

Internally Funded Student Projects Management System

ICS1312---Java Programming Lab ---M.Tech-Int---2024

A PROJECT REPORT

Submitted By

Pandiarajan D (3122237001035)

Rushabh S (3122237001044)



**5-Year Integrated M.Tech
Computer Science and Engineering**

**Sri Sivasubramaniya Nadar College of Engineering
(An Autonomous Institution, Affiliated to Anna University)
Kalavakkam – 603110**

November 2024

TABLE OF CONTENTS

S.No	Section	Page Number
1	Problem Statement	3
2	Motivation	5
3	Scopes and Limitations	7
4	Design of Solution(Class Diagram)	8
5	Modules Split-up	9
6	Implementation	11
7	Output Screenshots	14
8	Object oriented features used	20
9	Inference and future extension	23

PROBLEM STATEMENT:

The project aims to develop a **Project Management System** that provides tailored functionalities for **Admin**, **Student**, and **Faculty** users. Each user category will have different capabilities that align with their role in managing and participating in projects. The goal is to streamline project management, submission, tracking, and status updates.

Inputs

- Admin:
 - Login credentials for authentication.
 - Project details for displaying all projects.
 - Request data for viewing project requests.
 - Status change input for updating project request statuses.
- Student:
 - Login credentials for authentication.
 - Registration data including personal and project-related information for new registrations.
 - Digital ID for viewing project details.
 - Task for a student.
 - Task progress updates (progress percentage or notes).
 - Student ID for viewing tasks and checking status.
- Faculty:
 - Login credentials for authentication.
 - Faculty ID or project group ID for viewing projects and tasks.

Outputs

- Admin:
 - Display of all projects and their current status.
 - List of requested projects.
 - Confirmation of status changes for project requests.
- Student:
 - Confirmation of successful registration.
 - Detailed display of project data for a given digital ID.
 - Confirmation of project submission.
 - Display of student details.
 - Updated task progress.
 - Task list and current status (accepted/rejected/completed).
- Faculty:
 - List of projects handled by the faculty.
 - Task list for assigned project groups.

Motivation for the Problem :

In any project management or academic environment, effectively managing and tracking project progress, user credentials, and task status is essential. This project tackles several real-world problems within the realm of academic institutions and collaborative work environments, focusing on students, faculty, and administrators. The motivations behind this project include:

1. Streamlined Project Tracking :

- **Current Problem:** Students often struggle with tracking their project progress, managing their tasks, and keeping up with deadlines. Faculty members face difficulties in overseeing multiple student projects, while administrators need to maintain a global view of all project activities.
- **Solution Motivation:** Creating a system that consolidates project information and task management into a single interface helps all stakeholders stay informed and efficient. By providing real-time access to project statuses and tasks, students, faculty, and administrators can work in a more synchronized and productive manner.

2. Centralized Data Management :

- **Current Problem:** Data fragmentation across multiple platforms and manual records leads to inefficiency, duplication, and difficulty in data retrieval.
- **Solution Motivation:** A unified system that stores and organizes data—such as student details, task progress, and project information—enhances data accessibility and reliability. This system minimizes errors associated with manual data handling and simplifies data management for academic institutions.

3. Enhanced User Experience :

- **Current Problem:** Students and faculty members often face challenges in accessing relevant project information, task details, and status updates. Existing systems may not be user-friendly or interactive enough, creating frustration and delays.

- **Solution Motivation:** A user-centric design with an intuitive interface makes it easier for students to check their project progress, submit work, and view feedback. Faculty members can more readily monitor student performance and provide assistance as needed. Administrators can efficiently manage and review project submissions and statuses without the need for complex procedures.

4. Accountability and Progress Transparency :

- **Current Problem:** Without a clear system in place, students may not have a concrete way to show their task progress or get feedback from their mentors. Faculty members also face the difficulty of documenting and reviewing student submissions and project developments.
- **Solution Motivation:** Providing tools to log and visualize task progress and project statuses promotes accountability and transparency. Students can track their progress, and faculty members can easily assess student contributions and offer timely feedback. Administrators can verify project milestones and ensure compliance with institutional standards.

5. Role-Based Access Control :

- **Current Problem:** Systems that do not differentiate between user roles can lead to data mishandling and security issues. Allowing unrestricted access to sensitive project details compromises data privacy and workflow efficiency.
- **Solution Motivation:** Implementing a role-based system ensures that students, faculty, and administrators only have access to relevant data and features. This enhances security, maintains data integrity, and helps each user focus on their respective responsibilities.

OVERALL GOAL :

The main goal is to create a comprehensive project management system that incorporates modern object-oriented design principles like inheritance and polymorphism to create an extendable, maintainable, and scalable platform. The system should serve as a digital bridge connecting students, faculty, and administrators to their respective project-related tasks, reducing administrative overhead and enhancing productivity and satisfaction.

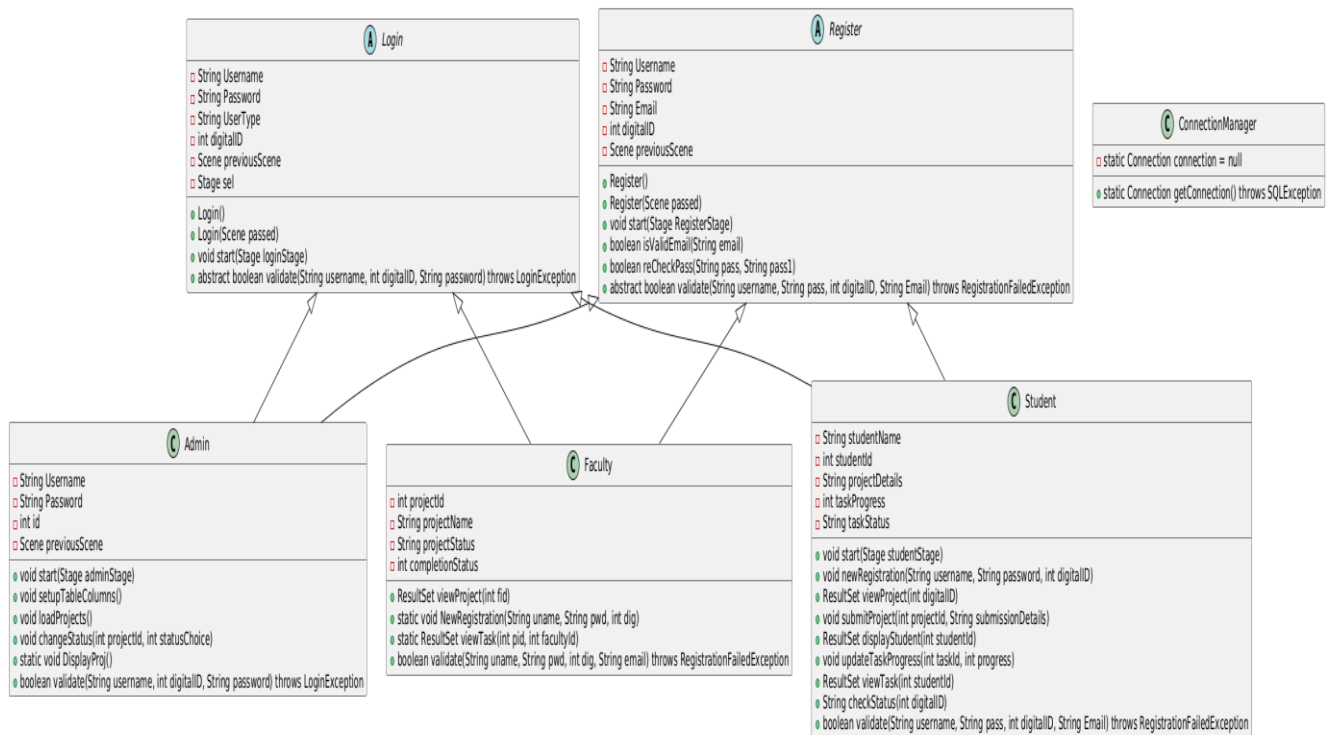
1. Scope

- The system is intended for educational institutions or project-based organizations to manage projects and facilitate communication between students, faculty, and admin.
- Ensures secure login for each user type to protect data integrity.
- Allows tracking of project progress and submission statuses.
- Helps faculty oversee and guide student projects efficiently.
- Facilitates admin control over project management and approval processes.

2. Limitations

- **Scalability:** The system may not perform optimally with a very large number of concurrent users without further enhancement
- **Data Integrity:** Relies on accurate input from users. Input errors can affect the accuracy of project records or task tracking.
- **User Interface:** Assumes a basic frontend without advanced design elements, potentially impacting user experience.
- **Features:** Does not include collaborative features such as real-time editing or in-app communication tools, which might limit interactivity and coordination.

CLASS DIAGRAM :



Inspect and suggest at least one design alternative.

INSPECTION OF CURRENT PROJECT :

The current design focuses on a role-based project management system with separate functionalities for **Admin**, **Student**, and **Faculty**. User interactions involve logging in, retrieving or updating project data, and managing tasks through direct database queries.

Design Limitations

- **Security Risks:** Direct database interactions may expose the system to SQL injection if user input isn't properly sanitized.
- **Tight Coupling:** Directly binding input handling and SQL logic can make code difficult to maintain or extend.
- **Scalability and Extensibility:** The current structure might face challenges as new features are added or user load increases.

NEW DESIGN ARCHITECTURE:

Three-Tier Architecture is a well-known design pattern that separates the system into three logical layers :

1. **Presentation Layer** (Frontend)
2. **Business Logic Layer** (Backend/Application Layer)
3. **Data Access Layer** (Database Interaction)

Working:

- **Presentation Layer:** Handles user interactions, displays information, and collects inputs.
- **Business Logic Layer:** Processes user inputs and applies business rules, ensuring that logic is decoupled from the frontend and database.
- **Data Access Layer:** Responsible for interacting with the database using parameterized queries or ORM (Object-Relational Mapping) tools to prevent SQL injection.

Benefits of This Alternative:

1. **Enhanced Security:** By isolating database interactions in the data access layer and using ORM or parameterized queries, SQL injection risks are minimized.
2. **Improved Maintainability:** Logical separation allows for more straightforward updates and testing of individual components.
3. **Extensibility:** Easier to add new features or integrate additional security measures.
4. **Scalability:** The architecture supports distributed and scalable solutions where each layer can be scaled independently as needed.

MODULES SPLIT UP :

Admin Module

- Purpose: Contains all functionalities related to the Admin role.
- Classes:
 - Admin: Handles admin-specific operations.
- Methods:
 - login(): Authenticates the admin user.
 - displayProj(): Displays all projects.
 - viewReq(): Displays all requested projects.
 - changeStatus(): Changes the status of requested projects.

Student Module

- Purpose: Contains functionalities specific to students.
- Classes:
 - Student: Handles student-related operations.
 - TaskDetails: Class to hold task-related information.
- Methods:
 - login(): Authenticates the student user.
 - newRegistration(): Allows new students to register.
 - viewProject(digitalId: int): Displays project details for a given digital ID.
 - submitProject(): Submits a project.
 - displayStudent(): Displays details of a specific student.
 - updateTaskProgress(): Updates the progress of assigned tasks.
 - viewTask(): Views tasks associated with the student.
 - checkStatus(): Checks the acceptance/rejection status of a project.

Faculty Module

- Purpose: Contains functionalities specific to faculty members.
- Classes:
 - Faculty: Handles faculty-specific operations.
- Methods:
 - login(): Authenticates the faculty user.
 - viewProject(): Displays projects assigned to the faculty.
 - viewTask(): Views tasks assigned to a specific project group.

Database Module

- Purpose: Handles all database connections and queries.
- Classes:
 - ConnectionManager: Manages database connections.
- Methods:
 - getConnection(): Establishes and returns a database connection.

IMPLEMENTATION SPECIFICS

Student :

1. submitProject Method

- **Purpose:** Submits a project and handles task assignments.
- **Technical Details:**
 - Uses JDBC for database connectivity.
 - SQL queries are constructed to check for existing projects and to insert/update data.
- **Data Structures:**
 - Uses ResultSet to handle query results.
 - Uses Scanner for user input.
- **Algorithms:**
 - Sequential checks to ensure a project isn't already selected.
 - Loop for selecting students or faculty.
- **Error Handling:**
 - Custom exceptions (ProjectAlreadySelectedException, TaskProgressException) are thrown for specific error scenarios.
- **Performance Considerations:**
 - The method ensures that database connections and statements are properly closed in the finally block.
- **User Interaction:** Prompts the user for project details and selections.

2. displayStudent Method

- **Purpose:** Fetches and returns a student's record based on provided credentials.
- **Technical Details:**
 - Uses JDBC to connect to the database and execute a query.
- **Data Structures:**
 - Returns a ResultSet containing the student record.
- **Algorithms:**
 - Checks if the record exists and retrieves it.
- **Error Handling:**
 - Throws a NoStudentFoundException if no records match the credentials.
- **User Interaction:** None, as it returns data rather than printing it.

3. updateTaskProgress Method

- **Purpose:** Updates the task progress for a student.
- **Technical Details:**
 - Uses JDBC to query and update student progress.
- **Data Structures:**
 - Uses ResultSet to retrieve current progress.
 - Uses basic data types (int, double) for calculations.
- **Algorithms:**
 - Calculates average progress across all students on the same project.
- **Error Handling:**
 - Throws a TaskProgressException for invalid progress values or if the student record is not found.
- **User Interaction:** Prompts user for new progress value.

4. viewTask Method

- **Purpose:** Displays a student's task details.
- **Technical Details:**
 - Connects to the database to fetch task information.
- **Data Structures:**
 - Uses ResultSet to hold the query results.
- **Algorithms:**
 - Retrieves project and task details and checks if progress is null.
- **Error Handling:**
 - Throws a ViewTaskException if no task is found for the student.
- **User Interaction:** None, as it returns task details.

5. Wrapper Class for Task Details

- **Purpose:** Encapsulates task-related information.
- **Technical Details:**
 - A simple POJO (Plain Old Java Object) class to hold properties like project name, task name, and task progress.
- **Data Structures:**
 - Uses standard Java types (String, Integer).
- **Error Handling:** Not applicable, as it's a data holder.
- **User Interaction:** None.

6. Custom Exceptions

- **Purpose:** Handles specific error scenarios.
- **Technical Details:**
 - Custom exception classes like ProjectAlreadySelectedException, NoStudentFoundException, and ViewTaskException extend Exception.
- **Error Handling:**
 - Provides meaningful messages for different failure scenarios, enhancing debugging and user feedback.

7. Database Interaction

- **Purpose:** Perform CRUD operations on the student and project tables.
- **Technical Details:**
 - Establishes a connection using ConnectionManager.
 - Uses Statement and ResultSet to execute SQL commands and fetch results.
- **Error Handling:**
 - SQLException handling is managed in the finally block to ensure resources are closed.
- **Performance Considerations:**
 - Prepared statements could be considered for improved performance and security (protection against SQL injection).

Faculty:

1. viewProject Method

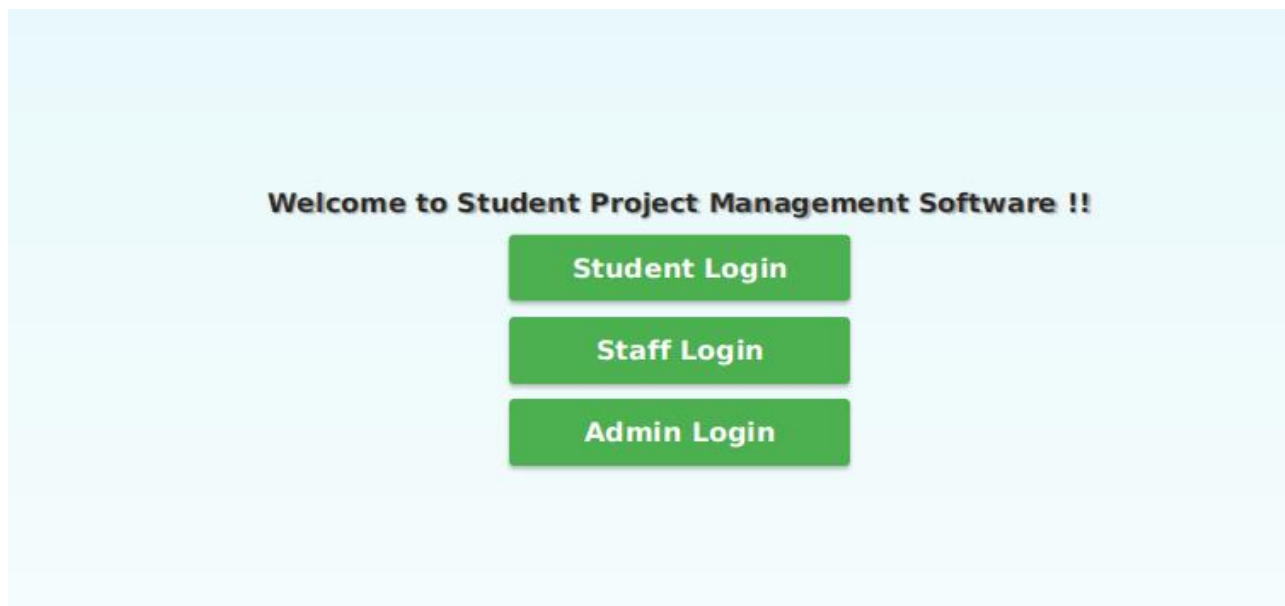
- **Purpose:** Allows faculty to view their assigned projects.
- **Technical Details:**
 - Connects to the database to retrieve projects associated with the faculty.
- **Data Structures:**
 - Uses ResultSet to hold query results.
- **Algorithms:**
 - Executes a SQL query to fetch projects linked to the faculty's ID.
- **Error Handling:**
 - Throws a custom exception (e.g., NoProjectsFoundException) if no projects are found.
- **User Interaction:** Displays project details.

2. viewTask Method

- **Purpose:** Displays the tasks assigned to the faculty for specific projects.
- **Technical Details:**
 - Retrieves task details based on faculty assignments.
- **Data Structures:**
 - Uses ResultSet to gather task data.
- **Algorithms:**
 - Executes a SQL query to fetch tasks associated with the faculty.
- **Error Handling:**
 - Throws a NoTasksFoundException if no tasks are available for the faculty.
- **User Interaction:** Presents task details in a readable format.

OUTPUT SCREENSHOTS :

HOME PAGE



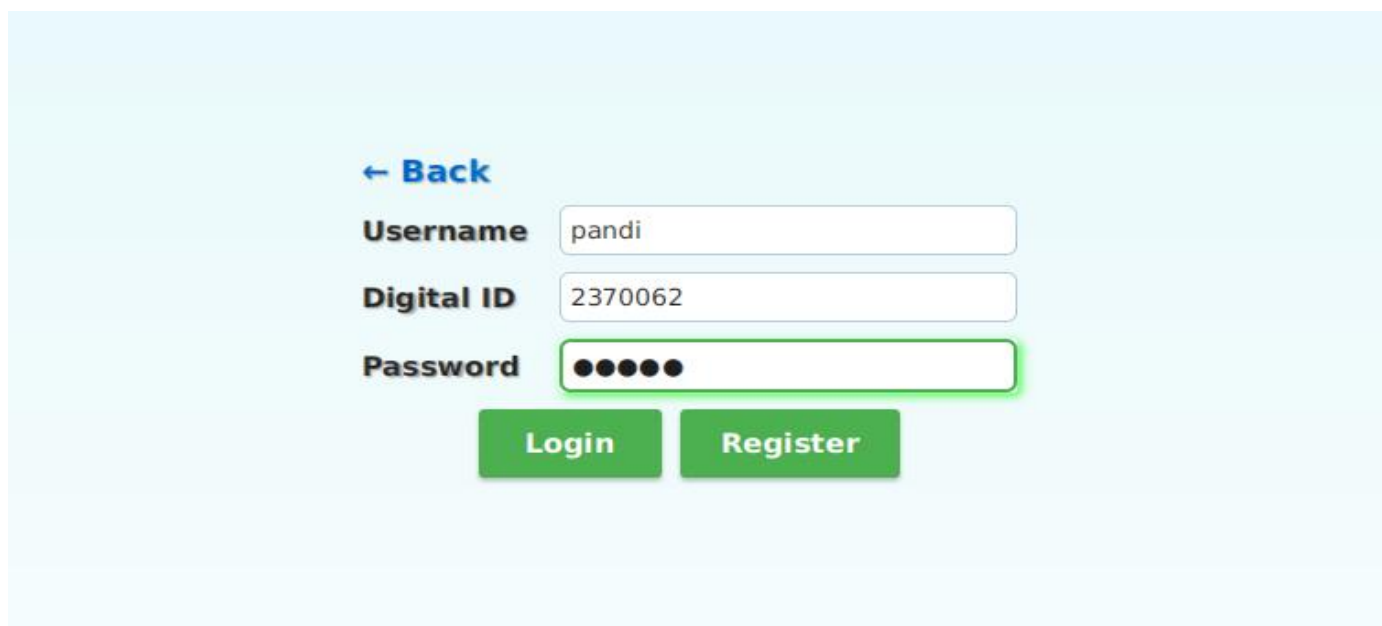
>Welcome to Student Project Management Software !!

Student Login

Staff Login

Admin Login

STUDENT :



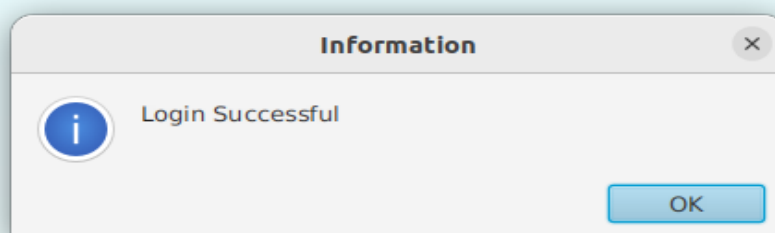
← Back

Username

Digital ID

Password

Login Register



[← Back](#)

Username

Digital ID

Password

Login

Register

Choose the Option

Display Project

View Task

Update Task

Project Status

Logout

Project and Faculty Details

Project Name : Sample1

Student Information :-

Student Name	Task Progress	Task Name	
pandi	10	Print	
xyz	0	Input	

Faculty Mentor Information :-

Faculty ID :

7654321

Faculty Name :

abc

[← Back](#)

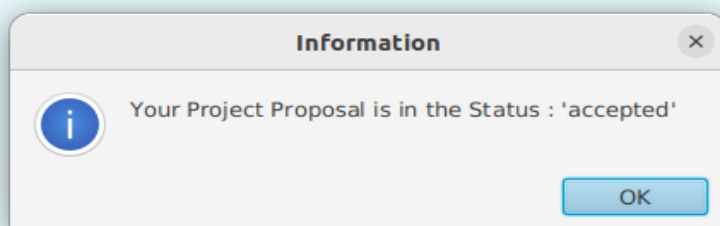
[← Back](#)

Your Current Status of your Task in the Project !

Project Name : Sample1

Task Name : Print

Progress : 10%



Display Project

View Task

Update Task

Project Status

Logout

FACULTY :[← Back](#)**Username**

abc

Digital ID

7654321

Password

●●●

Login**Register****Choose the Option****View All Projects****Check Progress****Logout****Project Details**[← Back](#)

Project ID : 1 Project Name : Sample1 Project Status : accepted Project Completion Status : 5

ADMIN :[← Back](#)**Username**

admin1

Digital ID

1234567

Password

●●●●●●●●●●

Login**Register**

Choose the Option

Display Request**View Project****Logout**

allprojects.txt

1	Project ID	Name	Email	Project Name
2	null	ghi	ghi@gmail.com	null
3	1	pandi	pandi@gmail.com	Sample1
4	1	xyz	xyz@gmail.com	Sample1
5	2	rus	rus@gmail.com	Sample2
6	3	def	def@gmail.com	Sample2

[← Back](#)

Project ID	Project Name	Current St...	Action
------------	--------------	---------------	--------

No content in table

OOPs CONCEPTS USED :-

ENCAPSULATION :

1. Encapsulated data in classes such as StudentInfo, TaskDetails, and ViewProject by using private fields and providing public methods (getters) to access them.

```
public class StudentInfo
{
    private String facultyName; 2 usages
    private String facultyId; 2 usages
    private ResultSet studentResultSet; 10 usages
```

```
public class TaskDetails
{
    private String projectName; 3 usages
    private String taskName; 3 usages
    private String taskProgress; 3 usages
```

INHERITANCE :

1. The base abstract class Login is inherited by AdminLogin, FacultyLogin, StudentLogin.

Base class :

```
public abstract class Login extends Application { 3 inheritors

    public String Username, Password, UserType;
    public int digitalID; 4 usages
    private Scene previousScene; 4 usages
    private Stage sel; 12 usages
```

Child class :

```
public class AdminLogin extends Login
```

```
public class FacultyLogin extends Login
```

2. The base abstract class Register is inherited by 2 classes, FacultyRegister, StudentRegister.

Base class :

```
abstract public class Register extends Application 2 inheritors
{
    String Username, Password, Email; 6 usages
    int digitalID; 7 usages
    Scene previousScene; 4 usages
```

Child class :

```
public class FacultyRegister extends Register {
```

```
public class StudentRegister extends Register {
```

ABSTRACTION :

- **Abstract Classes:** The Login and Register classes are defined as abstract classes, which means they cannot be instantiated directly. They serve as a base for other classes to inherit from, providing a template for common functionality while allowing derived classes to implement specific behavior (e.g., AdminLogin and StudentRegister).
- **Abstract Methods:** The abstract methods validate and redirect in the Login and Register classes define a contract for derived classes. Each derived class must implement these methods, allowing for specific login and registration behaviors.

POLYMORPHISM :

- **Method Overriding:** The derived classes override the abstract methods validate and redirect. This allows different behaviors for different types of users (e.g., admin vs. student) while maintaining a consistent interface.

ERROR HANDLING :

- **Custom Exceptions:** You have created custom exception classes (e.g., LoginException, RegistrationFailedException) to handle specific error conditions. This promotes clean error handling and makes the code easier to understand and maintain.

COMPOSITION :

- **Using Other Classes:** These classes often instantiate or utilize other classes (e.g., Admin, Student, ConnectionManager, and AlertBox). This is a form of composition, where a class is made up of one or more objects from other classes, allowing for more complex behavior and interactions.

INFERENCE AND FUTURE EXTENSIONS :

This **project management system** serves as a comprehensive tool for managing projects, tasks, and users (students, faculty, and admins). The existing codebase already outlines the fundamental operations, and understanding how to extend and improve upon it is crucial for future-proofing the application.

Inference

1. User Roles and Responsibilities:

- The system distinctly separates user roles (Student, Faculty, Admin) with clear responsibilities:
 - **Students** can submit projects, update progress, and view tasks.
 - **Faculty** can view projects and tasks assigned to them.
 - **Admins** can manage projects, review requests, and alter statuses.

2. Data Management:

- The use of a database to store user data, project details, and task progress ensures data persistence and integrity.
- The system utilizes SQL queries effectively to interact with the database, though there are opportunities to enhance security and efficiency (e.g., using PreparedStatement).

3. Error Handling:

- Custom exceptions provide robust error handling, allowing the application to manage various failure scenarios gracefully.
- Implementing specific exception types for different functionalities enhances debugging and user experience.

4. User Interaction:

- The system primarily interacts with users through the console, making it straightforward for command-line operations but potentially limiting for user experience.

5. Scalability:

- The current design appears to handle basic operations well but may face challenges as the user base grows or if more features are added.

Future Extensions

1. User Interface Improvements:

- **Graphical User Interface (GUI):** Transitioning from a console-based application to a GUI using JavaFX could enhance user experience, making the system more accessible and visually appealing.
- **Web Interface:** Consider developing a web-based front end, allowing users to access the system remotely via browsers.

2. Advanced Reporting:

- Implement reporting functionalities that provide insights into project statuses, team performances, and task completion rates. This could include generating reports in PDF or Excel formats.

3. Notification System:

- Develop a notification system to alert users about important updates, such as project approvals, deadline reminders, or task updates.

4. Real-Time Collaboration:

- We can integrate features for real-time collaboration among students and faculty, potentially using websockets for immediate updates.

5. Mobile Application Development:

- As mobile usage increases, developing a mobile application could make the system more accessible for users on the go.

6. Feedback and Continuous Improvement:

- Implement a feedback mechanism to gather user input on system performance and desired features, allowing for continuous enhancement based on real user needs.

7. Budget:

- Allocated budget for projects can be added so that students can limit their purchases.