22AIE305

Cloud Computing

A Project Report
Submitted by

Group -12

Kolla Lokesh-CB.EN.U4AIE22027
Tharun balaji-CB.EN.U4AIE22040
Naga koushik-CB.EN.U4AIE22046
Bhavya sainath- CB.EN.U4AIE22055

in partial fulfilment for the award of the degree of

BACHELOR OF TECHNOLOGYIN

CSE(AI)

Centre for Computational Engineering and Networking

AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE

AMRITA VISHWA VIDYAPEETHAM

COIMBATORE - 641 112 (INDIA)

NOVEMBER – 2024

AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE

AMRITA VISHWA VIDYAPEETHAM

COIMBATORE - 641 112

BONAFIDE CERTIFICATE

This is to certify that the project entitled “CLOUD-ENABLED ATTENDANCE SYSTEM ON MICROSOFT AZURE”submitted by KollaLokesh(CB.EN.U4AIE22027) ,TharunBalaji(CB.EN.U4AIE22040),Nagakoushik(CB.EN.U4AIE22046),Bhavyasainath(CB.EN.U4 AIE22055) to Ms. Prajisha Sheejith, for the award of the Degree of Bachelor of Technology in the “CSE(AI)” is a bonafiderecord of the work carried out by her under our guidance and supervision at Amrita School ofArtificial Intelligence, Coimbatore.

Ms. Prajisha Sheejith
Project Guide

Dr. K.P. Soman
Professor and Head CEN

Submitted for the university examination held on 12/11/24.

AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCEAMRITA

VISHWA VIDYAPEETHAM

COIMBATORE - 641 112

DECLARATION

We, Kolla Lokesh(CB.EN.U4AIE22027) ,Tharun Balaji(CB.EN.U4AIE22040),Naga koushik (CB.EN.U4AIE22046),Bhavya sainath(CB.EN.U4AIE22055) hereby declare that this project entitled "CLOUD-ENABLED ATTENDANCE SYSTEM ON MICROSOFT AZURE", is the record of the original work done by us under the guidance of Ms. Prajisha Sheejith, Professor, Amrita School of Artificial Intelligence, Coimbatore. To the best of our knowledge, this work has notformed the basis for the award of any degree/diploma/ associate ship/fellowship/or a similar award to any candidate in any University.

Place: Coimbatore

Date: 12-11-2024 Signature of the Student

CONTENTS

TABLE OF FIGURES

ACKNOWLEDGMENT

We would like to express our special thanks of gratitude to our teacher Ms. Prajisha Sheejith, who gave us the golden opportunity to do this wonderful project on the topic, “CLOUD-ENABLED ATTENDANCE SYSTEM ON MICROSOFT AZURE”, which also helped us in doing a lot of research and we came to know about so many new things. We are thankful for the opportunity given. We would also like to thank our group members, as without their cooperation, we would not have been able to complete the project within the prescribed time.

ABSTRACT

Automated attendance tracking systems are crucial for increasing the precision, effectiveness, and accessibility of attendance data in business and educational settings in the current digital era. This project describes the creation of an attendance tracking system that integrates cloud services and is deployed using Microsoft Azure. A PostgreSQL database powers the system's sturdy Python backend, while Flask was used to create an intuitive interface that makes data entry and visualization simple.

High stability, security, and worldwide accessibility were ensured by deploying the system using Microsoft Azure services, which included Azure SQL Database for data storage and Azure App Service for scalable hosting. The architecture of the system facilitates real-time data processing and provides effective resource management by utilizing Azure's adaptable infrastructure. The attendance monitoring system offers a cost-effective solution for real-time attendance tracking across several platforms and devices with this deployment method, in addition to achieving high availability and scalability.

INRODUCTION

Efficient attendance management is crucial for academic institutions, where accurate record-keeping and accessibility can streamline administrative processes and support educational goals. This project introduces an attendance monitoring system with a web-based interface that provides role-specific functionalities for administrators and teachers, leveraging cloud deployment on Microsoft Azure for scalability and reliability.

The system begins with a predefined administrator role, allowing login options for either an administrator or a teacher. Upon successful authentication, the administrator gains access to functionalities for managing the institutional database, including registering new students and teachers, updating student details, and generating attendance reports for analysis. Teachers, on the other hand, have tailored capabilities that enable them to create class records and mark attendance for each class, simplifying the attendance process. This role-based access control ensures data security and operational efficiency while meeting the specific needs of each user type.

By integrating these features within an Azure-hosted environment, the system benefits from cloud-based advantages such as high availability, secure data handling, and real-time access, making it a robust solution for modern educational needs.

METHODOLOGY

approach for your attendance monitoring system that takes into account administrators, instructors, and students while emphasizing the important procedures and elements involved in the system's creation, implementation, and upkeep:

1. Designing the Frontend and Backend

- The frontend of the Attendance Monitoring System is developed using Flask with HTML, CSS, and JavaScript to create an interactive, responsive user interface.
- Admins have a dashboard for managing users and generating reports, while teachers have a dashboard to manage classes and mark attendance. Students can view their attendance records.
- The backend is built with Python and Flask to handle logic such as attendance validation, user authentication, and data management. The application uses role-based access control (RBAC) to ensure the right features are available based on user roles.

2. Setting Up Data Storage with Azure SQL Database

- Azure SQL Database is used to store all user and attendance data securely and scalably. The database holds tables for:
- Users: Admins, teachers, and students, with their roles and other relevant details.
- Classes: Information about each class, such as class name, section, teacher, and schedule.
- Attendance Records: Data related to student attendance, including the class, date, and attendance status (present/absent).
- This database setup ensures reliable data management and retrieval for both frontend and backend interactions.

3. User Authentication and Role-Based Access Control

- The application uses session-based or token-based authentication (JWT) to ensure secure login and protect access to user-specific data.
- Role-based access control (RBAC) is implemented:
- Admins can manage users (register new students and teachers), generate attendance reports, and access all data.
- Teachers can create classes, mark attendance, and view attendance records for their classes.

- Students can only view their own attendance records.
- This ensures that users only have access to the functionalities relevant to their roles.

4. Building and Deploying the Application on Azure

- The Flask application is deployed to Azure App Service, which provides a scalable and reliable hosting environment.
- Continuous integration and continuous deployment (CI/CD) pipelines are set up using Azure DevOps or GitHub to automate code deployment, ensuring quick updates and bug fixes.
- The backend and frontend interact seamlessly with Azure SQL Database and Azure Blob Storage, where data such as user profiles, class details, and attendance logs are securely stored.

5. Ensuring Data Security and Access Control

- Azure Active Directory (AD) is configured for user authentication, with multi-factor authentication (MFA) enabled for admins for extra security.
- Sensitive data, including database credentials and API keys, are stored securely in Azure Key Vault, ensuring only authorized services can access them.
- The application is hosted within a Virtual Network (VNet) to isolate and protect sensitive data and restrict unauthorized access.

6. Managing Traffic and Improving Performance with Application Gateway

- To ensure high availability and efficient traffic management, Azure Application Gateway is used to load balance incoming user requests and provide Web Application Firewall (WAF) protection.
- The application automatically scales based on usage, ensuring a seamless experience even during periods of high user traffic.

7. Maintenance and Future Enhancements

- Regular Updates: The system is periodically updated to include new features, security patches, and database optimizations.
- Scalability: The system is designed to scale horizontally on Azure to handle increasing loads as more students, teachers, and admins use the system.
- Future Features: Plans for future enhancements include integrating mobile app support, adding facial recognition for attendance, and providing advanced analytics for tracking attendance trends and performance.

CODE

1. Azure database creation code:

```
import psycopg2
from psycopg2 import sql

# Connect to PostgreSQL database
conn = psycopg2.connect(
    host="localhost",
    database="college_attendance_system",  # Replace with your database name
    user="postgres",  # Replace with your username
    password="123"  # Replace with your password
)

# Create a cursor object to execute SQL queries
cur = conn.cursor()

# Create tables
try:
    # Drop tables if they already exist (optional)
    cur.execute("DROP TABLE IF EXISTS attendance, students, classes, admins,
teachers CASCADE;")

    # Create `teachers` table
    cur.execute("""
        CREATE TABLE teachers (
            username VARCHAR(50) PRIMARY KEY,
            teacher_name VARCHAR(20),
            password VARCHAR(50),
            email VARCHAR(20),
            phone VARCHAR(10)
        );
    """)

    # Insert dummy data into `teachers`
    cur.execute("""
        INSERT INTO teachers (username, teacher_name, password, email, phone)
VALUES
        ('teacher1', 'Jenny', 'password1', 'jenny@gmail.com', '9183885580'),
        ('teacher2', 'Mary', 'password2', 'marys@gmail.com', '9801802223');
    """)

    # Create `admins` table
    cur.execute("""
        CREATE TABLE admins (
            username VARCHAR(50) PRIMARY KEY,
            admin_name VARCHAR(20),
            password VARCHAR(50),
            email VARCHAR(20),
            phone VARCHAR(10)
        );
    """)

    # Insert dummy data into `admins`
    cur.execute("""
        INSERT INTO admins (username, admin_name, password, email, phone) VALUES
        ('admin1', 'Will Smith', 'password1', 'smithw@gmail.com', '9550634682'),
        ('admin2', 'John Wick', 'password2', 'wjohn@gmai.com', '8192083447');
    """)

    # Create `classes` table
    cur.execute("""
        CREATE TABLE classes (
            class_id SERIAL PRIMARY KEY,
            class_sec VARCHAR(50),
            class_name VARCHAR(100),
            class_date DATE,
            teacher_username VARCHAR(50),
            FOREIGN KEY (teacher_username) REFERENCES teachers(username)
        );
    """)

    # Insert dummy data into `classes`
    cur.execute("""
        INSERT INTO classes (class_name, teacher_username, class_date, class_sec)
VALUES
        ('Machine Learning', 'teacher1', '2023-05-31', 'CSE1'),
        ('BCT', 'teacher2', '2023-05-31', 'CSE2'),
        ('CPNM', 'teacher1', '2023-05-30', 'CSE2'),
        ('Statistics', 'teacher1', '2023-06-01', 'CSE2'),
        ('Algorithms', 'teacher2', '2023-05-30', 'CSE1'),
        ('DS', 'teacher2', '2023-06-01', 'CSE1');
    """)

    # Create `students` table
    cur.execute("""
        CREATE  TABLE  students  (
            student_id SERIAL PRIMARY KEY,
            student_name VARCHAR(50),
            email VARCHAR(20),
            class_sec VARCHAR(10),
            phone VARCHAR(10)
        );
    """)

    # Insert dummy data into `students`
    cur.execute("""
        INSERT INTO students (student_name, email, class_sec, phone) VALUES
        ('John Doe', 'john@gmail.com', 'CSE1', '8866443210'),
        ('Jane Smith', 'smithj@gmail.com', 'CSE1', '9313302392'),
        ('Mark Johnson', 'johnm@gmail.com', 'CSE2', '9314123452'),
        ('Emily Davis', 'davis@gmail.com', 'CSE2', '9233351521'),
        ('Jacob White', 'dean58@example.net', 'CSE1', '6059073030');
    """)

    # Create `attendance` table
    cur.execute("""
        CREATE TABLE attendance (
            id SERIAL PRIMARY KEY,
            class_id INT,
            student_id INT,
            status VARCHAR(10) CHECK (status IN ('present', 'absent')),
            FOREIGN KEY (class_id) REFERENCES classes(class_id),
            FOREIGN KEY (student_id) REFERENCES students(student_id)
        );
    """)

    # Insert dummy data into `attendance`
    cur.execute("""
        INSERT INTO attendance (class_id, student_id, status) VALUES
        (1, 1, 'present'),
        (1, 2, 'absent'),
        (2, 3, 'present'),
        (2, 4, 'absent');
    """)

    # Commit the transaction
    conn.commit()
    print("Database schema created and dummy data inserted successfully.")

except Exception as e:
    print("An error occurred:", e)
    conn.rollback()
```

```
finally:
    # Close cursor and connection
    cur.close()
    conn.close()
```

Explanation:

This Python script, using the pyodbc library, is designed to connect to an Azure SQL Server database, create tables, and insert sample data. The function get_conn() sets up a connection to the Azure SQL Server using a connection string, which includes parameters such as the driver, server details, and login credentials. The create_tables() function then defines the database schema by creating five main tables: admins, teachers, students, classes, and attendance.

In the admins table, admin details such as username, name, password, email, and phone are stored, with username as the primary key. Similarly, the teachers table stores details of each teacher, with username as the primary key. The students table includes a unique, auto-incremented student_id, along with the student's name, email, class section (class_sec), and phone number. The classes table stores information about each class, including its name, assigned teacher (linked through teacher_username), date, and section. The attendance table links classes and students by recording attendance status and references both the classes and students tables, ensuring that any deletion of records cascades to maintain data integrity.

The insert_sample_data() function populates the tables with sample records, adding one admin, two teachers, one student, and five classes. The attendance data insertion is commented out, requiring specific class_id and student_id values for further configuration. Finally, the main() function allows for initializing the database setup by calling the table creation and data insertion functions; however, the create_tables() call is commented out to prevent re-creating tables. When executed, this script should print success messages upon creating tables and inserting sample data, making it a foundational setup for further application development.

2. Application backend code:

```
from flask import Flask, render_template, request, redirect, session, Response,
json, make_response
import psycopg2
from psycopg2 import sql

app = Flask(__name__)
```

```python
app.secret_key = 'your_secret_key'  # Add your secret key here

# Function to create a database if it doesn't exist
import psycopg2
from psycopg2 import sql




# Database connection
try:
    db = psycopg2.connect(
        host='localhost',
        user='postgres',  # Your PostgreSQL username
        password='123',  # Your password
        database='attendancemonitoring'  # The database you just created
    )
    print("Database connected successfully")
except Exception as e:
    print(f"Error connecting to database: {e}")

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/index')
def index():
    return render_template('index.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        role = request.form['role']
        cur = db.cursor()
        if role == 'teacher':
            cur.execute("SELECT * FROM teachers WHERE username = %s AND password
= %s", (username, password))
            user = cur.fetchone()
            if user:
                session['loggedin'] = True
                session['username'] = user[0]
                session['role'] = 'teacher'
                return redirect('/teacher/dashboard')
            else:
                error = 'Invalid login credentials.'
                return render_template('login.html', error=error)
        elif role == 'admin':
            cur.execute("SELECT * FROM admins WHERE username = %s AND password =
%s", (username, password))
            user = cur.fetchone()
            if user:
                session['loggedin'] = True
                session['username'] = user[0]
                session['role'] = 'admin'
                return redirect('/admin/admin_dashboard')
            else:
                error='Invalid login credentials'
                return render_template('login.html',error=error)
        cur.close()
    return render_template('login.html')

@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == "POST":
        username = request.form['username']
        name = request.form['name']
        password = request.form['password']
        email = request.form['email']
        phone = request.form['phone']

        # Check length constraints
        if len(username) > 10:
            error_message = "Username must not exceed 10 characters."
            return render_template('teacher_registration.html',
error=error_message)
        if len(password) > 10:
            error_message = "Password must not exceed 10 characters."
            return render_template('teacher_registration.html',
error=error_message)
        if len(phone) > 10:  # Adjust this based on your requirements
            error_message = "Phone number must not exceed 10 characters."
            return render_template('teacher_registration.html',
error=error_message)

        cursor = db.cursor()
        query = "SELECT username FROM teachers WHERE username = %s"
        cursor.execute(query, (username,))
        existing_user = cursor.fetchone()
        if existing_user:
            error_message = "Username already exists. Please choose a different
one."
            return render_template('teacher_registration.html',
error=error_message)

        insert_query = "INSERT INTO teachers (teacher_name, username, password,
email, phone) VALUES (%s, %s, %s, %s, %s)"
        cursor.execute(insert_query, (name, username, password, email, phone))
        db.commit()
        return render_template('teacher_registration.html', message="Teacher
successfully registered.")

    return render_template('teacher_registration.html')


@app.route('/student_register', methods=['GET', 'POST'])
def student_register():
    if request.method == "POST":
        student_id = request.form['student_id']  # Check length
        name = request.form['name']  # Check length
        class_sec = request.form['class_sec']  # Check length
        email = request.form['email']
        phone = request.form['phone']
        cursor = db.cursor()

        # Example length checks
        if len(student_id) > 10 or len(class_sec) > 10:
            error_message = "Student ID or Class Section exceeds allowed length."
            return render_template('student_registration.html',
error=error_message)

        query = "SELECT student_id FROM students WHERE student_id = %s"
        cursor.execute(query, (student_id,))
        existing_user = cursor.fetchone()
        if existing_user:
            error_message = "Invalid student_id"
            return render_template('student_registration.html',
error=error_message)
        insert_query = "INSERT INTO students (student_id, student_name,
class_sec, email, phone) VALUES (%s, %s, %s, %s, %s)"
        cursor.execute(insert_query, (student_id, name, class_sec, email, phone))
        db.commit()
        return render_template('student_registration.html', message="Student
successfully registered.")
```

```
    return render_template('student_registration.html')


@app.route('/get_student',methods=['POST','GET'])
def get_student():
    if request.method=='POST':
        student_id=request.form['student_id']
        cur = db.cursor()
        cur.execute("SELECT * FROM students WHERE student_id = %s",
(student_id,))
        student = cur.fetchone()
        if student:
            message="Fetched student details"
            return render_template('update_student.html', student=student,
msg=message)
        else:
            error = "Invalid Student ID"
            return render_template('update_student.html', err=error)
    return render_template('update_student.html')

@app.route('/update_student', methods=['GET', 'POST'])
def update_student():
    if request.method == 'POST':
        student_id = request.form['student_id']
        new_name = request.form['new_name']
        new_email = request.form['new_email']
        new_phone = request.form['new_phone']
        cur = db.cursor()
        cur.execute("SELECT * FROM students WHERE student_id = %s",
(student_id,))
        student = cur.fetchone()
        if student:
            cur.execute("UPDATE students SET student_name = %s, email = %s, phone
= %s WHERE student_id = %s",(new_name, new_email, new_phone, student_id))
            db.commit()
            message = "Student ID " +str(student_id)+" details have been
successfully updated."
            return render_template('update_student.html', student=student,
message=message)
        else:
            error = "Invalid Student ID"
            return render_template('update_student.html', error=error)
    return render_template('update_student.html')

@app.route('/teacher/dashboard')
```

```
def teacher_dashboard():
    if 'loggedin' in session and session['role'] == 'teacher':
        cur = db.cursor()
        cur.execute("SELECT * FROM classes WHERE teacher_username = %s",
(session['username'],))
        classes = cur.fetchall()
        cur.close()
        return render_template('teacher_dashboard.html', classes=classes)
    else:
        return redirect('/')

@app.route('/teacher/teacher_profile')
def teacher_profile():
    cur = db.cursor()
    cur.execute("SELECT teacher_name, email, phone FROM teachers WHERE username =
%s", (session['username'],))
    profile_data = cur.fetchone()
    cur.close()
    return render_template('teacher_profile.html', profile_data=profile_data)

@app.route('/teacher/update_profile', methods=['POST'])
def update_profile():
    name = request.form.get('name')
    email = request.form.get('email')
    phone = request.form.get('phone')
    cur = db.cursor()
    cur.execute("UPDATE teachers SET teacher_name = %s, email = %s, phone = %s
WHERE username = %s",(name, email, phone, session['username']))
    db.commit()
    cur.close()
    return redirect('/teacher/teacher_profile')

@app.route('/teacher/add_class', methods=['GET', 'POST'])
def add_class():
    if 'loggedin' in session and session['role'] == 'teacher':
        cur = db.cursor()
        if request.method == 'POST':
            class_name = request.form['class_name']
            class_section = request.form['class_section']
            attendance_date = request.form['attendance_date']
            cur.execute("INSERT INTO classes (class_sec, class_name,class_date,
teacher_username) VALUES (%s,%s, %s,
%s)",(class_section,class_name,attendance_date, session['username']))
            db.commit()
            cur.close()
            return redirect('/teacher/dashboard')
        cur.execute('SELECT DISTINCT class_sec from classes')
        class_sections = cur.fetchall()
        cur.close()
        return render_template('add_class.html', class_sections=class_sections)
    else:
        return redirect('/')



@app.route('/teacher/mark_attendance', methods=['GET'])
def mark_attendance():
    if 'loggedin' in session and session['role'] == 'teacher':
        # Retrieve any necessary data for the attendance marking page here if
needed
        return render_template('mark_attendance.html')
    else:
        return redirect('/teacher/login')  # Redirect to login if not logged in




@app.route("/teacher/mark_attendance/validate", methods=['POST'])
def validate_class_details():
    if 'loggedin' in session and session['role'] == 'teacher':
        data = request.get_json()
        print("Received data:", data)  # Debugging line

        # Validate input data
        if not data or 'date' not in data or 'class_name' not in data or
'class_sec' not in data:
            response_data = {"success": False, "message": "Missing input data"}
            return Response(json.dumps(response_data), status=400,
mimetype='application/json')

        class_date = data['date']
        class_name = data['class_name']
        class_sec = data['class_sec']

        cur = db.cursor()
        cur.execute(
            "SELECT class_id FROM classes WHERE class_name = %s AND class_sec =
%s AND class_date = %s",
            (class_name, class_sec, class_date)
        )
        class_info = cur.fetchone()

        if class_info:
            class_id = class_info[0]
            cur.execute("SELECT student_id, student_name FROM students WHERE
class_sec = %s", (class_sec,))
            students = [{"id": student[0], "name": student[1]} for student in
cur.fetchall()]
            cur.close()
            response_data = {"success": True, "students": students}
            return Response(json.dumps(response_data), status=200,
mimetype='application/json')
        else:
            cur.close()
            response_data = {"success": False, "message": "Class not found or
invalid date."}
            return Response(json.dumps(response_data), status=404,
mimetype='application/json')
    else:
        return redirect('/teacher/login')


@app.route("/teacher/mark_attendance/<class_id>", methods=['GET'])
def display_attendance_form(class_id):
    if 'loggedin' in session and session['role'] == 'teacher':
        cur = db.cursor()
        cur.execute("SELECT * FROM classes WHERE class_id = %s", (class_id,))
        classes = cur.fetchone()

        if classes is None:
            # Handle case where the class ID does not exist
            return "Class not found", 404

        cur.execute("SELECT * FROM students WHERE class_sec = %s", (classes[3],))
        students = cur.fetchall()
        cur.close()

        return render_template('mark_attendance.html', students=students,
                               class_date=classes[4],
                               class_name=classes[1],
                               class_sec=classes[3],
                               class_id=classes[0])
    else:
        # Redirect to the login page or appropriate route if not logged in
        return redirect('/teacher/login')  # Update this to your actual login
route




@app.route("/teacher/mark_attendance/update", methods=['POST'])
def update_attendance():
    if 'loggedin' in session and session['role'] == 'teacher':
        # Getting values from the form and stripping whitespace
        class_date = request.form.get('class_date').strip()
        class_sec = request.form.get('class_sec').strip()
        class_name = request.form.get('class_name').strip()

        # Debugging logs
        print("Class Name:", class_name)
        print("Class Section:", class_sec)
        print("Class Date:", class_date)

        try:
            # Fetch class_id based on class_name, class_sec, and class_date
            query = "SELECT class_id FROM classes WHERE class_name = %s AND
class_sec = %s AND class_date = %s"
            print("Executing query:", query)
            print("With parameters:", (class_name, class_sec, class_date))

            cur = db.cursor()
            cur.execute(query, (class_name, class_sec, class_date))
            class_info = cur.fetchone()

            if class_info:
                class_id = class_info[0]
                print("Found class ID:", class_id)

                # Loop through the students to update or insert attendance
                for student in request.form:
                    if student.startswith('attendance_'):
                        student_id = student.split('_')[1]  # Extract student ID
from the input name
                        status = request.form[student]  # Get the status
(present/absent)

                        print(f"Updating attendance: Class ID: {class_id},
Student ID: {student_id}, Status: {status}")
```

```python
                        # Check if an attendance record already exists for this
student and class
                        cur.execute(
                            "SELECT * FROM attendance WHERE class_id = %s AND
student_id = %s",
                            (class_id, student_id)
                        )
                        existing_record = cur.fetchone()

                        if existing_record:
                            # If a record exists, update the status
                            cur.execute(
                                "UPDATE attendance SET status = %s WHERE class_id
= %s AND student_id = %s",
                                (status, class_id, student_id)
                            )
                        else:
                            # If no record exists, insert a new one
                            cur.execute(
                                "INSERT INTO attendance (class_id, student_id,
status) VALUES (%s, %s, %s)",
                                (class_id, student_id, status)
                            )

                db.commit()  # Commit changes to the database
                cur.close()
                return make_response(json.dumps({"success": True, "message":
"Attendance updated successfully!"}), 200)
            else:
                cur.close()
                print("Class not found or invalid date.")
                return make_response(json.dumps({"success": False, "message":
"Class not found or invalid date."}), 400)

        except Exception as e:
            print("Error occurred:", e)  # Log the error
            return make_response(json.dumps({"success": False, "message": "An
error occurred while submitting attendance."}), 500)

    else:
        return redirect('/teacher/login')
```

```
@app.route('/admin/admin_dashboard')
def admin_dashboard():
    if 'loggedin' in session and session['role'] == 'admin':
        return render_template('admin_dashboard.html')
    else:
        return redirect('/')




@app.route("/admin/get_attendance_report", methods=['GET', 'POST'])
def get_attendance_report():
    if 'loggedin' in session and session['role'] == 'admin':
        if request.method == 'POST':
            class_sec = request.form.get('class_sec')
            class_date = request.form.get('class_date')
            class_name = request.form.get('class_name')

            cur = db.cursor()
            cur.execute("SELECT student_id, student_name FROM students WHERE
class_sec = %s", (class_sec,))
            students = cur.fetchall()

            cur.execute("SELECT student_id, status FROM attendance WHERE class_id
IN (SELECT class_id FROM classes WHERE class_sec = %s AND class_date = %s AND
class_name = %s)", (class_sec, class_date, class_name))
            attendance = cur.fetchall()

            cur.close()

            return render_template('attendance_report.html', class_sec=class_sec,
class_date=class_date, class_name=class_name, students=students,
attendance=attendance)
        else:
            cur = db.cursor()
            cur.execute("SELECT DISTINCT class_sec FROM classes")
            secs = cur.fetchall()
            cur.execute("SELECT DISTINCT class_name FROM classes")
            class_names = cur.fetchall()
            cur.close()

            return render_template('attendance_report.html', class_secs=secs,
class_names=class_names)
    else:
        return redirect('/admin/login')
```

```python
@app.route('/admin/admin_profile')
def admin_profile():
    cur = db.cursor()
    cur.execute("SELECT admin_name, email, phone FROM admins WHERE username = 
%s", (session['username'],))
    profile_data = cur.fetchone()
    cur.close()
    return render_template('admin_profile.html', profile_data=profile_data)

@app.route('/admin/update_admin_profile', methods=['POST'])
def update_admin_profile():
    name = request.form.get('name')
    email = request.form.get('email')
    phone = request.form.get('phone')
    cur = db.cursor()
    cur.execute("UPDATE admins SET admin_name = %s, email = %s, phone = %s WHERE 
username = %s",(name, email, phone, session['username']))
    db.commit()
    cur.close()
    return redirect('/admin/admin_profile')

@app.route('/logout')
def logout():
    session.pop('loggedin', None)
    session.pop('username', None)
    session.pop('role', None)
    return redirect('/')


@app.route('/admin/student_report', methods=['GET', 'POST'])
def student_report():
    if 'loggedin' in session and session['role'] == 'admin':
        if request.method == 'POST':
            student_id = request.form['student_id']
            cur = db.cursor()

            # Retrieve student details
            cur.execute("SELECT student_id, student_name FROM students WHERE 
student_id = %s", (student_id,))
            stud = cur.fetchone()
            # Retrieve attendance records for the student, joining with the
classes table
            cur.execute("""
                    SELECT classes.class_date, classes.class_id,
classes.class_name, attendance.status
                    FROM attendance
                    JOIN classes ON attendance.class_id = classes.class_id
                    WHERE attendance.student_id = %s
                """, (student_id,))
            attn = cur.fetchall()

            if stud and attn:
                total_classes = len(attn)
                attended_classes = sum(1 for entry in attn if entry[3] ==
'present')
                p = (attended_classes / total_classes) * 100 if total_classes > 0
else 0

                return render_template('student_attendance.html', stud=stud,
attn=attn, p=p)
            else:
                error = "No records found for the given student ID."
                return render_template('student_attendance.html', error=error)
        return render_template('student_attendance.html')
    return redirect('/')




if __name__== '_main_':
    app.run(debug=True)
```

Explanation:

The main components include a Flask application script app.py that handles all routing, database interactions, and session management. Data is stored in an Azure SQL Database, which manages information about users, classes, students, and attendance records. HTML templates are designed to provide a cohesive user experience with pages like login, teacher dashboard, and admin dashboard. Additionally, environment variables, such as FLASK_SECRET_KEY, ensure secure session management, reinforcing the application's security practices.

The login functionality is role-based, validating teachers and admins separately. When users enter credentials, the application authenticates them against either the teachers or admins table based on their role. If the login is successful, session variables are set to store the user's status, username, and role. This structure allows restricted access, where teachers and admins only see the data and options pertinent to their role. Invalid login attempts are handled gracefully, displaying error messages on the login page.

The db() function establishes a secure connection to the Azure SQL Database using pyodbc. The connection string includes necessary details like driver, server, and credentials. Parameterized SQL queries with placeholders are used to avoid SQL injection risks. Each query runs inside a try-except-finally block, ensuring proper error handling and database connection cleanup. This method prevents resource leaks and allows smooth operation, even in the case of errors.

The application supports separate registration routes for teachers and students. Duplicate checks prevent registration of already existing usernames or student IDs. Once registered, users can also update their profile details. Both the registration and profile update functionalities are implemented with secure SQL queries and input validation, preserving data integrity.

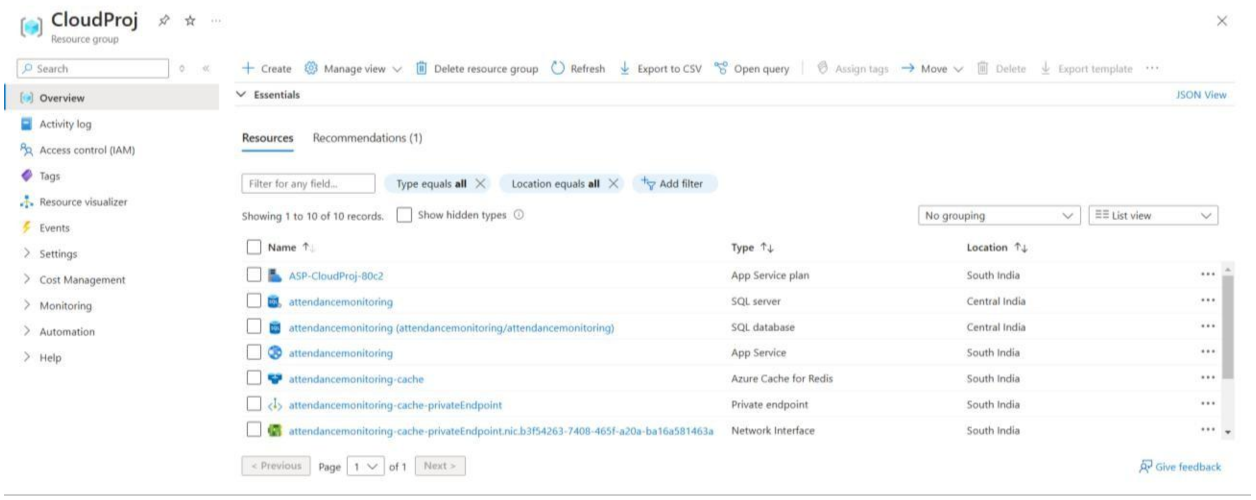
Teachers can add classes and manage attendance records. The application fetches teacher-specific classes from the database, displaying them on the teacher's dashboard. Marking attendance involves a multi-step process where the application first verifies class details and retrieves a list of students before updating their attendance status. The attendance table stores records with fields such as class_id, student_id, and status, enabling detailed tracking and easy querying for reporting purposes.

The admin interface offers features like a dashboard to view registered users, attendance reports by class, and individual student attendance history. These tools provide admins with insights into class participation and attendance trends. The queries used in these reporting functions pull relevant data from multiple tables, like classes and attendance, to generate detailed records and attendance percentages, supporting both administrative oversight and decision-making.

IMPLEMENTATION

1. Resource Setup

- I began by setting up Azure resources, including an Azure SQL Database and an App Service for deployment. These resources were created in the Azure Portal under a specific resource group for easier management.



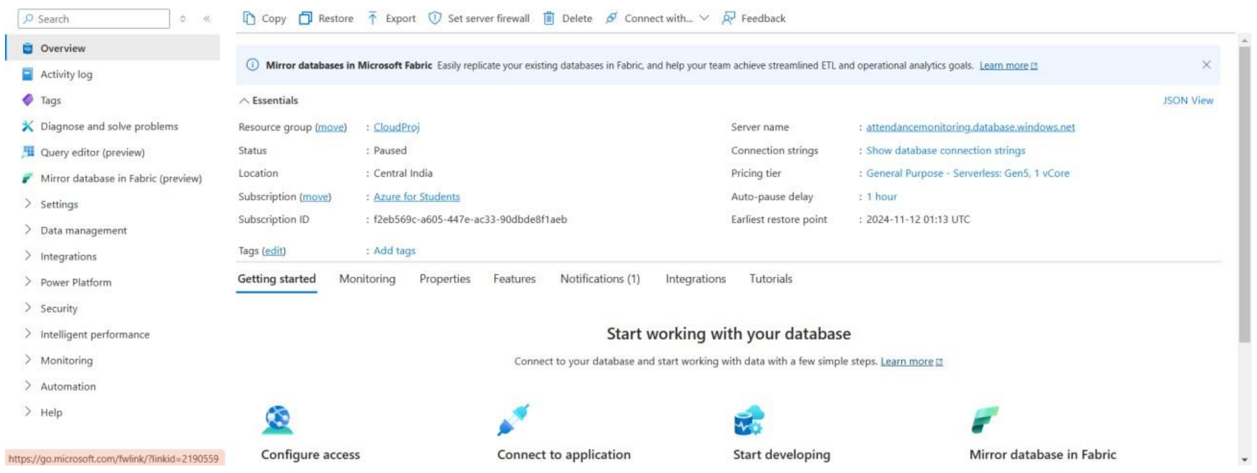
Showing 1 to 10 of 10 records. Show hidden types

Name	Type	Location
ASP-CloudProj-80c2	App Service plan	South India
attendancemonitoring	SQL server	Central India
attendancemonitoring (attendancemonitoring/attendancemonitoring)	SQL database	Central India
attendancemonitoring	App Service	South India
attendancemonitoring-cache	Azure Cache for Redis	South India
attendancemonitoring-cache-privateEndpoint	Private endpoint	South India
attendancemonitoring-cache-privateEndpoint.nic.b3f54263-7408-465f-a20a-ba16a581463a	Network Interface	South India

Figure 1:Azure resource

2. Azure SQL Database Creation

- In the Azure Portal, I created a SQL Database by selecting Create a Resource > SQL Database. I chose an existing or new resource group, specified a database name, and configured a new SQL server with admin credentials.
- I then set up firewall rules within the SQL Server settings to allow my IP address access, ensuring remote connectivity.
- Finally, I accessed the connection string from the database's "Connection strings" section, which I used to connect securely from my Python code.



Mirror databases in Microsoft Fabric Easily replicate your existing databases in Fabric, and help your team achieve streamlined ETL and operational analytics goals. Learn more

Essentials

JSON View

Resource group (move)	: CloudProj	Server name	: attendancemonitoring.database.windows.net
Status	: Paused	Connection strings	: Show database connection strings
Location	: Central India	Pricing tier	: General Purpose - Serverless: Gen5, 1 vCore
Subscription (move)	: Azure for Students	Auto-pause delay	: 1 hour
Subscription ID	: f2eb569c-a605-447e-ac33-90dbde8f1aeb	Earliest restore point	: 2024-11-12 01:13 UTC
Tags (edit)	: Add tags		

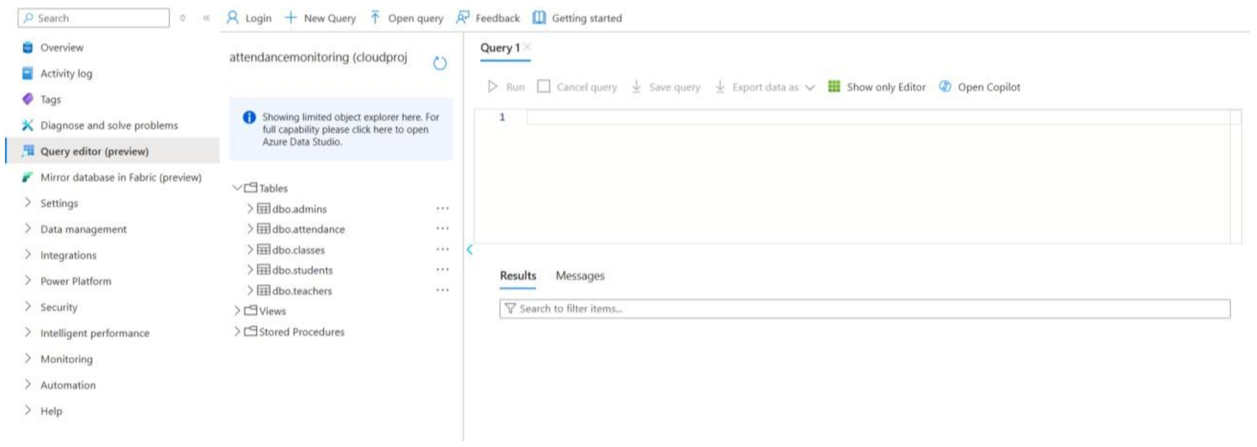
Start working with your database

Connect to your database and start working with data with a few simple steps. Learn more

Figure 2:Azure database

3. Database Setup and Data Insertion with Python

- Using Python, I wrote code to connect to the Azure SQL Database with pyodbc. This script created tables (like students, teachers, classes, etc.) and inserted sample data to verify everything was working as expected. This provided the backend data structure for my app.

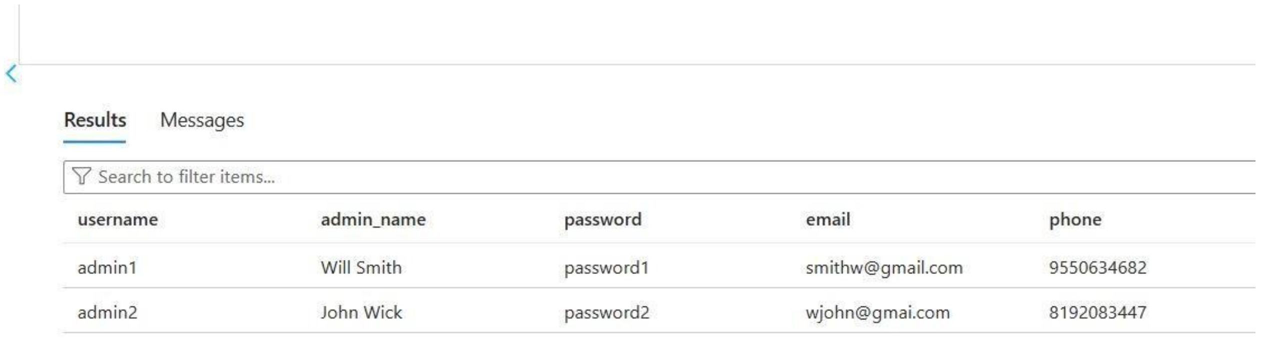


attendancemonitoring (cloudproj

Showing limited object explorer here. For full capability please click here to open Azure Data Studio.

```
Tables
  dbo.admins
  dbo.attendance
  dbo.classes
  dbo.students
  dbo.teachers
Views
Stored Procedures
```

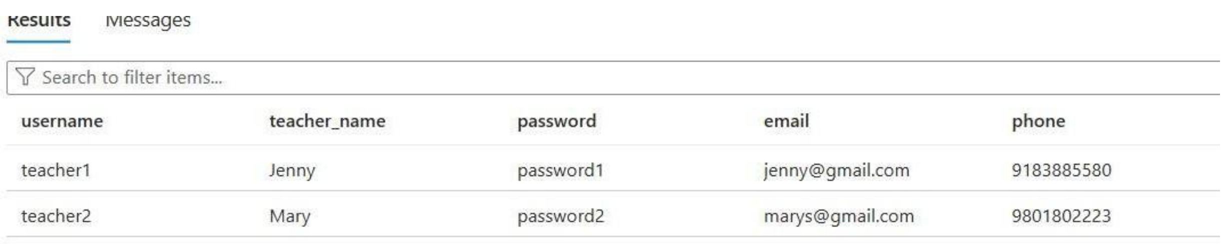
Figure 3:Tables in database



Results Messages

Search to filter items...

username	admin_name	password	email	phone
admin1	Will Smith	password1	smithw@gmail.com	9550634682
admin2	John Wick	password2	wjohn@gmai.com	8192083447



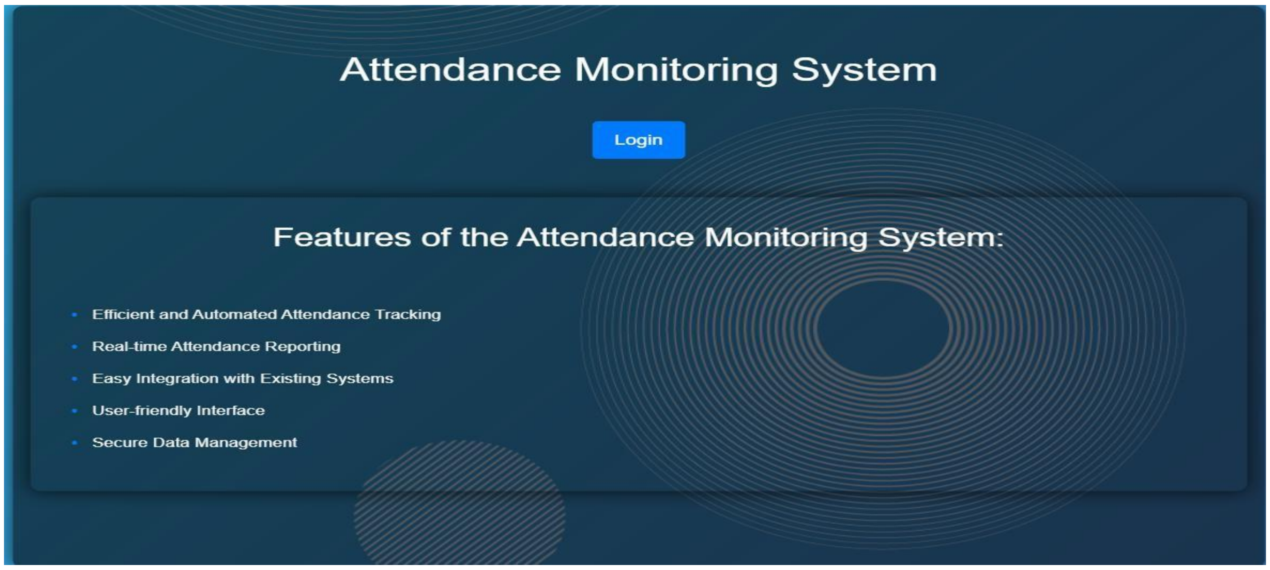
Results Messages

Search to filter items...

username	teacher_name	password	email	phone
teacher1	Jenny	password1	jenny@gmail.com	9183885580
teacher2	Mary	password2	marys@gmail.com	9801802223

4. App Development with Flask

- I built a basic Flask application with routes to handle requests and display content, connecting it to the Azure SQL Database to manage data through the app. I tested it with a basic route to ensure the Flask app was functional.



Attendance Monitoring System

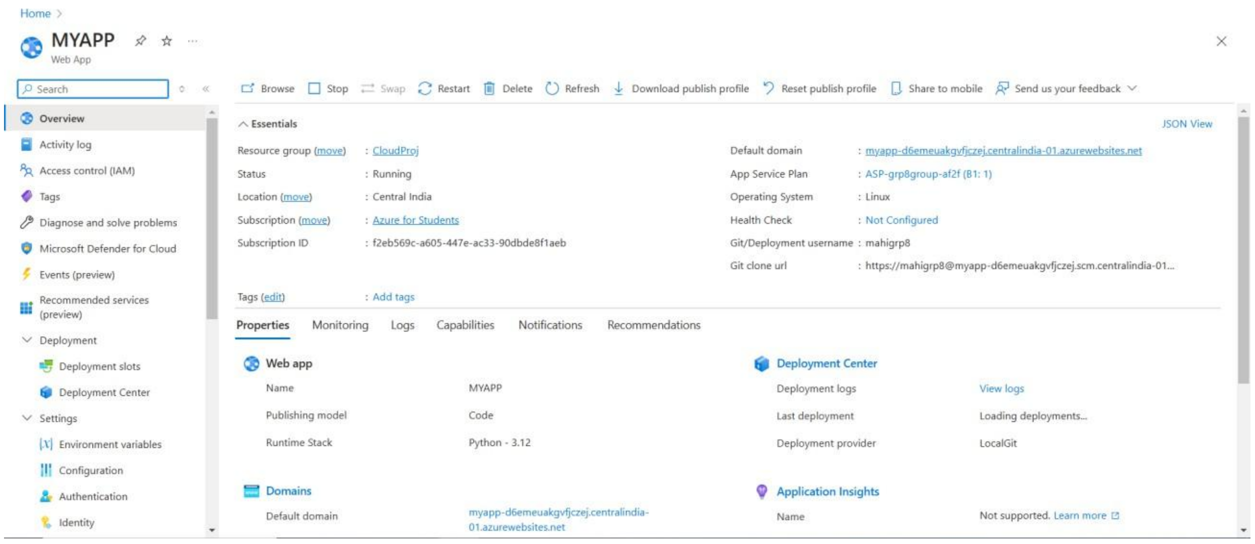
Features of the Attendance Monitoring System:

- Efficient and Automated Attendance Tracking
- Real-time Attendance Reporting
- Easy Integration with Existing Systems
- User-friendly Interface
- Secure Data Management

Figure 4:App deployment with flask

5. Deployment to Azure App Service

- In the Azure Portal, I created an App Service by selecting Create a Resource > App Service. I configured it with my preferred runtime (Python) and linked it to the same resource group as the database.
- Using the Azure CLI, I deployed my Flask app to App Service. I also set environment variables, including the database connection string, in the App Service configuration to securely connect to the database.



Resource group (move)	: CloudProj
Status	: Running
Location (move)	: Central India
Subscription (move)	: Azure for Students
Subscription ID	: f2eb569c-a605-447e-ac33-90dbde8f1aeb
Tags (edit)	: Add tags
Default domain	: myapp-d6emeuakgvfjczej.centralindia-01.azurewebsites.net
App Service Plan	: ASP-grp8group-af2f (B1: 1)
Operating System	: Linux
Health Check	: Not Configured
Git/Deployment username	: mahigrp8
Git clone url	: https://mahigrp8@myapp-d6emeuakgvfjczej.scm.centralindia-01...

Web app

Name	MYAPP
Publishing model	Code
Runtime Stack	Python - 3.12

Domains

Default domain	myapp-d6emeuakgvfjczej.centralindia-01.azurewebsites.net

Deployment Center

Deployment logs	View logs
Last deployment	Loading deployments...
Deployment provider	LocalGit

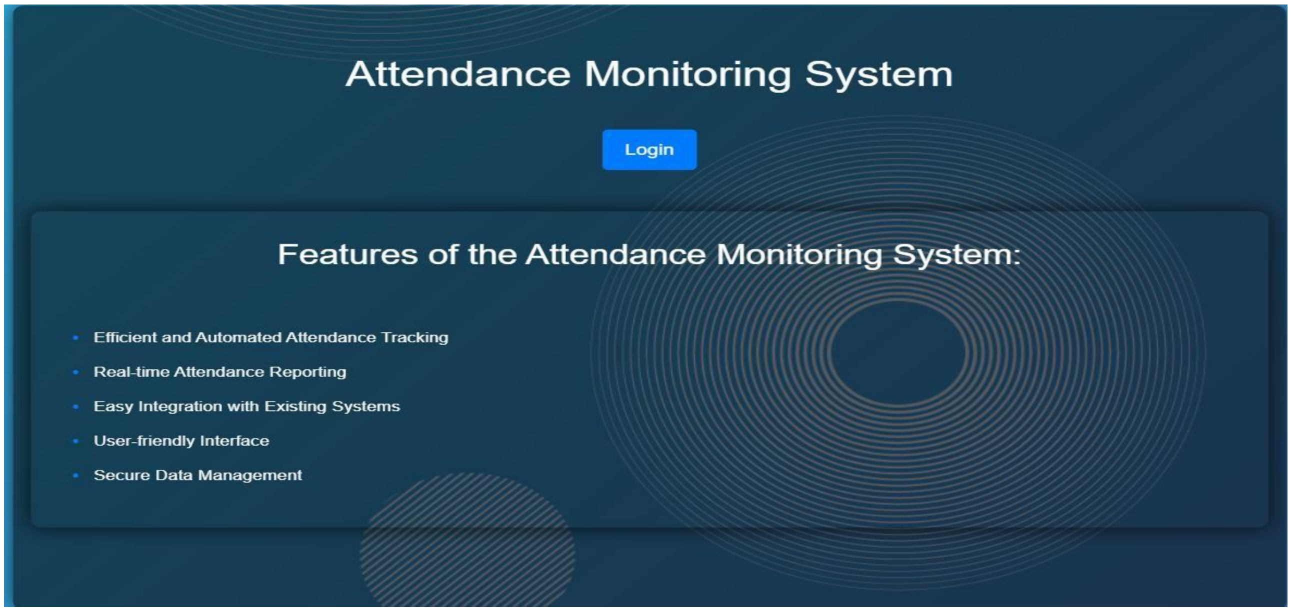
Application Insights

Name	Not supported. Learn more

Figure 5:Azure web app

RESULTS

- After successful azure deployment



Attendance Monitoring System

Features of the Attendance Monitoring System:

- Efficient and Automated Attendance Tracking
- Real-time Attendance Reporting
- Easy Integration with Existing Systems
- User-friendly Interface
- Secure Data Management

Figure 6:App home page

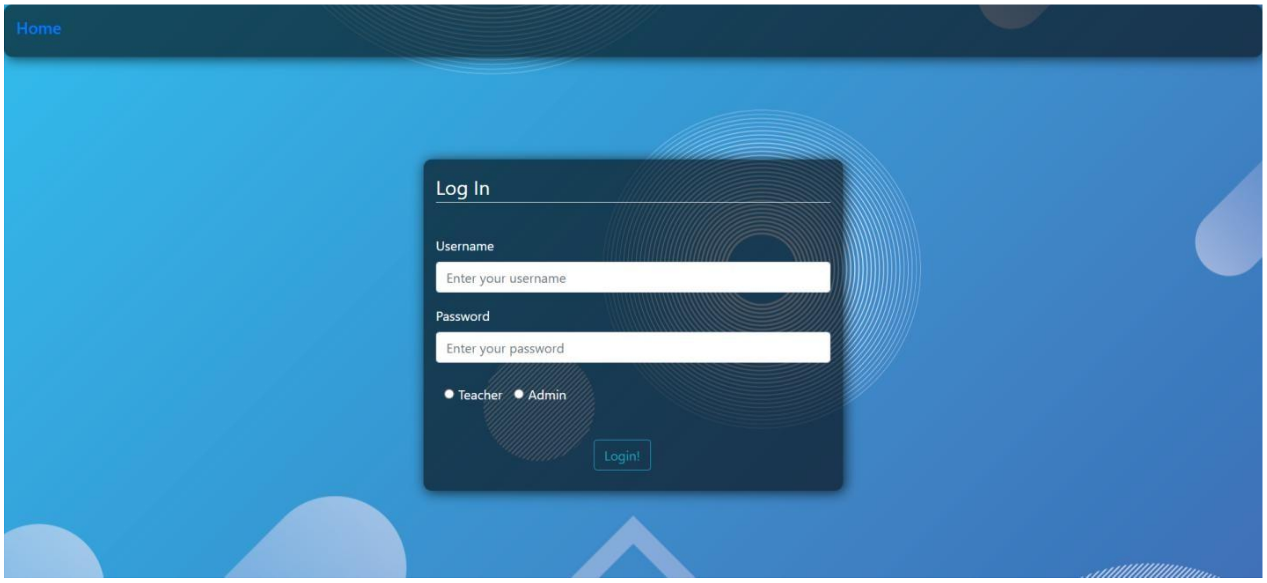


Figure 7:Tacher,admin login page

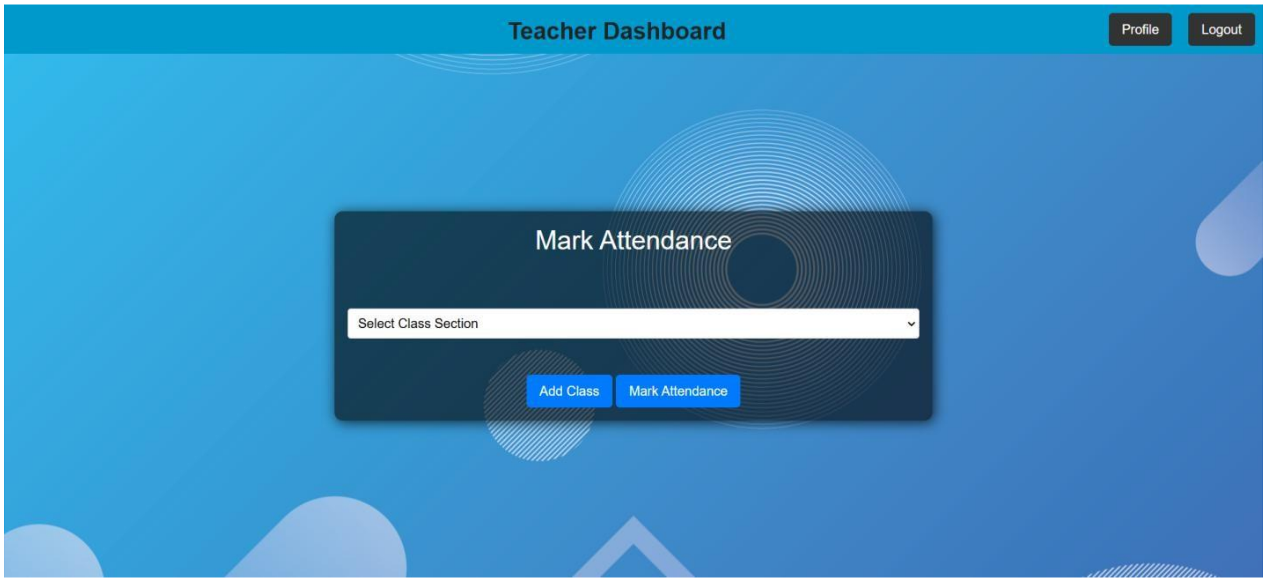


Figure 8:Teacher dashboard

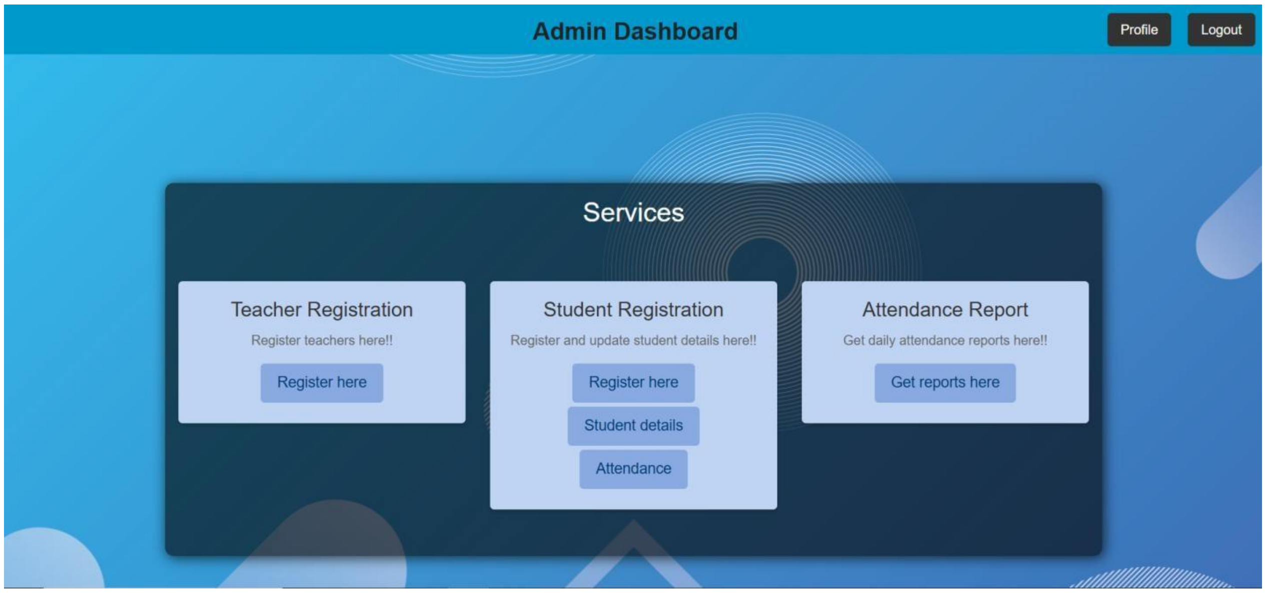


Figure 9:Admin dashboard

CONCLUSION

In conclusion, this Attendance Monitoring System successfully fulfills its purpose as a comprehensive, role-based application for managing attendance in an educational setting. Through its structured implementation, the system offers a secure login and registration process for teachers and admins, ensuring user authentication and role-specific access. The integration with Azure SQL Database enables reliable data management and scalability, making the application suitable for a cloud environment. Additionally, the system provides robust features for class and attendance management, detailed reporting, and user profile updates, which support efficient oversight by both teachers and administrators.

The project's security measures, such as session management, parameterized queries, and error handling, ensure data protection and secure operations. With further enhancements, such as password hashing, mobile compatibility, and advanced analytics, the system can evolve to offer even more functionality. Overall, this project demonstrates a well-rounded approach to developing a scalable, cloud-based application that meets the operational needs of educational institutions and sets a foundation for future improvements.

REFERENCES

https://www.knowledgehut.com/blog/cloud-computing/cloud-computing-project-ideas

https://github.com/Sgvkamalakar/Attendance_monitoring_system