PROJECT REPORT : HOSPITAL STAFF MANAGEMENT SYSTEM

CONTENTS

EXTRA CREDIT

Hashing password

ER diagram

Abstract

HSMS is a highly-developed JavaFX application for effective managing the staff operations occurring in the IT environment of the hospital. It uses Model-View-Controller (MVC) structuring that discovers a thoroughly independent View, responsible for the interface, Controller, managing business logic, and Model, responsible for handling data. This way, code maintainability also enhances, that simplifies the moments of debugging, and prepares for further improvements in the future.

This application supports multiple user roles, including admin, manager, and staff, each with role-specific access permissions and features. For instance:

Admins have full control, allowing them to manage user accounts, assign roles, and oversee all operational data.(has access to both staff and department data)

Managers are empowered to supervise team performance, allocate tasks, and generate reports.

Staff members can access their schedules and view task assignments.

Introduction

This project uses comprehensive system to manage hospital staff , using JavaFX for the front -end and MYSQL database for back -end. MVC architecture helps in distinguishing the role based access and making it easier to manage and work efficiently.

Login Nexus

To start the application, execute the `*Main.java*` file. You will be presented with a pop out box where we need to select the main java file and run the programme that redirects to login screen. Use the following credentials to log in:

- Regular user (manager):
    - Username: vyshnavi
    - Password: myjava

- Regular user (staff):
    - Username: staff
    - Password: staff
- Admin user (admin):
    - Username: admin
    - Password: admin

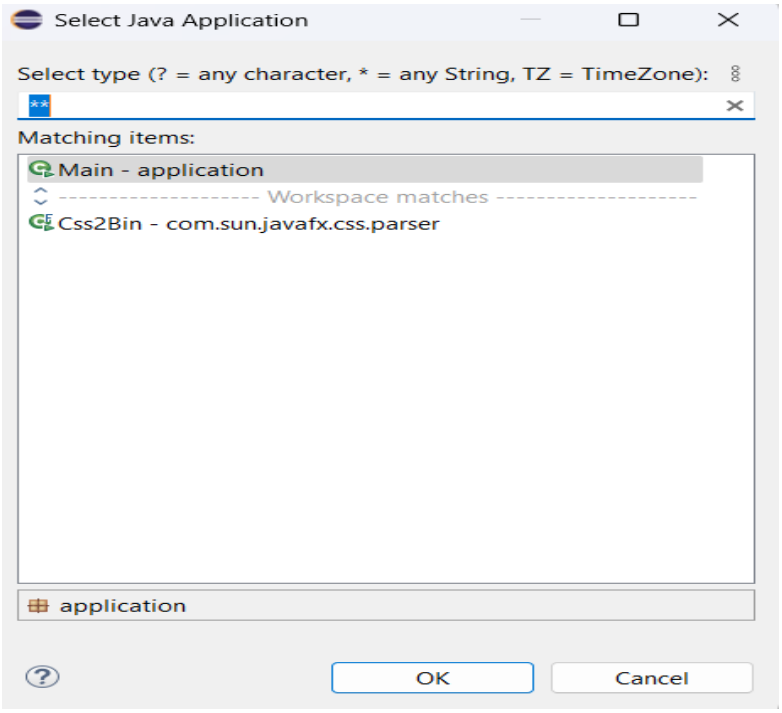


Fig : popout window after running the programme to select the main java file

System Design

Database Schema

- The database consists of three main tables majorly **: Registered_Users , Staff_Profiles , ORG_Departments** to store crucial information about the hospital system.

Implementation

Main.java:

This JavaFX application displays a splash screen with a progress bar while loading the main Hospital Staff Management System UI. It uses a Task to simulate the loading process and switches to the login screen (loaded from an FXML file) upon completion. Timeout and error alerts are implemented to handle delays or issues during loading.

DashBoardController:

This JavaFX controller manages the Dashboard view of a Hospital Staff Management System. It dynamically sets a welcome message and button accessibility based on the user's role (admin or other). The user can navigate to staff and department views (restricted to admins) or sign out, with error handling for view transitions.

DepartmentController:

This JavaFX controller manages the Department view in the Hospital Staff Management System. It allows the user to add, update, or delete departments using a table view and forms, with changes dynamically reflected. It also supports navigation back to the Dashboard and provides alert dialogs for user feedback and error handling.

LoginController:

This JavaFX controller handles user login for the Hospital Staff Management System. It authenticates users based on entered credentials, navigates to the dashboard upon successful login while passing user data, or displays error alerts

for invalid credentials. It also supports navigation to the signup view in case users want to register.

signupController:

This JavaFX controller manages the signup process for the Hospital Staff Management System. It allows users to register with a username, password (hashed for security), and role selection (admin, staff, or manager). Upon successful signup, it displays a confirmation alert and provides an option to navigate back to the login screen.

StaffController:

The Staff Controller class manages staff-related operations in a hospital management JavaFX application. It enables CRUD operations (Create, Read, Update, Delete) on staff data, with role-based access control: Admin (full access), Manager (add, update, delete), and Staff (view-only). It initializes the Table View for staff details and includes methods to add, delete, and update staff using dialogs for user input. The table dynamically refreshes to display updated staff data, and the controller ensures seamless navigation between views while maintaining session details.

DBconnect:

The DB Connect class establishes a connection to the database using configurations from a config.properties file. It loads database credentials (URL, username, and password) and provides a static connect() method to create and return a Connection object. The class uses Logger to handle and log errors, ensuring robust exception handling during file loading and database connection attempts.

Department:

The Department class represents a department entity and provides methods to create, read, update, and delete department records in a database. It interacts with the hms_departments table and uses DB Connect to establish database connections. Key methods include createDepartment(), getDepartment(), updateDepartment(), deleteDepartment(), and getAllDepartments(). The class

also provides logging through Logger to handle and report SQL errors, ensuring effective error handling during database operations.

Staff:

The Staff class manages staff records in the hms_staff database table with methods for creating, retrieving, updating, and deleting staff members. It uses DB Connect for database connections and logs any errors using Logger. The class provides an Observable List of staff members for JavaFX applications, allowing easy integration with UI components.

User:

The User class handles user authentication and management in the database. It provides methods for creating a new user, which includes securely storing hashed passwords, and for authenticating a user by verifying the entered password against the stored hash. The class uses DBConnect for database connections and utilizes SecurityUtil to handle password verification using hashing. It logs all operations and errors for tracking and debugging purposes.

SecurityUtils:

The SecurityUtil class provides methods for hashing and verifying passwords using the BCrypt library. It uses a static salt to enhance the security of password hashing. The hash Password method hashes the password by concatenating the static salt with the raw password and generating a secure hash. The check Password method verifies the entered password against the stored hash by also including the static salt and using Crypt's check PW method to ensure a match. This class is used for securely storing and validating user passwords.

DashBoardview:

This FXML layout defines a dashboard with three main sections. The top section has navigation buttons for staff and department management, and a sign-out button. The centre section displays a welcome message. The layout is styled with modern colours and rounded buttons for a clean, user-friendly interface.

DepartmentView:

This FXML layout represents a department management interface with a vertical box layout. It includes buttons for adding, deleting, updating, refreshing, and navigating back in the button section. A table view is provided to display department information, including department ID and name. The layout features

modern button styling with colour differentiation for various actions, ensuring a clean and user-friendly design.

LoginView:

This FXML layout defines a user-friendly login interface with a clean, modern design. It includes a header with the title "Welcome to Hospital Management Login" and a form container in the centre. The form features input fields for the username and password, styled with subtle colours and rounded corners for a smooth appearance. Below the form, two buttons for "Login" and "Sign Up" are aligned horizontally, styled with bold text and distinct colours to highlight their functions. The layout ensures an intuitive and welcoming user experience.

Signupview:

This FXML layout defines a clean, user-friendly signup interface. The header section invites users to "Create Your Account" with a bold and modern design. The form includes fields for entering a username, password, and selecting a role, with all elements styled to provide a smooth, intuitive user experience. The "Sign Up" and "Back to Login" buttons are clearly labelled and positioned for easy interaction, each with distinct colour schemes to highlight their actions. This layout ensures a seamless registration process within a visually appealing environment.

StaffView:

This FXML layout defines a staff management interface with a modern and clean design. The interface includes a header section that provides staff management functionality, allowing the user to add, delete, update, and refresh staff data. The buttons are visually appealing, each with distinctive colours for different actions (Add Staff, Delete Staff, Update Staff, Refresh, and Back). Below the buttons, a Table View is used to display staff information, including columns for First Name, Last Name, Department, Job Title, and Email. This layout is user-friendly, visually clear, and designed for efficient staff management operations.

Unique Features

- **Progress Bar:**
  A simple yet effective way to show users how far along they are in a process, keeping them informed and engaged.

- **Welcome Note:**

A personalized touch that greets users warmly every time they log in or start using the app, making them feel valued.

- **Enhanced Logging System:**
  Standard Logger Keeps track of routine operations in plain black text for easy readability.
  Severity-Based Coloured Logs: Highlights critical issues in *red*, making errors impossible to miss and ensuring prompt attention.

- **Severity of Exceptions:**
  The app doesn't just handle errors—it categorizes them by seriousness, making it easier to identify and resolve major issues quickly.

- **Advanced Exception Handling:**
  Prevents crashes by catching and managing errors efficiently while providing detailed reports for troubleshooting.

- **Efficient Logging Over Print Statements:**
  Instead of cluttering the code with print statements, the app uses a modern logging framework that's faster, cleaner, and better suited for larger-scale applications.

- **Secure Database Credentials:**
  No hardcoding here! Database credentials are securely stored in a separate config.properties file, keeping sensitive information safe from prying eyes.

- **Password Hashing:**
  User passwords are hashed to ensure they're stored securely, offering strong protection against unauthorized access. B crypt algorithm along with STATIC _SALT is used for hashing for additional security to the password.

- **Timestamp for Sign-Up:**
  Every user sign-up is recorded with a timestamp, adding an extra layer of tracking and accountability.

Functionality

Login Procedures

The system utilizes role-based login to authenticate users and provide them with dashboards customized to their specific roles (admin, manager, or staff). This approach ensures that users can only access functionalities aligned with their assigned permissions. Considering admin , admin has access to both staff and department data he can alter both the data based on the role .

CRUD Operations

The system enables users to perform Create, Read, Update, and Delete

(CRUD) operations based on their designated roles, ensuring that each user has access to only the functionalities that are pertinent to their responsibilities. This role-based access control provides a structured and secure way to manage data while preventing unauthorized modifications.

Detailed Breakdown of CRUD Operations by Role:

1. **Admins:**
    - **Create:** Can add new user accounts, roles, and permissions to the system.
    - **Read:** Have full access to view all records, including staff profiles and their related departments and can also make changes in departments .
    - **Update:** Can modify system settings, user details, and organizational data.

    Delete: Authorized to remove user accounts, outdated records, or obsolete data from the system

2. **Managers:**
    - **Create:** Can add new staff profiles, assign shifts, and allocate tasks.
    - **Read:** Have access to view staff records, schedules, and performance reports.
    - **Update:** Can edit staff details, reassign shifts, and modify team assignments.
    - **Delete:** Allowed to remove or deactivate staff records when necessary, such as in cases of resignation or termination but can't access to departments.
3. **Staff:**
    - **Read:** Can view their schedules, assigned tasks, and personal performance reports.
    - **Delete:** Typically, no delete permissions to ensure data integrity.

Role-Based Access

The application ensures that users can only access features that match their specific roles. When a user logs in, their role is verified, and this restriction is maintained throughout their session. This approach not only keeps the system secure by blocking unauthorized actions but also makes it easier for users to focus on the tools and information they need, creating a smoother and more efficient experience.

Screenshots

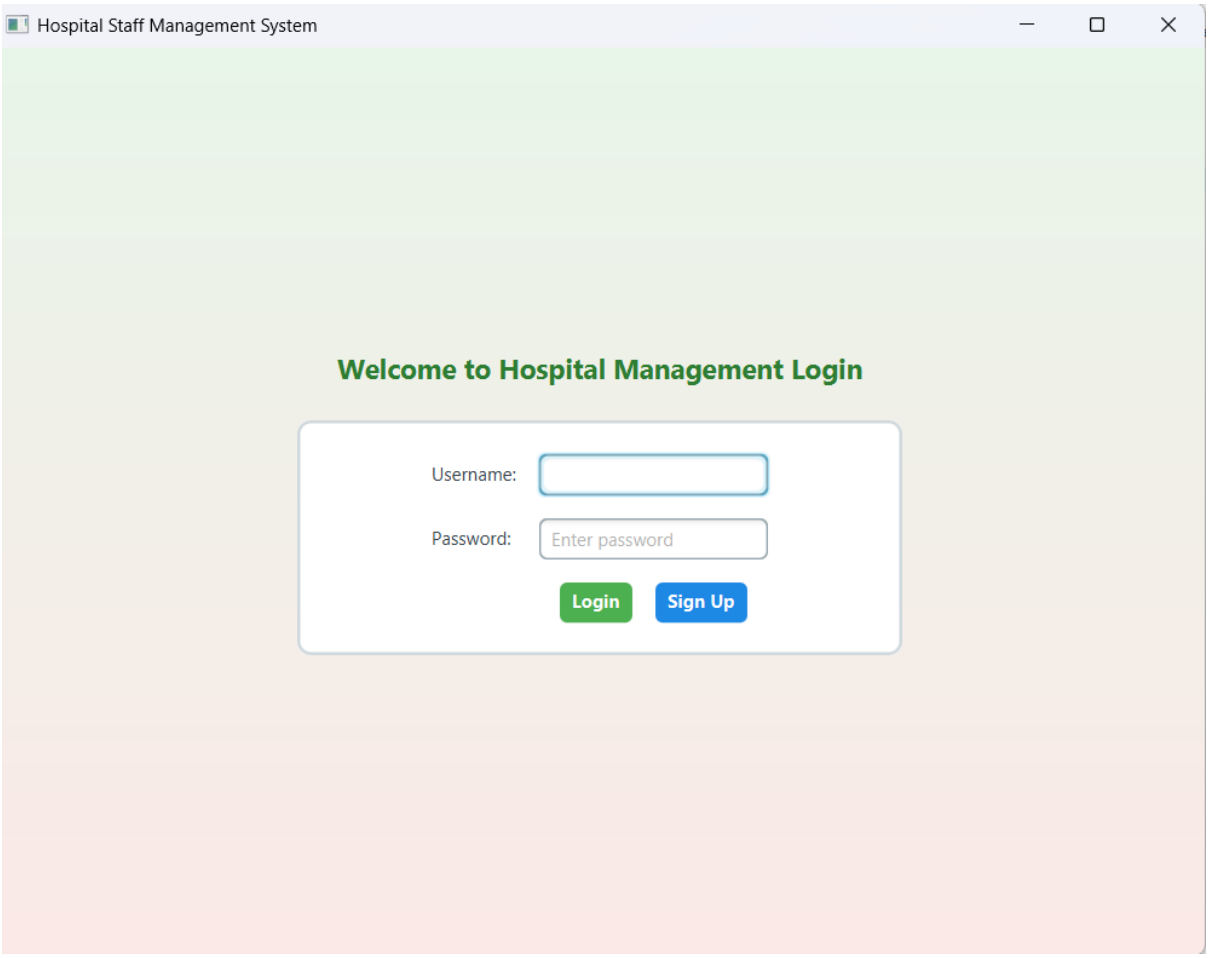


Fig 1: Welcome Page

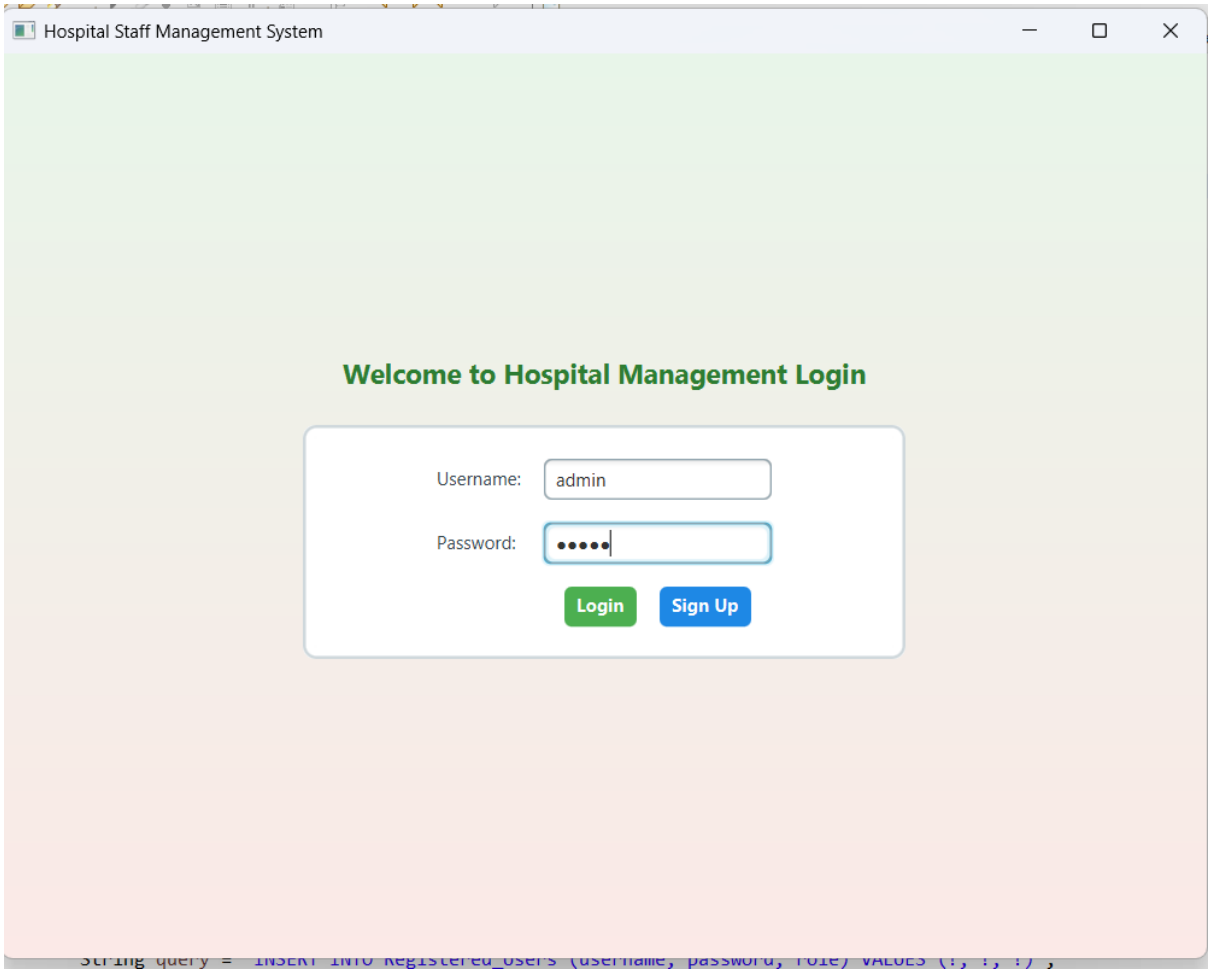


Fig 2 : Admin Logging in

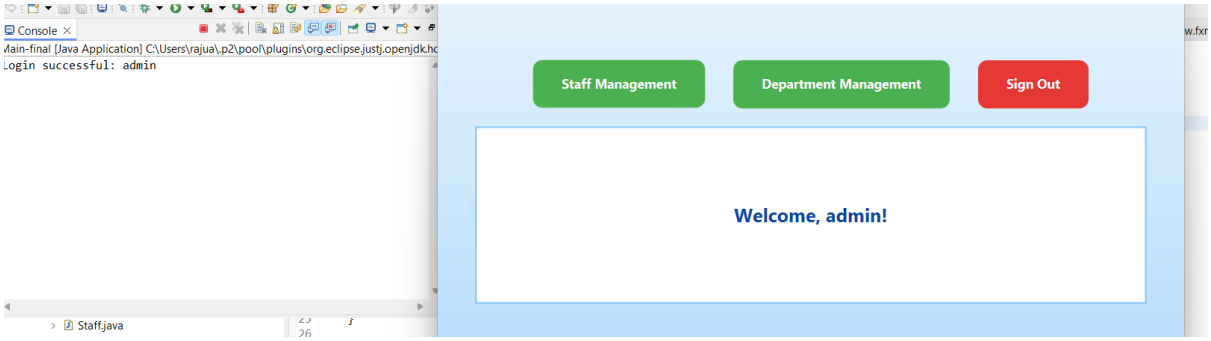


Fig 3 : Admin login page showing all accesses to the admin and login confirmation

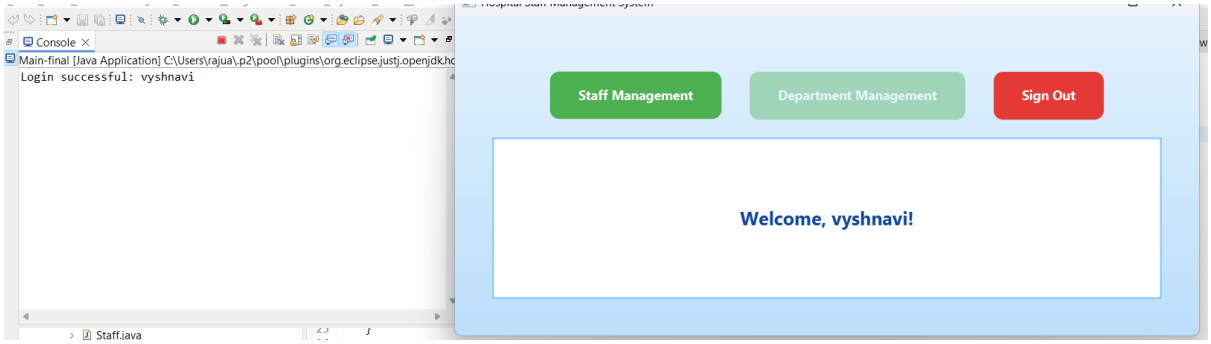


Fig 4 : Page confirmation for manager login

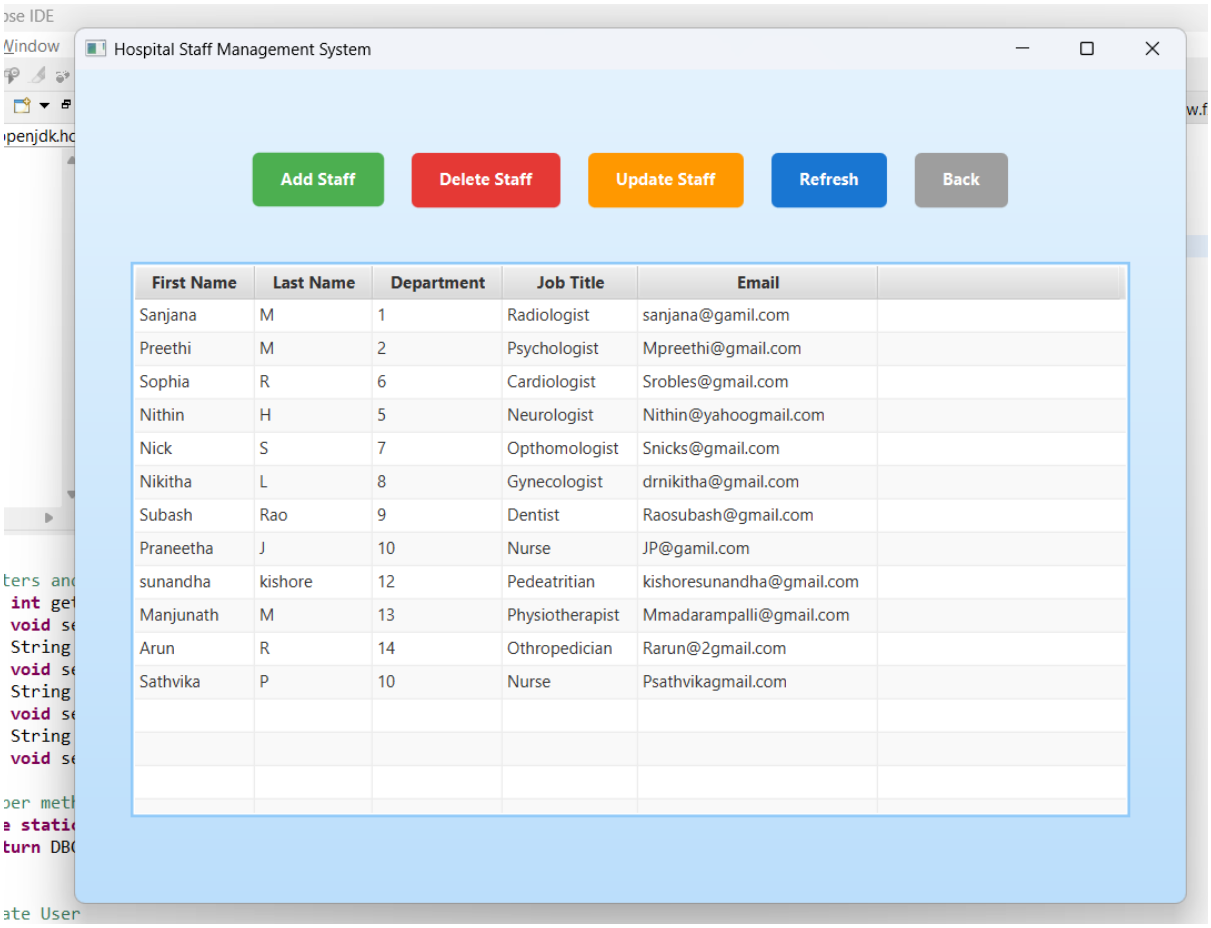


Fig 5 : Table Showing staff data

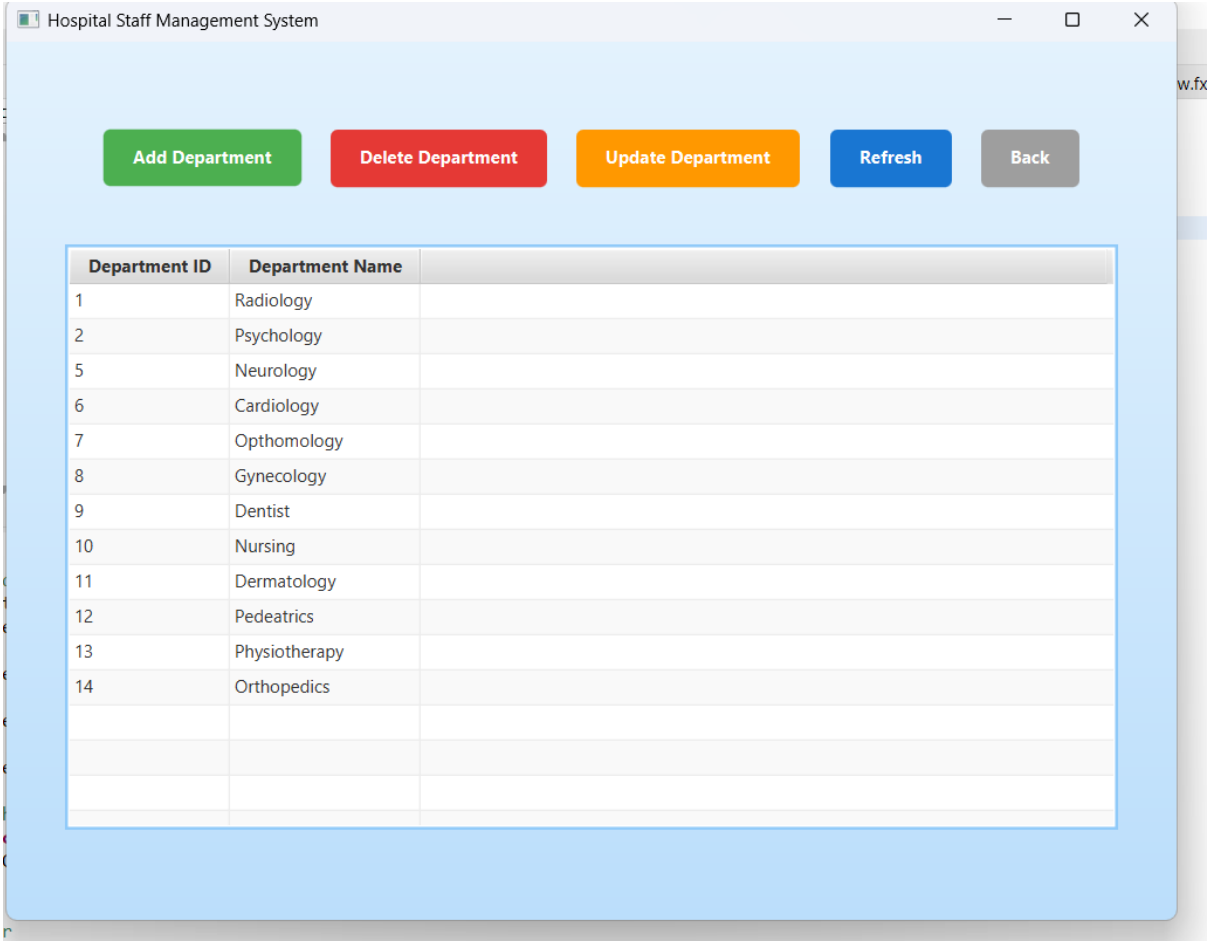


Fig 6 : Table Showing department data

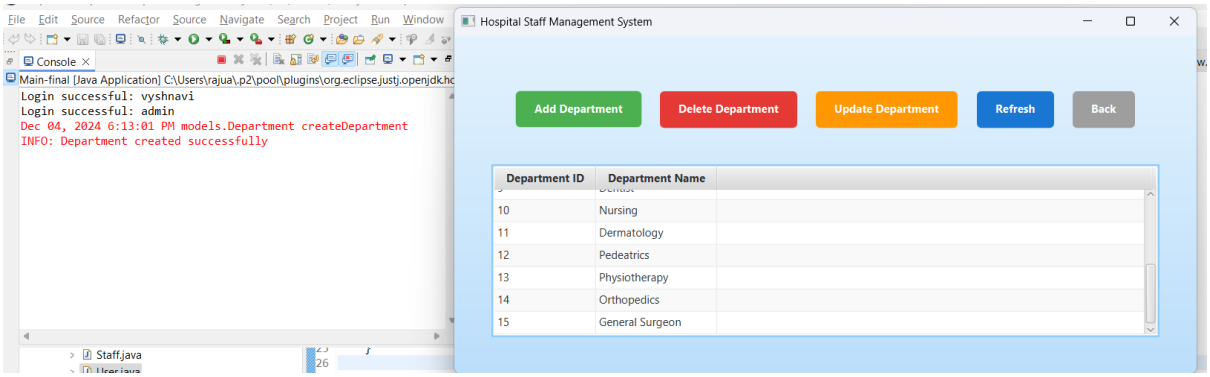


Fig 7 : Admin creating a department

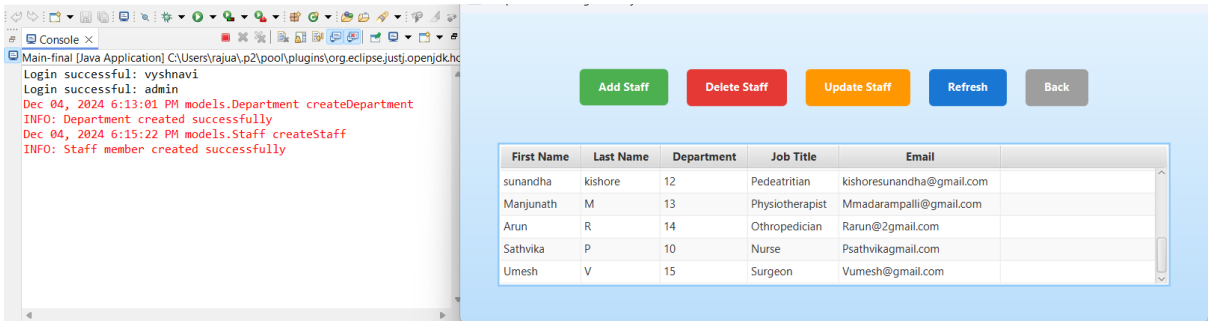


Fig 8 : Admin creating a staff

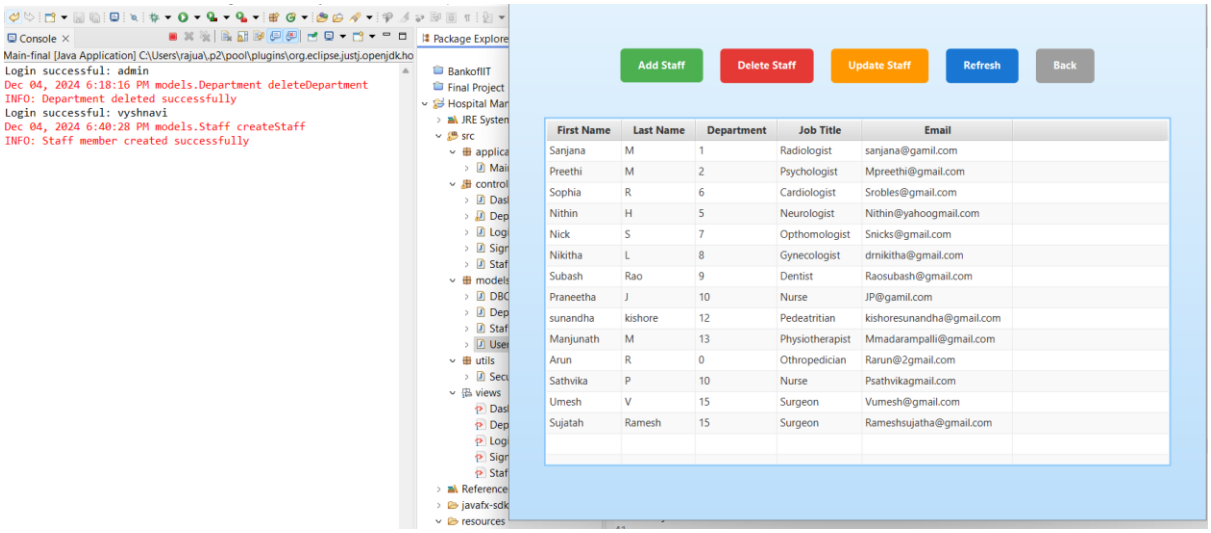


Fig 9: Manager (Vyshnavi) creating a staff

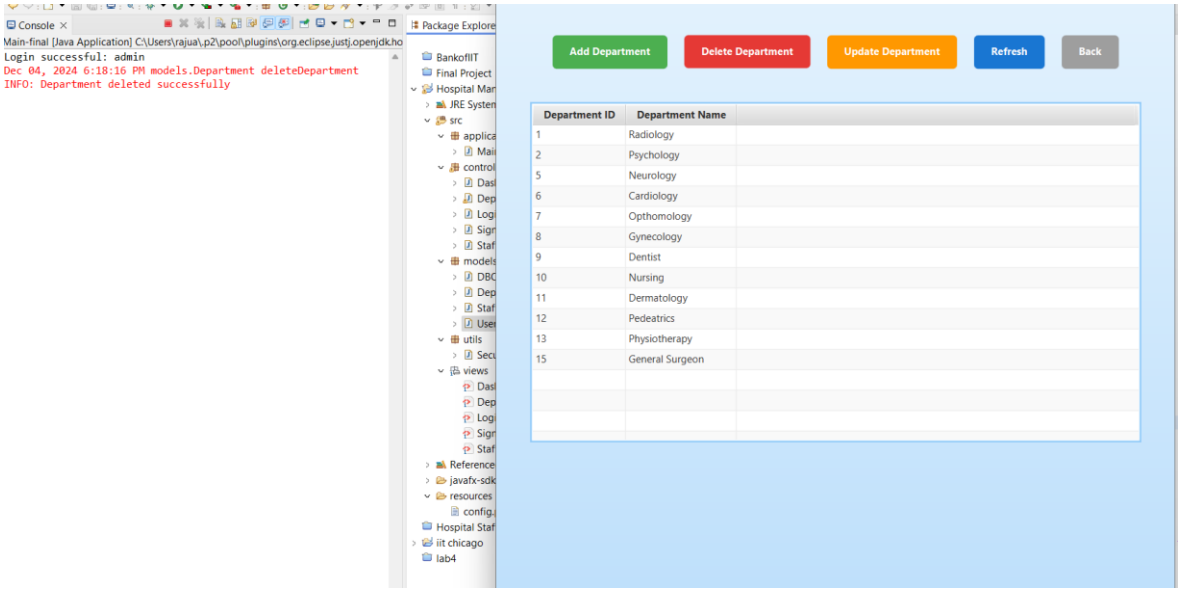
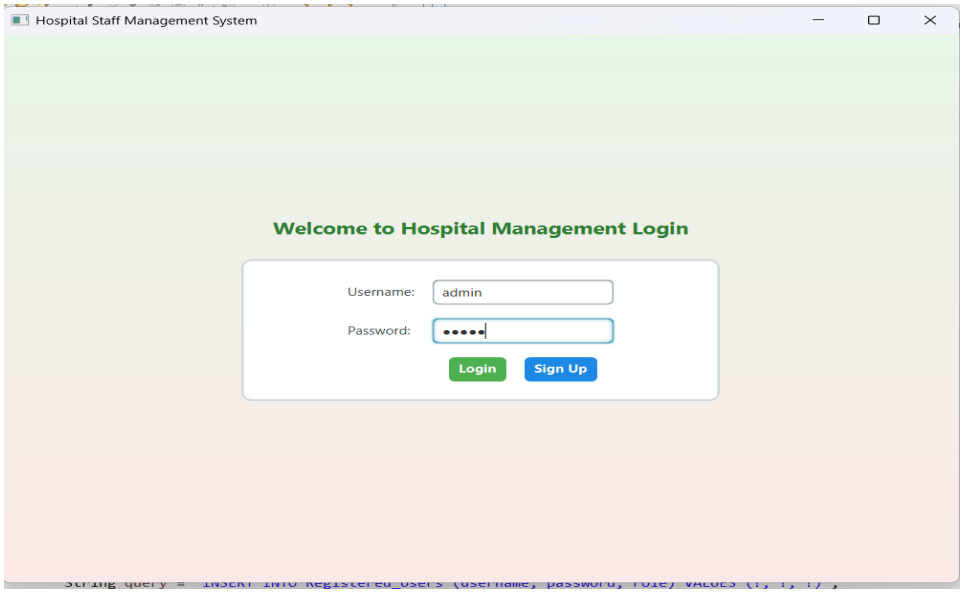


Fig 10 : Admin deleting a department and the table displayed with the data deleted

ExtraCredits:

- **Password Hashing:**
  User passwords are hashed to ensure they're stored securely, offering strong protection against unauthorized access. B crypt algorithm along with STATIC _SALT is used for hashing for additional security to the password.

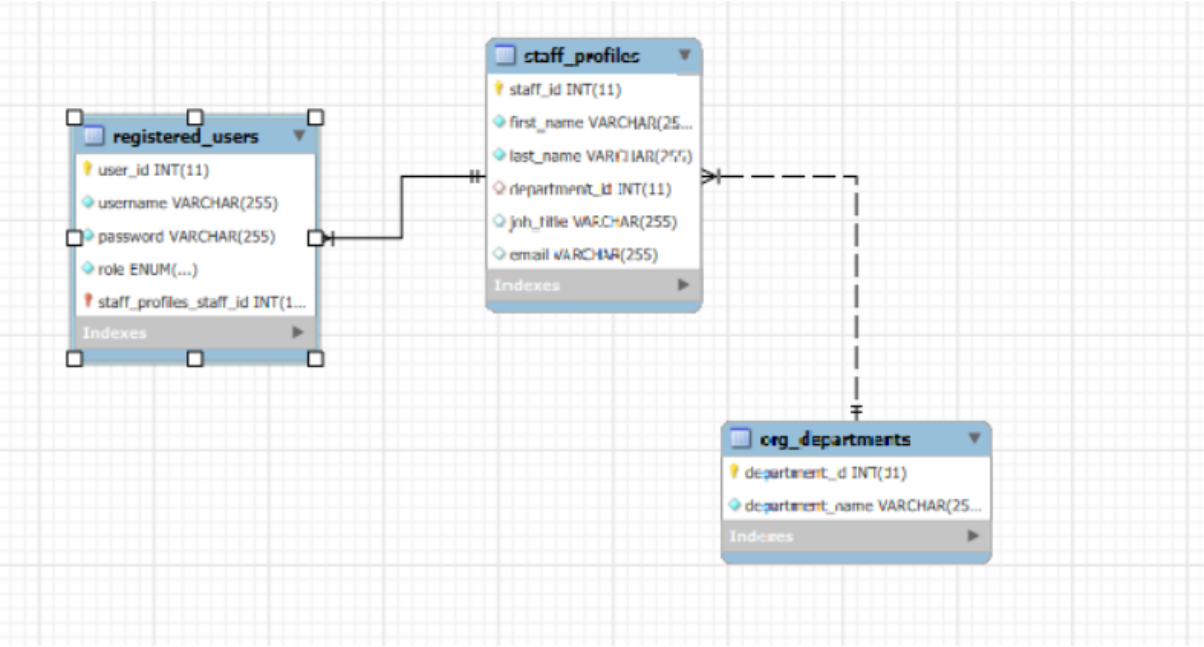


Code For Hashing:

SecurityUtil... | User.java | DashboardCo... | LoginView.fxml | SignupView.fxml | StaffView.fxml

```java
package utils;

import org.mindrot.jbcrypt.BCrypt;

public class SecurityUtil {

    // Static salt that will be used for hashing passwords
    private static final String STATIC_SALT = "SimpleSatyaJava";  // Use your own static salt

    /**
     * Hashes the password using jBCrypt with a static salt.
     *
     * @param password The raw password entered by the user.
     * @return The hashed password.
     */
    public static String hashPassword(String password) {

        return BCrypt.hashpw(STATIC_SALT + password, BCrypt.gensalt(12));


    }


    /**
     * Verifies the login password against the stored hash.
     *
     * @param loginPassword The password entered by the user during login.
     * @param storedHash The hashed password retrieved from the database.
     * @return True if the password matches; otherwise, false.
     */
    public static boolean checkPassword(String loginPassword, String storedHash) {
        // Concatenate the static salt with the entered password before verification
        // Use BCrypt's checkpw method for comparison
        return BCrypt.checkpw(STATIC_SALT + loginPassword, storedHash);

    }
}

```

ER Diagram:


staff_profiles
staff_id INT(11)
first_name VARCHAR(25...
last_name VARCHAR(255)
department_id INT(11)
job_title VARCHAR(255)
email VARCHAR(255)
Indexes
registered_users
user_id INT(11)
username VARCHAR(255)
password VARCHAR(255)
role ENUM(...)
staff_profiles_staff_id INT(1...
Indexes
org_departments
department_id INT(11)
department_name VARCHAR(25...
Indexes