STUDENT MANAGEMENT SYSTEM

SUBMITED BY

Arshdeep palial	102216099
Jaskaran Singh	102216110
Tarun Bhatti	102216105
Preetinder singh kundi	102216125
Sehajdeep Singh	102216117

INDEX

Table of Contents

INTRODUCTION

In the dynamic landscape of educational institutions, the efficient management of student data, staff information, and administrative tasks is paramount for ensuring smooth operations. The advent of digital technologies has revolutionized traditional methods of managing educational institutions, ushering in an era where Student Management Systems (SMS) play a pivotal role in streamlining administrative processes. This introduction provides an overview of our proposed Student Management System, designed to cater to the diverse needs of administrators, staff, and students alike.

1. Background

Educational institutions, ranging from schools to universities, are characterized by their complex organizational structures involving administrators, teaching staff, and students. Traditionally, managing this complexity relied heavily on manual paperwork, leading to inefficiencies, errors, and delays. Recognizing the need for modernization, the development of Student Management Systems became imperative. These systems integrate various functionalities, such as student enrollment, attendance tracking, academic records management, and communication tools, into a centralized platform.

2. Purpose of the Student Management System

- The primary objective of our Student Management System is to provide a comprehensive solution for managing administrative tasks, facilitating

communication, and enhancing collaboration among stakeholders within the educational institution.

- Efficiently Manage Student Data: The system centralizes student information, including personal details, academic records, attendance, and course enrollment, enabling administrators to access accurate and up-to-date information at their fingertips.

3. Requirement Analysis

- In order to develop a robust Student Management System, a thorough analysis of requirements was conducted to identify the specific needs and functionalities desired by administrators and students. The following key requirements were identified:

- User Authentication and Authorization: The system must support user authentication to ensure secure access to sensitive information. Different user roles,

such as administrators, and students, require varying levels of access and permissions within the system.

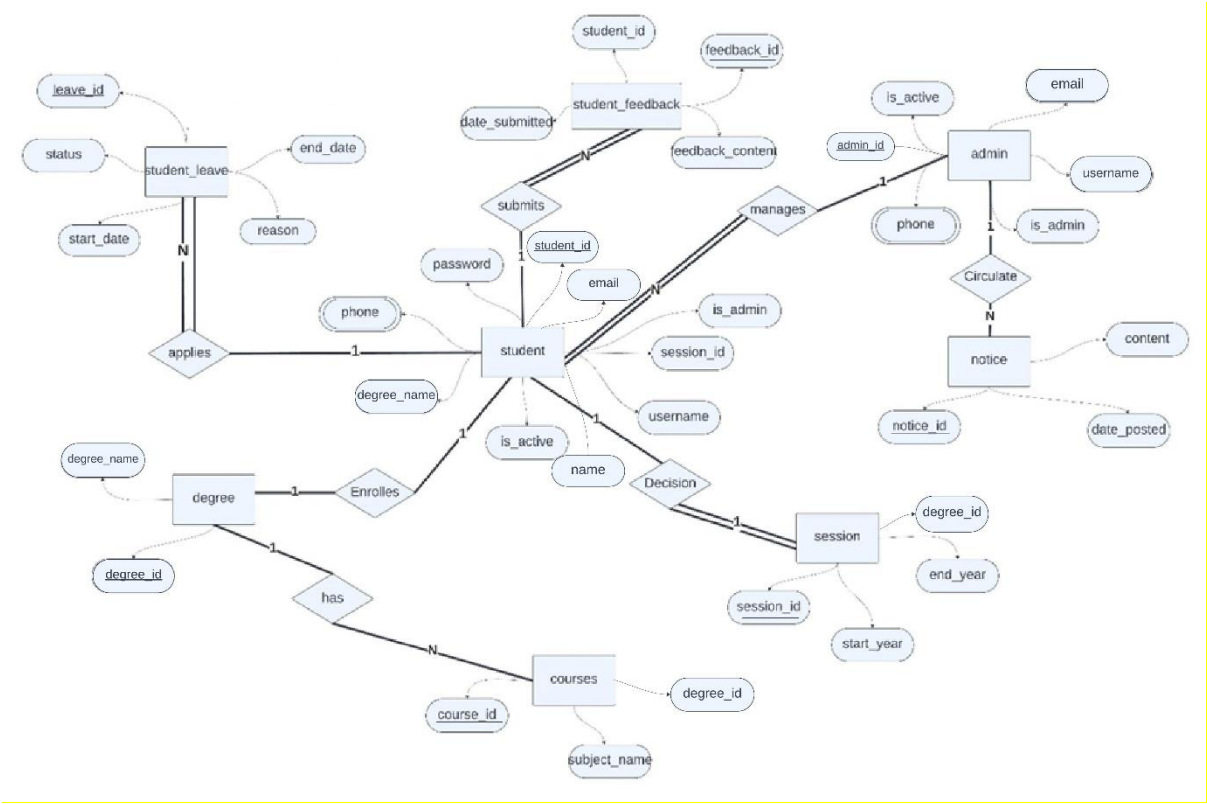
- Student Management: The system should allow administrators to manage student data, including personal details, academic records, course enrollment, and attendance tracking. Students should also have access to their own information, such as course schedules and academic progress.

- Communication Features: The system should facilitate communication among administrators, staff, and students through features such as messaging, feedback submission, and announcement dissemination. Administrators should be able to respond to feedback and manage leave requests.

- Course and Subject Management: Administrators should have the ability to add, update, and remove courses and subjects offered by the institution. Students should be

able to view available courses and subjects and select their preferences.

ER DIAGRAM



Student and **Student Leave**: One-to-Many.
A student can have many leaves (absences), but a leave record belongs to one student.

Student and **Student Feedback**: One-to-Many.
A student can submit many feedbacks, but a feedback record is submitted by one student.

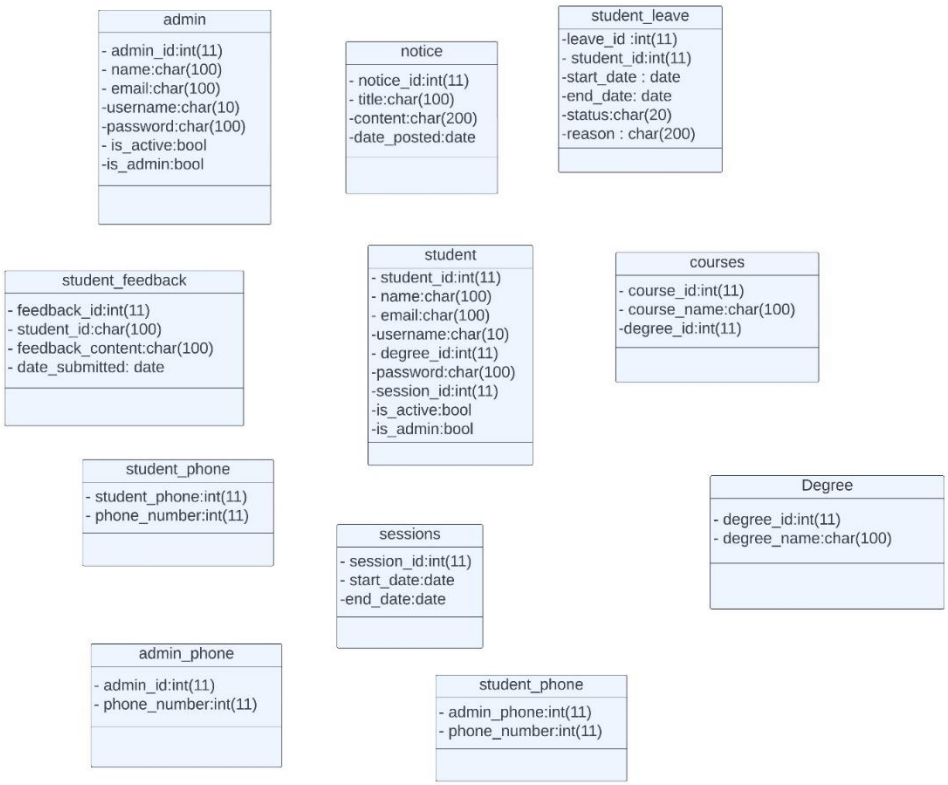
Student and **Degree** : One-to-One.
A student can enroll in one degree, and a degree can have one students enrolled in it.

Student and **Session**: One-to-Many A course can have one session, and a session belongs to many student.

Degree and **Subject**: Many-to-Many. A Degree can cover many subjects, and a subject can be covered by many Degrees.

Admin and **Notice**: One-to-Many. An admin can post many notices, but a notice is posted by one admin.

ER TO TABLE



Student: This table stores information about students, such as their name, email, username, password, and student ID. It also includes fields for whether the student is active and if they are an admin.

Admin: This table stores information about admins, such as their name, email, username, password, and admin ID. It also includes a field for whether the admin is active.

Course: This table stores information about courses, such as the course name and course ID.

Degree: This table stores information about degrees, such as the degree name and degree ID.

Session: This table stores information about sessions, such as the session ID, start date, and end date.

Student feedback: This table stores information about student feedback, such as the feedback ID, student ID, feedback content, and date submitted.

Leave: This table stores information about student leave, such as the leave ID, student ID, start date, end date, status, reason for leave.

Notice: This table stores information about notices, such as the notice ID, title, content, date posted.

Student phone: This table stores the phone number for a student. It includes a foreign key that references the student ID in the student table.

Admin phone: This table stores the phone number for an admin. It includes a foreign key that references the admin ID in the admin table.

Normalization

First Normal Form (1NF):

- No repeating groups exist within the table.
- All attributes contain atomic (indivisible) values.

Second Normal Form (2NF):

- The table adheres to 1NF.
- All non-key attributes are fully dependent on the table's primary key. There are no partial dependencies where a non-key attribute relies only on a portion of the primary key.

Third Normal Form (3NF):

- The table adheres to 1NF and 2NF.

- There are no transitive dependencies between non-key attributes. This means that a non-key attribute is not dependent on another non-key attribute, which in turn depends on the primary key.

- **First Normal Form (1NF):** A table is in 1NF if it contains only atomic values (indivisible data) and no repeating groups. All the tables satisfy this condition except student and admin So they are divided and two new tables are formed **student phone and admin phone.**

- **Second Normal Form (2NF):** A table is in 2NF if it follows 1NF and all non-key attributes are fully dependent on the primary key. This means there are no partial dependencies where a non-key attribute depends on only a part of the primary key.

  - **Admin**: The primary key is `admin_id`, and all attributes (`name`, `email`, `is_active`) are fully dependent on it, so it's in 2NF.

  - **Student**: The primary key is `student_id`, and all attributes (`student_name`, `student_email`, `student_phone`) are fully dependent on it, so it's in 2NF.

  - **Course**: The primary key is `course_id`, and all attributes (`course_name`) are fully dependent on it, so it's in 2NF.

  - **Session**: The primary key is `session_id`, and all attributes (`course_id`, `start_date`, `end_date`) are fully dependent on it, so it's in 2NF.

  - **Subject**: The primary key is `subject_id`, and all attributes (`subject_name`) are fully dependent on it, so it's in 2NF.

- **Notice:** The primary key is `notice_id`, and all attributes (`title`, `content`, `date_posted`) are fully dependent on it, so it's in 2NF.
- **Student Leave:** The primary key is `leave_id`, and all attributes (`student_id`, `start_date`, `end_date`, `reason`) are fully dependent on it, so it's in 2NF.
- **Student Feedback:** The primary key is `feedback_id`, and all attributes (`student_id`, `feedback_content`, `date_submitted`) are fully dependent on it, so it's in 2NF.
- **Course Enrollment (Associative Table):** The primary key is likely a composite key of `course_id` and `student_id`, and both attributes are part of the key so there are no partial dependencies. Thus, it's in 2NF.
- **Subjects (Associative Table):** The primary key is likely a composite key of `course_id` and `subject_id`, and both attributes are part of the key so there are no partial dependencies. Thus, it's in 2NF.

Third Normal Form (3NF): A table is in 3NF if it follows 1NF and 2NF, and there are no transitive dependencies. A transitive dependency exists when a non-key attribute is dependent on another non-key attribute, which is in turn dependent on the primary key.

Additional Notes (for tables dues multivalue attributes):

The tables **Student Phone** and **Admin Phone** are not included in the analysis as they are standalone tables with just a single attribute, and don't directly participate in any relationships with other tables.

SQL & PL/SQL

Stored Procedures and functions

Triggers

Trigger and procedure to add id sequence wise and adding phone to student_phone table inserting the data

PROCEDURE

```
-- Sequence for generating student_id values
CREATE SEQUENCE student_seq
    START WITH 1
    INCREMENT BY 1
    NOCACHE
    NOCYCLE;
-- Procedure to create student without student_id parameter
CREATE OR REPLACE PROCEDURE create_student (
    name IN VARCHAR2,
    email IN VARCHAR2,
    username IN VARCHAR,
    password IN VARCHAR,
    degree_id IN NUMBER,
    phone IN VARCHAR2,
    session_id IN NUMBER
)
IS
BEGIN
    INSERT INTO Student (student_id, name, email, degree_id,session_id)
    VALUES (student_seq.NEXTVAL, name, email, degree_id,session_id);
    INSERT INTO student_phone (student_id, phone_number)
    VALUES (student_seq.CURRVAL, phone);
END;
```

TRIGGER

```
-- Trigger to ensure that the student_id is automatically generated
CREATE OR REPLACE TRIGGER trg_student_id_auto_generate
BEFORE INSERT ON Student
FOR EACH ROW
BEGIN
    IF :NEW.student_id IS NULL THEN
        SELECT student_seq.NEXTVAL INTO :NEW.student_id FROM DUAL;
    END IF;
END;
/
```

EXECUTION

```
EXEC create_student('A','A@gmail.com','Ar','ASD',1,'1234567890',1);
EXEC create_student('B','B@gmail.com','Br','ASD',2,'197131730',1);
EXEC create_student('C','C@gmail.com','Br','ASD',1,'2911287131',1);
EXEC create_student('D','D@gmail.com','Cr','ASD',2,'2911357131',1);
EXEC create_student('E','E@gmail.com','Dr','ASD',2,'2971511131',1);
EXEC create_student('F','F@gmail.com','Er','ASD',1,'2971387131',1);
```

OUTPUT

STUDENT_ID	NAME	USERNAME	PASSWORD	EMAIL	IS_ACTIVE	IS_ADMIN	DEGREE_ID	SESSION_ID
1	A	-	-	A@gmail.com	1	0	1	1
2	B	-	-	B@gmail.com	1	0	2	1
3	C	-	-	C@gmail.com	1	0	1	1
4	D	-	-	D@gmail.com	1	0	2	1
5	E	-	-	E@gmail.com	1	0	2	1
6	F	-	-	F@gmail.com	1	0	1	1

STUDENT_ID	PHONE_NUMBER
1	1234567890
2	197131730
3	2911287131
4	2911357131
5	2971511131
6	2971387131

FUNCTION to get current date

```
-- function for current date

CREATE OR REPLACE FUNCTION get_current_date RETURN DATE IS
    csubmited_date DATE;
BEGIN
    SELECT SYSDATE INTO submited_date FROM DUAL;
    RETURN submited_date;
END;
/
```

Trigger to add feedback_id sequence wise and procedure to add current date to date_submitted

PROCEDURE

```
-- Procedure to create student feedback without feedback_id parameter
CREATE OR REPLACE PROCEDURE create_feedback (
    student_id NUMBER,
    feedback_content VARCHAR2
)
IS
    submited_date DATE;
BEGIN
    submited_date := get_current_date();
    INSERT INTO Student_Feedback (feedback_id, student_id, feedback_content, date_submitted)
    VALUES (student_feedback_seq.NEXTVAL, student_id, feedback_content, submited_date);
END;
/
```

TRIGGER

```
-- Trigger to ensure that the feedback_id is automatically generated
CREATE OR REPLACE TRIGGER trg_feedback_id_auto_generate
BEFORE INSERT ON Student_Feedback
FOR EACH ROW
BEGIN
    IF :NEW.feedback_id IS NULL THEN
        SELECT student_feedback_seq.NEXTVAL INTO :NEW.feedback_id FROM DUAL;
    END IF;
END;
/
```

EXECUTION

```
EXEC create_feedback(1,'good project');
EXEC create_feedback(2,'vvvgood project');
EXEC create_feedback(2,'vvgood project');
EXEC create_feedback(5,'vgood project');
```

OUTPUT

FEEDBACK_ID	STUDENT_ID	FEEDBACK_CONTENT	DATE_SUBMITTED
1	1	good project	24-APR-24
2	2	vvvgood project	24-APR-24
3	2	vvgood project	24-APR-24
4	5	vgood project	24-APR-24

Trigger AND Procedure to add student_id sequence wise to student_leave

PROCEDURE

```
-- Sequence for generating leave_id values
CREATE SEQUENCE student_leave_seq
    START WITH 1
    INCREMENT BY 1
    NOCACHE
    NOCYCLE;

-- Procedure to create student leave without leave_id parameter
CREATE OR REPLACE PROCEDURE create_leave (
    student_id IN NUMBER,
    start_date IN DATE,
    end_date IN DATE,
    reason IN VARCHAR2
)
IS
BEGIN
    INSERT INTO Student_Leave (leave_id, student_id, start_date, end_date, reason,status)
    VALUES (student_leave_seq.NEXTVAL, student_id, start_date, end_date, reason,'waiting'
    );
END;
/
```

TRIGGER

```
-- Trigger to ensure that the leave_id is automatically generated
CREATE OR REPLACE TRIGGER trg_leave_id_auto_generate
BEFORE INSERT ON Student_Leave
FOR EACH ROW
BEGIN
    IF :NEW.leave_id IS NULL THEN
        SELECT student_leave_seq.NEXTVAL INTO :NEW.leave_id FROM DUAL;
    END IF;
END;
/
```

EXECUTION

```
EXEC create_leave(1,TO_DATE('23 APRIL 2024','DD MON YYYY'),TO_DATE('1 MAY 2024' , 'DD MON YYYY'),'SOMETHING IMPORTANT');
EXEC create_leave(2,TO_DATE('20 APRIL 2024','DD MON YYYY'),TO_DATE('1 MAY 2024' , 'DD MON YYYY'),'SOMETHING IMPORTANT');
EXEC create_leave(2,TO_DATE('12 APRIL 2024','DD MON YYYY'),TO_DATE('1 MAY 2024' , 'DD MON YYYY'),'SOMETHING IMPORTANT');
EXEC create_leave(3,TO_DATE('9 APRIL 2024','DD MON YYYY'),TO_DATE('1 MAY 2024' , 'DD MON YYYY'),'SOMETHING IMPORTANT');
EXEC create_leave(5,TO_DATE('13 APRIL 2024','DD MON YYYY'),TO_DATE('1 MAY 2024' , 'DD MON YYYY'),'SOMETHING IMPORTANT');
```

OUPUT

LEAVE_ID	STUDENT_ID	START_DATE	END_DATE	STATUS	REASON
1	1	23-APR-24	01-MAY-24	waiting	SOMETHING IMPORTANT
2	2	20-APR-24	01-MAY-24	waiting	SOMETHING IMPORTANT
3	2	12-APR-24	01-MAY-24	waiting	SOMETHING IMPORTANT
4	3	09-APR-24	01-MAY-24	waiting	SOMETHING IMPORTANT
5	5	13-APR-24	01-MAY-24	waiting	SOMETHING IMPORTANT

PROCEDURE TO ADD THE notice_id sequence wise and add current date taken from function defined to date_posted

PROCEDURE

```
-- Sequence for generating notice_id values
CREATE SEQUENCE notice_seq
    START WITH 1
    INCREMENT BY 1
    NOCACHE
    NOCYCLE;

-- Procedure to create a notice with automatically generated notice_id
CREATE OR REPLACE PROCEDURE create_notice (
    title IN VARCHAR2,
    content IN VARCHAR2
)
IS
    submited_date DATE;
BEGIN
    submited_date := get_current_date();
    INSERT INTO Notice (notice_id, title, content, date_posted)
    VALUES (notice_seq.NEXTVAL, title, content, submited_date);
END;
/
```

EXECUTION

```
EXEC create_notice('NOTICE','CONTENT');
EXEC create_notice('NOTICE','CONTENT');
EXEC create_notice('NOTICE','CONTENT');
EXEC create_notice('NOTICE','CONTENT');
```

OUPUT

NOTICE_ID	TITLE	CONTENT	DATE_POSTED
1	NOTICE	CONTENT	28-APR-24
2	NOTICE	CONTENT	28-APR-24
3	NOTICE	CONTENT	28-APR-24
4	NOTICE	CONTENT	28-APR-24

Trigger and procedure to add id sequence wise and adding phone to admin_phone table inserting the data

PROCEDURE

```
-- Procedure to create admin without admin_id parameter
CREATE OR REPLACE PROCEDURE create_admin (
    name IN VARCHAR2,
    email IN  VARCHAR2,
    username IN VARCHAR,
    password IN VARCHAR,
    phone IN VARCHAR
)
IS
BEGIN
    INSERT INTO Admin(admin_id, name, email,username,password,is_active,is_admin)
    VALUES (admin_seq.NEXTVAL,name,email,username,password,1,1);

    INSERT INTO admin_phone(admin_id, phone_number)
    VALUES (admin_seq.CURRVAL, phone);
END;
/
```

TRIGGER

```
-- Trigger to ensure that the admin_id is automatically generated
CREATE OR REPLACE TRIGGER trg_admin_id_auto_generate
BEFORE INSERT ON Admin
FOR EACH ROW
BEGIN
    IF :NEW.admin_id IS NULL THEN
        SELECT admin_seq.NEXTVAL INTO :NEW.admin_id FROM DUAL;
    END IF;
END;
/
```

EXECUTION

```
EXEC create_admin('Arshdeep','arshdeep@palial','A','bpass','20384758782');
EXEC create_admin('Arsh','palial@arshdeep','B','apss','23498188374');
```

OUTPUT

ADMIN_ID	NAME	EMAIL	USERNAME	PASSWORD	IS_ACTIVE	IS_ADMIN
1	Arshdeep	arshdeep@palial	A	bpass	1	1
2	Arsh	palial@arshdeep	B	apss	1	1

ADMIN_ID	PHONE_NUMBER
1	20384758782
2	23498188374

Trigger and procedure to add id sequence wise to table Degree inserting the data

PROCEDURE

```
-- Sequence for generating course_id values
CREATE SEQUENCE degree_seq
    START WITH 1
    INCREMENT BY 1
    NOCACHE
    NOCYCLE;

-- Procedure to create a degree with automatically generated degree_id
CREATE OR REPLACE PROCEDURE create_degree (
    p_name IN VARCHAR2
)
IS
BEGIN
    INSERT INTO Degree (degree_id, degree_name)
    VALUES (degree_seq.NEXTVAL,p_name);
END;
/
```

EXECUTION

```
EXEC create_degree('computer science');
EXEC create_degree('Robotics');
EXEC create_degree('Mechanical');
```

OUTPUT

DEGREE_ID	DEGREE_NAME
1	computer science
2	Robotics
3	Mechanical

Trigger and procedure to add id sequence wise to courses , inserting the data

PROCEDURE

```
-- Sequence for generating course_id values
CREATE SEQUENCE course_seq
    START WITH 1
    INCREMENT BY 1
    NOCACHE
    NOCYCLE;

-- Procedure to create a subject with automatically generated subject_id
CREATE OR REPLACE PROCEDURE create_course (
    course_name IN VARCHAR2,
    degree_id IN NUMBER
)
IS
BEGIN
    INSERT INTO Courses(course_id, course_name,degree_id)
    VALUES (course_seq.NEXTVAL, course_name, degree_id);
END;
/
```

EXECUTION

```
EXEC create_course('physics',1);
EXEC create_course('maths',1);
EXEC create_course('physics',2);
EXEC create_course('maths',2);
```

OUTPUT

COURSE_ID	COURSE_NAME	DEGREE_ID
1	physics	1
2	maths	1
3	physics	2
4	maths	2

Trigger and procedure to add id sequence wise to session inserting the data

PROCEDURE

```
-- Sequence for generating session_id values
CREATE SEQUENCE sessions_seq
    START WITH 1
    INCREMENT BY 1
    NOCACHE
    NOCYCLE;

-- Procedure to create a session with automatically generated session_id
CREATE OR REPLACE PROCEDURE create_session (
    start_date IN DATE,
    end_date IN DATE,
    degree_id IN NUMBER
)
IS
BEGIN
    INSERT INTO sessions (session_id,  start_date, end_date,degree_id)
    VALUES (sessions_seq.NEXTVAL,  start_date, end_date,degree_id);
END;
/
```

OUTPUT

SESSION_ID	START_DATE	END_DATE	DEGREE_ID
1	02-JUL-22	02-JUN-26	1

Conclusion

In conclusion, our proposed Student Management System represents a holistic approach to addressing the administrative challenges faced by educational institutions. By leveraging technology to streamline processes, enhance communication, and improve data management, the system aims to empower administrators, staff, and students to achieve their academic goals efficiently. Through continuous refinement and adaptation to evolving needs, we envision our Student Management System as a catalyst for positive change within the realm of educational administration.

References

https://djangocentral.com/

https://github.com/