EX NO:1	WRITE THE COMPLETE PROBLEM STATEMENT
DATE	

AIM:

To prepare a PROBLEM STATEMENT for a student result management system.

ALGORITHM:

- The problem statement is the initial starting point for a project.
- A problem statement describes what needs to be done without describing how.
- It is generally a one-to-three-page document that all project stakeholders agree upon, describing the goals of the project at a high level.
- The problem statement is intended for a broad audience and should be written in non-technical terms.
- It helps both technical and non-technical personnel communicate effectively by providing a clear description of the problem.
- The problem statement does not describe the solution to the problem.

INPUT:

- The input to requirement engineering is the problem statement prepared by the customer.
- It may include an overview of the existing system and the broad expectations from the new system.
- The first phase of requirements engineering begins with requirements elicitation, i.e., gathering information about the requirements.

  Here, requirements are identified with the help of the customer and existing system processes.

Problem:

The traditional methods of managing student results involve manual or semi-automated processes that are time-consuming, prone to errors, and lack standardization. These inefficiencies lead to challenges such as delays in result generation, difficulties in maintaining accurate records, and errors in grade calculations, which can negatively impact both students and educational institutions.

Background:

Traditional result management relies on manual processes, leading to inefficiencies, errors, and delays in handling student academic records. As institutions grow, these challenges become more pronounced, affecting accuracy and accessibility. A digital system ensures efficiency, security, and transparency, enhancing the experience for all stakeholders.

Relevance:

A Student Result Management System is vital for modernizing academic processes in educational institutions. It addresses the inefficiencies of manual result handling, ensuring accurate and timely processing of student performance data.

By leveraging technology, it enhances accessibility, security, and transparency, benefiting students, educators, and administrators.

Objectives:

1. **Automate Result Processing**: Streamline the calculation and generation of student results to eliminate manual errors and reduce processing time.
2. **Enhance Data Accessibility**: Provide secure, real-time access to results for students, teachers, and administrators through a centralized platform.
3. **Improve Record Management**: Maintain accurate, well-organized records of student performance for easy retrieval and analysis.
4. **Ensure Security and Confidentiality**: Implement robust data encryption and user authentication to protect sensitive academic data.
5. **Support Scalability**: Adapt to the needs of growing student populations and evolving academic structures.
6. **Promote Transparency**: Enable stakeholders to view and verify results, fostering trust and accountability in academic processes.

Result:

EX NO:2	WRITE THE SOFTWARE REQUIREMENT SPECIFICATION DOCUMENT
DATE	

AIM:

To do requirement analysis and develop Software Requirement Specification Sheet (SRS) for Student Result Management System.

ALGORITHM:

SRS shall address are the following:

a) **Functionality.** What is the software supposed to do?

b) **External interfaces.** How does the software interact with people, the system‘s hardware, other hardware, and other software?

c) **Performance.** What is the speed, availability, response time, recovery time of various software functions, etc.?

d) **Attributes.** What is the portability, correctness, maintainability, security, etc. considerations?

e) **Design constraints imposed on an implementation.** Are there any required standards in effect, implementation language, policies for database integrity, resource limits, operating environment(s) etc.?

1. Introduction

- **1.1 Purpose:**
  The purpose of this SRS document is to define the functional, non-functional, and technical requirements of the Student Result Management System.
- **1.2 Scope:**
  The Student Result Management System is designed to automate the process of recording, processing, and accessing student academic results.
- **1.3 Definitions, Acronyms, and Abbreviations:**
  List all key terms, abbreviations, and acronyms used throughout the document.
- **1.4 References:**
  Institutional academic policies for result calculation and reporting. Data privacy regulations (e.g., GDPR for student data security)

---

2. Overall Description

- **2.1 Product Perspective:**
  The Student Result Management System is a centralized, web-based application designed to replace traditional manual methods.

- **2.2 Product Features:**
  Provide an overview of the core features of the system, such as Student profile management , Student profile management
- **2.3 User Classes and Characteristics:**
  Identify the different user types (e.g., Students, Teachers, administrators) and their needs and permissions.
- **2.4 Operating Environment:**
  Describe the technical environment (e.g., web platform, mobile app, server specifications, operating systems) in which Student Result Management System will operate.
- **2.5 Design and Implementation Constraints:**
  Identify any constraints on the system design, such as compliance with regulations, security standards, or technological limitations.
- **2.6 Assumptions and Dependencies:**
  List assumptions made during system development and any external dependencies (e.g., Availability of a reliable student database).

---

3. System Features

- **3.1 Feature 1: Student Profile Management**
  - **Description:** Create and manage individual student profiles with academic details.
  - **Functional Requirements:** Data entry for student details, course enrollment, and academic history.
- **3.2 Feature 2: Grade Entry and Calculation**
  - **Description:** Teachers enter grades, and the system calculates final results based on predefined criteria.
  - **Functional Requirements**: Grade input, formula-based calculation, and error-checking mechanisms.
- **3.3 Feature 3: Report Generation**
  - **Description:** Generate detailed performance reports for students and classes.
  - **Functional Requirements:** Dynamic report generation in formats like PDF or Excel, with graphical insights.
- **3.4 Feature 4: Secure Access and Role Management**
  - **Description:** Role-based access for students, teachers, and administrators to ensure data privacy.
  - **Functional Requirements:** Secure login, role assignment, and access control mechanisms.

---

4. External Interface Requirements

- **4.1 User Interfaces:**
  Intuitive and responsive web-based interfaces for all users.

- **4.2 Hardware Interfaces:**
  Intuitive and responsive web-based interfaces for all users.
- **4.3 Software Interfaces:**
  Compatibility with existing academic management systems and third-party tools.
- **4.4 Communication Interfaces:**
  Encrypted communication protocols (e.g., HTTPS) for secure data exchange.

---

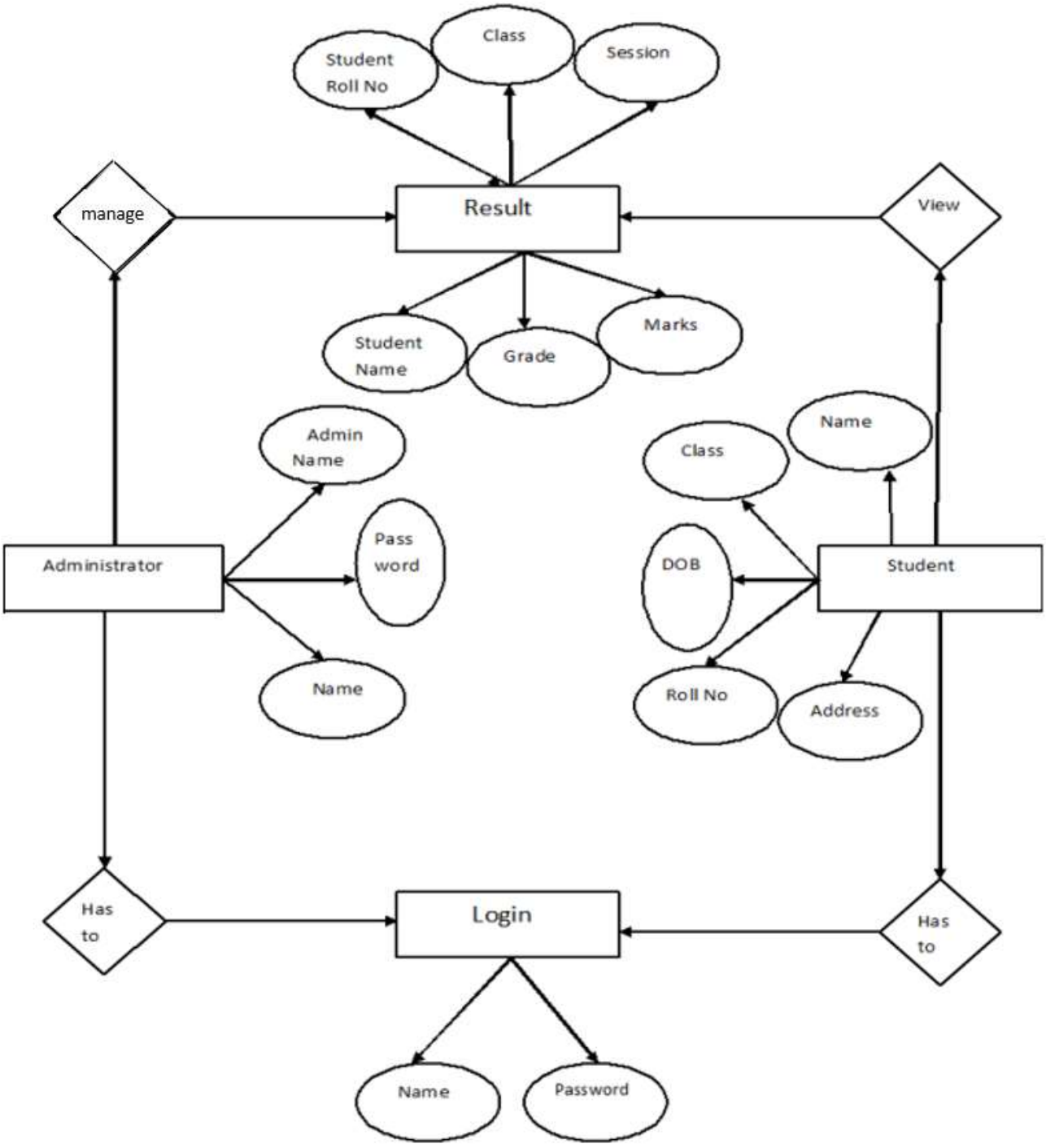
5. System Attributes

- **5.1 Performance Requirements:**
  Outline system performance criteria, such as response times, load handling and data processing speed.
- **5.2 Security Requirements:**
  Specify the security standards and features required for the system, including encryption, audit trails, and protection against cyberattacks (e.g., DDoS).
- **5.3 Reliability:**
  Define expected system uptime, fault tolerance, and backup/recovery requirements.
- **5.4 Availability:**
  System must be accessible 24/7 with minimal scheduled downtime.

Result:

SAMPLE OUTPUT:

ER DIAGRAM:



EX NO:3	DRAW THE ENTITY RELATIONSHIP DIAGRAM
DATE	

AIM:

To Draw the Entity Relationship Diagram for Student Result Management System.

ALGORITHM:

Step 1: Mapping of Regular Entity Types

Step 2: Mapping of Weak Entity Types

Step 3: Mapping of Binary 1:1 Relation Types

Step 4: Mapping of Binary 1:N Relationship Types.

Step 5: Mapping of Binary M:N Relationship Types.

Step 6: Mapping of Multivalued attributes.

INPUT:

Entities

Entity Relationship Matrix

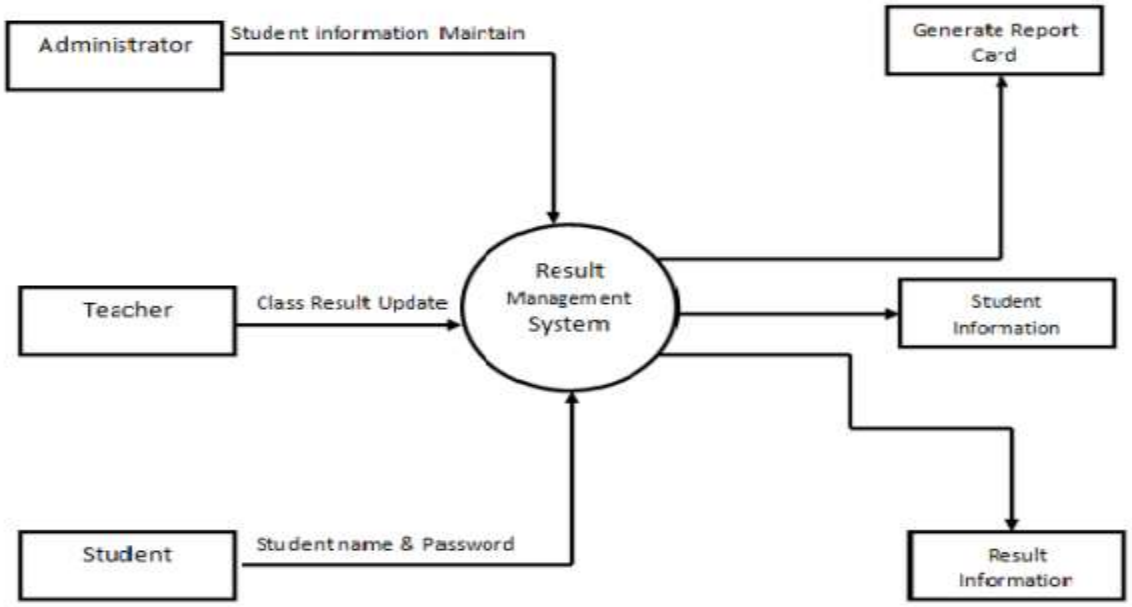
Primary Keys

Attributes

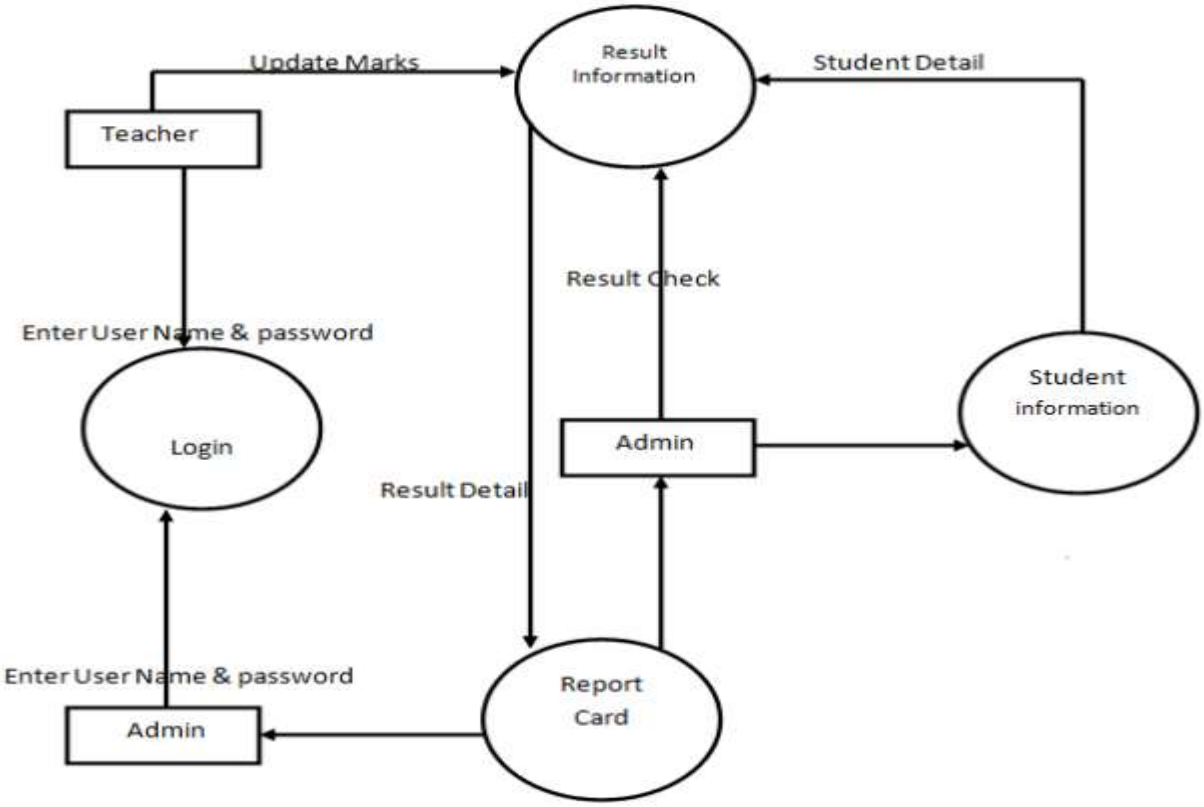
Mapping of Attributes with Entities

Result:

ZERO LEVEL:



FIRST LEVEL:



EX NO:4	DRAW THE DATA FLOW DIAGRAMS AT LEVEL 0 AND LEVEL 1
DATE	

AIM:

To Draw the Data Flow Diagram for Student Result Management System and List the Modules in the Application.

ALGORITHM:

1. Open the Visual Paradigm to draw DFD (Ex.Lucidchart)
2. Select a data flow diagram template
3. Name the data flow diagram
4. Add an external entity that starts the process
5. Add a Process to the DFD
6. Add a data store to the diagram
7. Continue to add items to the DFD
8. Add data flow to the DFD
9. Name the data flow
10. Customize the DFD with colours and fonts
11. Add a title and share your data flow diagram

INPUT:

Processes

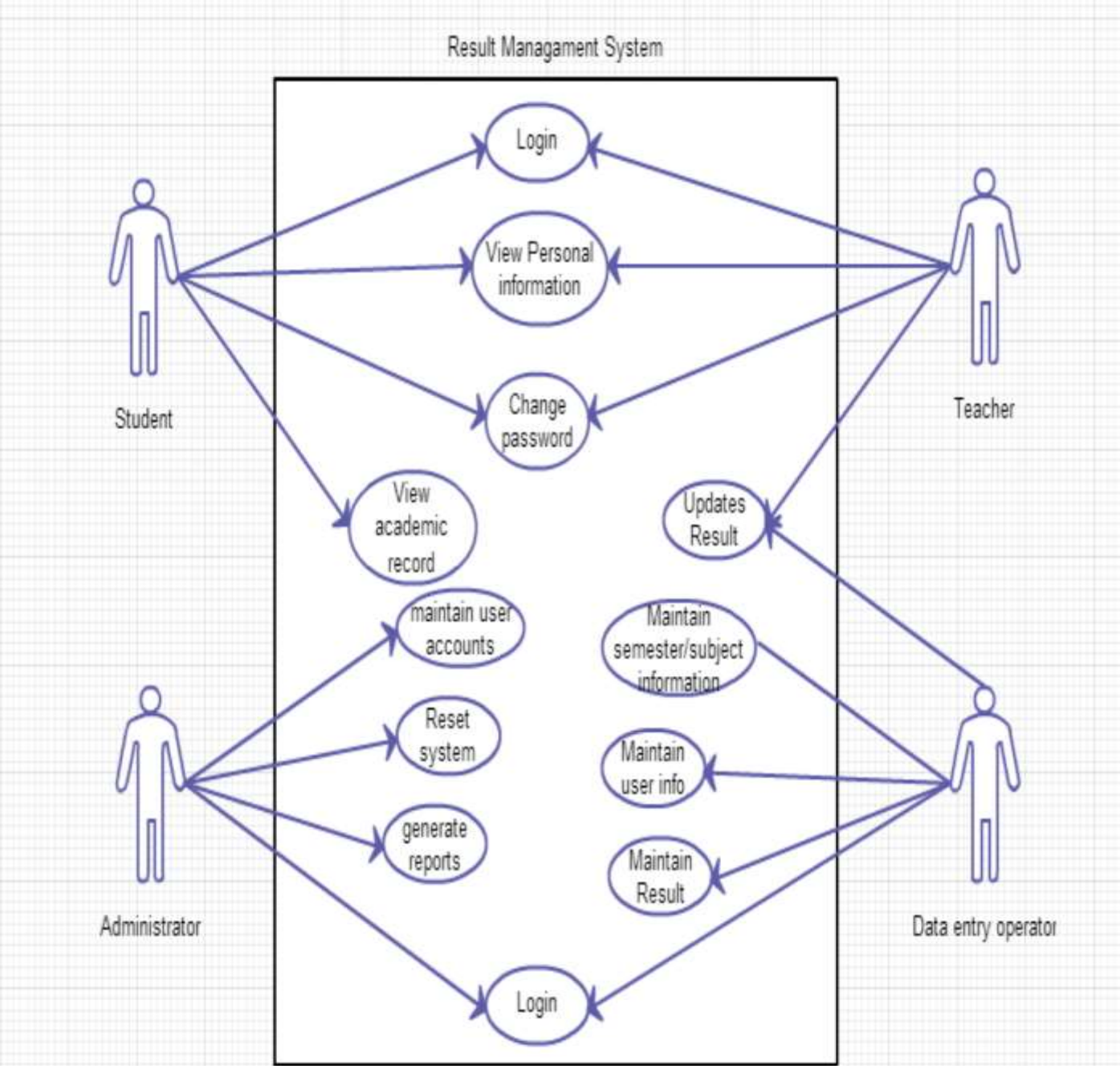
Datastores

External Entities

Result:

SAMPLE OUTPUT:

USE CASE DIAGRAM:



EX NO:5	DRAW USE CASE DIAGRAM
DATE	

AIM:

To Draw the Use Case Diagram for Student Result Management System**.**

ALGORITHM:

Step 1: Identify Actors

Step 2: Identify Use Cases

Step 3: Connect Actors and Use Cases

Step 4: Add System Boundary

Step 5: Define Relationships

Step 6: Review and Refine

Step 7: Validate

INPUTS:

Actors

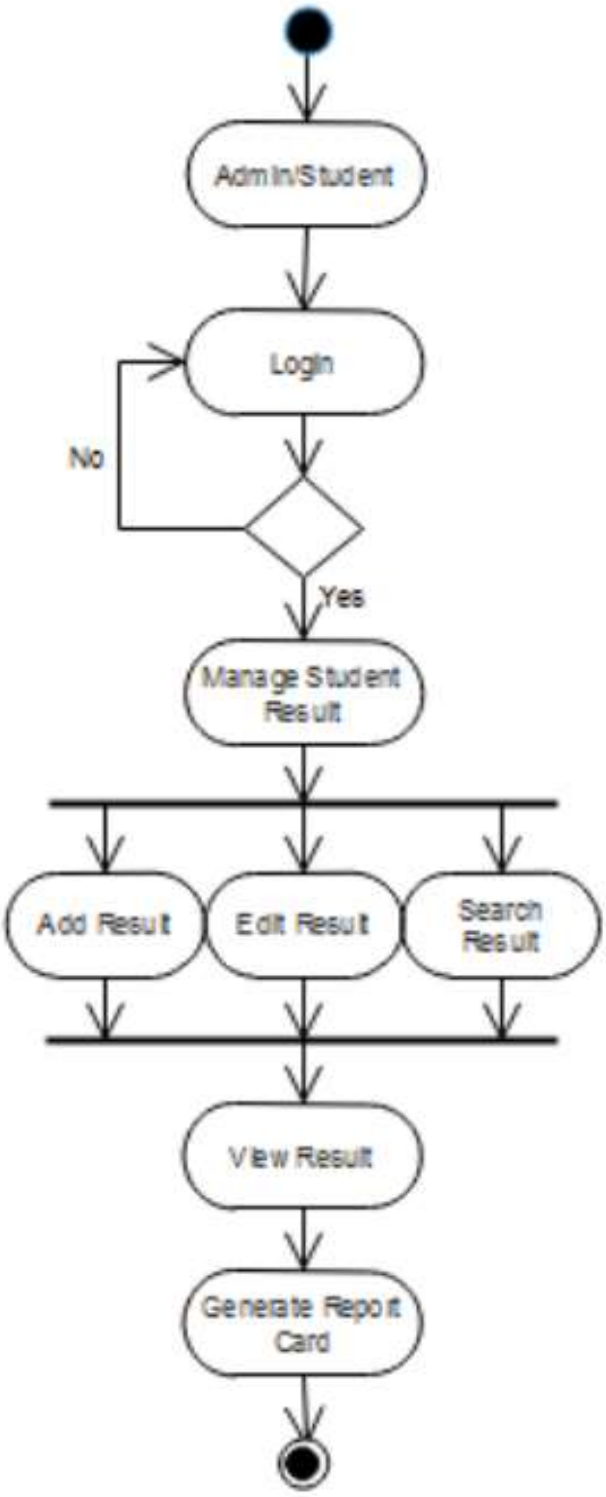
Use Cases

Relations

Result:

SAMPLE OUTPUT:

ACTIVITY DIAGRAM:



EX NO:6	DRAW ACTIVITY DIAGRAM OF ALL USE CASES.
DATE	

AIM:

To Draw the activity Diagram Student Result Management System.

ALGORITHM:

Step 1: Identify the Initial State and Final States

Step 2: Identify the Intermediate Activities Needed

Step 3: Identify the Conditions or Constraints

Step 4: Draw the Diagram with Appropriate Notations

INPUTS:

Activities

Decision Points

Guards

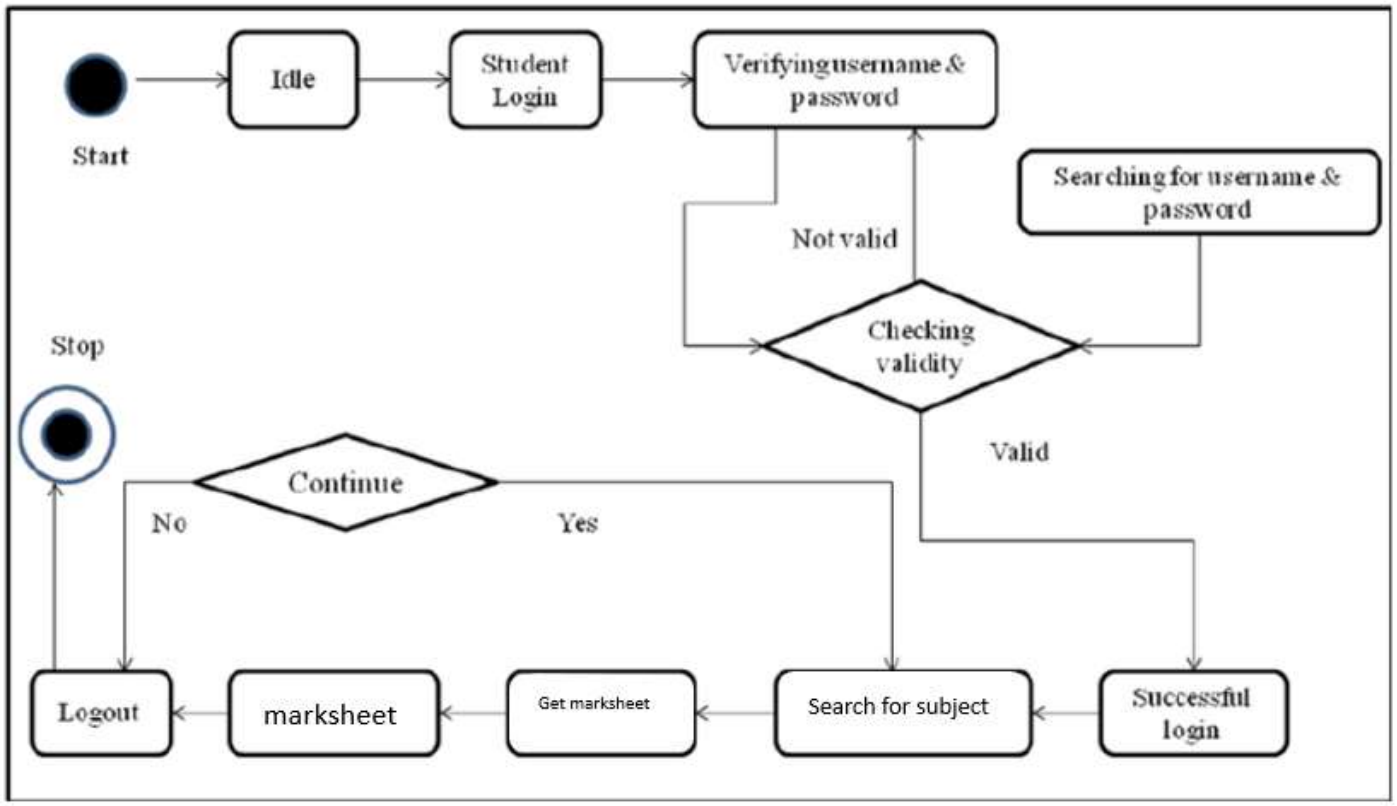
Parallel Activities

Conditions

Result:

SAMPLE OUTPUT:

STATE CHART DIAGRAM:



EX NO:7	DRAW STATE CHART DIAGRAM OF ALL USE CASES.
DATE	

AIM:

To Draw the State Chart Diagram for Student Result Management System.

ALGORITHM:

STEP-1: Identify the important objects to be analysed.

STEP-2: Identify the states.

STEP-3: Identify the events.

INPUTS:

Objects

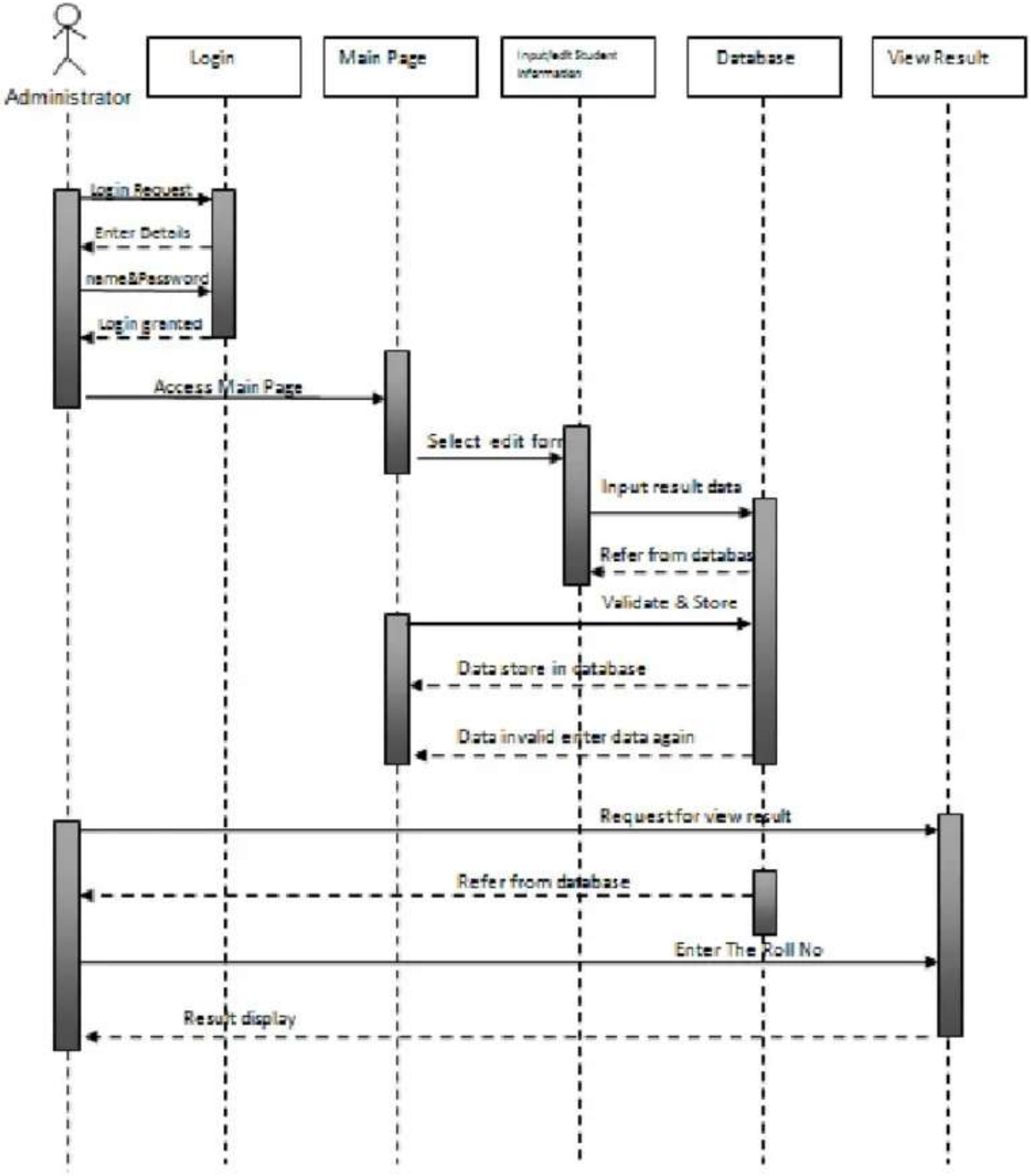
States

Events

Result:

SAMPLE OUTPUT:

SEQUENCE DIAGRAM:



EX NO:8	DRAW SEQUENCE DIAGRAM OF ALL USE CASES.
DATE	

AIM:

To Draw the Sequence Diagram for Student Result Management System.

ALGORITHM:

1. Identify the Scenario
2. List the Participants
3. Define Lifelines
4. Arrange Lifelines
5. Add Activation Bars
6. Draw Messages
7. Include Return Messages
8. Indicate Timing and Order
9. Include Conditions and Loops
10. Consider Parallel Execution
11. Review and Refine
12. Add Annotations and Comments
13. Document Assumptions and Constraints
14. Use a Tool to create a neat sequence diagram

INPUTS:

Objects taking part in the interaction.

Message flows among the objects.

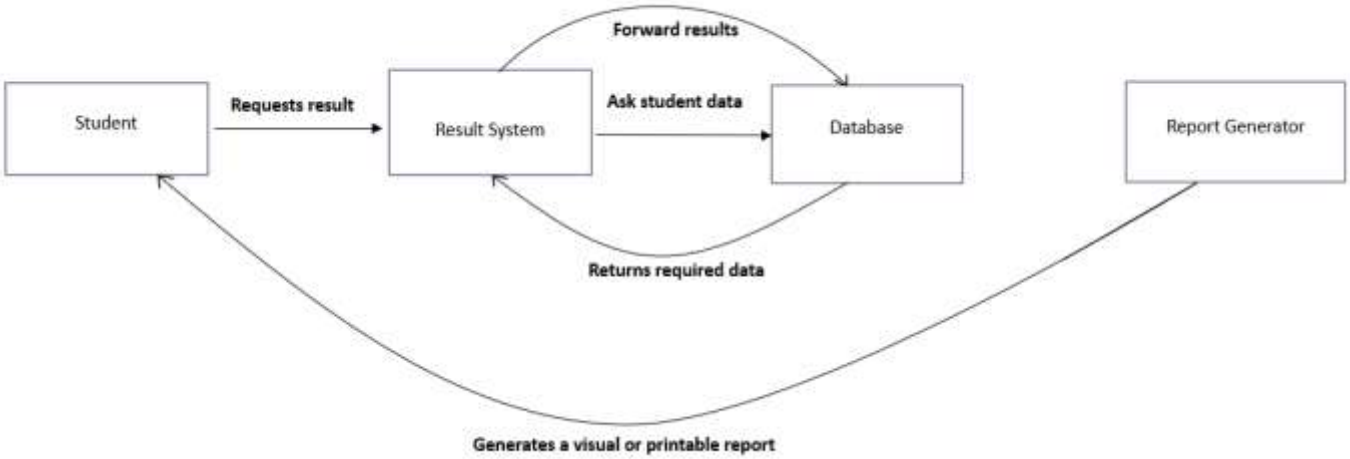
The sequence in which the messages are flowing.

Object organization.

Result:

SAMPLE OUTPUT:

COLLABORATION DIGRAM:



EX NO:9	DRAW COLLABORATION DIAGRAM OF ALL USE CASES
DATE	

AIM:

To Draw the Collaboration Diagram for Student Result Management System.

ALGORITHM:

Step 1: Identify Objects/Participants

Step 2: Define Interactions

Step 3: Add Messages

Step 4: Consider Relationships

Step 5: Document the collaboration diagram along with any relevant explanations or annotations.

INPUTS:

Objects taking part in the interaction.

Message flows among the objects.

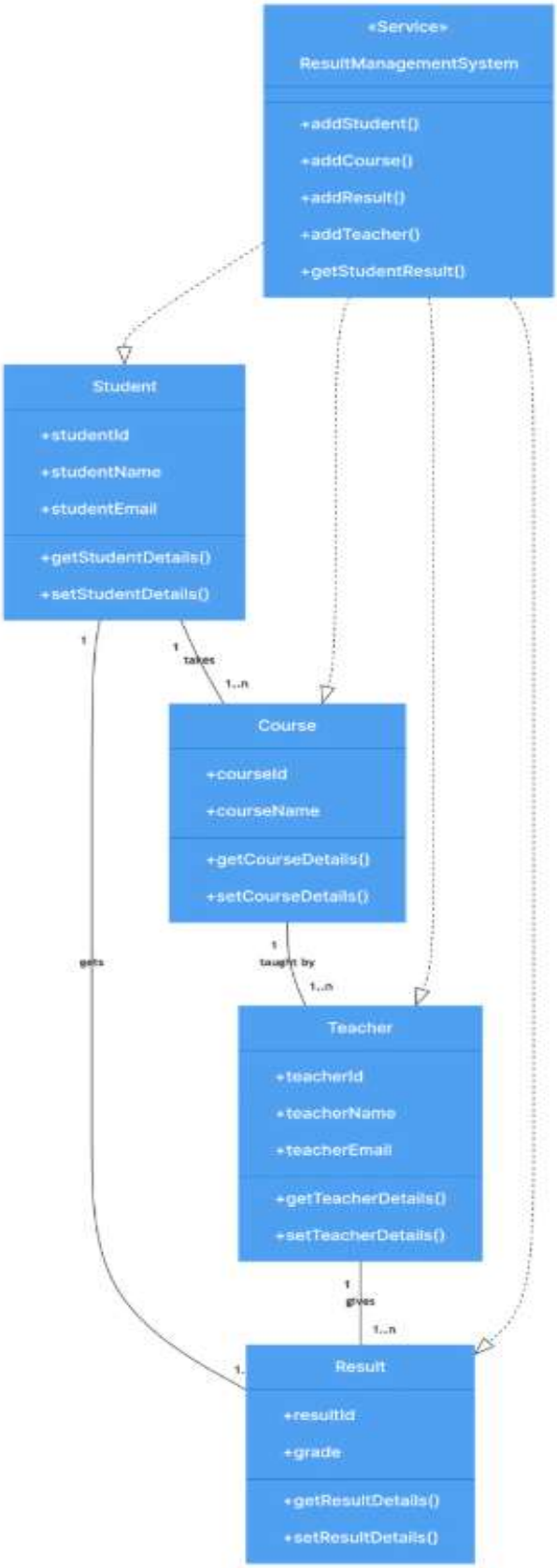
The sequence in which the messages are flowing.

Object organization.

Result:

SAMPLE OUTPUT:

CLASS DIAGRAM:



EX NO:10	ASSIGN OBJECTS IN SEQUENCE DIAGRAM TO CLASSES AND MAKE CLASS DIAGRAM.
DATE	

AIM:

To Draw the Class Diagram for Student Result Management System.

ALGORITHM:

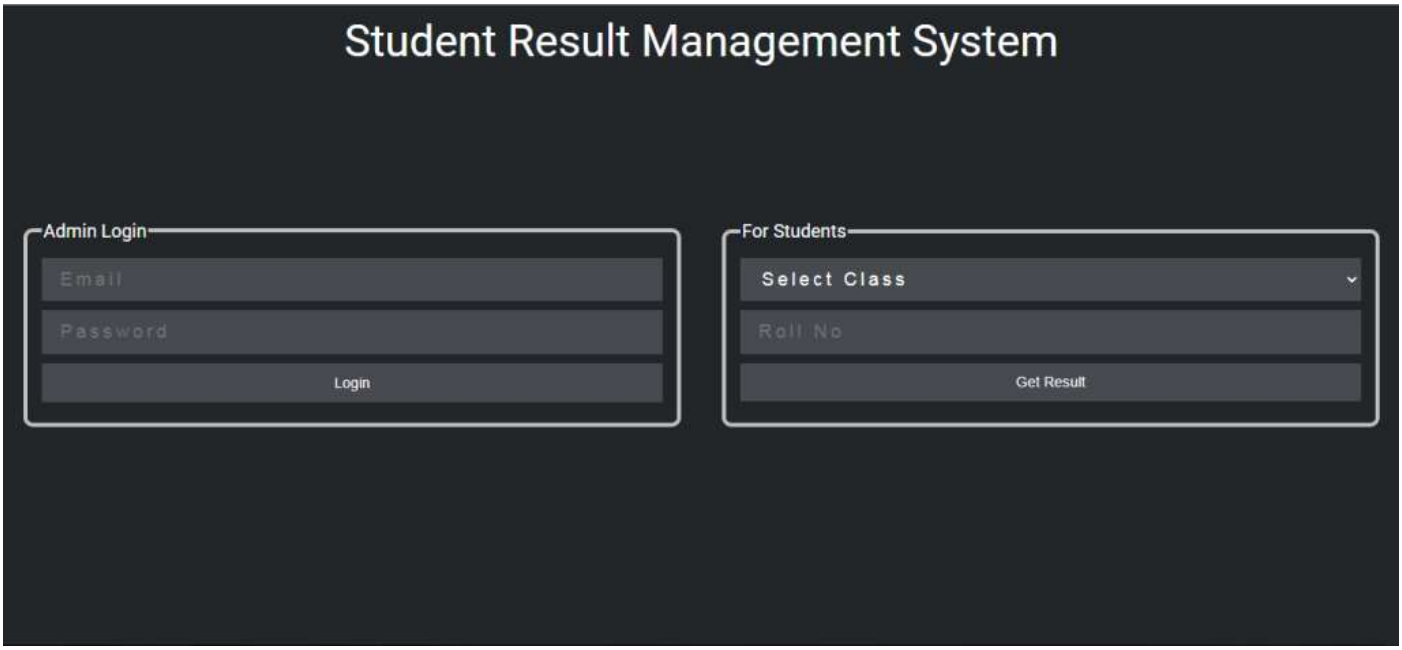
1. Identify Classes
2. List Attributes and Methods
3. Identify Relationships
4. Create Class Boxes
5. Add Attributes and Methods
6. Draw Relationships
7. Label Relationships
8. Review and Refine
9. Use Tools for Digital Drawing

INPUTS:

1. Class Name
2. Attributes
3. Methods
4. Visibility Notation

RESULT:

OUTPUT:


Student Result Management System
Admin Login
Email
Password
Login
For Students
Select Class
Roll No
Get Result


srms
Dashboard
Logout
Classes
Students
Results
Manage Students
NAME ROLL NO CLASS
Ajay 1011 First Year
Vijay 1012 First Year
Arjun 2022 Second Year
Karan 2021 Second Year
Ram 3031 Third Year
Sham 3032 Third Year

EX NO:11	MINI PROJECT - **Student Result Management System**
DATE	

AIM:

The primary aim of this mini-project is to develop a secure and user-friendly Student Result Management System. By utilizing MySQL for robust data storage and Streamlit for a seamless user interface.

ALGORITHM:

1. **Initialize** session state variables (students DataFrame and admin_authenticated status).
2. Display **Welcome Screen** with the title and system description.
3. Provide **Admin Login** with username/password validation.
4. If logged in, display the **Admin Dashboard** with options to manage student records.
5. Allow admin to **Add Student Records** by entering Student ID, Name, Subject, and Marks.
6. Provide options to **View All Records**, **Search Results** by Student ID, and calculate **Average Marks**.
7. Include an **Admin Logout** feature to reset authentication.

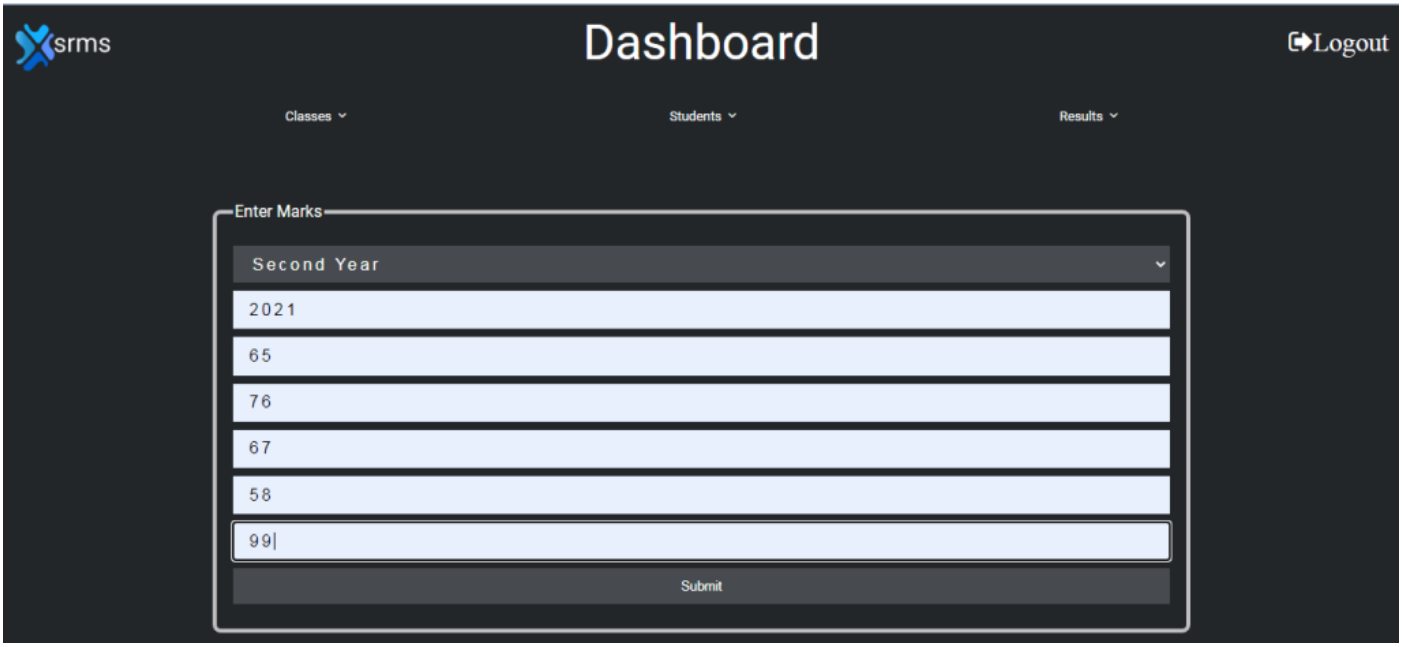
PROGRAM:

```
import streamlit as st
import pandas as pd

# Initialize session state to store data
if 'students' not in st.session_state:

# Example dataset
    st.session_state.students = pd.DataFrame(columns=['Student ID', 'Name', 'Subject', 'Marks'])
if 'admin_authenticated' not in st.session_state:
    st.session_state.admin_authenticated = False  # For admin authentication

# Admin credentials (for simplicity, using hardcoded values)
ADMIN_USERNAME = "admin"
ADMIN_PASSWORD = "admin123"
```



Name: Karan
Class: Second Year
Roll No: 2021

Subjects	Marks
Paper 1	65
Paper 2	76
Paper 3	67
Paper 4	58
Paper 5	99

Total Marks: 365

Percentage: 73%

Print Result

```
# App Title
st.title("Student Result Management System")
st.write("Welcome to the Student Result Management System! Admins can manage student records and view results.")

# Admin Authentication Section
if not st.session_state.admin_authenticated:
    st.subheader("Admin Login")
    admin_username = st.text_input("Enter Admin Username", max_chars=20)
    admin_password = st.text_input("Enter Admin Password", type="password")

    if st.button("Login as Admin"):
        if admin_username == ADMIN_USERNAME and admin_password == ADMIN_PASSWORD:
            st.session_state.admin_authenticated = True
            st.success("Admin authenticated successfully!")
        else:
            st.error("Invalid admin credentials.")

# Main Application - Admin Functions
if st.session_state.admin_authenticated:
    st.subheader("Admin Dashboard")

    # Add Student Records
    st.write("### Add Student Results")
    student_id = st.text_input("Enter Student ID", key="student_id")
    name = st.text_input("Enter Student Name", key="name")
    subject = st.text_input("Enter Subject", key="subject")
    marks = st.number_input("Enter Marks (0-100)", min_value=0, max_value=100, key="marks")

        if st.button("Add Record"):
            if student_id and name and subject:

                # Add data to the session state DataFrame
                new_record = pd.DataFrame({'Student ID': [student_id], 'Name': [name], 'Subject': [subject], 'Marks': [marks]})
                st.session_state.students = pd.concat([st.session_state.students, new_record], ignore_index=True)
                st.success("Record added successfully!")
            else:
                st.error("Please fill in all fields.")
```

```
# View All Records
st.write("### View All Student Records")
if not st.session_state.students.empty:
    st.dataframe(st.session_state.students)
else:
    st.write("No records found.")

# Search Student Results
st.write("### Search Student Results")
search_id = st.text_input("Enter Student ID to Search Results", key="search_id")
if st.button("Search"):
    student_records = st.session_state.students[st.session_state.students['Student ID'] == search_id]
    if not student_records.empty:
        st.write(f"Results for Student ID: {search_id}")
        st.dataframe(student_records)
    else:
        st.error("No records found for this Student ID.")

# Calculate Average Marks
st.write("### Calculate Average Marks")
if not st.session_state.students.empty:
    average_marks = st.session_state.students['Marks'].mean()
    st.write(f"Average Marks: **{average_marks:.2f}**")
else:
    st.write("No data available to calculate averages.")

# Admin Logout
if st.button("Logout"):
    st.session_state.admin_authenticated = False
    st.success("Logged out successfully!")
```

Conclusion:

The Student Result Management System is a streamlined and efficient tool designed to help administrators manage student records effectively. It provides a secure login mechanism for admins and features like adding, viewing, and searching student records, as well as calculating average marks. The system ensures ease of use with a user-friendly interface and real-time data handling. This scalable solution can be further enhanced with persistent storage and role-based access, making it a valuable asset for educational institutions in managing academic performance data.