**Software Requirements Specification**

**for**

**Student Information System**

**Version 1.0 approved**

**Prepared by Shubham Bankar**

**192050024**

**VJTI, Matunga**

**29/11/2019**

Table of Contents [ii](#__RefHeading___Toc441230970)

Revision History [ii](#__RefHeading___Toc441230971)

1. Introduction [1](#__RefHeading___Toc441230972)

1.1 Purpose [1](#__RefHeading___Toc441230973)

1.2 Document Conventions [1](#__RefHeading___Toc441230974)

1.3 Intended Audience and Reading Suggestions [1](#__RefHeading___Toc441230975)

1.4 Product Scope [1](#__RefHeading___Toc441230976)

1.5 References [1](#__RefHeading___Toc441230977)

2. Overall Description [2](#__RefHeading___Toc441230978)

2.1 Product Perspective [2](#__RefHeading___Toc441230979)

2.2 Product Functions [2](#__RefHeading___Toc441230980)

2.3 User Classes and Characteristics [2](#__RefHeading___Toc441230981)

2.4 Operating Environment [2](#__RefHeading___Toc441230982)

2.5 Design and Implementation Constraints [2](#__RefHeading___Toc441230983)

2.6 User Documentation [2](#__RefHeading___Toc441230984)

2.7 Assumptions and Dependencies [3](#__RefHeading___Toc441230985)

3. External Interface Requirements [3](#__RefHeading___Toc441230986)

3.1 User Interfaces [3](#__RefHeading___Toc441230987)

3.2 Hardware Interfaces [3](#__RefHeading___Toc441230988)

3.3 Software Interfaces [3](#__RefHeading___Toc441230989)

3.4 Communications Interfaces [3](#__RefHeading___Toc441230990)

4. System Features [4](#__RefHeading___Toc441230991)

4.1 System Feature 1 [4](#__RefHeading___Toc441230992)

4.2 System Feature 2 (and so on) [4](#__RefHeading___Toc441230993)

5. Other Nonfunctional Requirements [4](#__RefHeading___Toc441230994)

5.1 Performance Requirements [4](#__RefHeading___Toc441230995)

5.2 Safety Requirements [5](#__RefHeading___Toc441230996)

5.3 Security Requirements [5](#__RefHeading___Toc441230997)

5.4 Software Quality Attributes [5](#__RefHeading___Toc441230998)

5.5 Business Rules [5](#__RefHeading___Toc441230999)

6. Other Requirements [5](#__RefHeading___Toc441231000)

Appendix A: Glossary [5](#__RefHeading___Toc441231001)

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
|  |  |  |  |
|  |  |  |  |

# Introduction

## Purpose

The main objective of this document is to illustrate the requirements of the project Student Information System. The document gives the detailed description of the both functional and non-functional requirements for this project. SIS is mainly designed for the use of students. The purpose of this project is to provide a user friendly environment for the students and view the information regarding marks, attendance details, placement details. This project also maintain easy circulation system using computers to provide efficient access to the end user.

## Document Conventions

Throughout the document, the font used is Droid Sans and the font sizes are such that the topic is easily understood and followed as in the case of any conventional document practice. IEEE Standard Software Requirement Specification is used.

**Title:**

Font Style: Droid Sans

Size: 32

**Heading:**

Font Style: Droid Sans

Size: 22

**Subheading:**

Font Style: Droid Sans

Size: 16

**Content:**

Font Style: Droid Sans

Size: 12

## Intended Audience and Reading Suggestions

This document is intended for readers that include:

* Project Managers
* Developers
* Testers
* Staff
* Researchers working in this domain.

This SRS is categorized into five sections, viz. Introduction, Overall Description, System Features, External Interface Requirements, Other Non-functional Requirements and Other Requirements. The document will give you all the details of the project that has been laid down after very careful analysis.

## Product Scope

This software system will be a java applicaton which can be accessed from anywhere within the college/university campus for any university/college wishing to manage their student details online. More specifically to design and develop a simple and intuitive system which shall cater the need of a suitable interface for students as well as for faculties, to get the up-to-date notifications from TPO regarding placements. It shall also carter the details about the student’s academics. The portal increases the efficiency of college record management because previously the college is dependent on paper records only. The time required is less when compared with the paper records. For example generating the eligibility list for students based on the companies requirements will consume more time because the list differs from company to company, by using SIMS it can be generated easily and reduces the man power to do this. Also all users are authenticated to avail the services of this portal.

**Benefits:**

* User friendly automated tool.
* Paper work minimized.
* Cost effective.
* More security through authentication.
* Data can be retrieved at any moment and from anywhere within the campus.

**About Software:**

* Easy insertion, updating and retrieval of data using GUI is achieved through this system.
* Data is being stored in multiple tables, helping to achieve redundancy which may be useful if the system crashes.
* Can be executed on any system within the university/college campus.
* Input validation can be performed.

## References

**Books**

* Software Requirements and Specifications: A Lexicon of Practice, Principles and Prejudices (ACM Press) by Michael Jackson
* Software Engineering: A Practitioner’s Approach Fifth Edition By Roger S. Pressman

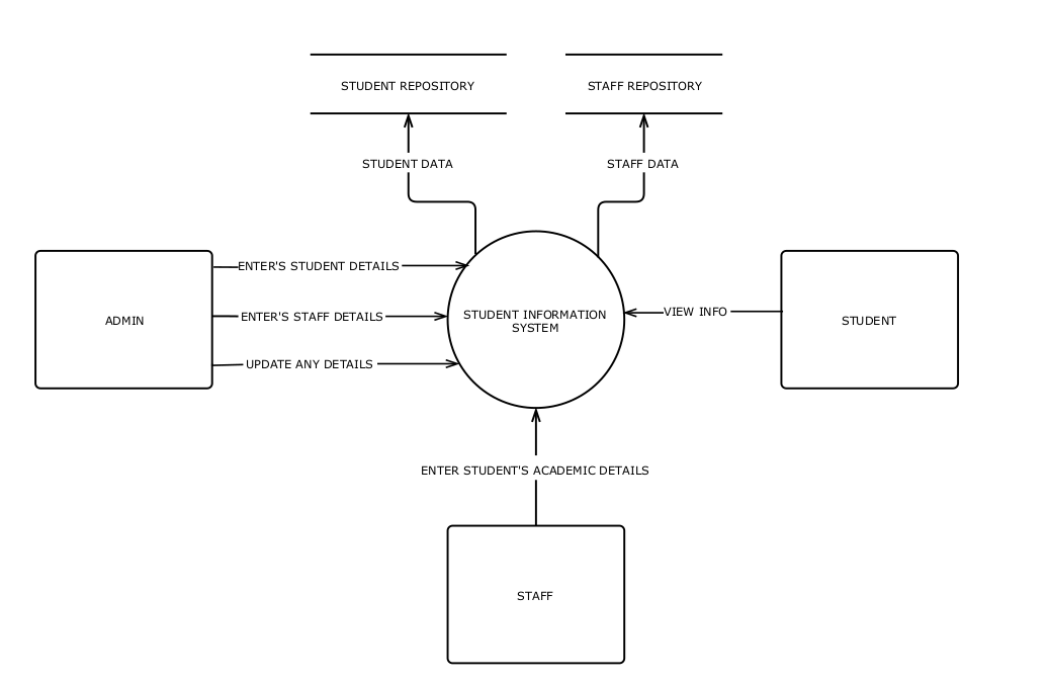
**Papers**

* Standard IEEE 830: Recommended practice for SRS is referred.

# Overall Description

## Product Perspective

SIS shall be developed using the client-server architecture. SIS is an upgrade to traditional SIS as it’s an Internet based application that can be accessed throughout the Net and can be accessed by anyone who has a net connection**.** It is an automatic system, where we will automate the viewing of all the info about student. After inserting the data to database, student can simply login into the portal using his own id and password and can see the details regarding his marks, attendance and placements. The cool thing about this is that the student need not go to professors and ask each and every time about details.

 DFD – Level 0 (Context Diagram)

## Product Functions

The SIS will allow access only to authorized users with specific roles (System administrator, Staff and Student). Depending upon the user’s role, he/she will be able to access only specific modules of the system.

A summary of major functions that the SIS will perform are :

* A login interface enabling authorized access to the system.
* Faculty will be able to add in the attendance details, and marks.
* Making a separate database for attendance and marks details.
* TPO will create a eligibility list of students based on their marks. This makes the work of TPO very easy as he doesn’t have to search for marks of students by searching many papers.
* Student upon login will be able to view their personal and academic details.
* Finally an administrator who will be adding and updating details of the students. Also administrator can drop any student data.

## User Classes and Characteristics

User of the system should be able to retrieve the information of his/her academic, personal and placement details from database easily. Also, some users act as an administrator who insert data into the database like marks and attendance details. The system will support three types of user privileges, Admin, Staff, Students.

**Administrator:**

The main role of administrator is to add or update the details of the students. This adding and updating entirely takes place on the Student database. Updation is entirely done using studentID to withdraw ambiguity. Administrator also stores in the Staff information which is helpful as it makes adding details like attendance and marks. Once new student or staff data is being added into database, admin provides with the username and password for the same. Finally, admin can also drop any student data, also taking aid of studentID to remove nay ambiguity.

**Faculty:**

The main role a faculty plays is that he/she is the one who collects all the personal and academic records of the student. Like he sends in the google sheet through which he collects the data of the students. Also he/she enters the marks and attendance details of the students. Faculty database contains all the details about the year, branch and respective subject he/she teaches. Thus the system automatically fetches all the information once the faculty logs in. All faculty needs to do is enter studentID and the respective count of attendance or marks

**TPO:**

The main role of TPO is to create an eligibility list. He checks the marks details of the students through mark database and shortlists the eligible students who can sit for the process of company placements. The eligible students are stored into another database which includes his name and studenetID and students have access to this database and can view if they are eligible for any company or not. Shortlisting is done automatically saving a lot of time.

**Student:**

The most important role is that of the student. He/She is the one who views the details, checks about his marks or attendance, if any queries still persists than informs the respective faculty, and also checks for placement details. He sends the info to the faculty upon request.

## Operating Environment

The product will be operating in windows environment or Linux environment. The SIS is a JAVA based application and shall operate in all computers supporting Java. For database purpose we will take MS Access or MySQL. Anyone with latest Java (8 or higher) can access this application.

The hardware configuration include Hard Disk: 40 GB, Monitor: 15” Color monitor, Keyboard: 122 keys. The basic input devices required are keyboard, mouse and output devices are monitor etc.

## Design and Implementation Constraints

The design constraint is that SIS is able to handle only a maximum of 2000 users, 50 faculty members, and 1 admin concurrently at a time. Also faculty cannot perform more than two operations at a time like adding marks and attendance simultaneously for the same student. Also student can run this application provided he uses his internet based login id and password; hence if the net is slow, he might not be able to access the system.

Users must use the latest operating system win7 and above as well as latest JAVA (8 or higher) to access the content of the application safely without any failure to gain access. SIS uses MySQL as its primary database and Java as its primary language and hence many external libraries like JDBC to connect mysql with java is required. Adding to it is the fact that t the system is implemented using JavaFX, and thus, JavaFX libraries also needs to be added.

## User Documentation

The manual to use the SIS will be available on the college or university website. Also the the application itself is easy to use since it is very user friendly. The steps for using SIS will be given in the manual itself.

## Assumptions and Dependencies

The assumptions are:

* The code should be error free
* The system should be user-friendly so that it is easy to use for the users
* The information of all users, transactions, components collected must be stored in a database that is accessible by the administrator.
* The system should have more storage capacity and provide fast access to the database.
* Users may access from any computer within the university premises that has Internet browsing capabilities and an Internet connection, since student can access the application only if he log in using his internet id and password, since SIS is uploaded on server.
* Users must have their correct userID and passwords to enter into their online accounts and do actions.
* Staff should also use proper userID and password in order to enter details regarding students. Assumption is that while entering student details like marks and attendance, studentID must be correctly specified.
* Since JavaFX and MySQL is used, all necessary libraries must be present on the server in order to run the application without any type of error.

The dependencies are:-

* The specific hardware and software due to which the product will be run.
* On the basis of listing requirements and specification the project will be developed and run.
* The end users, staff and admin should have proper understanding of the system.
* The information of all the users must be stored in a database that is accessible by the administrator.
* Any update regarding the student is to be recorded to the database and the data entered should be correct.
* Since TPO, Faculty uses the student data to add details, studentID will be the dependent identity.

# External Interface Requirements

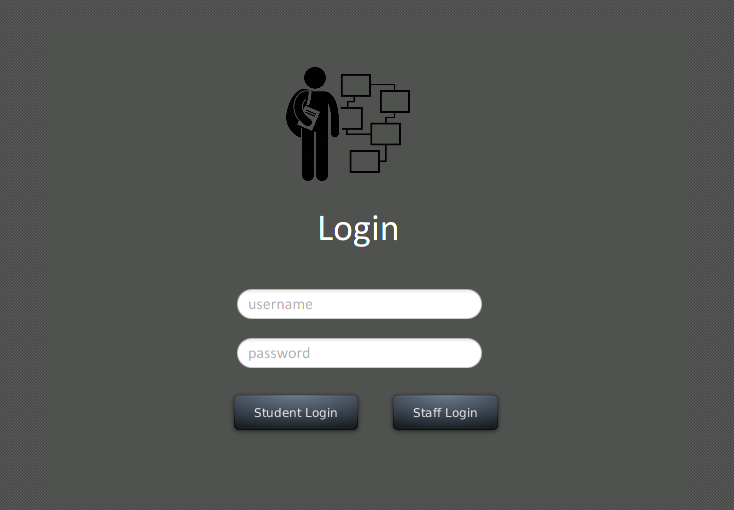
## User Interfaces

The software provides good graphical interface for the user, staff and the administrator and hence they can operate on the system, performing the required tasks.

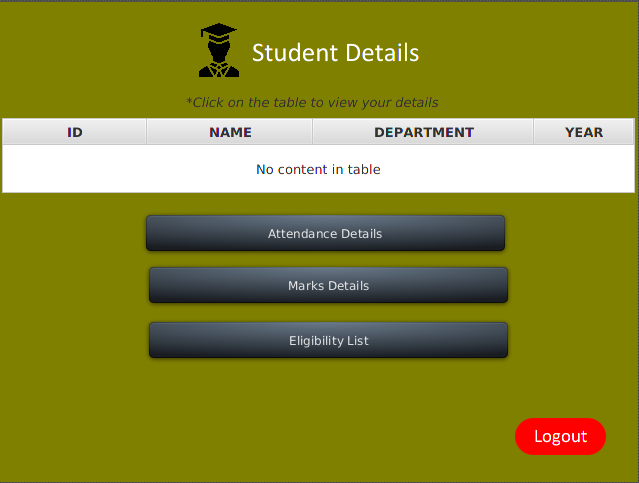
* User-interface must be simple, secure and user friendly.
* It must be simplified and self-explanatory.
* There must be robust error message system if user commits some error.
* It should visualize and explain the basic motivation behind this application.
* The system has a standardized user interface for all types of user.

Following are some of the snapshots of the user interfaces used in the system.

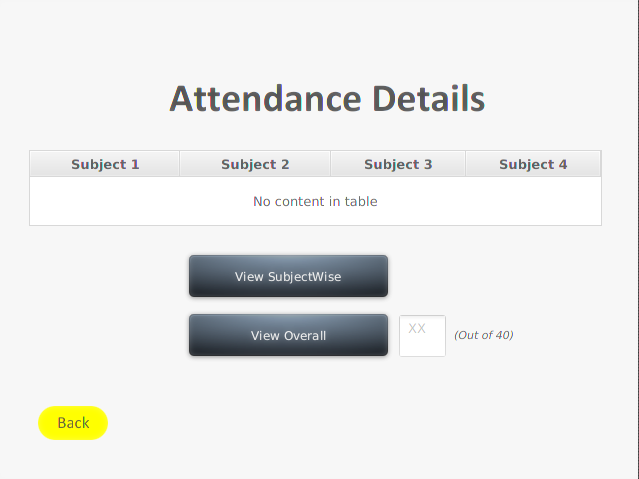
**Common Login Page for Student and Staff**



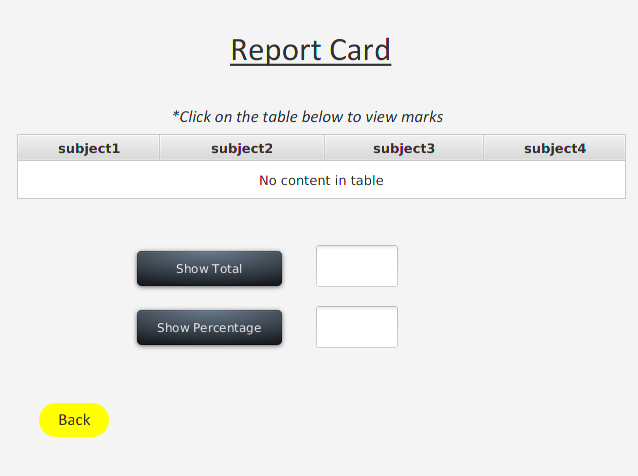
**Student Details Page**



**Attendance Details Page**

****

**Marks Details Page**

****

## Hardware Interfaces

Processor: Pentium(R) Dual-core CPU

Hard Disk: 40GB

RAM: 4GB or more

## Software Interfaces

This software package is developed using Java as front end; it uses JavaFX which is a MVC framework used by Java. Microsoft MS Access or MySQL can be used as the back end,primary Database used in the application is MySQL to store the databases.

Operating System: Windows 8/10, Any LINUX

Language: JavaFX (front end)

Language:Java(back end)

Database: MySQL (back end)

## Communications Interfaces

The communication interface for this software is possible using various internet protocol that we use to transfer the data and information online.

* FTP (File transfer protocol)
* HTTP (Hypertext transfer protocol)

# System Features

|  |  |
| --- | --- |
| Type of User | Function |
| 1. Administrator | * Login * Add details of students * Add details of Staff * Drop details of student |
| 1. Students | * Login * View personal details * View academic results * View eligibility for placements |
| 1. Faculty | * Login * Updates marks * Updates attendance * View Student details |
| 1. TPO | * Checks marks * Creates eligibility list |

## **View Attendance**

4.1.1 Description and Priority

After signing in, student is automatically navigated to student details page. Here upon clicking the view attendance details, student is again navigated to new page wherein he can view subject-wise attendance or total attendance.

4.1.2 Stimulus/Response Sequences

Students on clicking view attendance button, a backend java program runs which opens another page, wherein students can either see subject wise attendance or overall total attendance. On clicking subwise attendance, a query is fired at backend which retrieves the results from the attendance repository.

4.1.3 Functional Requirements

The only functional requirement needed to carry out the operation is that the student id should be correct. If incorrect username or password is entered, a dialog appears telling that the username or password is incorrect.

## **View Marks**

4.2.1 Description and Priority

After signing in, student is automatically navigated to student details page. Here upon clicking the view attendance details, student is again navigated to new page wherein he can view subject-wise attendance or total attendance.

4.2.2 Stimulus/Response Sequences

Students on clicking view marks button, a back end java program runs which opens another page, wherein students can either see subject wise marks, overall total marks, and percentage . On clicking sub wise marks, a query is fired at back end which retrieves the results from the attendance repository. Similarly for total and percentage, also a query is fired on clicking the button.

4.2.3 Functional Requirements

The only functional requirement needed to carry out the operation is that the student id should be correct. If incorrect username or password is entered, a dialog appears telling that the username or password is incorrect.

# Other Nonfunctional Requirements

## Performance Requirements

SIS that we are going to develop can be used as the main system within the many colleges or universities of different states where the it will be implemented to view student’s information in a simple and efficient way. Therefore, it is expected that the database would perform functionally all the requirements properly and maintain all the transactions details.

## Safety Requirements

Since the data we are going to access is stored on the database, it may crash due to hardware or software failures. Hence it is must to keep the backup of the data so that it is not lost.

## Security Requirements

* System will use secured database
* Normal users can just read information but they cannot edit or modify anything
* Proper user authentication should be provided
* There should be separate accounts for admin and end users such that no user can access the database and only admin has the rights to update the database.

## Software Quality Attributes

SQA attributes define the quality of the product as per the fulfillment of demands of the end user. The users are assumed to have basic knowledge of the computers and internet browsing. The administrators of the system should have more knowledge of the internals of the system and is able to rectify the small problems that may arise due to disk crashes, power failures and other catastrophes to maintain the system. The proper user interface, user manual, online help and the guide to tell how portal works must be sufficient to educate the users on how to use the system without any problems.

The admin provides certain facilities to the users in the form of:-

* Backup and Recovery
* Data migration i.e. whenever user registers for the first time then the data is stored in the server
* Maintaining files i.e. File Organization
* The server must be maintained regularly and it has to be updated from time to time

# Other Requirements

**6.1 Error handling**

If any of the validations or if authentication fails then appropriate error messages will be prompted to the user for doing the needful.

**6.2 Reliability**

Data validation and verification needs to be done at every stage of activity.

* Validating user input

Appendix A: Glossary

* SIS: Student Information System
* CPU: Central processing unit
* RAM: Random access memory

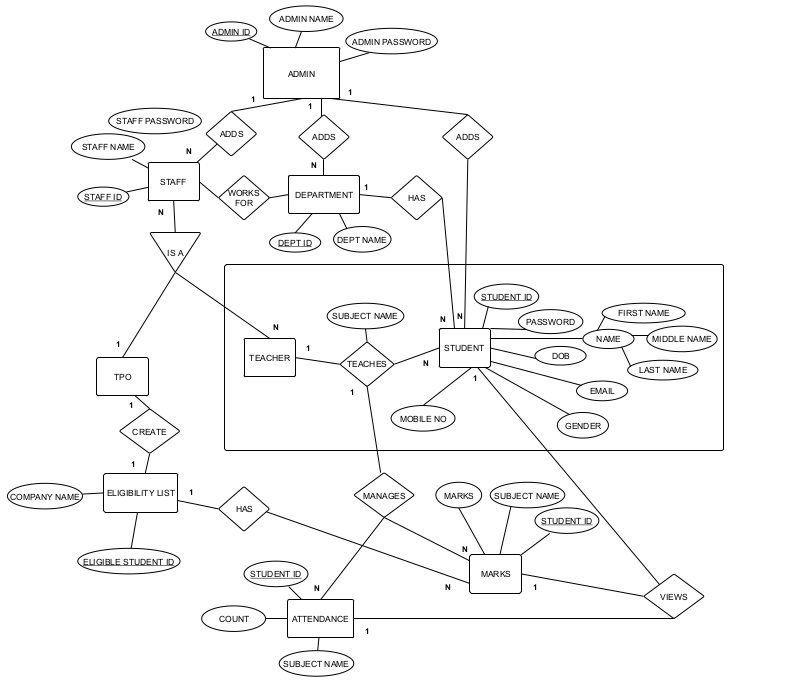
**Structured Diagrams**

1. **E-R Diagram**

The ER or (Entity Relational Model) is a high-level conceptual data model diagram. Entity-Relation model is based on the notion of real-world entities and the relationship between them. ER modeling helps you to analyze data requirements systematically to produce a well-designed database. So, it is considered a best practice to complete ER modeling before implementing your database. This model is based on three basic concepts:

* Entities
* Attributes
* Relationships

The E-R Diagram for the system:



**Entities and their Attributes**

* **STUDENT:**

*Attributes:* STUDENTID, PASSWORD, NAME, MOBILE, DOB, GENDER, EMAIL, DEPARTMENT, YEAR

* **STAFF:**

*Attributes:* STAFFID, STAFF\_PASSWORD, STAFF\_NAME

* **MARKS:**

*Attributes:* SUBJECT\_NAME, MARKS, STUDENTID

* **ATTENDANCE:**

*Attributes:* STUDENTID, SUBJECT\_NAME, COUNT

* **ELIGIBILITY LIST:**

*Attributes:* STUDENTID, COMPANY\_NAME

* **TEACHES (RELATIONSHIP):**

*Attributes:* TEACHES

* **DEPARTMENT:**

*Attributes:* DEPT\_ID, DEPT\_NAME

* **ADMIN:**

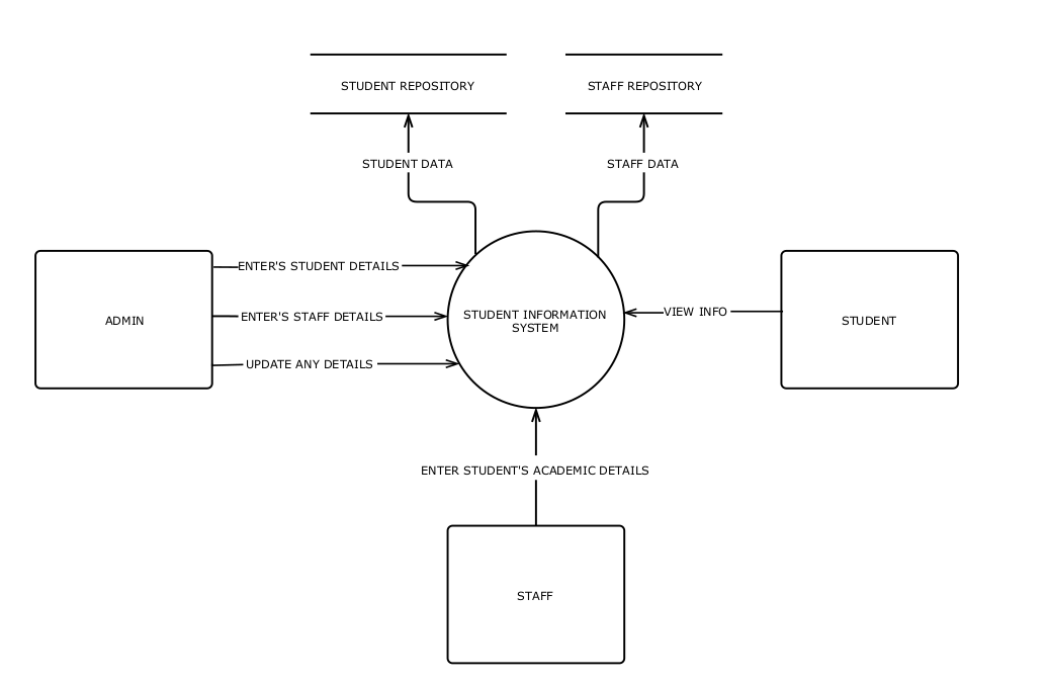
*Attributes:* ADMIN\_ID, ADMIN\_NAME, ADMIN\_PASSWORD

1. **Data-Flow Diagram**

Data flow diagrams describes the processes that are involved in a system to transfer data from the input to the file storage and result generation. DFD are used to graphically represent the flow of data in a business information system. There are four basic symbols that are used to represent a data-flow diagram:

1. A process receives input data and produces output with a different content or form.
2. A data-flow for data to move from one part of the information system to another.
3. A data store or data repository is used in a data-flow diagram to represent a situation when the system must retain data.
4. An external entity is a person, department, outside organization that provides data to the system or receives outputs from the system.

**2.1 DFD – Level 0 (Context Diagram)**



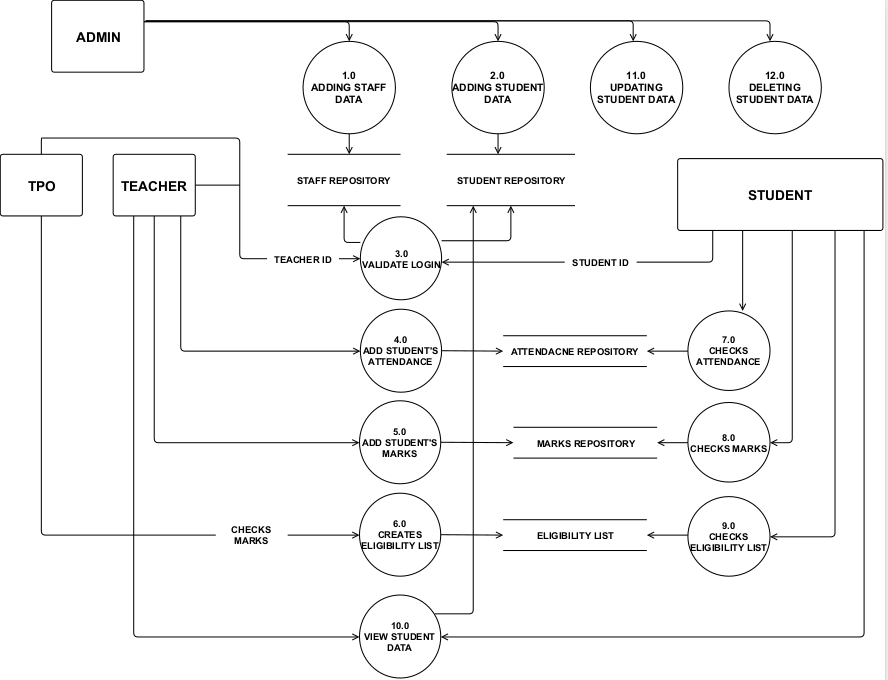
The entities are Administrator, Student, and Staff.

Administrator enters all the students and staff details through an user interface and then these details will be stored in student and staff repositories respectively.

Staff can log in to the application and then add student details like marks and attendance. This again is stored in another database.

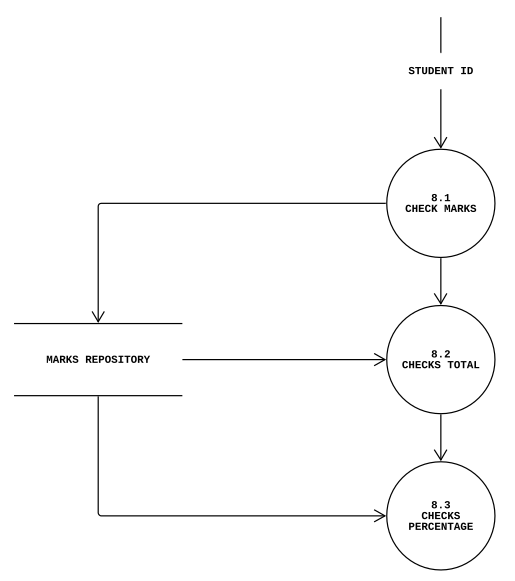
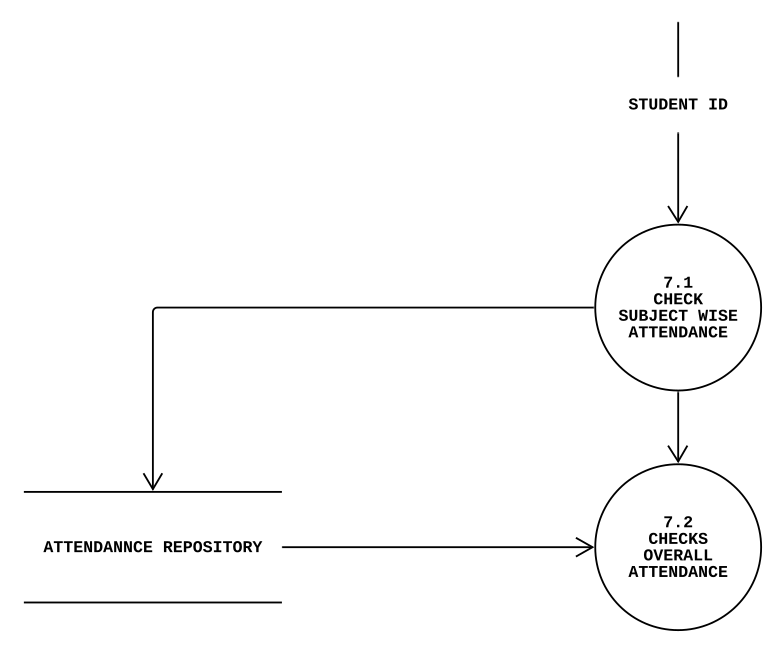
Students can simply log in to the application and view the attendance and marks details.

**2.1 DFD – Level 1**

 Level 1 DFD has following processes:

1. **Adding Staff data :** This process is used by administrator to add all the staff details which includes staff\_name, staff\_id, designation as the main fields into the Staff Details database.
2. **Adding Student data :** This process is used by administrator to add all the student details which includes student\_name, student\_id as the main fields into the Student Details database.
3. **Validate login:** This process is used when either of the student or the staff is trying to login using their respective user-id and password.
4. **Add Attendance details :** This process is used by the faculty to add the attendance of the student. The attendance is added based on the subject the staff teaches and the student id.
5. **Add Marks Details :** This process is used by the faculty to add the marks of the student. The attendance is added based on the subject the staff teaches and the student id.
6. **Creates Eligibility List :** This process is used by the TPO to select those students who are eligible for placements based on the marks of the student.
7. **Checks Attendance:** This process is used by the student to check upon the attendance, which is filled up by the faulty. The attendance is retrieved from attendance repository.
8. **Checks Marks :** This process is used by the student to check upon the marks, which is filled up by the faulty. The marks is retrieved from marks repository.
9. **Check Eligibility List:** This process is used by the student to view whether he/she is eligible for the placement drive or not.
10. **View Student data:** This process can either be used by student or staff to view sudent details like student name, id, even academic details like marks.
11. **Update Student Data:** This process is used by the administrator to update any details of the student. Administrator first verifies the details using student-id and then proceed for updation.
12. **Delete Student data :** This process is used by the administrator to delete the details of the student. Deletion will delete all the records of the student

**2.2 DFD – Level 2**



Process 7 is further divided into two more processes:

* 7.1 student can check attendance subject wise or
* 7.2 can view the total attendance directly.

Similarly, Process 8 is further divided into three more processes,

* 8.1 student can check marks either subject wise or
* 8.2 can view total marks or
* 8.3 can directly check for percentage.

**Object-oriented Diagrams**

1. **Use case diagrams**

A use case diagram is a dynamic or behavior diagram in UML. Use case diagrams model the functionality of a system using actors and use cases. Use cases are a set of actions, services, and functions that the system needs to perform. In this context, a "system" is something being developed or operated, such as a web site. The "actors" are people or entities operating under defined roles within the system. Use case diagrams are valuable for visualizing the functional requirements of a system that will translate into design choices and development priorities

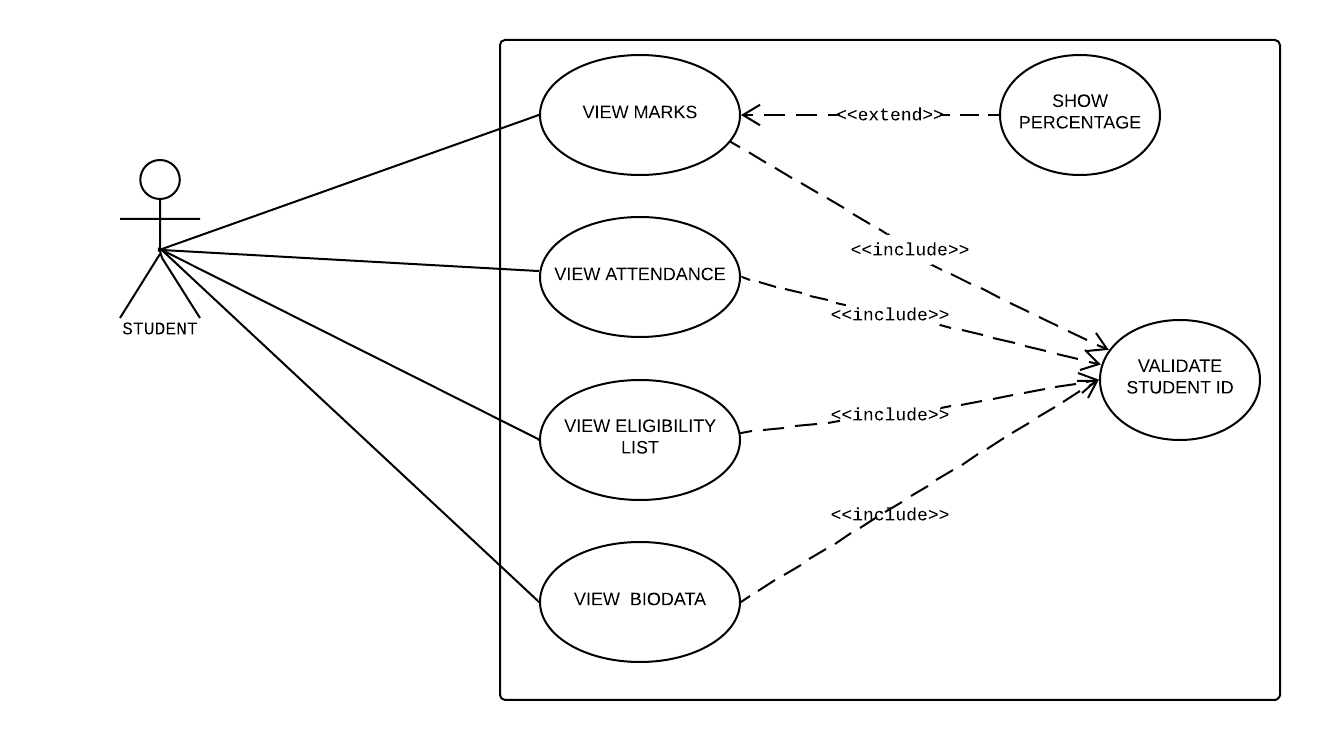
****

Fig : Student Use-Case

**a)** **Actors**

* Students

**b)** **Use Cases**

* View Marks
* View Attendance
* View Eligibility list
* View BioData

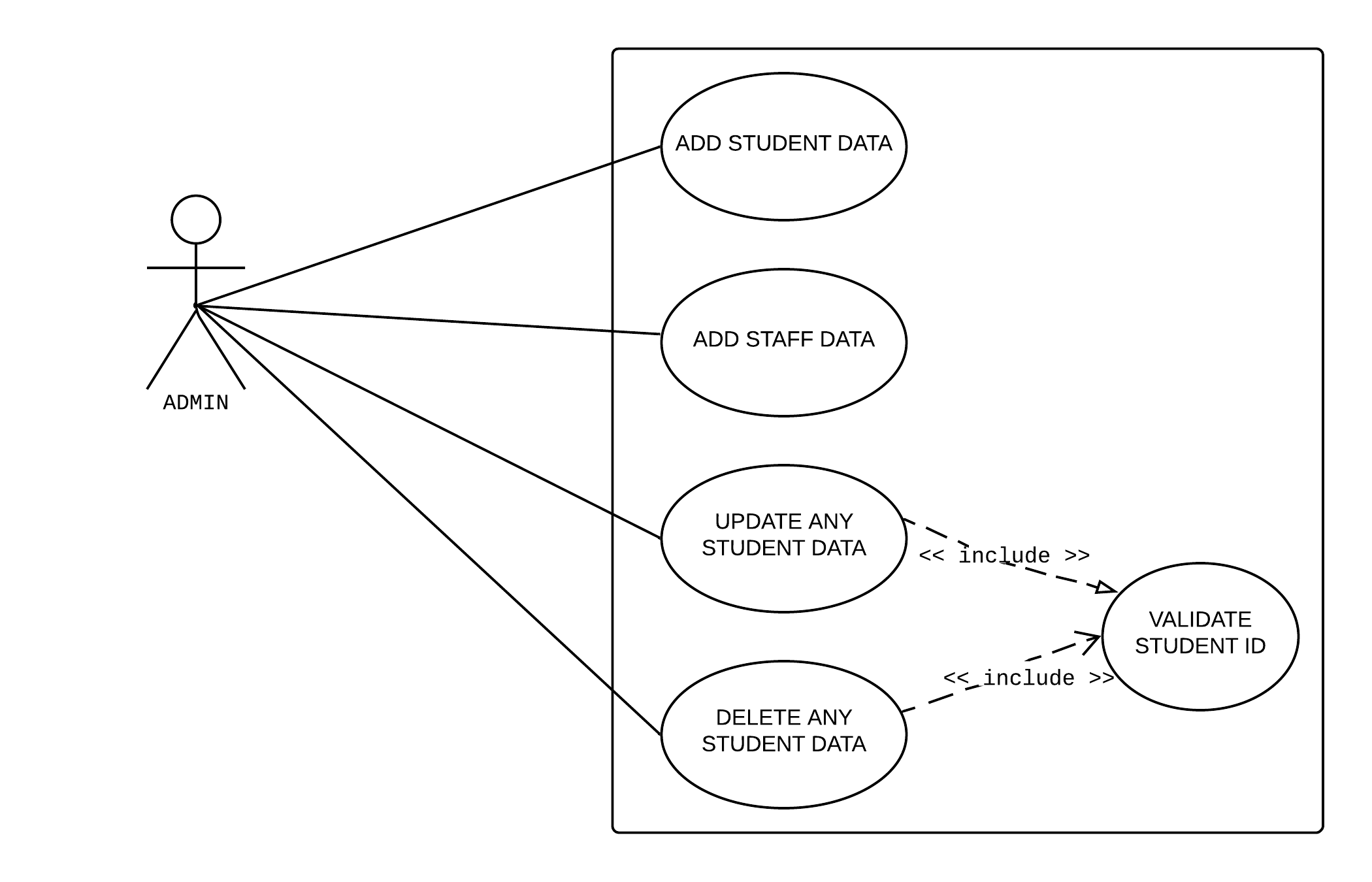


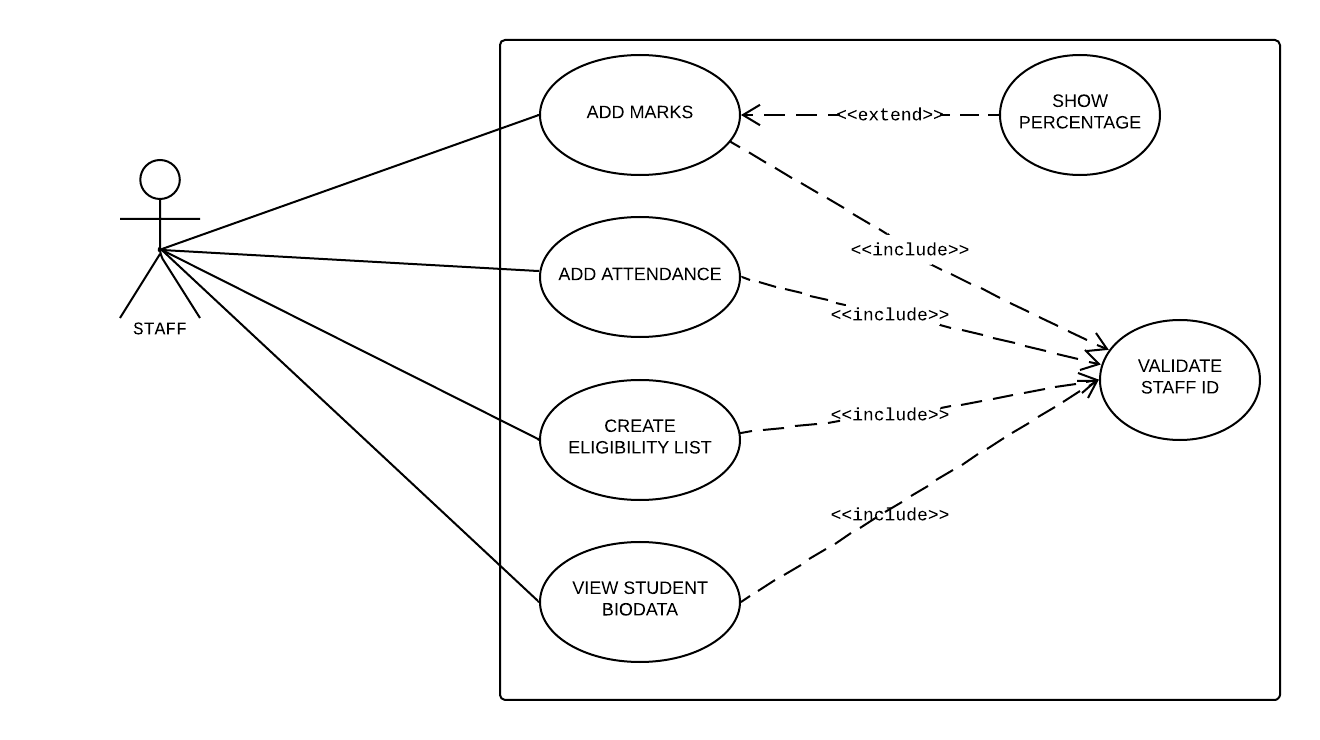
Fig : Admin Use-Case

**a) Actors**

* Administrator

**b) Use Cases**

* Add Student Data
* Add Staff Data
* Update any Student Data
* Delete any Student data

Fig : Staff Use-Case

**a) Actors**

* Staff

**b) Use Cases**

* Add Marks
* Add Attendance
* Create Eligibility List
* View Student BioData

1. **State transition diagram**

State-transition diagrams describe all of the states that an object can have, the events under which an object changes state (transitions), the conditions that must be fulfilled before the transition will occur (guards), and the activities undertaken during the life of an object (actions). State-transition diagrams are very useful for describing the behavior of individual objects over the full set of use cases that affect those objects.

**2.1 State-transition diagram for View Eligibility list use-Case**

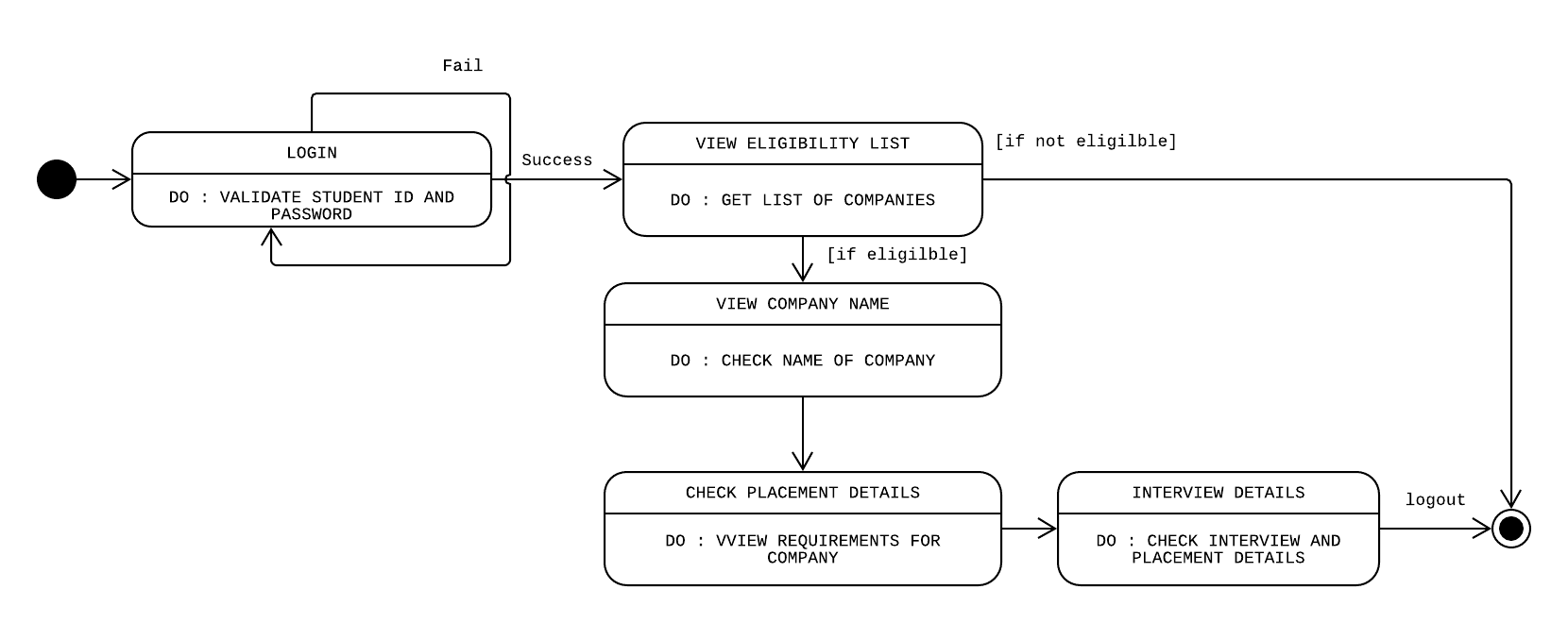
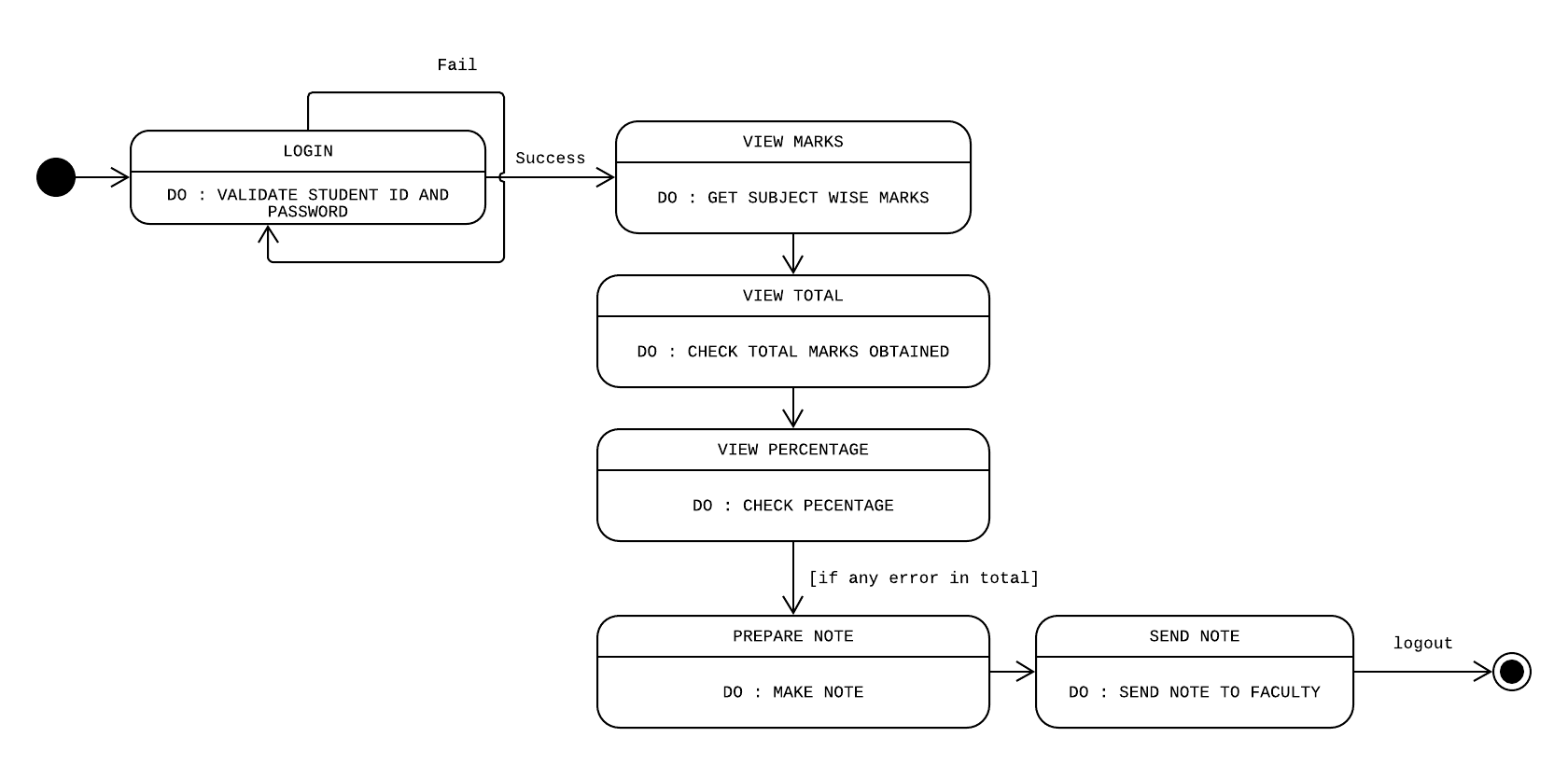
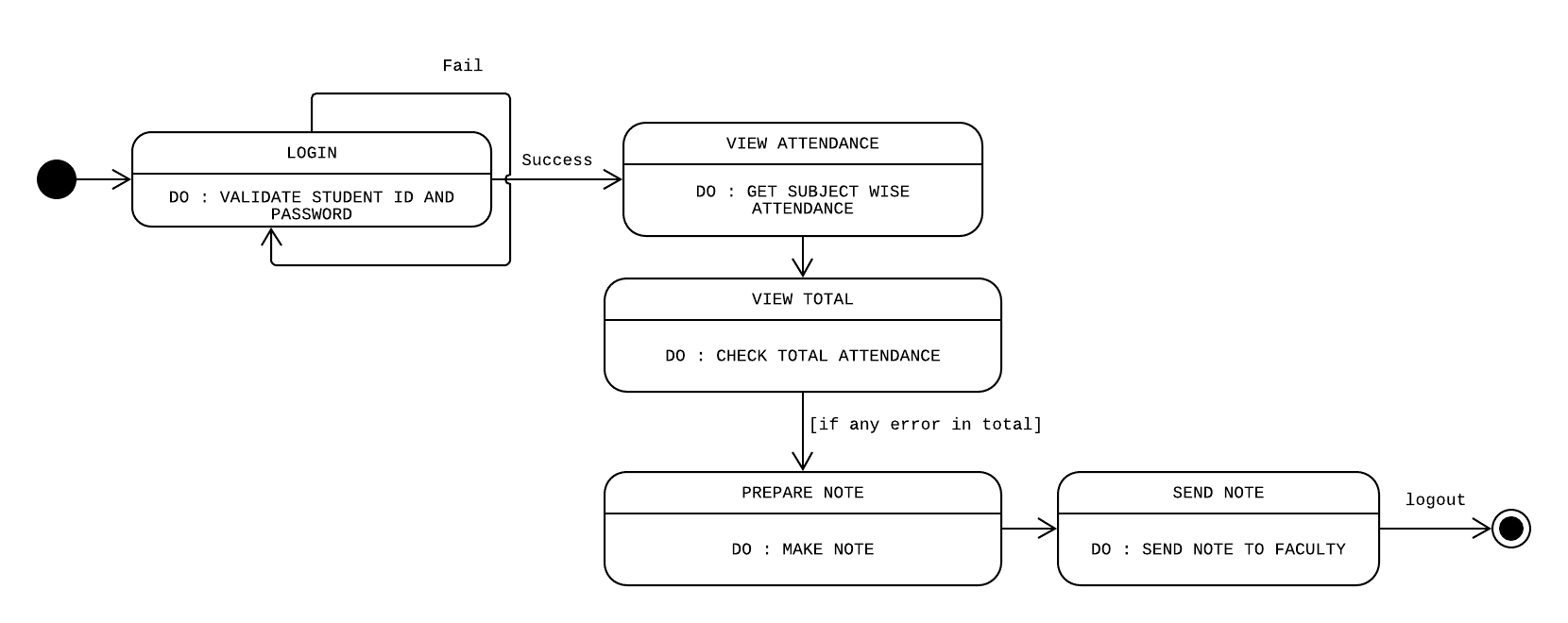
****

Fig : State-transition diagram for View Eligibility list use-Case

**2.2 State-transition diagram for View Marks use-Case**

Fig: State transition diagram for View Marks Use-case

**2.3 State-transition diagram for View Attendance use-Case**

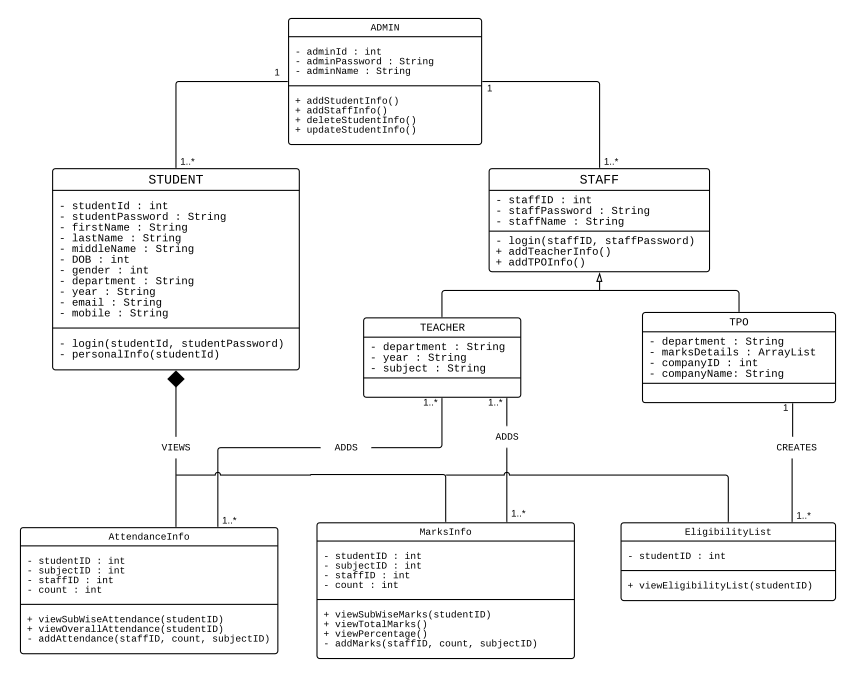
Fig : State-transition diagram for View Attendance use-Case

1. **Class diagram**

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

**A class is a description of a group of objects all with similar roles in the system, which consists of:**

* Structural features (attributes) define what objects of the class "know"
  + Represent the state of an object of the class
  + Are descriptions of the structural or static features of a class
* Behavioral features (operations) define what objects of the class "can do"
  + Define the way in which objects may interact
  + Operations are descriptions of behavioral or dynamic features of a class

****

**Class Data Models :**

* **Student**

student-id, student\_password, firstName,lastName, middleName, DOB, email, gender, year, department, mobile

* **Staff**

staff\_id, staffName

* **Teacher**

department,year,subject

* **TPO**

department, marksDetails, companyName, companyID

* **AttendanceInfo**

studentID, staffID, subjectName, count

* **MarksInfo**

studentID, staffID, subjectName, count

* **EligibilityList**

studentID

**Class Process Models :**

* **Student**

login(studentID, password), personalInfo(studentId)

* **Staff**

login(studentID, password)

* **AttendanceInfo**

viewSubWiseAttendace(studentID),viewOverallAttendance(studentID), addAttendance(staffID,count, subjectID)

* **MarksInfo**

viewSubWiseMarks(studentID),viewTotalMarks(),viewPercentage(), addMarks(staffID,count, subjectID)

* **EligibilityList**

viewEligibilityList(studentID)

**4. Activity diagram**

**4.1 Activity diagram to view Attendance**

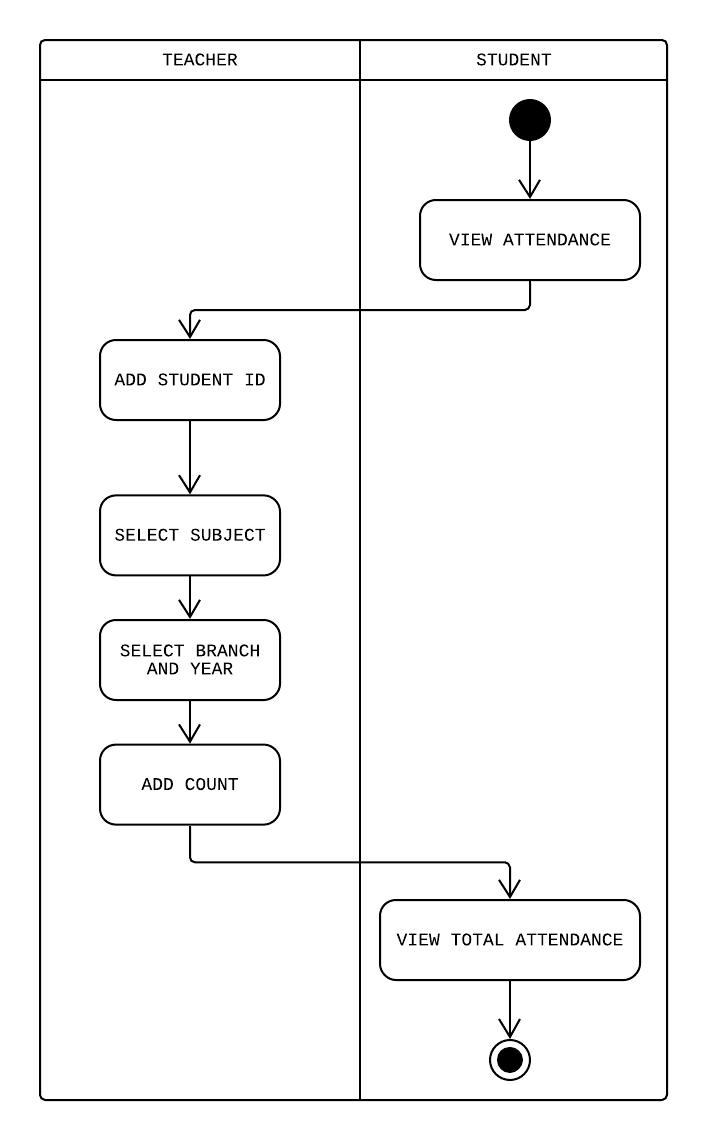
****

Fig: Activity diagram to view Attendance

**Activities**

* Teacher/Faculty is the one responsible for adding marks of the student. Faculty thus adds marks of the student studying in the respective subject of the teacher. Faculty selects the year, branch, and subject name whose marks details are to be filled in. Faculty simply write the student id and enter the marks.
* After logging in, student goes to Student Details page, where student can click on view Marks button. Upon clicking the button student can have choice either to view total marks, or subject wise marks or directly opt for percentage.

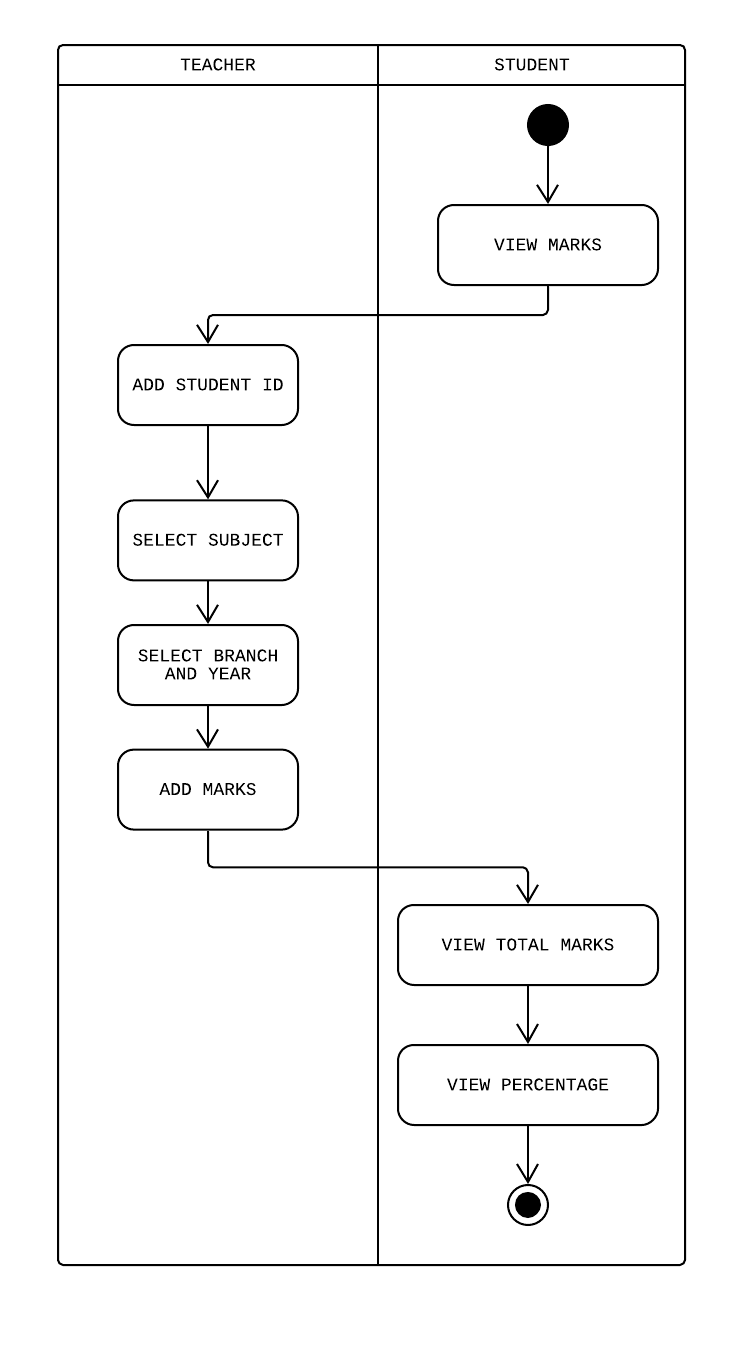
 **4.2 Activity diagram to view Marks**

Fig : Activity diagram to view Marks

**Activities**

* Teacher/Faculty is the one responsible for adding marks of the student. Faculty thus adds marks of the student studying in the respective subject of the teacher. Faculty selects the year, branch, and subject name whose marks details are to be filled in. Faculty simply write the student id and enter the marks.
* After logging in, student goes to Student Details page, where student can click on view Marks button. Upon clicking the button student can have choice either to view total marks, or subject wise marks or directly opt for percentage.

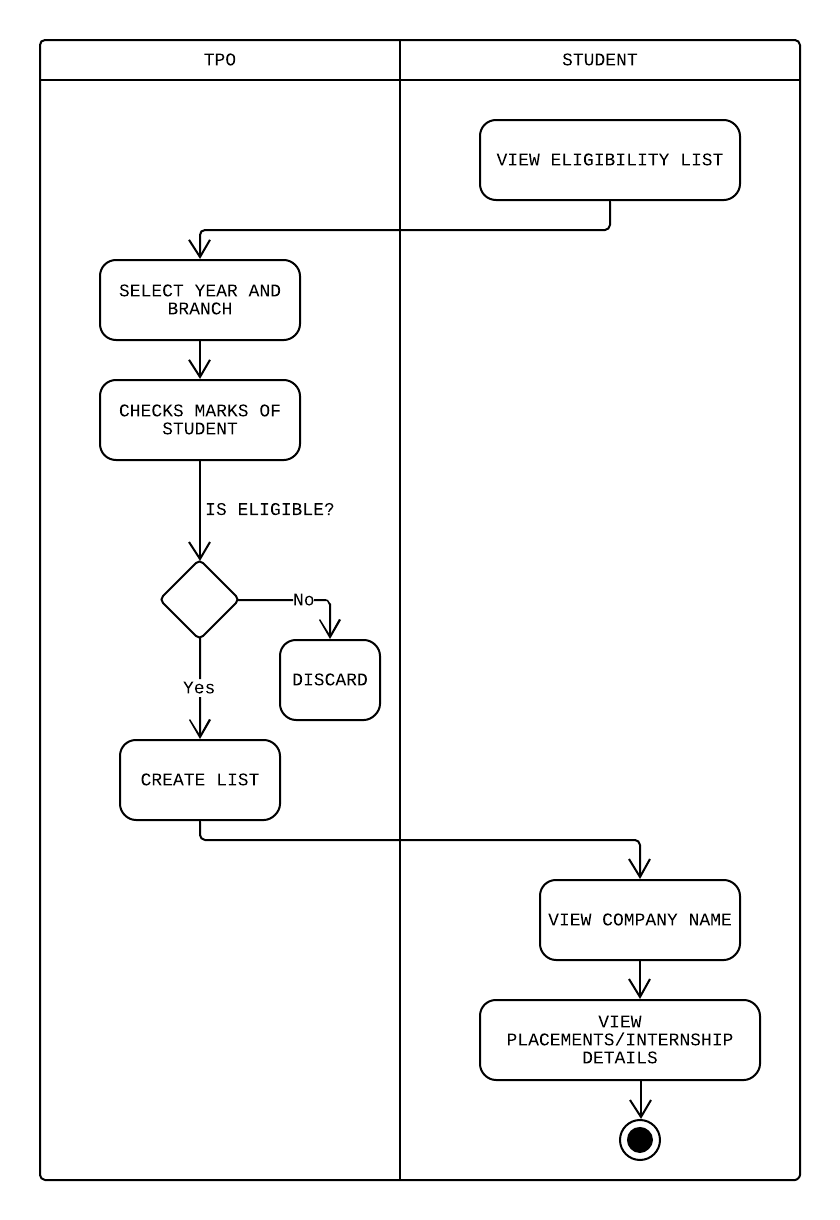
 **4.3 Activity diagram to view Eligibility list**

Fig : Activity diagram to view eligibility list

**Activities**

* TPO can view marks details of the student after faculty adds it. Now, TPO has a list of companies along with its criteria. TPO enters the name of the company who is bound to come for placements and enters the minimum criteria, On entering the criteria, a list of student id and student names is generated who are eligible for the drive. This list is stored into database.
* After logging in, student goes to Student Details page, where student can click on view Eligibility list. Upon clicking the button student can see whether or not he is eligible for placements for the company which is going to come.

**5. Sequence diagram**

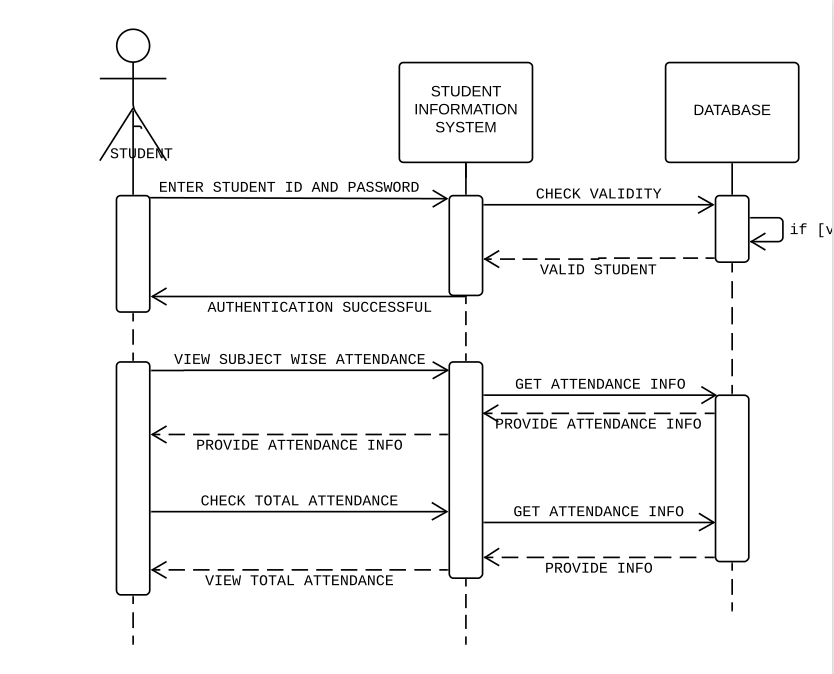


Fig : Sequence diagram to view attendance

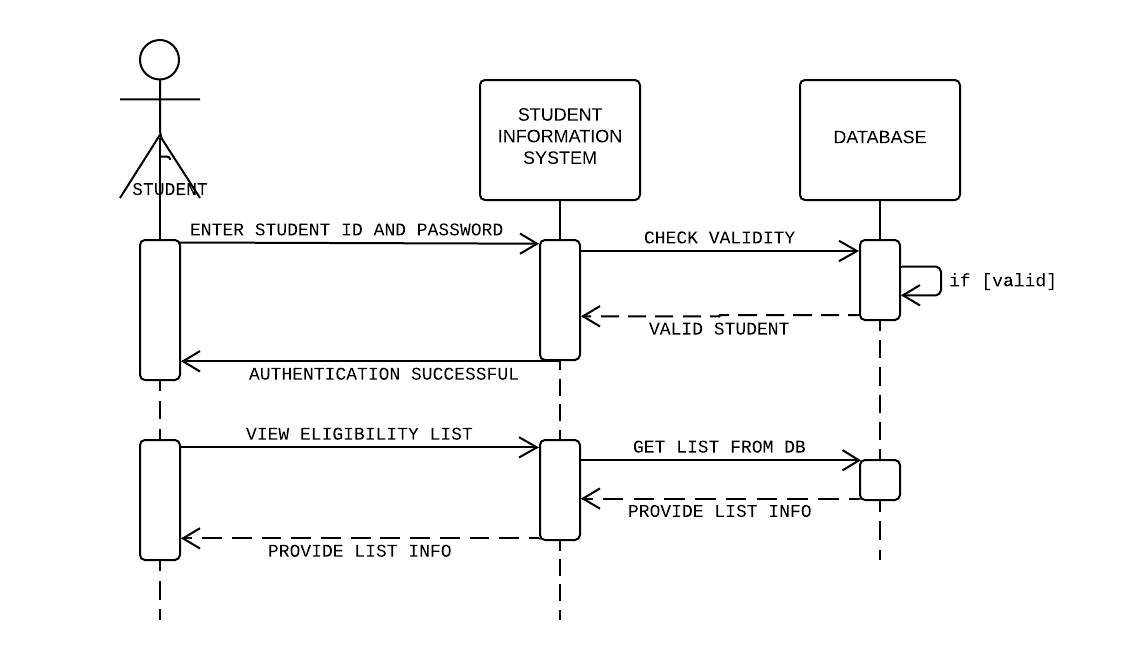


Fig : Sequence diagram to view eligibility list

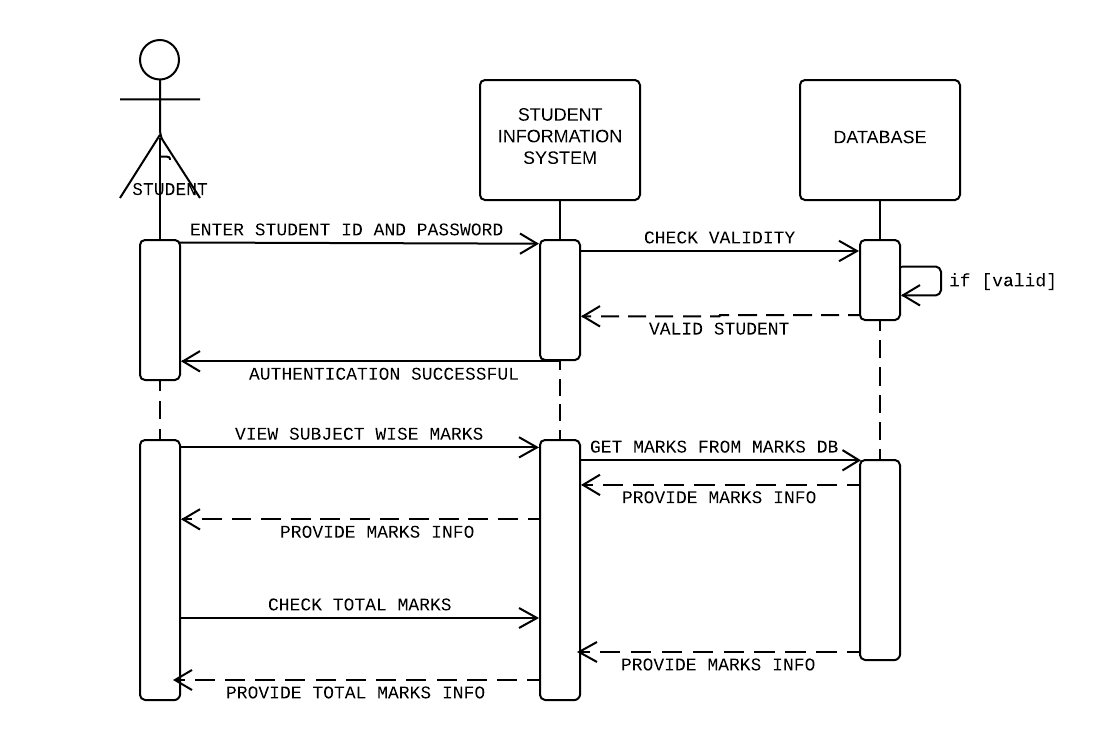
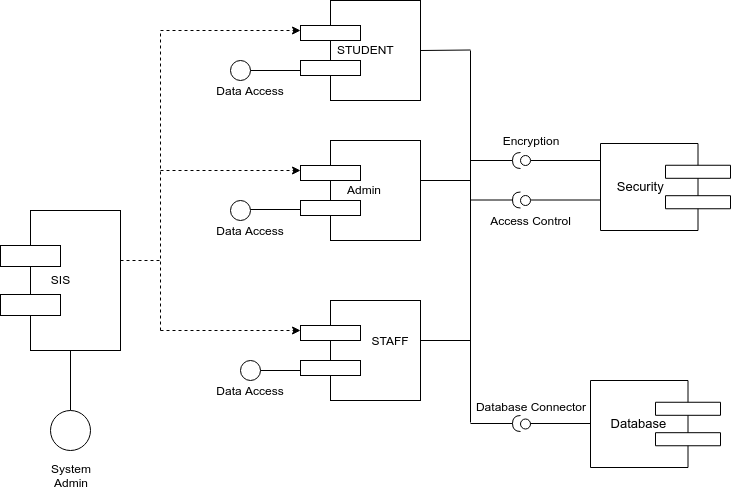
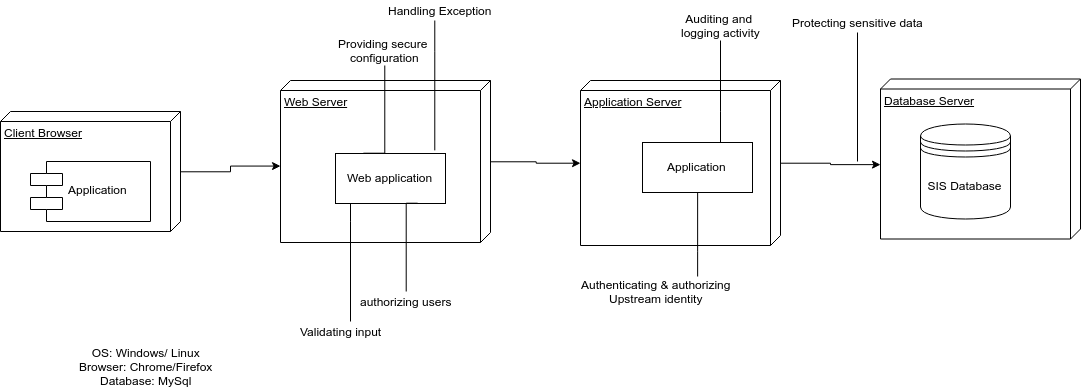


Fig : Sequence diagram to view Marks

**6. Component diagram**

****

**7. Deployment diagram**

****

**Web Engineering:**

Web engineering is basically all about designing and promoting web based systems. The application of systematic and quantifiable approaches to cost-effective analysis, design, implementation, testing, operation, and maintenance of high-quality Web applications.

**Web engineering versus software engineering Developing Web-based systems is significantly different from traditional software development and cause many additional challenges. There are slight differences in the nature and life cycle of Web-based and software systems. Multidisciplinary Building a complex Web-based system calls for knowledge and expertise from many different disciplines and requires a team of various people with expertise in different areas.**

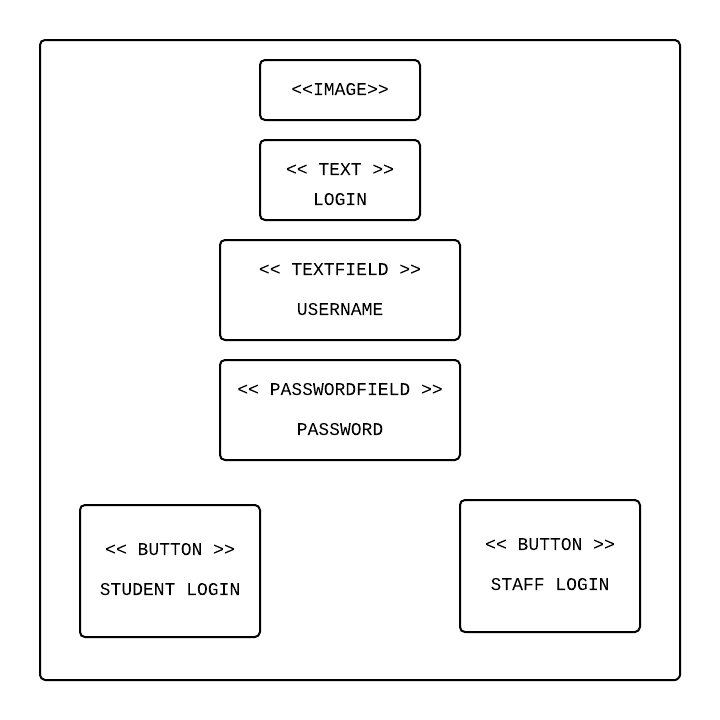
**1) LoginPage**

Fig : Login Page

**2) Students Details**

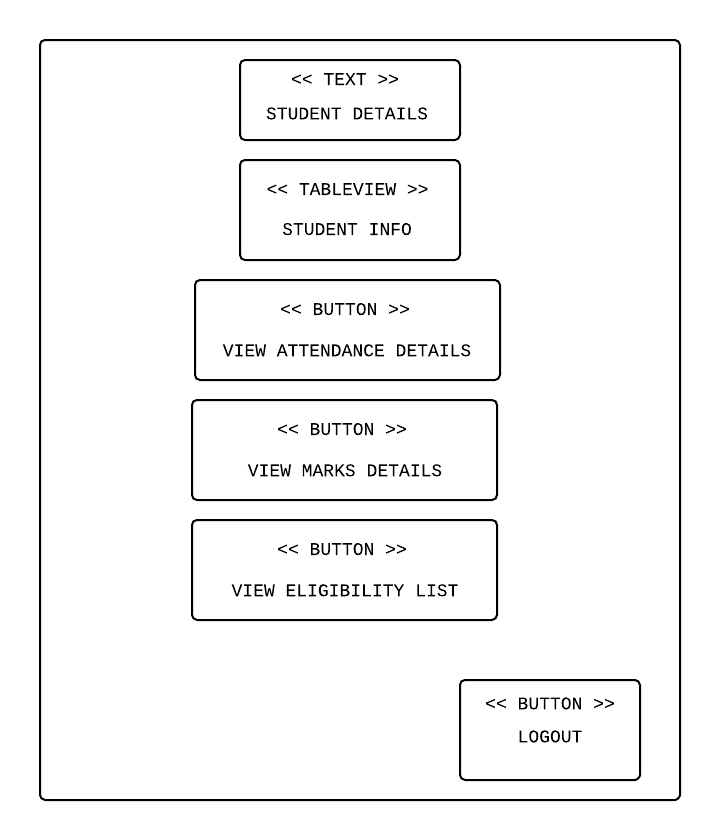
****

Fig : Student details page

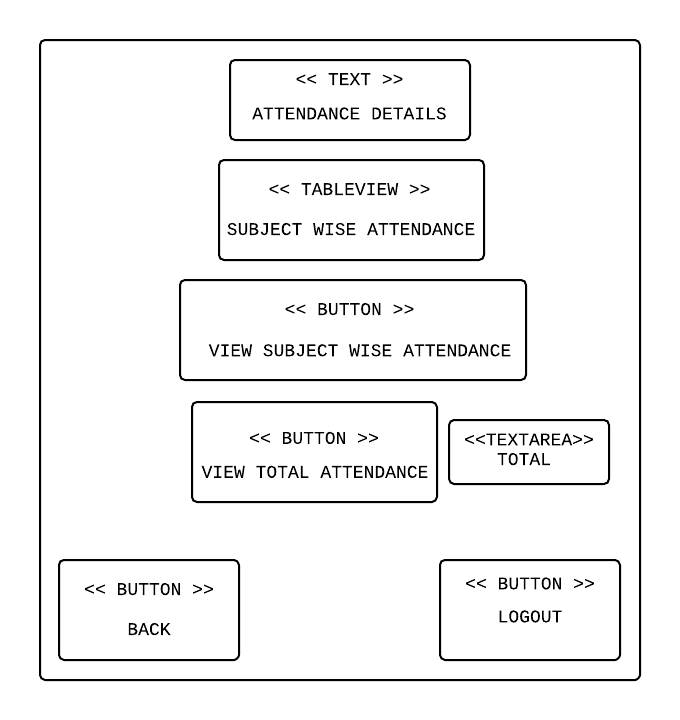
**3) Attendance Details page**

Fig : Attendance details page

**4) Marks Details**

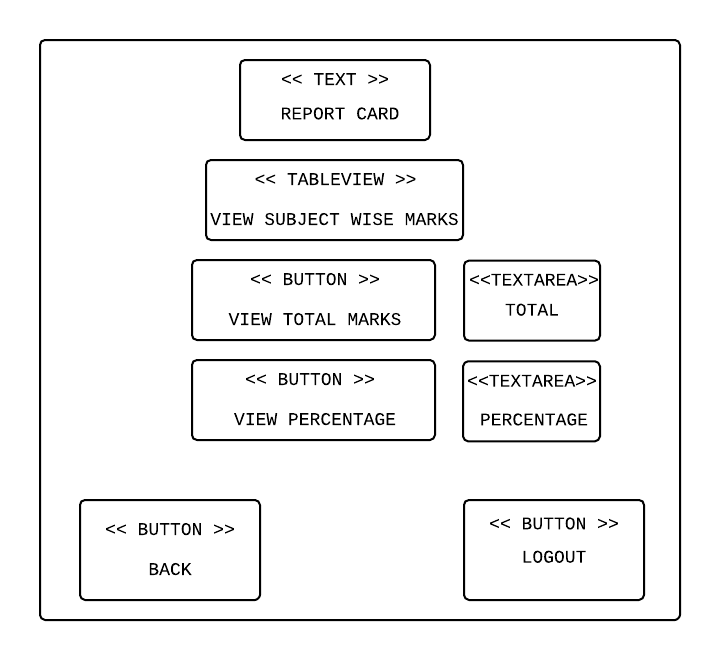


Fig : Marks details page

**5) Eligibility List page**

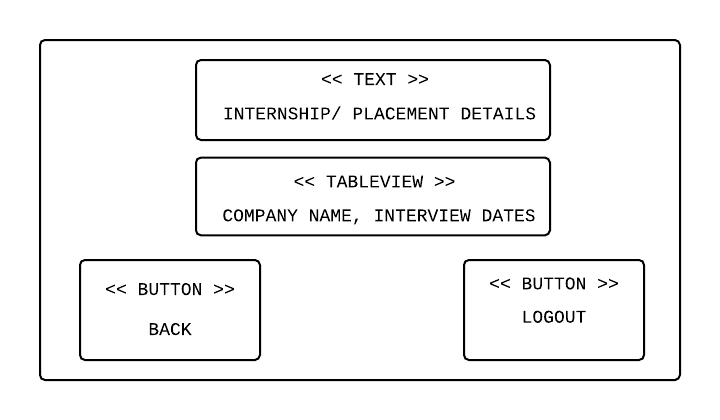
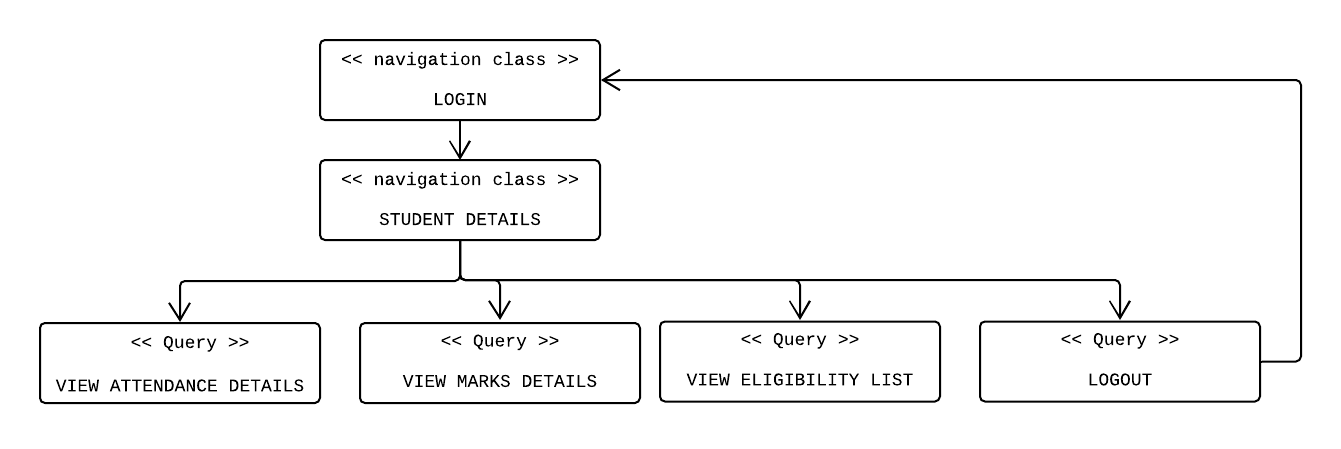
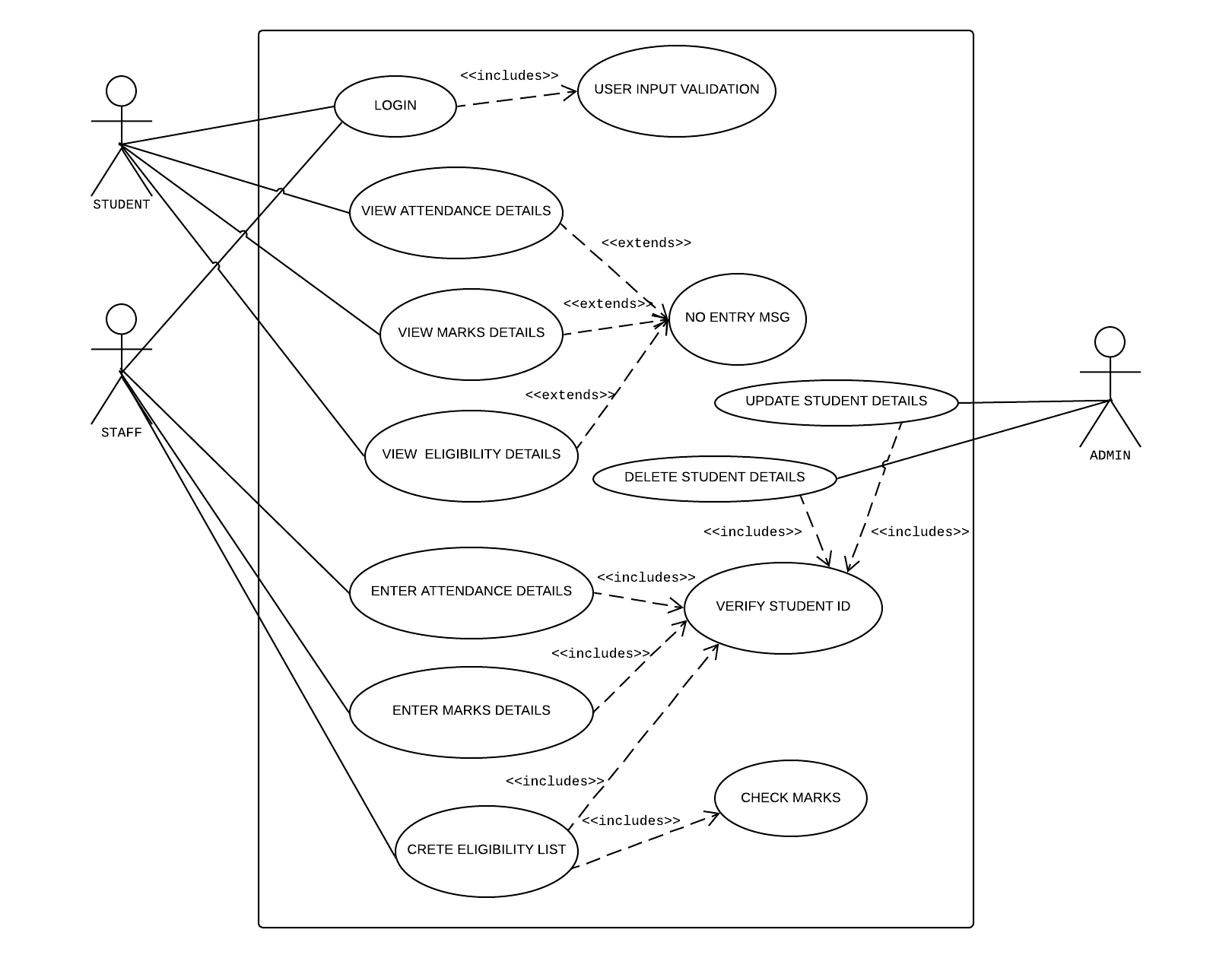
****

Fig :Eligibility list page

**6) Hypertext Structural Model**

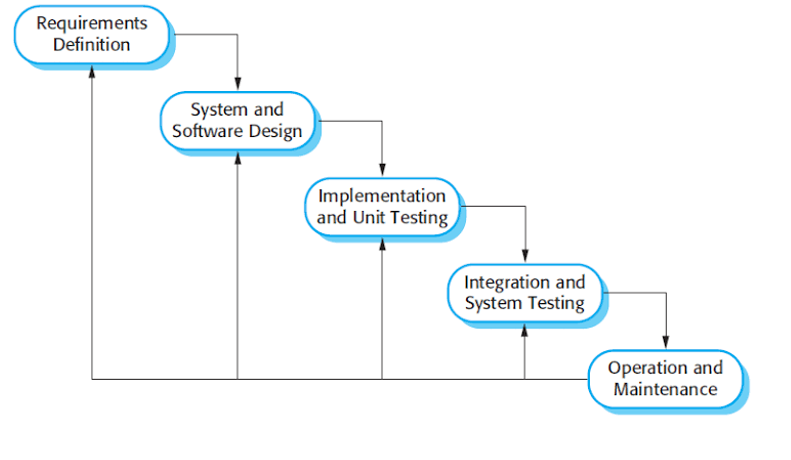


**7) Security Use case**



The proposed system has used waterfall model for developing the Workflow Management System.

In "The Waterfall" approach, the whole process of software development is divided into separate process phases. The phases in Waterfall model are: Requirement Specifications phase, Software Design, Implementation and Testing & Maintenance. All these phases are cascaded to each other so that second phase is started as and when defined set of goals are achieved for first phase and it is signed off, so the name "Waterfall Model". All the methods and processes undertaken in Waterfall Model are more visible.



Requirements Definition:

All possible requirements of the system to be developed are captured in this phase. Requirements are set of functionalities and constraints that the end-user (who will be using the system) expects from the system. The requirements are gathered from the end-user by consultation, these requirements are analyzed for their validity and the possibility of incorporating the requirements in the system to be development is also studied. Finally, a Requirement Specification document is created which serves the purpose of guideline for the next phase of the model.

The proposed Student Information system is a system that is capable of Managing student details and retrieving it upon request. The system is developed with functional and non-functional requirements.

**Functional Requirements**: The following requirements are captured for the intended use of the system:

* Student upon receiving student id and password can directly log in to system.
* Staff upon receiving staff id and password can directly log in to system
* By Designation of staff the access is provided by the system.
* Only admin is able to add the new user to the system.
* Student can only view the profile and academic details.

**Non-functional requirements**: The proposed system is designed to fulfill the following non-functional requirements.

* Performance Requirements: Performance of the system is dependent on the bandwidth of the internet.
* Security Requirements: There is only one authorized person who can see the student’s confidential information apart from the student himself. The information of the student is only available for the administrator.
* Software Quality Attributes: The system is user friendly, inter-operable and flexible.

**System & Software Design:**

The requirement specifications from first phase are studied in this phase and system design is prepared. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture. The system design specifications serve as input for the next phase of the model.

**Implementation & Unit Testing:**

On receiving system design documents, the work is divided in modules/units and actual coding is started. The system is first developed in small programs called units, which are integrated in the next phase. Initially home page, login page and sign up page was developed. After that database was created and “view profile” and “view experience” was created. Each unit is developed and tested for its functionality and after testing each unit meet their specification needs.

**Integration & System Testing:**

As specified above, the system is first divided in units, which are developed and tested for their functionalities. These units or pages were integrated into a complete system or folder in the Integration phase and tested to check if all modules/units coordinate between each other and the system as a whole behaves as per the specifications. After successfully testing the software, it is hosted locally.

Unit testing for “Login form” ,”Add Marks Details”

LOGIN FORM:

|  |  |  |  |
| --- | --- | --- | --- |
| Sr No | Test Case | Expected Result | Test Result |
| 1 | Enter valid user name and password & click on login button | Software should display main window | Successful |
| 2 | Enter invalid | Software should not display main window | Successful |

ADD MARKS FORM:

|  |  |  |  |
| --- | --- | --- | --- |
| Sr No | Test Case | Expected Result | Test Result |
| 1 | On the click of ADD MARKS button | At first the staff logs in using staff id and password, if staff is a faculty than only he can add marks of the student, if not he cannot. | Successful |

**Operations & Maintenance:**

This phase of "The Waterfall Model" is virtually never ending phase. Generally, problems with the system developed (which are not found during the development life cycle) come up after its practical use starts, so the issues related to the system are solved after deployment of the system. Not all the problems come in picture directly but they arise time to time and needs to be solved; hence this process is referred as Maintenance.

**Deployment Cost of the System**:

Since the system is developed in Java, it must be uploaded and run on server. Hence cost for deployment is directly proportional to that of a server.

1. Cost to deploy a website

Here are the four things we need to do to deploy a website.

* Web hosting and domain name

1. Domain Cost new: Rs700/- ~ Rs1000/-

2. Web Host Cost Shared: Rs200/- ~ Rs1000/-

3. VPS: Rs1000/- ~ Rs3500/-

* Web design (icons, logos, themes)

Designer Fees:

* Average: Rs2000/ hour
* Highest: Rs6000/hour
* Web content (writing and relevant images / videos)

Writer Fees:

* Average: Rs2000/hour
* Web development (special functions and backend systems)

Web developer fees:

* Average: Rs2000/hour

These are the primary areas where we spend our money on.

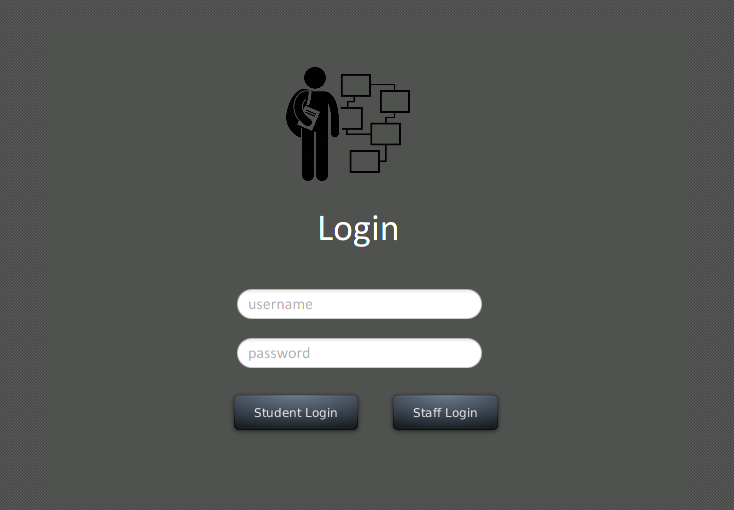
We can use WordPress as the foundation for running our website and use either free or premium designed templates.

1. Cost to host a website:

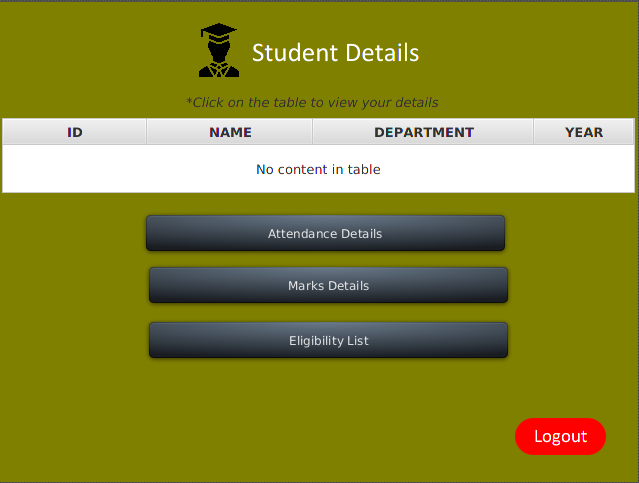
To host website the cost price for “GoDaddy.com” is Rs3000/month

**SnapShots of the System**

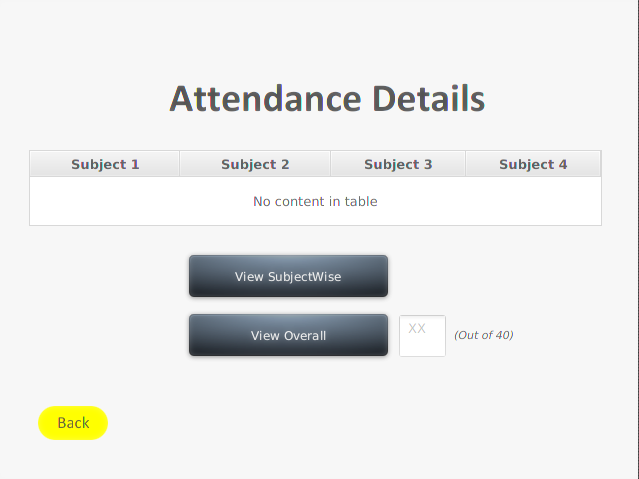
**Common Login Page for Student and Staff**



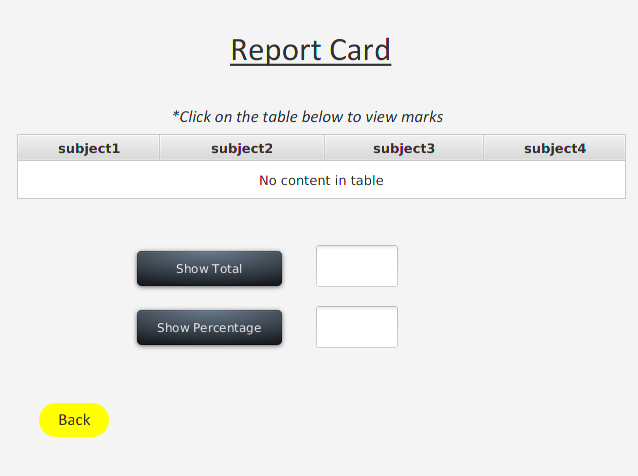
**Student Details Page**



**Attendance Details Page**

****

**Marks Details Page**

****

**Important Code**

**For Student and Staff**

**LoginController.java**

package SIS;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.\*;

import javafx.scene.control.cell.PropertyValueFactory;

import javafx.stage.Stage;

import java.io.IOException;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

public class LoginController {

public static String staffuname,staffpass;

public static String stuuname, stupass;

@FXML

private TextField username;

@FXML

private PasswordField password;

@FXML

private Button stulogin;

@FXML

private Button stafflogin;

/\* Staff Login \*/

public void staffLogin(ActionEvent actionEvent) throws IOException {

staffuname = username.getText();

staffpass = password.getText();

if (staffLoginQuery.CheckLoginUser(staffuname, staffpass)) { // sending data to databasehandler class to connection data

stafflogin.getScene().getWindow().hide();

Stage dashboardStage = new Stage();

dashboardStage.setTitle("Staff Details");

Parent root = FXMLLoader.load(getClass().getResource("staffDetails.fxml"));

Scene scene = new Scene(root);

dashboardStage.setScene(scene);

dashboardStage.show();

}

else {

alert();

}

}

/\* Student Login \*/

public void studentLogin() throws IOException {

stuuname = username.getText();

stupass = password.getText();

if (studentLoginQuery.CheckLoginUser(stuuname, stupass)) { // sending data to databasehandler class to connection data

stulogin.getScene().getWindow().hide();

Stage dashboardStage = new Stage();

Parent root = FXMLLoader.load(getClass().getResource("studentDetails.fxml"));

dashboardStage.setTitle("Student Details");

Scene scene = new Scene(root);

dashboardStage.setScene(scene);

dashboardStage.show();

}

else{

alert();

}

}

private void alert() {

Alert alert = new Alert(Alert.AlertType.INFORMATION);

alert.setTitle("Error");

alert.setContentText("Username or Password Incorrect!!. Please enter Correctly.");

alert.show();

// reseting user and pass field

username.setText("");

password.setText("");

}

}

**studentLoginQuery.java**

package SIS;

import java.sql.\*;

public class studentLoginQuery extends DatabaseController{

public static boolean CheckLoginUser(String uname, String pass) {

System.out.println(uname+" "+pass);//get input from login system module

Connection connection = GetDatabaseConnection();

// String checkQuery = "select \* from loginDetails where user = '"+uname+"' and pass = '"+pass+"' ";

String checkQuery = "select \*from studentLogin where id = ? and password = ? "; // i don't use id from database table.

PreparedStatement preparedStatement = null;

boolean status = false; //initially false

try {

preparedStatement = connection.prepareStatement(checkQuery);

preparedStatement.setString(1, uname); // dynamically sending username

preparedStatement.setString(2, pass); // sending password to checkquery statement

ResultSet resultSet = preparedStatement.executeQuery();

// while (resultSet.next()) {

// return status;

// }

status = resultSet.next();

preparedStatement.close();

return status;

} catch (SQLException e) {

// e.getLocalizedMessage();

e.printStackTrace();

}

return status;

}

}

**staffLoginQuery.java**

package SIS;

import java.sql.\*;

public class staffLoginQuery extends DatabaseController{

public static boolean CheckLoginUser(String uname, String pass) {

System.out.println(uname+" "+pass);//get input from login system module

Connection connection = GetDatabaseConnection();

// String checkQuery = "select \* from loginDetails where user = '"+uname+"' and pass = '"+pass+"' ";

String checkQuery = "select \*from staffDetails where id = ? and password = ? "; // i don't use id from database table.

PreparedStatement preparedStatement = null;

boolean status = false; //initially false

try {

preparedStatement = connection.prepareStatement(checkQuery);

preparedStatement.setString(1, uname); // dynamically sending username

preparedStatement.setString(2, pass); // sending password to checkquery statement

ResultSet resultSet = preparedStatement.executeQuery();

// while (resultSet.next()) {

// return status;

// }

status = resultSet.next();

preparedStatement.close();

return status;

} catch (SQLException e) {

// e.getLocalizedMessage();

e.printStackTrace();

}

return status;

}

}

**AttendanceQuery.java**

package SIS;

import java.io.IOException;

import java.sql.\*;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.fxml.Initializable;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.\*;

import javafx.scene.control.cell.PropertyValueFactory;

import javafx.stage.Stage;

import java.net.URL;

import java.util.ResourceBundle;

public class AttendanceQuery implements Initializable {

@FXML

private Button back;

@FXML

private Button viewOverall;

@FXML

private Button viewSubWise;

@FXML

private TableView<AttendanceData> subWiseTable;

@FXML

private TableColumn<AttendanceData, String> id;

@FXML

private TableColumn<AttendanceData, String> sub1;

@FXML

private TableColumn<AttendanceData, String> sub2;

@FXML

private TableColumn<AttendanceData, String> sub3;

@FXML

private TableColumn<AttendanceData, String> sub4;

private ObservableList<AttendanceData> data;

@FXML

private TextArea totalAttendance;

private String username = LoginController.stuuname;

public void back(ActionEvent actionEvent) throws IOException {

back.getScene().getWindow().hide();

Stage dashboardStage = new Stage();

Parent root = FXMLLoader.load(getClass().getResource("studentDetails.fxml"));

dashboardStage.setTitle("Student Details");

Scene scene = new Scene(root);

dashboardStage.setScene(scene);

dashboardStage.show();

}

public void subWise(ActionEvent actionEvent) {

Connection conn = DatabaseController.GetDatabaseConnection();

this.data = FXCollections.observableArrayList();

String checkQuery = "select \*from stuSubAttendanceDetails where id = ? ";

PreparedStatement preparedStatement = null;

try {

preparedStatement = conn.prepareStatement(checkQuery);

preparedStatement.setString(1, username);

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

this.data.add(new AttendanceData(resultSet.getString("subject1"),resultSet.getString("subject2"),resultSet.getString("subject3"),resultSet.getString("subject4")));

}

} catch (SQLException e) {

e.printStackTrace();

}

this.sub1.setCellValueFactory(new PropertyValueFactory<AttendanceData, String>("sub1"));

this.sub2.setCellValueFactory(new PropertyValueFactory<AttendanceData, String>("sub2"));

this.sub3.setCellValueFactory(new PropertyValueFactory<AttendanceData, String>("sub3"));

this.sub4.setCellValueFactory(new PropertyValueFactory<AttendanceData, String>("sub4"));

this.subWiseTable.setItems(null);

this.subWiseTable.setItems(this.data);

}

public void viewOverall(ActionEvent actionEvent) {

Connection conn = DatabaseController.GetDatabaseConnection();

String checkQuery2 = "SELECT sum(subject1 + subject2 + subject3 + subject4) as total from stuSubAttendanceDetails where id= ?";

PreparedStatement preparedStatement2 = null;

try {

preparedStatement2 = conn.prepareStatement(checkQuery2);

preparedStatement2.setString(1, username);

ResultSet resultSet = preparedStatement2.executeQuery();

System.out.println(resultSet);

while(resultSet.next()) {

int result = resultSet.getInt(1);

System.out.println(result);

totalAttendance.setText(String.valueOf(result));

// showTotalTextField.setText(resultSet.getString(total));

}

} catch (SQLException e) {

e.printStackTrace();

}

}

@Override

public void initialize(URL url, ResourceBundle resourceBundle) {

}

}

**MarksDetails.java**

package SIS;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.control.TableColumn;

import javafx.scene.control.TableView;

import javafx.scene.control.TextArea;

import javafx.scene.control.cell.PropertyValueFactory;

import javafx.scene.input.MouseEvent;

import javafx.stage.Stage;

import java.io.IOException;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

public class MarksDetails {

@FXML

private Button back;

@FXML

private TableView<MarksData> reportCard;

private ObservableList<MarksData> data;

@FXML

private TableColumn<MarksData, String> subject1;

@FXML

private TableColumn<MarksData, String> subject2;

@FXML

private TableColumn<MarksData, String> subject3;

@FXML

private TableColumn<MarksData, String> subject4;

String StudentUserName;

@FXML

private TextArea showTotalTextField;

@FXML

private TextArea showPerTextField;

public void reportCard(MouseEvent mouseEvent) {

StudentUserName = LoginController.stuuname;

Connection conn = DatabaseController.GetDatabaseConnection();

this.data = FXCollections.observableArrayList();

String checkQuery1 = "select subject1,subject2,subject3,subject4 from stuMarks where id = ? ";

PreparedStatement preparedStatement1 = null;

try {

preparedStatement1 = conn.prepareStatement(checkQuery1);

preparedStatement1.setString(1, StudentUserName);

ResultSet resultSet1 = preparedStatement1.executeQuery();

while (resultSet1.next()) {

this.data.add(new MarksData(resultSet1.getString("subject1"), resultSet1.getString("subject2"), resultSet1.getString("subject3"), resultSet1.getString("subject4")));

}

} catch (SQLException e) {

e.printStackTrace();

}

this.subject1.setCellValueFactory(new PropertyValueFactory<MarksData, String>("subject1"));

this.subject2.setCellValueFactory(new PropertyValueFactory<MarksData, String>("subject2"));

this.subject3.setCellValueFactory(new PropertyValueFactory<MarksData, String>("subject3"));

this.subject4.setCellValueFactory(new PropertyValueFactory<MarksData, String>("subject4"));

this.reportCard.setItems(null);

this.reportCard.setItems(this.data);

}

public void showTotal(ActionEvent actionEvent) {

Connection conn = DatabaseController.GetDatabaseConnection();

String checkQuery2 = "SELECT sum(subject1 + subject2 + subject3 + subject4) as total from stuMarks where id= ?";

PreparedStatement preparedStatement2 = null;

try {

preparedStatement2 = conn.prepareStatement(checkQuery2);

preparedStatement2.setString(1, StudentUserName);

ResultSet resultSet = preparedStatement2.executeQuery();

System.out.println(resultSet);

while(resultSet.next()) {

int result = resultSet.getInt(1);

System.out.println(result);

showTotalTextField.setText(String.valueOf(result));

// showTotalTextField.setText(resultSet.getString(total));

}

} catch (SQLException e) {

e.printStackTrace();

}

}

public void showPer(ActionEvent actionEvent) {

Connection conn = DatabaseController.GetDatabaseConnection();

String checkQuery3 = "SELECT (((subject1 + subject2 + subject3 + subject4)/400)\*100) as percentage from stuMarks where id= ?";

PreparedStatement preparedStatement3 = null;

try {

preparedStatement3 = conn.prepareStatement(checkQuery3);

preparedStatement3.setString(1, StudentUserName);

ResultSet resultSet = preparedStatement3.executeQuery();

System.out.println(resultSet);

while(resultSet.next()) {

int result = resultSet.getInt(1);

System.out.println(result);

showPerTextField.setText(String.valueOf(result));

// showTotalTextField.setText(resultSet.getString(total));

}

} catch (SQLException e) {

e.printStackTrace();

}

}

public void back(ActionEvent actionEvent) throws IOException {

back.getScene().getWindow().hide();

Stage dashboardStage = new Stage();

Parent root = FXMLLoader.load(getClass().getResource("studentDetails.fxml"));

dashboardStage.setTitle("Student Details");

Scene scene = new Scene(root);

dashboardStage.setScene(scene);

dashboardStage.show();

}

}

**studentDetails.java**

package SIS;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.Alert;

import javafx.scene.control.Button;

import javafx.scene.control.TableColumn;

import javafx.scene.control.TableView;

import javafx.scene.control.cell.PropertyValueFactory;

import javafx.stage.Stage;

import java.io.IOException;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

public class studentDetails extends LoginController{

@FXML

private Button attendanceDetails;

@FXML

private Button logout;

@FXML

private Button MarksDetails;

@FXML

TableView<StudentData> stuDetails;

@FXML

private TableColumn<StudentData, String> id;

@FXML

private TableColumn<StudentData, String> name;

@FXML

private TableColumn<StudentData, String> department;

@FXML

private TableColumn<StudentData, String> year;

private ObservableList<StudentData> data;

private static String StudentUserName;

// public studentDetails() {

// this.showdata();

// }

public void showdata () {

StudentUserName = LoginController.stuuname;

Connection conn = DatabaseController.GetDatabaseConnection();

this.data = FXCollections.observableArrayList();

String checkQuery1 = "select id,firstName,department,year from studentDetails where id = ? ";

PreparedStatement preparedStatement1 = null;

try {

preparedStatement1 = conn.prepareStatement(checkQuery1);

preparedStatement1.setString(1, StudentUserName);

ResultSet resultSet1 = preparedStatement1.executeQuery();

while (resultSet1.next()) {

this.data.add(new StudentData(resultSet1.getString("id"), resultSet1.getString("firstName"), resultSet1.getString("department"), resultSet1.getString("year")));

}

} catch (SQLException e) {

e.printStackTrace();

}

this.id.setCellValueFactory(new PropertyValueFactory<StudentData,String>("id"));

this.name.setCellValueFactory(new PropertyValueFactory<StudentData, String>("name"));

this.department.setCellValueFactory(new PropertyValueFactory<StudentData, String>("department"));

this.year.setCellValueFactory(new PropertyValueFactory<StudentData, String>("year"));

this.stuDetails.setItems(null);

this.stuDetails.setItems(this.data);

}

public void attendanceDetails(ActionEvent event) throws IOException, SQLException {

// if (AttendanceQuery.CheckAttendance(username)) { // sending data to databasehandler class to connection data

// AttendanceQuery.CheckAttendance(username);

attendanceDetails.getScene().getWindow().hide();

Stage dashboardStage = new Stage();

Parent root = FXMLLoader.load(getClass().getResource("ViewAtendance.fxml"));

dashboardStage.setTitle("Attendance Info");

Scene scene = new Scene(root);

dashboardStage.setScene(scene);

dashboardStage.show();

// } else {

// alert();

// }

}

private void alert() {

Alert alert = new Alert(Alert.AlertType.INFORMATION);

alert.setTitle("Error");

alert.setContentText("Please enter Correctly.");

alert.show();

// reseting user and pass field

}

public void logout(ActionEvent actionEvent) throws IOException {

logout.getScene().getWindow().hide();

Stage dashboardStage = new Stage();

Parent root = FXMLLoader.load(getClass().getResource("login.fxml"));

dashboardStage.setTitle("Student Details");

Scene scene = new Scene(root);

dashboardStage.setScene(scene);

dashboardStage.show();

}

public void marksDetails(ActionEvent actionEvent) throws IOException {

MarksDetails.getScene().getWindow().hide();

Stage dashboardStage = new Stage();

Parent root = FXMLLoader.load(getClass().getResource("MarksDetails.fxml"));

dashboardStage.setTitle("Student Report Card");

Scene scene = new Scene(root);

dashboardStage.setScene(scene);

dashboardStage.show();

}

public void eligibilityDetails(ActionEvent actionEvent) {

}

}

**MarksInsertionQuery.java**

package SIS;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.Alert;

import javafx.scene.control.Button;

import javafx.scene.control.TextField;

import javafx.stage.Stage;

import java.io.IOException;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

public class MarksInsertionQuery {

@FXML

private Button logout;

@FXML

private Button back;

@FXML

private TextField id;

@FXML

private TextField marks;

@FXML

private Button insert;

String username = LoginController.staffuname;

String subName;

public void insert(ActionEvent actionEvent) {

Connection connection = DatabaseController.GetDatabaseConnection();

String checkQuery = "select subject from staffDetails where id = ? ";

PreparedStatement preparedStatement = null;

try {

preparedStatement = connection.prepareStatement(checkQuery);

preparedStatement.setString(1, username);

ResultSet resultSet = preparedStatement.executeQuery();

while(resultSet.next()) {

subName = resultSet.getString("subject");

}

} catch (SQLException e) {

e.printStackTrace();

}

System.out.println(subName);

if(!subName.contains("placements")) {

String stuID = id.getText();

String subMarks = marks.getText();

System.out.println(stuID+" "+subMarks);

if(!subMarks.isEmpty()&&!stuID.isEmpty()) {

Connection conn = DatabaseController.GetDatabaseConnection();

//NSERT INTO Student(ROLL\_NO,NAME,Age) SELECT ROLL\_NO, NAME, Age FROM LateralStudent;

String checkUpdateQuery = "update stuMarks set "+subName+" = ? where id = ?";

PreparedStatement prepareStatement = null;

try {

prepareStatement = conn.prepareStatement(checkUpdateQuery);

prepareStatement.setString(1, subMarks); // sending password to checkquery statement

prepareStatement.setString(2, stuID);

int resultSet = prepareStatement.executeUpdate();

id.setText(" ");

marks.setText(" ");

} catch (SQLException e) {

e.printStackTrace();

}

}

else {

Alert alert = new Alert(Alert.AlertType.INFORMATION);

alert.setTitle("Error");

alert.setContentText("Fields cannot be empty!!");

alert.show();

}

}

}

public void logout(ActionEvent actionEvent) throws IOException {

logout.getScene().getWindow().hide();

Stage dashboardStage = new Stage();

Parent root = FXMLLoader.load(getClass().getResource("login.fxml"));

dashboardStage.setTitle("Student Details");

Scene scene = new Scene(root);

dashboardStage.setScene(scene);

dashboardStage.show();

}

public void back(ActionEvent actionEvent) throws IOException {

back.getScene().getWindow().hide();

Stage dashboardStage = new Stage();

Parent root = FXMLLoader.load(getClass().getResource("staffDetails.fxml"));

dashboardStage.setTitle("Student Details");

Scene scene = new Scene(root);

dashboardStage.setScene(scene);

dashboardStage.show();

}

}

**AttendanceInsertionQuery.java**

package SIS;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.Alert;

import javafx.scene.control.Button;

import javafx.scene.control.TextField;

import javafx.stage.Stage;

import java.io.IOException;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

public class AttendanceInsertionQuery{

@FXML

private Button logout;

@FXML

private Button back;

@FXML

private TextField id;

@FXML

private TextField count;

String username = LoginController.staffuname;

String subName;

public void insert(ActionEvent actionEvent) {

Connection connection = DatabaseController.GetDatabaseConnection();

String checkQuery = "select subject from staffDetails where id = ? ";

PreparedStatement preparedStatement = null;

try {

preparedStatement = connection.prepareStatement(checkQuery);

preparedStatement.setString(1, username);

ResultSet resultSet = preparedStatement.executeQuery();

while(resultSet.next()) {

subName = resultSet.getString("subject");

}

} catch (SQLException e) {

e.printStackTrace();

}

System.out.println(subName);

if(!subName.contains("placements")) {

String stuID = id.getText();

String subAttendance = count.getText();

System.out.println(stuID+" "+subAttendance);

if(!subAttendance.isEmpty()&&!stuID.isEmpty()) {

Connection conn = DatabaseController.GetDatabaseConnection();

//NSERT INTO Student(ROLL\_NO,NAME,Age) SELECT ROLL\_NO, NAME, Age FROM LateralStudent;

String checkUpdateQuery = "update stuSubAttendanceDetails set "+subName+" = ? where id = ?";

PreparedStatement prepareStatement = null;

try {

prepareStatement = conn.prepareStatement(checkUpdateQuery);

prepareStatement.setString(1, subAttendance); // sending password to checkquery statement

prepareStatement.setString(2, stuID);

int resultSet = prepareStatement.executeUpdate();

id.setText(" ");

count.setText(" ");

} catch (SQLException e) {

e.printStackTrace();

}

}

else {

Alert alert = new Alert(Alert.AlertType.INFORMATION);

alert.setTitle("Error");

alert.setContentText("Fields cannot be empty!!");

alert.show();

}

}

}

public void logout(ActionEvent actionEvent) throws IOException {

logout.getScene().getWindow().hide();

Stage dashboardStage = new Stage();

Parent root = FXMLLoader.load(getClass().getResource("login.fxml"));

dashboardStage.setTitle("Student Details");

Scene scene = new Scene(root);

dashboardStage.setScene(scene);

dashboardStage.show();

}

public void back(ActionEvent actionEvent) throws IOException {

back.getScene().getWindow().hide();

Stage dashboardStage = new Stage();

Parent root = FXMLLoader.load(getClass().getResource("staffDetails.fxml"));

dashboardStage.setTitle("Student Details");

Scene scene = new Scene(root);

dashboardStage.setScene(scene);

dashboardStage.show();

}

}

**For ADMIN**

**AdminLoginController.java**

package Admin;

import SIS.DatabaseController;

import SIS.staffLoginQuery;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.Alert;

import javafx.scene.control.Button;

import javafx.scene.control.TextField;

import javafx.stage.Stage;

import java.io.IOException;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

public class AdminLoginController {

@FXML

private TextField adminUsername;

@FXML

private TextField adminPassword;

@FXML

private Button adminlogin;

public String aUsername, aPassword;

public void adminLogin(ActionEvent actionEvent) throws IOException {

aUsername = adminUsername.getText();

aPassword = adminPassword.getText();

if (CheckLoginUser(aUsername, aPassword)) { // sending data to databasehandler class to connection data

adminlogin.getScene().getWindow().hide();

Stage dashboardStage = new Stage();

dashboardStage.setTitle("Admin Controller");

Parent root = FXMLLoader.load(getClass().getResource("adminControls.fxml"));

Scene scene = new Scene(root);

dashboardStage.setScene(scene);

dashboardStage.show();

}

else {

Alert alert = new Alert(Alert.AlertType.INFORMATION);

alert.setTitle("Error");

alert.setContentText("Username or Password Incorrect!!");

alert.show();

// reseting user and pass field

adminUsername.setText("");

adminPassword.setText("");

}

}

private boolean CheckLoginUser(String aUsername, String aPassword) {

Connection connection = SIS.DatabaseController.GetDatabaseConnection();

// String checkQuery = "select \* from loginDetails where user = '"+uname+"' and pass = '"+pass+"' ";

String checkQuery = "select \*from adminDetails where adminId = ? and adminPassword = ? "; // i don't use id from database table.

PreparedStatement preparedStatement = null;

boolean status = false; //initially false

try {

preparedStatement = connection.prepareStatement(checkQuery);

preparedStatement.setString(1, aUsername); // dynamically sending username

preparedStatement.setString(2, aPassword); // sending password to checkquery statement

ResultSet resultSet = preparedStatement.executeQuery();

// while (resultSet.next()) {

// return status;

// }

status = resultSet.next();

preparedStatement.close();

return status;

} catch (SQLException e) {

// e.getLocalizedMessage();

e.printStackTrace();

}

return status;

}

}

**InsertQuery.java**

package Admin;

import SIS.DatabaseController;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.\*;

import javafx.stage.Stage;

import java.io.IOException;

import java.sql.\*;

import java.time.LocalDate;

public class InsertQuery {

@FXML

TextField firstName;

@FXML

TextField middleName;

@FXML

TextField lastName;

@FXML

TextField stuID;

@FXML

TextField password;

@FXML

TextField email;

@FXML

ComboBox<String> department;

@FXML

ComboBox<String> year;

@FXML

RadioButton male;

@FXML

RadioButton female;

@FXML

DatePicker dob;

@FXML

TextField mobileNo;

@FXML

Button back;

String sex,id;

LocalDate ld;

String regex = "^(.+)@(.+)$";

public void insertQuery(ActionEvent actionEvent) {

if(male.isSelected()) {

sex = "male";

}

else if(female.isSelected()) {

sex ="female";

}

else

sex="";

ld = dob.getValue();

// if(String.valueOf(ld).contains(null)) {

//

// }

// else {

System.out.println(ld);

id = stuID.getText();

System.out.println(id);

if(ld == null || id.equals("") || sex.equals("") || firstName.getText().equals("") ||middleName.getText().equals("")||

lastName.getText().equals("") || email.getText().equals("") || mobileNo.getText().equals("")||

department.getValue() == null || year.getValue() == null) {

alert("Please fill all the fields correctly");

}

if(!email.getText().matches(regex)) {

alert("Enter email correctly");

}

else {

Connection connection = DatabaseController.GetDatabaseConnection();

String checkQuery = "insert into studentDetails(firstName,middleName,lastName,id,password,gender,DOB,department,year,mobileNo,email) values (?,?,?,?,?,?,?,?,?,?,?)";

PreparedStatement preparedStatement = null;

try {

preparedStatement = connection.prepareStatement(checkQuery);

preparedStatement.setString(1, firstName.getText().toUpperCase());

preparedStatement.setString(2, middleName.getText().toUpperCase());

preparedStatement.setString(3, lastName.getText().toUpperCase());

preparedStatement.setString(4, stuID.getText().toUpperCase());

preparedStatement.setString(5, password.getText().toLowerCase());

preparedStatement.setString(6, sex.toUpperCase());

preparedStatement.setString(7, String.valueOf(ld).toUpperCase());

preparedStatement.setString(8, department.getValue().toUpperCase());

preparedStatement.setString(9, year.getValue().toUpperCase());

preparedStatement.setString(10, mobileNo.getText().toUpperCase());

preparedStatement.setString(11, email.getText().toLowerCase());

int resultSet = preparedStatement.executeUpdate();

} catch (SQLException e) {

e.printStackTrace();

}

String checkQuery1 = "insert into studentLogin(id,password) values (?,?)";

PreparedStatement preparedStatement1 = null;

try {

preparedStatement1 = connection.prepareStatement(checkQuery1);

preparedStatement1.setString(1, stuID.getText());

preparedStatement1.setString(2, password.getText());

int resultSet1 = preparedStatement1.executeUpdate();

} catch (SQLException e) {

e.printStackTrace();

}

String checkQuery2 = "insert into stuMarks(id,subject1,subject2,subject3,subject4) values (?,?,?,?,?)";

PreparedStatement preparedStatement2 = null;

try {

preparedStatement2 = connection.prepareStatement(checkQuery2);

preparedStatement2.setString(1, stuID.getText());

preparedStatement2.setString(2, "0");

preparedStatement2.setString(3, "0");

preparedStatement2.setString(4, "0");

preparedStatement2.setString(5, "0");

int resultSet2 = preparedStatement2.executeUpdate();

} catch (SQLException e) {

e.printStackTrace();

}

String checkQuery3 = "insert into stuSubAttendanceDetails(id,subject1,subject2,subject3,subject4) values (?,?,?,?,?)";

PreparedStatement preparedStatement3 = null;

try {

preparedStatement3 = connection.prepareStatement(checkQuery3);

preparedStatement3.setString(1, stuID.getText());

preparedStatement3.setString(2, "0");

preparedStatement3.setString(3, "0");

preparedStatement3.setString(4, "0");

preparedStatement3.setString(5, "0");

int resultSet3 = preparedStatement3.executeUpdate();

} catch (SQLException e) {

e.printStackTrace();

}

firstName.setText("");

firstName.setPromptText("firstName");

middleName.setText("");

middleName.setPromptText("middleName");

lastName.setText("");

lastName.setPromptText("lastName");

stuID.setText("");

stuID.setPromptText("stuID");

password.setText("");

password.setPromptText("password");

mobileNo.setText("");

mobileNo.setPromptText("98xxxxxxx");

email.setText("");

email.setPromptText("abc@pqr.com");

department.setValue("select department");

year.setValue("select year");

dob.disableProperty().setValue(null);

if(male.isSelected()) {

male.selectedProperty().setValue(false);

}

else

female.selectedProperty().setValue(false);

}

}

public void back(ActionEvent actionEvent) throws IOException {

back.getScene().getWindow().hide();

Stage dashboardStage = new Stage();

Parent root = FXMLLoader.load(getClass().getResource("adminControls.fxml"));

dashboardStage.setTitle("Student Details");

Scene scene = new Scene(root);

dashboardStage.setScene(scene);

dashboardStage.show();

}

public void alert(String msg) {

Alert alert = new Alert(Alert.AlertType.INFORMATION);

alert.setTitle("Error");

alert.setContentText(msg);

alert.show();

}

}

**DeleteStudentData.java**

package Admin;

import SIS.DatabaseController;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.\*;

import javafx.scene.input.MouseEvent;

import javafx.stage.Stage;

import java.io.IOException;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.Optional;

public class DeleteStudentData extends AdminLoginController {

@FXML

private Button dropButton;

@FXML

private TextField stuIdforDrop;

@FXML

private PasswordField AdminPassword;

String APass;

String AID;

public void dropEntry(ActionEvent actionEvent) {

Connection connection = SIS.DatabaseController.GetDatabaseConnection();

// String checkQuery = "select \* from loginDetails where user = '"+uname+"' and pass = '"+pass+"' ";

String checkQuery = "delete from studentDetails where id = ? "; // i don't use id from database table.

PreparedStatement preparedStatement = null;

try {

preparedStatement = connection.prepareStatement(checkQuery);

preparedStatement.setString(1, stuIdforDrop.getText()); // dynamically sending username

Alert alert = new Alert(Alert.AlertType.CONFIRMATION);

alert.setTitle("Confirmation Dialog");

alert.setHeaderText("WARNING");

alert.setContentText("Do you really want to delete the data??");

Optional<ButtonType> result = alert.showAndWait();

if (result.get() == ButtonType.OK) {

// TextInputDialog dialog = new TextInputDialog("");

// dialog.setTitle("Text Input Dialog");

// dialog.setHeaderText("Look, a Text Input Dialog");

// dialog.setContentText("Please enter your name:");

// Optional<String> result5 = dialog.showAndWait();

// if (result5.isPresent()) {

// System.out.println("Your name: " + result5.get());

// }

preparedStatement.executeUpdate();

preparedStatement.close();

dropButton.getScene().getWindow().hide();

Stage dashboardStage = new Stage();

Parent root = FXMLLoader.load(getClass().getResource("AdminLogin.fxml"));

dashboardStage.setTitle("Student Information System - Admin Panel");

Scene scene = new Scene(root);

dashboardStage.setScene(scene);

dashboardStage.show();

} else {

}

} catch (SQLException | IOException e) {

// e.getLocalizedMessage();

e.printStackTrace();

}

}

private void alert() {

}

}

**UpdateStudentInfo.java**

package Admin;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.\*;

import javafx.stage.Stage;

import java.io.IOException;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.time.LocalDate;

public class UpdateStudentInfo {

@FXML

private TextField updateStudentid;

@FXML

private TextField UpdatedfirstName;

@FXML

private TextField UpdatedmiddleName;

@FXML

private TextField UpdatedlastName;

@FXML

private TextField Updatedemail;

@FXML

private TextField UpdatedmobileNo;

@FXML

private DatePicker UpdatedDOB;

@FXML

private CheckBox fname;

@FXML

private CheckBox mname;

@FXML

private CheckBox lname;

@FXML

private CheckBox dob;

@FXML

private CheckBox mnumber;

@FXML

private CheckBox email;

@FXML

private Button updateInfo;

String stuID,UfirstName, UlastName, UmiddleName, Uemail, UmNumber, Udob;

String regex = "^(.+)@(.+)$";

boolean firstname, lastname, mobileNumber, DOB, Email, middlename;

LocalDate ld;

public void UpdateDetails(ActionEvent actionEvent) {

stuID = updateStudentid.getText();

Connection connection = SIS.DatabaseController.GetDatabaseConnection();

String checkQuery = "select id from studentDetails where id = ?"; // i don't use id from database table.

PreparedStatement preparedStatement = null;

boolean status = false; //initially false

try{

preparedStatement = connection.prepareStatement(checkQuery);

preparedStatement.setString(1, stuID); // dynamically sending username

ResultSet resultSet = preparedStatement.executeQuery();

while(resultSet.next()) {

String result = resultSet.getString(1);

System.out.println(result);

System.out.println(stuID);

if(result.equals(stuID)) {

if(fname.isSelected()) {

firstname = true;

UfirstName = UpdatedfirstName.getText();

String checkQuery1 = "update studentDetails set firstName = ? where id = ?";

PreparedStatement preparedStatement1 = null;

try {

preparedStatement1 = connection.prepareStatement(checkQuery1);

preparedStatement1.setString(1, UfirstName.toUpperCase()); // dynamically sending username

preparedStatement1.setString(2, stuID); // dynamically sending username

preparedStatement1.executeUpdate();

} catch (SQLException e) {

e.printStackTrace();

}

}

if (mname.isSelected()) {

middlename = true;

UmiddleName = UpdatedmiddleName.getText();

String checkQuery2 = "update studentDetails set middleName = ? where id = ?";

PreparedStatement preparedStatement2 = null;

try {

preparedStatement2 = connection.prepareStatement(checkQuery2);

preparedStatement2.setString(1, UmiddleName.toUpperCase()); // dynamically sending username

preparedStatement2.setString(2, stuID); // dynamically sending username

preparedStatement2.executeUpdate();

} catch (SQLException e) {

e.printStackTrace();

}

}

if(lname.isSelected()) {

lastname = true;

UlastName = UpdatedlastName.getText();

String checkQuery3 = "update studentDetails set lastName = ? where id = ?";

PreparedStatement preparedStatement3 = null;

try {

preparedStatement3 = connection.prepareStatement(checkQuery3);

preparedStatement3.setString(1, UlastName.toUpperCase()); // dynamically sending username

preparedStatement3.setString(2, stuID); // dynamically sending username

preparedStatement3.executeUpdate();

} catch (SQLException e) {

e.printStackTrace();

}

}

if(dob.isSelected()) {

DOB = true;

ld = UpdatedDOB.getValue();

String checkQuery4 = "update studentDetails set DOB = ? where id = ?";

PreparedStatement preparedStatement4 = null;

try {

preparedStatement4 = connection.prepareStatement(checkQuery4);

preparedStatement4.setString(1, String.valueOf(ld).toUpperCase()); // dynamically sending username

preparedStatement4.setString(2, stuID); // dynamically sending username

preparedStatement4.executeUpdate();

} catch (SQLException e) {

e.printStackTrace();

}

}

if(mnumber.isSelected()) {

mobileNumber = true;

UmNumber = UpdatedmobileNo.getText();

String checkQuery5 = "update studentDetails set mobileNo = ? where id = ?";

PreparedStatement preparedStatement5 = null;

try {

preparedStatement5 = connection.prepareStatement(checkQuery5);

preparedStatement5.setString(1, UmNumber); // dynamically sending username

preparedStatement5.setString(2, stuID); // dynamically sending username

preparedStatement5.executeUpdate();

} catch (SQLException e) {

e.printStackTrace();

}

}

if(email.isSelected()) {

if (Updatedemail.getText().matches(regex)) {

Email = true;

Uemail = Updatedemail.getText();

String checkQuery6 = "update studentDetails set email = ? where id = ?";

PreparedStatement preparedStatement6 = null;

try {

preparedStatement6 = connection.prepareStatement(checkQuery6);

preparedStatement6.setString(1, Uemail.toLowerCase()); // dynamically sending username

preparedStatement6.setString(2, stuID); // dynamically sending username

preparedStatement6.executeUpdate();

} catch (SQLException e) {

e.printStackTrace();

}

}

else {

Alert alert = new Alert(Alert.AlertType.INFORMATION);

alert.setTitle("Error");

alert.setContentText("Enter email correctly");

alert.show();

}

}

}

else {

Alert alert = new Alert(Alert.AlertType.INFORMATION);

alert.setTitle("Error");

alert.setContentText("StudentId incorrect");

alert.show();

// reseting user and pass field

updateStudentid.setText("");

}

}

} catch (SQLException e) {

e.printStackTrace();

}

}

}