.out.println("Converted filter values:");
    System.out.println("Type filter: " + typeFilter);
    System.out.println("Location ID: " + locationId);
    System.out.println("Status ID: " + statusId);
    System.out.println("Availability ID: " + availabilityId);

    // Get filtered equipment
```

```
        List<Equipment> equipmentList =
equipmentDAO.getFilteredEquipment(
            typeFilter,
            locationId,
            statusId,
            availabilityId
        );


        System.out.println("Retrieved " + equipmentList.size() + " filtered
equipment records");


        // Update table with filtered results
        DefaultTableModel model = (DefaultTableModel)
tblEquipmentList.getModel();
        model.setRowCount(0);


        for (Equipment equipment : equipmentList) {
            Object[] row = new Object[6];
            row[0] = equipment.getEquipmentId();
            row[1] = equipment.getEquipmentType();
            row[2] = equipment.getSkill() != null ?
equipment.getSkill().getSkillLevel() : "N/A";
            row[3] = equipment.getStatus() != null ?
equipment.getStatus().getStatusName() : "Unknown";
            row[4] = equipment.getEmployee() != null ?
                equipment.getEmployee().getFirstName() + " " +
equipment.getEmployee().getLastName() :
                "None";
            row[5] = "";
```

```
            model.addRow(row);
        }

        System.out.println("Filtered equipment data loaded successfully");

    } catch (Exception e) {
        System.err.println("Error in filterEquipment: " + e.getMessage());
        e.printStackTrace();
        javax.swing.JOptionPane.showMessageDialog(this,
            "Error applying filters: " + e.getMessage(),
            "Filter Error",
            javax.swing.JOptionPane.ERROR_MESSAGE);
    }
}

// Helper method to get status ID from status name
private Integer getStatusIdFromName(String statusName) {
    System.out.println("Getting status ID for: " + statusName);
    switch (statusName) {
        case "Damaged": return 2;
        case "NeedsRepair": return 3;
        case "InRepair": return 4;
        default: return 1; // Functional
    }
}
```

```
// Helper method to get availability ID from availability name
private Integer getAvailabilityIdFromName(String availabilityName) {
    System.out.println("Getting availability ID for: " + availabilityName);
    switch (availabilityName) {
        case "CheckedOut": return 2;
        case "Unavailable": return 3;
        default: return 1; // Available
    }
}

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JButton btnExportUserReport;
    private javax.swing.JButton btnSearchInventory;
    private javax.swing.JButton btnSearchInventory1;
    private javax.swing.JComboBox<String> cmbEquipAvailability;
    private javax.swing.JComboBox<String> cmbEquipCondition;
    private javax.swing.JComboBox<String> cmbEquipLocation;
    private javax.swing.JComboBox<String> cmbEquipType;
    private javax.swing.JScrollPane inventoryReportPane;
    private javax.swing.JLabel lbInventoryReoportlTitle;
    private javax.swing.JLabel lblEquipAvailability;
    private javax.swing.JLabel lblEquipLocation;
    private javax.swing.JLabel lblEquipStatus;
    private javax.swing.JLabel lblSearchFilter;
    private javax.swing.JTable tblEquipmentList;
    // End of variables declaration//GEN-END:variables
}
```

LogInWindow.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/GUIForms/JFrame.java to edit this template
 */
package ecsapplication.UserInterface;
import ecsapplication.Model.Employee;
import ecsapplication.Model.Action;
import ecsapplication.Model.Auditing;
import ecsapplication.DataAccess.EmployeeDAO;
import ecsapplication.DataAccess.ActionDAO;
import ecsapplication.DataAccess.AuditingDAO;
import ecsapplication.DataAccess.DBManager;
import java.time.LocalDateTime;
import javax.swing.JOptionPane;

/**
 *
 * @author Redux
 */
public class LogInWindow extends javax.swing.JFrame
{
```

```
    /**
     * Creates new form LogInWindow
     */
    public LogInWindow()
    {
        initComponents();
        setLocationRelativeTo(null); //Center the window
    }


    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-
BEGIN:initComponents
    private void initComponents()
    {


        jPanel1 = new javax.swing.JPanel();
        jPanel2 = new javax.swing.JPanel();
        lblEquipmentCheckoutSystem = new javax.swing.JLabel();
        lblGBManufacturing = new javax.swing.JLabel();
        lblStaffID = new javax.swing.JLabel();
```

```
        txtStaffID = new javax.swing.JTextField();
        btnCancel = new javax.swing.JButton();
        btnLogin = new javax.swing.JButton();

        javax.swing.GroupLayout jPanel1Layout = new
javax.swing.GroupLayout(jPanel1);
        jPanel1.setLayout(jPanel1Layout);
        jPanel1Layout.setHorizontalGroup(

jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
            .addGap(0, 100, Short.MAX_VALUE)
        );
        jPanel1Layout.setVerticalGroup(

jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
            .addGap(0, 100, Short.MAX_VALUE)
        );


setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("Equipment Checkout System - Login");
        setResizable(false);

        jPanel2.setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

        lblEquipmentCheckoutSystem.setFont(new java.awt.Font("Segoe UI", 1,
18)); // NOI18N
```

```
lblEquipmentCheckoutSystem.setHorizontalAlignment(javax.swing.SwingCon
stants.CENTER);
        lblEquipmentCheckoutSystem.setText("Equipment Checkout System");
        jPanel2.add(lblEquipmentCheckoutSystem, new
org.netbeans.lib.awtextra.AbsoluteConstraints(90, 50, 250, -1));


        lblGBManufacturing.setFont(new java.awt.Font("Segoe UI", 0, 14)); //
NOI18N
        lblGBManufacturing.setText("GB Manufacturing");
        jPanel2.add(lblGBManufacturing, new
org.netbeans.lib.awtextra.AbsoluteConstraints(160, 80, -1, -1));


        lblStaffID.setFont(new java.awt.Font("Segoe UI", 0, 14)); // NOI18N
        lblStaffID.setText("Staff ID:");
        jPanel2.add(lblStaffID, new
org.netbeans.lib.awtextra.AbsoluteConstraints(70, 130, -1, -1));


        txtStaffID.setColumns(20);
        jPanel2.add(txtStaffID, new
org.netbeans.lib.awtextra.AbsoluteConstraints(70, 150, 290, 30));


        btnCancel.setText("Cancel");
        btnCancel.setPreferredSize(new java.awt.Dimension(80, 35));
        jPanel2.add(btnCancel, new
org.netbeans.lib.awtextra.AbsoluteConstraints(110, 210, -1, -1));


        btnLogin.setText("Login");
        btnLogin.setPreferredSize(new java.awt.Dimension(80, 35));
```

```
        btnLogin.addActionListener(new java.awt.event.ActionListener()
        {
            public void actionPerformed(java.awt.event.ActionEvent evt)
            {
                btnLoginActionPerformed(evt);
            }
        });
        jPanel2.add(btnLogin, new
org.netbeans.lib.awtextra.AbsoluteConstraints(230, 210, -1, -1));


        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jPanel2, javax.swing.GroupLayout.DEFAULT_SIZE,
420, Short.MAX_VALUE)
        );
        layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jPanel2, javax.swing.GroupLayout.DEFAULT_SIZE,
300, Short.MAX_VALUE)
        );


        pack();
    }// </editor-fold>//GEN-END:initComponents
```

```
    private void btnLoginActionPerformed(java.awt.event.ActionEvent
evt)//GEN-FIRST:event_btnLoginActionPerformed
    {//GEN-HEADEREND:event_btnLoginActionPerformed
          String employeeIdStr = txtStaffID.getText().trim();


    // Check if input is empty
    if (employeeIdStr.isEmpty()) {
        JOptionPane.showMessageDialog(this,
            "Please enter your Employee ID.",
            "Login Error",
            JOptionPane.WARNING_MESSAGE);
        return;
    }


    // Check if input is numeric
    try {
        int employeeId = Integer.parseInt(employeeIdStr);


        // Connect to database and validate employee
        try {
            // Get an instance of the EmployeeDAO
            EmployeeDAO employeeDAO =
DBManager.getInstance().getDAO(EmployeeDAO.class);


            // Get employee by ID
            Employee employee = employeeDAO.getEmployeeById(employeeId);
```

```
            if (employee == null) {
                // Employee ID not found
                JOptionPane.showMessageDialog(this,
                    "Employee ID not found. Access denied.",
                    "Login Failed",
                    JOptionPane.ERROR_MESSAGE);
                return;
            }

            // Check employee role
            int roleId = employee.getRole().getRoleId();
            String roleName = employee.getRole().getRoleName();

            // Create audit log entry
            createLoginAuditLog(employee);

            // Login successful
            JOptionPane.showMessageDialog(this,
                "Login successful. Welcome " + employee.getFirstName() + " " +
employee.getLastName() +
                " (Role: " + roleName + ")",
                "Login Success",
                JOptionPane.INFORMATION_MESSAGE);

            // Open Hub window with employee information
            HubWindow hub = new HubWindow(employee);
            hub.setLocationRelativeTo(null);
```

```
            hub.setVisible(true);

            // Close login window
            this.dispose();

        } catch (Exception e) {
            JOptionPane.showMessageDialog(this,
                "Database error: " + e.getMessage(),
                "Login Error",
                JOptionPane.ERROR_MESSAGE);
            e.printStackTrace();
        }

    } catch (NumberFormatException e) {
        JOptionPane.showMessageDialog(this,
            "Please enter a valid numeric Employee ID.",
            "Login Error",
            JOptionPane.WARNING_MESSAGE);
    }

    }//GEN-LAST:event_btnLoginActionPerformed
// Helper method to create login audit log
private void createLoginAuditLog(Employee employee) {
    try {
        // Get DAOs
        ActionDAO actionDAO =
DBManager.getInstance().getDAO(ActionDAO.class);
```

```
        AuditingDAO auditingDAO =
DBManager.getInstance().getDAO(AuditingDAO.class);

        // Get Login action (ID 12)
        Action loginAction = actionDAO.getActionById(12);

        if (loginAction == null) {
            System.err.println("Login action not found in the database");
            return;
        }

        // Create audit entry
        Auditing audit = new Auditing();
        audit.setTimestamp(LocalDateTime.now());
        audit.setDetails("Employee " + employee.getFirstName() + " " +
employee.getLastName() +
                         " (ID: " + employee.getEmployeeId() + ") logged in
successfully");
        audit.setEmployee(employee);
        audit.setAction(loginAction);

        // Save audit log
        auditingDAO.createAuditLog(audit);

    } catch (Exception e) {
        System.err.println("Error creating login audit log: " + e.getMessage());
        e.printStackTrace();
    }
```

```
}
    /**
     * @param args the command line arguments
     */
    public static void main(String args[])
    {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
         * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try
        {
            for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels())
            {
                if ("Nimbus".equals(info.getName()))
                {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex)
        {
```

```
java.util.logging.Logger.getLogger(LogInWindow.class.getName()).log(java.uti
l.logging.Level.SEVERE, null, ex);
        } catch (InstantiationException ex)
        {

java.util.logging.Logger.getLogger(LogInWindow.class.getName()).log(java.uti
l.logging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex)
        {

java.util.logging.Logger.getLogger(LogInWindow.class.getName()).log(java.uti
l.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex)
        {

java.util.logging.Logger.getLogger(LogInWindow.class.getName()).log(java.uti
l.logging.Level.SEVERE, null, ex);
        }
        //</editor-fold>


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

```
    }

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JButton btnCancel;
    private javax.swing.JButton btnLogin;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JLabel lblEquipmentCheckoutSystem;
    private javax.swing.JLabel lblGBManufacturing;
    private javax.swing.JLabel lblStaffID;
    private javax.swing.JTextField txtStaffID;
    // End of variables declaration//GEN-END:variables
}
```

MainWindow.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/GUIForms/JFrame.java to edit this template
 */
package ecsapplication.UserInterface;
import javax.swing.ImageIcon;


/**
 *
 * @author Ughon
 */
public class MainWindow extends javax.swing.JFrame
{


    /**
     * Creates new form NewJFrame
     */
    public MainWindow()
    {
        initComponents();
        setLocationRelativeTo(null);
```

```
        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
        try {
            ImageIcon icon = new
ImageIcon(getClass().getResource("/Resources/GBM_Icon_16x16.png"));
            this.setIconImage(icon.getImage());
        } catch (Exception e) {
            System.out.println("Icon image could not be loaded: " +
e.getMessage());
        }
    }



    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-
BEGIN:initComponents
    private void initComponents()
    {


        jLabel1 = new javax.swing.JLabel();
        checkOutButton = new javax.swing.JButton();
        checkInButton = new javax.swing.JButton();
        statusLabel = new javax.swing.JLabel();
```

```
setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("Equipment Checkout System");
        setResizable(false);
        setSize(new java.awt.Dimension(450, 350));
        getContentPane().setLayout(new
org.netbeans.lib.awtextra.AbsoluteLayout());


        jLabel1.setFont(new java.awt.Font("Segoe UI", 0, 16)); // NOI18N
        jLabel1.setText("GB Manufacturing Equipment Checkout System");
        jLabel1.setAlignmentX(30.0F);
        jLabel1.setAlignmentY(20.0F);
        jLabel1.setPreferredSize(new java.awt.Dimension(390, 30));
        getContentPane().add(jLabel1, new
org.netbeans.lib.awtextra.AbsoluteConstraints(90, 0, -1, -1));


        checkOutButton.setText("Check Out Equipment");
        checkOutButton.setAlignmentX(60.0F);
        checkOutButton.setAlignmentY(80.0F);
        checkOutButton.setBorder(new javax.swing.border.LineBorder(new
java.awt.Color(0, 0, 255), 3, true));
        checkOutButton.setOpaque(true);
        checkOutButton.setPreferredSize(new java.awt.Dimension(330, 50));
        checkOutButton.addActionListener(new java.awt.event.ActionListener()
        {
            public void actionPerformed(java.awt.event.ActionEvent evt)
            {
```

```
            checkOutButtonActionPerformed(evt);
        }
    });
    getContentPane().add(checkOutButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(60, 80, -1, -1));

    checkInButton.setText("Check In Equipment");
    checkInButton.setAlignmentX(60.0F);
    checkInButton.setAlignmentY(80.0F);
    checkInButton.setBorder(new javax.swing.border.LineBorder(new
java.awt.Color(0, 0, 255), 3, true));
    checkInButton.setOpaque(true);
    checkInButton.setPreferredSize(new java.awt.Dimension(330, 50));
    checkInButton.addActionListener(new java.awt.event.ActionListener()
    {
        public void actionPerformed(java.awt.event.ActionEvent evt)
        {
            checkInButtonActionPerformed(evt);
        }
    });
    getContentPane().add(checkInButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(60, 160, -1, -1));

statusLabel.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
    statusLabel.setText("System Ready");
    statusLabel.setAlignmentX(0.5F);
    statusLabel.setPreferredSize(new java.awt.Dimension(390, 20));
```

```
        getContentPane().add(statusLabel, new
org.netbeans.lib.awtextra.AbsoluteConstraints(30, 260, 390, 20));


        pack();
    }// </editor-fold>//GEN-END:initComponents


    private void checkOutButtonActionPerformed(java.awt.event.ActionEvent
evt)//GEN-FIRST:event_checkOutButtonActionPerformed
    {//GEN-HEADEREND:event_checkOutButtonActionPerformed
      // Create and display the CheckOutWindow
    CheckOutWindow checkOutWindow = new CheckOutWindow(this);
    checkOutWindow.setVisible(true);
    }//GEN-LAST:event_checkOutButtonActionPerformed


    private void checkInButtonActionPerformed(java.awt.event.ActionEvent
evt)//GEN-FIRST:event_checkInButtonActionPerformed
    {//GEN-HEADEREND:event_checkInButtonActionPerformed
      // Create and display the CheckInWindow
    CheckInWindow checkInWindow = new CheckInWindow();
    checkInWindow.setVisible(true);
    }//GEN-LAST:event_checkInButtonActionPerformed


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

```
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
         * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try
        {
            for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels())
            {
                if ("Nimbus".equals(info.getName()))
                {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex)
        {
            java.util.logging.Logger.getLogger(MainWindow.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (InstantiationException ex)
        {
            java.util.logging.Logger.getLogger(MainWindow.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex)
```

```
        {

java.util.logging.Logger.getLogger(MainWindow.class.getName()).log(java.util
.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex)
        {

java.util.logging.Logger.getLogger(MainWindow.class.getName()).log(java.util
.logging.Level.SEVERE, null, ex);
        }
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>

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

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JButton checkInButton;
    private javax.swing.JButton checkOutButton;
```

```
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel statusLabel;
    // End of variables declaration//GEN-END:variables
}
```

OverdueEquipmentReportFrame.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/GUIForms/JPanel.java to edit this template
 */
package ecsapplication.UserInterface;
/**
 *
 * @author Ughon
 */
public class OverdueEquipmentReportFrame extends javax.swing.JPanel
{

    /**
     * Creates new form OverdueEquipmentReportFrame
     */
    public OverdueEquipmentReportFrame()
    {
        initComponents();
    }



    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
```

```
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents
    private void initComponents()
    {

        lblDaysOverdueFilter = new javax.swing.JLabel();
        cmbDaysOverdueFilter = new javax.swing.JComboBox<>();
        lblDepartmentsFilter = new javax.swing.JLabel();
        btnGenerateReport = new javax.swing.JButton();
        ReportPane = new javax.swing.JScrollPane();
        jTable1 = new javax.swing.JTable();
        lblOverDueequipmentTitle = new javax.swing.JLabel();
        btnExportUserReport = new javax.swing.JButton();
        lblCheckOutEquipmentLocation = new javax.swing.JComboBox<>();
        cmbDaysOverdueFilter1 = new javax.swing.JComboBox<>();

        setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

        lblDaysOverdueFilter.setText("Days Overdue:");
        lblDaysOverdueFilter.setPreferredSize(new java.awt.Dimension(90, 25));
        add(lblDaysOverdueFilter, new org.netbeans.lib.awtextra.AbsoluteConstraints(40, 70, -1, -1));

        cmbDaysOverdueFilter.setModel(new javax.swing.DefaultComboBoxModel<>(new String[] { "All", ">1 day", ">3 days", ">7 days", ">14 days", ">30 days" }));
```

```
        cmbDaysOverdueFilter.setPreferredSize(new java.awt.Dimension(100,
25));

        add(cmbDaysOverdueFilter, new
org.netbeans.lib.awtextra.AbsoluteConstraints(140, 100, -1, -1));


        lblDepartmentsFilter.setText("Equipment Type:");

        lblDepartmentsFilter.setPreferredSize(new java.awt.Dimension(90, 25));

        add(lblDepartmentsFilter, new
org.netbeans.lib.awtextra.AbsoluteConstraints(40, 100, 110, -1));


        btnGenerateReport.setText("Search");

        btnGenerateReport.setPreferredSize(new java.awt.Dimension(100, 25));

        add(btnGenerateReport, new
org.netbeans.lib.awtextra.AbsoluteConstraints(310, 80, -1, -1));


        ReportPane.setPreferredSize(new java.awt.Dimension(580, 385));


        jTable1.setModel(new javax.swing.table.DefaultTableModel(
            new Object [][]
            {
                {null, null, null, null, null},
                {null, null, null, null, null},
                {null, null, null, null, null},
                {null, null, null, null, null}
            },
            new String []
            {
                "ID", "Type", "Due Date", "Days Overdue", "Holder"
```

```
        }
    ));
    ReportPane.setViewportView(jTable1);

    add(ReportPane, new org.netbeans.lib.awtextra.AbsoluteConstraints(35,
150, 500, 250));

    lblOverDueequipmentTitle.setFont(new java.awt.Font("Segoe UI", 1,
14)); // NOI18N
    lblOverDueequipmentTitle.setText("Overdue Equipment Report");
    add(lblOverDueequipmentTitle, new
org.netbeans.lib.awtextra.AbsoluteConstraints(180, 9, -1, -1));

    btnExportUserReport.setText("Export");
    btnExportUserReport.setPreferredSize(new java.awt.Dimension(100,
39));
    add(btnExportUserReport, new
org.netbeans.lib.awtextra.AbsoluteConstraints(210, 410, -1, -1));

    lblCheckOutEquipmentLocation.setModel(new
javax.swing.DefaultComboBoxModel<>(new String[] { "1 North Warehouse",
"2 South Warehouse " }));
    lblCheckOutEquipmentLocation.setPreferredSize(new
java.awt.Dimension(102, 20));
    lblCheckOutEquipmentLocation.addActionListener(new
java.awt.event.ActionListener()
    {
        public void actionPerformed(java.awt.event.ActionEvent evt)
        {
            lblCheckOutEquipmentLocationActionPerformed(evt);
```

```
            }
        });
        add(lblCheckOutEquipmentLocation, new
org.netbeans.lib.awtextra.AbsoluteConstraints(140, 270, 190, -1));


        cmbDaysOverdueFilter1.setModel(new
javax.swing.DefaultComboBoxModel<>(new String[] { "All", ">1 day", ">3
days", ">7 days", ">14 days", ">30 days" }));
        cmbDaysOverdueFilter1.setPreferredSize(new java.awt.Dimension(100,
25));
        add(cmbDaysOverdueFilter1, new
org.netbeans.lib.awtextra.AbsoluteConstraints(140, 70, -1, -1));
    }// </editor-fold>//GEN-END:initComponents


    private void
lblCheckOutEquipmentLocationActionPerformed(java.awt.event.ActionEvent
evt)//GEN-FIRST:event_lblCheckOutEquipmentLocationActionPerformed
    {//GEN-
HEADEREND:event_lblCheckOutEquipmentLocationActionPerformed
        // TODO add your handling code here:
    }//GEN-LAST:event_lblCheckOutEquipmentLocationActionPerformed



    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JScrollPane ReportPane;
    private javax.swing.JButton btnExportUserReport;
    private javax.swing.JButton btnGenerateReport;
    private javax.swing.JComboBox<String> cmbDaysOverdueFilter;
    private javax.swing.JComboBox<String> cmbDaysOverdueFilter1;
```

```
    private javax.swing.JTable jTable1;
    private javax.swing.JComboBox<String> lblCheckOutEquipmentLocation;
    private javax.swing.JLabel lblDaysOverdueFilter;
    private javax.swing.JLabel lblDepartmentsFilter;
    private javax.swing.JLabel lblOverDueequipmentTitle;
    // End of variables declaration//GEN-END:variables
}
```

TransactionReport.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/GUIForms/JFrame.java to edit this template
 */
package ecsapplication.UserInterface;

import ecsapplication.DataAccess.DatabaseConnection;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;
import org.netbeans.lib.awtextra.AbsoluteLayout;
import org.netbeans.lib.awtextra.AbsoluteConstraints;

/**
 *
 * @author Ughon
 */
public class TransactionReport extends javax.swing.JFrame
{

    /**
     * Creates new form UserActivityReportWindow
```

```
     */
    public TransactionReport()
    {
        initComponents();
    }
    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-
BEGIN:initComponents
    private void initComponents()
    {


        lblEmployeeID = new javax.swing.JLabel();
        txtUserIdFilter = new javax.swing.JTextField();
        lblDateRange = new javax.swing.JLabel();
        dateChooserFrom = new com.toedter.calendar.JDateChooser();
        dateChooserTo = new com.toedter.calendar.JDateChooser();
        lblToFilter = new javax.swing.JLabel();
        lblFromFilter = new javax.swing.JLabel();
        userActivityReportPane = new javax.swing.JScrollPane();
        tblUserActivityReport = new javax.swing.JTable();
        btnExportUserReport = new javax.swing.JButton();
```

```
        btnGenerate = new java.awt.Button();
        txtEmployeeActivityReportTitle = new javax.swing.JLabel();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        getContentPane().setLayout(new
org.netbeans.lib.awtextra.AbsoluteLayout());

        lblEmployeeID.setText("User ID:");
        getContentPane().add(lblEmployeeID, new
org.netbeans.lib.awtextra.AbsoluteConstraints(10, 50, -1, -1));

        txtUserIdFilter.setPreferredSize(new java.awt.Dimension(100, 25));
        getContentPane().add(txtUserIdFilter, new
org.netbeans.lib.awtextra.AbsoluteConstraints(70, 50, -1, -1));

        lblDateRange.setText("Date Range:");
        lblDateRange.setPreferredSize(new java.awt.Dimension(80, 25));
        getContentPane().add(lblDateRange, new
org.netbeans.lib.awtextra.AbsoluteConstraints(10, 90, -1, -1));
        getContentPane().add(dateChooserFrom, new
org.netbeans.lib.awtextra.AbsoluteConstraints(140, 90, -1, -1));
        getContentPane().add(dateChooserTo, new
org.netbeans.lib.awtextra.AbsoluteConstraints(270, 90, -1, -1));

        lblToFilter.setText("To:");
        lblToFilter.setPreferredSize(new java.awt.Dimension(18, 25));
        getContentPane().add(lblToFilter, new
org.netbeans.lib.awtextra.AbsoluteConstraints(250, 90, -1, -1));
```

```
        lblFromFilter.setText("From:");
        lblFromFilter.setPreferredSize(new java.awt.Dimension(35, 25));
        getContentPane().add(lblFromFilter, new
org.netbeans.lib.awtextra.AbsoluteConstraints(100, 90, -1, -1));


        userActivityReportPane.setPreferredSize(new java.awt.Dimension(100,
250));


        tblUserActivityReport.setModel(new
javax.swing.table.DefaultTableModel(
            new Object [][]
            {
                {null, null, null, null, null},
                {null, null, null, null, null},
                {null, null, null, null, null},
                {null, null, null, null, null}
            },
            new String []
            {
                "Emplyee ID", "Equpiment Type", "Equipment Id", "Transaction
Date", "Return Date"
            }
        ));
        tblUserActivityReport.setPreferredSize(new java.awt.Dimension(400,
80));
        userActivityReportPane.setViewportView(tblUserActivityReport);
```

```
        getContentPane().add(userActivityReportPane, new
org.netbeans.lib.awtextra.AbsoluteConstraints(5, 150, 550, -1));


        btnExportUserReport.setText("Export");

        btnExportUserReport.setPreferredSize(new java.awt.Dimension(100,
39));

        getContentPane().add(btnExportUserReport, new
org.netbeans.lib.awtextra.AbsoluteConstraints(210, 410, -1, -1));


        btnGenerate.setLabel("Generate");

        btnGenerate.setPreferredSize(new java.awt.Dimension(100, 25));

        getContentPane().add(btnGenerate, new
org.netbeans.lib.awtextra.AbsoluteConstraints(400, 90, -1, -1));


        txtEmployeeActivityReportTitle.setFont(new java.awt.Font("Segoe UI", 1,
14)); // NOI18N

        txtEmployeeActivityReportTitle.setText("Employee Activity Report");

        getContentPane().add(txtEmployeeActivityReportTitle, new
org.netbeans.lib.awtextra.AbsoluteConstraints(10, 20, -1, -1));


        pack();


        // Load initial transaction data

        loadTransactionData();


        // Add button action listeners

        btnGenerate.addActionListener(new java.awt.event.ActionListener() {

            public void actionPerformed(java.awt.event.ActionEvent evt) {

                btnGenerateActionPerformed(evt);
```

```
            }
        });

        btnExportUserReport.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                btnExportUserReportActionPerformed(evt);
            }
        });
    }// </editor-fold>//GEN-END:initComponents

    private void loadTransactionData() {
        DefaultTableModel model = (DefaultTableModel)
tblUserActivityReport.getModel();
        model.setRowCount(0); // Clear existing data

        StringBuilder queryBuilder = new StringBuilder(
            "SELECT t.employee_id, eq.equipment_type, t.equipment_id, " +
            "t.transaction_date, t.return_date " +
            "FROM transaction t " +
            "JOIN equipment eq ON t.equipment_id = eq.equipment_id " +
            "WHERE 1=1 "
        );

        // Add filters if they are set
        String employeeId = txtUserIdFilter.getText().trim();
        java.util.Date fromDate = dateChooserFrom.getDate();
```

```
        java.util.Date toDate = dateChooserTo.getDate();

        java.util.ArrayList<Object> params = new java.util.ArrayList<>();

        if (!employeeId.isEmpty()) {
            queryBuilder.append("AND t.employee_id = ? ");
            params.add(Integer.parseInt(employeeId));
        }

        if (fromDate != null) {
            queryBuilder.append("AND t.transaction_date >= ? ");
            params.add(new java.sql.Timestamp(fromDate.getTime()));
        }

        if (toDate != null) {
            queryBuilder.append("AND t.transaction_date <= ? ");
            params.add(new java.sql.Timestamp(toDate.getTime()));
        }

        queryBuilder.append("ORDER BY t.transaction_date DESC LIMIT 100");

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(queryBuilder.toString())) {

            // Set parameters
            for (int i = 0; i < params.size(); i++) {
```

```
            stmt.setObject(i + 1, params.get(i));
        }

        try (ResultSet rs = stmt.executeQuery()) {
            while (rs.next()) {
                model.addRow(new Object[]{
                    rs.getInt("employee_id"),
                    rs.getString("equipment_type"),
                    rs.getInt("equipment_id"),
                    rs.getTimestamp("transaction_date"),
                    rs.getDate("return_date")
                });
            }
        }
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(this,
            "Error loading transaction data: " + e.getMessage(),
            "Database Error",
            JOptionPane.ERROR_MESSAGE);
        e.printStackTrace();
    } catch (NumberFormatException e) {
        JOptionPane.showMessageDialog(this,
            "Please enter a valid Employee ID number",
            "Input Error",
            JOptionPane.ERROR_MESSAGE);
    }
}
```

```
    private void btnGenerateActionPerformed(java.awt.event.ActionEvent evt)
{
        loadTransactionData();
    }


    private void
btnExportUserReportActionPerformed(java.awt.event.ActionEvent evt) {
        try {
            DefaultTableModel model = (DefaultTableModel)
tblUserActivityReport.getModel();
            StringBuilder csv = new StringBuilder();


            // Add headers
            for (int i = 0; i < model.getColumnCount(); i++) {
                csv.append(model.getColumnName(i));
                if (i < model.getColumnCount() - 1) {
                    csv.append(",");
                }
            }
            csv.append("\n");


            // Add data
            for (int i = 0; i < model.getRowCount(); i++) {
                for (int j = 0; j < model.getColumnCount(); j++) {
                    Object value = model.getValueAt(i, j);
                    csv.append(value != null ? value.toString() : "");
                    if (j < model.getColumnCount() - 1) {
```

```
                csv.append(",");
            }
        }
        csv.append("\n");
    }

    // Write to file
    java.io.File file = new java.io.File("transaction_report.csv");
    try (java.io.FileWriter fw = new java.io.FileWriter(file)) {
        fw.write(csv.toString());
    }

    JOptionPane.showMessageDialog(this,
        "Report exported successfully to: " + file.getAbsolutePath(),
        "Export Successful",
        JOptionPane.INFORMATION_MESSAGE);

  } catch (Exception e) {
    JOptionPane.showMessageDialog(this,
        "Error exporting report: " + e.getMessage(),
        "Export Error",
        JOptionPane.ERROR_MESSAGE);
    e.printStackTrace();
  }
}

/**
```

```
     * @param args the command line arguments
     */
    public static void main(String args[])
    {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
         * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try
        {
            for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels())
            {
                if ("Nimbus".equals(info.getName()))
                {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex)
        {
java.util.logging.Logger.getLogger(TransactionReport.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (InstantiationException ex)
```

```
        {

java.util.logging.Logger.getLogger(TransactionReport.class.getName()).log(ja
va.util.logging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex)
        {

java.util.logging.Logger.getLogger(TransactionReport.class.getName()).log(ja
va.util.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex)
        {

java.util.logging.Logger.getLogger(TransactionReport.class.getName()).log(ja
va.util.logging.Level.SEVERE, null, ex);
        }
        //</editor-fold>
        //</editor-fold>


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


    // Variables declaration - do not modify//GEN-BEGIN:variables
```

```
    private javax.swing.JButton btnExportUserReport;
    private java.awt.Button btnGenerate;
    private com.toedter.calendar.JDateChooser dateChooserFrom;
    private com.toedter.calendar.JDateChooser dateChooserTo;
    private javax.swing.JLabel lblDateRange;
    private javax.swing.JLabel lblEmployeeID;
    private javax.swing.JLabel lblFromFilter;
    private javax.swing.JLabel lblToFilter;
    private javax.swing.JTable tblUserActivityReport;
    private javax.swing.JLabel txtEmployeeActivityReportTitle;
    private javax.swing.JTextField txtUserIdFilter;
    private javax.swing.JScrollPane userActivityReportPane;
    // End of variables declaration//GEN-END:variables
}
```

UserManagementWindow.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/GUIForms/JFrame.java to edit this template
 */
package ecsapplication.UserInterface;

import ecsapplication.DataAccess.DBManager;
import ecsapplication.DataAccess.EmployeeDAO;
import ecsapplication.Model.Employee;
import java.util.List;
import javax.swing.table.DefaultTableModel;

/**
 *
 * @author Redux
 */
public class UserManagementWindow extends javax.swing.JFrame
{
    private EmployeeDAO employeeDAO;
    private DefaultTableModel userTableModel;

    /**
     * Creates new form UserManagementWindow
     */
    public UserManagementWindow()
```

```
    {
        initComponents();
        setLocationRelativeTo(null);
        setDefaultCloseOperation(DISPOSE_ON_CLOSE);

        // Initialize DAO
        employeeDAO =
DBManager.getInstance().getDAO(EmployeeDAO.class);

        // Get the table model
        userTableModel = (DefaultTableModel) tblUserList.getModel();

        // Load all employees
        loadAllEmployees();
    }

    private void loadAllEmployees() {
        try {
            List<Employee> employees = employeeDAO.getAllEmployees();
            userTableModel.setRowCount(0); // Clear existing data

            for (Employee employee : employees) {
                userTableModel.addRow(new Object[]{
                    employee.getEmployeeId(),
                    employee.getFirstName() + " " + employee.getLastName(),
                    employee.getHireDate().toString(),
                    employee.getRole().getRoleName(),
```

```
                employee.getSkill().getSkillLevel(),
                employee.getEmail(),
                "Edit/Delete" // Placeholder for actions
            });
        }
    } catch (Exception e) {
        javax.swing.JOptionPane.showMessageDialog(this,
            "Error loading employees: " + e.getMessage(),
            "Error",
            javax.swing.JOptionPane.ERROR_MESSAGE);
    }
}

/**
 * This method is called from within the constructor to initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is always
 * regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents
private void initComponents()
{

    jPanel1 = new javax.swing.JPanel();
    btnSearchUser = new javax.swing.JButton();
```

```
        txtSearchUser = new javax.swing.JTextField();
        btnAddNewUser1 = new javax.swing.JButton();
        jScrollPane1 = new javax.swing.JScrollPane();
        tblUserList = new javax.swing.JTable();
        lblSearch = new javax.swing.JLabel();


setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("Equipment Checkout System - User Management");
        setPreferredSize(new java.awt.Dimension(920, 565));
        setResizable(false);

        jPanel1.setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

        btnSearchUser.setBackground(new java.awt.Color(51, 102, 255));
        btnSearchUser.setForeground(new java.awt.Color(255, 255, 255));
        btnSearchUser.setText("Search");
        btnSearchUser.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                btnSearchUserActionPerformed(evt);
            }
        });
        jPanel1.add(btnSearchUser, new
org.netbeans.lib.awtextra.AbsoluteConstraints(760, 20, 120, 30));
        jPanel1.add(txtSearchUser, new
org.netbeans.lib.awtextra.AbsoluteConstraints(510, 20, 240, 30));

        btnAddNewUser1.setBackground(new java.awt.Color(51, 102, 255));
```

```
        btnAddNewUser1.setForeground(new java.awt.Color(255, 255, 255));
        btnAddNewUser1.setText("Add New User");
        btnAddNewUser1.addActionListener(new
java.awt.event.ActionListener()
        {
            public void actionPerformed(java.awt.event.ActionEvent evt)
            {
                btnAddNewUser1ActionPerformed(evt);
            }
        });
        jPanel1.add(btnAddNewUser1, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 20, 120, 30));


        tblUserList.setModel(new javax.swing.table.DefaultTableModel(
            new Object [][]
            {
                {null, null, null, null, null, null, null},
                {null, null, null, null, null, null, null},
                {null, null, null, null, null, null, null},
                {null, null, null, null, null, null, null},
                {null, null, null, null, null, null, null},
                {null, null, null, null, null, null, null},
                {null, null, null, null, null, null, null},
                {null, null, null, null, null, null, null},
                {null, null, null, null, null, null, null},
                {null, null, null, null, null, null, null},
                {null, null, null, null, null, null, null},
                {null, null, null, null, null, null, null},
```

```
                {null, null, null, null, null, null, null},
                {null, null, null, null, null, null, null},
                {null, null, null, null, null, null, null},
                {null, null, null, null, null, null, null},
                {null, null, null, null, null, null, null},
                {null, null, null, null, null, null, null},
                {null, null, null, null, null, null, null},
                {null, null, null, null, null, null, null},
                {null, null, null, null, null, null, null},
                {null, null, null, null, null, null, null},
                {null, null, null, null, null, null, null},
                {null, null, null, null, null, null, null},
                {null, null, null, null, null, null, null}
            },
            new String []
            {
                "Employee ID", "Name", "Department", "Role", "Skill Level",
"Email", "Actions"
            }
        ));
        jScrollPane1.setViewportView(tblUserList);


        jPanel1.add(jScrollPane1, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 80, 860, 410));


        lblSearch.setText("Search By Last Name");
```

```
        jPanel1.add(lblSearch, new
org.netbeans.lib.awtextra.AbsoluteConstraints(370, 30, -1, -1));


        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE,
917, Short.MAX_VALUE)
        );
        layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
        );


        pack();
    }// </editor-fold>//GEN-END:initComponents


    private void btnAddNewUser1ActionPerformed(java.awt.event.ActionEvent
evt)//GEN-FIRST:event_btnAddNewUser1ActionPerformed
    {//GEN-HEADEREND:event_btnAddNewUser1ActionPerformed
        AddUserWindow addUserWindow = new AddUserWindow();
        addUserWindow.setLocationRelativeTo(null); // center the window
        addUserWindow.setVisible(true);
    }//GEN-LAST:event_btnAddNewUser1ActionPerformed
```

```
    private void btnSearchUserActionPerformed(java.awt.event.ActionEvent
evt) {
        String searchText = txtSearchUser.getText().trim().toLowerCase();
        userTableModel.setRowCount(0); // Clear existing data

        try {
            List<Employee> employees = employeeDAO.getAllEmployees();

            for (Employee employee : employees) {
                // Check if the last name contains the search text
                if (employee.getLastName().toLowerCase().contains(searchText)) {
                    userTableModel.addRow(new Object[]{
                        employee.getEmployeeId(),
                        employee.getFirstName() + " " + employee.getLastName(),
                        employee.getHireDate().toString(),
                        employee.getRole().getRoleName(),
                        employee.getSkill().getSkillLevel(),
                        employee.getEmail(),
                        "Edit/Delete"
                    });
                }
            }

            if (userTableModel.getRowCount() == 0) {
                javax.swing.JOptionPane.showMessageDialog(this,
                    "No employees found with last name containing: " + searchText,
```

```
                "Search Results",
                javax.swing.JOptionPane.INFORMATION_MESSAGE);
            }
        } catch (Exception e) {
            javax.swing.JOptionPane.showMessageDialog(this,
                "Error searching employees: " + e.getMessage(),
                "Error",
                javax.swing.JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void main(String args[])
    {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
         * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try
        {
            for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels())
            {
                if ("Nimbus".equals(info.getName()))
                {
```

```
                javax.swing.UIManager.setLookAndFeel(info.getClassName());
                break;
            }
        }
    } catch (ClassNotFoundException ex)
    {

java.util.logging.Logger.getLogger(UserManagementWindow.class.getName()
).log(java.util.logging.Level.SEVERE, null, ex);
    } catch (InstantiationException ex)
    {

java.util.logging.Logger.getLogger(UserManagementWindow.class.getName()
).log(java.util.logging.Level.SEVERE, null, ex);
    } catch (IllegalAccessException ex)
    {

java.util.logging.Logger.getLogger(UserManagementWindow.class.getName()
).log(java.util.logging.Level.SEVERE, null, ex);
    } catch (javax.swing.UnsupportedLookAndFeelException ex)
    {

java.util.logging.Logger.getLogger(UserManagementWindow.class.getName()
).log(java.util.logging.Level.SEVERE, null, ex);
    }
    //</editor-fold>

    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable()
```

```
        {
            public void run()
            {
                new UserManagementWindow().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JButton btnAddNewUser1;
    private javax.swing.JButton btnSearchUser;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JLabel lblSearch;
    private javax.swing.JTable tblUserList;
    private javax.swing.JTextField txtSearchUser;
    // End of variables declaration//GEN-END:variables
}
```

ValueAnalysisReportPanel.java

```
/*
```

```
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/GUIForms/JPanel.java to edit this template
 */
package ecsapplication.UserInterface;

import ecsapplication.DataAccess.DBManager;
import ecsapplication.DataAccess.ValueLossDAO;
import ecsapplication.DataAccess.EquipmentDAO;
import ecsapplication.Model.Equipment;
import java.util.List;
import java.util.Map;
import java.util.HashMap;
import javax.swing.table.DefaultTableModel;
import javax.swing.*;
import java.awt.*;
import java.math.BigDecimal;
import java.math.RoundingMode;

/**
 *
 * @author Ughon
 */
public class ValueAnalysisReportPanel extends javax.swing.JPanel {
    private final ValueLossDAO valueLossDAO;
    private final EquipmentDAO equipmentDAO;
```

```
    private Map<String, BigDecimal> valueByType; // Store for chart rendering

    /**
     * Creates new form ValueAnalysisReportPanel
     */
    public ValueAnalysisReportPanel() {
        // Initialize DAOs first
        DBManager dbManager = DBManager.getInstance();
        this.valueLossDAO = dbManager.getDAO(ValueLossDAO.class);
        this.equipmentDAO = dbManager.getDAO(EquipmentDAO.class);

        // Then initialize components
        initComponents();

        // Load data after components are initialized
        loadValueAnalysis();
        loadEquipmentValueBreakdown();
    }

    private void loadValueAnalysis() {
        try {
            // Get total inventory value
            BigDecimal totalInventoryValue =
valueLossDAO.getTotalInventoryValue();
            lblTotalInventoryValue.setText("$" + String.format("%,.2f",
totalInventoryValue));
```

```
        // Get total lost value
        BigDecimal totalLostValue = valueLossDAO.getTotalLostValue();
        lblTotalLostValue.setText("$" + String.format("%,.2f", totalLostValue));

        // Calculate percentage lost
        if (totalInventoryValue.compareTo(BigDecimal.ZERO) > 0) {
            BigDecimal percentageLost =
totalLostValue.divide(totalInventoryValue, 4, BigDecimal.ROUND_HALF_UP)
                .multiply(new BigDecimal("100"));
            lblPercentageLost.setText(String.format("%.2f%%",
percentageLost));
        } else {
            lblPercentageLost.setText("0.00%");
        }

        // Calculate average value per item
        List<Equipment> allEquipment = equipmentDAO.getAllEquipment();
        if (!allEquipment.isEmpty()) {
            BigDecimal totalValue = allEquipment.stream()
                .map(Equipment::getValue)
                .filter(v -> v != null)
                .reduce(BigDecimal.ZERO, BigDecimal::add);
            BigDecimal avgValue = totalValue.divide(new
BigDecimal(allEquipment.size()), 2, BigDecimal.ROUND_HALF_UP);
            lblAverageValue.setText("$" + String.format("%,.2f", avgValue));
        }

        // Find most valuable equipment
```

```
            Equipment mostValuable = allEquipment.stream()
                .max((e1, e2) -> e1.getValue().compareTo(e2.getValue()))
                .orElse(null);
            if (mostValuable != null) {
                lblMostValuableType.setText(mostValuable.getEquipmentType() +
                    " ($" + String.format("%,.2f", mostValuable.getValue()) + ")");
            }

            // Calculate monthly loss trend
            Map<String, BigDecimal> monthlyLosses =
valueLossDAO.getMonthlyLossTrend();
            if (!monthlyLosses.isEmpty()) {
                String trend = monthlyLosses.entrySet().stream()
                    .sorted(Map.Entry.comparingByKey())
                    .map(e -> String.format("%s: $%,.2f", e.getKey(), e.getValue()))
                    .reduce((a, b) -> a + ", " + b)
                    .orElse("No data");
                lblMonthlyTrend.setText(trend);
            }

        } catch (Exception e) {
            e.printStackTrace();
            JOptionPane.showMessageDialog(this,
                "Error loading value analysis: " + e.getMessage(),
                "Data Loading Error",
                JOptionPane.ERROR_MESSAGE);
        }
```

```
    }

    private void loadEquipmentValueBreakdown() {
        try {
            // Get all equipment
            List<Equipment> equipmentList = equipmentDAO.getAllEquipment();

            // Group equipment by type and calculate total value
            valueByType = new HashMap<>();
            Map<String, Integer> countByType = new HashMap<>();

            for (Equipment equipment : equipmentList) {
                String type = equipment.getEquipmentType();
                BigDecimal value = equipment.getValue() != null ?
equipment.getValue() : BigDecimal.ZERO;

                valueByType.merge(type, value, BigDecimal::add);
                countByType.merge(type, 1, Integer::sum);
            }

            // Update the table
            DefaultTableModel model = (DefaultTableModel)
tblValueBreakdown.getModel();
            model.setRowCount(0);

            for (Map.Entry<String, BigDecimal> entry : valueByType.entrySet()) {
                String type = entry.getKey();
```

```
            BigDecimal totalValue = entry.getValue();
            int count = countByType.get(type);
            BigDecimal avgValue = totalValue.divide(new BigDecimal(count),
2, BigDecimal.ROUND_HALF_UP);

            model.addRow(new Object[]{
                type,
                count,
                String.format("$%,.2f", totalValue),
                String.format("$%,.2f", avgValue)
            });
        }

        // Trigger chart repaint
        chartPanel.repaint();

    } catch (Exception e) {
        e.printStackTrace();
        JOptionPane.showMessageDialog(this,
            "Error loading equipment value breakdown: " + e.getMessage(),
            "Data Loading Error",
            JOptionPane.ERROR_MESSAGE);
    }
}

// Custom chart panel class
private class ChartPanel extends JPanel {
```

```
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        if (valueByType == null || valueByType.isEmpty()) return;

        Graphics2D g2d = (Graphics2D) g;
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);

        int padding = 40;
        int barWidth = 50;
        int spacing = 20;
        int chartWidth = getWidth() - 2 * padding;
        int chartHeight = getHeight() - 2 * padding;

        // Find maximum value for scaling
        BigDecimal maxValue = valueByType.values().stream()
            .max(BigDecimal::compareTo)
            .orElse(BigDecimal.ONE);

        // Draw axes
        g2d.setColor(Color.BLACK);
        g2d.drawLine(padding, getHeight() - padding, getWidth() - padding,
getHeight() - padding); // X-axis
        g2d.drawLine(padding, getHeight() - padding, padding, padding); // Y-
axis

        // Draw bars
```

```
        int x = padding + spacing;
        int maxBarHeight = chartHeight - padding;
        int colorIndex = 0;
        Color[] colors = {new Color(51, 102, 204), new Color(220, 57, 18),
                new Color(16, 150, 24), new Color(153, 0, 153)};

        for (Map.Entry<String, BigDecimal> entry : valueByType.entrySet()) {
            // Calculate bar height
            int barHeight = maxBarHeight * entry.getValue().divide(maxValue,
2, RoundingMode.HALF_UP)
                .min(BigDecimal.ONE).intValue();

            // Draw bar
            g2d.setColor(colors[colorIndex % colors.length]);
            g2d.fillRect(x, getHeight() - padding - barHeight, barWidth,
barHeight);
            g2d.setColor(Color.BLACK);
            g2d.drawRect(x, getHeight() - padding - barHeight, barWidth,
barHeight);

            // Draw label
            g2d.rotate(-Math.PI / 4);
            g2d.drawString(entry.getKey(),
                (float) (-(getHeight() - padding + 20)),
                (float) (x + barWidth/2));
            g2d.rotate(Math.PI / 4);

            x += barWidth + spacing;
```

```
            colorIndex++;
        }
    }
}

@Override
public Dimension getPreferredSize() {
    return new Dimension(800, 600);
}

private void initComponents() {
    // Initialize components
    panelSummary = new javax.swing.JPanel();
    lblTitle = new javax.swing.JLabel();
    lblTotalInventoryValueLabel = new javax.swing.JLabel();
    lblTotalInventoryValue = new javax.swing.JLabel();
    lblTotalLostValueLabel = new javax.swing.JLabel();
    lblTotalLostValue = new javax.swing.JLabel();
    lblPercentageLostLabel = new javax.swing.JLabel();
    lblPercentageLost = new javax.swing.JLabel();

    scrollPaneBreakdown = new javax.swing.JScrollPane();
    tblValueBreakdown = new javax.swing.JTable();
    lblBreakdownTitle = new javax.swing.JLabel();
    btnRefresh = new javax.swing.JButton();
    btnExport = new javax.swing.JButton();
```

```
        setLayout(new java.awt.BorderLayout(10, 10));

        // Setup summary panel
        panelSummary.setLayout(new java.awt.GridBagLayout());
        GridBagConstraints summaryGbc = new GridBagConstraints();
        summaryGbc.insets = new Insets(5, 5, 5, 5);
        summaryGbc.anchor = GridBagConstraints.WEST;

        lblTitle.setFont(new java.awt.Font("Segoe UI", 1, 24));
        lblTitle.setText("Inventory Value Analysis");
        summaryGbc.gridx = 0;
        summaryGbc.gridy = 0;
        summaryGbc.gridwidth = 2;
        summaryGbc.anchor = GridBagConstraints.CENTER;
        panelSummary.add(lblTitle, summaryGbc);

        // Total Inventory Value
        lblTotalInventoryValueLabel.setFont(new java.awt.Font("Segoe UI", 0,
14));
        lblTotalInventoryValueLabel.setText("Total Inventory Value:");
        summaryGbc.gridy = 1;
        summaryGbc.gridwidth = 1;
        summaryGbc.anchor = GridBagConstraints.EAST;
        panelSummary.add(lblTotalInventoryValueLabel, summaryGbc);

        lblTotalInventoryValue.setFont(new java.awt.Font("Segoe UI", 1, 14));
        lblTotalInventoryValue.setText("$0.00");
```

```
summaryGbc.gridx = 1;
summaryGbc.anchor = GridBagConstraints.WEST;
panelSummary.add(lblTotalInventoryValue, summaryGbc);

// Total Lost Value
lblTotalLostValueLabel.setFont(new java.awt.Font("Segoe UI", 0, 14));
lblTotalLostValueLabel.setText("Total Lost Value:");
summaryGbc.gridx = 0;
summaryGbc.gridy = 2;
summaryGbc.anchor = GridBagConstraints.EAST;
panelSummary.add(lblTotalLostValueLabel, summaryGbc);

lblTotalLostValue.setFont(new java.awt.Font("Segoe UI", 1, 14));
lblTotalLostValue.setText("$0.00");
summaryGbc.gridx = 1;
summaryGbc.anchor = GridBagConstraints.WEST;
panelSummary.add(lblTotalLostValue, summaryGbc);

// Percentage Lost
lblPercentageLostLabel.setFont(new java.awt.Font("Segoe UI", 0, 14));
lblPercentageLostLabel.setText("Percentage Lost:");
summaryGbc.gridx = 0;
summaryGbc.gridy = 3;
summaryGbc.anchor = GridBagConstraints.EAST;
panelSummary.add(lblPercentageLostLabel, summaryGbc);

lblPercentageLost.setFont(new java.awt.Font("Segoe UI", 1, 14));
```

```
        lblPercentageLost.setText("0.00%");
        summaryGbc.gridx = 1;
        summaryGbc.anchor = GridBagConstraints.WEST;
        panelSummary.add(lblPercentageLost, summaryGbc);

        add(panelSummary, BorderLayout.NORTH);

        // Setup breakdown table
        lblBreakdownTitle.setFont(new java.awt.Font("Segoe UI", 1, 18));
        lblBreakdownTitle.setText("Equipment Value Breakdown");
        lblBreakdownTitle.setBorder(BorderFactory.createEmptyBorder(10, 10,
10, 10));
        add(lblBreakdownTitle, BorderLayout.CENTER);

        tblValueBreakdown.setModel(new javax.swing.table.DefaultTableModel(
            new Object [][] {},
            new String [] {
                "Equipment Type", "Count", "Total Value", "Average Value"
            }
        ) {
            boolean[] canEdit = new boolean [] {
                false, false, false, false
            };

            public boolean isCellEditable(int rowIndex, int columnIndex) {
                return canEdit[columnIndex];
            }
```

```
        });
        scrollPaneBreakdown.setViewportView(tblValueBreakdown);
        add(scrollPaneBreakdown, BorderLayout.SOUTH);

        // Setup buttons
        JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));

        btnRefresh = new JButton("Refresh");
        btnRefresh.addActionListener(evt -> {
            loadValueAnalysis();
            loadEquipmentValueBreakdown();
        });
        buttonPanel.add(btnRefresh);

        btnExport = new JButton("Export Report");
        btnExport.addActionListener(evt -> {
            // TODO: Implement export functionality
            JOptionPane.showMessageDialog(this, "Export functionality will be
implemented soon.");
        });
        buttonPanel.add(btnExport);

        add(buttonPanel, BorderLayout.EAST);

        // Add new labels for additional metrics
        lblAverageValueLabel = new JLabel("Average Value per Item:");
        lblAverageValue = new JLabel("$0.00");
```

```
        lblAverageValue.setFont(new Font("Segoe UI", Font.BOLD, 14));

        lblMostValuableTypeLabel = new JLabel("Most Valuable Equipment:");
        lblMostValuableType = new JLabel("None");
        lblMostValuableType.setFont(new Font("Segoe UI", Font.BOLD, 14));

        lblMonthlyTrendLabel = new JLabel("Monthly Loss Trend:");
        lblMonthlyTrend = new JLabel("No data");
        lblMonthlyTrend.setFont(new Font("Segoe UI", Font.BOLD, 14));

        // Create chart panel
        chartPanel = new ChartPanel();
        chartPanel.setPreferredSize(new Dimension(500, 300));
        chartPanel.setBorder(BorderFactory.createTitledBorder("Value
Distribution by Equipment Type"));

        // Main content panel
        JPanel contentPanel = new JPanel(new BorderLayout(10, 10));
        contentPanel.add(chartPanel, BorderLayout.CENTER);
        contentPanel.add(new JScrollPane(tblValueBreakdown),
BorderLayout.SOUTH);

        add(contentPanel, BorderLayout.CENTER);
    }

    private void addToGrid(JPanel panel, JLabel label, JLabel value, int row,
GridBagConstraints gbc) {
        gbc.gridx = 0;
```

```
        gbc.gridy = row;
        gbc.anchor = GridBagConstraints.EAST;
        panel.add(label, gbc);

        gbc.gridx = 1;
        gbc.anchor = GridBagConstraints.WEST;
        panel.add(value, gbc);
    }

    // Variables declaration
    private javax.swing.JPanel panelSummary;
    private javax.swing.JLabel lblTitle;
    private javax.swing.JLabel lblTotalInventoryValueLabel;
    private javax.swing.JLabel lblTotalInventoryValue;
    private javax.swing.JLabel lblTotalLostValueLabel;
    private javax.swing.JLabel lblTotalLostValue;
    private javax.swing.JLabel lblPercentageLostLabel;
    private javax.swing.JLabel lblPercentageLost;
    private javax.swing.JLabel lblBreakdownTitle;
    private javax.swing.JScrollPane scrollPaneBreakdown;
    private javax.swing.JTable tblValueBreakdown;
    private javax.swing.JButton btnRefresh;
    private javax.swing.JButton btnExport;
    private JLabel lblAverageValueLabel;
    private JLabel lblAverageValue;
    private JLabel lblMostValuableTypeLabel;
    private JLabel lblMostValuableType;
```

```
    private JLabel lblMonthlyTrendLabel;
    private JLabel lblMonthlyTrend;
    private ChartPanel chartPanel;
    // End of variables declaration
}
```