

RAJALAKSHMI ENGINEERING COLLEGE

**RAJALAKSHMI NAGAR, THANDALAM – 602 105**

**Laboratory Record Note Book**

**CB23332 SOFTWARE ENGINEERING LAB**

**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**

**1. PREPARING PROBLEM STATEMENT**

**Aim** **:** To prepare Problem Statement for Student Result Analysis Project.

**Algorithm** :

1. **Start**
2. **Input**:
   * Collect student details (ID, name, subjects, scores).
   * Collect teacher details and associated subjects.
   * Collect parent details and link to student IDs.
3. **Data Validation**:
   * Check if all required fields are filled.
   * Validate score ranges (e.g., 0-100).
4. **Data Processing**:
   * Calculate the total score and average score for each student.
   * Calculate the average score for each subject taught by staff.
5. **Report Generation**:
   * Generate student performance report.
   * Generate subject-wise average report for teachers.
   * Create parent engagement report.
   * Generate management overview report.
6. **Visualization**:
   * Use charts (bar graphs, pie charts) to represent data visually.
   * Provide downloadable report formats (PDF, Excel).
7. **Output**:
   * Display reports and visualizations.
   * Provide options to save or print reports.
8. **End**

**Objectives :**

1. **Generate Statistics Reports**: Provide comprehensive performance reports for students.
2. **Staff Performance Insights**: Allow teachers to see the average scores of their subjects.
3. **Parental Engagement**: Offer a platform for parents to track their children's academic performance.
4. **Management Overview**: Enable management to access aggregated data for decision-making.

**Requirements :**

**Functional Requirements:**

1. Input student data (name, ID, subjects, scores).
2. Validate and process the data.
3. Generate individual student performance reports.
4. Create subject-wise average score reports for teachers.
5. Provide a dashboard for parents to view their child’s performance.
6. Generate overall statistics for management.

**Non-Functional Requirements:**

1. User-friendly interface.
2. Secure data handling and storage.
3. Scalability to handle multiple users.
4. Performance efficiency for quick report generation.

**Problem Statement :**

Many educational institutions lack a streamlined method for analyzing student performance, leading to difficulties in tracking academic progress, identifying areas for improvement, and engaging parents and management effectively.

**Solutions :**

1. **Centralized Data Input**: Implement a simple interface for entering student data.
2. **Automated Validation**: Use automated checks to ensure data integrity.
3. **Dynamic Reporting**: Develop reporting features that update in real-time.
4. **User Access Control**: Different user roles (students, teachers, parents, management) with varying access levels to reports.

**Sample Input :**

Using the database where student details, course details, their grades, and all requirement data are stored and given as input for this result analysis system.

**Sample Output :**

By using the given database using SQL, Java, Python and R the report card or final report on the academic performance of the student, course average and all other statistical report is generated using this system which helps to study & improve the education performance in overall college.

**Result :**

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

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

**Aim :**

To do requirement analysis and develop Software Requirement Specification Sheet

(SRS) for Student Result Analysis Project.

**Algorithm :**

1. **Input**: Collect student details, scores, teacher information, and parent data.

2. **Data Validation**: Check for completeness and correctness of input data.

3. **Data Processing**:

* Calculate total scores for each student.
* Calculate average scores for each subject.

4. **Report Generation**:

* Create performance reports for each student.
* Generate average reports for teachers based on subject performance.
* Prepare parental engagement reports.

1. **Output**: Display reports and visualizations to the respective users.

## 1. Introduction

### 1.1 Purpose

The purpose of this document is to outline the requirements for the Student Result Analysis System (SRAS), which aims to streamline the process of analyzing student performance, providing insights for students, parents, teachers, and management.

### 1.2 Scope

The SRAS will allow:

* Students to view their performance reports.
* Teachers to access average scores for their subjects.
* Parents to track their children's academic progress.
* Management to gain insights into overall academic performance across the institution.

### 1.3 Definitions, Acronyms, and Abbreviations

* **SRAS**: Student Result Analysis System
* **SRS**: Software Requirements Specification
* **UI**: User Interface

### 1.4 Overview

This document will describe the functional and non-functional requirements, algorithms, features, and interface requirements of the SRAS.

## 2. Overall Description

### 2.1 Product Perspective

The SRAS is a standalone web-based application that integrates with the existing student management systems to gather data, process it, and present it in a user-friendly manner.

### 2.2 Product Functions

* Input student, teacher, and parent data.
* Validate and process scores.
* Generate performance reports for students.
* Generate subject average reports for teachers.
* Provide parental access to their children's academic records.
* Allow management to view aggregated data for decision-making.

### 2.3 User Classes and Characteristics

* **Students**: Can view their performance reports.
* **Teachers**: Can access average scores for their subjects and view class performance.
* **Parents**: Can monitor their child's academic performance.
* **Management**: Can view overall statistics and reports.

### 2.4 Operating Environment

The application will operate in a web environment, accessible from modern browsers on various devices (desktop, tablets, smartphones).

## 3. System Features

### 3.1 User Input

* Users can input student data, including names, IDs, subjects, and scores.
* Teachers can input their information, including subjects taught.
* Parents can register and link to their children’s profiles.

### 3.2 Data Validation

* The system will validate the input data to ensure that all required fields are filled correctly.
* Scores must be within defined ranges (e.g., 0 to 100).

### 3.3 Report Generation

* The system will generate individual student performance reports, including total and average scores.
* Teachers will receive subject average reports.
* Parents will receive engagement reports summarizing their child’s performance.

### 3.4 Visualization

* The application will provide visual representations of data, such as charts and graphs.

## 4. External Interface Requirements

### 4.1 User Interfaces

* **Student Dashboard**: Displays individual performance reports and visual statistics.
* **Teacher Dashboard**: Shows average scores and class performance insights.
* **Parent Portal**: Allows parents to view their child’s academic records.
* **Management Overview**: Provides access to aggregated statistics.

### 4.2 Hardware Interfaces

* The system will operate on standard web servers and can be accessed from any device with an internet connection.

### 4.3 Software Interfaces

* The application will interface with existing student management systems for data retrieval and integration.

### 4.4 Communications Interfaces

* The application will use HTTPS for secure communication between the client and server.

**5. Nonfunctional Requirements**

**5.1 Performance Requirements**

* The system must generate reports and statistical data within 5 seconds.
  1. **Safety Requirements**
* Ensure data integrity with regular backups.

**5.3 Security Requirements**

* Implement user authentication and data encryption.

**5.4** **Software Quality Attributes**

* The system must be reliable, maintainable, and user-friendly.

**Result :**

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

**3. ENTITY RELATIONSHIP MODEL**

**Aim :**

To Draw the Entity Relationship Diagram for any project.

**Algorithm** :

Step 1: **Define Purpose**: Determine the scope of the ERD for your project.

Step 2: **Identify Entities**: List key entities (e.g., Student, Course, Grade, User).

Step 3: **Define Attributes**: Specify attributes for each entity (e.g., Student: student\_id, name).

Step 4: **Identify Relationships**: Determine how entities relate (e.g., Student enrolls in Course).

Step 5: **Determine Cardinality**: Define cardinality (e.g., Many-to-Many between Student and Course).

Step 6: **Relationship Attributes**: Identify any attributes for relationships if needed.

Step 7: **Draw Diagram**: Use a tool to visually represent entities and their relationships.

Step 8: **Review**: Validate the ERD with stakeholders for accuracy.

Step 9: **Revise**: Make necessary adjustments based on feedback.

Step 10: **Finalize**: Complete the ERD for inclusion in project documentation.

**Input :**

Entities - Student, Course, Grade, User

Entity Relationship Matrix - Student, Course, Grade, User

Primary Keys - student\_id, course\_id, grade\_id, user\_id

Attributes –

Student: name, email, date\_of\_birth

Course: course\_name, credits, department

Grade: student\_id, course\_id, grade

User: username, password, role

Mapping of Attributes with Entities -

Student: student\_id, name, email, date\_of\_birth

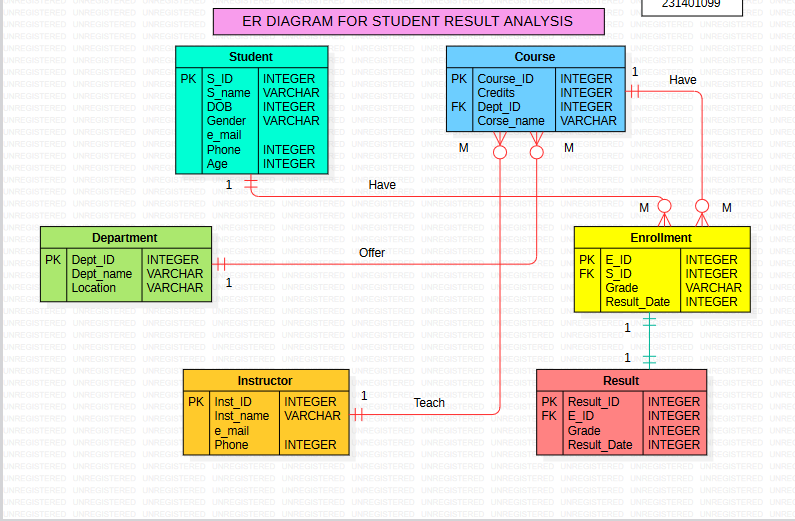
Course: course\_id, course\_name, credits, department

Grade: grade\_id, student\_id, course\_id, grade

User: user\_id, username, password, role

**Sample Output :**

**The Fully Attributed ERD**

****

## 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 Analysis Project and List the Modules in the Application.

**Algorithm :**

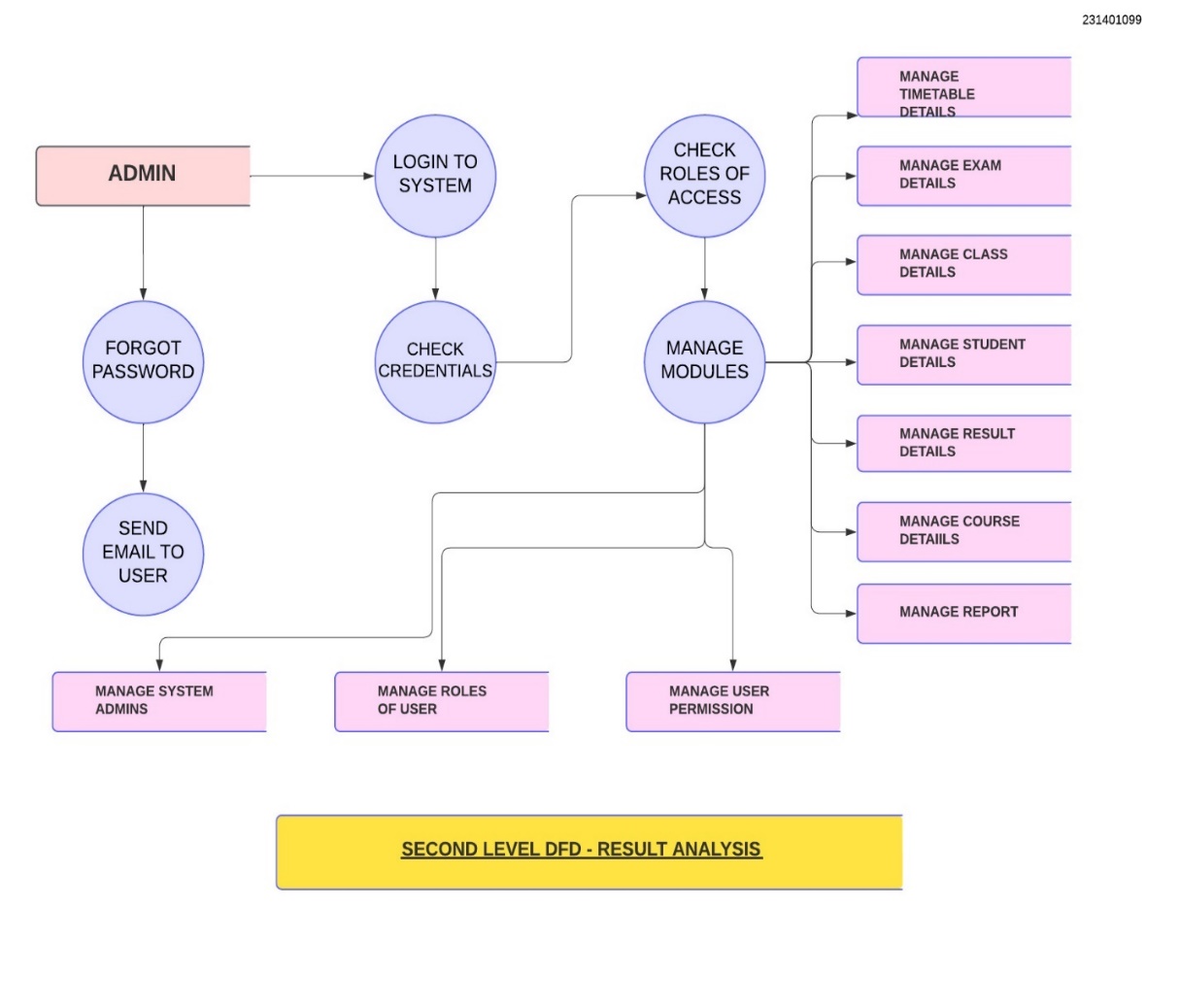
1. **Define Purpose**: Identify the system to model (e.g., Student Result Analysis System).
2. **Identify External Entities**: List entities interacting with the system (e.g., Students, Teachers).
3. **Create Level 0 DFD**: Draw the system as a single process bubble.
4. **Identify Data Flows**: Connect external entities to the process with labeled arrows (e.g., "Student Data").
5. **Review Level 0 DFD**: Ensure all interactions are captured correctly.
6. **Create Level 1 DFD**: Break down the main process into sub-processes (e.g., Data Entry, Reporting).
7. **Draw Sub-processes**: Create separate bubbles for each sub-process.
8. **Identify Data Stores**: Add data stores (e.g., Student Database) linked to sub-processes.
9. **Define Data Flows**: Connect sub-processes, data stores, and external entities with labeled arrows.
10. **Finalize DFDs**: Ensure clarity and accuracy; use diagramming tools for presentation.

**Input :**

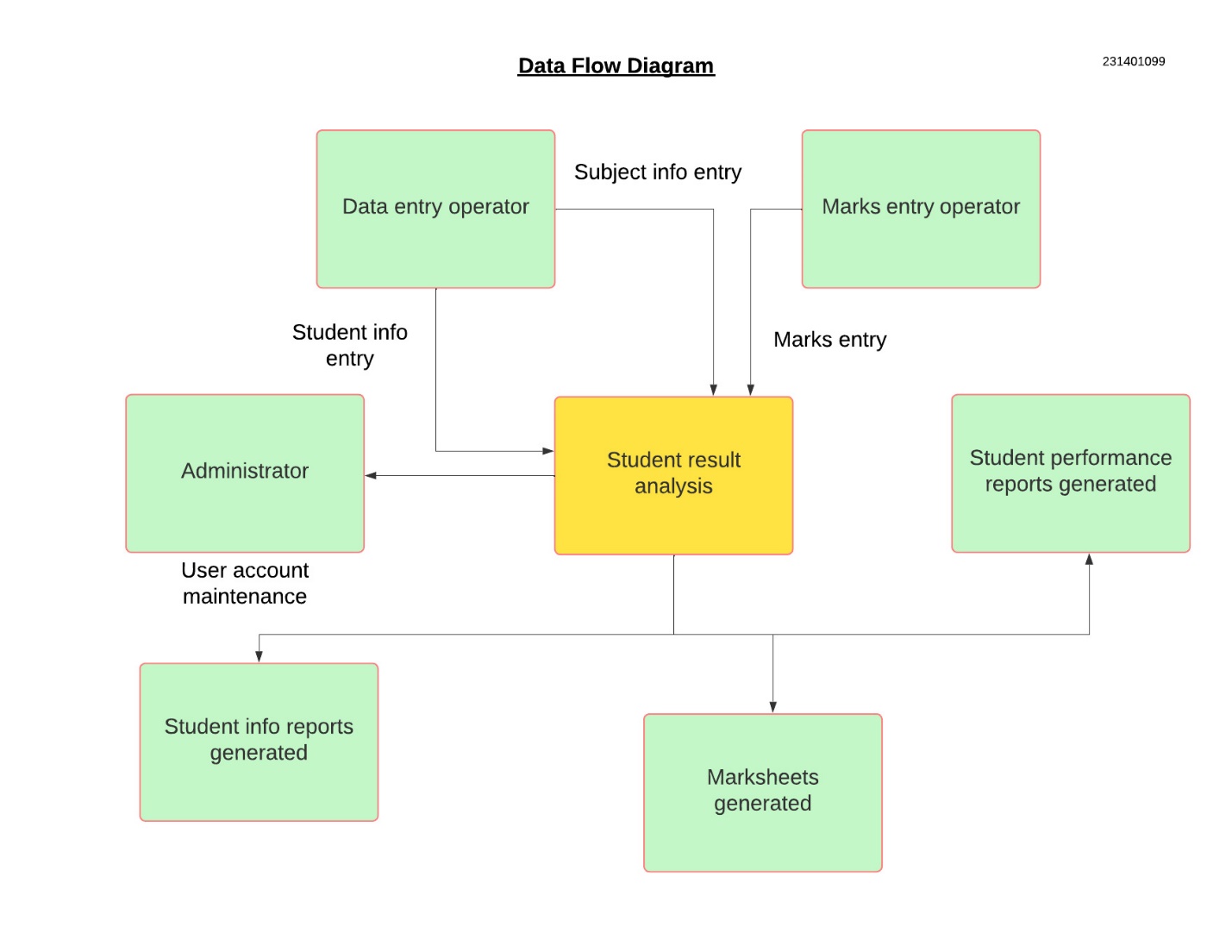
Processes

Datastores

External Entities



**Sample Output :**



**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 Analysis Project.

**Algorithm :**

1. **Identify the System**: Define the system you are modeling (e.g., Student Result Analysis System).
2. **Identify Actors**: List all external users or systems that interact with the system (e.g., Students, Teachers, Admin).
3. **Identify Use Cases**: Define the key functionalities the system provides (e.g., View Grades, Generate Reports).
4. **Draw the System Boundary**: Create a box to represent the system and label it with the system name.
5. **Place Actors**: Position actors outside the system boundary, connected to relevant use cases.
6. **Connect Actors to Use Cases**: Draw lines to show interactions between actors and use cases.
7. **Review and Finalize**: Ensure all actors and use cases are accurately represented and clear; refine as needed.

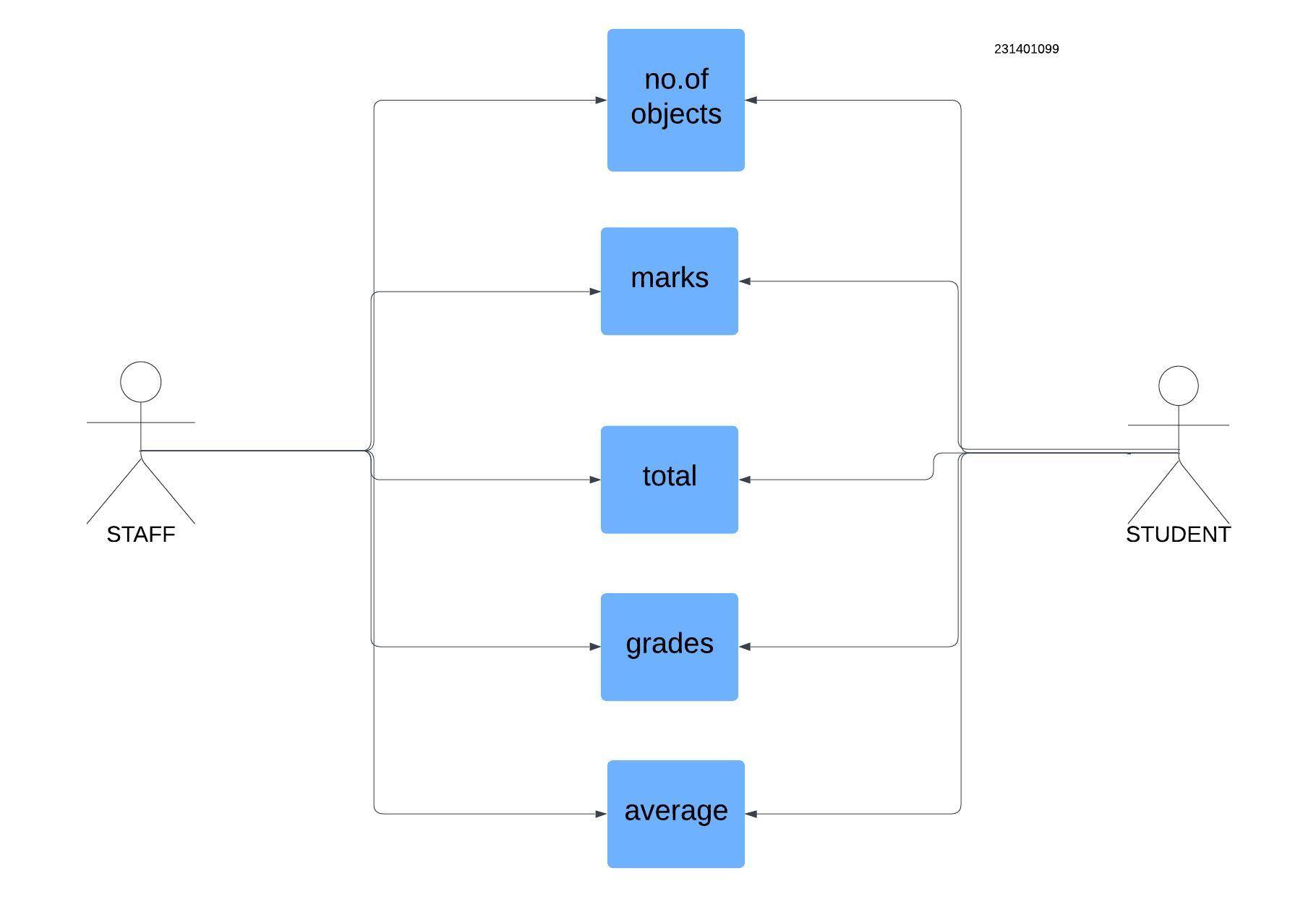
**Input :**

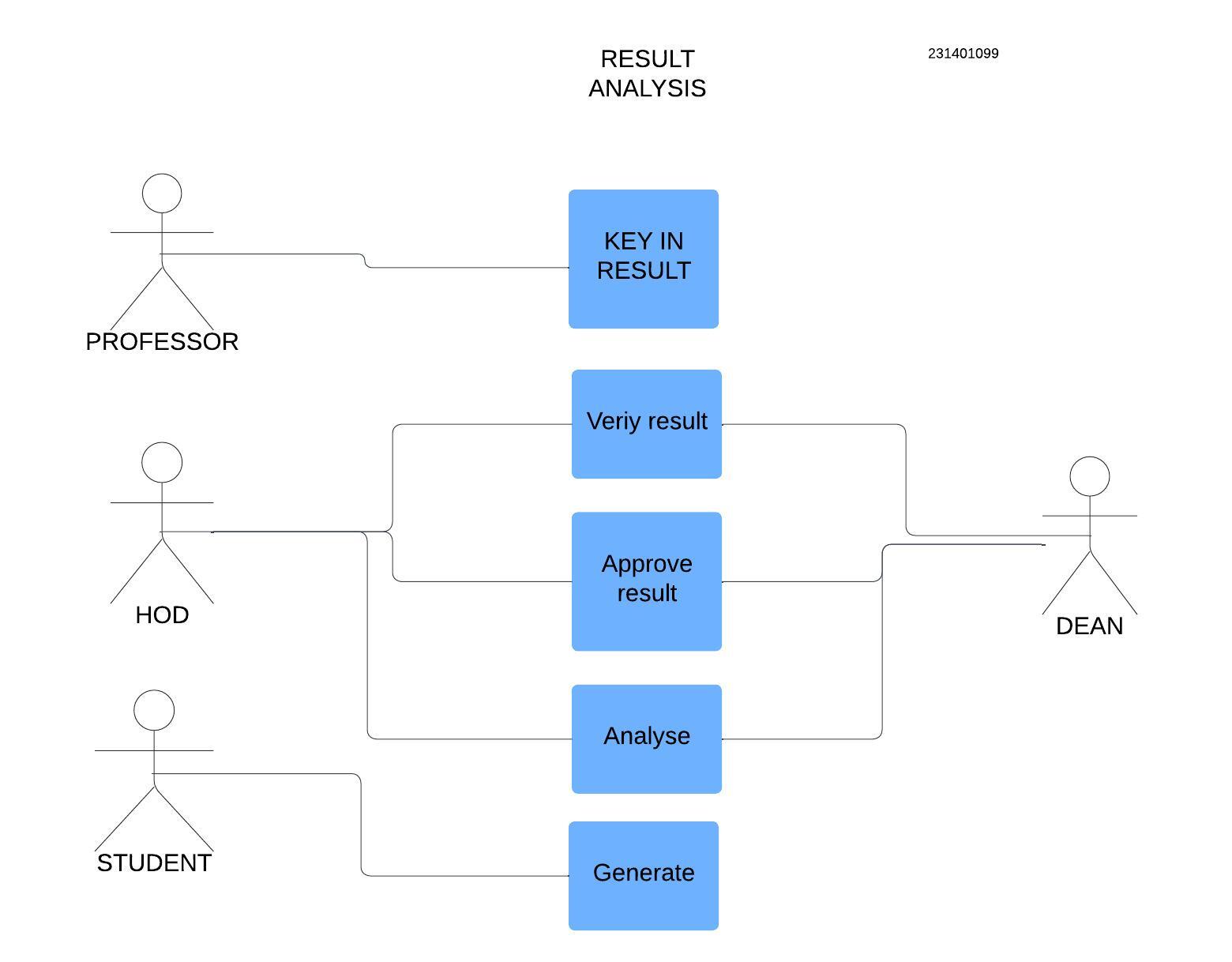
Actors

Use Cases

Relations

**Sample Output :**





**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 Analysis Project.

**Algorithm :**

1. **Identify the Use Cases**: List all relevant use cases from your Use Case Diagram (e.g., View Grades, Generate Reports).
2. **Define Activities**: Break down each use case into specific activities or steps (e.g., Log In, Select Course, View Grades).
3. **Determine Flow**: Establish the sequence of activities, including decisions and parallel processes.
4. **Draw the Diagram**: Use standard symbols (ovals for start/end, rectangles for activities, diamonds for decisions) to represent the flow of activities.
5. **Connect Activities**: Use arrows to indicate the flow between activities and decisions.
6. **Review and Finalize**: Ensure clarity and completeness, making adjustments as necessary.

**Inputs :**

Activities

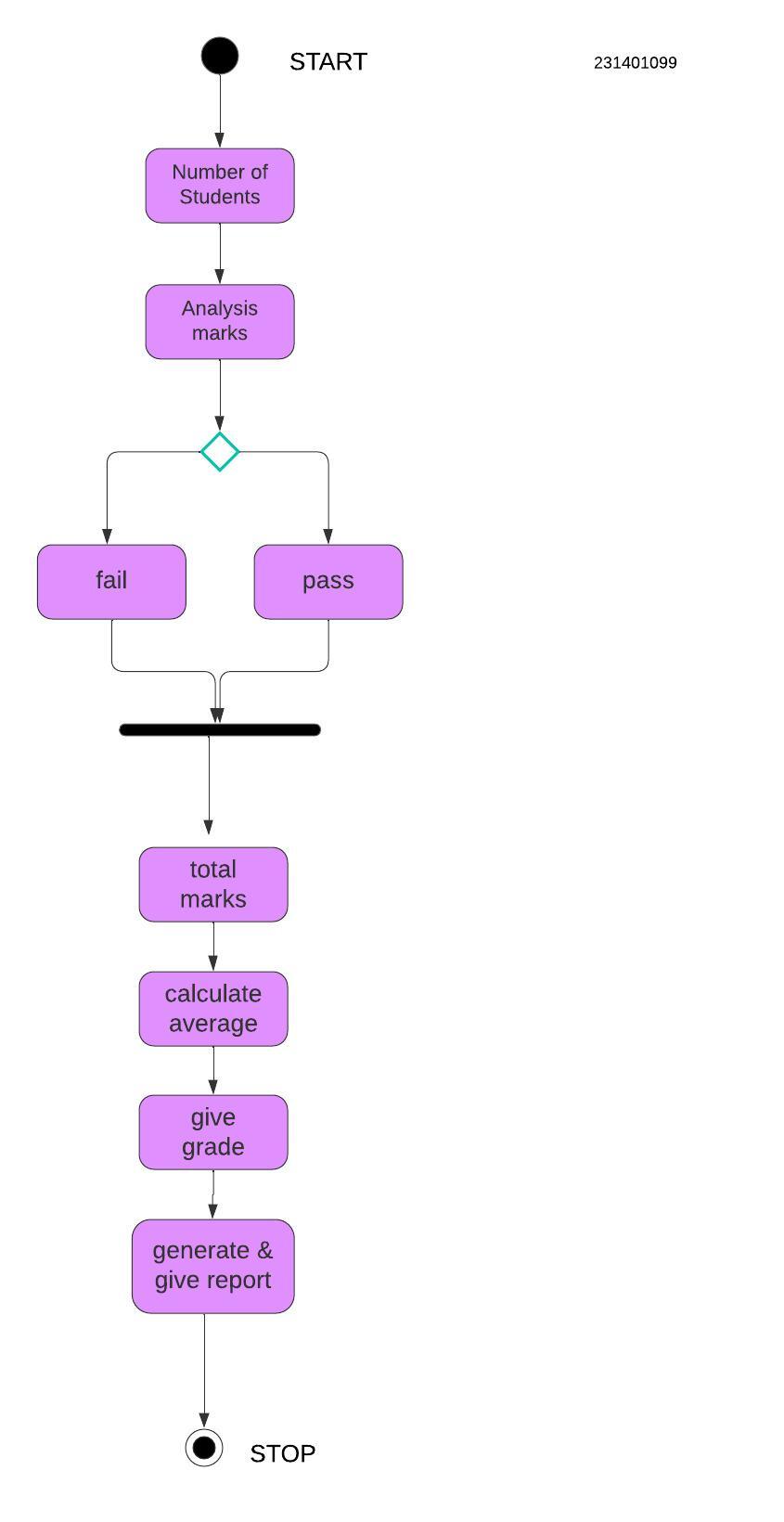
Decision Points

Guards

Parallel Activities

Conditions

**Sample Output :**



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

**7. STATE CHART DIAGRAM**

**Aim :**

To Draw the State Chart Diagram for Student Result Analysis Project.

**Algorithm :**

1. **Identify the Stateful Objects**: Determine the objects or entities that will have states (e.g., Student, Course).
2. **Define States**: List all possible states for each object (e.g., Enrolled, Graduated, Inactive for Student).
3. **Identify Transitions**: Determine events that trigger state changes (e.g., Enrollment, Withdrawal, Graduation).
4. **Draw the Diagram**: Use standard symbols (rounded rectangles for states, arrows for transitions) to represent states and transitions.
5. **Label Events**: Clearly label the arrows with the events that cause transitions between states.
6. **Review and Finalize**: Ensure all states and transitions are accurately represented, making adjustments as needed.

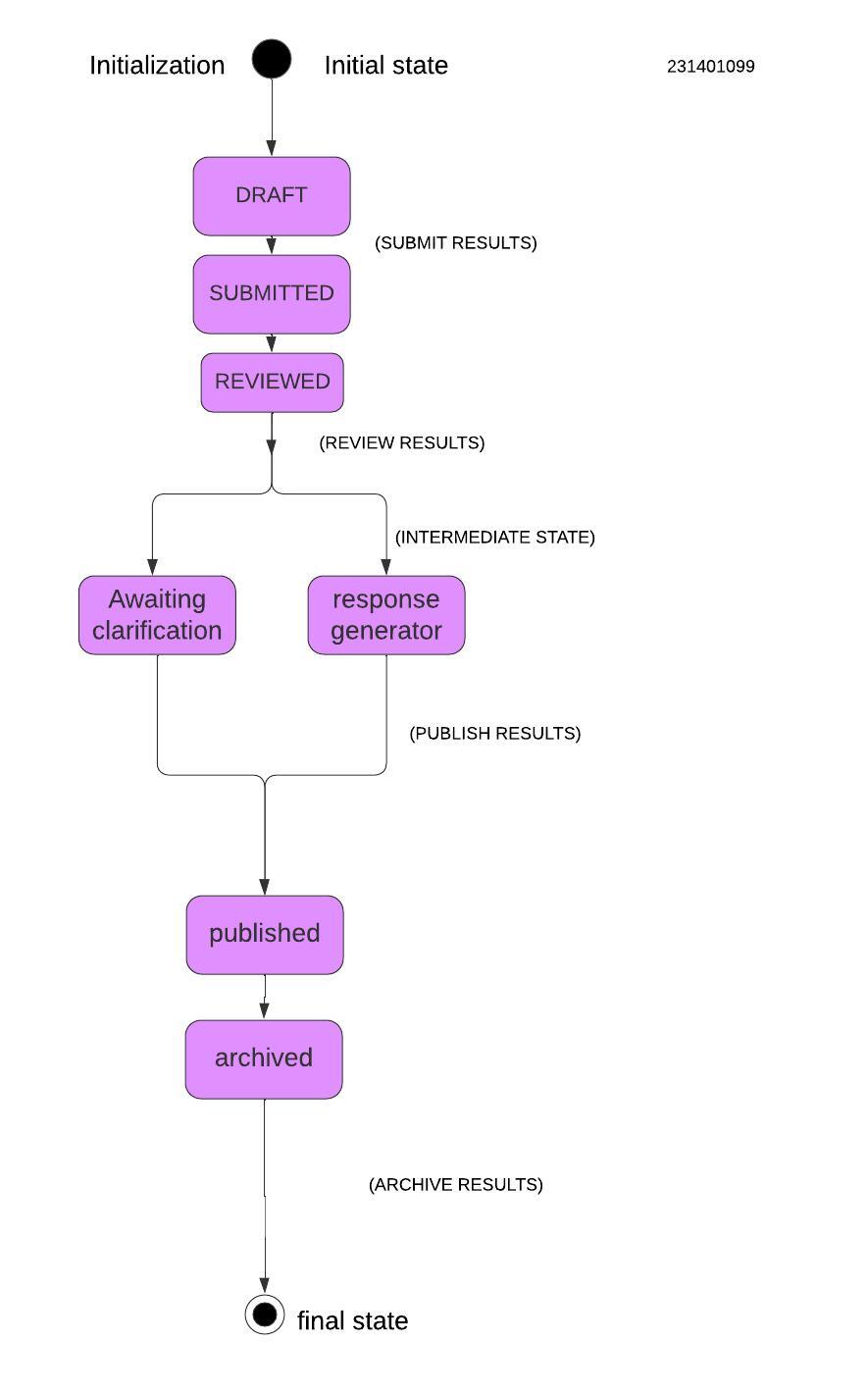
**Inputs :**

Objects

States

Events

**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 any project Student Result Analysis Project.

**Algorithm :**

1. **Identify Use Cases**: List all relevant use cases.
2. **Define Actors**: Identify interacting actors (e.g., Student, Teacher).
3. **Identify Objects**: List system objects involved in the use case.
4. **Draw Lifelines**: Create vertical dashed lines for each actor and object.
5. **Sequence Messages**: Arrange messages between actors and objects in chronological order.
6. **Indicate Activations**: Show activations using rectangles on lifelines during interactions.
7. **Include Conditions**: Note any conditions for messages or actions.
8. **Represent Return Messages**: Indicate return messages with dashed arrows.
9. **Review Diagram**: Check for clarity and completeness.
10. **Finalize Diagram**: Make adjustments as necessary for accuracy.

**Inputs :**

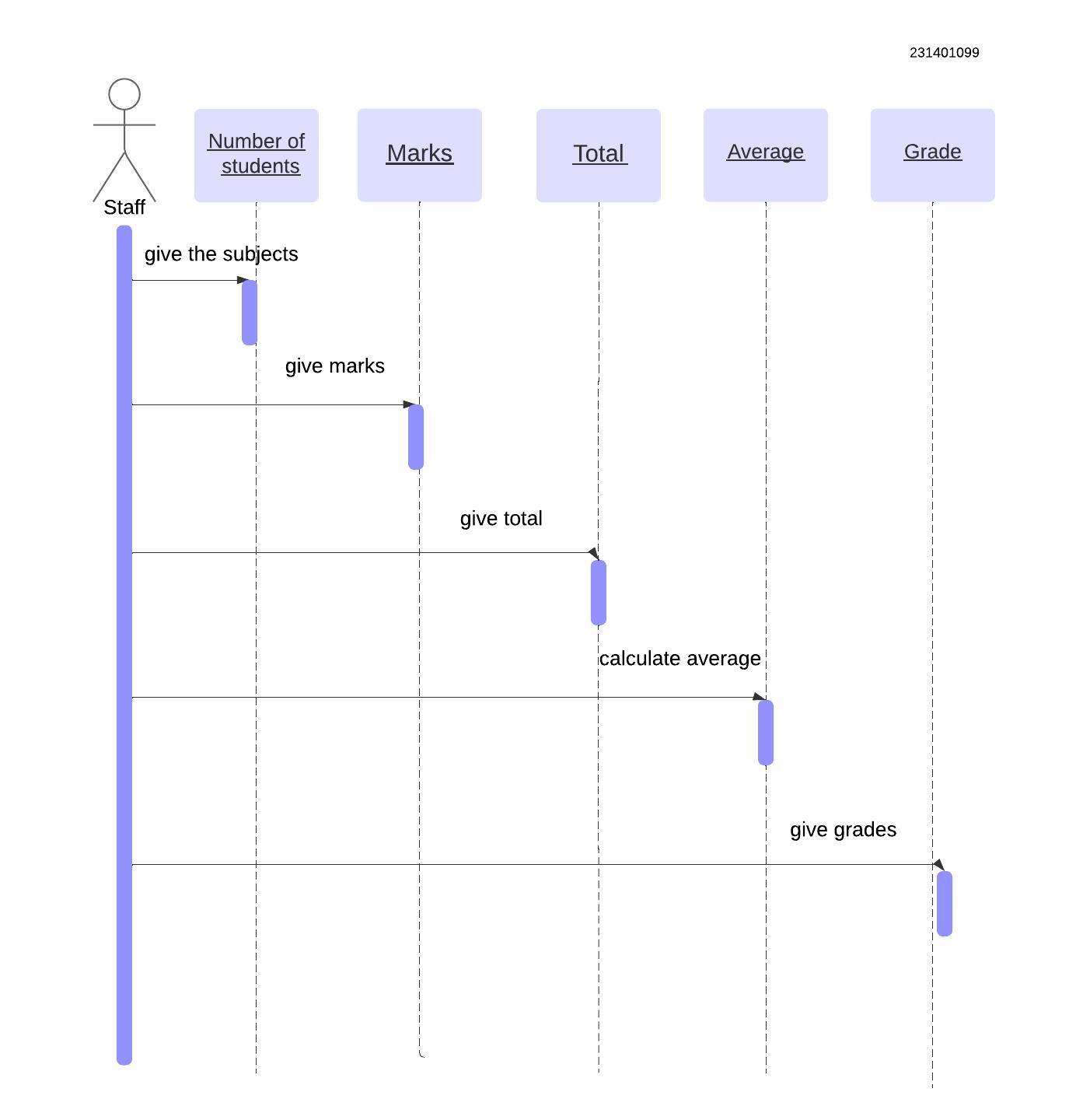
Objects taking part in the interaction.

Message flows among the objects.

The sequence in which the messages are flowing.

Object organization.

**Sample Output :**



**Result** :

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

**9. COLLABORATION DIAGRAM**

**Aim :**

To Draw the Collaboration Diagram for Student Result Analysis Project.

**Algorithm :**

1. **Identify Use Cases**: List all relevant use cases to be represented.
2. **Define Objects**: Identify the objects involved in each use case.
3. **Identify Actors**: List the actors interacting with the objects.
4. **Draw Objects and Actors**: Represent objects as boxes and actors as stick figures.
5. **Connect with Lines**: Use lines to show relationships between actors and objects.
6. **Label Messages**: Number and label the messages exchanged between objects.
7. **Review and Finalize**: Ensure clarity and accuracy, making adjustments as needed.

**Inputs :**

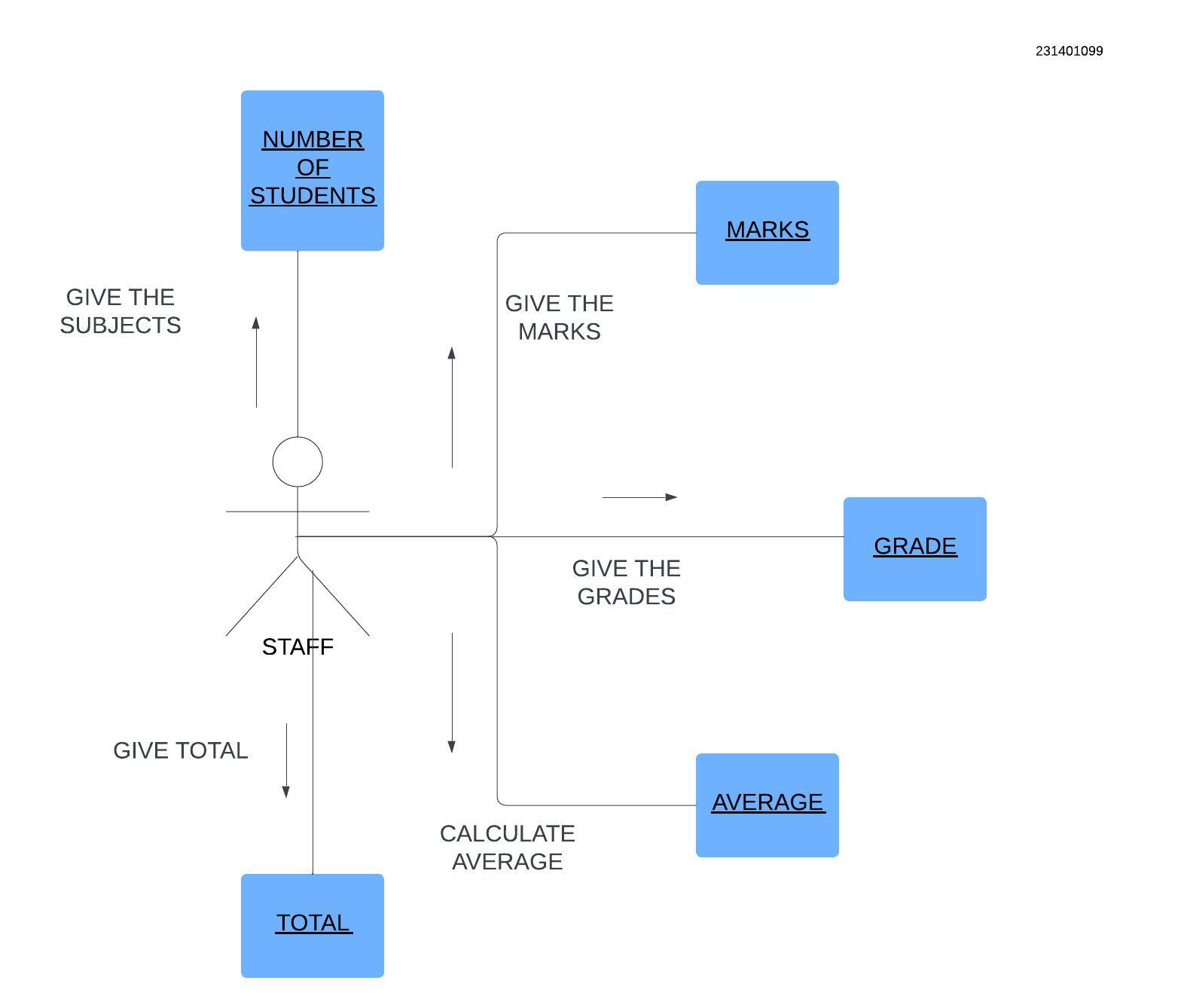
Objects taking part in the interaction.

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 Analysis Project.

**Algorithm :**

1. **Review Sequence Diagram**: Examine the sequence diagram for all objects and messages.
2. **Identify Classes**: Determine the classes that correspond to each object in the diagram.
3. **Define Class Attributes**: List relevant attributes for each identified class based on object properties.
4. **Define Class Methods**: Identify methods for each class based on messages exchanged in the sequence diagram.
5. **Draw Class Diagram**: Create a diagram with rectangles representing classes, including attributes and methods.
6. **Establish Relationships**: Indicate relationships (associations, inheritance) between classes using appropriate lines.
7. **Review and Finalize**: Ensure accuracy and completeness of the class diagram, making adjustments as necessary.

**Inputs :**

1. Class Name

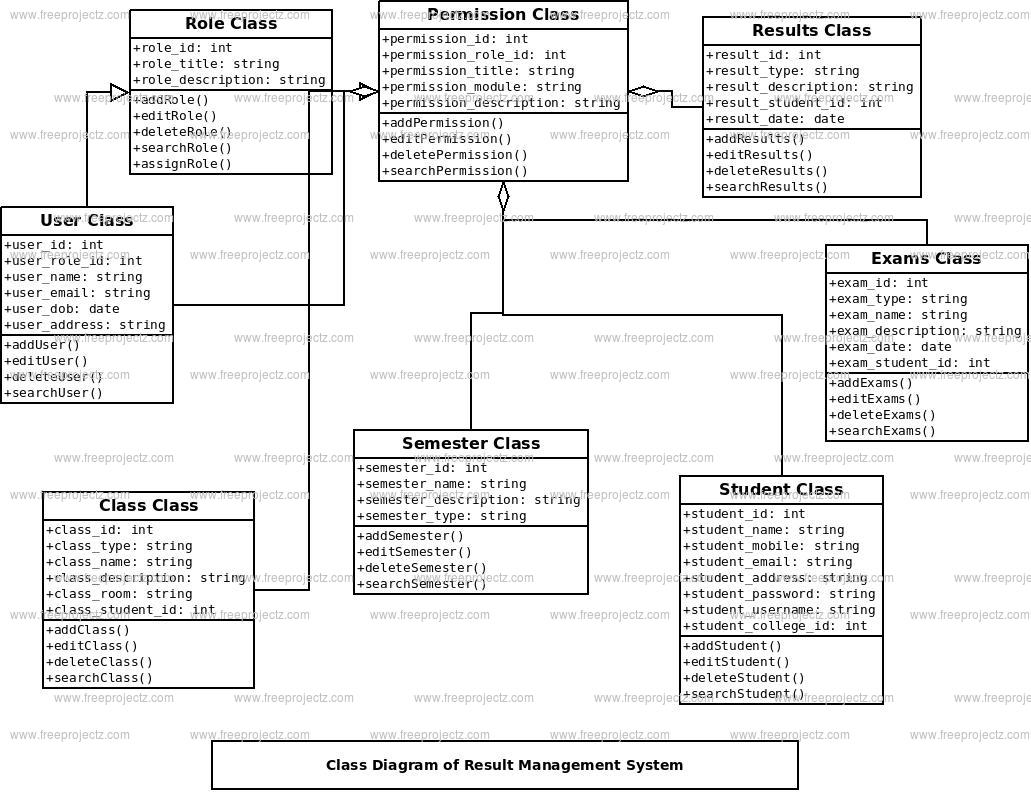
2. Attributes

3. Methods

4. Visibility Notation

**Sample Output :**

231401099



**Result:**

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

**CODE FOR MINI PROJECT**

**STUDENT RESULT ANALYSIS SYSTEM PROJECT**

**CODE:**

**Java (Using OOP concepts):**

class Student {

String studentID;

String name;

Map<String, Integer> subjectsScores; // Subject -> Score

public Student(String studentID, String name) {

this.studentID = studentID;

this.name = name;

this.subjectsScores = new HashMap<>();

}

public void addScore(String subject, int score) {

subjectsScores.put(subject, score);

}

public int getTotalScore() {

return subjectsScores.values().stream().mapToInt(Integer::intValue).sum();

}

public double getAverageScore() {

return getTotalScore() / subjectsScores.size();

}

}

class Teacher {

String teacherID;

String name;

List<String> subjectsTaught;

public Teacher(String teacherID, String name, List<String> subjectsTaught) {

this.teacherID = teacherID;

this.name = name;

this.subjectsTaught = subjectsTaught;

}

}

class Parent {

String parentID;

String name;

List<String> studentIDs;

public Parent(String parentID, String name, List<String> studentIDs) {

this.parentID = parentID;

this.name = name;

this.studentIDs = studentIDs;

}

}

**Python (OOP implementation):**

class Student:

def \_\_init\_\_(self, student\_id, name):

self.student\_id = student\_id

self.name = name

self.subject\_scores = {} # subject -> score

def add\_score(self, subject, score):

self.subject\_scores[subject] = score

def get\_total\_score(self):

return sum(self.subject\_scores.values())

def get\_average\_score(self):

return self.get\_total\_score() / len(self.subject\_scores)

class Teacher:

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

self.teacher\_id = teacher\_id

self.name = name

self.subjects\_taught = subjects\_taught

class Parent:

def \_\_init\_\_(self, parent\_id, name, student\_ids):

self.parent\_id = parent\_id

self.name = name

self.student\_ids = student\_ids

**2**. **Data Validation and Processing**

**CODE:**

**In this module, we validate the score range and calculate totals and averages.**

**Java:**

public class DataValidation {

public static boolean validateScore(int score) {

return score >= 0 && score <= 100;

}

public static void processStudentData(Student student) {

int totalScore = student.getTotalScore();

double averageScore = student.getAverageScore();

System.out.println("Total Score: " + totalScore);

System.out.println("Average Score: " + averageScore);

}

}

**Python :**

def validate\_score(score):

return 0 <= score <= 100

def process\_student\_data(student):

total\_score = student.get\_total\_score()

average\_score = student.get\_average\_score()

print(f"Total Score: {total\_score}")

print(f"Average Score: {average\_score}")

**3. Report Generation**

**CODE:**

**Reports include student performance, subject averages, and parent engagement.**

**Java:**

public class ReportGenerator {

public static void generateStudentReport(Student student) {

System.out.println("Student Report for: " + student.name);

student.subjectsScores.forEach((subject, score) -> {

System.out.println(subject + ": " + score);

});

System.out.println("Total Score: " + student.getTotalScore());

System.out.println("Average Score: " + student.getAverageScore());

}

}

**Python :**

def generate\_student\_report(student):

print(f"Student Report for: {student.name}")

for subject, score in student.subject\_scores.items():

print(f"{subject}: {score}")

print(f"Total Score: {student.get\_total\_score()}")

print(f"Average Score: {student.get\_average\_score()}")

**4.Visualization Using Charts**

**CODE:**

For generating charts, you can use libraries like **JFreeChart** in Java or **matplotlib** in Python.

**Python (Using matplotlib):**

import matplotlib.pyplot as plt

def visualize\_student\_performance(student):

subjects = list(student.subject\_scores.keys())

scores = list(student.subject\_scores.values())

plt.bar(subjects, scores)

plt.xlabel('Subjects')

plt.ylabel('Scores')

plt.title(f'{student.name} Performance')

plt.show()

**5. Final Integration and Execution**

**Next Steps:**

* Define a database to store student, teacher, and parent data (optional: you can use SQLite for Python and JDBC for Java).
* Set up authentication systems for different user roles.
* Implement a user interface (UI) using a framework like **Tkinter** for Python or **JavaFX** for Java, or a web-based solution using **Flask** (Python) or **Spring** (Java).

**STATISTICAL VISUAL DATA (ANALYSIS REPORT) :**

**CODE:**

**Install necessary libraries:**

“ pip install matplotlib seaborn pandas “

**Python :**

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

# Sample data for student marks in various subjects

data = {

'Student': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],

'Math': [85, 75, 95, 65, 80],

'Science': [78, 88, 82, 70, 85],

'English': [90, 60, 75, 85, 95],

'History': [88, 79, 85, 90, 75]

}

# Create DataFrame

df = pd.DataFrame(data)

# 1. Bar Chart: Student-wise performance

def plot\_bar\_chart(df):

df.set\_index('Student').plot(kind='bar', figsize=(10, 6))

plt.title('Student-wise Subject Performance')

plt.ylabel('Marks')

plt.xlabel('Student')

plt.show()

# 2. Pie Chart: Grade distribution for each student

def plot\_pie\_chart(df):

for student in df['Student']:

marks = df[df['Student'] == student].drop('Student', axis=1).values.flatten()

total\_marks = len(marks) \* 100 # Assuming each subject has a max score of 100

grade\_dist = [

sum(marks >= 90), # Grade A

sum((marks >= 75) & (marks < 90)), # Grade B

sum((marks >= 60) & (marks < 75)), # Grade C

sum(marks < 60) # Grade D

]

labels = ['Grade A (90-100)', 'Grade B (75-89)', 'Grade C (60-74)', 'Grade D (<60)']

plt.pie(grade\_dist, labels=labels, autopct='%1.1f%%', startangle=140)

plt.title(f'Grade Distribution for {student}')

plt.show()

# 3. Heatmap: Subject-wise performance for all students

def plot\_heatmap(df):

df\_heatmap = df.set\_index('Student')

plt.figure(figsize=(8, 6))

sns.heatmap(df\_heatmap, annot=True, cmap='coolwarm', linewidths=0.5)

plt.title('Subject-wise Performance Heatmap')

plt.show()

# Plot all visualizations

plot\_bar\_chart(df)

plot\_pie\_chart(df)

plot\_heatmap(df)