

RAJALAKSHMI ENGINEERING COLLEGE

**RAJALAKSHMI NAGAR, THANDALAM – 602 105**

**RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)**

**RAJALAKSHMI NAGAR, THANDALAM – 602-105 BONAFIDE CERTIFICATE**

**NAME: REGISTER NO.:**

**ACADEMIC YEAR**: 2024-25 **SEMESTER:** III **BRANCH:** B.E/B.Tech

This Certification is the bonafide record of work done by the above student in the

**CB23332-SOFTWARE ENGINEERING -** Laboratory during the year 2024 – 2025.

Signature of Faculty -in – Charge

Submitted for the Practical Examination held on

Internal Examiner External Examiner



|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 1. | Preparing Problem Statement |  |  |
| 2. | Software Requirement Specification (SRS) |  |  |
| 3. | Entity-Relational Diagram |  |  |
| 4. | Data Flow Diagram |  |  |
| 5. | Use Case Diagram |  |  |
| 6. | Activity Diagram |  |  |
| 7. | State Chart Diagram |  |  |
| 8. | Sequence Diagram |  |  |
| 9. | Collaboration Diagramt |  |  |
| 10. | Class Diagram |  |  |

**1. PREPARING PROBLEM STATEMENT**

**Aim :**

**1. PREPARING PROBLEM STATEMENT**

In educational institutions, managing and disseminating student results is a critical function that directly impacts students’ academic journeys. Traditional result management involves a significant amount of paperwork, manual calculations, and data entry, leading to inefficiencies, delays, and potential errors. An automated Student Result Management System is essential to streamline these processes, offering reliable, fast, and accurate results management that is accessible to both students and faculty.

**Background**

With increasing class sizes and the need for regular assessments, maintaining student records and ensuring their timely availability has become challenging. Faculty and administrative staff spend substantial time processing results, often delaying feedback that is essential for students’ academic progression. Furthermore, students frequently face challenges accessing their results, tracking their academic standing, and obtaining transcripts in a timely manner.

**Relevance**

An efficient system is vital to support students in planning their academic goals and to aid administrators in providing an organized, transparent, and errorfree recordkeeping process. Modern educational environments necessitate a digital approach to result management to enhance efficiency, accessibility, and transparency.

**2. Current System Challenges**

The traditional approach to managing student results poses several challenges:

**Manual Data Entry and Errors:** Grades and marks are often recorded manually in spreadsheets or paper forms, increasing the likelihood of data entry errors. Such errors can lead to incorrect records, impacting students' academic assessments.

**TimeConsuming Process:** Teachers and administrators spend significant time compiling, verifying, and calculating results. This process can delay result announcements and affect students’ ability to make timely academic decisions.

**Inconsistent Access to Information**: In traditional setups, students may need to wait to receive their results physically or via email, lacking immediate access to their academic records.

**Data Loss Risks:** Paperbased records are susceptible to physical damage and loss. Even digital files in decentralized systems face risks of data loss due to inadequate backup systems.

**Limited Analytics and Reporting** : Traditional systems lack integrated tools for performance analysis. Generating insights on class averages, student progression, or subjectspecific performance requires additional, often manual, effort.

These challenges underscore the need for a reliable Student Result Management System that is capable of addressing both functional and nonfunctional requirements in a modern educational context.

**3. Objectives of the Student Result Management System**

The Student Result Management System (SRMS) aims to address these issues by providing a streamlined, automated platform for recording, calculating, and sharing student academic results. The following objectives outline the primary goals:

**1. Efficient Result Entry and Management:** Facilitate quick and accurate entry of student grades, reducing the need for manual calculations and minimizing errors.

**2. 21Automated GPA and Performance Calculations:** Automatically calculate semester and cumulative Grade Point Averages (GPA) based on entered grades, providing students with uptodate academic standings.

**3. RoleBased Access Control:** Provide secure access for students, teachers, and administrators, ensuring that each user only accesses information pertinent to their role.

**4. Report Generation and Analytics:** Enable teachers and administrators to generate various reports, such as individual student transcripts, class performance summaries, and subjectspecific statistics.

**5. Secure Data Storage and Backup:** Ensure the security and longevity of student records with regular backups and data encryption, preventing data loss and unauthorized access.

**6. RealTime Access for Students:** Allow students to access their results instantly upon release, enabling them to track academic progress and make timely educational decisions.

**4. System Requirements and Functional Specifications**

**The SRMS will encompass multiple functional modules designed to streamline result management:**

**a) Result Entry and Update Module**

Input Validation: Ensure that all entered grades adhere to predefined grading scales.

Bulk Upload: Support bulk upload of marks through spreadsheets or other formats to minimize data entry time.

Modification and Verification: Enable teachers to modify grades before finalization, with verification processes to ensure accuracy.

**b) GPA Calculation Module**

Automated GPA Calculation: Calculate GPAs for each semester based on entered grades, factoring in subject weights and credits.

Cumulative Performance Tracking: Maintain a record of cumulative GPAs to provide students with a comprehensive view of their academic performance.

**c) Report and Analytics Module**

Result Reports: Generate detailed reports for individual students, showing marks, grades, GPA, and cumulative scores.

Analytics for Teachers: Allow teachers to view and analyze trends in student performance, providing insights on areas where students may need support.

Institutional Reporting: Enable administrators to generate reports on overall class or batch performance.

**d) Secure Login and RoleBased Access Control**

Student Access: Students can view their own results, download reports, and access historical records.

Teacher Access: Teachers can enter and update results for their assigned classes and generate performance reports.

Admin Access: Administrators can oversee the entire system, manage user roles, and perform audits of result entries and modifications.

**e) Backup and Security Module**

Regular Data Backups: Scheduled backups to ensure data integrity and continuity in case of system failure.

Data Encryption and Secure Storage: Encrypt sensitive data, such as student records and login credentials, to protect against unauthorized access.

**5. Expected Outcomes and Benefits**

The implementation of a Student Result Management System is anticipated to yield several benefits:

**1. Improved Efficiency**: The automated system will reduce the time and effort required for result management, allowing teachers and administrators to focus on other critical tasks.

**2. Enhanced Accuracy**: By minimizing manual data entry, the system will reduce errors, ensuring that students’ academic records are reliable and accurate.

**3. Greater Accessibility** for Students: Students will be able to access their results immediately upon release, which supports transparency and allows students to make informed academic choices.

**4. Data Security and Integrity**: The use of secure login, encryption, and regular backups will protect student records from unauthorized access and data loss.

**5. Informed DecisionMaking**: Teachers and administrators will have access to performance analytics, allowing for datadriven decisions to improve academic programs and address areas of student difficulty.

**Result :**

The problem statement was written successfully by following the steps described above.

**Ex.NO. 2 : Write the software requirement specification document**

**Aim :**

The aim of the **Student Result Management System (SRMS)** is to design and implement a robust, user-friendly, and efficient platform for managing and automating student results. This system ensures accurate, timely, and secure handling of academic records while reducing manual effort and errors.

**Algorithm :**

**1. Introduction**

**1.1 Purpose**

The purpose of this document is to define the requirements for a Student Result Management System (SRMS) that automates the process of result entry, GPA calculation, and report generation. It provides secure access for students, faculty, and administrators to view and manage academic records efficiently.

**1.2 Scope**

The SRMS is designed for educational institutions to manage and streamline resultrelated processes. Key functionalities include result entry, GPA calculation, student and faculty access to records, and report generation for academic analysis.

**1.3 Definitions, Acronyms, and Abbreviations**

SRMS: Student Result Management System

GPA: Grade Point Average

Admin: Administrator

1.4 References

Academic grading policies and GPA calculation standards

Institution’s data security protocols

**2. Overall Description**

2.1 Product Perspective

The SRMS is a webbased application integrated into the institution’s IT infrastructure, allowing rolebased access to students, faculty, and administrators. It centralizes academic data, supports secure login, and offers data analytics for performance insights.

**2.2 Product Features**

Result entry and modification

Automated GPA calculation

Realtime result access for students

Report generation and performance analytics

Secure, rolebased access control

**2.3 User Classes and Characteristics**

Students: View personal academic results and download reports.

Faculty: Enter and manage student grades for their courses.

Administrators: Oversee system operations, manage user accounts, and generate institutionwide performance reports.

**2.4 Operating Environment**

The system is webbased, compatible with major browsers (Chrome, Firefox, Edge), and optimized for mobile access. It uses a cloudhosted database for secure data storage.

**3. System Features**

**3.1 Result Entry and Update**

Description: Faculty can enter and modify student results. The system validates entries against grading criteria.

Inputs: Student ID, course code, and grades.

Outputs: Updated result database and student report.

**3.2 GPA Calculation**

Description: Automatically calculates GPAs based on course credits and grades, updated in realtime.

Inputs: Course credits and grades.

Outputs: GPA for each semester and cumulative GPA.

**3.3 Report Generation**

Description: Generates downloadable reports, including semesterwise and cumulative performance.

Inputs: Student ID, semester selection.

Outputs: PDF reports and downloadable transcripts.

**3.4 Access Control**

Description: Rolebased authentication to restrict access. Students, faculty, and admins have different levels of system permissions.

Inputs: User credentials.

Outputs: Access to features based on user role.

**4. External Interface Requirements**

**4.1 User Interfaces**

Student Dashboard: View academic results and download reports.

Faculty Dashboard: Enter grades, view class performance.

Admin Dashboard: Manage users, view performance analytics.

**4.2 Hardware Interfaces**

Supports desktop and mobile devices with internet access.

**4.3 Software Interfaces**

Integrates with the institution’s database and Student Information System (SIS) for seamless data transfer.

**4.4 Communication Interfaces**

Uses HTTPS for secure data transmission and supports API integrations for external reporting tools.

**5. System Requirements**

**5.1 Functional Requirements**

FR1: Faculty can enter and modify student results.

FR2: System calculates and displays GPA automatically.

FR3: Students can view and download their result reports.

FR4: Admin can generate and download institutionwide reports.

**5.2 NonFunctional Requirements**

**5.2.1 Performance Requirements**

The system should handle up to 1,000 concurrent users without performance degradation.

**5.2.2 Security Requirements**

Encryption: Encrypt user credentials and sensitive data.

Authentication: Implement secure login with rolebased access control.

**5.2.3 Usability Requirements**

Userfriendly interface with clear navigation.

**5.2.4 Reliability Requirements**

System uptime should be maintained at 99.9%.

**5.2.5 Scalability Requirements**

The system should accommodate future expansions in user count and data volume.

**6. Use Case Diagrams**

**6.1 Use Case 1: Enter and Update Results**

Actors: Faculty

Description: Faculty enter grades for assigned students and modify them if needed.

**6.2 Use Case 2: View Results**

Actors: Students

Description: Students log in to view their results, including individual grades and GPA.

**6.3 Use Case 3: Generate Reports**

Actors: Admin

Description: The admin generates academic performance reports and downloads them.

**7. System Models**

**7.1 Data Flow Diagram (DFD)**

Level 1: Depicts data flow between users (students, faculty, admin) and the SRMS.

**7.2 EntityRelationship Diagram (ERD)**

Shows relationships between entities: Student, Course, Faculty, Grade, and GPA.

**8. Security and Privacy**

Authentication: Rolebased authentication for different user levels.

Data Privacy: User data is encrypted and access is restricted.

Regular Backups: Automated backups ensure data is recoverable.

**Result :**

The SRS was made successfully by following the steps described above.

**3. ENTITY RELATIONSHIP MODEL**

**Aim :**

To Draw the Entity Relationship Diagram for student result management system(srms).

**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.

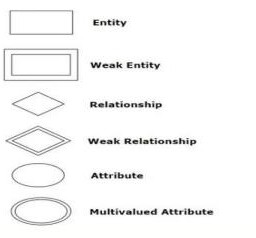
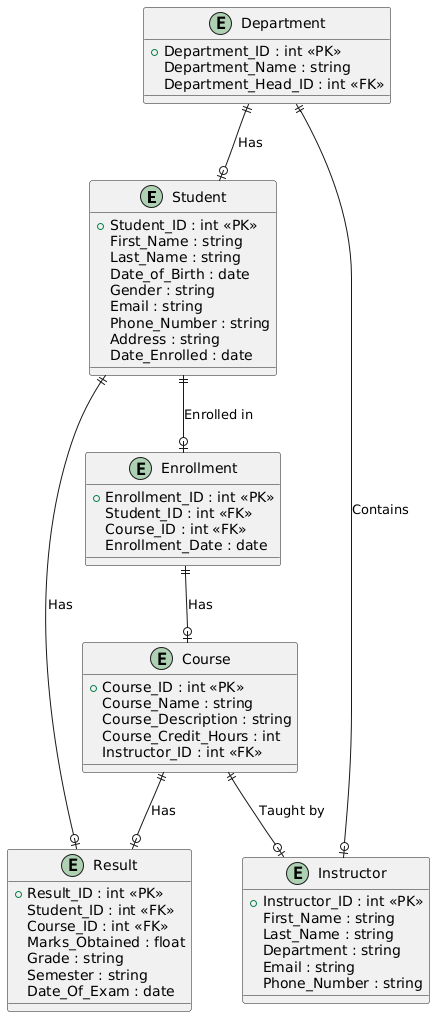
**Input :**

1. **Student**
   * **Student\_ID** (Primary Key)
   * First\_Name
   * Last\_Name
   * Date\_of\_Birth
   * Gender
   * Email
   * Phone\_Number
   * Address
   * Date\_Enrolled
2. **Course**
   * **Course\_ID** (Primary Key)
   * Course\_Name
   * Course\_Description
   * Course\_Credit\_Hours
   * Instructor\_ID (Foreign Key)
3. **Instructor**
   * **Instructor\_ID** (Primary Key)
   * First\_Name
   * Last\_Name
   * Department
   * Email
   * Phone\_Number
4. **Result**
   * **Result\_ID** (Primary Key)
   * Student\_ID (Foreign Key)
   * Course\_ID (Foreign Key)
   * Marks\_Obtained
   * Grade
   * Semester
   * Date\_Of\_Exam
5. **Department**
   * **Department\_ID** (Primary Key)
   * Department\_Name
   * Department\_Head\_ID (Foreign Key)
6. **Enrollment**
   * **Enrollment\_ID** (Primary Key)
   * Student\_ID (Foreign Key)
   * Course\_ID (Foreign Key)
   * Enrollment\_Date

|  |  |
| --- | --- |
| Entity | Relationship |
| User | User\_id, username, email, password |
| Book | Book\_id, tittle,author,isbn,published\_date |
| Loan |  |

## ER DIAGRAM:

SYMBOLS:



**Result:** The entity relationship diagram was made successfully by following the steps described above.

**4. DATA FLOW DIAGRAM**

**Aim** : To Draw the Data Flow Diagram for student result management system and List the Modules in the Application.

# Algorithm :

1. Open Visual Paradigm or Lucidchart.
2. Select a DFD Template and name it "Smart Library Management System."
3. Add External Entities: Include ‘Library User’ and ‘Librarian.’
4. Add Processes: Create processes like Login, Book Search, Book Issuing, Book Return, Online Book Renewal, and Fine Calculation.
5. Add Data Stores: Include stores like Library Database, User Account Data Store, and Book Inventory Data Store.
6. Connect Entities, Processes, and Data Stores: Draw data flows like "Login Credentials," "Book Details," and "Issue Record."
7. Label Data Flows: Clearly label the data exchanged between entities and processes.
8. Customize: Adjust colors, fonts, and titles for clarity.
9. Export/Share: Export or share the completed DFD.

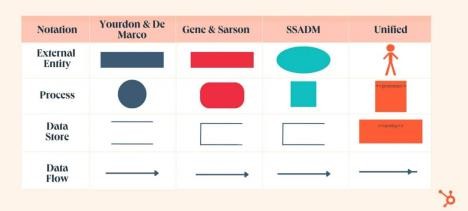
**Input :**

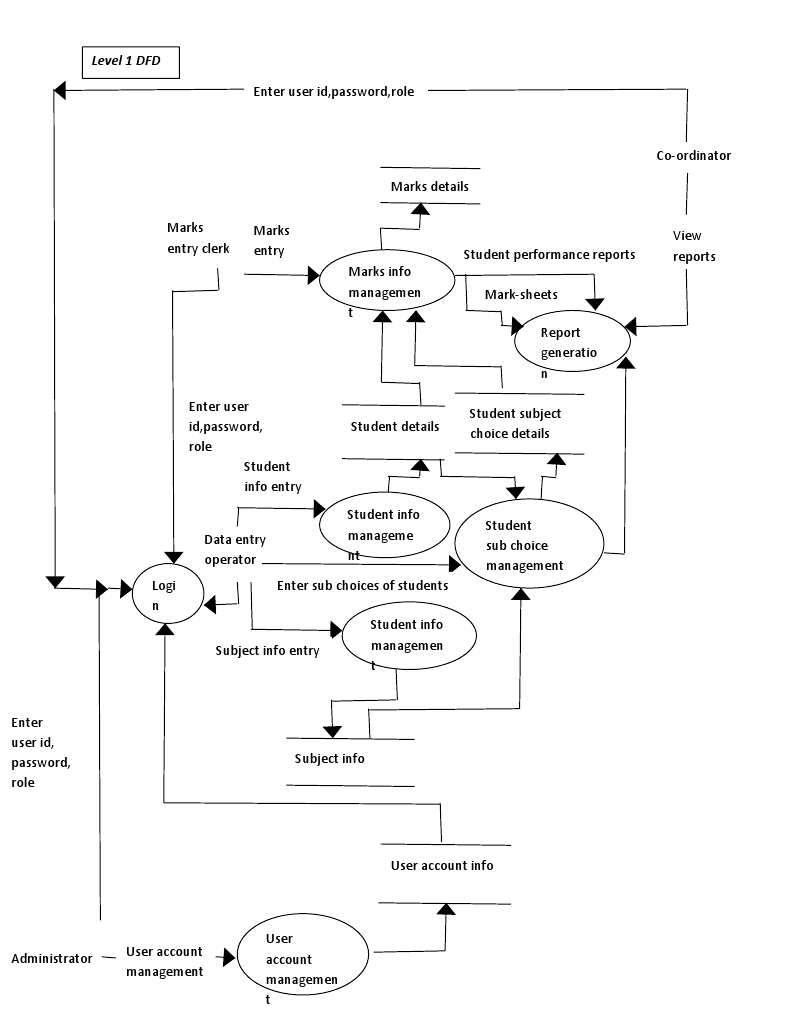
**Processes:**

* Enroll Student, Enter Results, Generate Reports,.

 **Relationships**:

* Entities (Student, Instructor, Admin) interact with specific processes, which then access the necessary data stores:





**Result:** The Data Flow diagram was made successfully by following the steps described above.

**5. USE CASE DIAGRAM**

**Aim :** To Draw the Use Case Diagram for student result management system

**Algorithm :**

**1. Login:**

1. Start.
2. User enters username and password.
3. System verifies the credentials.
4. If credentials are correct:
   * Grant access based on user role (Admin, Teacher, or Student).
5. If credentials are incorrect:
   * Deny access and prompt the user to re-enter or recover the password.
6. End.

**2. Manage Student Details (Admin and Teacher):**

1. Start.
2. Admin or Teacher logs in and selects the "Manage Student Details" option.
3. Search for a student by name or ID.
4. If student exists:
   * Update details (e.g., name, contact info, class) as needed.
5. If student does not exist:
   * Add new student details.
6. Save changes.
7. Notify user of successful update.
8. End.

**3. Enter/Edit Grades (Teacher):**

1. Start.
2. Teacher logs in and selects the "Enter/Edit Grades" option.
3. Select the student and course/subject.
4. Input or edit grades.
5. Save the updated grades in the system database.
6. Notify the teacher of successful update.
7. End.

**4. View Results (Admin, Teacher, and Student):**

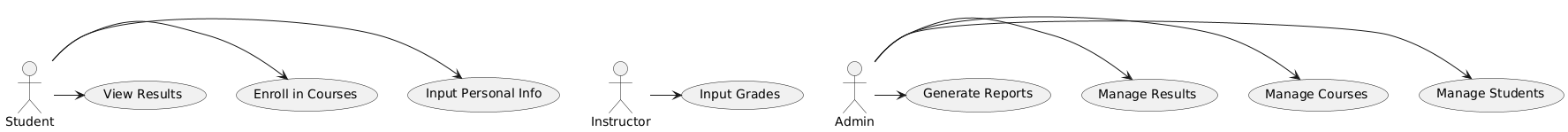
1. Start.
2. User (Admin, Teacher, or Student) logs in.
3. Select the "View Results" option.
4. User specifies the term/semester or course for which results are needed.
5. System retrieves and displays the student’s grades and overall performance.
6. End.

**5. Generate Report Cards (Admin and Teacher):**

1. Start.
2. Admin or Teacher logs in and selects the "Generate Report Cards" option.
3. Select the student and term/semester for report card generation.
4. System retrieves all grades for the selected period.
5. Calculate the total score, percentage, and any remarks.
6. Generate a structured report card.
7. Save and/or print the report card.
8. End.

**6. Send Notifications (Admin):**

1. Start.
2. Admin logs in and selects the "Send Notifications" option.
3. Select recipients (individual student, group, or class).
4. Compose the message (e.g., grade release, event reminder).
5. Send the notification.
6. Confirm that notifications have been sent successfully.
7. End.



**Result :**

The use case diagram has been created successfully by following the steps given.

**6. ACTIVITY DIAGRAM**

**Aim**:

To Draw the activity Diagram for STUDENT RESULT MANAGEMENT SYSTEM

**ALGORITHM:**

1. Identify Initial and Final States - Initial State: System Idle/Logged Out - Final State: User Logs Out/System Shutdow

2. Identify Intermediate Activities - User Logs In, Search Book, Issue Book, Return Book, Renew Book Online, Pay Fines.

3. Identify Conditions or Constraints - Successful Login, Book Availability, Valid Renewal, Fine Payment Completion.

4. Draw the Diagram - Use ovals for states (Login, Search, Issue, etc.), arrows for transitions, and the initial/final state symbols to map the flow.

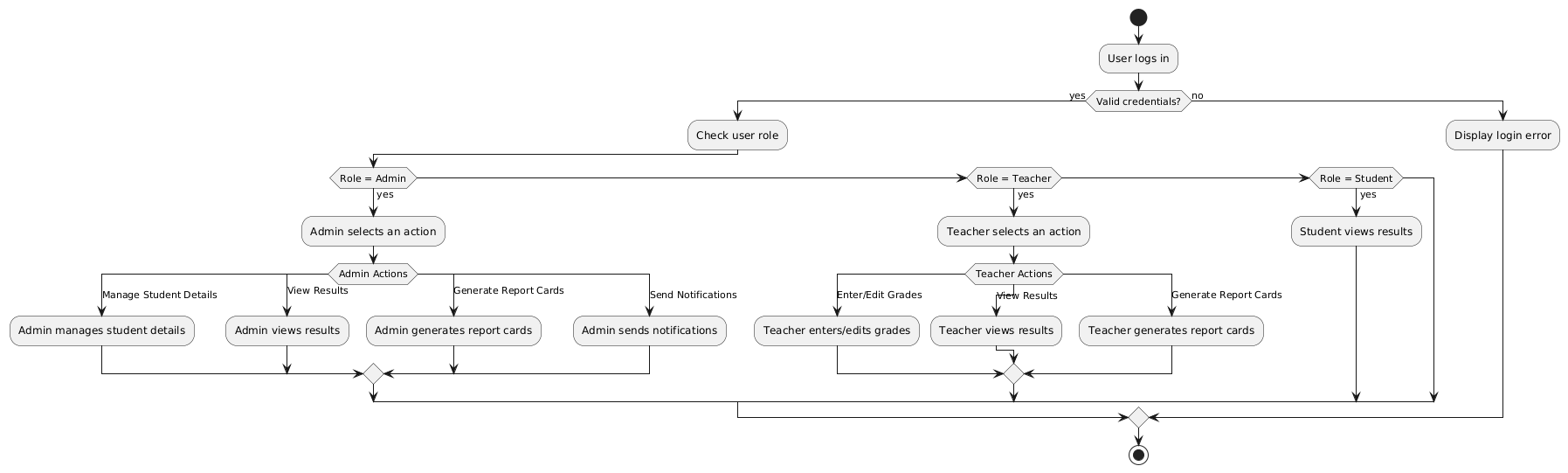
**Inputs :**

Activities Decision Points Guards

Parallel Activities Conditions

**Explanation:**

1. **Start** node: Represents the start of the login process.
2. **Login Activity**: User logs in.
3. **Decision Node** (Valid credentials?): Checks if credentials are valid.
   * If yes, it checks the user’s role.
   * If no, displays a login error and ends.
4. **Role Decision**:
   * **Admin**: Allows access to manage student details, view results, generate report cards, and send notifications.
   * **Teacher**: Can enter/edit grades, view results, and generate report cards.
   * **Student**: Only allowed to view results.
5. **End** node: Marks the end of the activity.



**Result** : The Activity diagram has been created successful following the steps given.

**7. STATE CHART DIAGRAM**

**Aim :**

To Draw the State Chart Diagram for STUDENT RESULT MANAGEMENT SYSTEM.

**Algorithm :**

1**. Identify Important Objects**

Library User, Book, Librarian, System.

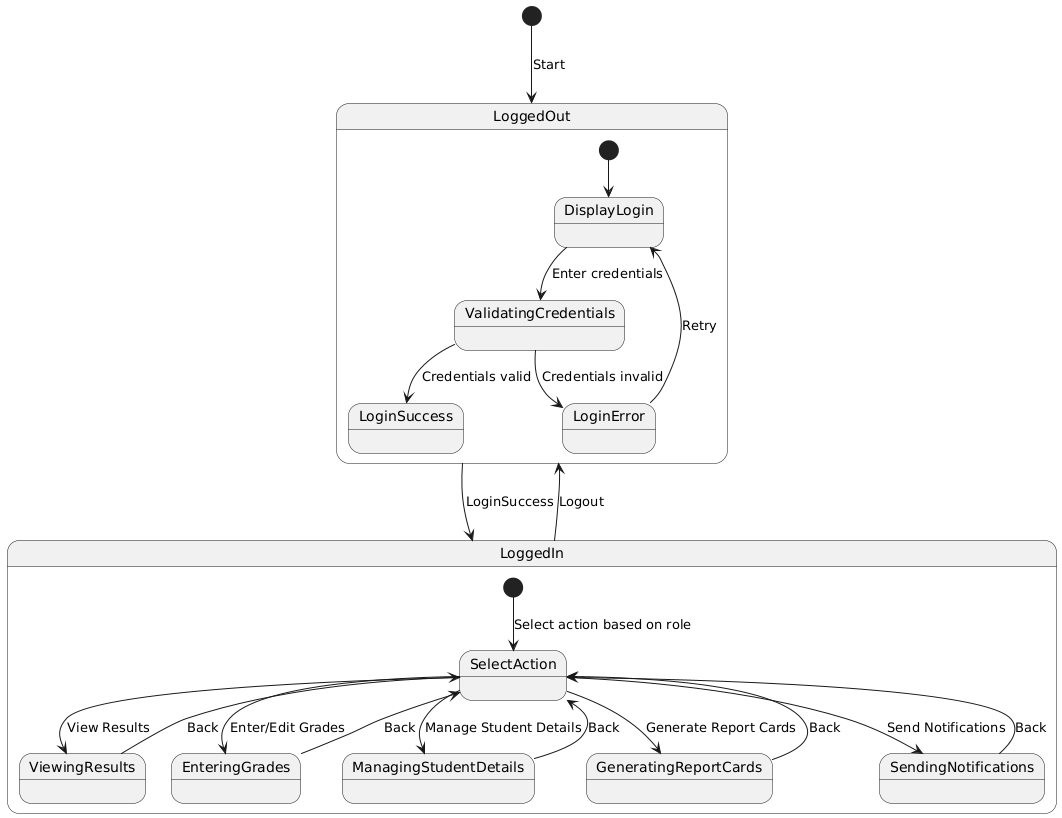
2. **Identify the States**

Logged Out, Logged In, Searching Book, Issuing Book, Returning Book, Renewing Book, Paying Fines.

**3. Identify the Events**

User Login, Book Search, Book Issue, Book Return, Renewal Request, Fine Payment.

**Sample Output :**



**Result** : The State Chart diagram has been created successfully by following the steps given.

**8. SEQUENCE DIAGRAM**

**Aim :**

To Draw the Sequence Diagram for STUDENT RESULT MANAGEMENT SYSTEM.

**ALGORITHM:**

**1. Identify the Scenario**

A student checks their results using the SRMS.

**2. List the Participants**

**Student,System (SRMS**),**Database**

**3. Define Lifelines**

**Student Lifeline**,**System Lifeline**,**Database Lifeline**

**4. Arrange Lifelines**

Place them in this order: **Student → System → Database**.

**5. Add Activation Bars**

Student,System.,Database.

**6. Draw Messages**

1. **Student → System**: Sends login request (e.g., "Enter credentials").
2. **System → Database**: Validates credentials ("Check student credentials").
3. **Database → System**: Returns validation result ("Valid/Invalid").
4. **System → Student**: Confirms login success or failure.
5. **Student → System**: Requests results ("View Results").
6. **System → Database**: Fetches result data ("Fetch result for Student ID").
7. **Database → System**: Sends result details.
8. **System → Student**: Displays results.

**7. Include Return Messages**

* Dashed arrows for responses

**8. Indicate Timing and Order**

Ensure messages are ordered sequentially from top to bottom.

**9. Include Conditions and Loops**

* Add conditions:

**10. Consider Parallel Execution**

* Parallel actions can be included if results are fetched for multiple courses simultaneously.

**11. Review and Refine**

Ensure all interactions align with the SRMS requirements.

**12. Add Annotations and Comments**

For example:

* Comment next to **Login Validation**: "This step ensures secure access."

**13. Document Assumptions and Constraints**

* Assumption: Students have unique credentials.
* Constraint: Database must be updated with the latest results.

**14. Use a Tool for Finalization**

Create the final sequence diagram using tools like **Lucidchart**, **Draw.io**, or **Microsoft Visio**.

**INPUT**

1. Login Credentials:

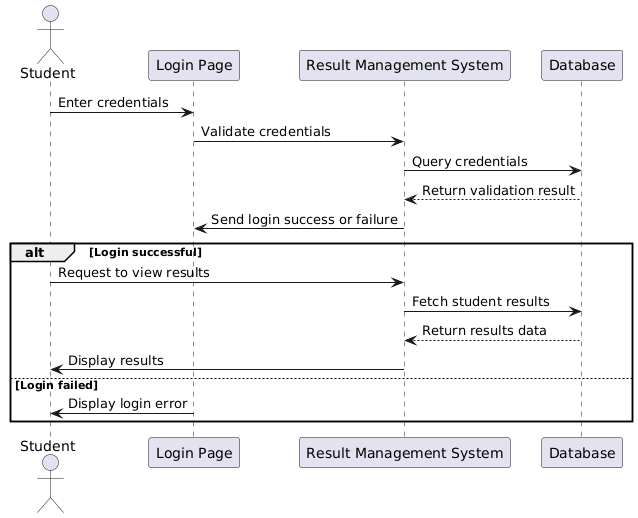
2. Student ID or Name (when managing or viewing specific student details):

3. Grades and Course Details (for teachers entering grades):

4. Term/Semester (when viewing or generating report cards):

5. Notification Details (for admins sending notifications):

**Sample Output :**



**Result** :

The Sequence diagram has been created successfully by following the steps given

**9. COLLABORATION DIAGRAM**

**Aim :**

To Draw the Collaboration Diagram forSTUDENT RESULT MANAGEMENT SYSTEM.

**.**

**Algorithm :**

**1. Identify Objects/Participants**

* **Student**: Requests to view results.
* **System (SRMS)**: Processes the request and interacts with the database.
* **Database**: Stores and provides student data and results.

**2. Define Interactions**

* **Student**: Logs in and requests to view results.
* **System**: Validates credentials, retrieves result data from the database, and displays results.
* **Database**: Responds to the system with student results.

**3. Add Messages**

**Messages Exchanged:**

1. **Student → System**: *"Enter credentials to log in."*
2. **System → Database**: *"Validate credentials for Student ID."*
3. **Database → System**: *"Validation success/failure."*
4. **System → Student**: *"Login success/failure message."*
5. **Student → System**: *"Request to view results."*
6. **System → Database**: *"Fetch results for Student ID."*
7. **Database → System**: *"Send result data."*
8. **System → Student**: *"Display result information."*

**4. Consider Relationships**

* **Student ↔ System**: Direct interaction for login, result requests, and responses.
* **System ↔ Database**: Backend interaction to validate and fetch data.

**5. Document the Collaboration Diagram**

A **Collaboration Diagram** shows:

* Objects (Student, System, Database) as rectangles.
* Arrows between objects represent the sequence of interactions.
* Messages labeled on arrows, numbered to indicate order.

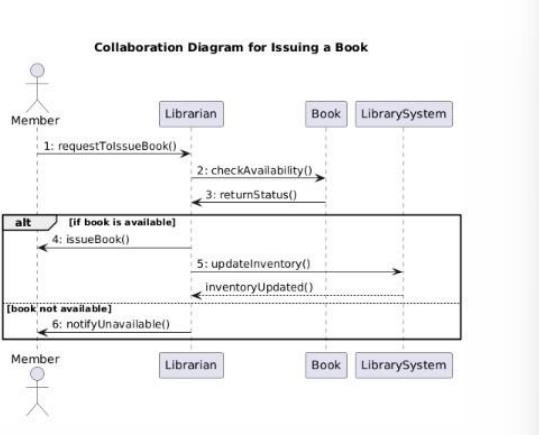
**System → Student**: "Display results."

**Inputs :**

Objects taking part in the interaction. Message flows among the objects.

The sequence in which the messages are flowing. Object organization.

**Sample Output :**



**Result** :

The Collaboration diagram has been created successfully by following the steps given.

**10. CLASS DIAGRAM**

**Aim:**

To Draw the Class Diagram for STUDENT RESULT MANAGEMENT SYSTEM.

**Algorithm :**

#  Identify the Classes

# Determine the main objects/entities (e.g., Student, Course, Result, Faculty).

# Classes typically represent nouns in the system.

#  Define Class Attributes

# List the properties (fields) of each class based on the information it needs to store. For example, the Student class might have attributes like studentID, name, email.

#  Define Class Methods

# Identify the actions (functions) each class should perform. For example, the Student class might have methods like viewResults().

#  Establish Relationships

# Define the relationships between classes:

# Association: One class uses or interacts with another.

# Aggregation: One class contains another, but both can exist independently.

# Composition: One class is part of another and cannot exist without it.

# Inheritance: One class inherits from another (e.g., UndergraduateStudent inherits from Student).

#  Add Multiplicity

# Specify how many objects of one class relate to objects of another class.

# 1:1 (One-to-One)

# 1:N (One-to-Many)

# N:M (Many-to-Many)

#  Organize Class Diagram Layout

# Place classes in a logical arrangement to make relationships clear.

# Group related classes together.

#  Add Access Modifiers

# Indicate the visibility of attributes and methods using:

# + for public.

# - for private.

# # for protected.

#  Iterate and Refine

# Validate the diagram against system requirements.

# Ensure all use cases and data relationships are represented.

#  Use a Diagramming Tool

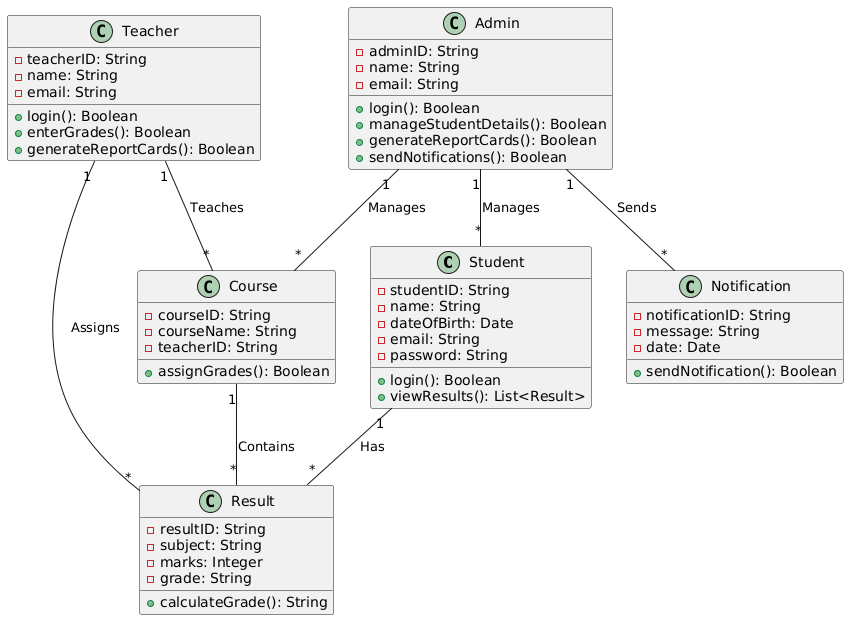
# Create the diagram using tools like Lucidchart, Draw.io, Visual Paradigm, or Microsoft Visio.

# 

# Inputs :

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

**Sample Output :**



# Result:

The Class diagram has been created successfully by following the steps given.

**CODE FOR MINI PROJECT**

**CODE:**

**IMPLEMENTATION CODE USING PYTHON**

# Defining the Student class

class Student:

def \_\_init\_\_(self, student\_id, name, date\_of\_birth, email, password):

self.student\_id = student\_id

self.name = name

self.date\_of\_birth = date\_of\_birth

self.email = email

self.password = password

self.results = []

def login(self, username, password):

# In a real-world scenario, this would check credentials against a database.

if username == self.email and password == self.password:

return True

else:

return False

def view\_results(self):

return self.results

def add\_result(self, result):

self.results.append(result)

# Defining the Teacher class

class Teacher:

def \_\_init\_\_(self, teacher\_id, name, email):

self.teacher\_id = teacher\_id

self.name = name

self.email = email

def login(self, username, password):

# In a real-world scenario, this would check credentials against a database.

if username == self.email and password == "teacher\_password": # Placeholder password

return True

return False

def enter\_grades(self, student, subject, marks):

result = Result(subject, marks)

student.add\_result(result)

def generate\_report\_cards(self, student):

return student.view\_results()

# Defining the Admin class

class Admin:

def \_\_init\_\_(self, admin\_id, name, email):

self.admin\_id = admin\_id

self.name = name

self.email = email

def login(self, username, password):

# In a real-world scenario, this would check credentials against a database.

if username == self.email and password == "admin\_password": # Placeholder password

return True

return False

def manage\_student\_details(self, student, new\_details):

# Admin can change student details (like name, email, etc.)

student.name = new\_details.get('name', student.name)

student.email = new\_details.get('email', student.email)

def send\_notifications(self, student, message):

print(f"Notification to {student.name}: {message}")

# Defining the Result class

class Result:

def \_\_init\_\_(self, subject, marks):

self.subject = subject

self.marks = marks

self.grade = self.calculate\_grade()

def calculate\_grade(self):

if self.marks >= 90:

return 'A'

elif self.marks >= 75:

return 'B'

elif self.marks >= 60:

return 'C'

else:

return 'F'

# Defining the Course class

class Course:

def \_\_init\_\_(self, course\_id, course\_name, teacher\_id):

self.course\_id = course\_id

self.course\_name = course\_name

self.teacher\_id = teacher\_id

def assign\_grades(self, student, marks):

# In a real system, grades would be assigned after course completion

result = Result(self.course\_name, marks)

student.add\_result(result)

# Defining the Notification class

class Notification:

def \_\_init\_\_(self, notification\_id, message, date):

self.notification\_id = notification\_id

self.message = message

self.date = date

def send\_notification(self, student):

print(f"Notification to {student.name} on {self.date}: {self.message}")

# Example Usage:

# Creating instances for testing

student1 = Student("S123", "John Doe", "2000-01-01", "john@example.com", "password123")

teacher1 = Teacher("T001", "Mr. Smith", "teacher@example.com")

admin1 = Admin("A001", "Admin User", "admin@example.com")

# Login functionality for Student

if student1.login("john@example.com", "password123"):

print(f"{student1.name} logged in successfully.")

else:

print("Login failed.")

# Teacher assigning grades

if teacher1.login("teacher@example.com", "teacher\_password"):

teacher1.enter\_grades(student1, "Mathematics", 85)

print(f"Grades for {student1.name} in Mathematics have been entered.")

# Viewing student results

print(f"Results for {student1.name}:")

for result in student1.view\_results():

print(f"{result.subject}: {result.marks} - Grade: {result.grade}")

# Admin sending notifications

if admin1.login("admin@example.com", "admin\_password"):

admin1.send\_notifications(student1, "Your results are available.")

else:

print("Admin login failed.")

**OUTPUT**

John Doe logged in successfully.

Grades for John Doe in Mathematics have been entered.

Results for John Doe:

Mathematics: 85 - Grade: B

Notification to John Doe: Your results are available.