RESOURCE MANAGEMENT REGISTER

23IT454 - MINI PROJECT

A REPORT

Submitted by

K.SHREE HARINI (9517202306092)

S.SOWMIYA SHIVANI (9517202306096)

G.A.VARSHA (95172023060111)

MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI

(AUTONOMOUS)

APRIL 2025

BONAFIDE CERTIFICATE

Certified that this project report titled **Resource Mangement Register** the bonafide work of **Ms.K.Shree Harini (9517202306092), Ms.S.Sowmiya Shivani (9517202306096) and Ms.G.A.Varsha (9517202306096)** who carried out the research under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

Ms.P.Rajalakshmi M.TECH	Dr.T.REVATHI, M.E., Ph.D.,
Internal guide	**Head of the Department**
Assistant Professor,	**Senior Professor,**
Department of Information Technology,	**Department of Information Technology,**
Mepco Schlenk Engineering College,	**Mepco Schlenk Engineering College,**
Sivakasi.	**Sivakasi.**

Submitted for Viva-Voce Examination held at **MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI (AUTONOMOUS)** on **…………………………**

Internal Examiner I **Internal Examiner II**

ABSTRACT

ABSTRACT

This project focuses on the design and development of a resource management software application using Java for the frontend and a Database Management System (DBMS) for the backend. The software aims to streamline and optimize resource management by offering an intuitive user interface for interacting with resources and a robust backend for efficient data handling. Key features include resource allocation tracking, inventory management, role-based access control, and analytics for monitoring usage trends. The project emphasizes scalability, user-friendliness, and reliability, making it suitable for various organizational contexts where effective resource management is critical. By integrating these functionalities, the software provides a comprehensive solution to improve efficiency and decision-making in resource utilization. Moreover, managers face additional challenges in overseeing operations due to the absence of enhanced functionalities for employee data management. Effective management requires access to real-time information about employee performance, resource allocation, and production metrics. Without a streamlined solution, managers may struggle to make informed decisions, leading to inefficiencies in operations and hindering the organization's growth potential. To address these challenges, the proposed project aims to develop a robust desktop application tailored specifically for the printing press industry. This application will serve as a centralized platform for employees to track stock counts, place orders, and monitor resource utilization for specific designs. By automating these processes, the application will reduce the likelihood of errors and improve overall efficiency in resource management. The application will feature an intuitive user interface that allows employees to easily input and access stock information, place orders for materials, and track the status of resources in real-time. This will enable employees to make informed decisions regarding material usage and production schedules, ultimately leading to improved operational efficiency.

TABLE OF CONTENTS

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

1.1 AIM:

The primary aim of this project is to develop an efficient and user-friendly resource management software that simplifies and enhances the process of organizing, allocating, and monitoring resources. By leveraging Java for a responsive frontend and a robust Database Management System (DBMS) for the backend, the software seeks to provide accurate tracking, effective utilization, and insightful analytics to optimize resource usage. This solution aims to cater to organizations or individuals requiring scalable and reliable tools for managing their resources seamlessly.

1.2 OBJECTIVES:

To Develop a software system that simplifies tracking, allocation, and utilization of resources in real-time.

To Create an intuitive user interface using Java, ensuring ease of navigation and accessibility for users of varying technical expertise.

To Design a secure and efficient backend using a Database Management System (DBMS) to store, retrieve, and manage resource data without compromising integrity.

To Integrate analytics and reporting tools to provide insights into resource usage trends, helping users make informed decisions.

To Enable controlled access through role-based permissions to maintain the security and confidentiality of sensitive data.

To Build the system with scalability in mind to accommodate growing organizational demands and resource complexity.

1.3 PROBLEM DEFINITON

Effective resource management is critical for the smooth operation of the printing press industry, where a wide range of materials—such as calendars, flex, lamination sheets, and wedding invitations—are purchased from various customers. Currently, there is a need for a desktop application exclusively accessible to organizational employees that facilitates tracking stock counts, placing orders, and monitoring resource utilization for specific designs. Additionally, managers require enhanced functionalities, including employee data management, to oversee operations effectively. The absence of such a streamlined solution leads to inefficiencies in stock tracking, resource allocation, and operational decision-making. This project addresses these challenges by providing a robust software solution to optimize resource management and support the organization's growth and efficiency.

1.4 OBJECT ORIENTED PROGRAMMING CONCEPTS

Java is a strongly object-oriented programming language that relies on key OOP principles to structure its code effectively

1.4.1 Encapsulation

Encapsulation refers to bundling the data (variables) and methods (functions) operating on the data into a single unit, called a class. It helps safeguard the internal state of an object from external interference. Access modifiers like private, protected, and public are used to control access to class members.

EXAMPLE:

```
public class Resource {

    private int stock;

    public int getStock() {

        return stock;

    }

    public void setStock(int stock) {

        this.stock = stock;

    }

}
```

1.4.2. Inheritance

Inheritance allows one class (child/subclass) to acquire the properties and methods of another class (parent/superclass). This promotes reusability and simplifies code maintenance

Example:

```
public class Employee {
    String name;
}
public class Manager extends Employee {
    int managerId;
}
```

1.4.3. Polymorphism

Polymorphism enables a single interface to support multiple methods or behaviors. It can take two forms: compile-time (method overloading) and runtime (method overriding).

Example

```
public class Resource {
    public void count (int stock) {
        System.out.println(stock);
    }
    public void count (String stockName) {
        System.out.println(stockName);
    }
}
```

1.4.4. Abstraction

Abstraction focuses on hiding implementation details and exposing only essential functionalities. Abstract classes and interfaces are used to achieve abstraction in Java

Example:

```
abstract class Order {

   abstract void placeOrder();

}

class CustomerOrder extends Order {

   void placeOrder() {

      System.out.println("Order placed.");

   }

}
```

1.5. Basic Concepts of Database Management System (DBMS)

1.5.1. Introduction to DBMS

A Database Management System (DBMS) is software designed to manage, organize, and retrieve data from databases efficiently. It acts as an intermediary between the user and the database, enabling seamless data manipulation while ensuring security, integrity, and consistency. Popular examples of DBMS include MySQL, Oracle, and Microsoft SQL Server.

1.5.2. Key Features of DBMS

1. **Data Organization**: Structures data systematically to reduce redundancy and enhance storage efficiency.

2. **Data Retrieval**: Offers powerful querying capabilities using Structured Query Language (SQL), enabling quick and accurate data access.

3. **Data Integrity**: Maintains consistency and correctness of data across operations.

4. **Concurrency Control**: Allows multiple users to access and manipulate the database simultaneously without conflicts.

5. **Security**: Ensures that only authorized users can access sensitive information.

1.5.3. Core Concepts

1. Data Models

A data model defines how data is organized, stored, and manipulated within the database. Common models include:

- **Hierarchical Model**: Organizes data in a tree-like structure.
- **Relational Model**: Represents data as tables (relations) with rows and columns. This is the most widely used model and the foundation of MySQL.
- **Network Model**: Uses a graph structure to represent relationships.
- **Object-Oriented Model**: Stores data as objects, compatible with modern programming paradigms.

2. Relational Database Concepts

- **Tables**: Data is stored in rows (records) and columns (fields), forming a two-dimensional structure.
- **Primary Key**: A unique identifier for a record in a table, ensuring no duplicate rows exist.
- **Foreign Key**: Establishes relationships between tables by referencing the primary key of another table.
- **Normalization**: A process to minimize redundancy by organizing data into multiple related tables.

3. SQL (Structured Query Language)

SQL is the standard language for interacting with relational databases like MySQL. It is categorized into:

- **DDL (Data Definition Language)**: Defines the structure of the database (e.g., CREATE, ALTER, DROP). Example:

sql

```
CREATE TABLE Resources (

   ResourceID INT PRIMARY KEY,

   ResourceName VARCHAR(50),

   Quantity INT
```

);

- **DML (Data Manipulation Language)**: Manages data within tables (e.g., INSERT, UPDATE, DELETE). Example:

sql

INSERT INTO Resources (ResourceID, ResourceName, Quantity)

VALUES (1, 'Paper', 100);

- **DCL (Data Control Language)**: Handles permissions and access controls (e.g., GRANT, REVOKE).

    - **GRANT**: This command is used to provide specific privileges to a user or role on the database objects. *Example:*

GRANT SELECT, INSERT ON Resources TO 'employee'@'localhost';

Explanation: This grants the SELECT and INSERT privileges to the user named employee for the table Resources.

REVOKE: This command removes previously granted privileges from a user or role. *Example:*

REVOKE INSERT ON Resources FROM 'employee'@'localhost';

Explanation: This revokes the INSERT privilege on the table Resources from the user named employee.

SHOW GRANTS: This command displays the privileges assigned to a specific user. *Example:*

sql

SHOW GRANTS FOR 'employee'@'localhost';

Explanation: This shows all the privileges granted to the user employee.

    - **SET PASSWORD**: This command changes the password for a specific user. *Example:*

sql

SET PASSWORD FOR 'employee'@'localhost' = PASSWORD('new_password');

Explanation: This updates the password for the user employee to new_password.

4. Database Transactions

Transactions are operations performed on a database to maintain data integrity. Each transaction follows the **ACID properties**:

- **Atomicity**: Ensures all operations in a transaction are completed; otherwise, none are executed.
- **Consistency**: Ensures the database remains in a consistent state before and after the transaction.
- **Isolation**: Prevents transactions from interfering with one another when executed concurrently.
- **Durability**: Ensures data persists even in the case of system failure.

1.5.4. Advantages of Using MySQL for backend

1. **Open Source**: MySQL is free to use and widely supported by the community.
2. **High Performance**: Optimized for fast and efficient data handling, making it ideal for resource management systems.
3. **Scalability**: Suitable for handling both small-scale applications and large, complex datasets.
4. **Cross-Platform Support**: Runs seamlessly across various operating systems like Windows, Linux, and macOS.
5. **Ease of Use**: User-friendly with a straightforward SQL syntax that is easy to learn and implement.

1.6. OUR UNDERSTANDING ON THE PROBLEM

The printing press industry operates in a dynamic environment where effective resource management is paramount to ensure smooth operations and meet customer demands. This industry deals with a diverse range of materials, including calendars, flex materials, lamination sheets, and wedding invitations, which are sourced from various customers. Each of these products requires careful tracking of stock levels, timely ordering of materials, and efficient utilization of resources to maintain production schedules and meet delivery deadlines.

The application will feature an intuitive user interface that allows employees to easily input and access stock information, place orders for materials, and track the status of resources in real-time. This will enable employees to make informed decisions regarding material usage and production schedules, ultimately leading to improved operational efficiency.

In addition to stock management, the application will include functionalities for employee data management, allowing managers to oversee employee performance, track attendance, and manage workloads effectively. This will provide managers with the insights needed to optimize team performance and ensure that resources are allocated appropriately.

Furthermore, the application will incorporate reporting and analytics features, enabling managers to generate reports on stock levels, resource utilization, and employee performance. This data-driven approach will facilitate better decision-making and strategic planning, allowing the organization to respond quickly to changing market demands and customer needs.

By implementing this software solution, the printing press industry can expect to see significant improvements in resource management, operational efficiency, and overall productivity. The streamlined processes will not only enhance the organization's ability to meet customer demands but also support its growth and competitiveness in the market.

The code addresses the problem statement by providing a comprehensive solution for resource management in the printing press industry. Here's how:

1.User Authentication: The application allows employees and managers to log in securely, ensuring that only authorized personnel can access sensitive information.

2. **Registration of New Users**: The registration functionality allows new employees to create accounts, which helps in maintaining an updated database of users. This is essential for tracking who has access to the system and managing user roles effectively.
3. **Manager Workspace**: The ManagerFrame class provides a dedicated workspace for managers to oversee operations. It includes buttons to manage employees, stock, and jobs, which streamlines the management process and enhances operational efficiency.
4. **Employee Management**: The EmployeeManage class allows managers to add, edit, and delete employee records. This functionality is vital for keeping the employee database current and ensuring that the organization can quickly adapt to changes in staffing.
5. **Job Management**: The JobsManage class enables managers to manage job records effectively. This includes adding new jobs, editing existing ones, and deleting jobs that are no longer relevant. This helps in maintaining an organized workflow and ensures that all job-related information is up to date.
6. **Stock Management**: The StockManage class provides functionalities to manage stock records, including adding new stock items, editing existing records, and deleting items that

are no longer in inventory. This is crucial for ensuring that the organization has the necessary materials on hand to meet customer demands.

7. **Viewing Stock**: The ViewStockFrame class allows users to view low stock resources. This feature is essential for proactive inventory management, enabling the organization to reorder materials before they run out, thus preventing delays in production.
8. **Database Connectivity**: The application connects to a MySQL database, which stores all relevant data, including user accounts, employee records, job details, and stock information. This centralized data management system ensures that all users have access to the most current information.
9. **Error Handling**: Throughout the application, error handling is implemented to provide feedback to users in case of issues, such as invalid input or database connection failures. This enhances the user experience by guiding users to correct mistakes and ensuring smooth operation.
10. **User -Friendly Interface**: The use of Java Swing for the graphical user interface (GUI) makes the application visually appealing and easy to navigate. This is important for user adoption, as a well-designed interface can significantly improve usability.
11. **Progress Feedback**: The splash screen with a progress bar provides users with feedback during the loading process, enhancing the overall user experience by indicating that the application is starting up.

1.6.1. THE NEED FOR RESOURCE MANAGEMENT

At the forefront of this application is the **Front Frame**, which serves as the initial splash screen, much like the welcoming facade of a grand theater. As users enter, they are greeted by a visually appealing interface that builds anticipation through a progress bar.

Transitioning from the splash screen, users are directed to the **Login Frame**, where they must enter their credentials to access the system..

Once logged in, managers are ushered into the **Manager Frame**, a digital command center equipped with buttons that lead to various management functionalities

The **Employee Management Frame** allows managers to add, edit, or delete employee records, functioning like a human resources department in a corporate office

Similarly, the **Job Management Frame** provides a platform for tracking job assignments and progress. This feature is akin to a project management tool, where tasks are assigned, deadlines

The **Stock Management Frame** is another critical component, allowing users to manage inventory levels of materials used in printing

In the **View Stock Frame**, users can access a detailed overview of low stock resources, akin to a dashboard displaying vital statistics for a business. This feature empowers managers to make informed decisions about reordering supplies, ensuring that the printing press operates without interruptions. The application's database interaction is the backbone of its functionality, akin to the circulatory system in a living organism. It connects various components, ensuring that data flows seamlessly between the user interface and the underlying database

Once logged in, managers enter the **Manager Frame**, which is like a control room. Here, they can manage employees, stock, and jobs

The **Employee Management Frame** allows managers to add or remove employee records. Imagine a new worker joining the team.

In the **Job Management Frame**, managers can track job assignments. For example, if a designer is working on a new wedding invitation, the manager can assign the job and set a deadline. This is like a teacher assigning homework to students and checking if they complete it on time.

The **Stock Management Frame** helps manage the inventory of materials used in printing. Picture a warehouse filled with paper, ink, and other supplies.

In the **View Stock Frame**, users can see a list of low stock items. This is like a dashboard that shows important information. If the system indicates that a certain type of paper is critically low, the manager can quickly place an order to avoid running out.

1.6.2. Database Structure

The application relies on a well-structured database to store and manage data effectively. Here's a breakdown of the key tables and their purposes:

1. **Users Table**: Stores user information, including names, emails, phone numbers, usernames, and passwords. This table is crucial for authentication and user management.
2. **Supplier Table**: Contains details about suppliers, including their names, contact information, and addresses.
3. **Resources Table**: Holds information about various resources used in the printing process, including resource names, types, quantities, and associated suppliers
4. **Employees Table**: Manages employee records, including names, roles, and contact information. This table is important for tracking employee details and managing human resources.

5. **Machine Table**: Contains information about the machines used in the printing process, including machine IDs, names, types, and statuses.
6. **Scoring Table**: Records details about scoring jobs, including job IDs, scoring dates, and output quantities.
7. **Laminating Table**: Manages records related to lamination jobs, including IDs, machine IDs, lamination dates, and output quantities
8. **Designing Table**: Contains information about design jobs, including design IDs, software used, project names, creation dates, and design type IDs.
9. **DesignType Table**: Stores different types of designs, allowing for categorization and organization of design jobs.
10.

1.6.3. Stored Procedures

The application utilizes stored procedures to encapsulate complex SQL queries and operations. For example, the **GetLowStockResources()** stored procedure retrieves resources that are low in stock, providing a streamlined way to access critical inventory information.

1.6.4. User Interface Design

The user interface is designed using Java Swing, which allows for the creation of a responsive and visually appealing application. Key design elements include:

- **Panels**: Different panels are used to organize components logically, such as header panels for titles and navigation buttons, and content panels for displaying tables and forms.
- **Buttons**: Action buttons are provided for user interactions, such as logging in, registering, and managing records. Icons are used to enhance the visual appeal and usability of buttons.
- **Tables**: JTable components are used to display data in a structured format, allowing users to view and manage records easily.

1.6.5. User Experience Considerations

1. **Feedback Mechanisms**: The application provides feedback to users through dialog boxes, informing them of successful operations or errors. This is crucial for maintaining user engagement and trust in the application.

2. **Input Validation**: Input fields are validated to ensure that users provide the correct data format, reducing the likelihood of errors and improving data integrity.
3. **Navigation**: The application allows for easy navigation between different functionalities, such as employee management, job management, and stock management. This is important for enhancing user efficiency and satisfaction.
4. **Accessibility**: The use of clear labels, tooltips, and intuitive layouts makes the application accessible to users with varying levels of technical expertises

1.6.6. JAVA DATABASE CONNECTIVITY

Java Database Connectivity (JDBC) is a Java-based API that enables Java applications to interact with various databases. It provides a standard interface for connecting to relational databases, executing SQL queries, and retrieving results. JDBC allows developers to perform operations such as creating, reading, updating, and deleting data in a database. The API consists of two main packages: java.sql and javax.sql, which provide classes and interfaces for database connectivity and data manipulation.

1.6.7. ALERTING THE USER THROUGH MAIL

The resource management desktop application continuously monitors the inventory levels of various resources. When the resource count falls below a predefined threshold, the application triggers an alert mechanism to notify the user. This process begins by identifying the specific resource with low availability and obtaining the user's registered email address. The application then uses an email-sending library or API, such as JavaMail or an SMTP client, to compose an informative message detailing the low resource count and suggesting necessary actions

CHAPTER 2

SYSTEM DESIGN

2.1. ENTITY-RELATIONSHIP DIAGRAM

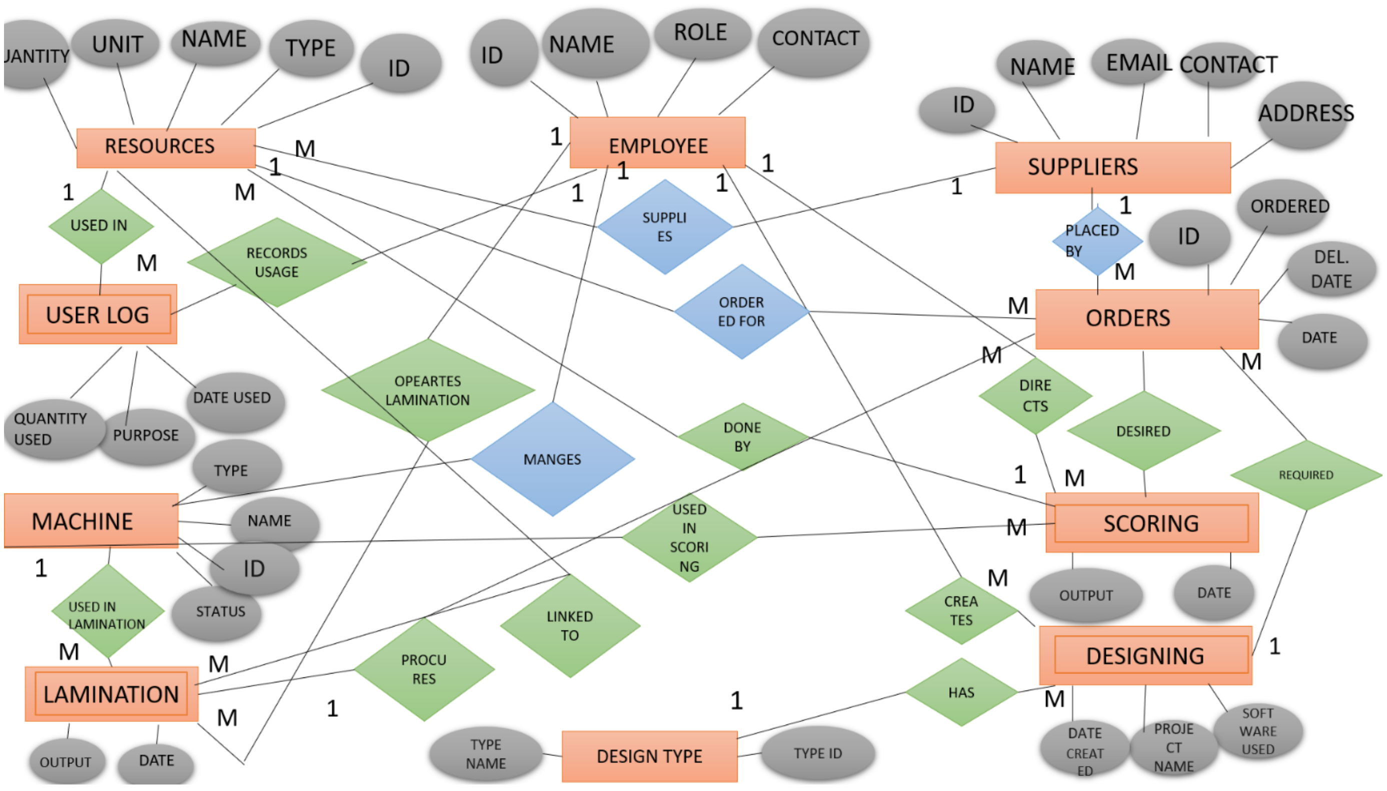


FIG 2.1

Fig 2.1 refers the ER DIAGRAM of the project,here it shows the relationship between the entities.

2.1.1. ENTITIES AND ATTRIBUTES

Resources(id,name,type,quantity,unit,supplier_id)

1. Suppliers(supplier_id,name,contact,email,address)
2. Employees(employee_id,name,role,contact)
3. Usage log(resource_id,employee_id,date used.quantity used,purpose)
4. Orders(orders_id,resource_id,supplier_id,order date,quantity ordered,expected delivery date)

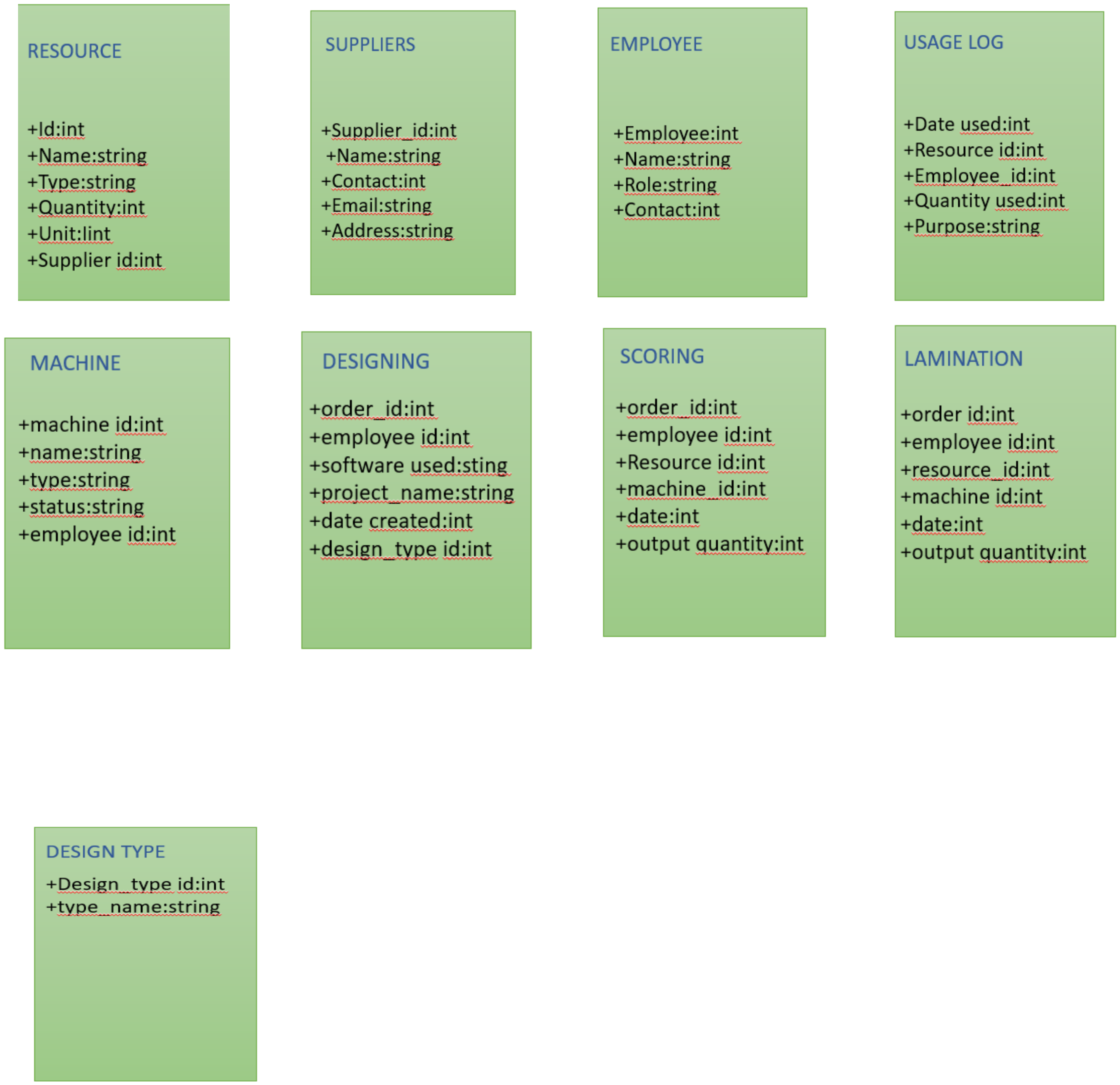
5. Machine(machine_id,name,type,status,employee_id)
6. Scoring(order_id,employee_id,resource_id,machine_id,date,output quantity)
7. Lamination (order_id,employee_id,resource_id,machine_id,date,output quantity)
8. Designing(order_id,employee_id,resource_id,machine_id,date,output quantity)
9. Design type(design_type id,type_name)

RELATIONSHIP	REDUNDANT ATTRIBUTE	INVOLVED ENTITES
Supplies	Supplier_id	Resource,supplier
Used in	Resource_id	Resorce,usagelog
Records_usage	Employee_id	Employee, usagelog
Order for	Resource_id	Resource, orders
Placed by	Supplier_id	Supplies, orders
manages	Employee_id	Employee, machine
Directs	Employee_id	Employee, scoring
Done by	Resource_id	Resource, scoring
Used in scoring	Machine_id	Machine, scoring
Operates lamination	Employee_id	Lamination, employee
Linked to	Resource_id	Lamination, resource
Used in lamination	Machine_id	Lamination,machine
Creates	Employee_id	Designing,desiging type
has	Design_type id	Designing,design type
desired	Order_id	Lamination,orders
procures	Order_id	Lamination,orders
required	Order_id	Designing,orders

TABLE 2.1.1

TABLE 2.1.1 refers to the identified relationship between the entities by reducing the values.

2.2.CLASS DIAGRAM



CHAPTER 3

IMPLEMENTATION METHODOLOGY

3.1. MODULES IDENTIFIED

3.1.1. Frontframe Class

- **Purpose**: Acts as the splash screen for the application.

Methods:

- **Frontframe()**: Constructor that initializes the components and starts the progress bar.
- **startProgressBar()**: Initializes and updates the progress bar, transitioning to the login screen once loading is complete.
- **initComponents()**: Sets up the user interface components for the splash screen.

3.1.2. LoginFrame Class

- **Purpose**: Handles user authentication.

Methods:

- **LoginFrame()**: Constructor that initializes the login frame and connects to the database.
- **initComponents()**: Sets up the user interface components, including labels, text fields, buttons, and checkboxes for showing/hiding passwords.
- **LogbuttonActionPerformed(ActionEvent evt)**: Validates user input and calls the **login()** method to authenticate the user.
- **User textActionPerformed(ActionEvent evt)**: Placeholder for handling actions on the username text field.
- **ShowpasswordActionPerformed(ActionEvent evt)**: Toggles the visibility of the password field based on the checkbox state.

3.1.3. Registeration Class

- **Purpose**: Allows new users to register.

Methods:

- **Registeration()**: Constructor that initializes the registration frame.
- **initComponents()**: Sets up the registration form with fields for name, email, gender, phone number, username, and password.
- **REgbuttonActionPerformed(ActionEvent evt)**: Validates input fields and calls the **reg()** method to add a new user to the database.

3.1.4. ManagerFrame Class

- **Purpose**: Provides a workspace for managers to oversee operations.

Methods:

- **ManagerFrame()**: Constructor that initializes the manager frame and displays a welcome message.
- **initComponents()**: Sets up the user interface with buttons for managing employees, stock, and jobs.
- **LogoutActionPerformed(ActionEvent evt)**: Handles the logout action, closing the manager frame and returning to the login screen.

3.1.5. EmployeeManage Class

- **Purpose**: Manages employee records.

Methods:

- **EmployeeManage()**: Constructor that initializes the employee management frame.
- **initComponents()**: Sets up the interface for adding, editing, and deleting employee records.
- **InsertbuttActionPerformed(ActionEvent evt)**: Inserts a new employee record into the database.
- **EditbuttActionPerformed(ActionEvent evt)**: Updates existing employee records based on user input.
- **DelbuttActionPerformed(ActionEvent evt)**: Deletes an employee record based on the provided employee ID.

3.1.6. JobsManage Class

- **Purpose**: Manages job records.

Methods:

- **JobsManage()**: Constructor that initializes the job management frame.
- **initComponents()**: Sets up the interface for managing job records.
- **InsertbuttActionPerformed(ActionEvent evt)**: Inserts a new job record into the database.
- **Del1buttActionPerformed(ActionEvent evt)**: Deletes a job record based on the provided job ID.
- **Ed1buttActionPerformed(ActionEvent evt)**: Updates an existing job record.

3.1.7. StockManage Class

- **Purpose**: Manages stock records.

Methods:

- **StockManage()**: Constructor that initializes the stock management frame.
- **initComponents()**: Sets up the interface for managing stock records.
- **InsertbuttActionPerformed(ActionEvent evt)**: Inserts a new stock record into the database.
- **EditbuttActionPerformed(ActionEvent evt)**: Updates an existing stock record.
- **DelbuttActionPerformed(ActionEvent evt)**: Deletes a stock record based on the provided product ID.

3.1.8. ViewStockFrame Class

- **Purpose**: Displays low stock resources.

Methods:

- **ViewStockFrame()**: Constructor that initializes the view stock frame.
- **loadData()**: Loads data from the database and populates the table with low stock resources.

- **showButtonActionPerformed(ActionEvent evt)**: Calls the **loadData()** method to refresh the table with current data.

3.1.9. Database Connection Classes (loginform, registerform, Employee, Jobs, Stock)

These classes handle database connections and operations related to their respective functionalities.

Common Methods:

- **connectToDatabase()**: Establishes a connection to the MySQL database and handles exceptions related to database connectivity.
- **getConnection()**: Returns the current database connection.

3.1.10. loginform Class

- **Purpose**: Handles the database connection and login functionality.

Methods:

- **connectToDatabase()**: Establishes a connection to the MySQL database and handles exceptions related to database connectivity.
- **login()**: Authenticates the user by checking the provided username, password, and role against the records in the database.

3.1.11. registerform Class

- **Purpose**: Handles the database connection and registration functionality.

Methods:

- **connectToDatabase()**: Establishes a connection to the MySQL database and handles exceptions related to database connectivity.
- **reg()**: Inserts a new user record into the database after validating the input fields.

3.1.12. Employee Class

- **Purpose**: Manages employee-related database operations.

Methods:

- **connectToDatabase()**: Establishes a connection to the MySQL database and handles exceptions related to database connectivity.
- **getConnection()**: Returns the current database connection.

3.1.13. Jobs Class

- **Purpose**: Manages job-related database operations.

Methods:

- **connectToDatabase()**: Establishes a connection to the MySQL database and handles exceptions related to database connectivity.
- **getConnection()**: Returns the current database connection.

3.1.14. Stock Class

- **Purpose**: Manages stock-related database operations.

Methods:

- **connectToDatabase()**: Establishes a connection to the MySQL database and handles exceptions related to database connectivity.
- **getConnection()**: Returns the current database connection.

3.1.15. DisplayTables Class

- **Purpose**: Displays various tables from the database in a tabbed interface.

Methods:

- **DisplayTables()**: Constructor that initializes the display tables frame and connects to the database.
- **connectToDatabase()**: Establishes a connection to the MySQL database.
- **createTablePanel(String sql)**: Creates a panel with a table based on the provided SQL query, populating it with data from the database.
- **refreshAllTabs()**: Refreshes the data displayed in all tabs by reloading the tables.

3.1.16. ViewStock Class

- **Purpose**: Displays low stock resources in a table format.

Methods:

- **ViewStock()**: Constructor that initializes the view stock frame and sets up the user interface.
- **loadData()**: Loads low stock resources from the database and populates the table with the current data.
- **showButtonActionPerformed(ActionEvent evt)**: Calls the **loadData()** method to refresh the table with current data.

3.1.17. Stockmanager1 Class

- **Purpose**: Manages stock records specifically for employees.

Methods:

- **Stockmanager1()**: Constructor that initializes the stock management frame.
- **initComponents()**: Sets up the user interface for managing stock records.
- **InsertbuttActionPerformed(ActionEvent evt)**: Inserts a new stock record into the database.
- **EditbuttActionPerformed(ActionEvent evt)**: Updates an existing stock record based on user input.
- **DelbuttActionPerformed(ActionEvent evt)**: Deletes a stock record based on the provided product ID.
- **showStock()**: Displays the current stock records in the table.

3.2. MODULE DESCRIPTION

3.2.1. Frontframe Class

- When the application starts, an instance of the **Frontframe** class is created, which displays a splash screen with a welcome message and a progress bar.
- The **initComponents()** method is called to set up the user interface components, including the progress bar and any labels or images.
- The **startProgressBar()** method initializes the progress bar's value to 0 and uses a **Timer** to increment the progress value. As the progress bar fills up, it simulates the loading process.
- Once the progress reaches 100%, the splash screen is disposed of, and the login frame (**LoginFrame**) is opened, allowing users to log in to the application.

3.2.2. LoginFrame Class

- The **LoginFrame** class initializes the login interface, which includes text fields for entering the username and password, as well as a button to submit the login request.
- The **initComponents()** method sets up the layout and components of the login form, including labels, text fields, and buttons.
- When the user clicks the login button, the **LogbuttonActionPerformed()** method is triggered. This method retrieves the input from the username and password fields and validates them.
- The **login()** method is called to check the provided credentials against the records stored in the database. It connects to the database, executes a query to find a matching username and password, and checks the user's role.
- If the credentials are valid, the user is granted access to the application, and the appropriate manager or employee interface is opened. If not, an error message is displayed to inform the user of the failed login attempt.

3.2.3. Registeration Class

- The **Registeration** class initializes the registration form, which includes fields for entering user details such as name, email, gender, phone number, username, and password.
- The **initComponents()** method sets up the layout and components of the registration form, ensuring that all necessary fields are present for user input.

- When the user clicks the register button, the **REgbuttonActionPerformed()** method is called. This method retrieves the input from the registration fields and performs validation to ensure all required fields are filled out correctly.
- The **reg()** method is then called to insert the new user record into the database. It connects to the database, prepares an SQL statement, and executes it to add the new user.
- If the registration is successful, the user is informed, and they can then log in using their new credentials.

3.2.4. ManagerFrame Class

- The **ManagerFrame** class initializes the manager's workspace, which includes buttons for managing employees, stock, and jobs.
- The **initComponents()** method sets up the user interface, arranging buttons and labels in a user-friendly layout.
- Each button is linked to an action listener that opens the corresponding management frame when clicked. For example, clicking the "Manage Employees" button opens the **EmployeeManage** frame, while the "Manage Stock" button opens the **StockManage** frame.
- This modular design allows managers to easily navigate between different management tasks without cluttering the interface.

3.2.5. EmployeeManage Class

- The **EmployeeManage** class initializes the employee management interface, which includes fields for entering employee details such as name, email, and role.
- The **initComponents()** method sets up the layout and components for managing employee records, including buttons for adding, editing, and deleting employees.
- When the user clicks the "Add Employee" button, the **InsertbuttActionPerformed()** method is triggered. This method retrieves the input from the fields and validates it before calling the database to insert the new employee record.
- The **EditbuttActionPerformed()** method allows managers to update existing employee records. It retrieves the selected employee's details, populates the fields, and updates the record in the database when the user submits the changes.

- The **DelbuttActionPerformed()** method enables managers to delete an employee record by providing the employee ID. It connects to the database, executes a delete query, and removes the specified record.

3.2.6. JobsManage Class

- The **JobsManage** class initializes the job management interface, which includes fields for entering job details such as job name, description, and status.
- The **initComponents()** method sets up the layout and components for managing job records, including buttons for adding, editing, and deleting jobs.
- When the user clicks the "Add Job" button, the **InsertbuttActionPerformed()** method is called. This method retrieves the job details from the input fields and validates them before inserting the new job record into the database.
- The **Del1buttActionPerformed()** method allows managers to delete a job record by providing the job ID. It connects to the database, executes a delete query, and removes the specified job record
- The **Ed1buttActionPerformed()** method allows managers to update existing job records. When the user selects a job from the list and clicks the edit button, this method retrieves the job's current details, populates the input fields with this information, and then updates the record in the database when the user submits the changes.
- The interface is designed to provide feedback to the user after each operation, such as confirming successful additions, updates, or deletions of job records.

3.2.7. StockManage Class

- The **StockManage** class initializes the stock management interface, which includes fields for entering stock details such as product name, quantity, and supplier information.
- The **initComponents()** method sets up the layout and components for managing stock records, including buttons for adding, editing, and deleting stock items.
- When the user clicks the "Add Stock" button, the **InsertbuttActionPerformed()** method is triggered.
- When the user clicks the "Add Stock" button, the **InsertbuttActionPerformed()** method is triggered. This method retrieves the stock details from the input fields, validates the input, and then calls the database to insert the new stock record.

- The **EditbuttActionPerformed()** method allows users to update existing stock records. When a stock item is selected and the edit button is clicked, this method retrieves the current details of the selected stock item, populates the input fields with this information, and updates the record in the database when the user submits the changes.
- The **DelbuttActionPerformed()** method enables users to delete a stock record by providing the product ID. It connects to the database, executes a delete query, and removes the specified stock record. After each operation, the interface provides feedback to the user, confirming the success or failure of the action.

3.2.8. ViewStockFrame Class

- The **ViewStockFrame** class initializes the interface for displaying low stock resources. This interface includes a table that shows items that are below a certain stock threshold.
- The **loadData()** method is responsible for retrieving low stock resources from the database. It executes a query to fetch items with quantities below the defined threshold and populates the table with this data.
- When the user clicks the button to refresh the stock view, the **showButtonActionPerformed()** method is called, which in turn calls the **loadData()** method to update the table with the most current low stock information.
- This class ensures that users can easily monitor stock levels and take necessary actions to reorder items that are running low.

3.2.9. Database Connection Classes (loginform, registerform, Employee, Jobs, Stock)

These classes handle the database connections and operations related to their respective functionalities.

Common Methods:

- **connectToDatabase()**: This method establishes a connection to the MySQL database. It uses JDBC (Java Database Connectivity) to connect to the database server, handling any exceptions that may occur during the connection process. If the connection is successful, it returns the connection object for further use.

- **getConnection()**: This method returns the current database connection object. It is used by other methods in the respective classes to perform database operations such as querying, inserting, updating, or deleting records.

3.2.10. loginform Class

- The **loginform** class is responsible for managing the login process. It connects to the database and checks user credentials.
- The **connectToDatabase()** method establishes a connection to the database, allowing the application to access user records.
- The **login()** method retrieves the username and password entered by the user, constructs a SQL query to check if the credentials match any records in the database, and verifies the user's role. If a match is found, the user is authenticated and granted access to the application.

3.2.11. registerform Class

- The **registerform** class manages the user registration process. It connects to the database to add new user records.
- The **connectToDatabase()** method establishes a connection to the database, enabling the application to insert new user data.
- The **reg()** method retrieves the user details from the registration form, constructs an SQL insert statement, and executes it to add the new user to the database. It also handles any validation to ensure that the input data is correct before attempting to insert it.

3.2.12. Employee Class

- The **Employee** class manages operations related to employee records in the database.
- The **connectToDatabase()** method establishes a connection to the database, allowing the application to perform operations on employee records.
- The **getConnection()** method returns the current database connection object, which is used by other methods to execute queries related to employee management.

3.2.13. Jobs Class

- The **Jobs** class manages operations related to job records in the database.
- The **connectToDatabase()** method establishes a connection to the database, enabling the application to perform operations on job records.
- The **getConnection()** method returns the current database connection object, which is used by other methods to execute queries related to job management.

3.2.14. Stock Class

- The **Stock** class manages operations related to stock records in the database.
- The **connectToDatabase()** method establishes a connection to the database, allowing the application to perform operations on stock records.
- The **getConnection()** method returns the current database connection object, which is used by other methods to execute queries related to stock management.

3.3. USAGE OF PROCEDURES

A procedure in DBMS is a set of SQL statement grouped together to perform a specifc task. It is a type of stored program in which you can define logic that is executed on database server .In our project we have used procedures to alert the employee about the reduction in stock count.

(i)Manager view

```
CREATE PROCEDURE GetLowStockResources()

BEGIN

   SELECT r.rid, r.rname, r.rquantity, s.sname, s.contact,

         CASE

            WHEN r.rquantity <= 5 THEN 'Low Stock Warning'

            ELSE ''

         END AS error_message,

         CASE

            WHEN r.rquantity <= 5 THEN 'Order ASAP'
            ELSE 'Sufficient'
        END AS order_details
    FROM resources r
    JOIN supplier s ON r.sname = s.sname;
END$$
```

(ii) Employee resources

```
CREATE PROCEDURE GetLowStock()
BEGIN
    SELECT r.rid, r.rname, r.rquantity, s.sname
        CASE
            WHEN r.rquantity <= 5 THEN 'Low Stock Warning'
            ELSE ''
        END AS error_message,
        CASE
            WHEN r.rquantity <= 5 THEN 'Order ASAP'
            ELSE 'Sufficient'
        END AS order_details
    FROM resources r
    JOIN supplier s ON r.sname = s.sname;
END$$
```

CHAPTER 4

PROGRAMS

4.1.JAVA CODE

LOGIN PAGE:

```
package miniproject;

import javax.swing.JOptionPane;

import java.sql.*;

public class LoginFrame extends javax.swing.JFrame {

     loginform l=new loginform();

    public LoginFrame() {

        initComponents();

        setLocationRelativeTo(null);

        l.connectToDatabase();

        setLocationRelativeTo(null);

    }

private void RegisterActionPerformed(java.awt.event.ActionEvent evt) {

         Registeration f=new Registeration();

       f.show();

        this.dispose();

    }

    private void ShowpasswordActionPerformed(java.awt.event.ActionEvent evt) {

                  if(Showpassword.isSelected()){

            Passfield.setEchoChar((char) 0); //show password

        }

        else{
            Passfield.setEchoChar('*');//Hide Password

         }

    }

    private void LogbuttonActionPerformed(java.awt.event.ActionEvent evt) {

        if(Usertext.getText().equals("")){

                JOptionPane.showMessageDialog(null, "Please fill out username");

            }

            else if(Passfield.getText().equals("")){

                JOptionPane.showMessageDialog(null,"Please fill out password");

            }

            else{

                l.login();

            }

        this.dispose();

    }

  public class loginform{

        private Connection conn ;

    public void connectToDatabase() {

        try {

                    Class.forName("com.mysql.cj.jdbc.Driver");

            String url
="jdbc:mysql://localhost:3306/SKFARTS?useSSL=false&allowPublicKeyRetrieval=true";

            String user = "root";

            String password = "Harini@05";

            conn = DriverManager.getConnection(url, user, password);

            System.out.println("Connected successfully!");

            JOptionPane.showMessageDialog(null, "Database connected successfully!");
        } catch (ClassNotFoundException e) {
            JOptionPane.showMessageDialog(null, "MySQL JDBC Driver not found!");
            e.printStackTrace();
        } catch (SQLException e) {
            JOptionPane.showMessageDialog(null, "Database connection failed!\n" + e.getMessage());
            e.printStackTrace();
        }
    }
        public void login(){
        String username=Usertext.getText();
        String password=Passfield.getText();
        String role = Rolecombobox.getSelectedItem().toString();
        try{
            String query="select * from users where username = ? and password= ? and role= ?";
            //prepareStatement pt=conn.prepareStatement(query);
            PreparedStatement pt = conn.prepareStatement(query);
            pt.setString(1,username);
            pt.setString(2,password);
            pt.setString(3, role);
            ResultSet rs=pt.executeQuery();
                    if(rs.next()){
                if (role.equals("MANAGER")) {
                  ManagerFrame managerFrame = new ManagerFrame();
                  managerFrame.setVisible(true);
                }
                  else
```

```
             {
              EmployeeFrame employeeFrame = new EmployeeFrame();
              employeeFrame.setVisible(true);
   }
        }else{
           JOptionPane.showMessageDialog(null, "invalid");
        }
     }
           catch(Exception e){
        JOptionPane.showMessageDialog(null, "Login failed!\n" + e.getMessage());
        e.printStackTrace();
              }
   }}
   public static void main(String args[]) {
      java.awt.EventQueue.invokeLater(new Runnable() {
         public void run() {
            new LoginFrame().setVisible(true);
         }
      });
   }
```

REGISTRATION PAGE:

```
package miniproject;
import javax.swing.JOptionPane;
import java.sql.*;
public class Registeration extends javax.swing.JFrame {
   public Registeration() {
        initComponents();

        setLocationRelativeTo(null);

    }

        private void REgbuttonActionPerformed(java.awt.event.ActionEvent evt) {

         if(Nametext.getText().equals("")){

                JOptionPane.showMessageDialog(null, "Please fill out name");

            }else if(Emailtext.getText().equals("")){

                JOptionPane.showMessageDialog(null, "Please fill out email id");

            }else if (!Emailtext.getText().matches("^[A-Za-z0-9_.-]+@[A-Za-z0-9.-]+$")) {

                JOptionPane.showMessageDialog(this, "Invalid email format.", "Error",
JOptionPane.ERROR_MESSAGE);

            }else if(Setpasstext.getText().equals("")){

                JOptionPane.showMessageDialog(null, "Please fill out password");

            }else if(Setpasstext.getText().length()<6){

               JOptionPane.showMessageDialog(null, "atleast 6 characters required in password");

            }

             else if(Phonetext.getText().length()!=10){

                JOptionPane.showMessageDialog(null, "Invalid phone no");

            }

            else if(Usernametext.getText().equals("")){

                JOptionPane.showMessageDialog(null, "Please fill out username");

            }

            else if(Conpasstext.getText().equals("")){

                JOptionPane.showMessageDialog(null,"Please fill out confirm password");

            }

            else{

              r.reg();
```

```
                        }
            }
            public class registerform{
                private Connection conn ;
                public registerform(){
                     connectToDatabase();
                }
            public void connectToDatabase() {
                try {
                    Class.forName("com.mysql.cj.jdbc.Driver");
                    String url = "jdbc:mysql://localhost:3306/?user";
                    String url =
"jdbc:mysql://localhost:3306/SKFARTS?useSSL=false&allowPublicKeyRetrieval=true";
                    String user = "root";
                    String password = "Harini@05";
                    conn = DriverManager.getConnection(url, user, password);
                    System.out.println("Connected successfully!");
                    JOptionPane.showMessageDialog(null, "Database connected successfully!");
                } catch (ClassNotFoundException e) {
                    JOptionPane.showMessageDialog(null, "MySQL JDBC Driver not found!");
                    e.printStackTrace();
                } catch (SQLException e) {
                    JOptionPane.showMessageDialog(null, "Database connection failed!\n" + e.getMessage());
                    e.printStackTrace();
                }
            }
                public void reg(){
```

```
        String name=Nametext.getText();

        String username=Usernametext.getText();

        String gmail=Emailtext.getText();

        String phnno=Phonetext.getText();

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

        String gender=Gendercombobox.getSelectedItem().toString();

        String role= "employee";

            try{

          String query="insert into
users(Name,Email,Gender,Phoneno,Username,Password,Role)values(?,?,?,?,?,?,?)";

          //prepareStatement pt=conn.prepareStatement(query);

          PreparedStatement pt = conn.prepareStatement(query);

          pt.setString(1,name);

          pt.setString(2,gmail);

          pt.setString(3,gender);

          pt.setString(4,phnno);

          pt.setString(5,username);

          pt.setString(6,password);

          pt.setString(7,role);

          pt.executeUpdate();

          JOptionPane.showMessageDialog(null, "registered successfully!");

        }

        catch(Exception e){

          JOptionPane.showMessageDialog(null, "Login failed!\n" + e.getMessage());

          e.printStackTrace();

        }

      }}
```

```
       public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new Registeration().setVisible(true);
            }       });       }
```

MANAGER FRAME:

```
public class ManagerFrame extends javax.swing.JFrame {
        public ManagerFrame() {
        JOptionPane.showMessageDialog(this, "Welcome, Manager!");
    }
    private void LogoutActionPerformed(java.awt.event.ActionEvent evt) {
        this.dispose();
        new LoginFrame().setVisible(true);
    }
    private void StockbuttActionPerformed(java.awt.event.ActionEvent evt) {
        this.dispose();
        new StockManage().setVisible(true);
    }
    private void EmpbuttActionPerformed(java.awt.event.ActionEvent evt) {
       this.dispose();
        new EmployeeManage().setVisible(true);
    }
    private void JobbuttActionPerformed(java.awt.event.ActionEvent evt) {
        this.dispose();
        new JobsManage().setVisible(true);
    }
    private void ViewbuttActionPerformed(java.awt.event.ActionEvent evt) {
        this.dispose();
        new ViewStockFrame().setVisible(true);
    }
    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new ManagerFrame().setVisible(true);
            }
```

EMPLOYEE MANAGEMENT:

```
import javax.swing.table.DefaultTableModel;
public class EmployeeManage extends javax.swing.JFrame {
    Employee e=new Employee();
    public EmployeeManage() {
        initComponents();
        try {
            showEmployee();
        } catch (SQLException ex) {
            Logger.getLogger(EmployeeManage.class.getName()).log(Level.SEVERE, null, ex);
        }
  }
   private void BackbuttActionPerformed(java.awt.event.ActionEvent evt) {
         this.dispose(); // Close the current frame
        ManagerFrame m = new ManagerFrame(); // Create instance of ManagerFrame
        m.setVisible(true);
    }
    private int updateEmpId = -1;

    private void EditbuttActionPerformed(java.awt.event.ActionEvent evt) {

       try {

        if (updateEmpId == -1) {

                    String empIdInput = JOptionPane.showInputDialog(null, "Enter Employee ID to
Update:");

           if (empIdInput != null && !empIdInput.trim().isEmpty()) {

               int empId = Integer.parseInt(empIdInput);

               Connection conn = e.getConnection();

               String selectSql = "SELECT ename, role, contact FROM employees WHERE eid=?";

               PreparedStatement selectStmt = conn.prepareStatement(selectSql);

               selectStmt.setInt(1, empId);

               ResultSet rs = selectStmt.executeQuery();

               if (rs.next()) {

                   Empnamtext.setText(rs.getString("ename"));

                   Rolecombo.setSelectedItem(rs.getString("role"));

                   Contattext.setText(String.valueOf(rs.getLong("contact")));

                   updateEmpId = empId; // Store ID for next update

                   JOptionPane.showMessageDialog(null, "Employee details loaded.\nNow edit the fields and
click UPDATE again.");

               } else {

                   JOptionPane.showMessageDialog(null, "Employee ID not found.");

               }

           } else {

               JOptionPane.showMessageDialog(null, "Employee ID is required.");

           }

        }
        else {
           Connection conn = e.getConnection();
           String updateSql = "UPDATE employees SET ename=?, role=?, contact=? WHERE eid=?";
           PreparedStatement updateStmt = conn.prepareStatement(updateSql);
           updateStmt.setString(1, Empnamtext.getText());
           updateStmt.setString(2, Rolecombo.getSelectedItem().toString());
           updateStmt.setLong(3, Long.parseLong(Contattext.getText()));
           updateStmt.setInt(4, updateEmpId);
           int rowsAffected = updateStmt.executeUpdate();
           if (rowsAffected > 0) {
               JOptionPane.showMessageDialog(null, "Employee Updated Successfully");
               showEmployee(); // Refresh table
               Empnamtext.setText("");
               Rolecombo.setSelectedIndex(0);
               Contattext.setText("");
               updateEmpId = -1; // Reset for next update
           } else {
               JOptionPane.showMessageDialog(null, "Update Failed");
           }
        }
       private void InsertbuttActionPerformed(java.awt.event.ActionEvent evt) {
    String empName = Empnamtext.getText().trim();
    String role =Rolecombo.getSelectedItem().toString();
    String contactStr = Contattext.getText().trim();
    if (empName.isEmpty() || role.isEmpty() || contactStr.isEmpty()) {
        JOptionPane.showMessageDialog(this, "Please fill all fields.");
```

```
        return;
    }
    try {
        long contact = Long.parseLong(contactStr);
        String insertQuery = "INSERT INTO employees (ename, role, contact) VALUES (?, ?, ?)";
        Employee db = new Employee();
        Connection con = db.getConnection();
        PreparedStatement pt = con.prepareStatement(insertQuery);
        pt.setString(1, empName);
        pt.setString(2, role);
        pt.setLong(3, contact);
        int rowsAffected = pt.executeUpdate();
        if (rowsAffected > 0) {
            JOptionPane.showMessageDialog(this, "Employee added successfully!");
            Empnamtext.setText("");
            Rolecombo.setSelectedIndex(0);
            Contattext.setText("");
            showEmployee();
        } else {
            JOptionPane.showMessageDialog(this, "Insert failed. Try again.");
        }
    } catch (NumberFormatException ex) {
        JOptionPane.showMessageDialog(this, "Contact must be a number.");
    } catch (SQLException ex) {
        JOptionPane.showMessageDialog(this, "Database error: " + ex.getMessage());
        ex.printStackTrace();
```

```
}

}

private void DelbuttActionPerformed(java.awt.event.ActionEvent evt) {

try {

            String input = JOptionPane.showInputDialog(this, "Enter Employee ID to delete:");

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

        JOptionPane.showMessageDialog(this, "Deletion cancelled or no ID entered.");

        return;

    }

    int empId = Integer.parseInt(input.trim());

    Connection conn = e.getConnection();

    String sql = "DELETE FROM employees WHERE eid=?";

    PreparedStatement pt = conn.prepareStatement(sql);

    pt.setInt(1, empId);

    int rowsAffected = pt.executeUpdate();

    if (rowsAffected > 0) {

        JOptionPane.showMessageDialog(this, "Employee Deleted Successfully");

        showEmployee(); // Refresh table

    } else {

        JOptionPane.showMessageDialog(this, "Employee ID not found");

    }

   Empnamtext.setText("");

    Rolecombo.setSelectedIndex(0);

    Contattext.setText("");

} catch (NumberFormatException ex) {

    JOptionPane.showMessageDialog(this, "Invalid ID. Please enter a valid number.");
    } catch (Exception ex) {
        JOptionPane.showMessageDialog(this, "Delete Error: " + ex.getMessage());
      private void showEmployee() throws SQLException {
        try{
        Connection conn = e.getConnection();
        String sql = "SELECT * FROM employees";
        PreparedStatement pt = conn.prepareStatement(sql);
        ResultSet rs = pt.executeQuery();
        DefaultTableModel model = (DefaultTableModel) Protable.getModel();
        model.setRowCount(0);
       while (rs.next()) {
        model.addRow(new Object[]{
        rs.getInt("eid"),
        rs.getString("ename"),
        rs.getString("role"),
        rs.getLong("contact")
    });
        }
    } catch (Exception e) {
        JOptionPane.showMessageDialog(null, "Display Error: " + e.getMessage());
    }
}
   public class Employee{
        private Connection conn ;
        public Employee(){
             connectToDatabase();
        }
}
    public static void main(String args[]) {
                java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new EmployeeManage().setVisible(true);
```

JOBS MANAGEMENT:

```
public class JobsManage extends javax.swing.JFrame {
        }
    private void Ins1buttActionPerformed(java.awt.event.ActionEvent evt) {
   try {
    Connection conn = jobs.getConnection();
    String sql = "INSERT INTO machine (mid, name, type, status) VALUES (?, ?, ?, ?);";
    PreparedStatement ps = conn.prepareStatement(sql);
    ps.setString(1, Macidtext.getText());
    ps.setString(2, Maccombo.getSelectedItem().toString());
    ps.setString(3, Typecombo.getSelectedItem().toString());
    ps.setString(4, Statcombo.getSelectedItem().toString());
    ps.executeUpdate();
    JOptionPane.showMessageDialog(null, "Machine added successfully!");
    showMachine();
   Proidtext .setText("");
   Macidtext.setText("");
   Maccombo.setSelectedIndex(0);
   Typecombo.setSelectedIndex(0);
```

```
  Statcombo.setSelectedIndex(0);
} catch (Exception e) {
   e.printStackTrace();
   JOptionPane.showMessageDialog(null, "Error: " + e.getMessage());
}
   }
private void showMachine() {
       try{
       Connection conn = jobs.getConnection();
       String sql = "SELECT * FROM machine";
       PreparedStatement ps = conn.prepareStatement(sql);
       ResultSet rs = ps.executeQuery();
       DefaultTableModel model = (DefaultTableModel) Mactable1.getModel();
       model.setRowCount(0); // Clear table first
       while (rs.next()) {
   model.addRow(new Object[]{
       rs.getString("mid"),
       rs.getString("name"),
       rs.getString("type”),
       rs.getString("status”)
   });
}
   } catch (Exception e) {
       JOptionPane.showMessageDialog(null, "Display Error: " + e.getMessage());
   }
}
```

```
    private void showLam() {
    try {
        Connection conn = jobs.getConnection();
        String sql = "SELECT id, rid, mid, laminationdate, outputquantity FROM laminating";
        PreparedStatement ps = conn.prepareStatement(sql);
        ResultSet rs = ps.executeQuery();
        DefaultTableModel model = (DefaultTableModel) Lamtable.getModel();
        model.setRowCount(0); // Clear table first
        while (rs.next()) {
            System.out.println("MID: " + rs.getString("mid"));  // Debug print
            model.addRow(new Object[]{
                rs.getInt("id"),
                rs.getInt("rid"),
                rs.getString("mid"),
                rs.getDate("laminationdate"),
                rs.getInt("outputquantity")
            });
        }
    } catch (Exception e) {
        JOptionPane.showMessageDialog(null, "Display Error: " + e.getMessage());
    }
}
    private void showDesign() {
    try {
        Connection conn = jobs.getConnection();
        String sql = "SELECT * FROM designing";
        PreparedStatement ps = conn.prepareStatement(sql);
        ResultSet rs = ps.executeQuery();
        DefaultTableModel model = (DefaultTableModel) Desintabel.getModel();
        model.setRowCount(0);
        while (rs.next()) {
            model.addRow(new Object[]{
                rs.getInt("did"),
                rs.getString("softwareused"),
                rs.getString("projectname"),
                rs.getDate("creationdate"),
                rs.getInt("designtypeid")
            });
        }
    } catch (Exception e) {
        JOptionPane.showMessageDialog(null, "Display Error: " + e.getMessage());
    }
}
    private void BackbuttActionPerformed(java.awt.event.ActionEvent evt) {
            ManagerFrame m = new ManagerFrame(); // Create instance of ManagerFrame
            m.setVisible(true);
    }
    private void Del1buttActionPerformed(java.awt.event.ActionEvent evt) {
            try {
        String machineIdInput = JOptionPane.showInputDialog(null, "Enter Machine ID to Delete:");
        if (machineIdInput != null && !machineIdInput.trim().isEmpty()) {
            Connection conn = jobs.getConnection();
        String sql = "DELETE FROM machine WHERE mid=?";
        PreparedStatement ps = conn.prepareStatement(sql);
        ps.setString(1, machineIdInput);  //
        int rowsAffected = ps.executeUpdate();
        if (rowsAffected > 0) {
          JOptionPane.showMessageDialog(null, "Deleted Successfully");
          showMachine(); // Refresh table
        } else {
          JOptionPane.showMessageDialog(null, "Machine ID not found. Deletion failed.");
        }
        Macidtext.setText("");
        Maccombo.setSelectedIndex(0);
        Typecombo.setSelectedIndex(0);
        Statcombo.setSelectedIndex(0);
      } else {
        JOptionPane.showMessageDialog(null, "Machine ID is required to delete.");
      }
  } catch (Exception e) {
      JOptionPane.showMessageDialog(null, "Delete Error: " + e.getMessage());
  }
  }
private String updateMachineId = null;
private void Ed1buttActionPerformed(java.awt.event.ActionEvent evt) {
  try {
    if (updateMachineId == null) {
        String machineIdInput = JOptionPane.showInputDialog(null, "Enter Machine ID to Update:");
```

```
            if (machineIdInput != null && !machineIdInput.trim().isEmpty()) {
                Connection conn = jobs.getConnection();
                String selectSql = "SELECT name, type, status FROM machine WHERE mid=?";
                PreparedStatement selectStmt = conn.prepareStatement(selectSql);
                selectStmt.setString(1, machineIdInput);
                ResultSet rs = selectStmt.executeQuery();
                if (rs.next()) {
                    Macidtext.setText(machineIdInput);
                    Maccombo.setSelectedItem(rs.getString("name"));
                    Typecombo.setSelectedItem(rs.getString("type"));
                    Statcombo.setSelectedItem(rs.getString("status"));
                    updateMachineId = machineIdInput;
                    JOptionPane.showMessageDialog(null, "Machine details loaded.\nEdit fields and click
UPDATE again.");
                } else {
                    JOptionPane.showMessageDialog(null, "Machine ID not found.");
                }
            } else {
                JOptionPane.showMessageDialog(null, "Machine ID is required.");
            }
        } else {
            Connection conn = jobs.getConnection();
            String sql = "UPDATE machine SET name=?, type=?, status=? WHERE mid=?";
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setString(1, Maccombo.getSelectedItem().toString());
            ps.setString(2, Typecombo.getSelectedItem().toString());
            ps.setString(3, Statcombo.getSelectedItem().toString());
        ps.setString(4, updateMachineId);
        int rowsAffected = ps.executeUpdate();
        if (rowsAffected > 0) {
            JOptionPane.showMessageDialog(null, "Machine details updated successfully!");
            showMachine(); // Refresh table
            Proidtext.setText("");
            Macidtext.setText("");
            Maccombo.setSelectedIndex(0);
            Typecombo.setSelectedIndex(0);
            Statcombo.setSelectedIndex(0);
            updateMachineId = null; // Reset after update
        } else {
            JOptionPane.showMessageDialog(null, "Update Failed");
        }
    }
} catch (Exception e) {
    e.printStackTrace();
    JOptionPane.showMessageDialog(null, "Error: " + e.getMessage());
}
}
    public class Jobs{
    private Connection conn ;
    public Jobs(){
         connectToDatabase();
    }
}
```

```
    public static void main(String args[]) {
        }
```

STOCK MANAGEMENT:

```
public class StockManage extends javax.swing.JFrame {
    Stock s=new Stock();
        public StockManage() {
        initComponents();
        showStock();
    }
   private void EditbuttActionPerformed(java.awt.event.ActionEvent evt) {
       try {
        if (Proidtext.getText().trim().isEmpty()) {
            String productIdInput = JOptionPane.showInputDialog(null, "Enter Product ID to Update:");
                                if (productIdInput != null && !productIdInput.trim().isEmpty()) {
                Connection conn = s.getConnection();
                String sql = "SELECT rname, rquantity, purchase_date, sname, rtype FROM resources
WHERE rid=?";
                PreparedStatement ps = conn.prepareStatement(sql);
                ps.setInt(1, Integer.parseInt(productIdInput)); // Set the Product ID from dialog input
                ResultSet rs = ps.executeQuery();
                 if (rs.next()) {
                    Pronamtext.setText(rs.getString("rname"));
                    Qyttext.setText(String.valueOf(rs.getInt("rquantity")));
                    Pucdattext.setText(rs.getDate("purchase_date").toString());
                    Dealcombo.setSelectedItem(rs.getString("sname"));
                    Catecombo.setSelectedItem(rs.getString("rtype"));
                    JOptionPane.showMessageDialog(null, "Now edit the fields and click Edit again to save 
changes.");
                } else {
                    JOptionPane.showMessageDialog(null, "Product ID not found.");
                }
            } else {
                JOptionPane.showMessageDialog(null, "Product ID is required.");
            }
        } else {
            Connection conn = s.getConnection();
            String sql = "UPDATE resources SET rname=?, rquantity=?, purchase_date=?, sname=?, rtype=? 
WHERE rid=?";
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setString(1, Pronamtext.getText()); // Product Name
            ps.setInt(2, Integer.parseInt(Qyttext.getText())); // Quantity
            ps.setDate(3, java.sql.Date.valueOf(Pucdattext.getText())); // Purchase Date
            ps.setString(4, Dealcombo.getSelectedItem().toString()); // Supplier Name
            ps.setString(5, Catecombo.getSelectedItem().toString()); // Resource Type
            ps.setInt(6, Integer.parseInt(Proidtext.getText())); // Product ID
            int rowsAffected = ps.executeUpdate();
            if (rowsAffected > 0) {
                JOptionPane.showMessageDialog(null, "Product record updated successfully!");
                showStock(); // Refresh the table display
            } else {
                JOptionPane.showMessageDialog(null, "No matching record found to update.");
            }
             Proidtext.setText("");
```

```
            Pronamtext.setText("");
            Qyttext.setText("");
            Pucdattext.setText("");
            Dealcombo.setSelectedIndex(0);
            Unitcombo.setSelectedIndex(0);
            Catecombo.setSelectedIndex(0);
        }
    } catch (Exception e) {
        e.printStackTrace();
        JOptionPane.showMessageDialog(null, "Update Error: " + e.getMessage());
    }
    }
    private void BackbuttActionPerformed(java.awt.event.ActionEvent evt) {
             this.dispose();
    ManagerFrame m = new ManagerFrame(); // Create instance of ManagerFrame
    m.setVisible(true); // Show the Manager frame
    }
    private void InsertbuttActionPerformed(java.awt.event.ActionEvent evt) {
        try {
            Connection conn = s.getConnection();
    String sql = "INSERT INTO resources (rid,rname,rtype,rquantity,unit,purchase_date,sname) VALUES
(?, ?, ?, ?, ?, ?,?)";
    PreparedStatement ps = conn.prepareStatement(sql);
    ps.setInt(1, Integer.parseInt( Proidtext.getText()));
    ps.setString(2, Pronamtext.getText());
    ps.setString(3, Catecombo.getSelectedItem().toString());
    ps.setInt(4, Integer.parseInt(Qyttext.getText()));
    ps.setString(5, Unitcombo.getSelectedItem().toString());
         ps.setDate(6, java.sql.Date.valueOf(Pucdattext.getText()));
    ps.setString(7, Dealcombo.getSelectedItem().toString());
    ps.executeUpdate();
    JOptionPane.showMessageDialog(null, "Resource added successfully!");
    showStock();
   Proidtext .setText("");
Pronamtext.setText("");
Qyttext.setText("");
Pucdattext.setText("");
Dealcombo.setSelectedIndex(0);
Unitcombo.setSelectedIndex(0);
Catecombo.setSelectedIndex(0);
} catch (Exception e) {
    e.printStackTrace();
    JOptionPane.showMessageDialog(null, "Error: " + e.getMessage());
}
    }
    private void DelbuttActionPerformed(java.awt.event.ActionEvent evt) {
       try {
        String productIdInput = JOptionPane.showInputDialog(null, "Enter Product ID to Delete:");
        if (productIdInput != null && !productIdInput.trim().isEmpty()) {
            int productId = Integer.parseInt(productIdInput);
            Connection conn = s.getConnection();
            String sql = "DELETE FROM resources WHERE rid=?";
            PreparedStatement ps = conn.prepareStatement(sql);
```

```
          ps.setInt(1, productId); // Set the Product ID from the dialog input
         int rowsAffected = ps.executeUpdate();
          if (rowsAffected > 0) {
              JOptionPane.showMessageDialog(null, "Product deleted successfully!");
              showStock(); // Refresh the table display
          } else {
              JOptionPane.showMessageDialog(null, "Product ID not found.");
          }
         Proidtext.setText("");
          Pronamtext.setText("");
          Qyttext.setText("");
          Pucdattext.setText("");
          Dealcombo.setSelectedIndex(0);
          Unitcombo.setSelectedIndex(0);
          Catecombo.setSelectedIndex(0);
      } else {
          JOptionPane.showMessageDialog(null, "Product ID is required.");
      }
  } catch (Exception e) {
      e.printStackTrace();
      JOptionPane.showMessageDialog(null, "Delete Error: " + e.getMessage());
  }
  }
 private void PucdattextFocusGained(java.awt.event.FocusEvent evt) {
      if(Pucdattext.getText().equals("YYYY-MM-DD")){
          Pucdattext.setText("");
```

```
        Pucdattext.setForeground(Color.BLACK);
    }
}
private void PucdattextFocusLost(java.awt.event.FocusEvent evt) {
     if(Pucdattext.getText().isEmpty()){
        Pucdattext.setText("YYYY-MM-DD");
        Pucdattext.setForeground(Color.GRAY);
     }
}
private void showStock() {
    try{
    Connection conn = s.getConnection();
    String sql = "SELECT * FROM resources";
    PreparedStatement ps = conn.prepareStatement(sql);
    ResultSet rs = ps.executeQuery();
    DefaultTableModel model = (DefaultTableModel) Protable.getModel();
    model.setRowCount(0); // Clear table first
    while (rs.next()) {
        model.addRow(new Object[]{
            rs.getInt("rid"),
            rs.getString("rname"),
            rs.getInt("rquantity"),
            rs.getDate("purchase_date"),
            rs.getString("sname"),
            rs.getString("rtype")
        });
        }
        } catch (Exception e) {
        JOptionPane.showMessageDialog(null, "Display Error: " + e.getMessage());
    }
}
 public class Stock{
        private Connection conn ;
        public Stock(){
             connectToDatabase();
        }
}
     public static void main(String args[]) {
                java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new StockManage().setVisible(true);
            }
```

VIEW STOCK:

```
import java.util.Properties;
import javax.mail.*;
import javax.mail.internet.*;
public class ViewStockFrame extends JFrame {
    JButton showButton, backButton, mailButton;
    JTable stockTable;
    DefaultTableModel model;
    public ViewStockFrame() {
        setTitle("View Low Stock Resources");
        backButton.addActionListener(e -> {
            dispose();
            ManagerFrame m = new ManagerFrame();
            m.setVisible(true);
        });
        showButton.addActionListener(e -> loadData());
        mailButton.addActionListener(e -> {
            int selectedRow = stockTable.getSelectedRow();
            if (selectedRow == -1) {
                JOptionPane.showMessageDialog(this, "Please select a row to send email.");
                return;
            }
            String email = JOptionPane.showInputDialog(this, "Enter Supplier Email:");
            if (email == null || email.trim().isEmpty()) {
                JOptionPane.showMessageDialog(this, "Email not provided.");
                return;
            }
            String supplierName = model.getValueAt(selectedRow, 3).toString();
            String productName = model.getValueAt(selectedRow, 1).toString();
            String quantity = model.getValueAt(selectedRow, 2).toString();
            sendOrderEmail(email, supplierName, productName, quantity);
        });
        setVisible(true);
    }
    private void loadData() {
        model.setRowCount(0); // Clear existing rows
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection con = DriverManager.getConnection(
                    "jdbc:mysql://localhost:3306/SKFARTS", "root", "Harini@05"
            );
            CallableStatement cs = con.prepareCall("{CALL GetLowStockResources()}");
            ResultSet rs = cs.executeQuery();
            while (rs.next()) {
                model.addRow(new Object[]{
                        rs.getInt("rid"),
                        rs.getString("rname"),
                        rs.getInt("rquantity"),
                        rs.getString("sname"),
                        rs.getString("contact"),
                        rs.getString("error_message"),
                        rs.getString("order_details")
                });
            }
            rs.close();
            cs.close();
            con.close();
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage());
            ex.printStackTrace();
        }
    }
```

```
    public void sendOrderEmail(String recipientEmail, String supplierName, String productName, String
quantity) {
        String fromEmail = "skfarts1980@gmail.com";
        String password = "myld wqfl ncxs sqrx";
        Properties props = new Properties();
        props.put("mail.smtp.auth", "true");
        props.put("mail.smtp.starttls.enable", "true");
        props.put("mail.smtp.host", "smtp.gmail.com");
        props.put("mail.smtp.port", "587");
        Session session = Session.getInstance(props, new Authenticator() {
            protected PasswordAuthentication getPasswordAuthentication() {
                return new PasswordAuthentication(fromEmail, password);
            }
        });
        try {
            Message message = new MimeMessage(session);
            message.setFrom(new InternetAddress(fromEmail));
            message.setRecipients(Message.RecipientType.TO, InternetAddress.parse(recipientEmail));
            message.setSubject("Stock Order Request");
            String body = "Regards\n"+
                    "We have run out of stock for the following product:\n" +
                    "Product: " + productName + "\n" +
                    "Required Quantity: " + quantity + "\n\n" +
                    "Please arrange for delivery as soon as possible.\n\n" +
                    "Regards,\nSHREE KESAVAN FINE ARTS";
            message.setText(body);
            Transport.send(message);
            JOptionPane.showMessageDialog(null, "Email sent successfully to supplier!");
        } catch (MessagingException e) {
            e.printStackTrace();
            JOptionPane.showMessageDialog(null, "Failed to send email: " + e.getMessage());
        }
    }
    public static void main(String[] args) {
        new ViewStockFrame();
    }
}
```

EMPLOYEE FRAME:

```
public class EmployeeFrame extends javax.swing.JFrame {
    public EmployeeFrame() {
        initComponents();
        setLocationRelativeTo(null);
         JOptionPane.showMessageDialog(this, "Welcome, Employee!");
    }
    private void LogoutActionPerformed(java.awt.event.ActionEvent evt) {
        this.dispose(); // Close the ManagerDashboard
        new LoginFrame().setVisible(true);
    }
    private void StockbuttActionPerformed(java.awt.event.ActionEvent evt) {
        this.dispose();
        new Stockmanage1().setVisible(true);
    }
    private void JobbuttActionPerformed(java.awt.event.ActionEvent evt) {
```

```
              this.dispose();
        new DisplayTables().setVisible(true);
    }
    private void ViewbuttActionPerformed(java.awt.event.ActionEvent evt) {
        this.dispose();
        new ViewStock().setVisible(true);
          java.awt.EventQueue.invokeLater(new Runnable() {
           public void run() {
               new EmployeeFrame().setVisible(true);
           }
```

STOCK UPDATE:

```
public class Stockmanage1 extends javax.swing.JFrame {
        Stocks s=new Stocks();
      private Connection conn;
      public Stockmanage1() {
 initComponents();
      showStock();
    }
    private void BackbuttActionPerformed(java.awt.event.ActionEvent evt) {
        this.dispose();
        EmployeeFrame m = new EmployeeFrame();
        m.setVisible(true);
    }
    private void EditbuttActionPerformed(java.awt.event.ActionEvent evt) {
try {
        String productIdInput = JOptionPane.showInputDialog(null, "Enter Product ID to Edit:");
    if (productIdInput != null && !productIdInput.trim().isEmpty()) {
        int productId = Integer.parseInt(productIdInput);
        Connection conn = s.getConnection();
        String fetchSql = "SELECT rname, rquantity, purchase_date, sname, rtype FROM resources
WHERE rid=?";
        PreparedStatement psFetch = conn.prepareStatement(fetchSql);
        psFetch.setInt(1, productId);
        ResultSet rs = psFetch.executeQuery();
        if (rs.next()) {
            Pronamtext.setText(rs.getString("rname"));
            Qyttext.setText(String.valueOf(rs.getInt("rquantity")));
            Pucdattext.setText(rs.getDate("purchase_date").toString());
            Dealcombo.setSelectedItem(rs.getString("sname"));
            Catecombo.setSelectedItem(rs.getString("rtype"));
            Proidtext.setText(String.valueOf(productId));
            JOptionPane.showMessageDialog(null, "Now edit the fields and click Edit again to save
changes.");
        } else {
            JOptionPane.showMessageDialog(null, "Product ID not found.");
            return;
        }
        String updateSql = "UPDATE resources SET rname=?, rquantity=?, purchase_date=?, sname=?,
rtype=? WHERE rid=?";
        PreparedStatement psUpdate = conn.prepareStatement(updateSql);
        String proname = Pronamtext.getText();
        String quantity = Qyttext.getText();
        String purDate = Pucdattext.getText();
        String dealer = Dealcombo.getSelectedItem().toString();
```

```
            String category = Catecombo.getSelectedItem().toString();
            String proid = Proidtext.getText();
            if (proname.isEmpty() || quantity.isEmpty() || purDate.isEmpty() || proid.isEmpty()) {
                JOptionPane.showMessageDialog(null, "Please fill all fields before updating.");
                return;
            }
            try {
                psUpdate.setDate(3, java.sql.Date.valueOf(purDate));
            } catch (IllegalArgumentException ex) {
                JOptionPane.showMessageDialog(null, "Invalid Date Format. Please use YYYY-MM-DD.");
                return;
            }
            psUpdate.setString(4, dealer);
            psUpdate.setString(5, category);
            psUpdate.setInt(6, Integer.parseInt(proid));
            int rowsAffected = psUpdate.executeUpdate();
            if (rowsAffected > 0) {
                JOptionPane.showMessageDialog(null, "Updated Successfully");
            } else {
                JOptionPane.showMessageDialog(null, "No record found to update.");
            }
        } else {
            JOptionPane.showMessageDialog(null, "Product ID is required.");
        }
    } catch (Exception e) {
        JOptionPane.showMessageDialog(null, "Update Error: " + e.getMessage());
    }
    }
    private void showStock() {
        try{
         Connection conn = s.getConnection();
        String sql = "SELECT * FROM resources";
        PreparedStatement ps = conn.prepareStatement(sql);
        ResultSet rs = ps.executeQuery();
        DefaultTableModel model = (DefaultTableModel) Protable.getModel();
        model.setRowCount(0); // Clear table first
        while (rs.next()) {
            model.addRow(new Object[]{
                rs.getInt("rid"),
                rs.getString("rname"),
                rs.getInt("rquantity"),
                rs.getDate("purchase_date"),
                rs.getString("sname"),
                rs.getString("rtype")
            });
        }
         } catch (Exception e) {
        JOptionPane.showMessageDialog(null, "Display Error: " + e.getMessage());
    }
}
   public class Stocks {
    private Connection conn;
```

```
}
   public static void main(String args[]) {
      java.awt.EventQueue.invokeLater(new Runnable() {
         public void run() {
            new Stockmanage1().setVisible(true);
         }
      });
```

VIEW STOCK:

```
public class ViewStock extends JFrame {
   JButton showButton, backButton;
   JTable stockTable;
   DefaultTableModel model;
   public ViewStock() {
      setTitle("View Low Stock Resources");
           backButton = new JButton();
            backButton.addActionListener(e -> {
         dispose();
         EmployeeFrame m = new EmployeeFrame(); // Create instance of EmployeeFrame
         m.setVisible(true); // Opens EmployeeFrame
      });
      showButton.addActionListener(e -> loadData());
      setVisible(true);
   }
   private void loadData() {
      model.setRowCount(0);
      try {
         Class.forName("com.mysql.cj.jdbc.Driver");
         Connection con = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/SKFARTS", "root", "Harini@05"
         );
         CallableStatement cs = con.prepareCall("{CALL GetLowStockResources()}");
         ResultSet rs = cs.executeQuery();
         while (rs.next()) {
            model.addRow(new Object[] {
                rs.getInt("rid"),
                rs.getString("rname"),
                rs.getInt("rquantity"),
                rs.getString("sname"),
                rs.getString("error_message"),
                rs.getString("order_details")
            });
         }
      } catch (Exception ex) {
         JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage());
         ex.printStackTrace();
}
   }
   public static void main(String[] args) {
      new ViewStock();
   }
}
```

DISPLAY JOBS:

```
public class DisplayTables extends JFrame {
    private Connection conn;
    private JTabbedPane tabbedPane;
    public DisplayTables() {
        setTitle("Database Viewer - SKFARTS");
   }
    private JPanel createTablePanel(String sql) {
        try {
            PreparedStatement ps = conn.prepareStatement(sql);
            ResultSet rs = ps.executeQuery();
            ResultSetMetaData meta = rs.getMetaData();
            int colCount = meta.getColumnCount();
            model.setColumnCount(0);
            for (int i = 1; i <= colCount; i++) {
                model.addColumn(meta.getColumnName(i));
            }
            while (rs.next()) {
                Object[] rowData = new Object[colCount];
                for (int i = 1; i <= colCount; i++) {
                    rowData[i - 1] = rs.getObject(i);
                }
                model.addRow(rowData);
            }
        } catch (Exception e) {
            e.printStackTrace();
            JOptionPane.showMessageDialog(null, "Error loading data: " + e.getMessage());
        }
        JScrollPane scrollPane = new JScrollPane(table);
        panel.add(scrollPane, BorderLayout.CENTER);
        return panel;
    }
    private void refreshAllTabs() {
        tabbedPane.setComponentAt(0, createTablePanel("SELECT * FROM machine"));
        tabbedPane.setComponentAt(1, createTablePanel("SELECT * FROM scoring"));
        tabbedPane.setComponentAt(2, createTablePanel("SELECT * FROM laminating"));
        tabbedPane.setComponentAt(3, createTablePanel("SELECT * FROM designtype"));
        tabbedPane.setComponentAt(4, createTablePanel("SELECT * FROM designing"));
    }
    public static void main(String[] args) {
         new DisplayTables();
    }
}
```

4.2. DBMS QUERIES

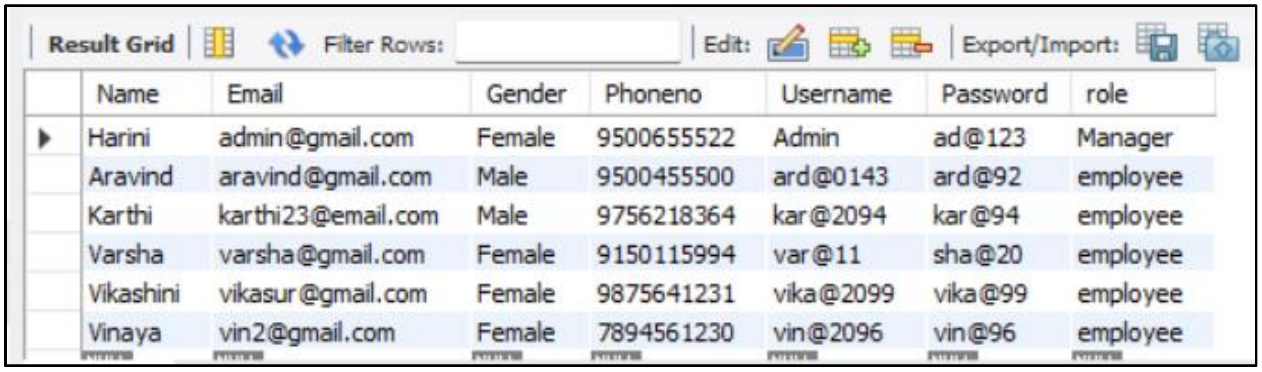
```
USE SKFARTS;
CREATE TABLE users (
    Name VARCHAR(20) NOT NULL,
    Email VARCHAR(20) NOT NULL UNIQUE,
    Dob DATE NOT NULL,
    Gender ENUM('Male', 'Female', 'Other') NOT NULL,
    Phoneno VARCHAR(9) NOT NULL UNIQUE,
    Username VARCHAR(10) NOT NULL UNIQUE,
    password VARCHAR(8) NOT NULL
```

```
);
CREATE TABLE supplier (
        sid INT PRIMARY KEY,
        sname VARCHAR(15)UNIQUE NOT NULL,
        contact BIGINT UNIQUE NOT NULL,
        remail VARCHAR(15) UNIQUE NOT NULL,
        address VARCHAR(20) NOT NULL
    );
CREATE TABLE resources (
    rid INT PRIMARY KEY,
    rname VARCHAR(15) NOT NULL,
    rtype VARCHAR(12) NOT NULL,
    rquantity INT CHECK (rquantity >= 0),
    unit INT NOT NULL,
    sname VARCHAR(15),
    FOREIGN KEY (sname) REFERENCES supplier(sname) ON DELETE SET NULL
);
CREATE TABLE employees (
        eid INT PRIMARY KEY,
        ename VARCHAR(15) NOT NULL,
        role VARCHAR(12) NOT NULL,
       contact BIGINT UNIQUE NOT NULL
    );
CREATE TABLE resources (
    rid INT PRIMARY KEY,
    rname VARCHAR(15) NOT NULL,
    rtype VARCHAR(12) NOT NULL,
    rquantity INT CHECK (rquantity >= 0),
    unit INT NOT NULL,
    purchase_date DATE NOT NULL,
    sname VARCHAR(15),
);
CREATE TABLE machine (
         mid INT PRIMARY KEY,
         name VARCHAR(15) NOT NULL,
      type VARCHAR(15) NOT NULL,
     status VARCHAR(12) CHECK (status IN ('Active', 'Inactive', 'Maintenance')));
CREATE TABLE scoring (
           rid INT NOT NULL,
          scoringdate DATE NOT NULL,
         outputquantity INT CHECK (outputquantity >= 0),
         PRIMARY KEY (rid),
CREATE TABLE laminating (
        id INT AUTO_INCREMENT PRIMARY KEY,
         rid INT NOT NULL,
         mid VARCHAR(10) NULL,
         laminationdate DATE NOT NULL,
         outputquantity INT CHECK (outputquantity >= 0),
        CREATE TABLE designtype (
        designtypeid INT PRIMARY KEY,
         typename VARCHAR(20) NOT NULL UNIQUE
     );
     CREATE TABLE designing (
         did INT PRIMARY KEY,
         softwareused VARCHAR(20) NOT NULL,
         projectname VARCHAR(20) NOT NULL,
         creationdate DATE NOT NULL,
         designtypeid INT NOT NULL,
    );
CREATE PROCEDURE GetLowStockResources()
BEGIN
    SELECT r.rid, r.rname, r.rquantity, s.sname, s.contact,
           CASE
               WHEN r.rquantity <= 5 THEN 'Low Stock Warning'
               ELSE ''
           END AS error_message,
           CASE
               WHEN r.rquantity <= 5 THEN 'Order ASAP'
               ELSE 'Sufficient'
           END AS order_details
    FROM resources r
    JOIN supplier s ON r.sname = s.sname;
END$$
```

CHAPTER 5

RESULTS

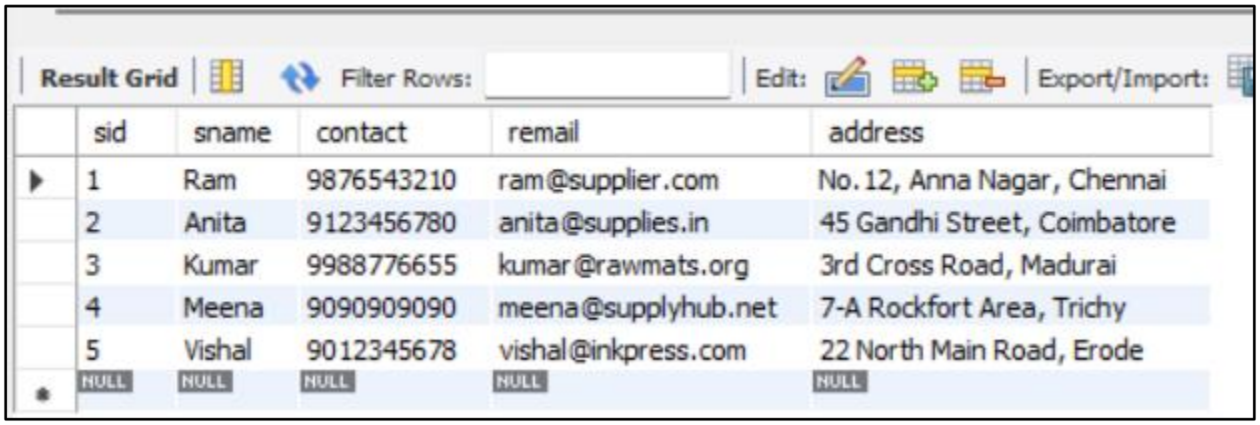
To support the functionality of the SKF Resource Register application, a structured backend was developed using MySQL. The database consists of well-designed tables for managing user roles, machine details, resource usage, stock information, and departmental data. Below are the screenshots of the key tables used in the application to ensure data consistency and efficient operations.



Name	Email	Gender	Phoneno	Username	Password	role
Harini	admin@gmail.com	Female	9500655522	Admin	ad@123	Manager
Aravind	aravind@gmail.com	Male	9500455500	ard@0143	ard@92	employee
Karthi	karthi23@email.com	Male	9756218364	kar@2094	kar@94	employee
Varsha	varsha@gmail.com	Female	9150115994	var@11	sha@20	employee
Vikashini	vikasur@gmail.com	Female	9875641231	vika@2099	vika@99	employee
Vinaya	vin2@gmail.com	Female	7894561230	vin@2096	vin@96	employee

FIG 5.1 LOGIN TABLE

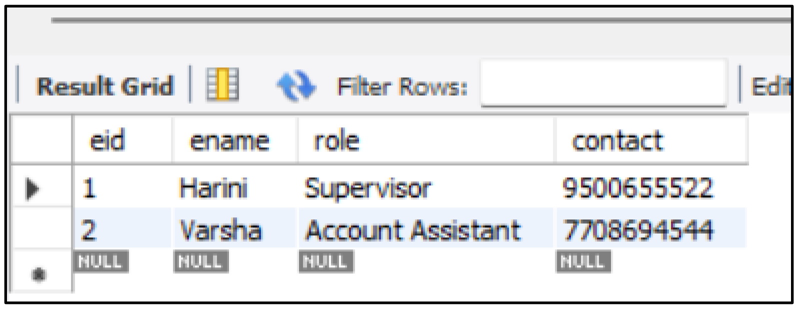
FIG 5.1 refers to the login table where it displays the authenticated users in the database.



sid	sname	contact	remail	address
1	Ram	9876543210	ram@supplier.com	No. 12, Anna Nagar, Chennai
2	Anita	9123456780	anita@supplies.in	45 Gandhi Street, Coimbatore
3	Kumar	9988776655	kumar@rawmats.org	3rd Cross Road, Madurai
4	Meena	9090909090	meena@supplyhub.net	7-A Rockfort Area, Trichy
5	Vishal	9012345678	vishal@inkpress.com	22 North Main Road, Erode
NULL	NULL	NULL	NULL	NULL

FIG 5.2 REGISTRATION TABLE

FIG 5.2 refers to the registration table where it displays the registered user in the database.

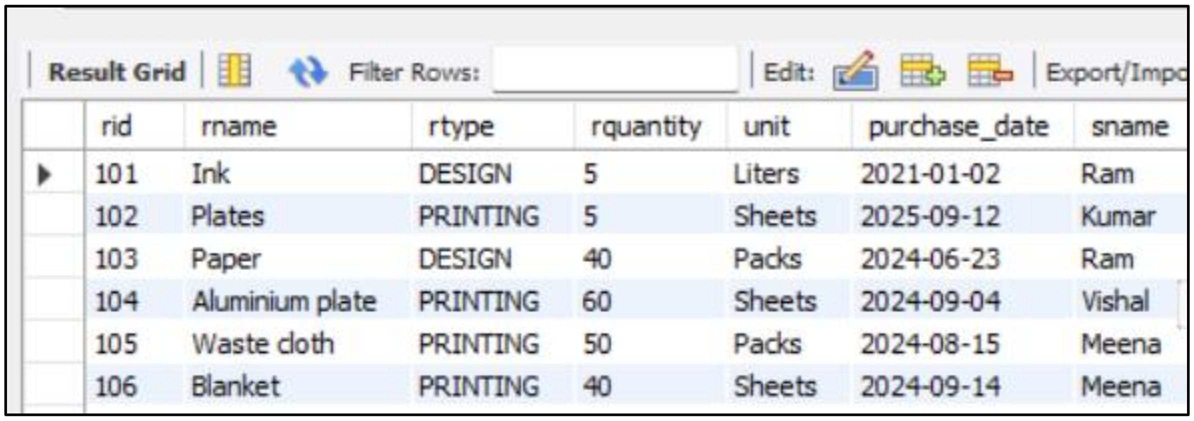


Result Grid | Filter Rows: | Edit

eid	ename	role	contact
1	Harini	Supervisor	9500655522
2	Varsha	Account Assistant	7708694544
NULL	NULL	NULL	NULL

FIG 5.3 EMPLOYEE DETAILS TABLE

FIG 5.3 refers to the employee details table where the employee details and the role is displayed in the database.

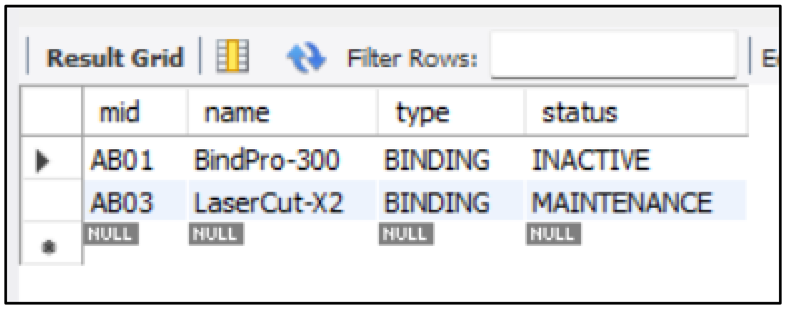


Result Grid | Filter Rows: | Edit: | Export/Impo

rid	rname	rtype	rquantity	unit	purchase_date	sname
101	Ink	DESIGN	5	Liters	2021-01-02	Ram
102	Plates	PRINTING	5	Sheets	2025-09-12	Kumar
103	Paper	DESIGN	40	Packs	2024-06-23	Ram
104	Aluminium plate	PRINTING	60	Sheets	2024-09-04	Vishal
105	Waste cloth	PRINTING	50	Packs	2024-08-15	Meena
106	Blanket	PRINTING	40	Sheets	2024-09-14	Meena

FIG 5.4 RESOURCE DETAILS TABLE

FIG 5.4 refers to the resource details table where it displays the resource name,id,type,etc…in the database.

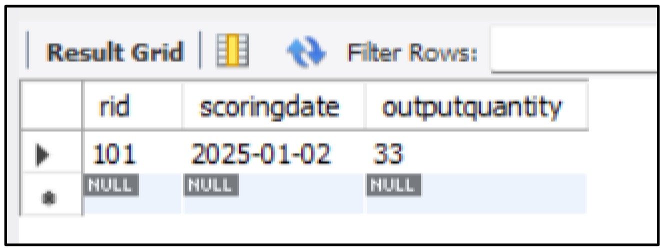


Result Grid | Filter Rows: | E

mid	name	type	status
AB01	BindPro-300	BINDING	INACTIVE
AB03	LaserCut-X2	BINDING	MAINTENANCE
NULL	NULL	NULL	NULL

FIG 5.5 MACHINE DETAILS TABLE

FIG 5.5 refers to the machine details table where it displays the machine id,m=name,etc…in the database.

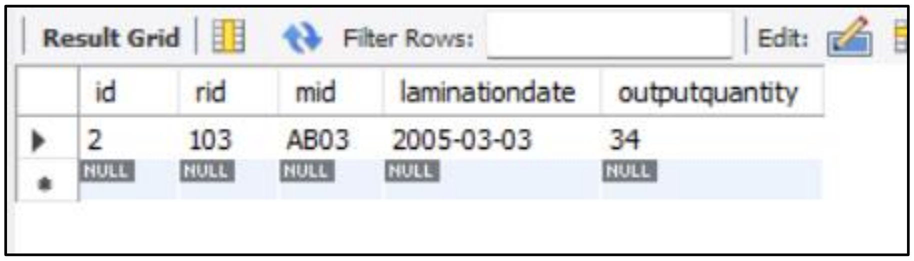


Result Grid | Filter Rows:

rid	scoringdate	outputquantity
101	2025-01-02	33
NULL	NULL	NULL

FIG 5.6 SCORING DETAILS TTABLE

FIG 5.6 refers to the scoring details table where it displays the scoring id ,name ,quantity in the database.

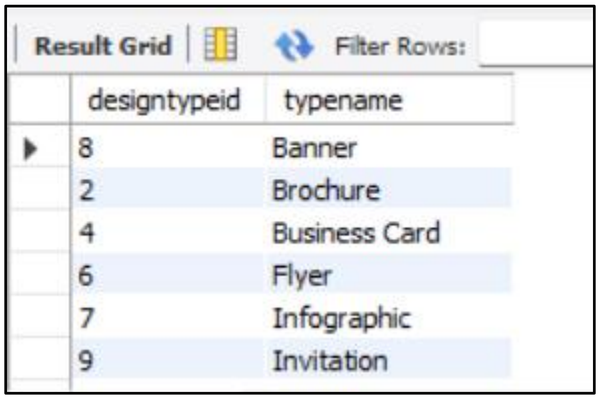


Result Grid | Filter Rows: | Edit:

id	rid	mid	laminationdate	outputquantity
2	103	AB03	2005-03-03	34
NULL	NULL	NULL	NULL	NULL

FIG 5.7 LAMINATION DETAILS TABLE

FIG 5.7 refers to the lamination details table where it displays the lamination id, name, output, etc… in the database.

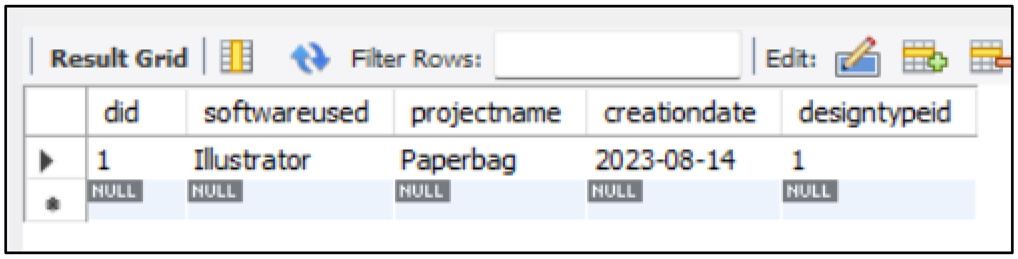


Result Grid | Filter Rows:

designtypeid	typename
8	Banner
2	Brochure
4	Business Card
6	Flyer
7	Infographic
9	Invitation

FIG 5.8 DESIGNTYPE DETAILS TABLE

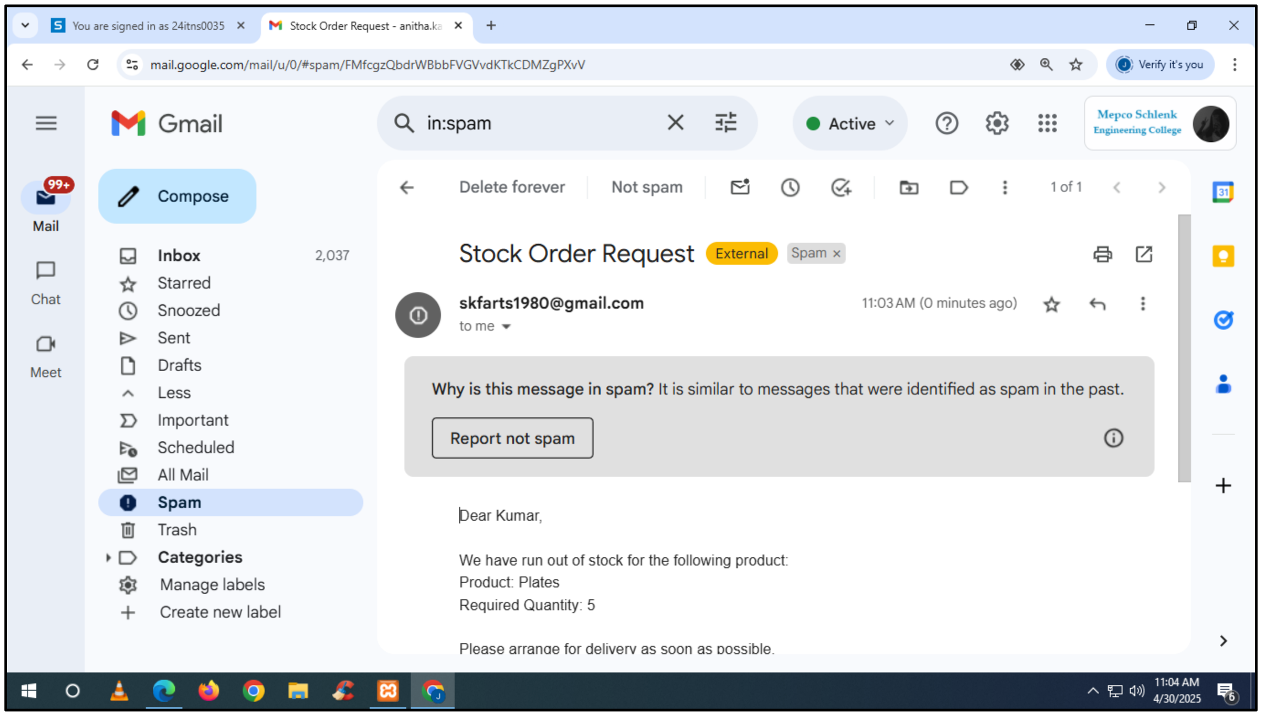
FIG 5.8 refers to designtype details table where it displays the designtype id, name in the database.



	did	softwareused	projectname	creationdate	designtypeid
▶	1	Illustrator	Paperbag	2023-08-14	1
	NULL	NULL	NULL	NULL	NULL

FIG 5.9 DESIGNING DETAILS TABLE

FIG 5.9 refers to the designing details table where it displays the designing id ,softwareused,etc..



Stock Order Request External Spam

skfarts1980@gmail.com 11:03 AM (0 minutes ago)
to me

Why is this message in spam? It is similar to messages that were identified as spam in the past.

Dear Kumar,

We have run out of stock for the following product:
Product: Plates
Required Quantity: 5

Please arrange for delivery as soon as possible.

FIG 5.10 LOW STOCK WARNING

FIG 5.10 refers to notifying the supplier about low stock using email

5.1. SCREEN SHOTS

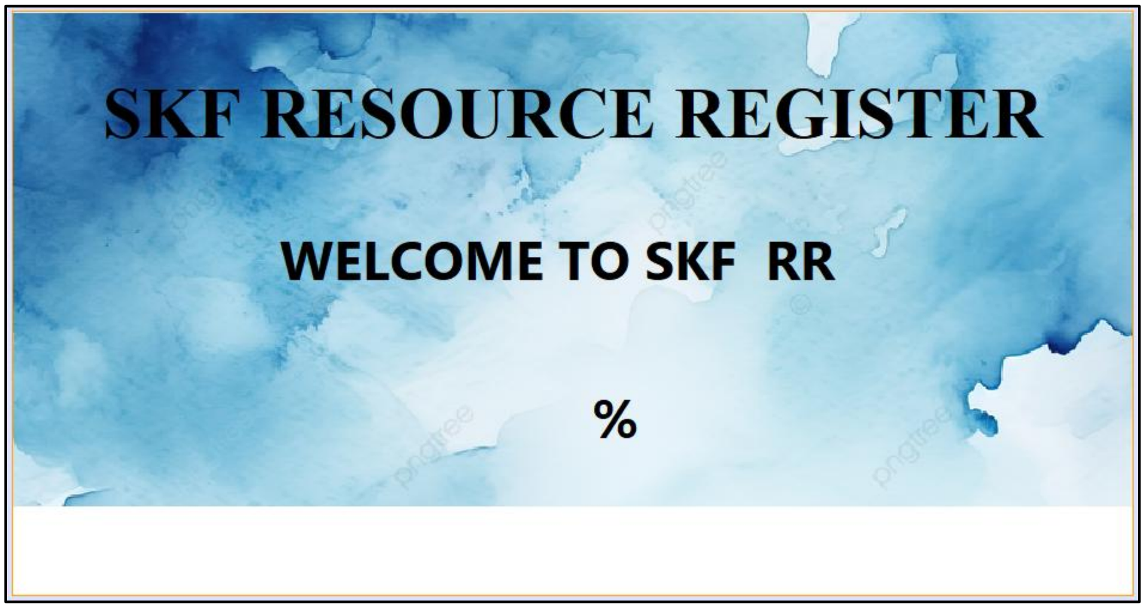


FIG 5.1.1 LOADING FRAME

FIG 5.1.1 refers to the loading frame where it is the front page of the project.

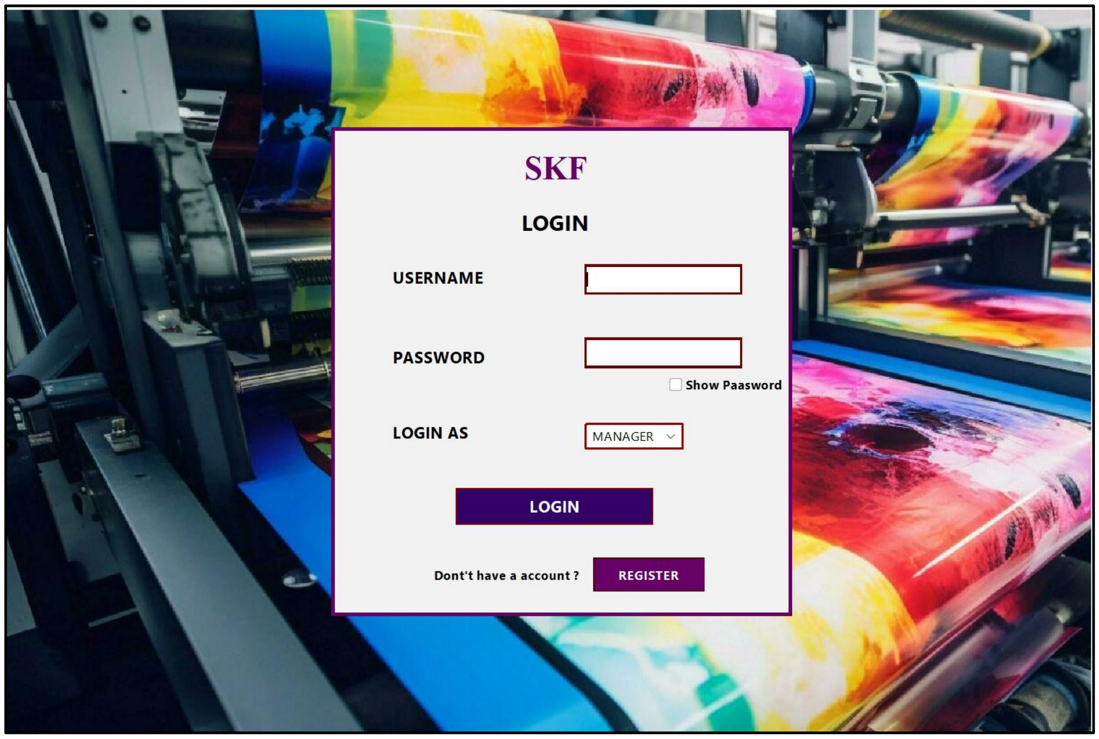


FIG 5.1.2 LOGIN FRAMEFIG 5.1.2 refers to the login frame where the registered users can login.

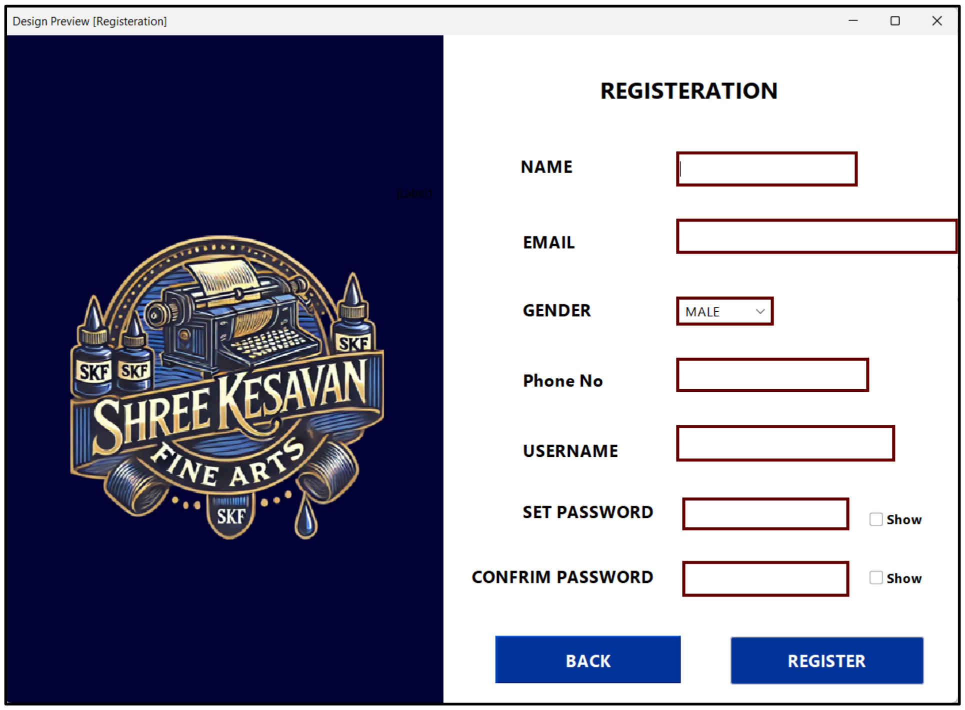


FIG 5.1.3 REGISTRATION FRAME

FIG 5.1.3 refers to the registration frame here the user can register their details.

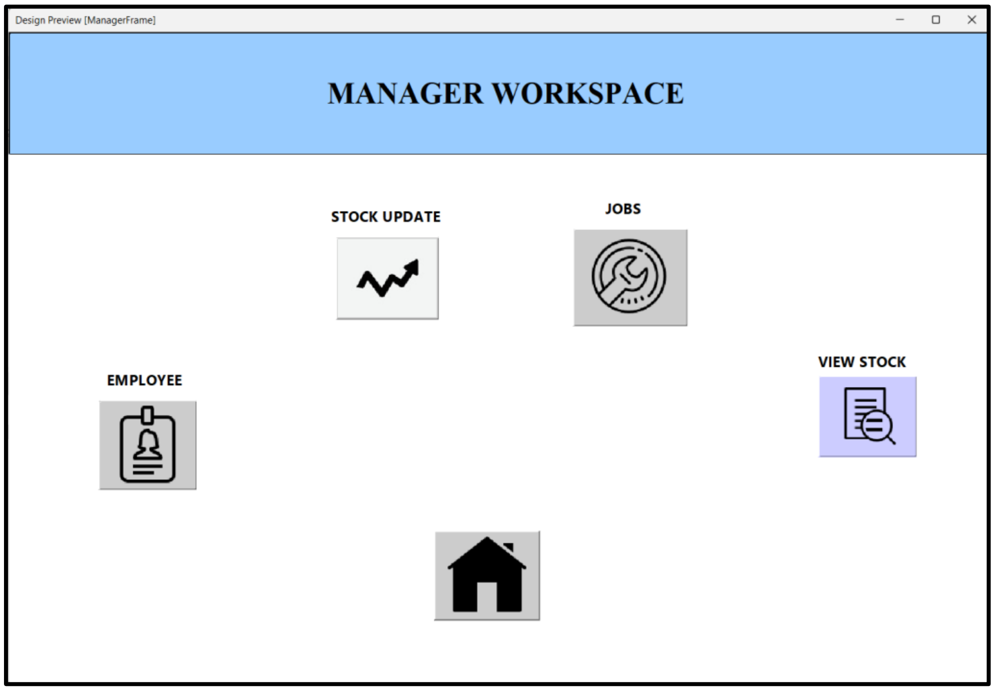


FIG 5.1.4 MANAGER FRAME

FIG 5.1.4 refers to the manager frame here it shows the privileges of the manager.

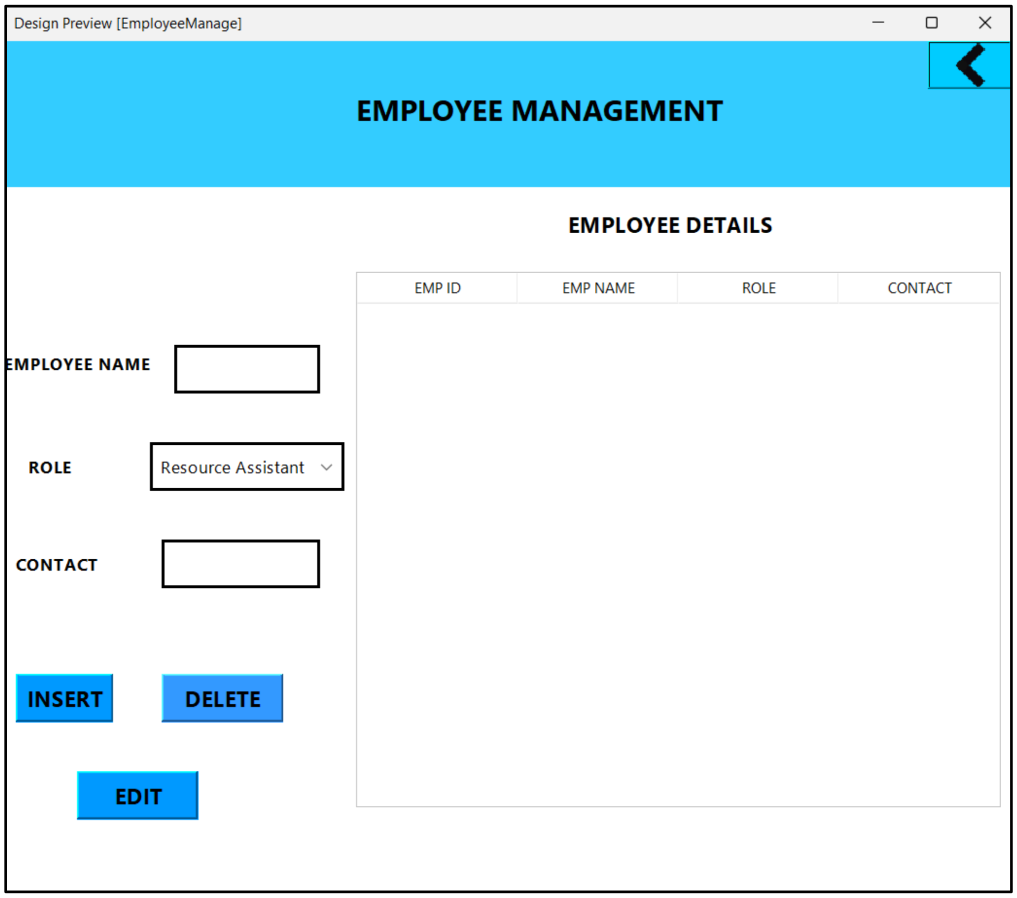
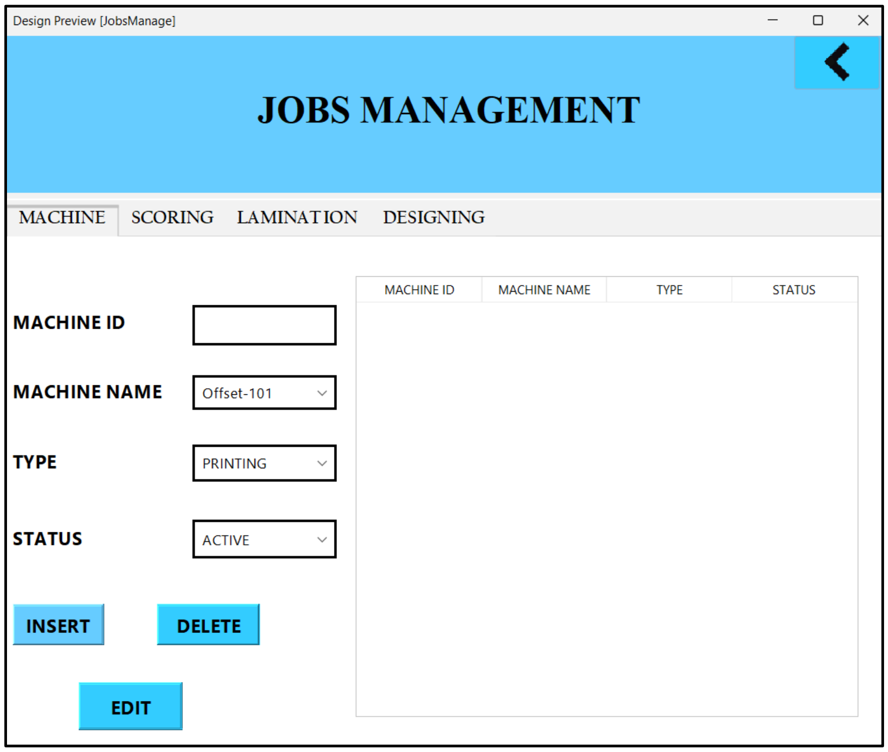


FIG 5.1.5 EMPLOYEE MANAGEMENT FRAME

FIG 5.1.5 refers to the employee management frame here the employee details are entered.



Design Preview [JobsManage]

JOBS MANAGEMENT

MACHINE SCORING LAMINATION DESIGNING

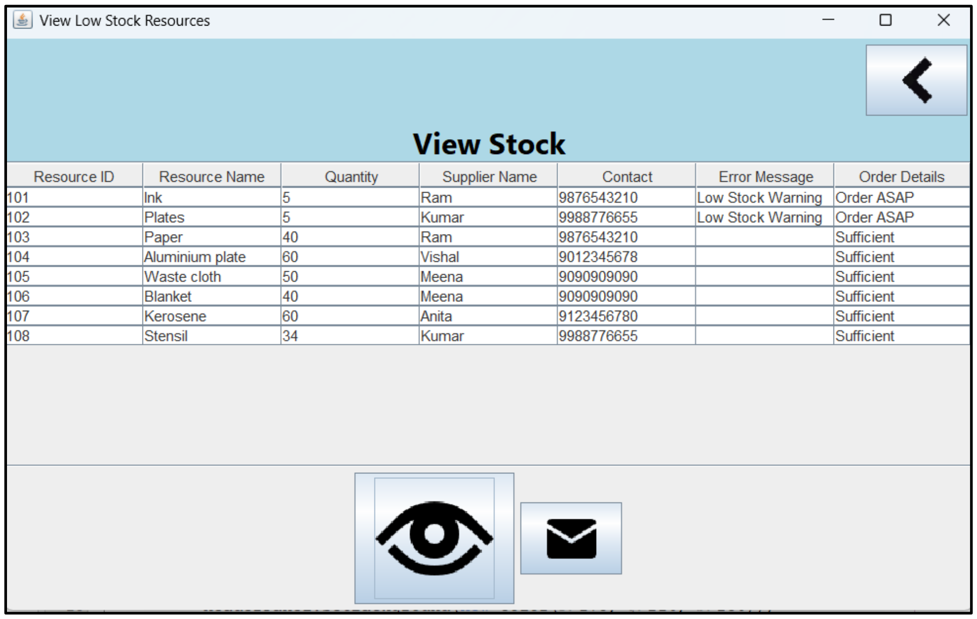
MACHINE ID	
MACHINE NAME	Offset-101
TYPE	PRINTING
STATUS	ACTIVE

INSERT DELETE EDIT

MACHINE ID	MACHINE NAME	TYPE	STATUS

FIG 5.1.6 JOBS MANAGEMENT FRAME

FIG 5.1.6 refers to the jobs management frame where the details of the jobs are entered .Here it contains machine ,scoring ,lamination and designing frame where the details are entered.



View Low Stock Resources

View Stock

Resource ID	Resource Name	Quantity	Supplier Name	Contact	Error Message	Order Details
101	Ink	5	Ram	9876543210	Low Stock Warning	Order ASAP
102	Plates	5	Kumar	9988776655	Low Stock Warning	Order ASAP
103	Paper	40	Ram	9876543210		Sufficient
104	Aluminium plate	60	Vishal	9012345678		Sufficient
105	Waste cloth	50	Meena	9090909090		Sufficient
106	Blanket	40	Meena	9090909090		Sufficient
107	Kerosene	60	Anita	9123456780		Sufficient
108	Stensil	34	Kumar	9988776655		Sufficient

FIG 5.1.7 VIEW STOCK FRAME

FIG 5.1.7 refers to the view stock frame where the stock can be viewed and lowstock mail is sent

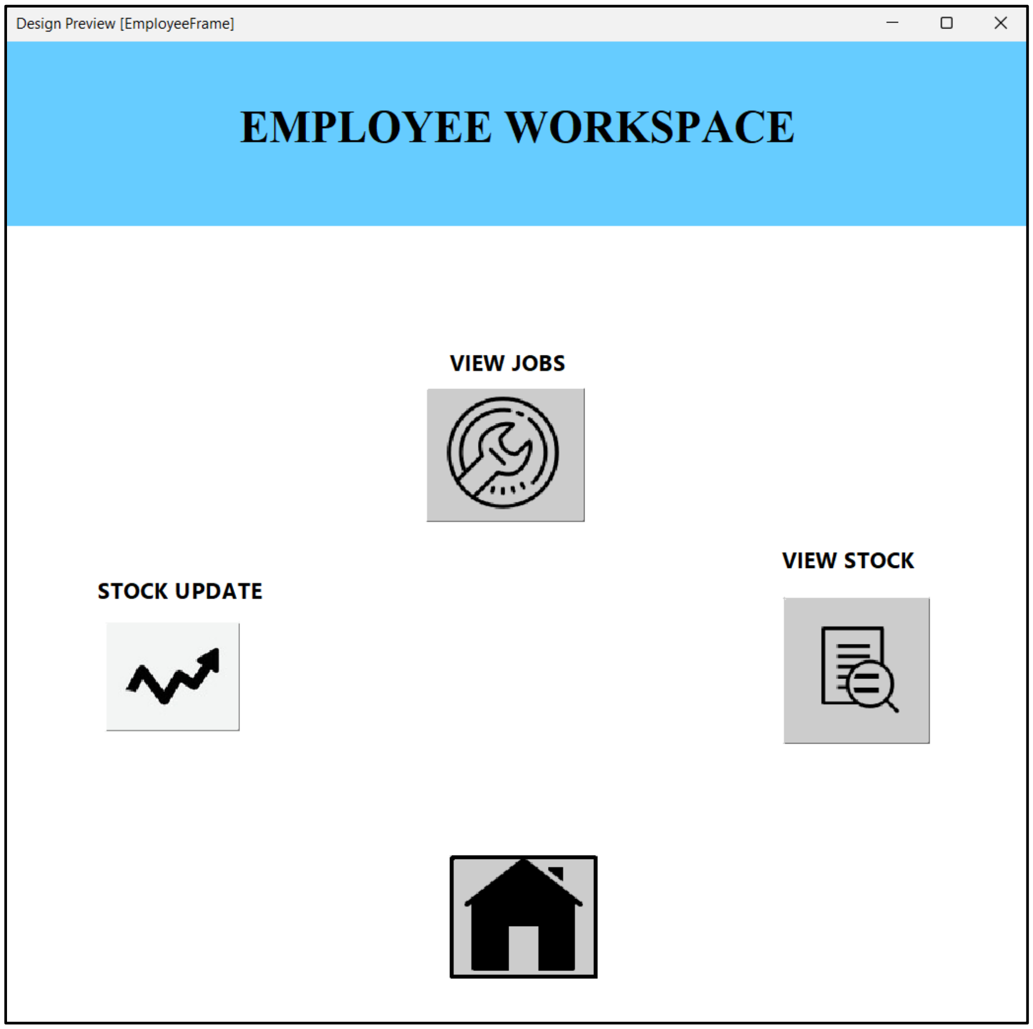
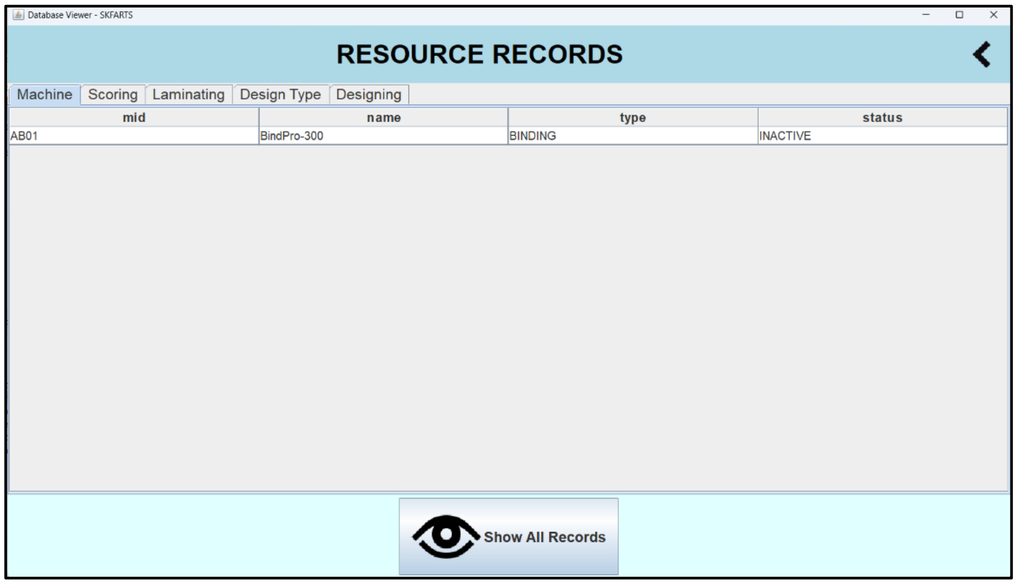


FIG 5.1.8 EMPLOYEE FRAME

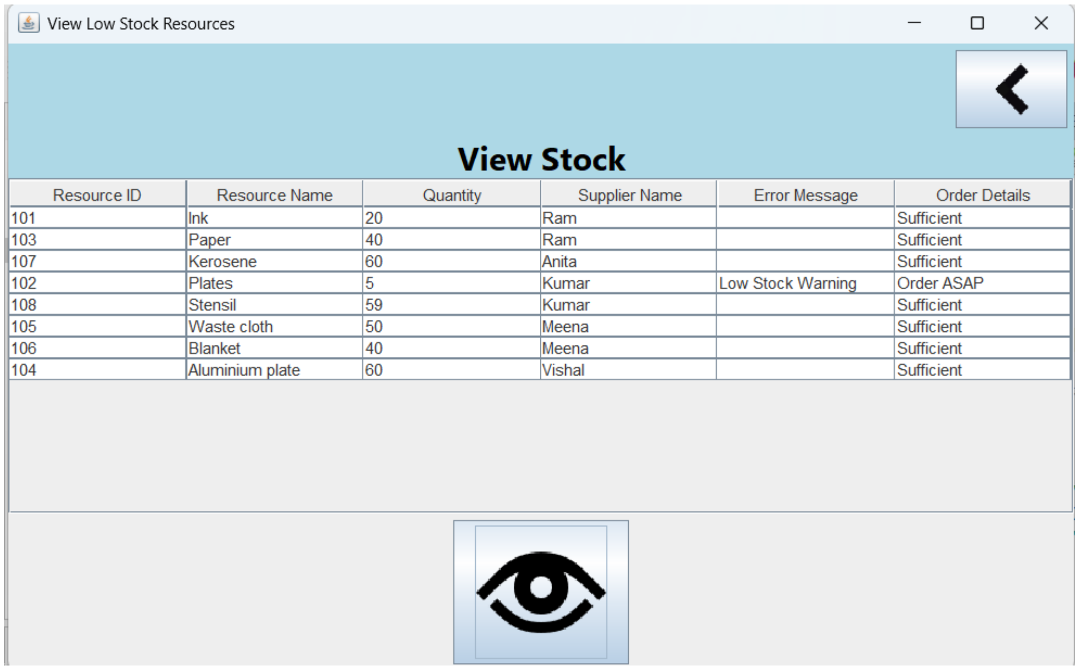
FIG 5.1.8 refers to the employee frame here it shows the privileges of the employees.



mid	name	type	status
AB01	BindPro-300	BINDING	INACTIVE

FIG 5.1.9 JOB RECORDS

FIG 5.1.9 refers to the job records where the employees can view the various jobs registered.



Resource ID	Resource Name	Quantity	Supplier Name	Error Message	Order Details
101	Ink	20	Ram		Sufficient
103	Paper	40	Ram		Sufficient
107	Kerosene	60	Anita		Sufficient
102	Plates	5	Kumar	Low Stock Warning	Order ASAP
108	Stensil	59	Kumar		Sufficient
105	Waste cloth	50	Meena		Sufficient
106	Blanket	40	Meena		Sufficient
104	Aluminium plate	60	Vishal		Sufficient

FIG 5.1.10 VIEW STOCK

FIG 5.1.10 refers to the view stock frame where the stock can be viewed .

CHAPTER 6

CONCLUSION

The Java Swing application developed for the printing press industry effectively addresses the challenges outlined in the problem statement. By providing a comprehensive solution for resource management, the application enhances operational efficiency, improves data accuracy, and supports the organization's growth.

The various classes and methods work together to create a seamless user experience, allowing employees and managers to manage resources, track stock, and make informed decisions. The integration of a robust database structure and user-friendly interface ensures that the application meets the needs of the organization while providing a solid foundation for future enhancements

.In summary, this application not only solves the immediate problems faced by the printing press industry but also lays the groundwork for ongoing improvements in resource management and operational efficiency. By leveraging technology effectively, the organization can better serve its customers and adapt to the ever-changing demands of the market.

CHAPTER 7

REFERENCES

Websites:

1.https://www.javatpoint.com/jdbc-mysql-database-connection-in-java

2. https://netbeans.apache.org/kb/index.html

3. https://dev.mysql.com/doc/connector-j/en/

4. https://www.javatpoint.com/example-of-jdbc-with-oracle-database

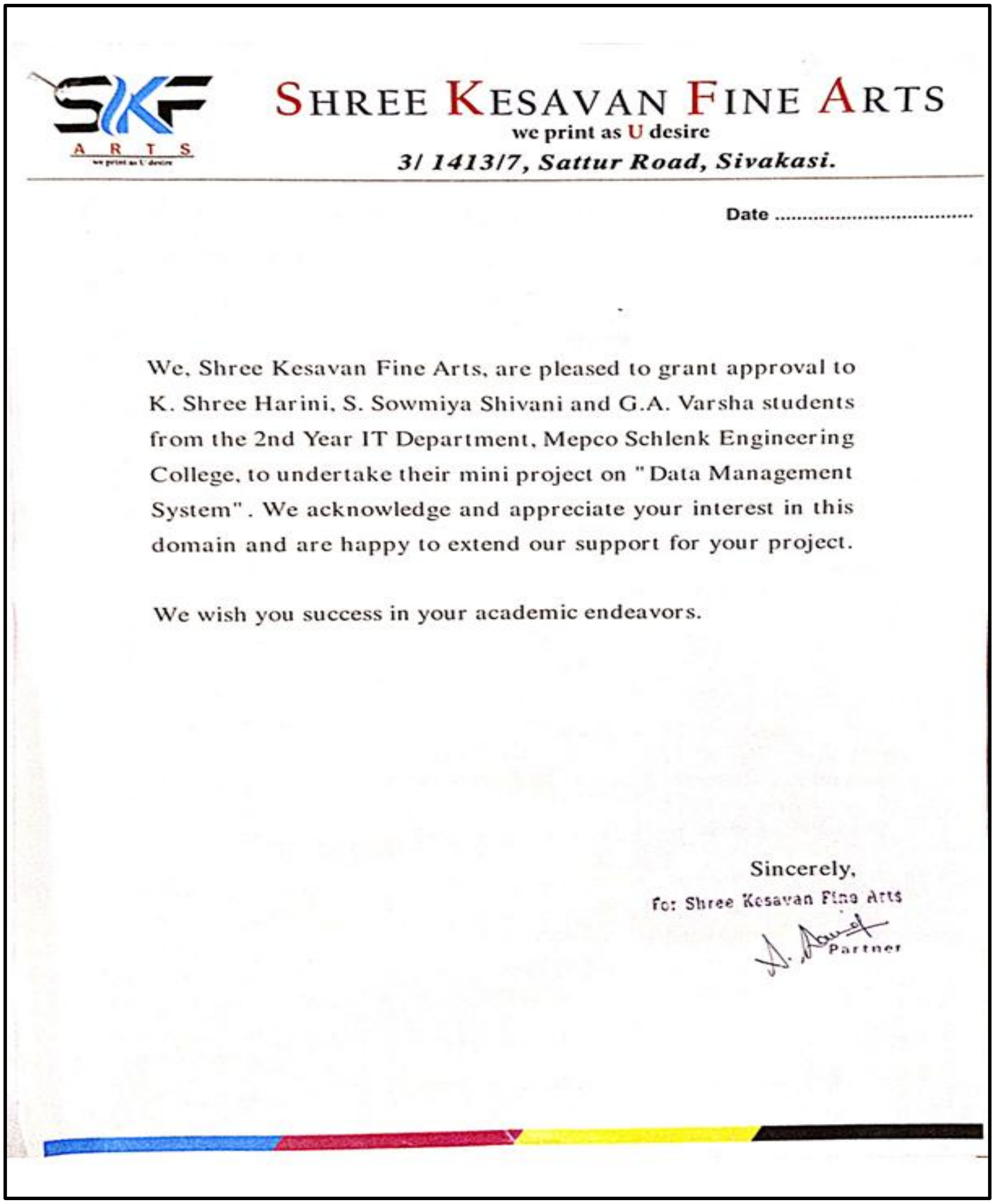
Books:

1. Java: The Complete Reference (11th Edition) – *Herbert Schildt*
2. Beginning Java Programming: The Object-Oriented Approach – *Bart Baesens et al.*
3. Java Programming for Beginners – *Mark Lassoff*
4. Database System Concepts by Silberschatz, Korth, and Sudarshan
5. An Introduction to Database Systems by C.J. Date

PDF And Short Notes:

1. An Introduction to GUI Programming with Java Swing
2. Oracle JDBC Developer's Guide
3. Introduction to Programming Using Java (Swing Edition)
4. SQL Tutorial – TutorialsPoint
5. TutorialsPoint – DBMS Quick Guide

7.1. APPENDIX



SHREE KESAVAN FINE ARTS
we print as U desire
3/ 1413/7, Sattur Road, Sivakasi.

Date ....

We, Shree Kesavan Fine Arts, are pleased to grant approval to K. Shree Harini, S. Sowmiya Shivani and G.A. Varsha students from the 2nd Year IT Department, Mepco Schlenk Engineering College, to undertake their mini project on "Data Management System". We acknowledge and appreciate your interest in this domain and are happy to extend our support for your project.

We wish you success in your academic endeavors.

Sincerely,
For Shree Kesavan Fine Arts
[illegible]
Partner