Object Oriented System Design

Student Record Management System

Software Requirements Specification

15 November 2022

Prepared for

OOSD

Instructor: Prof.Sanjeev Patel

Table of Contents

Prepared by:

- Devraj Changkakati (120CS0143)
- Anirudh Jayakumar (120CS0161)
- Pramod Kumar Mallik (120CS0180)
- Tapaswini Mishra(120CS0186)
- Sabitri Mohapatra(120CS0188)
- B. Naga Pravallika (120CS0121)
- K. Srujana Kavya (120CS0169)
- L. Hema (120CS0158)
- Swayam Kumar Nahak (120CS0193)

Revision History

Name	Date	Reason For Changes	Version
Version 1.0	15/11/22	Implemented the first code for the Student Record Management	1.0

1. Introduction

A Student Record Management System (SRMS) is an integrated software system that is designed to help colleges for management of students and manage the records of students regarding admission and examination.Extensive information is available at your fingertips through this System. Viewing student data, managing admission and reshuffling, managing seats, quota, board, semester, faculty, category and for examination, block allocation, subject management, scheduling exam, result and related issues are made simple and easy. There are custom search capabilities to aid in finding student information and working on student records. This can make the system easier to navigate and to use maximizing the effectiveness of time and other resources. SRMS allows the keeping of personnel data in a form that can be easily accessed and analyzed in a consistent way.

1.1 Problem Statement

The project is about handling all the information of the student regarding admission and examination. Also it manages resources which were managed and handled by manpower previously. The main purpose of the project is to integrate distinct sections of the organization into a consistent manner so that complex functions can be handled smoothly by any technical or non-technical person. To design an efficient system to track, maintain and manage all the data generated by a college including the grades of a student, their attendance, their activities, records etc. Maintaining the student records manually is a tedious task which may also lead to unwanted errors. Moreover, using different softwares for different tasks would be too tiresome and difficult to implement. Hence, a unified management system is required where all necessary records and changes can be updated in an errorless and efficient way.

The project aims at the following matters:

- Automation of admission and enrolment as per board, quota, category and available seats.
- Assistance in decision-making.
- To manage information about students, faculty and courses.
- Consistently update information of all the students.

1.2 Scope

The scope of the project includes the following:

- Any college can use this system as it is not client centric.
- All admission and examination related work for the student can be done using this system.
- Deliver Electronic Workplace
- Application Support & Maintenance after deployment to production
- The Admin Module can be reused for projects as well which have many users with different rights. Hence it is reusable.

1.3 Definitions, Acronyms, and Abbreviations

Definitions:

The student management system is an automated version of manual Student Management System. It can handle all details about a student. The details include college details, subject details, student personnel details, academic details, exam details etc.

Our system has two types of accessing modes, administrator and user. Student management system is managed by an administrator. It is the job of the administrator to insert, update and monitor the whole process. When a user logs in to the system. He would only view details of the student. He can't perform any changes

Acronyms:

SRMS: Student Record Management System

LM: Login Module

RUM: Registered Users Module

NUM: Normal Users Module

AM: Administrator Module

SM: Server Module

DB: Database

1.4 References

[1] http://www.slideshare.net/

[2] http://www.sourcecodesolutions.in/

[3] http://www.google.com/

[4] https://www.geeksforgeeks.org/unified-modeling-language-uml-introduction/

[5] https://www.academia.edu/28742043/Students_Record_Management_System_Project_By_Soita_Reuben

1.5 Overview

Student Record Management System (SRMS) is a web-based application that tracks current student's academic information. It maintains academic information for ready access by office staff, students, their faculty advisors, and committee members. Instead of tedious paperwork, students will be able to submit required information electronically, and the departments will be able to evaluate the submissions with a much quicker turnaround.

The Student Management System has been modularized into following modules:

LOGIN MODULE:

The purpose of this module is to provide entry to the system or website. Based on the type of login, the user is provided with various facilities and functionalities. The main function of this module is to allow the user to use SRMS. This module provides two types of login - Admin login and Student login.

ADMINISTRATOR MODULE:

In this module when the administrator will enter his/her username and password, then he/she will enter into the administrator page and this page consists of two following sub modules.The main purpose of the Admin Module is to introduce new things and configure important aspects. For e.g. only admin is authorized to introduce quota, board, subject, category, etc. and only admin is allowed to configure exams and set fee structure. So the master screens for all these are visible to only admin roles. This is done by the Admin Module. It also can create the users and Physical and Logical Locations. Thus the main purpose of the Admin Module is to manage the dynamic working of the system.

Student Addition/ Updation / Deletion: In SRMS, each Student is added, updated or deleted according to its branch.

Notice/Attendance/Result Generation: In SRMS, information about notice, attendance and Internal result is generated.

Fee Detail and Schedules: Fee information detail and schedule detail are managed.

STUDENT MODULE:

In this module, when a user enters his/her student id and password, then he/she can visit all the following pages.

Profile View: When the student clicks on this link he/she will get his/her information like student id, student name, password, father's name, date of birth, nationality, city, address, country, phone number, mobile number, email. If he/she wants then he/she can change the profile.

Notice View: When the student clicks on this link, he can see the latest notices released by the administrator.

Attendance View: When the student clicks on this one, the student can get his overall attendance percentage (present and absent).
Here one can check his/her attendance record and apply for official leave also.

Internal Results View:

1.When the student clicks on this, he/she will get the internals result in all the subjects. How much grade point he/she secured out of 20 he/she can know.
2.He/she can also upload the grade card .
3.Get previous year questions .
4.Apply for mark change if there is any query.

Fee Detail View: When the student clicks this link he/she can get all the fee structure semester wise and annual fee.

2. General Description

There are many departments of administration for the maintenance of college information and student databases in any institution. All these departments provide various records regarding students. Most of these track records need to maintain information about the students. This information could be the general details like student name, address, performance, attendance etc. or specific information related to departments like collection of data. All the modules in college administration are interdependent. They are maintained manually. So they need to be automated and centralized as, Information from one module will be needed by other modules.

For example, when a student needs his course completion certificate it needs to check many details about the student like his name, reg. number, year of study, exams he attended and many other details. So it needs to contact all the modules that are once, department and examination and the result of students. With that in mind, we overhauled the existing Student DatabaseManagement System and made necessary improvements to streamline the processes.

Administrators using the system will find that the process of recording and retrieving students information and managing their classes, including marking of attendance, is now a breeze. In general, this project aims to enhance efficiency and at the same time maintain information accurateness. Later in this report, features and improvements that allow achievement to this goal will be demonstrated and highlighted.

2.1 Product Perspective

The various system tools that have been used in developing both the front end, back end and other tools of the project are being discussed in this section.

FRONT END:

JSP, HTML, CSS, JAVA SCRIPTS are utilized to implement the frontend.

Java Server Page (JSP):

Different pages in the applications are designed using JSP. A java server page component is a type of java server that is designed to fulfill the role of a user interface for a java web application. Web developers write JSPs as text files that combine HTML or XHTML code, XML elements, and embedded JSP actions and commands. Using JSP, one can collect input from users through a web page.

HTML (HyperText Mark-up Language):

HTML is a syntax used to format a text document on the web.

CSS (Cascading Style Sheets):

CSS is a style sheet language used for describing the look and formatting of a document written in a mark-up language.

Java Script:

JS is a dynamic computer programming language. It is most commonly used as part of web browsers, whose implementations allow client-side scripts to interact with the user, control the browser, communicate asynchronously, and alter the document content that is displayed .

BACK END:

The back end is implemented using PHP which is used to design the databases.

PHP:

It is a programming language used by developers to create dynamic content that interacts with databases and is used to create applications that executes on servers and generates dynamic pages.

2.2 Product Functions

The primary function of the Student Management System web server is essentially to save the whole system information sequentially into the database server. The administration department will have access to the whole system environment and that can be modified as per their needs. The architecture of the whole system is made easy so that any person can login to the system and use the functions. The system database is only accessible to admin and admin can only modify the changes.

2.3 User Characteristics

The user profiles identified to have interaction with the Student Management System that anyone can register and login into the system and use the required resources. The students can easily fill up the registration form online and submit it. And the admin will check the details that the student is eligible as per the admission criteria. After the student will successfully register, he can use college/school system environments as per their limits decided by admin.

2.4 General Constraints

The Student Management System can be accessed successfully by any client location and it’s not necessary that every registration is genuine, so there are chances of fake registration that may reflect some errors. So, the system is designed in such a way that the database will keep updated by the administrator and there are better security options available on the server that can prevent fake IP addresses from accessing the system.

3. Specific Requirements

3.1 External Interface Requirements

For effective use of the system, it is important that users are fully involved and are given opportunities to participate as much as possible. This rectifies numerous problems associated with change management, users getting accustomed to using new ways of doing things as opposed to the traditional system of student records management system. During data collection, the researcher investigated and found out how the current system operates, not only that but also tried out which problems are faced and how best they can be settled. The users described some of the basic requirements of the system as;

- Search for Student
- Update, student records
- View all types of reports (Students attendance, Address book, and Phone list, Student by room, Students by level, Medical information.
- Generates students transcripts and certificates Assign access rights and privileges to the system users.

3.1.1 Hardware Interfaces

- Computer with Intel Core i5 or AMD RYZEN 5000 series processor or better
- 8GB RAM or 16GB RAM
- Windows (64 bit)
- Mouse
- Keyboard
- Windows compatible printer (laser printer recommended)
- Gigabit Ethernet port and router (for networking computers)
- Wired, high speed internet connection (only required for certain software/service options

3.1.2 Software Interfaces

All the interfaces will be ASPX pages running within the internet browser. The SRMS must integrate with the DB through SQL Interface. The system will be hosted in a web server running on Windows Server 2005.

3.2 Functional Requirements

3.2.1 Log in Module (LM)

Users (admin, student and teachers) shall be able to load the Login Module in the internet browser. The LM shall support the user to log into the system. The login panel shall contain fields to contain a user name and a field for password. The password field shall be masked with symbols when the user types. It shall also contain a button labeled as Login. When the user clicks on the Login button the username and password will be verified by the database administrator and then only the user will be able to use the system functions.

3.2.2 Registered Users Module (RUM)

After successful login, users shall be able to continue navigating through the website and view school/college detailed information. After successful login, users (admin, student and teachers) shall be able to update and maintain their profile, such as changing password and personal details.

3.2.3 Normal Users Module (NUM)

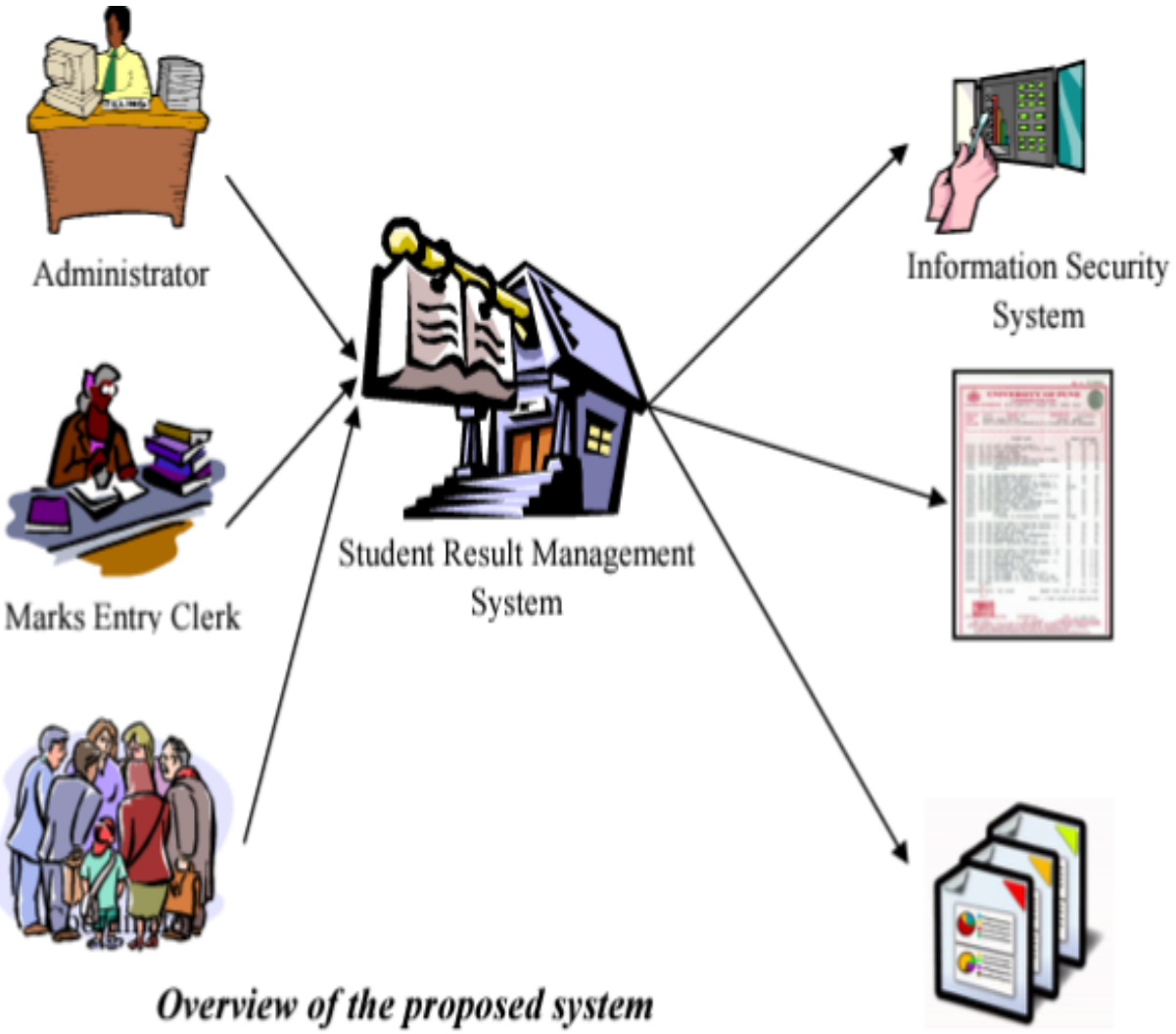
Users who visit SRMS but have not registered, are able to navigate through the website. Users shall be able to view currently held events & upcoming institute schedules. Users shall be able to view school/college timings and their faculties information. Users are able to register themselves as registered users, by clicking on the register now button.

3.2.4 Administrator Module (AM)

After successful login, the system shall display administrative functions. Administrative functions shown shall be added and updated. When an administrator clicks on the add button, the system shall display a section where the administrator can add new student details, remove unused student details and many more. When an administrator clicks on the update button, the system shall display a section where the administrator can update student details and schedule of lectures which are currently stored in the database. When an administrator adds, updates or deletes an entry, the AM module will send the request to the Server Module which will do the necessary changes to the DB.

3.2.5 Server Module (SM)

SM shall be between the various modules and the DB. SM shall receive all requests and format the pages accordingly to be displayed. SM shall validate and execute all requests from the other modules.



Overview of the proposed system

3.3 Non-functional Requirements

Non-functional requirements may exist for the following attributes. Often these requirements must be achieved at a system wide level rather than at a unit level. State the requirements in the following sections in measurable terms (e.g., 95% of transactions shall be processed in less than a second, system downtime may not exceed 1 minute per day, > 30 day MTBF value, etc.)

3.3.1 Performance

The Student Management System shall be built upon the web development technique and put on the web server online. The system and the server must be capable of handling the real-time error functionality that occurs by the defined users. In addition, the system must be safety critical. All failures reported by the server side must be handled instantaneously to allow for user and system safety.

3.3.2 Reliability

The system is safety critical. If it moves out of normal operation mode, the requirement is to drop or down the server and fix it as soon as possible and open it again. This emergency behavior shall not occur without reason.

3.3.3 Availability

When in normal operating conditions, requests by a user for an online system shall be handled within 1 second. Immediate feedback of the system's activities shall be communicated to the user by clearing the system and giving space and speed to their hospitality.

3.3.4 Security

There shall be a strong security mechanism should be placed in the server side of the system to keep unwanted users from hacking or damaging the system. However, all users of the system give and store the details of privacy related to personal information and many others. However, our system can be accessed online so we need a very secured system as far as security is concerned.

3.3.5 Maintainability

There shall be design documents describing maintenance of the software and database used to save the user details as well as the daily updates and modifications done in the system. There shall be an access on the control system by the admin to maintain it properly at the front end as well as at back end.

3.3.6 Portability

There is a portability requirement as far as our system is concerned because it is an online as well as offline (local server based) system so we can access it from anywhere through the internet connection. And we have to maintain the copy of stored data into our database.

3.4 Inverse Requirements

As far as Inverse Requirement is concerned, our system has the best inverse requirement which is most important according to our system view. That is if a student wants to cancel the registration so the admin can access the student details for cancellation and will refund the fees that was fetched from student and not complete payment will be back which is the trend and rule by every school/college management.

3.5 Logical Database Requirements

A one to many relational database shall be used in order to validate various student requests and details can be mismatched. Moreover, mismatches are to be logged for reference. The database shall be concurrent with the performance requirements of the Student Management System.

3.6 Other Requirements

When the system is completely developed and submitted to the client, few sessions will be required to make the users of the system understand about the functionality of it and some time to adapt to the system. After those sessions, it’s required that a member from the development team should spend some time in the system background for an agreed time period. That time period will be used in identifying new bugs that could not be reached in the earlier phases of the development process. Clients should have a valid e-mail account in order to receive reservation email notifications.

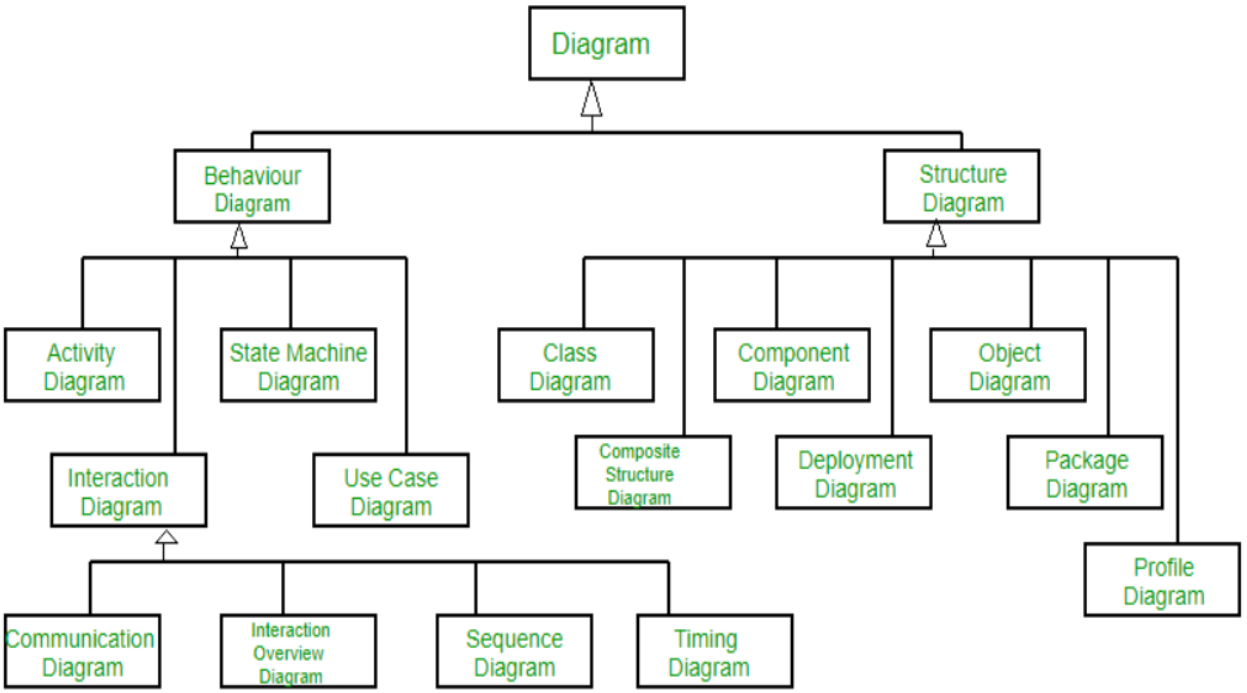
4. UML and Analysis Models

Unified Modeling Language (UML) is a general purpose modeling language. The main aim of UML is to define a standard way to visualize the way a system has been designed. It is quite similar to blueprints used in other fields of engineering.

UML is not a programming language, it is rather a visual language. We use UML diagrams to portray the behavior and structure of a system. UML helps software engineers, businessmen and system architects with modeling, design and analysis. UML is essential to communicate with non programmers, the essential requirements, functionalities and processes of the system.A lot of time is saved down the line when teams are able to visualize processes, user interactions and static structure of the system.

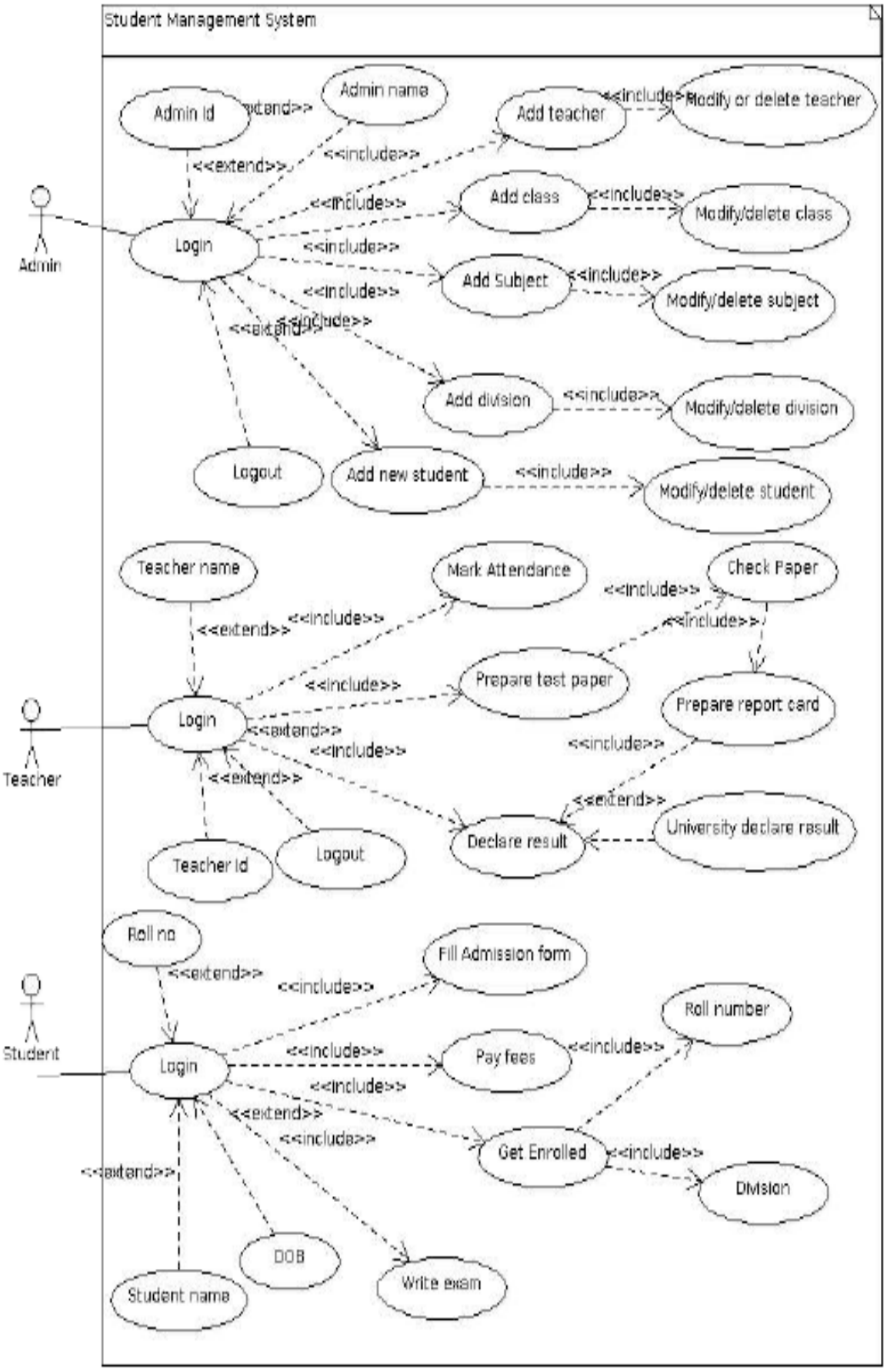
UML is linked with object oriented design and analysis. UML makes the use of elements and forms associations between them to form diagrams. Diagrams in UML can be broadly classified as:

1. **Structural Diagrams** – Capture static aspects or structure of a system. Structural Diagrams include: Object Diagrams and Class Diagrams
2. **Behavior Diagrams** – Capture dynamic aspects or behavior of the system. Behavior diagrams include: Use Case Diagrams, Activity diagrams and State Diagrams



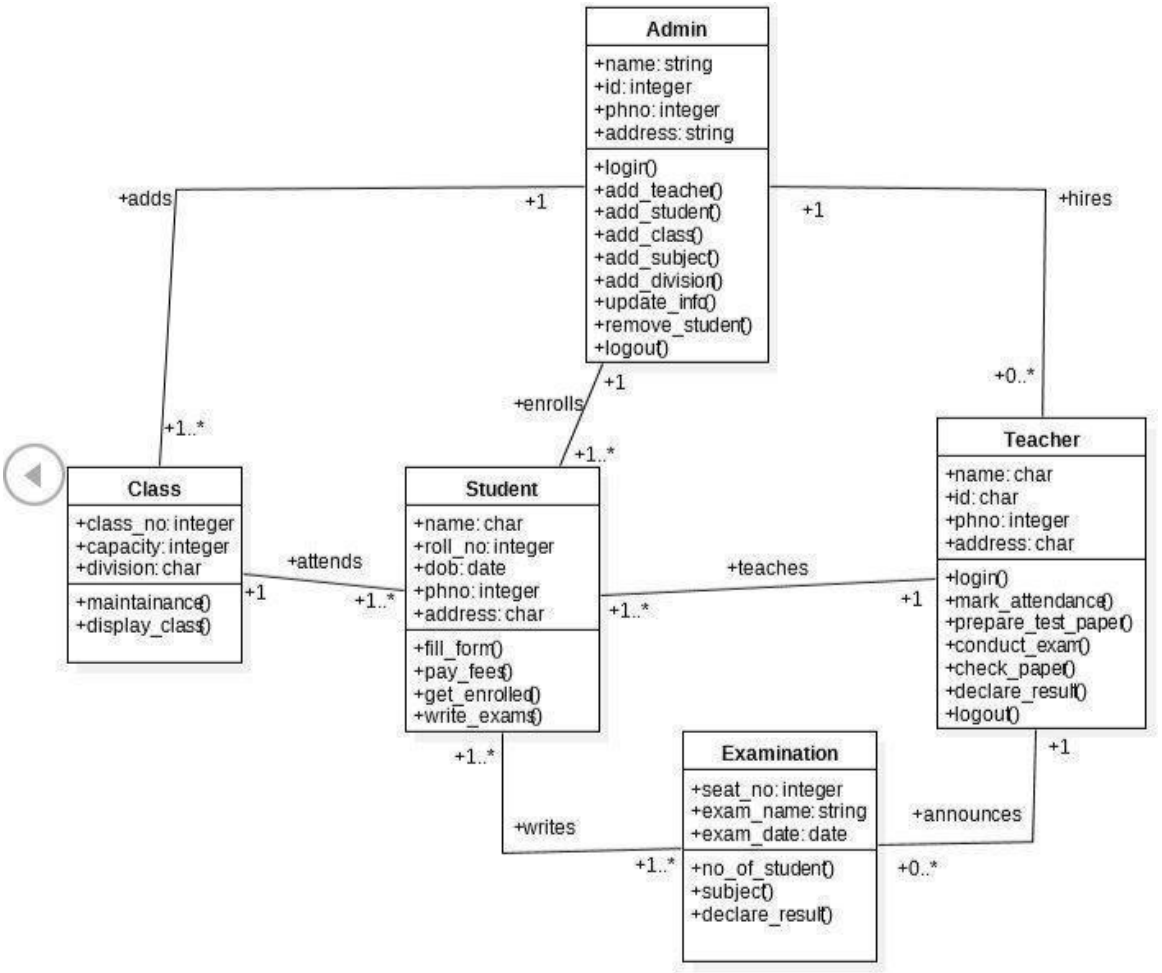
4 .1 Use Case Diagram

Use Case Diagrams are used to depict the functionality of a system or a part of a system. They are widely used to illustrate the functional requirements of the system and its interaction with external agents(actors). A use case is basically a diagram representing different scenarios where the system can be used. A use case diagram gives us a high level view of what the system or a part of the system does without going into implementation details.


Student Management System
Admin
Login
Admin Id
Admin name
<<extend>>
<<include>>
Add teacher
Modify or delete teacher
Add class
Modify/delete class
Add Subject
Modify/delete subject
Add division
Modify/delete division
Logout
Add new student
Modify/delete student
Teacher
Teacher name
Mark Attendance
Check Paper
Prepare test paper
Prepare report card
Declare result
University declare result
Teacher Id
Logout
Student
Roll no
Fill Admission form
Roll number
Pay fees
Get Enrolled
Division
DOB
Student name
Write exam

4.2 Class Diagram

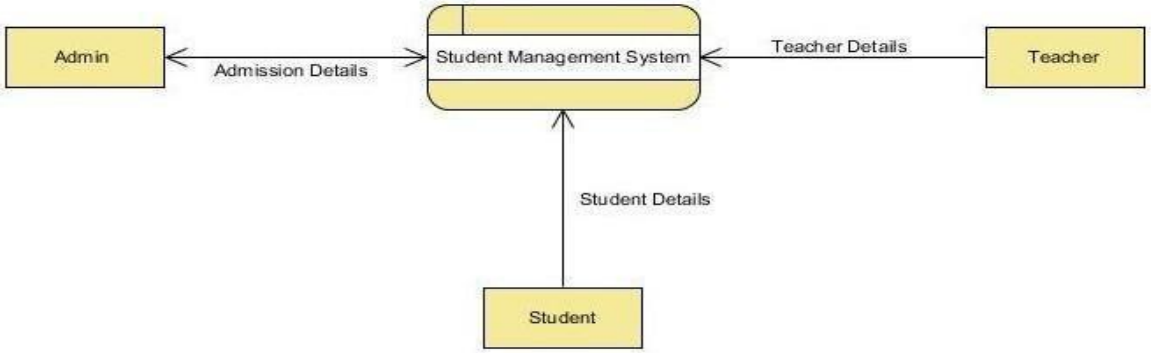
The most widely used UML diagram is the class diagram. It is the building block of all object oriented software systems. We use class diagrams to depict the static structure of a system by showing the system's classes,their methods and attributes. Class diagrams also help us identify relationships between different classes or objects.



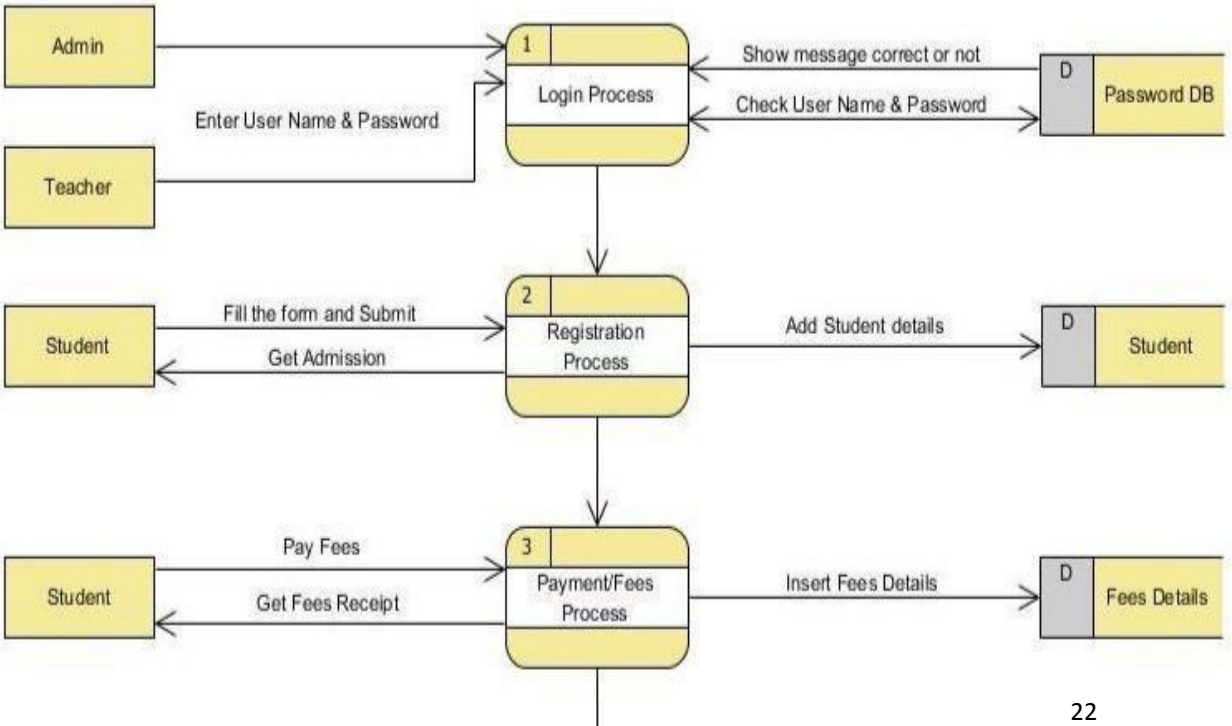
4.3 Data Flow Diagrams (DFD)

A data flow diagram (DFD) maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows, plus short text labels, to show data inputs, outputs, storage points and the routes between each destination. Data flowcharts can range from simple, even hand-drawn process overviews, to in-depth, multi-level DFDs that dig progressively deeper into how the data is handled.

Level 0

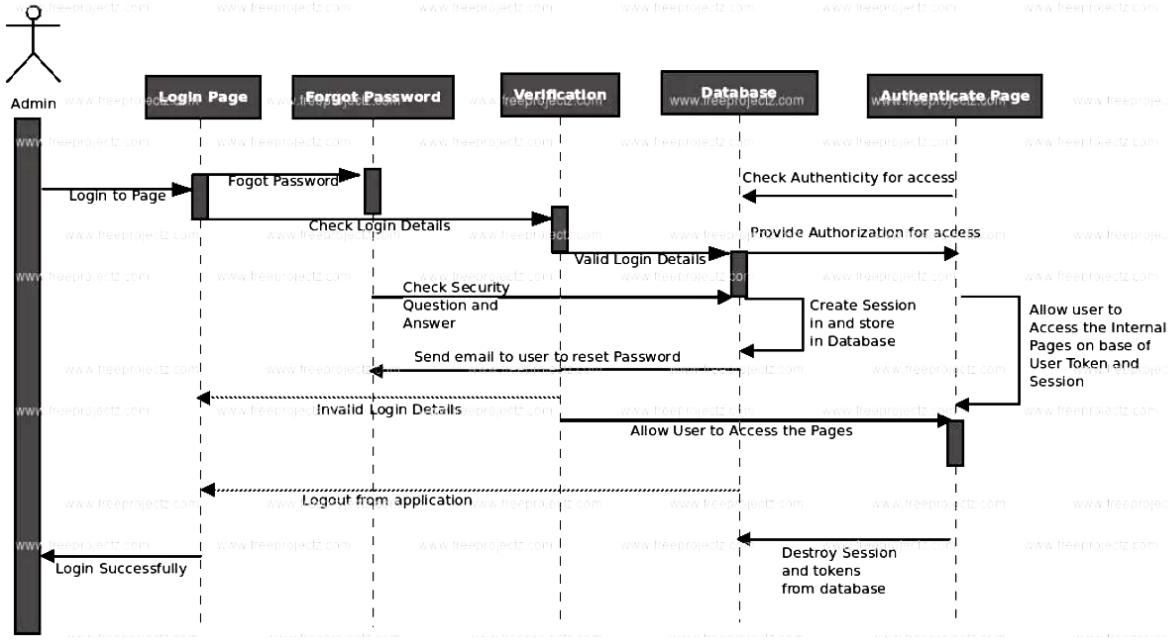


Level 1



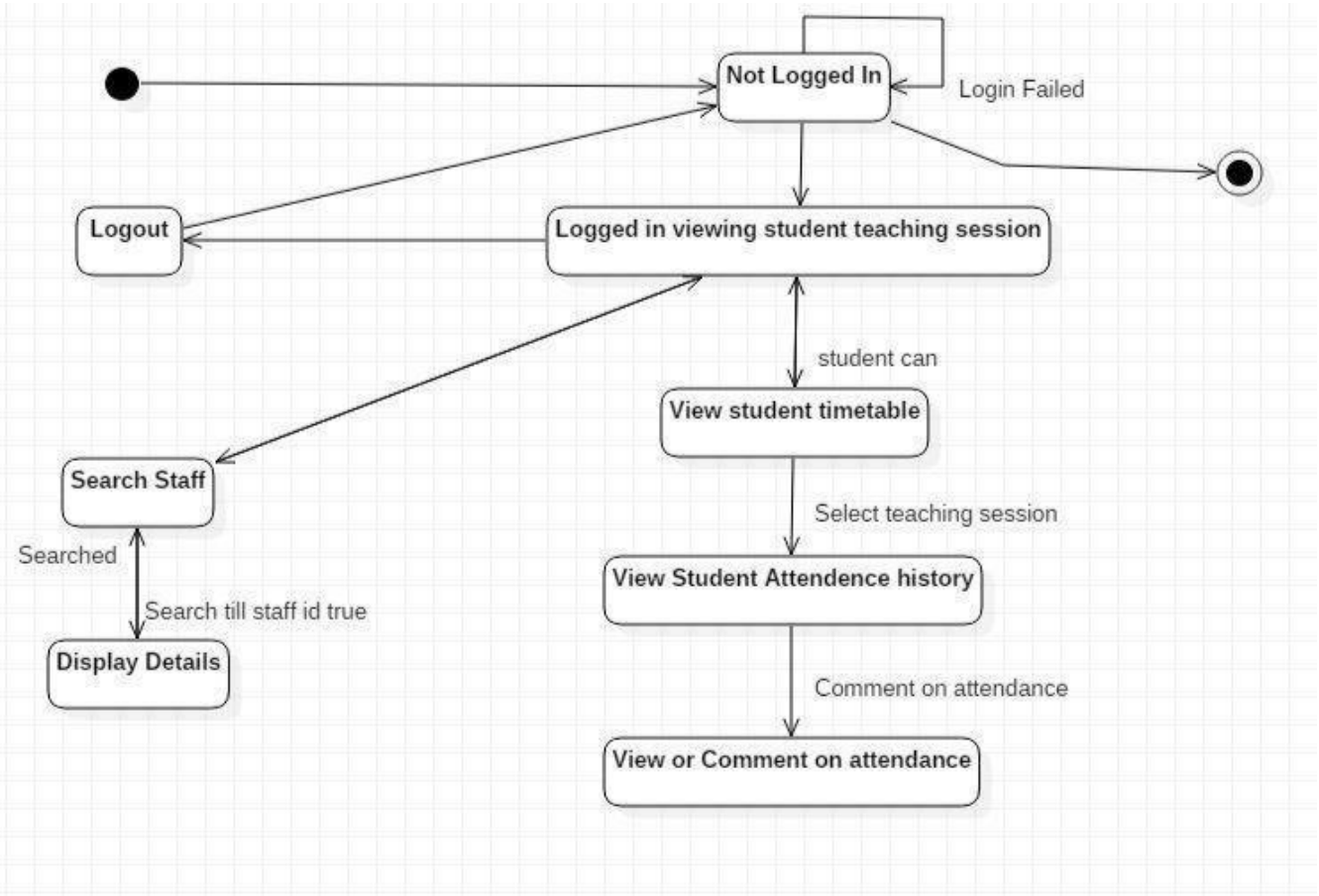
4.4 SEQUENCE DIAGRAM

A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place.We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function. These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems.



4.5 State Machine Diagram

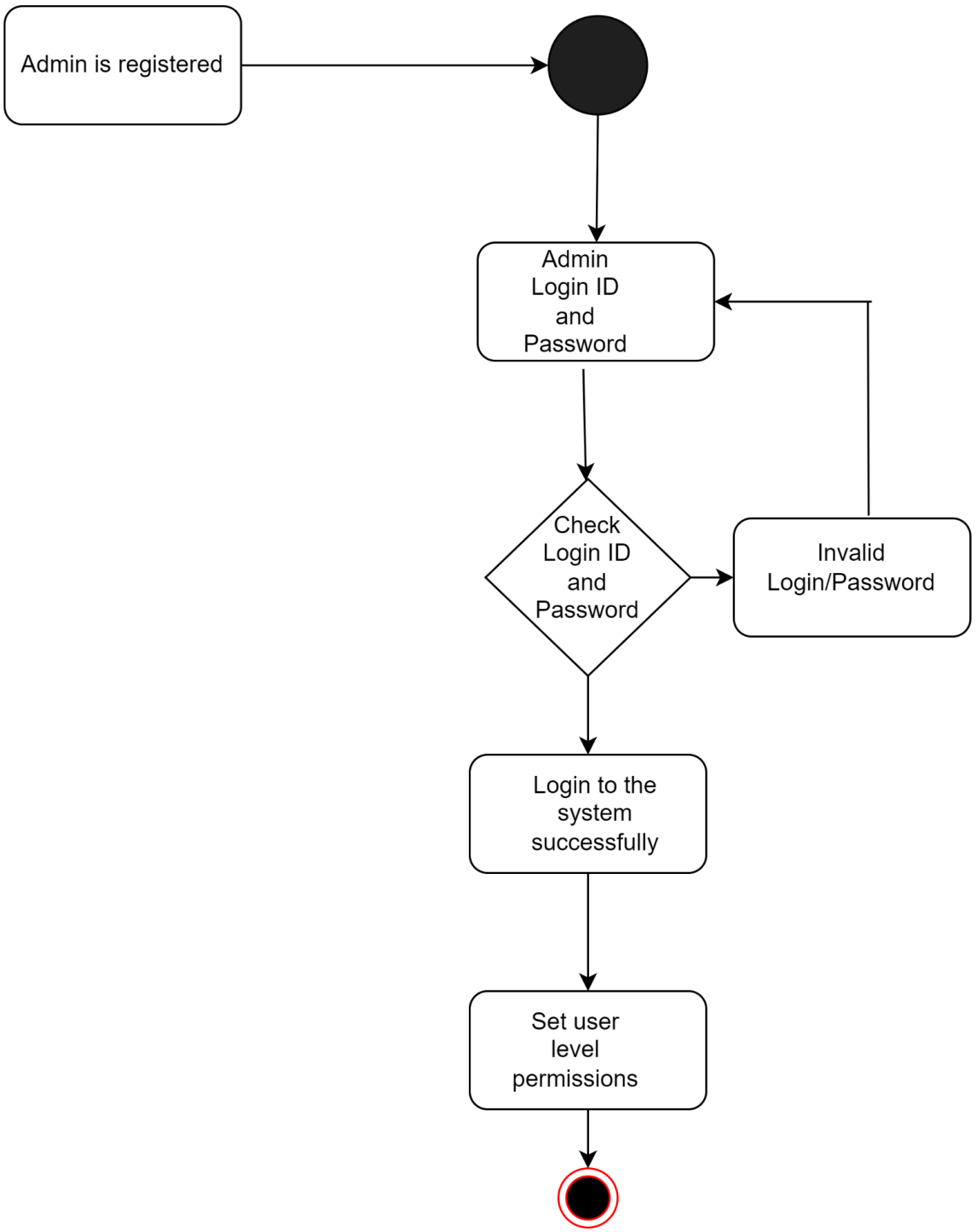
State Machine Diagrams – A state diagram is used to represent the condition of the system or part of the system at finite instances of time. It's a behavioral diagram and it represents the behavior using finite state transitions. State diagrams are also referred to as State machines and State-chart Diagrams . These terms are often used interchangeably.So simply, a state diagram is used to model the dynamic behavior of a class in response to time and changing external stimuli.



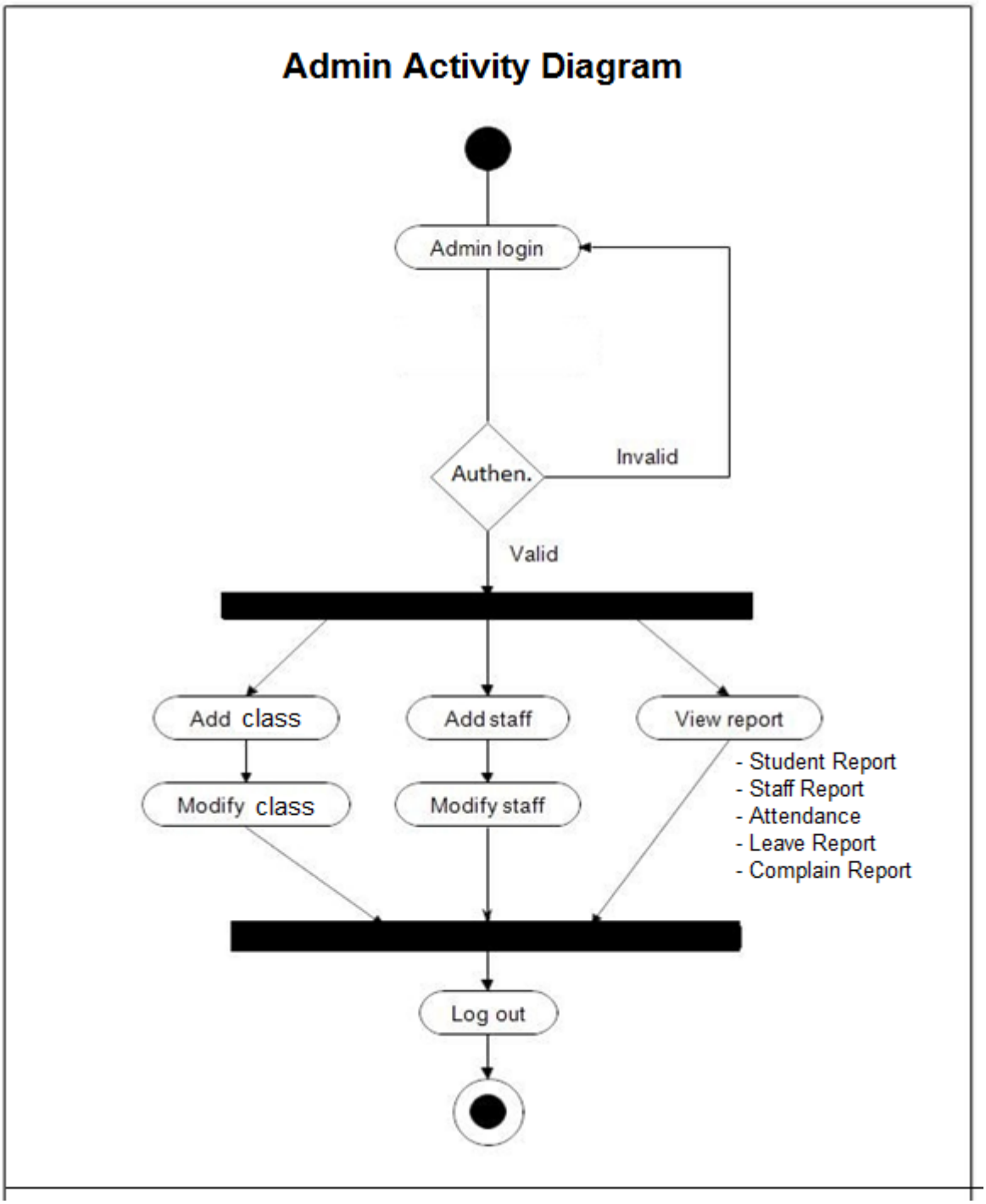
4.6 Activity Diagram

Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. Activity Diagrams describe how activities are coordinated to provide a service which can be at different levels of abstraction.

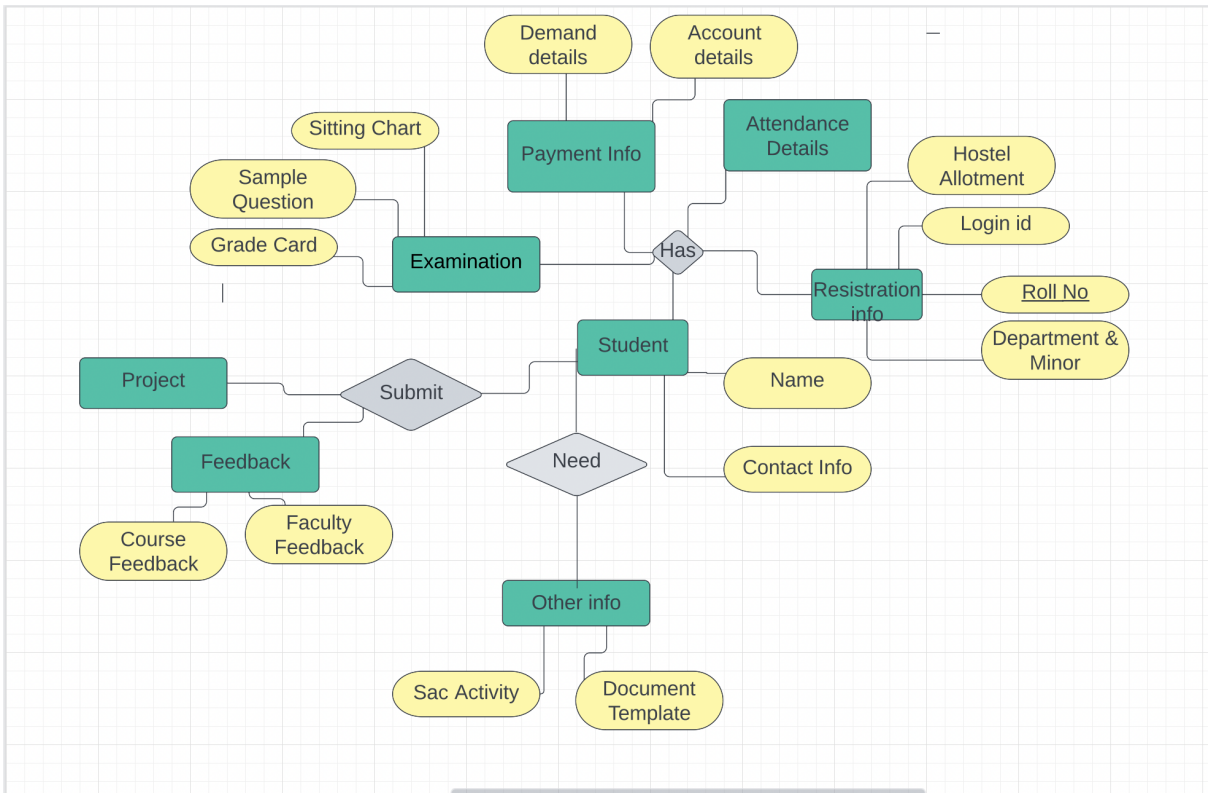
4.6.1. Login Activity Diagram



4.6.2. Admin Activity Diagram



4.7 Entity Relationship Diagram



5 .Code

HTML module

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">
```

```
    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width,
initial-scale=1.0">

     <!--Title-->

    <title>Simple Student Record</title>

    <link rel="stylesheet" href="style.css">

</head>

<body style="background-color: skyblue">

    <CENTER><h2> Student Records</h2></CENTER>

    <!--Table-->

    <table>

        <!--Row-1-->

        <tr>

            <!--Column-1-->

            <td>

                <!--Form-->

                <form
onsubmit="event.preventDefault();onFormSubmit();" autocomplete="off">

                    <h3>Student Form</h3>

                  <!--div-1-->

                    <div>

                        <label>Full Name</label>

                        <!--Validation Error-->

                        <label class="validation-error hide"
id="userNamevalidationError">This field is required</label>

                        <!--Input User Name-->
```

```
                        <input type="text" name="userName"
id="userName" placeholder="Enter the user Name">

                    </div>

                    <!--div-2-->

                    <div>

                        <label>Rollno</label>

                        <!--Validation Error-->

                        <label class="validation-error hide"
id="rollNovalidationError" >This field is required</label>

                        <!--Input Roll No-->

                        <input type="text" name="rollNo" id="rollNo"
placeholder="Enter the Roll number">

                    </div>

                    <!--div-3-->

                    <div>

                        <label>Student Branch</label>

                        <!--Validation Error-->

                        <label class="validation-error hide"
id="stdClassvalidationError">This field is required</label>

                       <!--Input Student Class-->

                        <input type="text" name="stdClass"
id="stdClass" placeholder="Enter the Student Branch">

                    </div>

                    <!--div-4-->

                    <div>

                        <label>Fill Admission form</label>

                        <!--Validation Error-->
```

```
                        <label class="validation-error hide"
id="tsubvalidationError">This field is required</label>

                        <!--Input Total Subject-->

                        <input type="number" name="tsub" id="tsub"
placeholder="Write your Admission Number">

                    </div>

                    <!--div-5-->

                    <div>

                        <label>Age</label>

                        <!--Validation Error-->

                        <label class="validation-error hide"
id="agevalidationError">This field is required</label>

                        <!--Input Age-->

                        <input type="number" name="age" id="age"
placeholder="Enter the Age">

                    </div>

                    <div>

                        <label>Fees Amount</label>

                        <!--Validation Error-->

                        <label class="validation-error hide"
id="agevalidationError">This field is required</label>

                        <!--Input Age-->

                        <input type="number" name="fee" id="fee"
placeholder="Fees Amount">

                    </div>

                    <div>

                        <label>Enrollement</label>

                        <!--Validation Error-->
```

```
                        <label class="validation-error hide"
id="agevalidationError">This field is required</label>

                        <!--Input Age-->

                        <input type="number" name="eroll" id="eroll"
placeholder="Enrollement Number">



                    </div>

                    <div>

                        <label>Exam</label>

                        <!--Validation Error-->

                        <label class="validation-error hide"
id="agevalidationError">This field is required</label>

                        <!--Input Age-->

                        <input type="number" name="exam" id="exam"
placeholder="Upload your Exam marks">

                    <!--div-6-->

                    <div class="form-action-buttons">

                        <!--Input Button-->

                        <input type="submit" value="Submit">

                    </div>

                </form>

            </td>

            <!--Column-2-->

            <td>

                <!--Nested Table-->

                <table class="list" id="stdlist"
style="background-color: #C2976D;">
```

```
                    <!--Table Head-->
                    <thead>
                        <!--Row-2-->
                        <tr>
                            <th>Full Name</th>
                            <th> Roll no</th>
                            <th>Branch</th>
                            <th>Total Subject</th>
                            <th>Age</th>
                            <th>Fee Due</th>
                            <th>Enrollement Number</th>
                            <th>Exam Marks</th>
                            <th>Action</th>
                        </tr>
                    </thead>
                    <!--Table Body-->
                    <tbody>
                    </tbody>
                </table>
            </td>
        </tr>
    </table>
    <script src="1.js"></script>
</body>
```

CSS MODULE

```
        body > table{
    width: 100%;
}

table{
    width :100%;
    border-collapse : collapse;
    text-align: center;
}
table.list{
    width:100%;
    text-align: center;
}
table.list td{
    text-align: center;
}
td, th {
    border: 1px solid #dddddd;
    text-align: left;
    padding: 8px;
}
tr:nth-child(even),table.list thead>tr {
```

```
    background-color: skyblue;
}
input[type=text], input[type=number] {
    width: 100%;
    padding: 12px 20px;
    margin: 8px 0;
    display: inline-block;
    border: 1px solid #ccc;
    border-radius: 4px;
    box-sizing: border-box;
}
input[type=submit]{
    width: 30%;
    background-color: #ddd;
    color: #000;
    padding: 14px 20px;
    margin: 8px 0;
    border: none;
    border-radius: 4px;
    cursor: pointer;
}
form{
    background-color: skyblue;
    padding: 10px;
```

```
}

form div.form-action-buttons{

    text-align: center;



}

a{

    cursor: pointer;

    text-decoration: underline;

    color: #0000ee;

    margin-right: 4px;

}

label.validation-error{

    color:   red;

    margin-left: 5px;

}

.hide{

    display:none;

}
```

JAVASCRIPT MODULE

```
console.log("aaaaaaaaa");

        var selectedRow = null



// Form Submit Function

function onFormSubmit() {
```

```
    // check validity
    if (validate()) {
        // store user data
        var formData = readFormData();
        // check empty row
        if (selectedRow == null)
        {
            // Insert New User Record
            insertNewRecord(formData);
        }
        else
        {
            // Update New User Record
            updateRecord(formData);
        }
        // Reset Input Values
        resetForm();
    }
}
// Get Values From Form
function readFormData() {
    var formData = {};
    // Get Values From  Input
    formData["userName"] = document.getElementById("userName").value;
    formData["rollNo"] = document.getElementById("rollNo").value;
```

```
    formData["stdClass"] = document.getElementById("stdClass").value;

    formData["tsub"] = document.getElementById("tsub").value;

    formData["age"] = document.getElementById("age").value;

    formData["fee"] = document.getElementById("fee").value;

    formData["eroll"] = document.getElementById("eroll").value;

    formData["exam"] = document.getElementById("exam").value;

    formData["Exam"] = document.getElementById("age").value;

    // return Form Data

    return formData;

}

// Insert New User Record

function insertNewRecord(data) {

    var table =
document.getElementById("stdlist").getElementsByTagName('tbody')[0];

    var newRow = table.insertRow(table.length);

    cell1 = newRow.insertCell(0);

    cell1.innerHTML = data.userName;

    cell2 = newRow.insertCell(1);

    cell2.innerHTML = data.rollNo;

    cell3 = newRow.insertCell(2);

    cell3.innerHTML = data.stdClass;

    cell4 = newRow.insertCell(3);

    cell4.innerHTML = data.tsub;

    cell5 = newRow.insertCell(4);

    cell5.innerHTML = data.age;
```

```
    cell5 = newRow.insertCell(5);

    cell5.innerHTML = data.fee;

    cell5 = newRow.insertCell(6);

    cell5.innerHTML = data.eroll;

    cell5 = newRow.insertCell(7);

    cell5.innerHTML = data.exam;

    cell5 = newRow.insertCell(8);

    cell5.innerHTML = `<a onClick="onEdit(this)">Edit</a>

    <a onClick="onDelete(this)">Delete</a>`;

}

// Reset Function

function resetForm() {

    document.getElementById("userName").value = "";

    document.getElementById("rollNo").value = "";

    document.getElementById("stdClass").value = "";

    document.getElementById("tsub").value = "";

    document.getElementById("age").value = "";

    document.getElementById("fee").value = "";

    document.getElementById("enroll").value = "";

    document.getElementById("exam").value = "";

    selectedRow = null;

}

// Edit Function

function onEdit(td) {

    selectedRow = td.parentElement.parentElement;
```

```
    document.getElementById("userName").value =
selectedRow.cells[0].innerHTML;

    document.getElementById("rollNo").value =
selectedRow.cells[1].innerHTML;

    document.getElementById("stdClass").value =
selectedRow.cells[2].innerHTML;

    document.getElementById("tsub").value =
selectedRow.cells[3].innerHTML;

    document.getElementById("age").value =
selectedRow.cells[4].innerHTML;

    document.getElementById("fee").value =
selectedRow.cells[5].innerHTML;

    document.getElementById("eroll").value =
selectedRow.cells[6].innerHTML;

    document.getElementById("exam").value =
selectedRow.cells[7].innerHTML;

}

// Update Record

function updateRecord(formData) {

    selectedRow.cells[0].innerHTML = formData.userName;

    selectedRow.cells[1].innerHTML = formData.rollNo;

    selectedRow.cells[2].innerHTML = formData.stdClass;

    selectedRow.cells[3].innerHTML = formData.tsub;

    selectedRow.cells[4].innerHTML = formData.age;

    selectedRow.cells[5].innerHTML = formData.fee;

    selectedRow.cells[6].innerHTML = formData.eroll;

    selectedRow.cells[7].innerHTML = formData.exam;

}

// Delete Function
```

```
function onDelete(td) {
    if (confirm('Are you sure to delete this record ?')) {
        row = td.parentElement.parentElement;
        document.getElementById("stdlist").deleteRow(row.rowIndex);
        resetForm();
    }
}
// Check User validation
function validate() {
    isValid = true;
    // userName validation
    if (document.getElementById("userName").value == "") {
        isValid = false;

document.getElementById("userNamevalidationError").classList.remove("hi
de");
    } else {
        isValid = true;
        if
(!document.getElementById("userNamevalidationError").classList.contains
("hide"))
        {

document.getElementById("userNamevalidationError").classList.add("hide"
);
        }
    }
    // Roll No validation
```

```
    if (document.getElementById("rollNo").value == "") {

        isValid = false;


document.getElementById("rollNovalidationError").classList.remove("hide
");

    } else {

        isValid = true;

        if
(!document.getElementById("rollNovalidationError").classList.contains("
hide"))

        {


document.getElementById("rollNovalidationError").classList.add("hide");

        }

    }

    // Std class validation

    if (document.getElementById("stdClass").value == "") {

        isValid = false;


document.getElementById("stdClassvalidationError").classList.remove("hi
de");

    } else {

        isValid = true;

        if
(!document.getElementById("stdClassvalidationError").classList.contains
("hide"))

        {


document.getElementById("stdClassvalidationError").classList.add("hide"
);
```

```
        }

    }

    // Tsub validation

    if (document.getElementById("tsub").value == "") {

        isValid = false;


document.getElementById("tsubvalidationError").classList.remove("hide")
;

    } else {

        isValid = true;

        if
(!document.getElementById("tsubvalidationError").classList.contains("hi
de"))

        {


document.getElementById("tsubvalidationError").classList.add("hide");

        }

    }

    // Age validation

    if (document.getElementById("age").value == "") {

        isValid = false;


document.getElementById("agevalidationError").classList.remove("hide");

    } else {

        isValid = true;

        if
(!document.getElementById("agevalidationError").classList.contains("hid
e"))

        {
```

```
document.getElementById("agevalidationError").classList.add("hide");

        }

    }

    return isValid;

}
```

5.2 STEPS TO IMPLEMENT THE CODE

Step 1: Open Visual studio code.

Step 2: Create a folder and name it.

Step 3: Create a file named “index.html” and type the code for the HTML module.

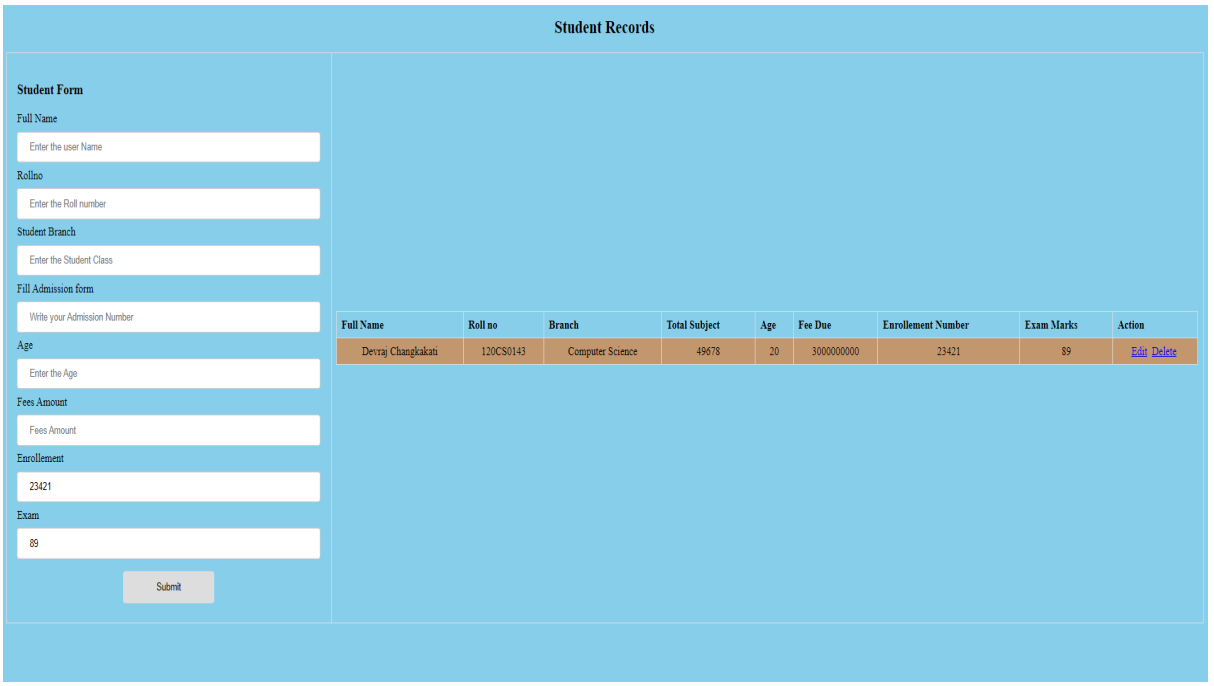
Step 4: Create a CSS file named “style.css ” and type the CSS code.

Step 5: Create a Javascript file named “1.js ” and type the Javascript code .

Step 6:save all the programs .

Step 7: to execute the program , right click on the file”index.html” and click open with live server in the drop down menu.

5.3 Screenshot of the GUI/Results that would include all the features:



A. Appendices

A.1 Appendix 1

- **Activity diagram manual**

UML Activity Diagram

Overview:

Activity diagram is another important diagram in UML to describe dynamic aspects of the system.

Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system.

So the control flow is drawn from one operation to another. This flow can be sequential, branched or concurrent. Activity diagrams deal with all types of flow control by using different elements like fork, join etc.

Purpose:

The basic purposes of activity diagrams are similar to the other four diagrams. It captures the dynamic behavior of the system. Other four diagrams are used to show the message flow from one object to another but the activity diagram is used to show message flow from one activity to another.

Activity is a particular operation of the system. Activity diagrams are not only used for visualizing the dynamic nature of a system but they are also used to construct the executable system by using forward and reverse engineering techniques. The only missing thing in the activity diagram is the message part.

It does not show any message flow from one activity to another. Activity diagram is sometimes considered as the flow chart. Although the diagram looks like a flow chart, it is not. It shows different flows like parallel, branched, concurrent and single.

So the purposes can be described as:

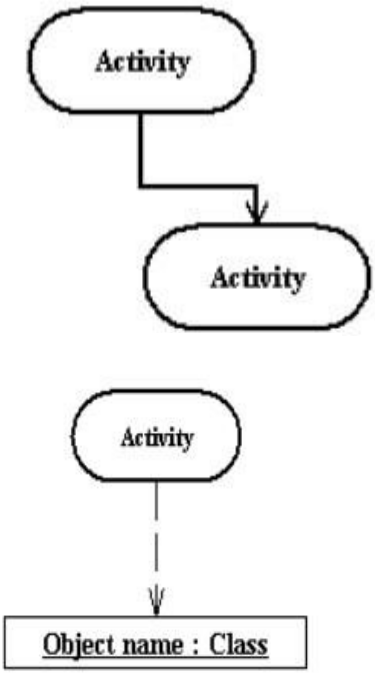
- Draw the activity flow of a system.
- Describe the sequence from one activity to another.
- Describe the parallel, branched and concurrent flow of the system.

Basic Activity Diagram Symbols and Notations

Action state Action states represent the non interruptible actions of objects. You can draw an action state in SmartDraw using a rectangle with rounded corners.

Action Flow

Action flow arrows illustrate the relationships among action states.



Object Flow

Object flow refers to the creation and modification of objects by activities. An object flow arrow from

an action to an object means that the action creates or influences the object. An object flow arrow from an object to an action indicates that the action state uses the object.

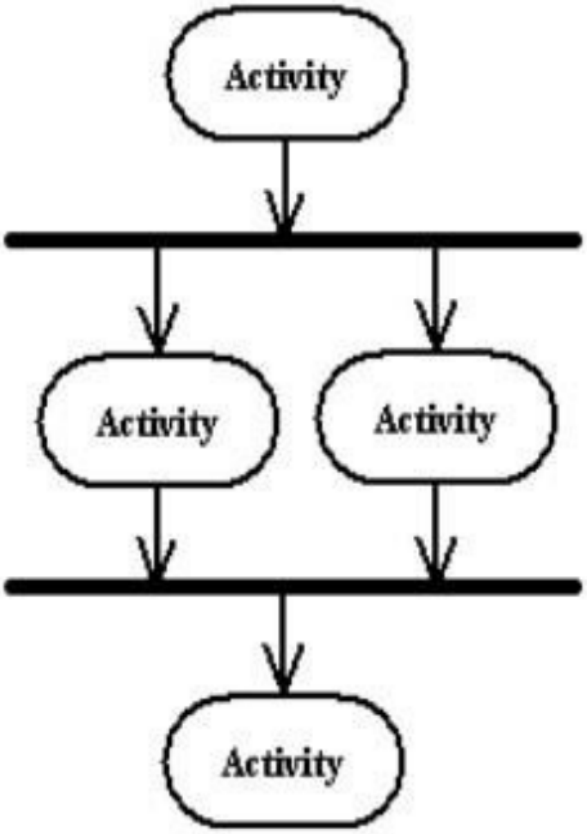
Initial State **Final State**

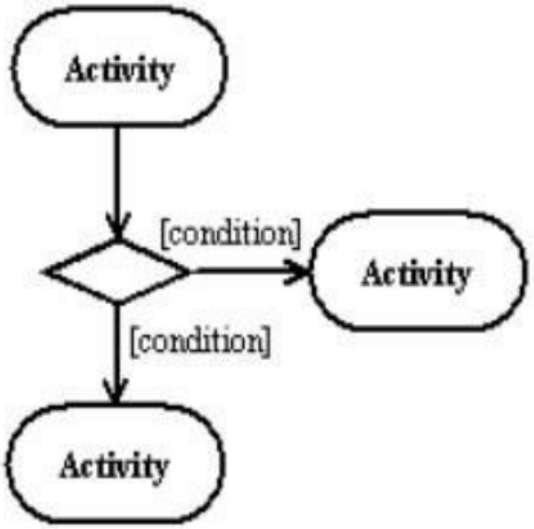
A filled circle followed by an arrow represents the initial action state.

An arrow pointing to a filled circle nested inside another circle represents the final action state.

Branching

A diamond represents a decision with alternate paths. The outgoing alternates should be labeled with a condition or guard expression. You can also label one of the paths "else."



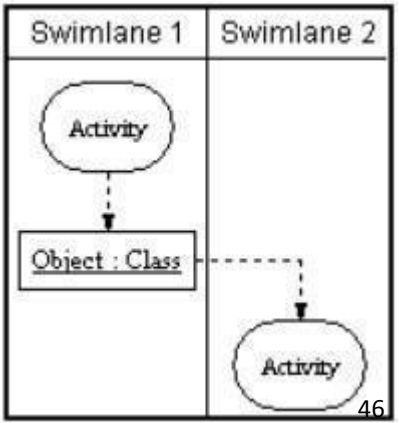


Synchronization

A synchronization bar helps illustrate parallel transitions. Synchronization is also called forking and joining.

Swimlanes

Swimlanes group related activities into one column.



Where to use Activity Diagrams?

The basic usage of the activity diagram is similar to the other four UML diagrams. The specific usage is to model the control flow from one activity to another. This control flow does not include messages. The activity diagram is suitable for modeling the activity flow of the system. An application can have multiple systems. Activity diagram also captures these systems and describes flow from one system to another. This specific usage is not available in other diagrams. These systems can be database, external queues or any other system. Now we will look into the practical applications of the activity diagram. From the above discussion it is clear that an activity diagram is drawn from a very high level. So it gives a high level view of a system. This high level view is mainly for business users or any other person who is not a technical person. This diagram is used to model the activities which are nothing but business requirements. So the diagram has more impact on business understanding rather than implementation details.
Following are the main usages of activity diagram:

- Modeling workflow by using activities.
- Modeling business requirements.
- High level understanding of the system's functionalities.
- Investigate business requirements at a later stage.

A.2 Appendix 2• StateChart Diagram manual

UML Statechart Diagram

Overview:

The name of the diagram itself clarifies the purpose of the diagram and other details. It describes different states of a component in a system. The states are specific to a component/object of a system.

A Statechart diagram describes a state machine. Now to clarify it, a state machine can be defined as a machine which defines different states of an object and these states are controlled by external or internal events. Activity diagram explained in the next chapter, is a special kind of a Statechart diagram. As the Statechart diagram defines states it is used to model lifetime of an object.

Purpose:

Statechart diagram is one of the five UML diagrams used to model the dynamic nature of a system. They define different states of an object during its lifetime. And these states are changed by events. So Statechart diagrams are useful to model reactive systems. Reactive systems can be defined as a system that responds to external or internal events.

Statechart diagram describes the flow of control from one state to another state. States are defined as a condition in which an object exists and it changes when some event is triggered. So the most important purpose of a Statechart diagram is to model the time of an object from creation to termination.

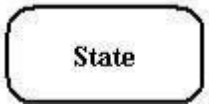
Statechart diagrams are also used for forward and reverse engineering of a system. But the main purpose is to model a reactive system.
Following are the main purposes of using Statechart diagrams:

- To model the dynamic aspect of a system.
- To model the time of a reactive system.
- To describe different states of an object during its lifetime.
- Define a state machine to model states of an object.

Basic Statechart Diagram Symbols and Notations

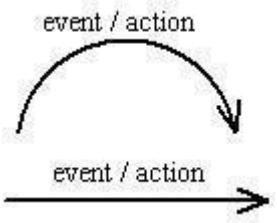
States

States represent situations during the life of an object. You can easily illustrate a state in SmartDraw by using a rectangle with rounded corners.



Transition

A solid arrow represents the path between different states of an object. Label the transition with the event that triggered it and the action that results from it.



Initial State

A filled circle followed by an arrow represents the object's initial

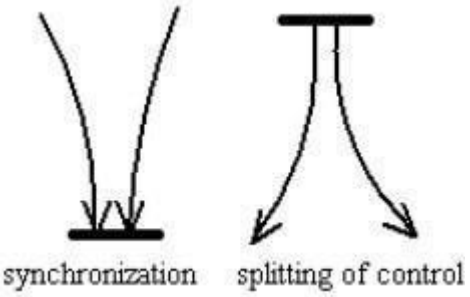
state.

Final State

An arrow pointing to a filled circle nested inside another circle represents the object's final state.

Synchronization and Splitting of Control

A short heavy bar with two transitions entering it represents a synchronization of control. A short heavy bar with two transitions leaving it represents a splitting of control that creates multiple states.



How to draw a Statechart Diagram?

Statechart diagram is used to describe the states of different objects in its life cycle. So the emphasis is given on the state changes upon some internal or external events. These states of objects are important to analyze and implement them accurately.

Statechart diagrams are very important for describing the states. States can be identified as the condition of objects when a particular event occurs. Before drawing a Statechart diagram we must have clarified the following points:

Identify important objects to be analyzed.

Identify the states.

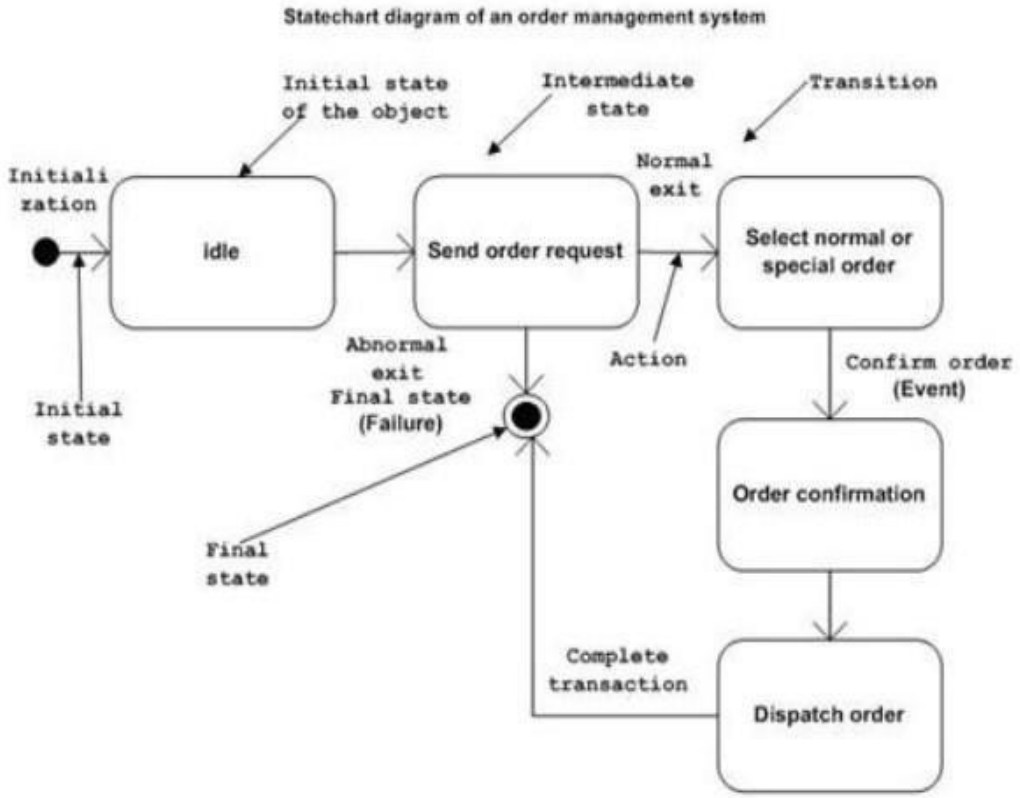
Identify the events.

The following is an example of a Statechart diagram where the state of *Order* object is analyzed.

The first state is an idle state from where the process starts. The next states are arrived for events like *send request*,*confirm request*, and *dispatch order*.

These events are responsible for state changes of order objects. During the life cycle of an object (here order object) it goes through the following states and there may be some abnormalities also. This abnormal exit may occur due to some problem in the system. When the entire life cycle is complete it is considered as the complete transaction as mentioned below.

The initial and final state of an object is also shown below.



Where to use Statechart Diagrams?

From the above discussion we can define the practical applications of a Statechart diagram. Statechart diagrams are used to model dynamic aspect of a system like other four diagrams

disused in this tutorial. But it has some distinguishing characteristics for modeling dynamic nature.

Statechart diagrams define the states of a component and these state changes are dynamic in nature. So its specific purpose is to define state changes triggered by events. Events are internal or external factors influencing the system.

Statechart diagrams are used to model states and also events operating on the system. When implementing a system it is very important to clarify different states of an object during its lifetime and statechart diagrams are used for this purpose. When these states and events are identified they are used to model it and these models are used during implementation of the system.

If we look into the practical implementation of the Statechart diagram then it is mainly used to analyze the object states influenced by events. This analysis is helpful to understand the system behavior during its execution. So the main usages can be described as:

- To model object states of a system.
- To model a reactive system. Reactive system consists of reactive objects.
- To identify events responsible for state changes.
- Forward and reverse engineering.

Team Members

- Devraj Changkakati (120CS0143)
- Anirudh Jayakumar (120CS0161)
- Pramod Kumar Mallik (120CS0180)
- Tapaswini Mishra(120CS0186)
- Sabitri Mohapatra(120CS0188)
- B. Naga Pravallika (120CS0121)
- K. Srujana Kavya (120CS0169)
- L. Hema (120CS0158)
- Swayam Kumar Nahak (120CS0193)