AUTOMATED STUDENT REGISTRATION NUMBER

Content:

Flask Application:

RESTful API endpoints to manage student data.

Routes for adding, viewing, updating, deleting, and filtering student records.

Staff login endpoint with basic authentication.

MySQL Database:

A table (table1) to store student details including unique register number, department, joining year, etc.

Automatic Register Number Generation:

Logic to generate unique student register numbers combining joining year, department code, and a sequential identifier.

Security Setup:

JWT configured for future token-based authentication.

Password hashing capability via bcrypt (currently unused for staff login).

CORS Enabled:

Allows cross-origin HTTP requests from web frontends.

Project Code:

from flask import Flask,request,jsonify

from flask\_mysqldb import MySQL

from flask\_bcrypt import Bcrypt

fromflask\_jwt\_extendedimport JWTManager,create\_access\_token,jwt\_required,get\_jwt,get\_jwt\_identity

from flask\_cors import CORS

from datetime import datetime,timedelta

app = Flask(\_\_name\_\_)

CORS(app)

app.config['JWT\_SECRET\_KEY']="vignan123" #configure

jwt=JWTManager(app)

bcrypt=Bcrypt(app)

app.config['MYSQL\_USER']="root"

app.config['MYSQL\_PASSWORD']="root"

app.config['MYSQL\_DB']="registerstudents"

app.config['MYSQL\_HOST']="localhost"

mysql=MySQL(app)

@app.route("/addstudent",methods=["POST"])

def addstudent():

data = request.json

name = data['name']

department = data['department']

joining\_year = int(data['joining\_year'])

status = "pending"

# Convert joining year to 2-digit format like '23'

year\_suffix = str(joining\_year)[-2:]

prefix = f"2k{year\_suffix}"

# Department short codes (in uppercase)

department\_codes ={

"Computer Science": "CSE",

"Electronics": "ECE",

"Mechanical": "MEC",

"Electrical": "EEE"

}

dept\_code=department\_codes.get(department, "gen") # default to 'gen'

cur=mysql.connection.cursor()

# Count students in the same department and year

cur.execute("SELECT COUNT(\*) FROM table1 WHERE department=%s AND joining\_year=%s",(department, joining\_year))

count=cur.fetchone()[0] + 1 # increment for new student

# Format sequence number to 3 digits

seq=str(count).zfill(3)

# Combine to create register number

register\_number=f"{prefix}{dept\_code}{seq}"

# Insert student record

cur.execute(

"INSERT INTO table1 (name, department, joining\_year, status, register\_number) VALUES (%s, %s, %s, %s, %s)",

(name, department, joining\_year, status, register\_number)

)

mysql.connection.commit()

return jsonify({"Message": "Student added successfully", "Register Number": register\_number})

@app.route("/viewstudents", methods=["GET"])

def viewstudents():

cur = mysql.connection.cursor()

cur.execute("SELECT id, name, department, joining\_year, status, register\_number FROM table1")

rows = cur.fetchall()

col\_names = [desc[0] for desc in cur.description]

results = [dict(zip(col\_names, row)) for row in rows]

return jsonify(results)

@app.route("/particularstudentdetails/<register\_number>", methods=["GET"])

def particularstudentdetails(register\_number):

cur = mysql.connection.cursor()

cur.execute("SELECT name, department, joining\_year, status, register\_number FROM table1 WHERE register\_number=%s", (register\_number,))

row = cur.fetchone()

if row:

col\_names = [desc[0] for desc in cur.description]

result = dict(zip(col\_names, row))

return jsonify(result)

return jsonify({"Error": "NO RECORD FOUND"})

@app.route("/updatestudentdetails/<int:id>", methods=["PUT"])

def updatestudentdetails(id):

cur = mysql.connection.cursor()

data = request.json

name = data['name']

department = data['department']

joining\_year = int(data['joining\_year'])

cur.execute("UPDATE table1 SET name=%s, department=%s, joining\_year=%s WHERE id=%s",

(name, department, joining\_year, id))

mysql.connection.commit()

rowcount = cur.rowcount

if rowcount == 0:

return jsonify({"Error": "Record not found to modify"})

return jsonify({"Message": "Data Updated successfully!", "id": id})

@app.route("/deletestudent/<register\_number>", methods=["DELETE"])

def deletestudent(register\_number):

cur = mysql.connection.cursor()

cur.execute("DELETE FROM table1 WHERE register\_number = %s", (register\_number,))

mysql.connection.commit()

rowcount = cur.rowcount

if rowcount == 0:

return jsonify({"Error": "Record not found to delete"})

return jsonify({"Message": "Record deleted successfully", "register\_number": register\_number})

@app.route("/filter", methods=["GET"])

def filter():

department = request.args.get("department")

cur = mysql.connection.cursor()

cur.execute("SELECT id, name, department, joining\_year, status, register\_number FROM table1 WHERE department = %s", (department,))

rows = cur.fetchall()

col\_names = [desc[0] for desc in cur.description]

results = [dict(zip(col\_names, row)) for row in rows]

return jsonify(results)

@app.route("/stafflogin",methods=["POST"])

def stafflogin():

data=request.json

username=data['username']

password=data['password']

if not username or not password:

return jsonify({"Error":"Missing credentials"})

if username=="vignan" and password=="vignan123":

return jsonify({"message":"Login success"})

else:

return jsonify({"message":"Login failed"})

@app.route("/studentbyregno/<string:regno>", methods=["GET"])

def studentbyregno(regno):

cur = mysql.connection.cursor()

cur.execute("SELECT id, name, department, joining\_year FROM table1 WHERE register\_number=%s", (regno,))

row = cur.fetchone()

if row:

return jsonify({

"id": row[0],

"name": row[1],

"department": row[2],

"joining\_year": row[3]

})

return jsonify({"Error": "Student with this Register Number not found"})

if \_\_name\_\_=="\_\_main\_\_":

app.run(debug=True)

Key Technologies:

Technology Purpose

Python Flask Web framework to create REST API

MySQL Relational database to store student records

flask\_mysqldb MySQL connector for Flask

flask\_bcrypt Password hashing for secure credential storage

flask\_jwt\_extended JWT token management for authentication

flask\_cors Enable Cross-Origin Resource Sharing

JSON Data format for requests and responses

Description:

The Automated Student Registration Number project is a Flask-based RESTful API designed to manage student records and automatically generate unique registration numbers. When a new student is added, the system creates a registration number based on the student’s joining year and department, combining:

-A fixed prefix (2k),

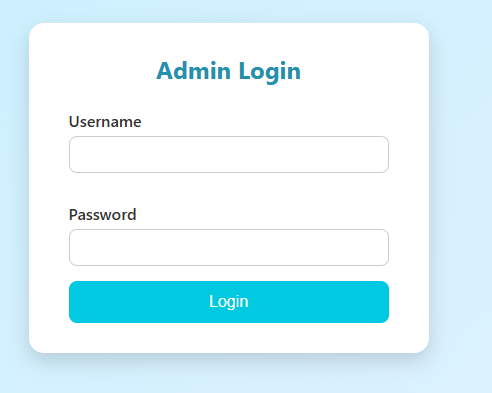
-The last two digits of the joining year,

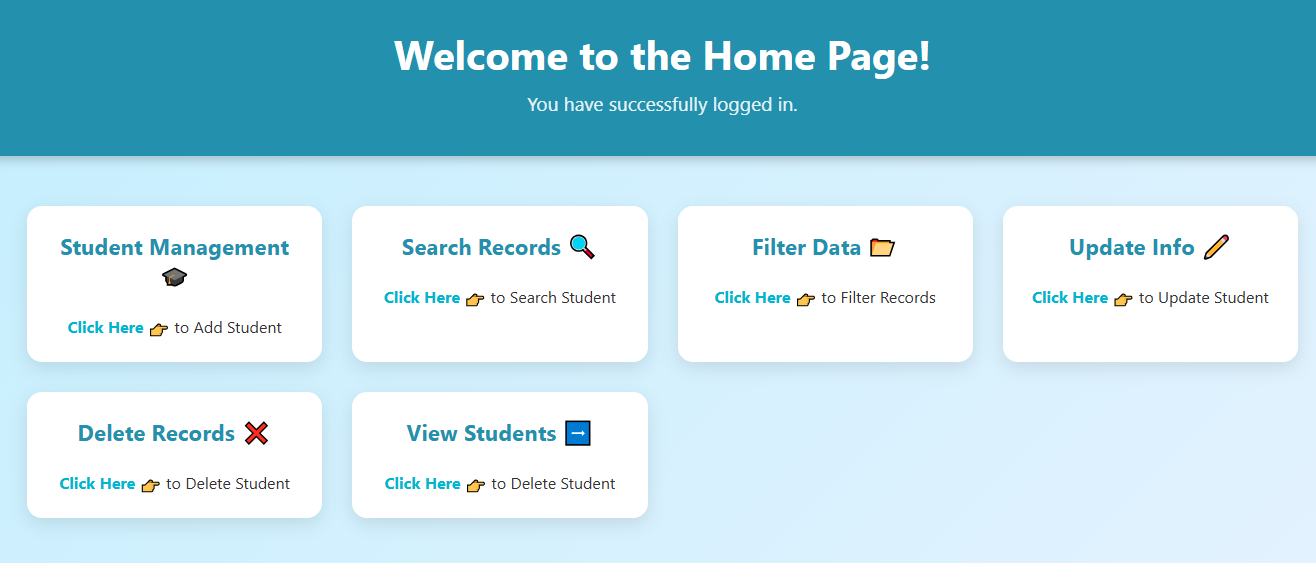
-A department code (like CSE for Computer Science),

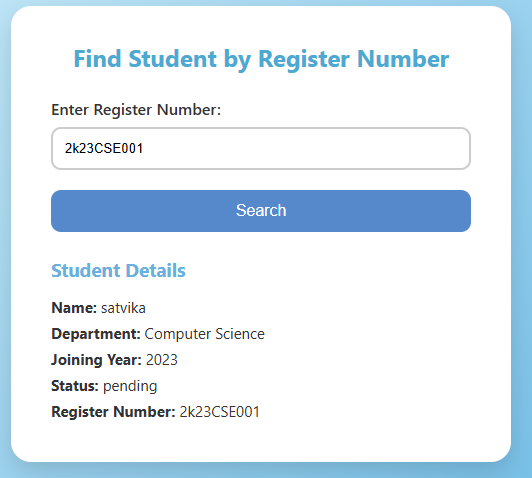
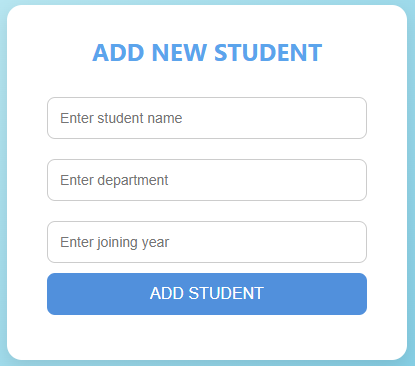
-A sequential 3-digit number representing their position among students from the same department and year.

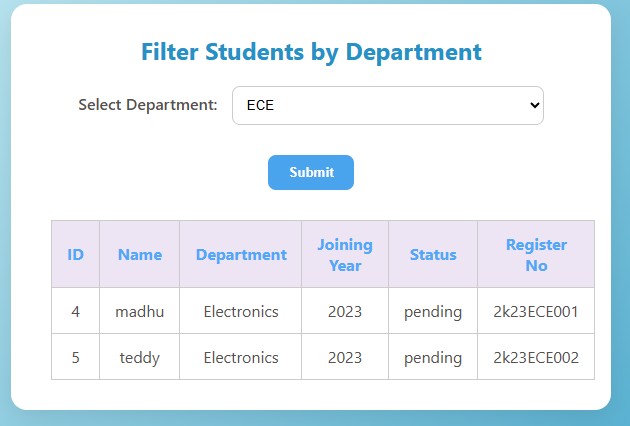
The API supports creating, reading, updating, deleting, and filtering student records. It also includes a basic staff login endpoint for authentication purposes. The backend is powered by a MySQL database, and CORS is enabled for frontend integration.

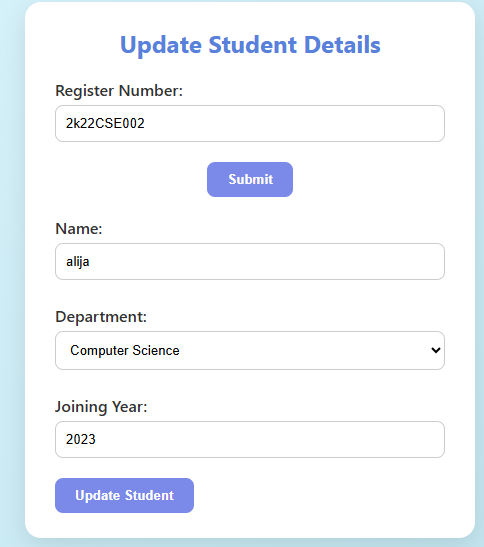
Output:

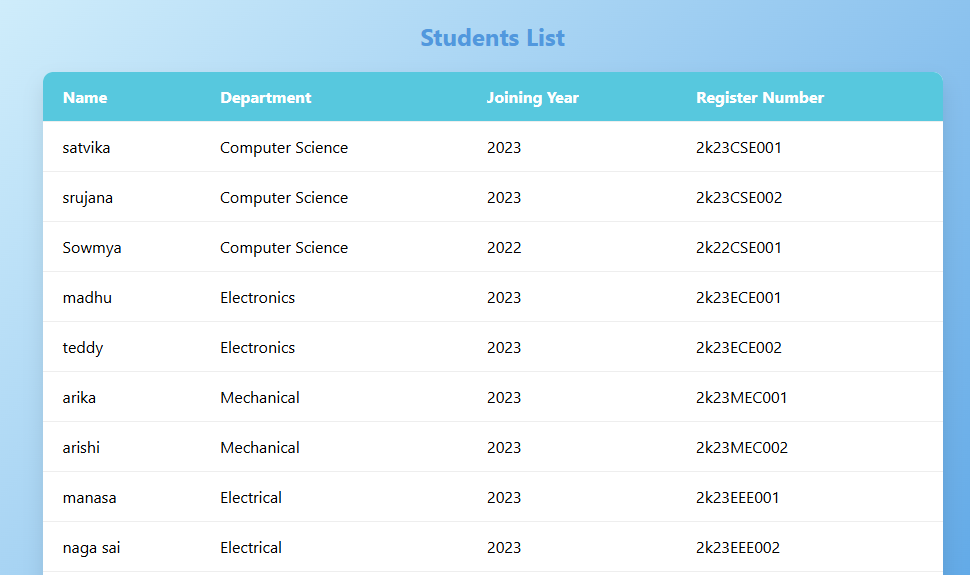


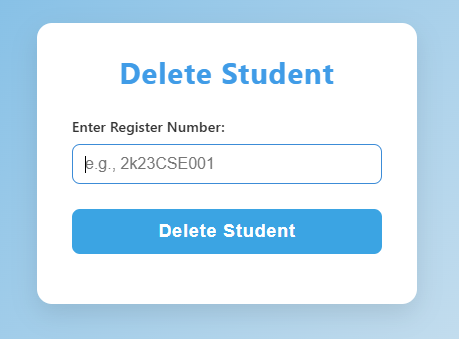












Further Research & Improvement Ideas:

Authentication & Authorization:

Implement full JWT-based login for staff with role-based access control.

Store staff credentials securely in the database with bcrypt hashing.

Protect sensitive API routes behind authentication.

Input Validation & Error Handling:

Add robust validation for input data to prevent SQL injection and bad data.

Use Flask Marshmallow or similar libraries for schema validation.

Database Design:

Normalize the database, possibly separate departments into their own table.

Use migrations (Flask-Migrate) to manage schema changes.

User Interface:

Build a frontend interface using React, Angular, or Vue.js for interacting with the API.

Provide forms for student registration and admin panels for staff.

Scalability & Performance:

Implement pagination on listing routes to handle large datasets.

Optimize queries and indexing on frequently queried fields.

Testing & CI/CD:

Add unit and integration tests for API endpoints.

Setup Continuous Integration pipelines for automatic testing and deployment.

Deployment:

Containerize the application using Docker.

Deploy on cloud platforms such as AWS, Azure, or Heroku.