Tech License Manager

Introduction:

The **Tech License Manager** is a software solution designed to efficiently **manage**, **track**, and **monitor software licenses** within an organization.

It ensures **proper license allocation**, prevents **unauthorized usage**, and helps maintain compliance by providing **real-time insights**, **alerts**, and **centralized control** over all licensing assets.

Project Structure and Packages:

The project is organized into three main packages:

- **`Manager`:** Contains the core business logic and data management classes. These classes interact directly with the database via `MongoCollection` objects and encapsulate the operations related to users, licenses, authentication, and notifications.
- **`Model`:** Defines the data structures that represent the entities within the application, such as `User`, `License`, and `Notification`. These classes are simple data holders with getters and setters.
- **`ui`:** Houses the graphical user interface (GUI) components built using Java Swing. Each class in this package represents a specific window or panel in the application, handling user interactions and displaying data.
- **`Utils`:** Provides utility classes for common tasks, such as database operations (`DbUtil`) and system-specific functionalities like detecting installed applications (`InstalledAppDetector`).

Key Properties and Functionalities

- **User Authentication and Registration**: Secure login and new user creation with password hashing.
- **License Management**: Comprehensive CRUD operations for licenses, including details like software name, type, key, and expiration date.
- **User Management**: CRUD operations for users, with the ability to assign licenses and view individual license profiles.
- **Notifications**: System for generating and displaying notifications, particularly for expiring licenses.

- **Reporting**:
  - **License Usage Report**: Shows which licenses are assigned to whom.
  - **License Expiry Report**: Lists licenses that are due to expire soon.
  - **User License Report**: Provides a detailed breakdown of licenses for each user.
  - **Expiry Reminders**: Functionality to trigger reminders for expiring licenses.
  - **Installed Applications**: Ability to detect and list applications installed on the local Windows machine.
- **Data Persistence**: Uses MongoDB as the backend database for storing all application data.
- **Modular Design**: The application is structured into `Model`, `Manager`, `ui`, and `Utils` packages, promoting separation of concerns and maintainability.
- **Swing GUI**: A desktop application built with Java Swing, featuring a modern look and feel with styled buttons and tables.
- **Error Handling**: Basic `try-catch` blocks are implemented for database operations and image loading.
- **Graceful Shutdown**: The MongoDB client is closed gracefully when the application exits.

Technologies Used

- **Language**: Java
- **GUI Framework**: Java Swing
- **Database**: MongoDB (with MongoDB Java Driver)
- **Build Tool**: (Assumed, typically Maven or Gradle for dependency management)
- **Hashing**: SHA-256 (for passwords, with a note on security improvement needed)

Package and Class Breakdown:

`Manager` Package

This package contains the "manager" classes responsible for handling the business logic and interactions with the data store (MongoDB).

AuthManager.java

- **Purpose**: Manages user authentication and registration processes. It relies on `Usermanager` for user data retrieval and password verification.
- **Key Properties/Methods**:
  - `AuthManager(Usermanager userManager)`: Constructor that injects the `Usermanager` dependency.

- `authenticateUser(String username, String password)`: Attempts to authenticate a user by checking their credentials against stored data. It uses `userManager.getUserById` and `userManager.checkPassword`.
- `registerNewUser(String userId, String name, String email, String imagePath, String plainTextPassword, String role)`: Registers a new user, ensuring the user ID is unique and the password is hashed before storage (handled internally by `Usermanager`).

Code:

```
package Manager;

import Model.User;
import Manager.Usermanager;

public class AuthManager {
    private final Usermanager userManager;

    public AuthManager(Usermanager userManager) {
        this.userManager = userManager;
    }


    public User authenticateUser(String username, String password) {
        try {
            User user = userManager.getUserById(username); // Assuming userId is
used as username for simplicity

            if (user != null && user.getPasswordHash() != null) {
                boolean passwordMatches = userManager.checkPassword(password,
user.getPasswordHash());

                if (passwordMatches) {
                    System.out.println("User '" + username + "' authenticated
successfully.");
                    return user;
                } else {
                    System.out.println("Authentication failed for user '" +
username + "': Invalid password.");
                    return null;
                }
            } else {
                System.out.println("Authentication failed for user '" + username
+ "': User not found or no password set.");
                return null; // User not found or no password hash stored
```

```
            }
        } catch (Exception e) { // General catch for unexpected issues
            System.err.println("An unexpected error occurred during
authentication for user '" + username + "': " + e.getMessage());
            e.printStackTrace();
            return null;
        }
    }

    /**
     * Registers a new user.
     * @param userId The unique ID for the new user.
     * @param name The user's name.
     * @param email The user's email.
     * @param imagePath The path to the user's profile image.
     * @param plainTextPassword The plain text password for the new user.
     * @param role The role of the new user (e.g., "admin", "standard").
     * @return true if registration is successful, false otherwise.
     */
    public boolean registerNewUser(String userId, String name, String email,
String imagePath, String plainTextPassword, String role) {
        try {
            if (userManager.getUserById(userId) != null) {
                System.out.println("Registration failed: User ID '" + userId + "'
already exists.");
                return false; // User ID already exists
            }

            // UserManager's addUser now handles hashing internally before saving
            User newUser = new User(userId, name, email, imagePath,
plainTextPassword, role);
            userManager.addUser(newUser); // This will hash the password within
UserManager
            System.out.println("User '" + userId + "' registered successfully.");
            return true;
        } catch (Exception e) { // General catch for unexpected issues during
registration
            System.err.println("An unexpected error occurred during user
registration for user '" + userId + "': " + e.getMessage());
            e.printStackTrace();
            return false;
        }
    }
}
```

Licensemanager.java

- **Purpose**: Manages CRUD (Create, Read, Update, Delete) operations for `License` objects and handles license-related business logic, including expiry reminders.
- **Key Properties/Methods**:
  - `Licensemanager(MongoCollection<Document> licenseCollection, Notificationmanager notificationManager)`: Constructor, injecting MongoDB collection for licenses and `Notificationmanager`.
  - `addLicense(License license)`: Adds a new license to the database.
  - `removeLicense(String id)`: Deletes a license by its ID.
  - `getAllLicenses()`: Retrieves all licenses from the database.
  - `updateLicense(License license)`: Updates an existing license's details.
  - `getExpiringLicenses(int days)`: Returns a list of licenses expiring within a specified number of days.
  - `sendExpiryReminders(List<User> users)`: Iterates through licenses and users to send notifications for expiring licenses via the `Notificationmanager`.

Code:

```
package Manager;

import com.mongodb.client.MongoCollection;
import com.mongodb.client.model.Filters;
import com.mongodb.client.model.Updates;
import org.bson.Document;
import Model.License;
import Model.User;
import java.util.*;
import java.util.stream.Collectors;

public class Licensemanager {
    private MongoCollection<Document> licenseCollection;
    private Notificationmanager notificationManager; // Dependency for
notifications

    public Licensemanager(MongoCollection<Document> licenseCollection,
Notificationmanager notificationManager) {
        this.licenseCollection = licenseCollection;
        this.notificationManager = notificationManager;
    }

    public void addLicense(License license) {
        Document doc = new Document("id", license.getId())
                .append("softwareName", license.getSoftwareName())
```

```
                .append("licenseType", license.getLicenseType())
                .append("licenseKey", license.getLicenseKey())
                .append("expirationDate", license.getExpirationDate())
                .append("assignedUserId", license.getAssignedUserId());
        licenseCollection.insertOne(doc);
    }

    public void removeLicense(String id) {
        licenseCollection.deleteOne(Filters.eq("id", id));
    }

    public List<License> getAllLicenses() {
        List<License> licenses = new ArrayList<>();
        for (Document doc : licenseCollection.find()) {
            licenses.add(docToLicense(doc));
        }
        return licenses;
    }

    public License getLicenseById(String id) {
        Document doc = licenseCollection.find(Filters.eq("id", id)).first();
        return doc != null ? docToLicense(doc) : null;
    }

    public List<License> getExpiringLicenses(int days) {
        Date today = new Date();
        Calendar cal = Calendar.getInstance();
        cal.setTime(today);
        cal.add(Calendar.DAY_OF_YEAR, days);
        Date future = cal.getTime();

        List<License> expiringLicenses = new ArrayList<>();
        for (Document doc : licenseCollection.find()) {
            License license = docToLicense(doc);
            if (license.getExpirationDate() != null &&
                license.getExpirationDate().after(today) &&
                license.getExpirationDate().before(future)) {
                expiringLicenses.add(license);
            }
        }
        return expiringLicenses;
    }

    public void sendExpiryReminders(List<User> users) {
        List<License> expiring = getExpiringLicenses(7);
```

```java
        for (License license : expiring) {
            if (license.getAssignedUserId() != null) {
                for (User user : users) {
                    if (user.getUserId().equals(license.getAssignedUserId())) {
                        String message = "License '" + license.getSoftwareName()
+ "' for " + user.getName() + " expires on " + license.getExpirationDate();
                        notificationManager.addNotification(user.getUserId(),
license.getId(), message);
                        System.out.println("[Reminder] " + message); // For
console logging
                        break;
                    }
                }
            }
        }
    }

    public void updateLicense(License license) {
        Document doc = new Document("id", license.getId())
                .append("softwareName", license.getSoftwareName())
                .append("licenseType", license.getLicenseType())
                .append("licenseKey", license.getLicenseKey())
                .append("expirationDate", license.getExpirationDate())
                .append("assignedUserId", license.getAssignedUserId());
        licenseCollection.replaceOne(Filters.eq("id", license.getId()), doc);
    }

    // Helper method to convert Document to License object
    private License docToLicense(Document doc) {
        License license = new License(
                doc.getString("id"),
                doc.getString("softwareName"),
                doc.getString("licenseType"),
                doc.getString("licenseKey"),
                doc.getDate("expirationDate")
        );
        license.setAssignedUserId(doc.getString("assignedUserId"));
        return license;
    }
}
```

Notificationmanager.java

- **Purpose**: Manages the creation and retrieval of `Notification` objects.
- **Key Properties/Methods**:
  - `Notificationmanager(MongoCollection<Document> notificationCollection)`: Constructor, injecting MongoDB collection for notifications.
  - `addNotification(String userId, String licenseId, String message)`: Creates and stores a new notification.
  - `getNotificationsByUserId(String userId)`: Retrieves notifications for a specific user.
  - `getAllNotifications()`: Retrieves all notifications.
  - `clearAllNotifications()`: Deletes all notifications from the database.

Code:

```
package Manager;

import com.mongodb.client.MongoCollection;
import com.mongodb.client.model.Filters;
import org.bson.Document;
import Model.Notification;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.UUID;

public class Notificationmanager {
    private final MongoCollection<Document> notificationCollection;

    public Notificationmanager(MongoCollection<Document> notificationCollection)
{
        this.notificationCollection = notificationCollection;
    }

    public void addNotification(String userId, String licenseId, String message)
{
        String notificationId = UUID.randomUUID().toString();
        Date sentAt = new Date();
        Document doc = new Document("id", notificationId)
                .append("userId", userId)
                .append("licenseId", licenseId)
                .append("message", message)
```

```
                .append("sentAt", sentAt);
        notificationCollection.insertOne(doc);
        System.out.println("New Notification: " + message + " for user " +
userId);
    }

    public List<Notification> getNotificationsByUserId(String userId) {
        List<Notification> userNotifications = new ArrayList<>();
        for (Document doc : notificationCollection.find(Filters.eq("userId",
userId))) {
            userNotifications.add(docToNotification(doc));
        }
        return userNotifications;
    }

    public List<Notification> getAllNotifications() {
        List<Notification> notifications = new ArrayList<>();
        for (Document doc : notificationCollection.find()) {
            notifications.add(docToNotification(doc));
        }
        return notifications;
    }

    public void clearAllNotifications() {
        notificationCollection.deleteMany(new Document()); // Deletes all
documents in the collection
    }


    private Notification docToNotification(Document doc) {
        return new Notification(
                doc.getString("id"),
                doc.getString("userId"),
                doc.getString("licenseId"),
                doc.getString("message"),
                doc.getDate("sentAt")
        );
    }
}
```

Usermanager.java

- **Purpose**: Manages CRUD operations for `User` objects, including password hashing and verification.
- **Key Properties/Methods**:
  - `Usermanager(MongoCollection<Document> userCollection)`: Constructor, injecting MongoDB collection for users.
  - `hashPassword(String plainTextPassword)`: Hashes a plain text password using SHA-256 (note: the snippet indicates this is "INSECURE" for production and should be replaced with stronger methods like BCrypt).
  - `checkPassword(String plainTextPassword, String hashedPassword)`: Verifies a plain text password against a stored hash.
  - `addUser(User user)`: Adds a new user, hashing their password internally.
  - `removeUser(String id)`: Deletes a user by their ID.
  - `getAllUsers()`: Retrieves all users from the database.
  - `getUserById(String id)`: Retrieves a single user by their ID (crucial for `AuthManager`).
  - `updateUser(User updatedUser)`: Updates an existing user's details.

Code:

```
package Manager;

import com.mongodb.client.MongoCollection;
import com.mongodb.client.model.Filters;
import com.mongodb.client.model.Updates; // Still needed for potential partial
updates
import org.bson.Document;
import Model.User;
import java.util.ArrayList;
import java.util.List;
import java.util.UUID;
import java.security.MessageDigest; // For simple hashing (INSECURE)
import java.security.NoSuchAlgorithmException; // For hashing (INSECURE)
import java.util.Base64; // For encoding hash (INSECURE)

public class Usermanager {
    private final MongoCollection<Document> userCollection;

    public Usermanager(MongoCollection<Document> userCollection) {
        this.userCollection = userCollection;
    }

    private String hashPassword(String plainTextPassword) {
        try {
```

```java
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            byte[] hashedBytes = md.digest(plainTextPassword.getBytes());
            return Base64.getEncoder().encodeToString(hashedBytes);
        } catch (NoSuchAlgorithmException e) {
            System.err.println("Hashing algorithm not found: " + e.getMessage());
            return null;
        }
    }

    public boolean checkPassword(String plainTextPassword, String hashedPassword)
{

        if (hashedPassword == null) return false;
        return hashPassword(plainTextPassword).equals(hashedPassword);
    }


    public void addUser(User user) {
        // Hash password before storing
        String hashedPassword = hashPassword(user.getPasswordHash());
        user.setPasswordHash(hashedPassword);

        Document doc = new Document("userId", user.getUserId())
                .append("name", user.getName())
                .append("email", user.getEmail())
                .append("imagepath", user.getimagepath())
                .append("passwordHash", user.getPasswordHash())
                .append("role", user.getRole())
                .append("licenseIds", user.getLicenseIds());
        userCollection.insertOne(doc);
    }

    public void removeUser(String id) {
        userCollection.deleteOne(Filters.eq("userId", id));
    }

    public List<User> getAllUsers() {
        List<User> users = new ArrayList<>();
        for (Document doc : userCollection.find()) {
            users.add(docToUser(doc));
        }
        return users;
    }
```

```java
    public User getUserById(String id) { // This method is crucial for
AuthManager
        Document doc = userCollection.find(Filters.eq("userId", id)).first();
        return doc != null ? docToUser(doc) : null;
    }

    public void updateUser(User updatedUser) {

        Document doc = new Document("userId", updatedUser.getUserId())
                .append("name", updatedUser.getName())
                .append("email", updatedUser.getEmail())
                .append("imagepath", updatedUser.getimagepath())
                .append("passwordHash", updatedUser.getPasswordHash())
                .append("role", updatedUser.getRole())
                .append("licenseIds", updatedUser.getLicenseIds());
        userCollection.replaceOne(Filters.eq("userId", updatedUser.getUserId()),
doc);
    }

    // Helper method to convert Document to User object
    private User docToUser(Document doc) {
        User user = new User(
                doc.getString("userId"),
                doc.getString("name"),
                doc.getString("email"),
                doc.getString("imagepath"),
                doc.getString("passwordHash"),
                doc.getString("role")
        );

        List<String> licenseIds = doc.getList("licenseIds", String.class);
        if (licenseIds != null) {
            for (String licenseId : licenseIds) {
                user.assignLicense(licenseId);
            }
        }
        return user;
    }
}
```

Model Package:

This package defines the data models (POJOs) used throughout the application.

License.java:

- **Purpose**: Represents a software license.
- **Key Properties**: `id`, `softwareName`, `licenseType`, `licenseKey`, `expirationDate`, `assignedUserId`.
- **Methods**: Getters and setters for all properties, a constructor, `toString()`, and `setAssignedUserId()`.

Code:

```
package Model;

import java.io.Serializable;
import java.util.Date;

public class License implements Serializable {
    private String id;
    private String softwareName;
    private String licenseType;
    private String licenseKey;
    private Date expirationDate;
    private String assignedUserId;

    public License(String id, String softwareName, String licenseType, String
licenseKey, Date expirationDate) {
        this.id = id;
        this.softwareName = softwareName;
        this.licenseType = licenseType;
        this.licenseKey = licenseKey;
        this.expirationDate = expirationDate;
    }

    public String getId() {
        return id;
    }
    public String getSoftwareName() {
        return softwareName;
    }
    public String getLicenseType() {
        return licenseType;
     }
    public String getLicenseKey() {
        return licenseKey;
     }
```

```java
    public Date getExpirationDate() {
        return expirationDate;
     }
    public String getAssignedUserId() {
        return assignedUserId;
 }

    public void setAssignedUserId(String userId) {
        this.assignedUserId = userId;
    }

    public String toString() {
        return softwareName + " (" + licenseType + ")";
    }

    public void setSoftwareName(String text) {

        throw new UnsupportedOperationException("Unimplemented method
'setSoftwareName'");
    }

    public void setLicenseType(String selectedItem) {

        throw new UnsupportedOperationException("Unimplemented method
'setLicenseType'");
    }

    public void setExpirationDate(Date expiryDate) {

        throw new UnsupportedOperationException("Unimplemented method
'setExpirationDate'");
    }
}
```

➢ Notification.java:

- **Purpose**: Represents a system notification.

- **Key Properties**: `id`, `userId`, `licenseId`, `message`, `sentAt`.
- **Methods**: Getters and setters for all properties, and a constructor.

Code:

```
package Model;

import java.io.Serializable;
import java.util.Date;

public class Notification implements Serializable {
    private String id;
    private String userId;
    private String licenseId;
    private String message;
    private Date sentAt;

    public Notification(String id, String userId, String licenseId, String
message, Date sentAt) {
        this.id = id;
        this.userId = userId;
        this.licenseId = licenseId;
        this.message = message;
        this.sentAt = sentAt;
    }

    public String getId() {
        return id;
    }

    public String getUserId() {
        return userId;
    }

    public String getLicenseId() {
        return licenseId;
    }

    public String getMessage() {
        return message;
    }

    public Date getSentAt() {
        return sentAt;
    }
```

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

    public void setSentAt(Date sentAt) {
        this.sentAt = sentAt;
    }
}
```

➢ User.java:

- **Purpose**: Represents a user in the system.
- **Key Properties**: `userId`, `name`, `email`, `imagepath`, `passwordHash`, `role`, `licenseIds` (a list of license IDs assigned to the user).
- **Methods**: Getters and setters for properties, constructors (one with password hash, one defaulting to "standard" role), `assignLicense(String licenseId)`, and `getLicenseProfile(List<License> allLicenses)` for generating a formatted string of assigned licenses.

Code:

```
package Model;

import java.io.Serializable;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.List;

public class User implements Serializable {
    private String userId;
    private String name;
    private String email;
    private String imagepath;
    private String passwordHash;
    private String role;
    private List<String> licenseIds;


    public User(String userId, String name, String email, String imagepath,
String passwordHash, String role) {
```

```java
        this.userId = userId;
        this.name = name;
        this.email = email;
        this.imagepath = imagepath;
        this.passwordHash = passwordHash;
        this.role = role;
        this.licenseIds = new ArrayList<>();
    }


    public User(String userId, String name, String email, String imagepath) {
        this(userId, name, email, imagepath, null, "standard"); // Default to no
password hash and "standard" role
    }

    public String getUserId() {
        return userId;
    }
    public String getName() {
        return name;
    }
    public String getEmail() {
        return email;
    }
    public List<String> getLicenseIds() {
        return licenseIds;
    }

    public String getimagepath() {
        return imagepath;
    }
    public void setimagepath(String imagepath) {
        this.imagepath = imagepath;
    }

    public String getPasswordHash() {
        return passwordHash;
    }
    public void setPasswordHash(String passwordHash) {
        this.passwordHash = passwordHash;
    }

    public String getRole() {
        return role;
    }
```

```java
    public void setRole(String role) {
        this.role = role;
    }

    public void assignLicense(String licenseId) {
        if (!licenseIds.contains(licenseId)) { // Prevent duplicates
            licenseIds.add(licenseId);
        }
    }

    public String getLicenseProfile(List<Model.License> allLicenses) {
        StringBuilder sb = new StringBuilder();
        sb.append("Licenses for ").append(name).append(" (Role:
").append(role).append("): \n");
        if (licenseIds.isEmpty()) {
            sb.append("  No licenses assigned.\n");
        } else {
            SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
            for (String licenseId : licenseIds) {
                for (Model.License l : allLicenses) {
                    if (l.getId().equals(licenseId)) {
                        sb.append("- Software:
").append(l.getSoftwareName()).append("\n")
                          .append("  Type:
").append(l.getLicenseType()).append("\n")
                          .append("  Key:
").append(l.getLicenseKey()).append("\n")
                          .append("  Expiry:
").append(sdf.format(l.getExpirationDate())).append("\n\n");
                        break;
                    }
                }
            }
        }
        return sb.toString();
    }
}
```

- **`ui`** Package:

This package contains the Swing GUI panels and frames.

➢ LoginPanel.java:

- **Purpose**: Provides the user login and new user registration interface. This is the entry point for user interaction.
- **Key Properties/Methods**:
  - `LoginPanel(AuthManager am, Licensemanager lm, Usermanager um, Notificationmanager nm)`: Constructor, taking all manager dependencies.
  - `attemptLogin()`: Handles login attempts, authenticating via `AuthManager`. If successful, it proceeds to `Mainmenu`.
  - `showRegisterDialog()`: Displays a dialog for new user registration, collecting user details and calling `authManager.registerNewUser`.
  - Includes UI components for username, password, and login/register buttons.

Code:

```
package ui;

import Manager.AuthManager;
import Manager.Licensemanager;
import Manager.Usermanager;
import Manager.Notificationmanager;
import Model.User;

import javax.swing.*;
import java.awt.*;
import java.io.File;

import javax.swing.border.EmptyBorder;

public class LoginPanel extends JFrame {
    private AuthManager authManager;
    private Licensemanager licenseManager;
    private Usermanager userManager;
    private Notificationmanager notificationManager;

    private JTextField usernameField;
    private JPasswordField passwordField;
    private JLabel messageLabel;

    public LoginPanel(AuthManager am, Licensemanager lm, Usermanager um,
Notificationmanager nm) {
        this.authManager = am;
        this.licenseManager = lm;
```

```
        this.userManager = um;
        this.notificationManager = nm;

        setTitle("Tech License Manager - Login");
        setSize(400, 280);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JPanel mainPanel = new JPanel(new BorderLayout(15, 15));
        mainPanel.setBorder(new EmptyBorder(20, 20, 20, 20));
        mainPanel.setBackground(new Color(240, 248, 255)); // Light background

        // --- Header ---
        JLabel loginHeader = new JLabel("Login to License Manager",
SwingConstants.CENTER);
        loginHeader.setFont(new Font("Segoe UI", Font.BOLD, 22));
        loginHeader.setForeground(new Color(60, 90, 150)); // Dark blue text
        mainPanel.add(loginHeader, BorderLayout.NORTH);

        // --- Input Panel ---
        JPanel inputPanel = new JPanel(new GridBagLayout());
        inputPanel.setBackground(new Color(240, 248, 255)); // Match main panel
background
        GridBagConstraints gbc = new GridBagConstraints();
        gbc.insets = new Insets(8, 8, 8, 8);
        gbc.fill = GridBagConstraints.HORIZONTAL;

        // Username
        gbc.gridx = 0; gbc.gridy = 0;
        inputPanel.add(new JLabel("Username:"), gbc);
        gbc.gridx = 1; gbc.gridy = 0; gbc.weightx = 1.0;
        usernameField = new JTextField(15);
        inputPanel.add(usernameField, gbc);

        // Password
        gbc.gridx = 0; gbc.gridy = 1;
        inputPanel.add(new JLabel("Password:"), gbc);
        gbc.gridx = 1; gbc.gridy = 1; gbc.weightx = 1.0;
        passwordField = new JPasswordField(15);
        inputPanel.add(passwordField, gbc);

        mainPanel.add(inputPanel, BorderLayout.CENTER);

        // --- Message Label ---
        messageLabel = new JLabel("", SwingConstants.CENTER);
```

```
        messageLabel.setForeground(Color.RED);
        messageLabel.setFont(new Font("Segoe UI", Font.PLAIN, 12));
        mainPanel.add(messageLabel, BorderLayout.SOUTH);

        add(mainPanel);

        // --- Buttons Panel (will be at the bottom of the content pane) ---
        JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 15,
10));
        buttonPanel.setBackground(new Color(240, 248, 255));

        JButton loginButton = createStyledButton("Login");
        JButton registerButton = createStyledButton("Register New User"); //
Option to register (for admin to add users)
        JButton exitButton = createStyledButton("Exit");

        loginButton.addActionListener(e -> performLogin());
        registerButton.addActionListener(e -> showRegisterDialog());
        exitButton.addActionListener(e -> System.exit(0));

        buttonPanel.add(loginButton);
        buttonPanel.add(registerButton);
        buttonPanel.add(exitButton);

        // Add buttonPanel to the frame's content pane directly, not mainPanel
        // to keep it separate at the bottom.
        getContentPane().add(mainPanel, BorderLayout.CENTER);
        getContentPane().add(buttonPanel, BorderLayout.SOUTH);


        setVisible(true);
        SwingUtilities.updateComponentTreeUI(this); // Apply L&F
    }


    private JButton createStyledButton(String text) {
        JButton button = new JButton(text);
        button.setBackground(new Color(60, 90, 150)); // Dark blue background
        button.setForeground(Color.WHITE); // White text
        button.setFont(new Font("Segoe UI", Font.BOLD, 12)); // Bold font
        button.setFocusPainted(false); // Remove focus border
        button.setBorder(BorderFactory.createEmptyBorder(10, 20, 10, 20)); //
Padding
        button.setCursor(new Cursor(Cursor.HAND_CURSOR)); // Hand cursor on hover
```

```
        // Add hover effect
        button.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseEntered(java.awt.event.MouseEvent evt) {
                button.setBackground(new Color(80, 110, 180)); // Lighter blue on
hover
            }
            public void mouseExited(java.awt.event.MouseEvent evt) {
                button.setBackground(new Color(60, 90, 150)); // Original color
on exit
            }
        });
        return button;
    }

    private void performLogin() {
        String username = usernameField.getText();
        String password = new String(passwordField.getPassword()); // Get
password as char array and convert to String

        if (username.isEmpty() || password.isEmpty()) {
            messageLabel.setText("Username and password cannot be empty.");
            return;
        }

        User authenticatedUser = authManager.authenticateUser(username,
password);

        if (authenticatedUser != null) {
            messageLabel.setText(""); // Clear any previous error message
            JOptionPane.showMessageDialog(this, "Login successful! Welcome, " +
authenticatedUser.getName() + "!", "Login Success",
JOptionPane.INFORMATION_MESSAGE);
            dispose(); // Close login panel

            // Launch the main application UI (Welcomepage, which then goes to
Mainmenu)
            new Welcomepage(licenseManager, userManager, notificationManager,
authManager);

            // Here you could also save the authenticatedUser to a static context
            // or pass it to Mainmenu to control access based on role.
            // Example: ApplicationContext.setLoggedInUser(authenticatedUser);
        } else {
            messageLabel.setText("Invalid username or password.");
            passwordField.setText(""); // Clear password field on failure
```

```
        }
    }

    private void showRegisterDialog() {
        JTextField newUserIdField = new JTextField(15);
        JTextField newNameField = new JTextField(15);
        JTextField newEmailField = new JTextField(15);
        JPasswordField newPasswordField = new JPasswordField(15);
        JPasswordField confirmPasswordField = new JPasswordField(15);
        JTextField newImagePathField = new JTextField(15);
        newImagePathField.setEditable(false);
        JButton browseImageBtn = new JButton("Browse");

        browseImageBtn.addActionListener(e -> {
            JFileChooser chooser = new JFileChooser();
            chooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
            int result = chooser.showOpenDialog(this);
            if (result == JFileChooser.APPROVE_OPTION) {
                File selectedFile = chooser.getSelectedFile();
                newImagePathField.setText(selectedFile.getAbsolutePath());
            }
        });

        String[] roles = {"standard", "admin"}; // Define available roles
        JComboBox<String> roleComboBox = new JComboBox<>(roles);
        roleComboBox.setSelectedItem("standard"); // Default role

        JPanel registerPanel = new JPanel(new GridBagLayout());
        registerPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));
        GridBagConstraints gbc = new GridBagConstraints();
        gbc.insets = new Insets(5, 5, 5, 5);
        gbc.fill = GridBagConstraints.HORIZONTAL;

        gbc.gridx = 0; gbc.gridy = 0; registerPanel.add(new JLabel("User ID
(Username):"), gbc);
        gbc.gridx = 1; gbc.gridy = 0; registerPanel.add(newUserIdField, gbc);

        gbc.gridx = 0; gbc.gridy = 1; registerPanel.add(new JLabel("Name:"),
gbc);
        gbc.gridx = 1; gbc.gridy = 1; registerPanel.add(newNameField, gbc);

        gbc.gridx = 0; gbc.gridy = 2; registerPanel.add(new JLabel("Email:"),
gbc);
        gbc.gridx = 1; gbc.gridy = 2; registerPanel.add(newEmailField, gbc);
```

```
        gbc.gridx = 0; gbc.gridy = 3; registerPanel.add(new JLabel("Password:"),
gbc);
        gbc.gridx = 1; gbc.gridy = 3; registerPanel.add(newPasswordField, gbc);

        gbc.gridx = 0; gbc.gridy = 4; registerPanel.add(new JLabel("Confirm
Password:"), gbc);
        gbc.gridx = 1; gbc.gridy = 4; registerPanel.add(confirmPasswordField,
gbc);

        gbc.gridx = 0; gbc.gridy = 5; registerPanel.add(new JLabel("Image
Path:"), gbc);
        gbc.gridx = 1; gbc.gridy = 5; registerPanel.add(newImagePathField, gbc);
        gbc.gridx = 2; gbc.gridy = 5; registerPanel.add(browseImageBtn, gbc);

        gbc.gridx = 0; gbc.gridy = 6; registerPanel.add(new JLabel("Role:"),
gbc);
        gbc.gridx = 1; gbc.gridy = 6; registerPanel.add(roleComboBox, gbc);


        int option = JOptionPane.showConfirmDialog(this, registerPanel, "Register
New User", JOptionPane.OK_CANCEL_OPTION, JOptionPane.PLAIN_MESSAGE);

        if (option == JOptionPane.OK_OPTION) {
            String userId = newUserIdField.getText();
            String name = newNameField.getText();
            String email = newEmailField.getText();
            String password = new String(newPasswordField.getPassword());
            String confirmPassword = new
String(confirmPasswordField.getPassword());
            String imagePath = newImagePathField.getText();
            String role = (String) roleComboBox.getSelectedItem();

            if (userId.isEmpty() || name.isEmpty() || email.isEmpty() ||
password.isEmpty() || confirmPassword.isEmpty()) {
                JOptionPane.showMessageDialog(this, "All fields marked with an
asterisk (*) must be filled.", "Input Error", JOptionPane.ERROR_MESSAGE);
                return;
            }
            if (!password.equals(confirmPassword)) {
                JOptionPane.showMessageDialog(this, "Passwords do not match.",
"Input Error", JOptionPane.ERROR_MESSAGE);
                return;
            }
```

```
            if (authManager.registerNewUser(userId, name, email, imagePath,
password, role)) {
                JOptionPane.showMessageDialog(this, "User registered
successfully!", "Registration Success", JOptionPane.INFORMATION_MESSAGE);
            } else {
                JOptionPane.showMessageDialog(this, "Failed to register user.
User ID might already exist.", "Registration Failed", JOptionPane.ERROR_MESSAGE);
            }
        }
    }
}
```

➢ Welcomepage.java:

- **Purpose**: A splash screen displayed at application startup.
- **Key Properties/Methods**:
  - `Welcomepage(Licensemanager lm, Usermanager um, Notificationmanager nm, AuthManager am)`: Constructor, taking all manager dependencies.
  - Displays a welcome message and an application icon (attempts to load from `/images/welcome_icon.png` or a local file).
  - Uses a `javax.swing.Timer` to automatically close after 3 seconds and open the `Mainmenu`.

Code:

```
package ui;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import Manager.Licensemanager;
import Manager.Notificationmanager;
import Manager.Usermanager;
import Manager.AuthManager;
import java.net.URL;

public class Welcomepage extends JFrame {

    private Licensemanager licenseManager;
    private Usermanager userManager;
    private Notificationmanager notificationManager;
    private AuthManager authManager;
```

```
    public Welcomepage(Licensemanager lm, Usermanager um, Notificationmanager nm,
AuthManager am) {
        this.licenseManager = lm;
        this.userManager = um;
        this.notificationManager = nm;
        this.authManager = am;

        setTitle("Welcome to Tech License App");
        setSize(500, 350);
        setLocationRelativeTo(null);
        setUndecorated(true);
        getRootPane().setWindowDecorationStyle(JRootPane.NONE);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JPanel panel = new JPanel(new GridBagLayout());
        panel.setBackground(new Color(230, 240, 250));
        panel.setBorder(BorderFactory.createLineBorder(new Color(60, 90, 150),
3));

        GridBagConstraints gbc = new GridBagConstraints();
        gbc.insets = new Insets(10, 10, 10, 10);
        gbc.gridx = 0;
        gbc.gridy = 0;
        gbc.fill = GridBagConstraints.BOTH;
        gbc.anchor = GridBagConstraints.CENTER;

        JLabel imageLabel = new JLabel();
        try {
            URL imageUrl = getClass().getResource("/images/welcome_icon.png");
            if (imageUrl == null) {
                imageUrl = new URL("file:./images/welcome_icon.png");
            }
            ImageIcon originalIcon = new ImageIcon(imageUrl);
            Image scaledImage = originalIcon.getImage().getScaledInstance(120,
120, Image.SCALE_SMOOTH);
            imageLabel.setIcon(new ImageIcon(scaledImage));
        } catch (Exception e) {
            System.err.println("Error loading image for Welcomepage: " +
e.getMessage());
            imageLabel.setText("(Image Missing)");
            imageLabel.setHorizontalAlignment(SwingConstants.CENTER);
            imageLabel.setVerticalAlignment(SwingConstants.CENTER);
            imageLabel.setPreferredSize(new Dimension(120, 120));
```

```
            imageLabel.setBorder(BorderFactory.createLineBorder(Color.RED));
        }
        panel.add(imageLabel, gbc);

        JLabel welcomeLabel = new JLabel("Welcome to Tech License App");
        welcomeLabel.setFont(new Font("Segoe UI", Font.BOLD, 28));
        welcomeLabel.setForeground(new Color(25, 75, 125));
        gbc.gridy = 1;
        panel.add(welcomeLabel, gbc);

        JLabel statusLabel = new JLabel("Loading managers...",
SwingConstants.CENTER);
        statusLabel.setFont(new Font("Segoe UI", Font.PLAIN, 14));
        statusLabel.setForeground(new Color(100, 100, 100));
        gbc.gridy = 2;
        gbc.insets = new Insets(5, 10, 20, 10);
        panel.add(statusLabel, gbc);

        add(panel);
        setVisible(true);

        javax.swing.Timer timer = new javax.swing.Timer(3000, new
ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                dispose(); // Close the welcome page
                // Pass all managers to Mainmenu
                new Mainmenu(licenseManager, userManager, notificationManager);
            }
        });
        timer.setRepeats(false);
        timer.start();
    }
}
```

➢ Mainmenu.java:

- **Purpose**: The main menu of the application, displayed after successful login. It provides navigation to other management panels.
- **Key Properties/Methods**:
    - `Mainmenu(Licensemanager lm, Usermanager um, Notificationmanager nm)`: Constructor, taking manager dependencies.

- Provides buttons for "License Management", "User Management", "Reports", "Notifications", "Settings", and "Exit".
- Action listeners for each button open the respective UI panels (e.g., `Licensepanel`, `Userpanel`).
- `createStyledButton(String text)`: A helper method to ensure consistent button styling.

Code:

```
package ui;

import javax.swing.*;
import java.awt.*;
import Manager.*;
import javax.swing.border.EmptyBorder;

public class Mainmenu extends JFrame {

    private Licensemanager licenseManager;
    private Usermanager userManager;
    private Notificationmanager notificationManager;

    public Mainmenu(Licensemanager lm, Usermanager um, Notificationmanager nm) {
        this.licenseManager = lm;
        this.userManager = um;
        this.notificationManager = nm;

        setTitle("Tech License Manager - Main Menu");
        setSize(500, 450); // Slightly larger for better spacing
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null); // Center the window

        // Set a clean background color for the frame content pane
        getContentPane().setBackground(new Color(240, 240, 240)); // Light gray
background


        JPanel headerPanel = new JPanel();
        headerPanel.setBackground(new Color(0, 150, 136)); // A nice blue for the
header
        headerPanel.setBorder(new EmptyBorder(15, 0, 15, 0)); // Padding around
the header
        headerPanel.setLayout(new FlowLayout(FlowLayout.CENTER)); // Center align
header content
```

```
        JLabel titleLabel = new JLabel("Welcome to License Management");
        titleLabel.setFont(new Font("Segoe UI", Font.BOLD, 28)); // Modern font,
bold, larger size
        titleLabel.setForeground(Color.WHITE); // White text for contrast
        headerPanel.add(titleLabel);

        // --- Buttons Panel ---
        JPanel buttonsPanel = new JPanel();
        buttonsPanel.setLayout(new BoxLayout(buttonsPanel, BoxLayout.Y_AXIS)); //
Stack buttons vertically
        buttonsPanel.setBorder(new EmptyBorder(20, 50, 20, 50)); // Padding
around the buttons panel
        buttonsPanel.setBackground(new Color(240, 240, 240)); // Match frame
background

        // Create buttons and apply common styling
        JButton btnLicense = createStyledButton("License Management");
        JButton btnUser = createStyledButton("User Management");
        JButton btnReport = createStyledButton("Reports");
        JButton btnNotifications = createStyledButton("Notifications");
        JButton btnSettings = createStyledButton("Settings");
        JButton btnExit = createStyledButton("Exit");

        // Add action listeners
        btnLicense.addActionListener(e -> new Licensepanel(licenseManager,
userManager, notificationManager).setVisible(true));
        btnUser.addActionListener(e -> new Userpanel(licenseManager, userManager,
notificationManager).setVisible(true));
        btnReport.addActionListener(e -> new Reportpanel(licenseManager,
userManager).setVisible(true));
        btnNotifications.addActionListener(e -> new
Notificationpanel(notificationManager, userManager).setVisible(true));
        btnSettings.addActionListener(e -> new Settingspanel().setVisible(true));
        btnExit.addActionListener(e -> System.exit(0));

        // Add buttons to the panel with alignment and spacing
        buttonsPanel.add(Box.createVerticalGlue()); // Pushes buttons to center
        buttonsPanel.add(btnLicense);
        buttonsPanel.add(Box.createVerticalStrut(15)); // Spacing between buttons
        buttonsPanel.add(btnUser);
        buttonsPanel.add(Box.createVerticalStrut(15));
        buttonsPanel.add(btnReport);
        buttonsPanel.add(Box.createVerticalStrut(15));
        buttonsPanel.add(btnNotifications);
        buttonsPanel.add(Box.createVerticalStrut(15));
```

```
        buttonsPanel.add(btnSettings);
        buttonsPanel.add(Box.createVerticalStrut(15));
        buttonsPanel.add(btnExit);
        buttonsPanel.add(Box.createVerticalGlue()); // Pushes buttons to center

        // Set maximum size for buttons to ensure consistent width in BoxLayout
        Dimension buttonSize = new Dimension(300, 100); // Wider and taller
buttons
        btnLicense.setMaximumSize(buttonSize);
        btnUser.setMaximumSize(buttonSize);
        btnReport.setMaximumSize(buttonSize);
        btnNotifications.setMaximumSize(buttonSize);
        btnSettings.setMaximumSize(buttonSize);
        btnExit.setMaximumSize(buttonSize);

        // Align buttons to center horizontally within the BoxLayout
        btnLicense.setAlignmentX(Component.CENTER_ALIGNMENT);
        btnUser.setAlignmentX(Component.CENTER_ALIGNMENT);
        btnReport.setAlignmentX(Component.CENTER_ALIGNMENT);
        btnNotifications.setAlignmentX(Component.CENTER_ALIGNMENT);
        btnSettings.setAlignmentX(Component.CENTER_ALIGNMENT);
        btnExit.setAlignmentX(Component.CENTER_ALIGNMENT);


        // Add panels to the frame
        add(headerPanel, BorderLayout.NORTH);
        add(buttonsPanel, BorderLayout.CENTER);

        setVisible(true);
    }

    // Helper method to create consistently styled buttons
    private JButton createStyledButton(String text) {
        JButton button = new JButton(text);
        button.setFont(new Font("Segoe UI", Font.PLAIN, 18)); // Modern font,
good size
        button.setBackground(new Color(0, 150, 136)); // Lighter blue for buttons
        button.setForeground(Color.WHITE); // White text
        button.setFocusPainted(false); // Remove focus border for cleaner look
        button.setBorder(BorderFactory.createEmptyBorder(10, 20, 10, 20)); //
Padding inside button
        button.setBorderPainted(false); // Let Nimbus handle border

        return button;
    }
```

```
}
```

➢ Licensepanel.java:

- **Purpose**: Manages license data through a tabular view, allowing users to add, edit, and remove licenses.
- **Key Properties/Methods**:
  - `Licensepanel(Licensemanager lm, Usermanager um, Notificationmanager nm)`: Constructor.
  - `JTable` and `DefaultTableModel`: Displays license data in a table.
  - `refreshTable()`: Populates the table with current license data.
  - `addLicenseDialog()`: Displays a dialog to add a new license.
  - `deleteSelectedLicense()`: Deletes the selected license from the table and database.
  - `editSelectedLicense()`: Displays a dialog to edit the selected license's details.
  - `assignLicenseDialog()`: Allows assigning a license to a user.

Code:

```
package ui;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.DefaultTableCellRenderer; // For table cell alignment
import Manager.*;
import Model.*;
import java.awt.*;
import java.awt.event.*;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.List;
import javax.swing.border.EmptyBorder; // For padding

public class Licensepanel extends JFrame {
    private Licensemanager licenseManager;
    private Usermanager userManager;
    private Notificationmanager notificationManager;
    private JTable table;
    private DefaultTableModel model;
    private SimpleDateFormat dateFormatter = new SimpleDateFormat("yyyy-MM-dd");
// Consistent date format

    public Licensepanel(Licensemanager lm, Usermanager um, Notificationmanager
nm) {
```

```
        this.licenseManager = lm;
        this.userManager = um;
        this.notificationManager = nm;

        setTitle("License Management");
        setSize(800, 600); // Increased size for better table visibility
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE); // Dispose on close so
it doesn't close the whole app

        // Set a clean background color for the frame content pane
        getContentPane().setBackground(new Color(245, 245, 245)); // Light
background

        model = new DefaultTableModel(new String[]{"ID", "Software", "Type",
"Expiration", "Assigned To"}, 0) {
            @Override
            public boolean isCellEditable(int row, int column) {
                return false; // Make cells non-editable
            }
        };
        table = new JTable(model);
        table.setFillsViewportHeight(true); // Fills the viewport height
        table.setRowHeight(25); // Increase row height for readability
        table.setFont(new Font("Segoe UI", Font.PLAIN, 14)); // Consistent font
        table.getTableHeader().setFont(new Font("Segoe UI", Font.BOLD, 14)); //
Bold header font
        table.getTableHeader().setBackground(new Color(220, 220, 220)); // Light
gray header
        table.setSelectionBackground(new Color(170, 200, 250)); // Light blue
selection background
        table.setGridColor(new Color(200, 200, 200)); // Lighter grid lines


        DefaultTableCellRenderer centerRenderer = new DefaultTableCellRenderer();
        centerRenderer.setHorizontalAlignment(JLabel.CENTER);
        table.getColumnModel().getColumn(0).setCellRenderer(centerRenderer);

        refreshTable();

        JScrollPane scrollPane = new JScrollPane(table);
        scrollPane.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10)); //
Padding around table

        // --- Buttons Panel ---
```

```
        JPanel buttonPanel = new JPanel();
        buttonPanel.setLayout(new FlowLayout(FlowLayout.CENTER, 15, 10)); //
Center align buttons with spacing
        buttonPanel.setBorder(new EmptyBorder(10, 0, 10, 0)); // Padding around
button panel
        buttonPanel.setBackground(new Color(245, 245, 245)); // Match background

        JButton addBtn = createStyledButton("Add License");
        JButton deleteBtn = createStyledButton("Delete License");
        JButton assignBtn = createStyledButton("Assign to User");
        JButton unassignBtn = createStyledButton("Unassign License");
        JButton editBtn = createStyledButton("Edit License"); // Added for
editing existing licenses

        addBtn.addActionListener(e -> addLicenseDialog());
        deleteBtn.addActionListener(e -> deleteSelectedLicense());
        assignBtn.addActionListener(e -> assignLicense());
        unassignBtn.addActionListener(e -> unassignLicense());
        editBtn.addActionListener(e -> editLicenseDialog()); // Action listener
for edit

        buttonPanel.add(addBtn);
        buttonPanel.add(deleteBtn);
        buttonPanel.add(assignBtn);
        buttonPanel.add(unassignBtn);
        buttonPanel.add(editBtn); // Add edit button to panel


        setLayout(new BorderLayout());
        add(scrollPane, BorderLayout.CENTER);
        add(buttonPanel, BorderLayout.SOUTH);

        setVisible(true);
    }

    // Helper method to create consistently styled buttons (reused from Mainmenu)
    private JButton createStyledButton(String text) {
        JButton button = new JButton(text);
        button.setFont(new Font("Segoe UI", Font.PLAIN, 16)); // Slightly smaller
font for panel buttons
        button.setBackground(new Color(60, 90, 150)); // Lighter blue
        button.setForeground(Color.WHITE);
        button.setFocusPainted(false);
        button.setBorder(BorderFactory.createEmptyBorder(8, 18, 8, 18)); //
Padding inside button
```

```
        button.setBorderPainted(false); // Let Nimbus handle border if any
        return button;
    }

    private void refreshTable() {
        model.setRowCount(0);
        for (License l : licenseManager.getAllLicenses()) {
            String assignedToName = "Unassigned";
            if (l.getAssignedUserId() != null &&
!l.getAssignedUserId().isEmpty()) {
                User assignedUser =
userManager.getUserById(l.getAssignedUserId());
                if (assignedUser != null) {
                    assignedToName = assignedUser.getName();
                }
            }
            model.addRow(new Object[]{
                l.getId(),
                l.getSoftwareName(),
                l.getLicenseType(),
                dateFormatter.format(l.getExpirationDate()), // Use dateFormatter
                assignedToName
            });
        }
    }

    private void addLicenseDialog() {
        JTextField softwareNameField = new JTextField(20);
        String[] licenseTypes = {"Perpetual", "Subscription", "Trial"};
        JComboBox<String> licenseTypeComboBox = new JComboBox<>(licenseTypes);
        JTextField licenseKeyField = new JTextField(20);
        JFormattedTextField expirationDateField = new
JFormattedTextField(dateFormatter);
        expirationDateField.setColumns(20); // Ensure proper size for date field

        // --- Dialog Content Panel ---
        JPanel panel = new JPanel(new GridBagLayout());
        panel.setBorder(new EmptyBorder(10, 10, 10, 10)); // Padding for the
dialog content
        GridBagConstraints gbc = new GridBagConstraints();
        gbc.insets = new Insets(5, 5, 5, 5); // Spacing between components
        gbc.fill = GridBagConstraints.HORIZONTAL;

        gbc.gridx = 0; gbc.gridy = 0; panel.add(new JLabel("Software Name:"),
gbc);
```

```
        gbc.gridx = 1; gbc.gridy = 0; panel.add(softwareNameField, gbc);

        gbc.gridx = 0; gbc.gridy = 1; panel.add(new JLabel("License Type:"),
gbc);
        gbc.gridx = 1; gbc.gridy = 1; panel.add(licenseTypeComboBox, gbc);

        gbc.gridx = 0; gbc.gridy = 2; panel.add(new JLabel("License Key:"), gbc);
        gbc.gridx = 1; gbc.gridy = 2; panel.add(licenseKeyField, gbc);

        gbc.gridx = 0; gbc.gridy = 3; panel.add(new JLabel("Expiration Date
(YYYY-MM-DD):"), gbc);
        gbc.gridx = 1; gbc.gridy = 3; panel.add(expirationDateField, gbc);

        int result = JOptionPane.showConfirmDialog(this, panel, "Add New
License",
                JOptionPane.OK_CANCEL_OPTION, JOptionPane.PLAIN_MESSAGE); // Use
PLAIN_MESSAGE for no icon

        if (result == JOptionPane.OK_OPTION) {
            try {
                Date expiryDate =
dateFormatter.parse(expirationDateField.getText());
                License newLicense = new License(
                    java.util.UUID.randomUUID().toString(),
                    softwareNameField.getText(),
                    (String) licenseTypeComboBox.getSelectedItem(),
                    licenseKeyField.getText(),
                    expiryDate
                );
                licenseManager.addLicense(newLicense);
                JOptionPane.showMessageDialog(this, "License added
successfully!");
                refreshTable();
            } catch (Exception ex) {
                JOptionPane.showMessageDialog(this, "Invalid date format or other
error: " + ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
            }
        }
    }

    private void deleteSelectedLicense() {
        int selected = table.getSelectedRow();
        if (selected >= 0) {
            int confirm = JOptionPane.showConfirmDialog(this, "Are you sure you
want to delete this license?", "Confirm Delete", JOptionPane.YES_NO_OPTION);
```

```
            if (confirm == JOptionPane.YES_OPTION) {
                String licenseId = model.getValueAt(selected, 0).toString();
                License licenseToDelete =
licenseManager.getLicenseById(licenseId);
                if (licenseToDelete != null) {
                    // Also unassign from user if assigned
                    if (licenseToDelete.getAssignedUserId() != null &&
!licenseToDelete.getAssignedUserId().isEmpty()) {
                        User assignedUser =
userManager.getUserById(licenseToDelete.getAssignedUserId());
                        if (assignedUser != null) {
                            assignedUser.getLicenseIds().remove(licenseId);

                            userManager.updateUser(assignedUser); // Ensure
user's license list is updated in DB
                        }
                    }
                    licenseManager.removeLicense(licenseId);
                    JOptionPane.showMessageDialog(this, "License deleted
successfully.");
                    refreshTable();
                }
            }
        } else {
            JOptionPane.showMessageDialog(this, "Select a license to delete.");
        }
    }

    private void assignLicense() {
        int selectedLicenseRow = table.getSelectedRow();
        if (selectedLicenseRow < 0) {
            JOptionPane.showMessageDialog(this, "Select a license to assign.",
"No License Selected", JOptionPane.WARNING_MESSAGE);
            return;
        }

        String licenseId = model.getValueAt(selectedLicenseRow, 0).toString();
        License licenseToAssign = licenseManager.getLicenseById(licenseId);

        if (licenseToAssign != null && licenseToAssign.getAssignedUserId() !=
null) {
            JOptionPane.showMessageDialog(this, "This license is already
assigned.", "License Already Assigned", JOptionPane.INFORMATION_MESSAGE);
            return;
        }
```

```
        List<User> allUsers = userManager.getAllUsers();
        if (allUsers.isEmpty()) {
            JOptionPane.showMessageDialog(this, "No users available to assign
licenses to. Please add users first.", "No Users", JOptionPane.WARNING_MESSAGE);
            return;
        }

        User[] usersArray = allUsers.toArray(new User[0]);
        JComboBox<User> userComboBox = new JComboBox<>(usersArray);
        userComboBox.setRenderer(new DefaultListCellRenderer() {
            @Override
            public Component getListCellRendererComponent(JList<?> list, Object
value, int index, boolean isSelected, boolean cellHasFocus) {
                super.getListCellRendererComponent(list, value, index,
isSelected, cellHasFocus);
                if (value instanceof User) {
                    setText(((User) value).getName()); // Display user's name
                }
                return this;
            }
        });

        JPanel panel = new JPanel(new FlowLayout(FlowLayout.CENTER, 10, 10));
        panel.add(new JLabel("Select User:"));
        panel.add(userComboBox);
        panel.setBorder(new EmptyBorder(10,10,10,10));

        int result = JOptionPane.showConfirmDialog(this, panel, "Assign License
to User",
                JOptionPane.OK_CANCEL_OPTION, JOptionPane.PLAIN_MESSAGE);

        if (result == JOptionPane.OK_OPTION && userComboBox.getSelectedItem() !=
null) {
            User selectedUser = (User) userComboBox.getSelectedItem();

            // Check if the license is already in the user's list (though it
should be null at this point for reassignment)
            if (selectedUser.getLicenseIds().contains(licenseId)) {
                JOptionPane.showMessageDialog(this, "User already has this
license assigned.", "Assignment Redundant", JOptionPane.INFORMATION_MESSAGE);
                return;
            }

            selectedUser.assignLicense(licenseId);
```

```
            licenseToAssign.setAssignedUserId(selectedUser.getUserId());

            userManager.updateUser(selectedUser); // Update user in DB
            licenseManager.updateLicense(licenseToAssign); // Update license in
DB

            // Add notification for assignment
            notificationManager.addNotification(selectedUser.getUserId(),
licenseId,
                "License '" + licenseToAssign.getSoftwareName() + "' has been
assigned to you.");

            JOptionPane.showMessageDialog(this, "License assigned successfully to
" + selectedUser.getName() + ".");
            refreshTable();
        }
    }

    private void unassignLicense() {
        int selected = table.getSelectedRow();
        if (selected >= 0) {
            String licenseId = model.getValueAt(selected, 0).toString();
            License license = licenseManager.getLicenseById(licenseId);

            if (license != null && license.getAssignedUserId() != null &&
!license.getAssignedUserId().isEmpty()) {
                User assignedUser =
userManager.getUserById(license.getAssignedUserId());
                if (assignedUser != null) {
                    int confirm = JOptionPane.showConfirmDialog(this,
                        "Are you sure you want to unassign '" +
license.getSoftwareName() + "' from " + assignedUser.getName() + "?",
                        "Confirm Unassign", JOptionPane.YES_NO_OPTION);

                    if (confirm == JOptionPane.YES_OPTION) {
                        assignedUser.getLicenseIds().remove(licenseId); // Remove
license from user's list
                        license.setAssignedUserId(null); // Set license's
assigned user to null

                        userManager.updateUser(assignedUser); // Update user in
DB
                        licenseManager.updateLicense(license); // Update license
in DB
```

```
                        // Add notification for unassignment
                        notificationManager.addNotification(assignedUser.getUserI
d(), licenseId,
                            "License '" + license.getSoftwareName() + "' has been
unassigned from your profile.");

                        JOptionPane.showMessageDialog(this, "License unassigned
successfully.");
                        refreshTable();
                    }
                }
            } else {
                JOptionPane.showMessageDialog(this, "Selected license is not
assigned to any user.", "Not Assigned", JOptionPane.INFORMATION_MESSAGE);
            }
        } else {
            JOptionPane.showMessageDialog(this, "Select a license to unassign.",
"No License Selected", JOptionPane.WARNING_MESSAGE);
        }
    }

    private void editLicenseDialog() {
        int selectedRow = table.getSelectedRow();
        if (selectedRow < 0) {
            JOptionPane.showMessageDialog(this, "Select a license to edit.", "No
License Selected", JOptionPane.WARNING_MESSAGE);
            return;
        }

        String licenseId = model.getValueAt(selectedRow, 0).toString();
        License existingLicense = licenseManager.getLicenseById(licenseId);

        if (existingLicense == null) {
            JOptionPane.showMessageDialog(this, "License not found.", "Error",
JOptionPane.ERROR_MESSAGE);
            return;
        }

        JTextField softwareNameField = new
JTextField(existingLicense.getSoftwareName(), 20);
        String[] licenseTypes = {"Perpetual", "Subscription", "Trial"};
        JComboBox<String> licenseTypeComboBox = new JComboBox<>(licenseTypes);
        licenseTypeComboBox.setSelectedItem(existingLicense.getLicenseType());
        JTextField licenseKeyField = new
JTextField(existingLicense.getLicenseKey(), 20);
```

```
        JFormattedTextField expirationDateField = new
JFormattedTextField(dateFormatter);
        expirationDateField.setValue(existingLicense.getExpirationDate()); // Set
existing date
        expirationDateField.setColumns(20);

        JPanel panel = new JPanel(new GridBagLayout());
        panel.setBorder(new EmptyBorder(10, 10, 10, 10));
        GridBagConstraints gbc = new GridBagConstraints();
        gbc.insets = new Insets(5, 5, 5, 5);
        gbc.fill = GridBagConstraints.HORIZONTAL;

        gbc.gridx = 0; gbc.gridy = 0; panel.add(new JLabel("Software Name:"),
gbc);
        gbc.gridx = 1; gbc.gridy = 0; panel.add(softwareNameField, gbc);

        gbc.gridx = 0; gbc.gridy = 1; panel.add(new JLabel("License Type:"),
gbc);
        gbc.gridx = 1; gbc.gridy = 1; panel.add(licenseTypeComboBox, gbc);

        gbc.gridx = 0; gbc.gridy = 2; panel.add(new JLabel("License Key:"), gbc);
        gbc.gridx = 1; gbc.gridy = 2; panel.add(licenseKeyField, gbc);

        gbc.gridx = 0; gbc.gridy = 3; panel.add(new JLabel("Expiration Date
(YYYY-MM-DD):"), gbc);
        gbc.gridx = 1; gbc.gridy = 3; panel.add(expirationDateField, gbc);

        int result = JOptionPane.showConfirmDialog(this, panel, "Edit License",
                JOptionPane.OK_CANCEL_OPTION, JOptionPane.PLAIN_MESSAGE);

        if (result == JOptionPane.OK_OPTION) {
            try {
                Date expiryDate =
dateFormatter.parse(expirationDateField.getText());
                existingLicense.setSoftwareName(softwareNameField.getText()); //
Assuming setters exist in License model
                existingLicense.setLicenseType((String)
licenseTypeComboBox.getSelectedItem()); // Assuming setters exist

                existingLicense.setExpirationDate(expiryDate); // Assuming setter
exists

                licenseManager.updateLicense(existingLicense);
                JOptionPane.showMessageDialog(this, "License updated
successfully!");
```

```
                refreshTable();
            } catch (Exception ex) {
                JOptionPane.showMessageDialog(this, "Invalid date format or other
error: " + ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
            }
        }
    }
}
```

Userpanel.java:

- **Purpose**: Manages user data through a tabular view, allowing users to add, delete, update, and view license profiles.
- **Key Properties/Methods**:
  - `Userpanel(Licensemanager lm, Usermanager um, Notificationmanager nm)`: Constructor.
  - `JTable` and `DefaultTableModel`: Displays user data.
  - `refreshTable()`: Populates the table with current user data, including the count of assigned licenses.
  - `addUserDialog()`: Displays a dialog to add a new user, including a file chooser for an image path.
  - `deleteSelectedUser()`: Deletes the selected user.
  - `viewLicenseProfile()`: Opens a `Licenseprofiledialog` for the selected user.
  - `updateUserDialog()`: Displays a dialog to update an existing user's details.

Code:

```
package ui;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import Manager.*;
import Model.*;
import java.awt.*;
import java.io.File;
import java.util.List;

public class Userpanel extends JFrame {
    private Licensemanager licenseManager;
    private Usermanager userManager;
    private Notificationmanager notificationManager;
    private JTable table;
    private DefaultTableModel model;
```

```java
    public Userpanel(Licensemanager lm, Usermanager um, Notificationmanager nm) {
        this.licenseManager = lm;
        this.userManager = um;
        this.notificationManager = nm;

        setTitle("User Management");
        setSize(700, 400); // Keep increased width for "Assigned Licenses" column
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

        // Table setup
        model = new DefaultTableModel(new String[]{"ID", "Name", "Email",
"Assigned Licenses"}, 0);
        table = new JTable(model);
        // Make table headers also look better with Nimbus
        table.getTableHeader().setFont(new Font("Segoe UI", Font.BOLD, 12));
        table.getTableHeader().setBackground(new Color(60, 90, 150)); // Darker
blue header
        table.getTableHeader().setForeground(Color.WHITE); // White text for
header
        table.setRowHeight(25); // Increase row height for better readability
        table.setFillsViewportHeight(true); // Table fills the viewport height in
scrollpane


        refreshTable();

        // Buttons using the new styling method
        JButton addBtn = createStyledButton("Add User");
        JButton deleteBtn = createStyledButton("Delete User");
        JButton viewProfileBtn = createStyledButton("View License Profile");
        JButton updateBtn = createStyledButton("Update User");

        // Action Listeners
        addBtn.addActionListener(e -> addUserDialog());
        deleteBtn.addActionListener(e -> deleteSelectedUser());
        viewProfileBtn.addActionListener(e -> viewLicenseProfile());
        updateBtn.addActionListener(e -> updateUserDialog());

        // Button Panel setup - explicitly use FlowLayout for horizontal
arrangement
        JPanel btnPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 15, 10));
// Center alignment, 10px horizontal/vertical gap
        btnPanel.setBackground(new Color(240, 248, 255)); // Light background for
button panel
```

```
        btnPanel.add(addBtn);
        btnPanel.add(deleteBtn);
        btnPanel.add(viewProfileBtn);
        btnPanel.add(updateBtn);

        // Frame Layout
        add(new JScrollPane(table), BorderLayout.CENTER);
        add(btnPanel, BorderLayout.SOUTH);

        setVisible(true);

        // Force UI update to ensure L&F is applied to all components
        SwingUtilities.updateComponentTreeUI(this);
    }


    private JButton createStyledButton(String text) {
        JButton button = new JButton(text);
        button.setBackground(new Color(60, 90, 150)); // Dark blue background
        button.setForeground(Color.WHITE); // White text
        button.setFont(new Font("Segoe UI", Font.BOLD, 12)); // Bold font
        button.setFocusPainted(false); // Remove focus border
        button.setBorder(BorderFactory.createEmptyBorder(10, 20, 10, 20)); //
Padding
        button.setCursor(new Cursor(Cursor.HAND_CURSOR)); // Hand cursor on hover

        // Add hover effect (optional, but enhances attractiveness)
        button.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseEntered(java.awt.event.MouseEvent evt) {
                button.setBackground(new Color(80, 110, 180)); // Lighter blue on
hover
            }
            public void mouseExited(java.awt.event.MouseEvent evt) {
                button.setBackground(new Color(60, 90, 150)); // Original color
on exit
            }
        });
        return button;
    }

    private void refreshTable() {
        model.setRowCount(0);
        for (User u : userManager.getAllUsers()) {
            int licenseCount = u.getLicenseIds().size();
```

```
            model.addRow(new Object[]{u.getUserId(), u.getName(), u.getEmail(),
licenseCount});
        }
    }

    private void addUserDialog() {
        JTextField idField = new JTextField();
        JTextField nameField = new JTextField();
        JTextField emailField = new JTextField();
        JTextField imagePathField = new JTextField();
        imagePathField.setEditable(false);
        JButton browseBtn = new JButton("Browse");

        browseBtn.addActionListener(e -> {
            JFileChooser chooser = new JFileChooser();
            chooser.setFileSelectionMode(JFileChooser.FILES_ONLY); // Ensure only
files can be selected
            int result = chooser.showOpenDialog(this);
            if (result == JFileChooser.APPROVE_OPTION) {
                File selected = chooser.getSelectedFile();
                imagePathField.setText(selected.getAbsolutePath());
            }
        });

        JPanel panel = new JPanel(new GridLayout(4, 3, 5, 5)); // Added gaps
        panel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10)); //
Padding for dialog content
        panel.add(new JLabel("User ID:")); panel.add(idField); panel.add(new
JLabel(""));
        panel.add(new JLabel("Name:")); panel.add(nameField); panel.add(new
JLabel(""));
        panel.add(new JLabel("Email:")); panel.add(emailField); panel.add(new
JLabel(""));
        panel.add(new JLabel("Image Path:")); panel.add(imagePathField);
panel.add(browseBtn);

        int result = JOptionPane.showConfirmDialog(this, panel, "Add User",
JOptionPane.OK_CANCEL_OPTION, JOptionPane.PLAIN_MESSAGE); // Use 'this' as parent
        if (result == JOptionPane.OK_OPTION) {
            // Basic validation
            if (idField.getText().isEmpty() || nameField.getText().isEmpty() ||
emailField.getText().isEmpty()) {
                JOptionPane.showMessageDialog(this, "All fields (except Image
Path) must be filled.", "Input Error", JOptionPane.ERROR_MESSAGE);
                return;
```

```
            }
            // Check for duplicate ID
            if (userManager.getUserById(idField.getText()) != null) {
                JOptionPane.showMessageDialog(this, "User with this ID already
exists. Please use a unique ID.", "Duplicate ID", JOptionPane.WARNING_MESSAGE);
                return;
            }
            User user = new User(idField.getText(), nameField.getText(),
emailField.getText(), imagePathField.getText());
            userManager.addUser(user);
            refreshTable();
            JOptionPane.showMessageDialog(this, "User added successfully!",
"Success", JOptionPane.INFORMATION_MESSAGE);
        }
    }

    private void deleteSelectedUser() {
        int selected = table.getSelectedRow();
        if (selected >= 0) {
            String userId = model.getValueAt(selected, 0).toString();
            int confirm = JOptionPane.showConfirmDialog(this, "Are you sure you
want to delete user: " + model.getValueAt(selected, 1).toString() + "?", "Confirm
Delete", JOptionPane.YES_NO_OPTION);
            if (confirm == JOptionPane.YES_OPTION) {
                userManager.removeUser(userId);
                refreshTable();
                JOptionPane.showMessageDialog(this, "User deleted successfully!",
"Success", JOptionPane.INFORMATION_MESSAGE);
            }
        } else {
            JOptionPane.showMessageDialog(this, "Select a user to delete.");
        }
    }

    private void viewLicenseProfile() {
        int selected = table.getSelectedRow();
        if (selected >= 0) {
            String userId = model.getValueAt(selected, 0).toString();
            User user = userManager.getUserById(userId);
            if (user != null) {
                List<License> allLicenses = licenseManager.getAllLicenses();
                new Licenseprofiledialog(this, user, allLicenses);
            }
        } else {
```

```
            JOptionPane.showMessageDialog(this, "Select a user to view their
license profile.");
        }
    }

    private void updateUserDialog() {
        int selected = table.getSelectedRow();
        if (selected >= 0) {
            String userId = model.getValueAt(selected, 0).toString();
            User currentUser = userManager.getUserById(userId);

            if (currentUser != null) {
                JTextField nameField = new JTextField(currentUser.getName());
                JTextField emailField = new JTextField(currentUser.getEmail());
                JTextField imagePathField = new
JTextField(currentUser.getimagepath());
                imagePathField.setEditable(false);
                JButton browseBtn = new JButton("Browse");

                browseBtn.addActionListener(e -> {
                    JFileChooser chooser = new JFileChooser();
                    chooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
                    int result = chooser.showOpenDialog(this);
                    if (result == JFileChooser.APPROVE_OPTION) {
                        File selectedFile = chooser.getSelectedFile();
                        imagePathField.setText(selectedFile.getAbsolutePath());
                    }
                });

                JPanel panel = new JPanel(new GridLayout(3, 3, 5, 5)); // Added
gaps
                panel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));
// Padding for dialog content
                panel.add(new JLabel("Name:")); panel.add(nameField);
panel.add(new JLabel(""));
                panel.add(new JLabel("Email:")); panel.add(emailField);
panel.add(new JLabel(""));
                panel.add(new JLabel("Image Path:")); panel.add(imagePathField);
panel.add(browseBtn);

                int result = JOptionPane.showConfirmDialog(this, panel, "Update
User", JOptionPane.OK_CANCEL_OPTION, JOptionPane.PLAIN_MESSAGE);
                if (result == JOptionPane.OK_OPTION) {
                    if (nameField.getText().isEmpty() ||
emailField.getText().isEmpty()) {
```

```
                        JOptionPane.showMessageDialog(this, "Name and Email
fields must be filled.", "Input Error", JOptionPane.ERROR_MESSAGE);
                        return;
                    }
                    User updatedUser = new User(userId, nameField.getText(),
emailField.getText(), imagePathField.getText());
                    userManager.updateUser(updatedUser);
                    refreshTable();
                    JOptionPane.showMessageDialog(this, "User updated
successfully!", "Success", JOptionPane.INFORMATION_MESSAGE);
                }
            }
        } else {
            JOptionPane.showMessageDialog(this, "Select a user to update.");
        }
    }
}
```

➢ Licenseprofiledialog.java:

- **Purpose**: Displays a detailed profile of a user, including their assigned licenses and an optional profile picture.
- **Key Properties/Methods**:
  - `Licenseprofiledialog(JFrame parent, User user, List<License> allLicenses)`: Constructor, taking the parent frame, the `User` object, and a list of all licenses to cross-reference.
  - Dynamically generates text showing user details and their assigned licenses.
  - Attempts to load and display a user's profile image from the specified `imagepath`.

Code:

```
package ui;

import javax.swing.*;
import java.awt.*;
import java.text.SimpleDateFormat;
import java.util.List;
import Model.*;

public class Licenseprofiledialog extends JDialog {
    public Licenseprofiledialog(JFrame parent, User user, List<License>
allLicenses) {
```

```
        super(parent, "License Profile for " + user.getName(), true);
        setSize(500, 400);
        setLayout(new BorderLayout());

        StringBuilder profileText = new StringBuilder();
        profileText.append("User: ").append(user.getName()).append("\n");
        profileText.append("Email: ").append(user.getEmail()).append("\n\n");

        List<String> licenseIds = user.getLicenseIds();
        if (licenseIds.isEmpty()) {
            profileText.append("No licenses assigned.\n");
        } else {
            profileText.append("Assigned Licenses:\n");
            SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
            for (String licenseId : licenseIds) {
                for (License l : allLicenses) {
                    if (l.getId().equals(licenseId)) {
                        profileText.append("- Software:
").append(l.getSoftwareName()).append("\n")
                                        .append("  Type:
").append(l.getLicenseType()).append("\n")
                                        .append("  Key:
").append(l.getLicenseKey()).append("\n")
                                        .append("  Expiry:
").append(sdf.format(l.getExpirationDate())).append("\n\n");
                    }
                }
            }
        }

        if (user.getimagepath() != null && !user.getimagepath().isEmpty()) {
            try {
                ImageIcon icon = new ImageIcon(user.getimagepath());
                Image scaled = icon.getImage().getScaledInstance(100, 100,
Image.SCALE_SMOOTH);
                JLabel picLabel = new JLabel(new ImageIcon(scaled));
                picLabel.setBorder(BorderFactory.createTitledBorder("Profile
Picture"));
                add(picLabel, BorderLayout.WEST);
            } catch (Exception e) {
                System.out.println("Could not load image: " + e.getMessage());
            }
        }

        JTextArea textArea = new JTextArea(profileText.toString());
```

```
        textArea.setEditable(false);
        textArea.setFont(new Font("Monospaced", Font.PLAIN, 12));
        add(new JScrollPane(textArea), BorderLayout.CENTER);

        JButton close = new JButton("Close");
        close.addActionListener(e -> dispose());
        add(close, BorderLayout.SOUTH);

        setLocationRelativeTo(parent);
        setVisible(true);
}
}
```

Notificationpanel.java:

- **Purpose**: Displays a list of all system notifications in a tabular format.
- **Key Properties/Methods**:
  - `Notificationpanel(Notificationmanager nm, Usermanager um)`: Constructor.
  - `JTable` and `DefaultTableModel`: Displays notification data.
  - `refreshTable()`: Populates the table with notifications, resolving user names for display.
  - Buttons for "Refresh" and "Clear All Notifications".

Code:

```
package ui;

import Manager.Notificationmanager;
import Manager.Usermanager;
import Model.Notification;
import Model.User;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.text.SimpleDateFormat;
import java.util.Date;

public class Notificationpanel extends JFrame {
    private Notificationmanager notificationManager;
    private Usermanager userManager;
    private JTable table;
    private DefaultTableModel model;
```

```java
    public Notificationpanel(Notificationmanager nm, Usermanager um) {
        this.notificationManager = nm;
        this.userManager = um;

        setTitle("Notifications");
        setSize(700, 500);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

        model = new DefaultTableModel(new String[]{"ID", "User", "License ID",
"Message", "Sent At"}, 0);
        table = new JTable(model);
        // Style table header
        table.getTableHeader().setFont(new Font("Segoe UI", Font.BOLD, 12));
        table.getTableHeader().setBackground(new Color(60, 90, 150)); // Darker
blue header
        table.getTableHeader().setForeground(Color.WHITE); // White text for
header
        table.setRowHeight(25); // Increase row height for better readability
        table.setFillsViewportHeight(true); // Table fills the viewport height in
scrollpane

        refreshTable();

        JButton refreshBtn = createStyledButton("Refresh");
        JButton clearAllBtn = createStyledButton("Clear All Notifications");

        refreshBtn.addActionListener(e -> refreshTable());
        clearAllBtn.addActionListener(e -> {
            int confirm = JOptionPane.showConfirmDialog(this, "Are you sure you
want to clear all notifications?", "Confirm Clear", JOptionPane.YES_NO_OPTION);
            if (confirm == JOptionPane.YES_OPTION) {
                notificationManager.clearAllNotifications();
                refreshTable();
                JOptionPane.showMessageDialog(this, "All notifications cleared!",
"Success", JOptionPane.INFORMATION_MESSAGE);
            }
        });

        JPanel btnPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 15, 10));
// Added gaps and center alignment
        btnPanel.setBackground(new Color(240, 248, 255)); // Light background for
button panel
        btnPanel.add(refreshBtn);
```

```
        btnPanel.add(clearAllBtn);

        add(new JScrollPane(table), BorderLayout.CENTER);
        add(btnPanel, BorderLayout.SOUTH);

        setVisible(true);
        SwingUtilities.updateComponentTreeUI(this); // Ensure L&F update
    }


    private JButton createStyledButton(String text) {
        JButton button = new JButton(text);
        button.setBackground(new Color(60, 90, 150)); // Dark blue background
        button.setForeground(Color.WHITE); // White text
        button.setFont(new Font("Segoe UI", Font.BOLD, 12)); // Bold font
        button.setFocusPainted(false); // Remove focus border
        button.setBorder(BorderFactory.createEmptyBorder(10, 20, 10, 20)); //
Padding
        button.setCursor(new Cursor(Cursor.HAND_CURSOR)); // Hand cursor on hover

        // Add hover effect
        button.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseEntered(java.awt.event.MouseEvent evt) {
                button.setBackground(new Color(80, 110, 180)); // Lighter blue on
hover
            }
            public void mouseExited(java.awt.event.MouseEvent evt) {
                button.setBackground(new Color(60, 90, 150)); // Original color
on exit
            }
        });
        return button;
    }

    private void refreshTable() {
        model.setRowCount(0);
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm");
        for (Notification n : notificationManager.getAllNotifications()) {
            String userName = "N/A";
            User user = userManager.getUserById(n.getUserId());
            if (user != null) {
                userName = user.getName();
            }
            model.addRow(new Object[]{n.getId(), userName, n.getLicenseId(),
n.getMessage(), sdf.format(n.getSentAt())});
```

```
        }
    }
}
```

Reportpanel.java:

- **Purpose**: Generates and displays various reports related to licenses and users, and can also list installed applications on the system.
- **Key Properties/Methods**:
  - `Reportpanel(Licensemanager lm, Usermanager um)`: Constructor.
  - `JTextArea reportArea`: Displays the generated reports.
  - `showUsageReport()`: Generates a report on license assignments.
  - `showExpiryReport()`: Lists licenses expiring within 30 days.
  - `showUserLicenseReport()`: Provides a detailed report of licenses assigned to each user.
  - `sendReminders()`: Triggers the `licenseManager` to send expiry reminders.
  - `showInstalledApplications()`: Uses `InstalledAppDetector` to list applications installed on the system (Windows-specific).

Code:

```
package ui;

import javax.swing.*;
import Manager.*;
import Model.*;
import java.awt.*;
import java.text.SimpleDateFormat;
import java.util.List;
import Utils.InstalledAppDetector; // Added import for InstalledAppDetector

public class Reportpanel extends JFrame {
    private Licensemanager licenseManager;
    private Usermanager userManager;
    private JTextArea reportArea;

    public Reportpanel(Licensemanager lm, Usermanager um) {
        this.licenseManager = lm;
        this.userManager = um;

        setTitle("Reports");
        setSize(600, 400);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
```

```
        reportArea = new JTextArea();
        reportArea.setEditable(false);
        reportArea.setFont(new Font("Monospaced", Font.PLAIN, 12));
        reportArea.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10)); //
Padding

        // *** ADD THESE TWO LINES FOR VERTICAL DISPLAY ***
        reportArea.setLineWrap(true); // Enables line wrapping
        reportArea.setWrapStyleWord(true); // Wraps at word boundaries

        JButton usageBtn = createStyledButton("License Usage Report");
        JButton expiryBtn = createStyledButton("License Expiry Report");
        JButton userLicenseBtn = createStyledButton("User License Report");
        JButton reminderBtn = createStyledButton("Send Expiry Reminders");
        JButton installedAppsBtn = createStyledButton("Show Installed Apps");

        usageBtn.addActionListener(e -> showUsageReport());
        expiryBtn.addActionListener(e -> showExpiryReport());
        userLicenseBtn.addActionListener(e -> showUserLicenseReport());
        reminderBtn.addActionListener(e -> sendReminders());
        installedAppsBtn.addActionListener(e -> showInstalledApplications());

        JPanel buttonPanel = new JPanel(new GridLayout(0, 1, 10, 10));
        buttonPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));
        buttonPanel.setBackground(new Color(240, 248, 255));
        buttonPanel.add(usageBtn);
        buttonPanel.add(expiryBtn);
        buttonPanel.add(userLicenseBtn);
        buttonPanel.add(reminderBtn);
        buttonPanel.add(installedAppsBtn);

        setLayout(new BorderLayout());
        add(new JScrollPane(reportArea), BorderLayout.CENTER);
        add(buttonPanel, BorderLayout.WEST);

        setVisible(true);
    }

    private void showUsageReport() {
        StringBuilder sb = new StringBuilder("License Usage Report:\n\n");
        List<License> allLicenses = licenseManager.getAllLicenses();
        if (allLicenses.isEmpty()) {
            sb.append("No licenses available.");
        } else {
```

```
            for (License l : allLicenses) {
                String assignedTo = "Unassigned";
                if (l.getAssignedUserId() != null &&
!l.getAssignedUserId().isEmpty()) {
                    User user = userManager.getUserById(l.getAssignedUserId());
                    if (user != null) {
                        assignedTo = user.getName();
                    }
                }
                sb.append("- Software: ").append(l.getSoftwareName())
                  .append(", Type: ").append(l.getLicenseType())
                  .append(", Assigned To: ").append(assignedTo).append("\n");
            }
        }
        reportArea.setText(sb.toString());
        reportArea.setCaretPosition(0);
    }

    private void showExpiryReport() {
        List<License> expiring = licenseManager.getExpiringLicenses(30);
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
        StringBuilder sb = new StringBuilder("Licenses Expiring Soon (30
days):\n\n");
        if (expiring.isEmpty()) {
            sb.append("No licenses expiring within 30 days.");
        } else {
            for (License l : expiring) {
                sb.append("- ").append(l.getSoftwareName()).append(" expires on
").append(sdf.format(l.getExpirationDate())).append("\n");
            }
        }
        reportArea.setText(sb.toString());
        reportArea.setCaretPosition(0);
    }

    private void showUserLicenseReport() {
        StringBuilder sb = new StringBuilder("User License Report:\n\n");
        List<User> allUsers = userManager.getAllUsers();
        if (allUsers.isEmpty()) {
            sb.append("No users available to report licenses.");
        } else {
            for (User u : allUsers) {
                sb.append(u.getLicenseProfile(licenseManager.getAllLicenses())).a
ppend("\n");
            }
```

```
        }
        reportArea.setText(sb.toString());
        reportArea.setCaretPosition(0);
    }

    private void sendReminders() {
        licenseManager.sendExpiryReminders(userManager.getAllUsers());
        JOptionPane.showMessageDialog(this, "Expiry reminders sent (check console
for details).");
    }

    private void showInstalledApplications() {
        List<String> installedApps = InstalledAppDetector.getInstalledApps();
        StringBuilder sb = new StringBuilder("Installed Applications:\n\n");
        if (installedApps.isEmpty()) {
            sb.append("No installed applications found or an error occurred.
(Note: This feature works on Windows only and requires WMIC.)");
        } else {
            for (String app : installedApps) {
                sb.append("- ").append(app).append("\n");
            }
        }
        reportArea.setText(sb.toString());
        reportArea.setCaretPosition(0);
    }

    private JButton createStyledButton(String text) {
        JButton button = new JButton(text);
        button.setBackground(new Color(60, 90, 150));
        button.setForeground(Color.WHITE);
        button.setFont(new Font("Segoe UI", Font.BOLD, 12));
        button.setFocusPainted(false);
        button.setBorder(BorderFactory.createEmptyBorder(10, 20, 10, 20));
        button.setCursor(new Cursor(Cursor.HAND_CURSOR));

        button.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseEntered(java.awt.event.MouseEvent evt) {
                button.setBackground(new Color(80, 110, 180));
            }
            public void mouseExited(java.awt.event.MouseEvent evt) {
                button.setBackground(new Color(60, 90, 150));
            }
        });
        return button;
    }
```

```
}
```

Settingspanel.java:

- **Purpose**: A placeholder panel for future application settings.
- **Key Properties/Methods**:
  - Currently displays a simple "Settings will go here!" message.
  - Includes a "Close" button.

Code:

```
package ui;

import javax.swing.*;
import java.awt.*;

public class Settingspanel extends JFrame {
    public Settingspanel() {
        setTitle("Application Settings");
        setSize(400, 300);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

        JPanel panel = new JPanel();
        panel.setLayout(new BorderLayout());
        panel.setBackground(new Color(240, 248, 255));

        JLabel messageLabel = new JLabel("Settings will go here!",
SwingConstants.CENTER);
        messageLabel.setFont(new Font("Serif", Font.BOLD, 20));
        messageLabel.setForeground(new Color(60, 90, 150)); // Dark blue text
        panel.add(messageLabel, BorderLayout.CENTER);

        JButton closeButton = createStyledButton("Close"); // Using the styled
button
        closeButton.addActionListener(e -> dispose());

        JPanel buttonPanel = new JPanel();
        buttonPanel.setBackground(new Color(240, 248, 255)); // Match panel
background
        buttonPanel.add(closeButton);
        panel.add(buttonPanel, BorderLayout.SOUTH);

        add(panel);
```

```java
        setVisible(true);
        SwingUtilities.updateComponentTreeUI(this);
    }


    private JButton createStyledButton(String text) {
        JButton button = new JButton(text);
        button.setBackground(new Color(60, 90, 150)); // Dark blue background
        button.setForeground(Color.WHITE); // White text
        button.setFont(new Font("Segoe UI", Font.BOLD, 12)); // Bold font
        button.setFocusPainted(false); // Remove focus border
        button.setBorder(BorderFactory.createEmptyBorder(10, 20, 10, 20)); //
Padding
        button.setCursor(new Cursor(Cursor.HAND_CURSOR)); // Hand cursor on hover

        // Add hover effect
        button.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseEntered(java.awt.event.MouseEvent evt) {
                button.setBackground(new Color(80, 110, 180)); // Lighter blue on
hover
            }
            public void mouseExited(java.awt.event.MouseEvent evt) {
                button.setBackground(new Color(60, 90, 150)); // Original color
on exit
            }
        });
        return button;
    }
}
```

- ## Utils Package:

  This package contains utility classes that provide common functionalities.

DbUtil.java:

- **Purpose**: Provides static utility methods for saving and loading application data to and from a MongoDB database.
- **Key Properties/Methods**:
  - `saveData(MongoDatabase database, List<User> users, List<License> licenses, List<Notification> notifications)`: Clears existing collections and inserts current user, license, and notification data into MongoDB.

- `loadData(MongoDatabase database)`: Retrieves all user, license, and notification data from MongoDB and returns them as an `Object[]`.

Code:

```
package Utils;

import Model.License;
import Model.Notification;
import Model.User;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import org.bson.Document;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

public class DbUtil {

    // Method to save all data to MongoDB
    public static void saveData(MongoDatabase database, List<User> users,
List<License> licenses, List<Notification> notifications) {
        // Clear existing data (optional, but good for full sync)
        database.getCollection("users").drop();
        database.getCollection("licenses").drop();
        database.getCollection("notifications").drop();

        // Save Users
        MongoCollection<Document> usersCollection =
database.getCollection("users");
        for (User user : users) {
            Document userDoc = new Document("_id", user.getUserId())
                    .append("name", user.getName())
                    .append("email", user.getEmail())
                    .append("imagepath", user.getimagepath())
                    .append("licenseIds", user.getLicenseIds()); // Store list of
license IDs
            usersCollection.insertOne(userDoc);
        }

        // Save Licenses
        MongoCollection<Document> licensesCollection =
database.getCollection("licenses");
        for (License license : licenses) {
            Document licenseDoc = new Document("_id", license.getId())
```

```
                    .append("softwareName", license.getSoftwareName())
                    .append("licenseType", license.getLicenseType())
                    .append("licenseKey", license.getLicenseKey())
                    .append("expirationDate", license.getExpirationDate())
                    .append("assignedUserId", license.getAssignedUserId()); //
Store assigned user ID
            licensesCollection.insertOne(licenseDoc);
        }

        // Save Notifications
        MongoCollection<Document> notificationsCollection =
database.getCollection("notifications");
        for (Notification notification : notifications) {
            Document notificationDoc = new Document("_id", notification.getId())
                    .append("userId", notification.getUserId())
                    .append("licenseId", notification.getLicenseId())
                    .append("message", notification.getMessage())
                    .append("sentAt", notification.getSentAt());
            notificationsCollection.insertOne(notificationDoc);
        }

        System.out.println("Data saved to MongoDB successfully.");
    }

    // Method to load all data from MongoDB
    public static Object[] loadData(MongoDatabase database) {
        List<User> users = new ArrayList<>();
        List<License> licenses = new ArrayList<>();
        List<Notification> notifications = new ArrayList<>();

        // Load Users
        MongoCollection<Document> usersCollection =
database.getCollection("users");
        for (Document doc : usersCollection.find()) {
            User user = new User(
                    doc.getString("_id"),
                    doc.getString("name"),
                    doc.getString("email"),
                    doc.getString("imagepath")
            );

            List<String> licenseIds = doc.getList("licenseIds", String.class);
            if (licenseIds != null) {
                for (String licenseId : licenseIds) {
                    user.assignLicense(licenseId);
```

```
                }
            }
            users.add(user);
        }

        // Load Licenses
        MongoCollection<Document> licensesCollection =
database.getCollection("licenses");
        for (Document doc : licensesCollection.find()) {
            License license = new License(
                    doc.getString("_id"),
                    doc.getString("softwareName"),
                    doc.getString("licenseType"),
                    doc.getString("licenseKey"),
                    doc.getDate("expirationDate")
            );
            license.setAssignedUserId(doc.getString("assignedUserId")); // Set
assigned user ID
            licenses.add(license);
        }

        // Load Notifications
        MongoCollection<Document> notificationsCollection =
database.getCollection("notifications");
        for (Document doc : notificationsCollection.find()) {
            Notification notification = new Notification(
                    doc.getString("_id"),
                    doc.getString("userId"),
                    doc.getString("licenseId"),
                    doc.getString("message"),
                    doc.getDate("sentAt")
            );
            notifications.add(notification);
        }

        System.out.println("Data loaded from MongoDB successfully.");
        return new Object[]{users, licenses, notifications};
    }
}
```

InstalledAppDetector.java:

- **Purpose**: Detects and lists installed applications on a Windows operating system.
- **Key Properties/Methods**:

- `getInstalledApps()`: Executes the `wmic product get name` command on Windows to retrieve a list of installed applications.
- **Note**: This utility is platform-specific (Windows) and relies on the `wmic` command.

Code:

```
package Utils;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.List;

public class InstalledAppDetector {

    public static List<String> getInstalledApps() {
        List<String> apps = new ArrayList<>();
        try {
            // Execute Windows command to list installed apps
            Process process = Runtime.getRuntime().exec("wmic product get name");

            BufferedReader reader = new BufferedReader(new
InputStreamReader(process.getInputStream()));
            String line;

            while ((line = reader.readLine()) != null) {
                // Skip empty lines and headers
                if (line.trim().isEmpty() || line.toLowerCase().contains("name"))
{
                    continue;
                }
                apps.add(line.trim());
            }
            process.waitFor(); // Wait for the process to complete

        } catch (Exception e) {
            e.printStackTrace();
            // Optionally add more robust error logging or user notification
        }
        return apps;
    }

    public static void main(String[] args) {
        List<String> installedApps = getInstalledApps();
```

```
        System.out.println("Installed Applications:\n");
        for (String app : installedApps) {
            System.out.println("-" + app);
        }
    }
}
```

Main.java:

- **Purpose**: The main entry point of the entire application. It initializes the MongoDB connection, creates instances of all manager classes, and starts the `LoginPanel`.
- **Key Properties/Methods**:
  - `main(String[] args)`:
    - Sets up the MongoDB connection string and `MongoClientSettings`.
    - Establishes a connection to MongoDB.
    - Retrieves `MongoCollection` instances for users, licenses, and notifications.
    - Initializes `Notificationmanager`, `Licensemanager`, `Usermanager`, and `AuthManager`, injecting their respective dependencies.
    - Starts the application by creating a new `LoginPanel` instance.
    - Includes a shutdown hook to gracefully close the MongoDB client when the application exits.
    - Includes basic error handling for MongoDB connection issues.

Code:

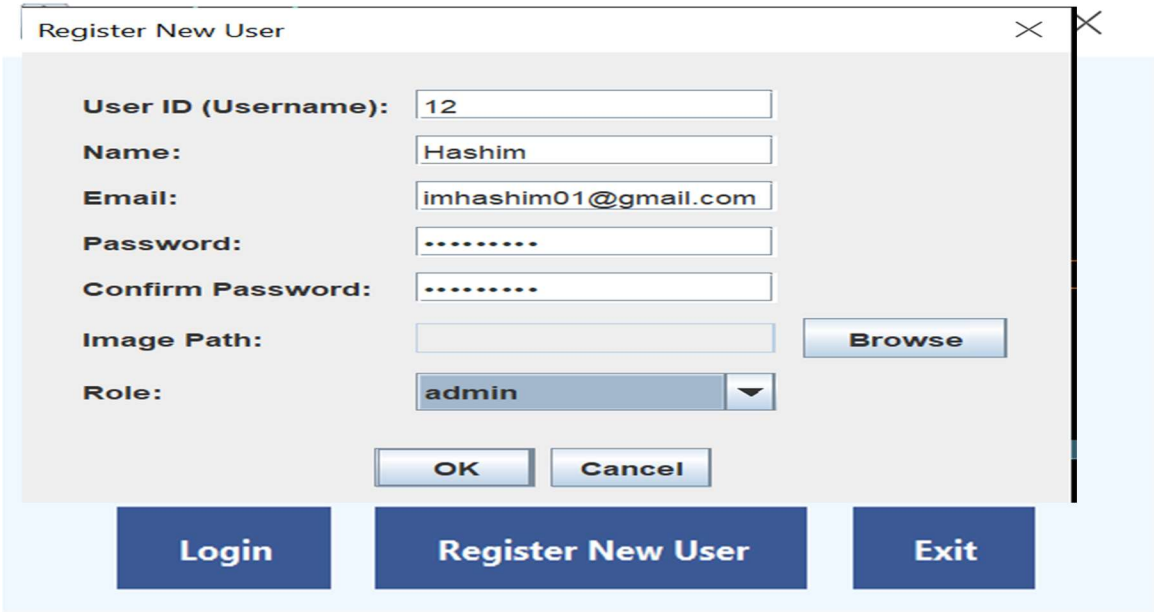
```
import com.mongodb.ConnectionString;
import com.mongodb.MongoClientSettings;
import com.mongodb.MongoException;
import com.mongodb.ServerApi;
import com.mongodb.ServerApiVersion;
import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoClients;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import javax.swing.UIManager;
import javax.swing.SwingUtilities;

import org.bson.Document;

import ui.LoginPanel;
import Manager.AuthManager;
import Manager.Licensemanager;
```

```
import Manager.Usermanager;
import Manager.Notificationmanager;
import ui.Mainmenu;
import ui.Welcomepage;

public class Main {
    public static void main(String[] args) {
        String connectionString =
"mongodb+srv://hafizmuhammadhashim05:1l3g2s4rabP6Bikf@cluster0.xhbfszv.mongodb.ne
t/?retryWrites=true&w=majority&appName=Cluster0";

        ServerApi serverApi = ServerApi.builder()
                .version(ServerApiVersion.V1)
                .build();

        MongoClientSettings settings = MongoClientSettings.builder()
                .applyConnectionString(new ConnectionString(connectionString))
                .serverApi(serverApi)
                .build();

        // Establish MongoDB connection and initialize application components
        try {
            MongoClient mongoClient = MongoClients.create(settings);
            // Get the specific database for your application
            // Replace "LicenseManagerDB" with your desired database name if
different.
            MongoDatabase database = mongoClient.getDatabase("LicenseManagerDB");

            MongoCollection<Document> licenseCollection =
database.getCollection("licenses");
            MongoCollection<Document> userCollection =
database.getCollection("users");
            MongoCollection<Document> notificationCollection =
database.getCollection("notifications");

            Notificationmanager notificationManager = new
Notificationmanager(notificationCollection);
            Licensemanager licenseManager = new Licensemanager(licenseCollection,
notificationManager);
            Usermanager userManager = new Usermanager(userCollection);
             AuthManager authManager = new AuthManager(userManager);


            new LoginPanel(authManager, licenseManager, userManager,
notificationManager);
```

```
            Runtime.getRuntime().addShutdownHook(new Thread(() -> {
                if (mongoClient != null) {
                    mongoClient.close();
                    System.out.println("MongoDB client closed gracefully.");
                }
            }));

        } catch (MongoException e) {
            System.err.println("Error connecting to MongoDB or initializing
managers: " + e.getMessage());
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

Screenshots of Output

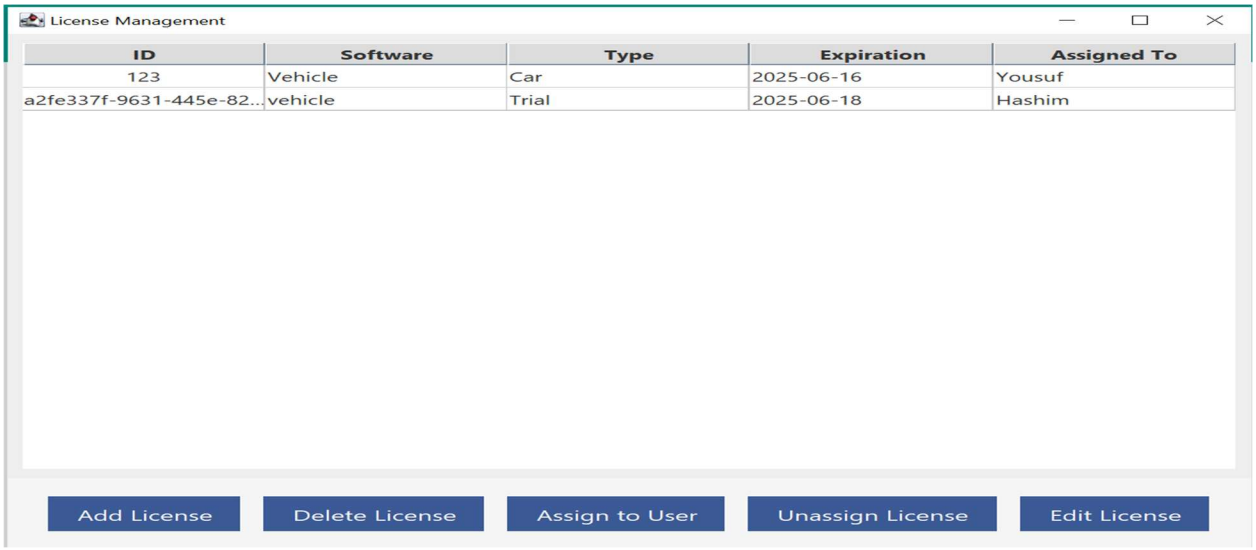
LoginPanel:



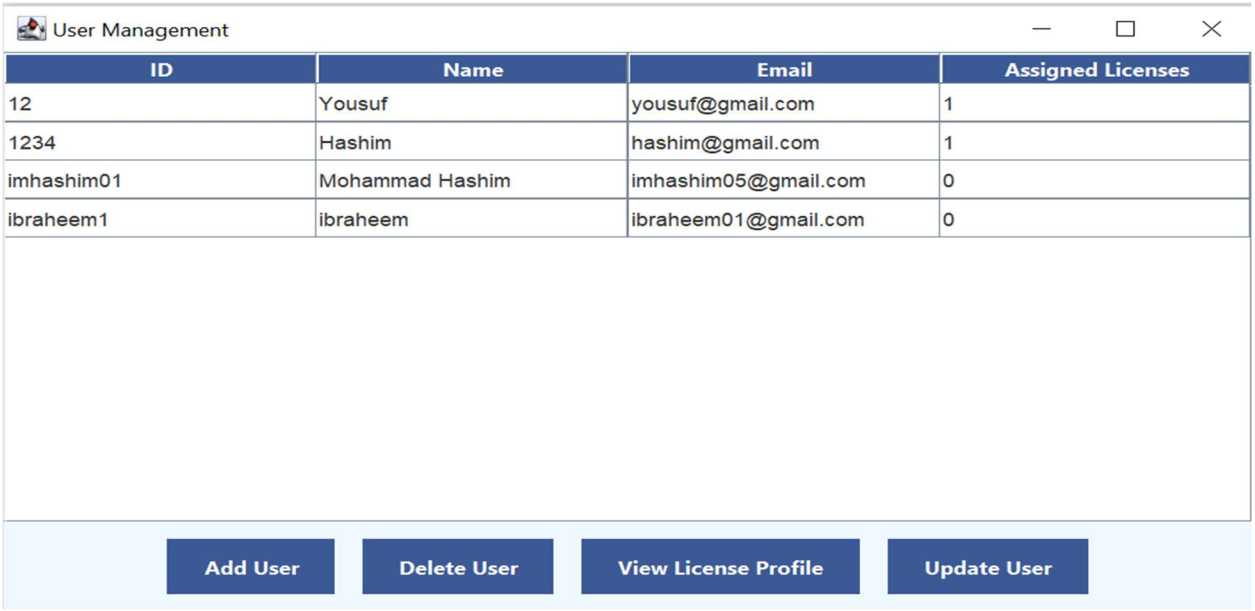
WelcomePage:

MainMenu:

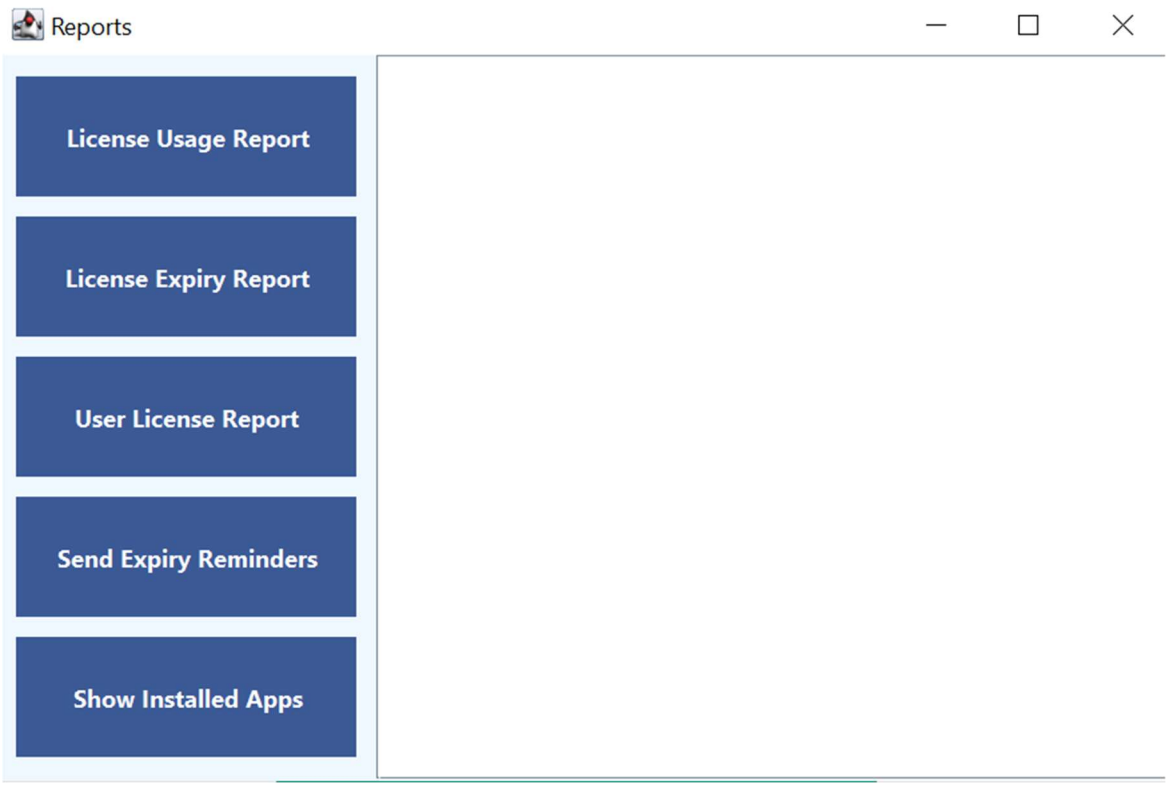
LicenseManagement:

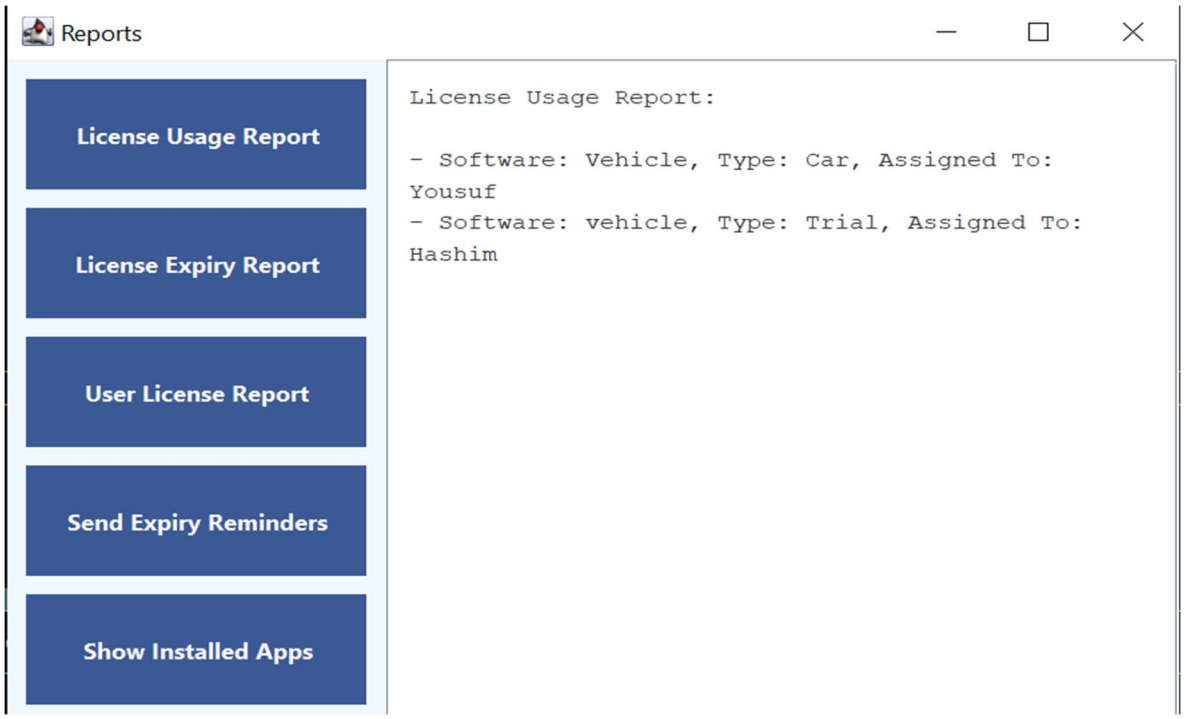


UserManagement:

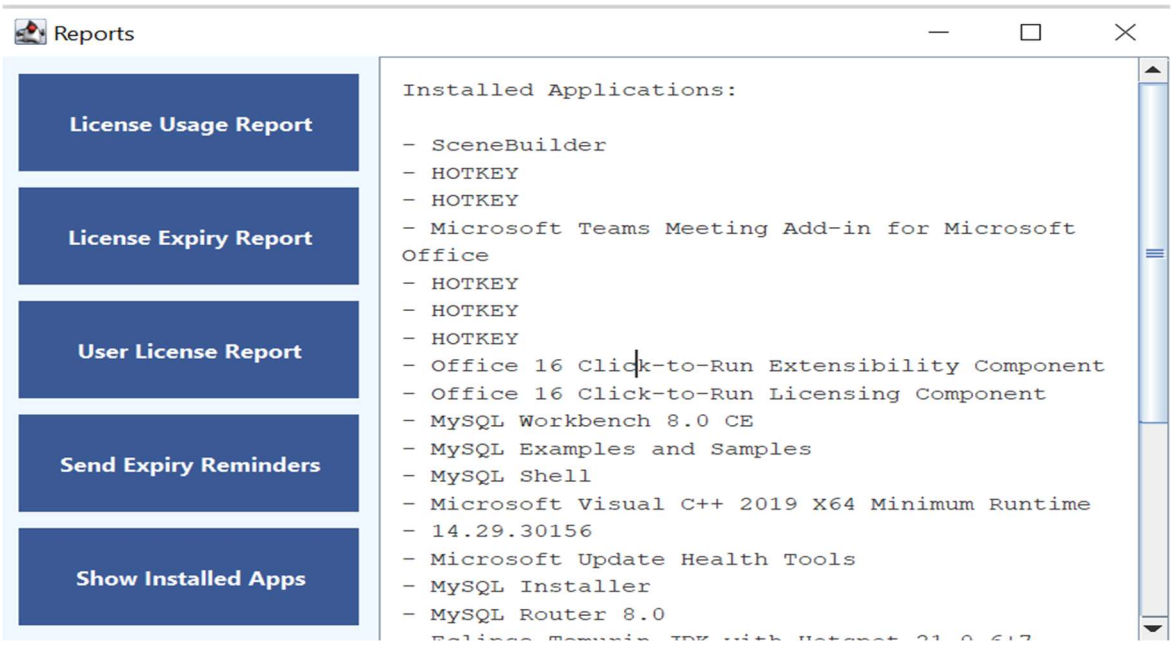


Reports:

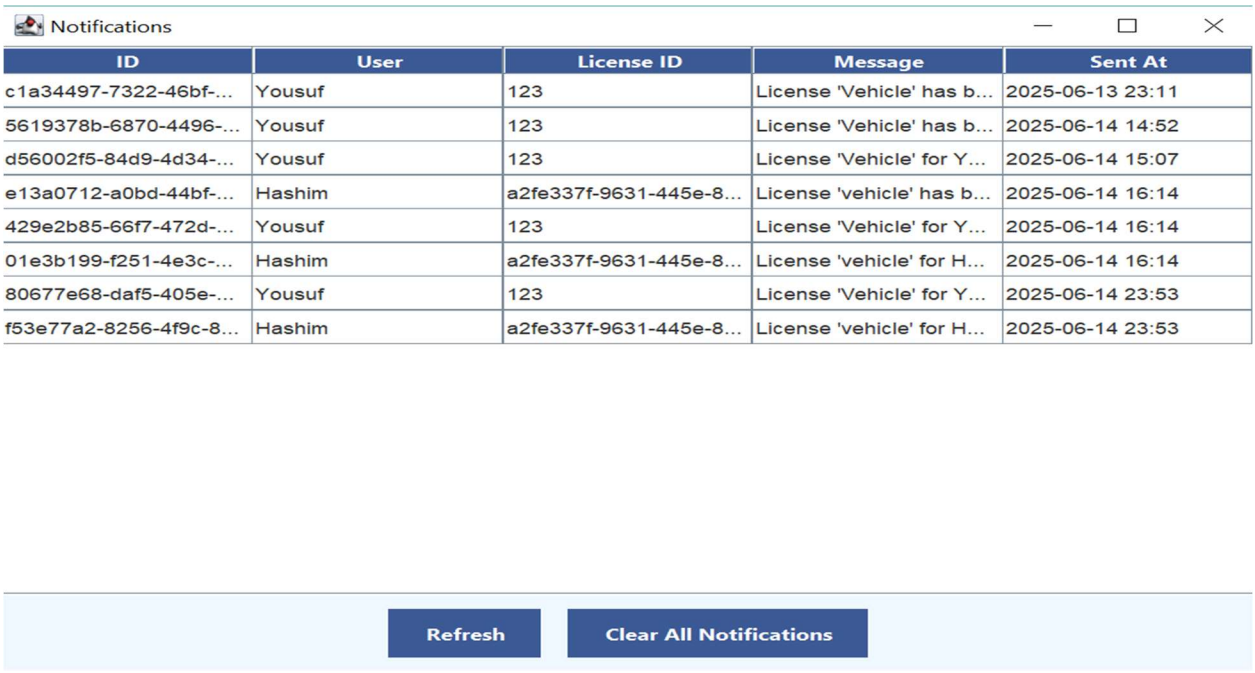




Show Installed Apps:



Notifications:



ID	User	License ID	Message	Sent At
c1a34497-7322-46bf-...	Yousuf	123	License 'Vehicle' has b...	2025-06-13 23:11
5619378b-6870-4496-...	Yousuf	123	License 'Vehicle' has b...	2025-06-14 14:52
d56002f5-84d9-4d34-...	Yousuf	123	License 'Vehicle' for Y...	2025-06-14 15:07
e13a0712-a0bd-44bf-...	Hashim	a2fe337f-9631-445e-8...	License 'vehicle' has b...	2025-06-14 16:14
429e2b85-66f7-472d-...	Yousuf	123	License 'Vehicle' for Y...	2025-06-14 16:14
01e3b199-f251-4e3c-...	Hashim	a2fe337f-9631-445e-8...	License 'vehicle' for H...	2025-06-14 16:14
80677e68-daf5-405e-...	Yousuf	123	License 'Vehicle' for Y...	2025-06-14 23:53
f53e77a2-8256-4f9c-8...	Hashim	a2fe337f-9631-445e-8...	License 'vehicle' for H...	2025-06-14 23:53

Areas for Future Improvement:

- **Enhanced Security**: Implement stronger password hashing (e.g., BCrypt, Argon2) and consider more robust authentication mechanisms.
- **Input Validation**: More comprehensive input validation in UI components to prevent invalid data entry.
- **User Roles and Permissions**: Implement fine-grained access control based on user roles (e.g., only admins can add/delete users/licenses).
- **Notification Delivery**: Extend notification capabilities beyond console logging (e.g., email, in-app pop-ups).
- **Cross-Platform Installed Apps**: Improve `InstalledAppDetector` to support macOS and Linux.
- **Reporting Features**: Add more advanced reporting, filtering, and export options (e.g., CSV, PDF).
- **UI/UX Enhancements**: Further refine the user interface and experience, potentially using more advanced Swing components or a different UI framework if a web-based solution is considered.
- **Concurrency**: For multi-user scenarios or heavy database operations, consider implementing concurrency best practices.
- **Logging**: Implement a dedicated logging framework (e.g., Log4j, SLF4j) for better application monitoring and debugging.