INDEX

1. Introduction

1.1 Project Description

SnapAttendance is an innovative solution designed to streamline the process of taking attendance in educational institutions through facial recognition technology. Developed as part of a final-year project for TYBCA, SnapAttendance aims to revolutionize traditional attendance-taking methods. By harnessing the power of facial recognition, this system eliminates the need for manual entry or paper-based attendance sheets, thereby saving time and reducing the likelihood of errors.

The frontend of SnapAttendance is crafted using HTML, CSS, and JavaScript, ensuring a user-friendly interface that is accessible across various devices. Students and faculty can easily navigate the system to mark attendance with just a simple facial scan. Behind the scenes, the backend utilizes Python, Flask, and SQLite3 to manage the database and handle the intricate processes of face detection and recognition. Through seamless integration of these technologies, SnapAttendance delivers a reliable and efficient attendance tracking solution.

In addition to its technical components, SnapAttendance prioritizes user privacy and security. Facial recognition data is securely stored and encrypted to safeguard sensitive information. Moreover, the system adheres to strict data protection protocols to ensure compliance with privacy regulations. SnapAttendance represents a significant advancement in attendance management, offering educational institutions a modern and reliable solution to track attendance effortlessly.

1.2 Project Profile

Project Title	Automated Attendance System By Face Recognition
Project Description	The System is Provided to have Following Module : ● Student Module ● Teacher Module

	Function of Student Module : 1. Login 2. Dashboard 3. Attendance Function of Teacher Module : 1. Login 2. Dashboard 3. Update Records 4. Insert Records 5. Take Attendance 6. Export Records 7. Capture Students Images
Duration	3 Months
Project Guide	Prof. Nisha G. Medhat
Front End Tool	HTML, CSS, JAVASCRIPT
Back End Tool	PYTHON, FLASK, SQLITE DATABASE
Operating System	Windows 11 (64-Bit)
Submitted By	Prashant Chaute, Bheda Krushita, Kishor Nikhare

2. Environment Description

2.1 Hardware and Software Requirements

Hardware Requirements

Processor	2.60 GHz
RAM	8 GB
Disk Space	256 GB of Available SSD
Graphic	AMD Ryzen 3 3250U with Radeon Graphics
Display	1920 X 1080
Camera	Webcam

Software Requirements

Operating System	Windows 11
Front End	HTML, CSS, JAVASCRIPT
Back End	PYTHON
Library / Framework	Flask, Face Recognition Model

Code Editor	Visual Studio
Database	Sqlite
Web Browser	Google Chrome
Drawing Tools	EdrawMax, Drawio

2.2 Technology Used

Frontend :

1. HTML (Hypertext Markup Language)

HTML is the backbone of web pages, providing the structure and layout for displaying content on the internet.

2. CSS (Cascading Style Sheets)

CSS is used to enhance the presentation of HTML elements by defining styles such as colors, fonts, and layouts.

3. JavaScript

JavaScript adds interactivity to web pages, allowing for dynamic content updates, form validation, and other client-side functionalities.

Backend :

1. Python

Python is a versatile and easy-to-learn programming language used for developing server-side applications, handling data processing tasks, and building web frameworks.

2. Flask

Flask is a lightweight web framework for Python, providing tools and libraries to develop web applications quickly and efficiently.

3. SQLite

SQLite is a self-contained, serverless, SQL database engine used for storing and managing data in relational database management systems.

4. Face Recognition Model

Face recognition technology enables the identification and verification of individuals based on their facial features, facilitating secure authentication and access control in applications like SnapAttendance.

3. System Analysis and Planning

3.1 Existing System and its Drawbacks

Existing System Overview

1. Attendance Method:

- The current system relies on manual methods or basic digital tools for attendance tracking.
- It may involve instructors manually marking attendance or using simple software solutions.

2. Features:

- Limited features such as face recognition, CRUD operations (Create, Read, Update, Delete), and record export functionality are absent.
- The system lacks advanced capabilities for efficient attendance management.

Its Drawbacks

1. Webcam Loading Time:

- Loading the webcam to capture attendance takes considerable time.
- This delay can disrupt the teaching process and lead to inefficiencies in classroom management.

2. Accuracy:

- The system exhibits a moderate accuracy rate, typically ranging from 70% to 80% in recognizing student faces.

- This margin of error can result in misidentifications and inaccurate attendance records.

3. Compatibility:

- The existing system may not accurately function on mobile devices such as smartphones or tablets.
- Limited compatibility hampers the flexibility and accessibility of the attendance tracking process.

3.2 Feasibility Study

1. Research Conducted:

- Extensive research was undertaken to develop SnapAttendance, exploring various technologies and methodologies for automating the attendance process.
- This research aimed to identify the most effective solutions to streamline attendance tracking in educational institutions.

2. Technical Challenges:

- One of the primary technical challenges faced during the development of SnapAttendance was the prolonged time required to capture attendance.
- Addressing this issue necessitated the exploration of alternative solutions, including the consideration of paid software options to enhance efficiency.

3. Economic Considerations:

- The feasibility study highlighted the potential economic benefits of investing in paid software solutions to improve the performance of SnapAttendance.
- While initial costs may be incurred, the long-term savings associated with increased efficiency and reduced manual labor justify the investment.

4. Operational Efficiency:

- The key conclusion of the feasibility study is that SnapAttendance offers significant operational benefits through its automated attendance process.

- By automatically capturing attendance as students enter the frame of the webcam, SnapAttendance eliminates the need for manual intervention, thereby supporting paperless processes and advancing the transition to a digital environment.

3.3 Requirement Gathering and Analysis

1. Identified Requirements:

- The requirement gathering phase for SnapAttendance focused on several key objectives. Firstly, to reduce paper-based processes prevalent in traditional attendance systems, the solution aimed to store attendance data digitally. This not only enhances efficiency but also contributes to environmental sustainability.

- Additionally, a crucial requirement was to ensure the security and privacy of stored data. SnapAttendance prioritizes data security through robust encryption methods and implements secure authentication mechanisms to prevent unauthorized access.

- Furthermore, the solution aimed to streamline the authentication process, ensuring that only authorized users, such as faculty members, have access to attendance records.

2. Stakeholder Engagement:

- To ensure the comprehensiveness of the requirements, extensive stakeholder engagement was conducted. Faculty members from various departments within the college were consulted to gather insights into their specific needs and preferences regarding attendance tracking.

- Feedback obtained from faculty members played a pivotal role in shaping the features and functionalities of SnapAttendance, ensuring that the solution aligns closely with the requirements of end-users.

3. Challenges and Considerations:

- One of the challenges encountered during the requirement gathering phase was the need to integrate multiple script files seamlessly within the project. Ensuring compatibility and smooth operation of all components required meticulous analysis and testing.

- However, through diligent analysis and collaboration with stakeholders, strategies were developed to address these challenges effectively. This involved prioritizing compatibility and ensuring that all script files function harmoniously to deliver a cohesive solution.

4.0 Purpose Of The System

4.1 Scope

The scope of the SnapAttendance system encompasses the development of an automated attendance system by face recognition platform. It includes the following key features :

1. Student Registration and Login
2. Student Dashboard
3. Teacher Dashboard
4. Reduce Paper Process
5. Easily Perform Operations on Records
6. Automatically Take Attendance By Face Recognition
7. Easily Captured Student Images

4.2 Project Modules

SnapAttendance comprises several essential modules, each serving a distinct purpose in the system's functionality :

1. Face Recognition Model

- The Face Recognition module is at the core of SnapAttendance, responsible for accurately identifying and recognizing faces. Leveraging machine learning algorithms, this module analyzes facial features captured by the webcam to match them against existing student profiles stored in the database.

- By integrating advanced facial recognition technology, SnapAttendance enables seamless and efficient attendance capture without the need for manual input.

2. Flask Framework

- Flask serves as the primary framework for developing and deploying web applications in SnapAttendance. With Flask, multiple webpages can be rendered and managed simultaneously, facilitating a cohesive user experience.

- This module handles the backend logic of the system, orchestrating interactions between the frontend interface, the database, and the face recognition model.

3. SQLite Database

- The SQLite Database module is responsible for storing and managing student data, attendance records, and other relevant information. Through the integration of SQLite, SnapAttendance ensures efficient data storage and retrieval, supporting the system's functionality.

- This module interacts seamlessly with the Flask framework, enabling dynamic data handling and manipulation within the application.

4. Additional Libraries

- In addition to the core modules, SnapAttendance incorporates various other libraries and dependencies to enhance its functionality and performance.

- These libraries may include but are not limited to libraries for image processing, data manipulation, and web development, all of which contribute to the overall robustness of the system.

By integrating these modules, SnapAttendance delivers a comprehensive solution for attendance management, leveraging advanced technologies and frameworks to automate processes and streamline operations.

4.3 Module-wise Objectives/Functionalities

Objectives

1. Face Recognition Model

- Reduce Manual Processes: Minimize manual intervention in attendance tracking by automating the identification and verification of student faces.

- Support Automation System: Support an automation system by seamlessly capturing attendance as students enter the webcam frame, enhancing efficiency in attendance management.

2. Flask Framework

- Support Automation System: Facilitate interactions between different components to support the automation system within SnapAttendance.
- Provide User-Friendly Interface: Offer a user-friendly interface for faculty members and administrators, ensuring ease of use and navigation.

3. SQLite Database

- Secure Data Management: Securely store and manage student data, attendance records, and other relevant information.

4. Additional Libraries

- Enhance the functionality and performance of SnapAttendance, supporting its objectives of reducing manual processes and facilitating automation.

Functionalities

1. Face Recognition Model

- Time-Saving: Save time by eliminating the need for manual attendance marking, allowing faculty members to focus on other tasks.
- Easy to Monitor: Provide a straightforward monitoring system, enabling faculty members to effortlessly track attendance records and identify any discrepancies.

2. Flask Framework

Rendering Multiple Webpages: Enable the rendering of multiple webpages together, creating a cohesive and intuitive user interface for SnapAttendance.

- Orchestrating Interactions: Orchestrate interactions between the frontend interface, database, and Face Recognition model, ensuring seamless integration and functionality.

- Quick Navigation: Enhance time-saving capabilities by enabling quick navigation and access to attendance records, improving overall efficiency.

- Real-Time Insights: Support easy monitoring by providing real-time insights and analytics on attendance data.

3. SQLite Database

- Efficient Data Storage: Efficiently store and retrieve student data and attendance records, supporting the automation system within SnapAttendance.

- Quick Data Access: Contribute to time-saving efforts by enabling quick data access and manipulation, enhancing overall efficiency.

- Integrated Querying: Ensure easy monitoring of attendance data through integrated querying and reporting capabilities.

4. Additional Libraries

Specialized Functionalities: Provide specialized functionalities such as image processing, data manipulation, and web development, complementing the core modules of SnapAttendance.

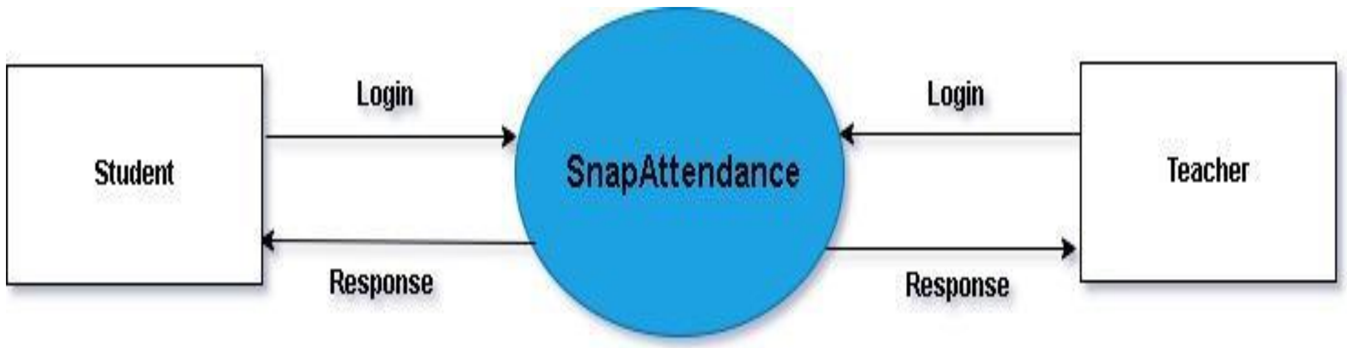
- Optimization: Optimize system performance and enable seamless integration of additional features, enhancing the overall user experience.

5. Detail Planning

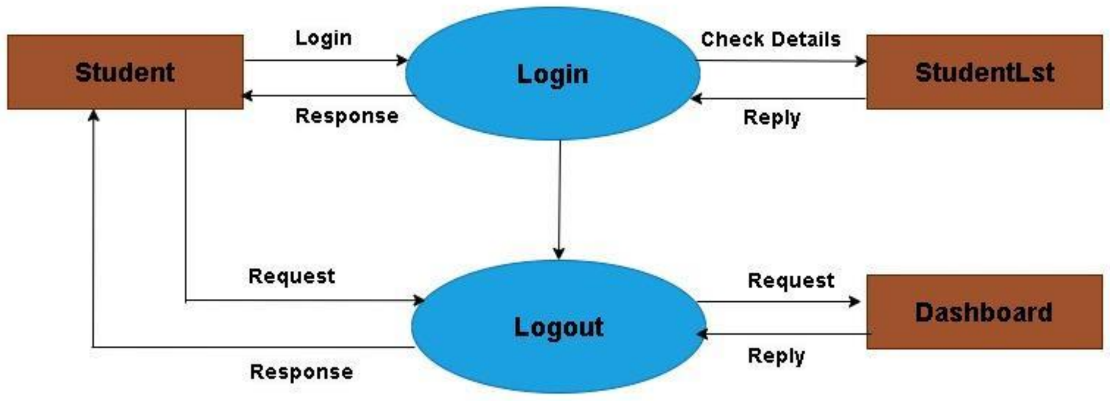
5.1 Data Flow Diagram / UML

- **0 Level DFD**

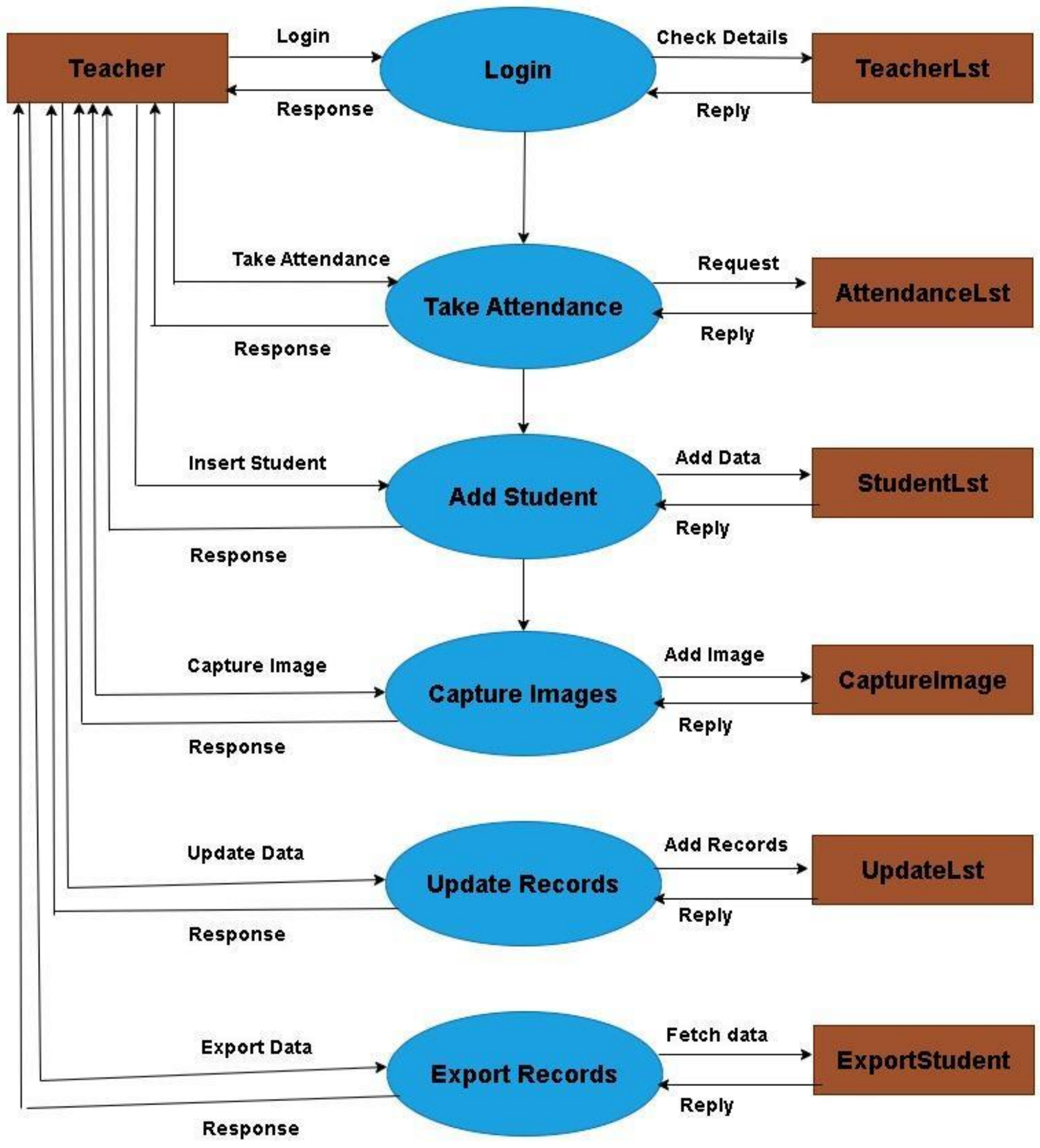
  Context diagram showing external entities (e.g., student login, teacher login).



- **1st Level DFD For Student**

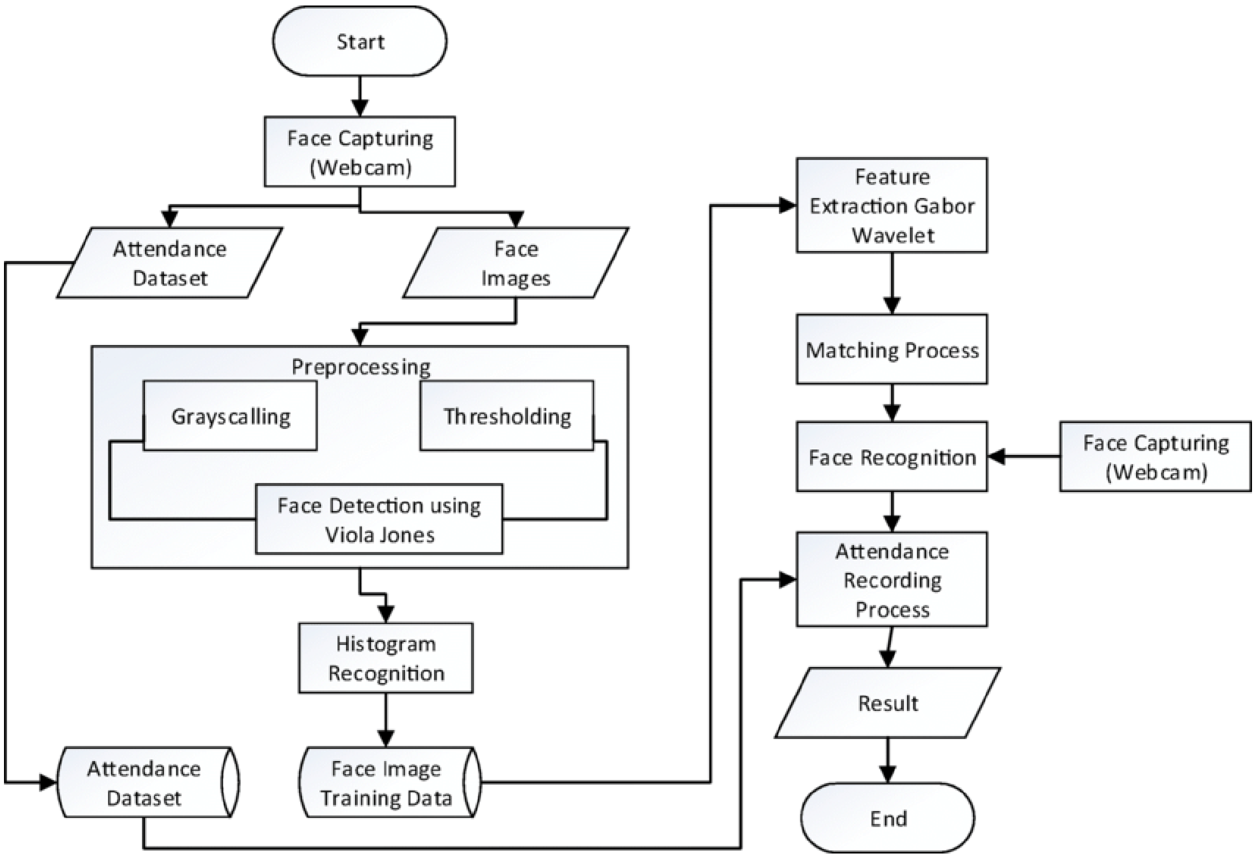


- **1st Level DFD For Teacher**



5.2 Process Specification / Activity Flow Diagram

These can be represented using activity flow diagrams, which describe the step-by-step flow of activities within a particular process.



5.3 Data Dictionary

1. students_record

1. **id**

- Description: Unique identifier for each student record.
- Data Type: Integer
- Constraints: Primary Key, Autoincrement

2. **student_id**

- Description: Unique identifier for each student.
- Data Type: Text
- Constraints: Not Null, Unique

3. **student_name**

- Description: Full name of the student.
- Data Type: Text
- Constraints: Not Null

4. **student_dob**

- Description: Date of birth of the student.
- Data Type: Date
- Constraints: Not Null

5. **roll_number**

- Description: Roll number of the student.
- Data Type: Text
- Constraints: Not Null, Unique

6. **student_class**

- Description: Class in which the student is enrolled.
- Data Type: Text
- Constraints: Not Null

7. **student_division**

- Description: Division of the student within the class.
- Data Type: Text
- Constraints: Not Null

8. **email**

- Description: Email address of the student.
- Data Type: Text
- Constraints: Not Null

9. **phone**

- Description: Phone number of the student.
- Data Type: Text
- Constraints: Not Null

2. students_photo

1. **id**

- Description: Unique identifier for each student photo entry.
- Data Type: Integer
- Constraints: Primary Key, Autoincrement

2. **student_id**

- Description: Identifier for the student associated with the photo.
- Data Type: Text
- Constraints: Not Null, Foreign Key (references students_record(student_id))

3. **photo_path**

- Description: Path to the photo file of the student.
- Data Type: Text
- Constraints: Not Null

3. attendance_entry

1. **id**

- Description: Unique identifier for each attendance entry.
- Data Type: Integer
- Constraints: Primary Key, Autoincrement

2. **student_id**

- Description: Identifier for the student whose attendance is recorded.
- Data Type: Integer
- Constraints: Foreign Key (references students_record(student_id))

3. **entry_time**

- Description: Time at which the attendance entry is recorded.
- Data Type: Datetime
- Constraints: Default Current Timestamp

4. **status**

- Description: Attendance status (Present or Absent) of the student.
- Data Type: Text
- Constraints: Check Constraint ('Present', 'Absent')

5. **recognize**

- Description: Recognition status (Recognized or Not Recognized) of the student's face.
- Data Type: Text
- Constraints: Check Constraint ('Recognized', 'Not Recognized')

6. **subject**

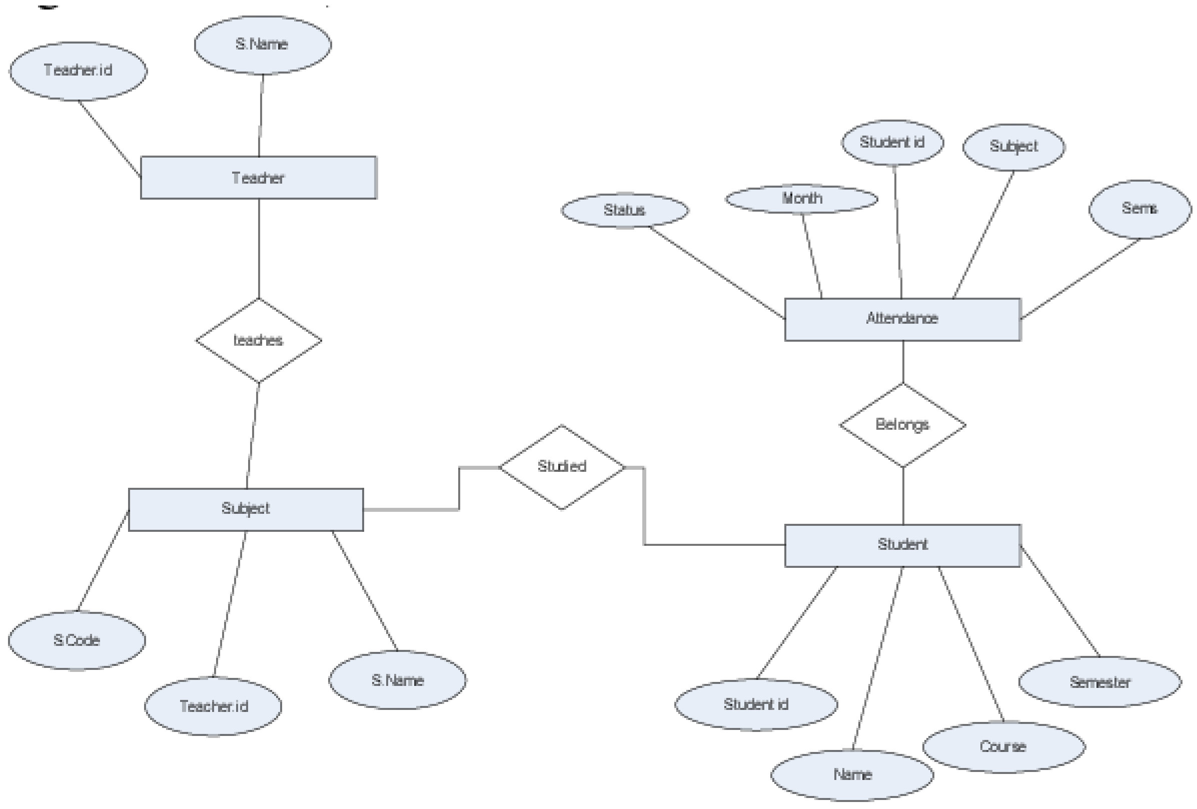
- Description: Subject for which the attendance is recorded.
- Data Type: Text

7. **teacher_name**

- Description: Name of the teacher who took the attendance.
- Data Type: Text

5.4 Entity-Relationship Diagram (ERD) / Class Diagram

An Entity-Relationship Diagram (ERD) or Class Diagram helps illustrate the relationships between different entities (in a database) or classes (in object-oriented programming). This is crucial for designing your database structure or software architecture.



6. System Design

6.1 Database Design

The database design for SnapAttendance plays a vital role in organizing and managing student data, attendance records, and other relevant information efficiently. It ensures data integrity, scalability, and performance while accommodating the system's requirements. Below is an overview of the database design for SnapAttendance

1. Tables

- students_record: This table stores information about students, including their unique student ID, name, date of birth, roll number, class, division, email, and phone number. The student ID serves as the primary key, ensuring each student record is uniquely identified.

- students_photo: This table maintains records of student photos captured for facial recognition purposes. It includes the student ID, which acts as a foreign key referencing the corresponding student record in the "students_record" table. The table also stores the path to the photo file.

- attendance_entry: This table records attendance entries for students. It includes the student ID, entry time, attendance status (Present or Absent), recognition status (Recognized or Not Recognized), subject, and the name of the teacher who took the attendance. The attendance entry ID serves as the primary key.

2. Relationships

- The "students_photo" table has a one-to-one relationship with the "students_record" table, where each student photo entry corresponds to one student record.

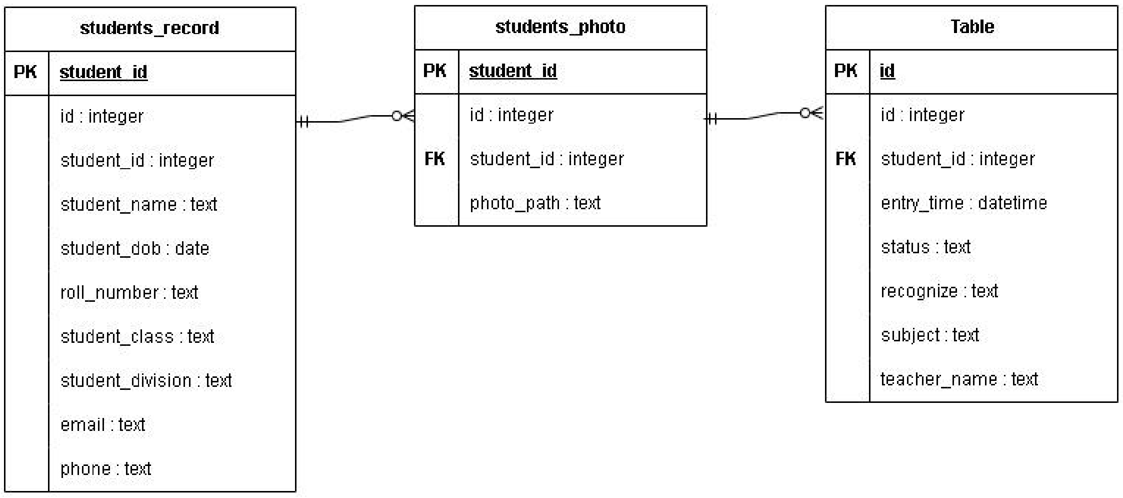
- The "attendance_entry" table has a many-to-one relationship with the "students_record" table, where multiple attendance entries can be associated with a single student record.

3. Constraints

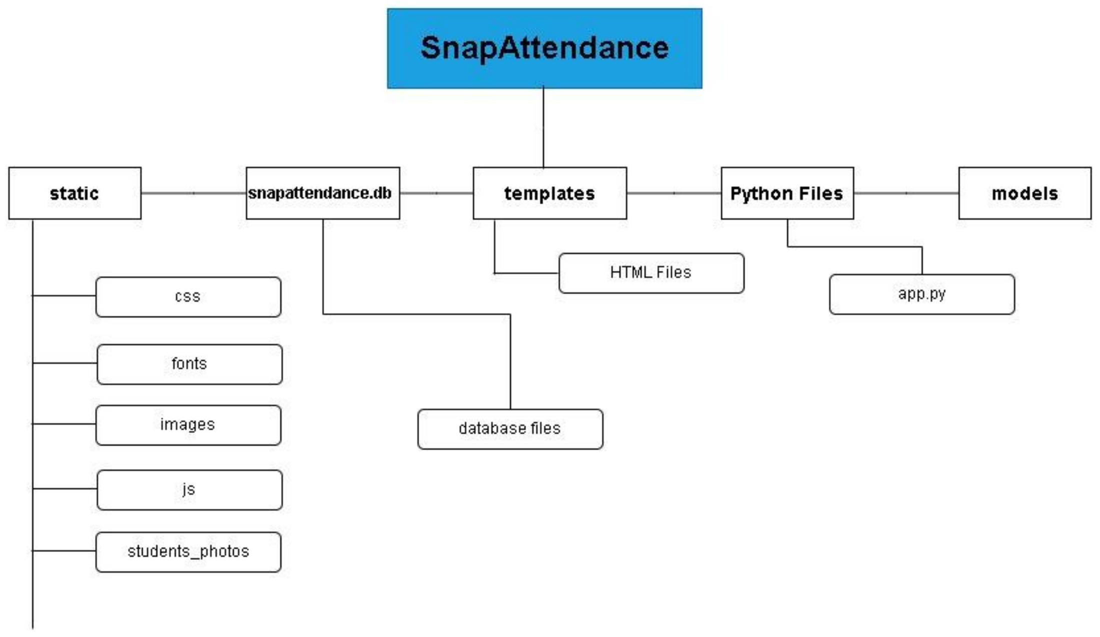
- The "student_id" field in the "students_record" table is defined as unique to ensure each student has a distinct identifier.
- The "student_id" field in the "students_photo" and "attendance_entry" tables serves as a foreign key referencing the corresponding student record in the "students_record" table, enforcing referential integrity.

4. Indexes

- Indexes may be added to columns frequently used in search queries or join operations, such as the "student_id" field in the "students_record" table or the "entry_time" field in the "attendance_entry" table, to improve query performance.

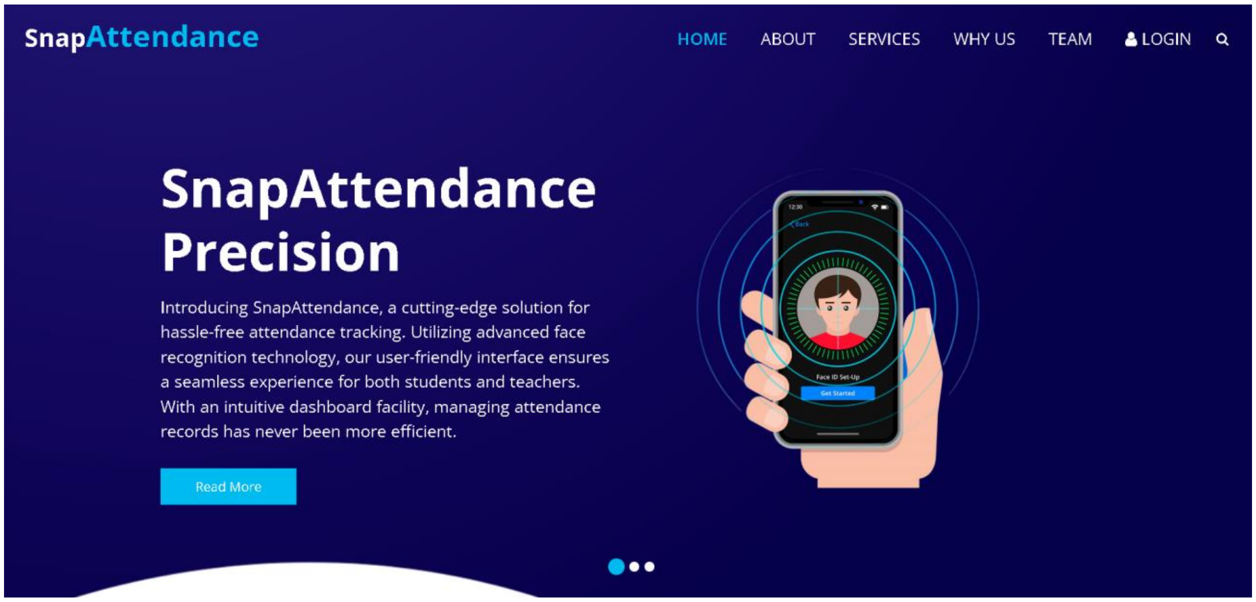


6.2 Directory Structure



6.3 Input Design

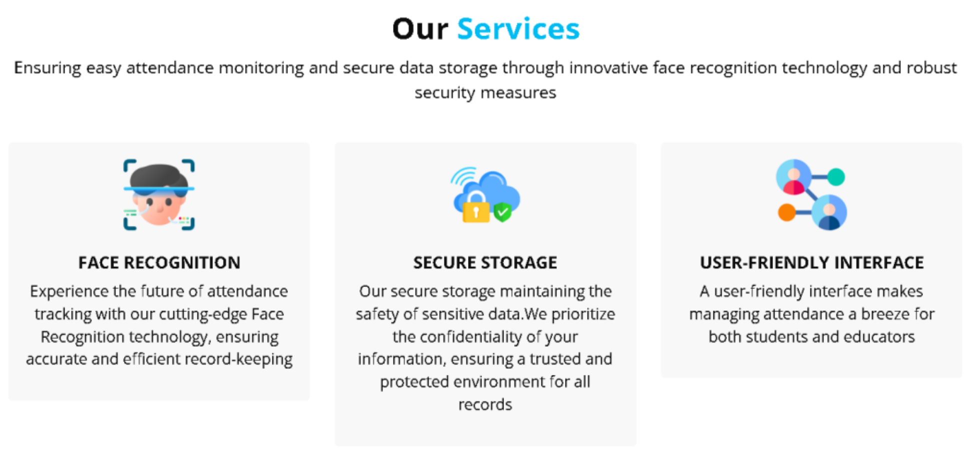
Landing Page



SnapAttendance Precision

Introducing SnapAttendance, a cutting-edge solution for hassle-free attendance tracking. Utilizing advanced face recognition technology, our user-friendly interface ensures a seamless experience for both students and teachers. With an intuitive dashboard facility, managing attendance records has never been more efficient.

Service Page



Our Services

Ensuring easy attendance monitoring and secure data storage through innovative face recognition technology and robust security measures

FACE RECOGNITION

Experience the future of attendance tracking with our cutting-edge Face Recognition technology, ensuring accurate and efficient record-keeping

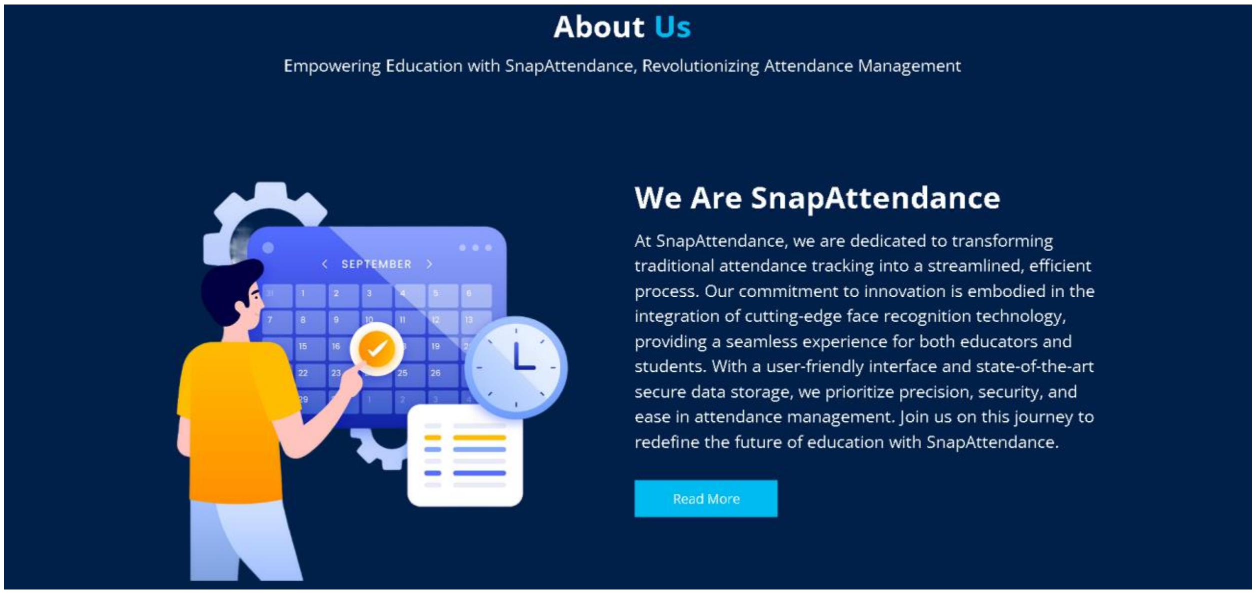
SECURE STORAGE

Our secure storage maintaining the safety of sensitive data.We prioritize the confidentiality of your information, ensuring a trusted and protected environment for all records

USER-FRIENDLY INTERFACE

A user-friendly interface makes managing attendance a breeze for both students and educators

About Us



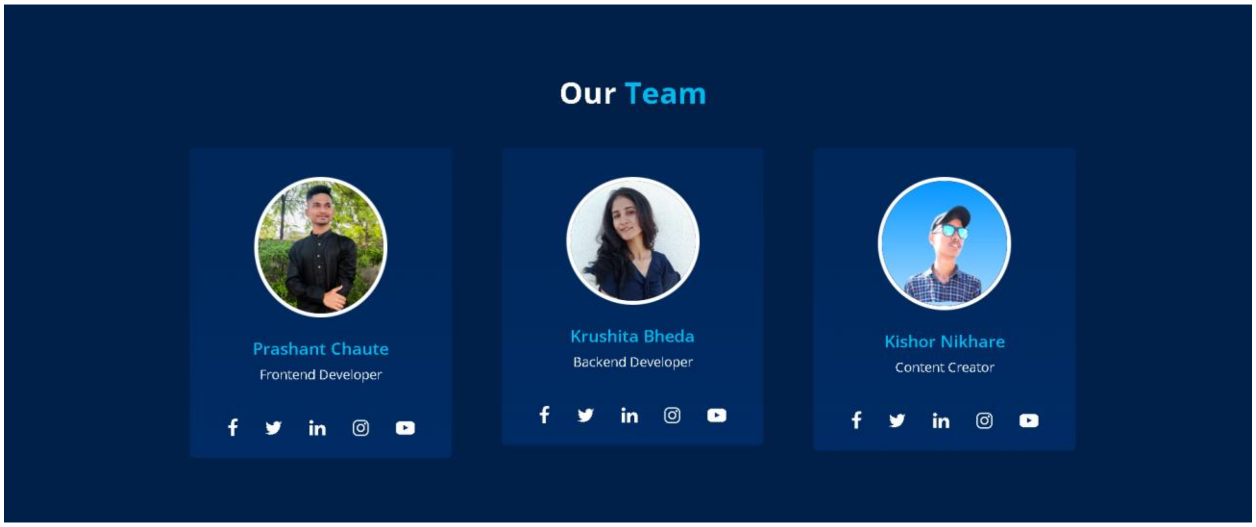
About Us

Empowering Education with SnapAttendance, Revolutionizing Attendance Management

We Are SnapAttendance

At SnapAttendance, we are dedicated to transforming traditional attendance tracking into a streamlined, efficient process. Our commitment to innovation is embodied in the integration of cutting-edge face recognition technology, providing a seamless experience for both educators and students. With a user-friendly interface and state-of-the-art secure data storage, we prioritize precision, security, and ease in attendance management. Join us on this journey to redefine the future of education with SnapAttendance.

Team



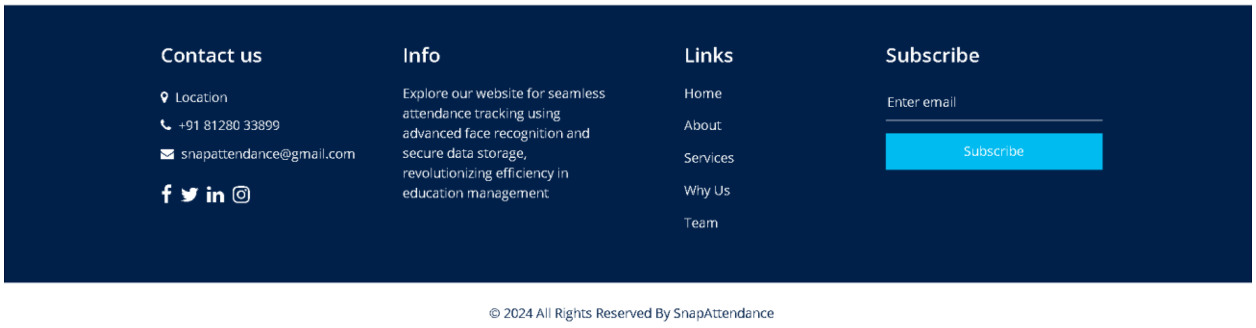
Our Team

Prashant Chaute
Frontend Developer

Krushita Bheda
Backend Developer

Kishor Nikhare
Content Creator

Contact Us



Contact us

Location

+91 81280 33899

snapattendance@gmail.com

Info

Explore our website for seamless attendance tracking using advanced face recognition and secure data storage, revolutionizing efficiency in education management

Links

Home

About

Services

Why Us

Team

Subscribe

Enter email

Subscribe

© 2024 All Rights Reserved By SnapAttendance

Logo

SnapAttendance

6.4 Coding

App.py

```
from flask import Flask, render_template, redirect, url_for, request, jsonify
import sqlite3
from Slogin import slogin
from attendance import attendance_bp
from AddStudent import AddStudent
from webcam_capture import webcam_capture_app
from UpdateRecords import update_records_app
from StudentRecords import student_records_app
from AttendanceRecord import attendance_records_app
from ExportRecords import export_records_app

app = Flask(__name__)
DATABASE = "snapattendance.db"

app.register_blueprint(slogin, url_prefix="/slogin")
app.register_blueprint(attendance_bp, url_prefix="/attendance")
app.register_blueprint(AddStudent, url_prefix="/add_student")
app.register_blueprint(webcam_capture_app, url_prefix="/webcam_capture_app")
app.register_blueprint(update_records_app, url_prefix="/update_records")
app.register_blueprint(student_records_app, url_prefix="/student_records")
app.register_blueprint(attendance_records_app, url_prefix="/attendance_records")
app.register_blueprint(export_records_app, url_prefix="/export_records")


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


@app.route("/login")
def login():
    return redirect(url_for("role_page"))


@app.route("/role")
def role_page():
    return render_template("Role.html")


@app.route("/student_login")
def student_login():
    return render_template("Slogin.html")


@app.route("/teacher_login")
def teacher_login():
    return render_template("Tlogin.html")


@app.route("/teacher_dashboard")
def teacher_dashboard():
    return render_template("Tactions.html")


@app.route("/attendance")
def attendance_app():
    return render_template("TakeAttendance.html")


@app.route("/add_student")
def add_student():
    return render_template("AddStudent.html")


@app.route("/submit", methods=["POST"])
def submit():
    if request.method == "POST":
        try:
            # Get form data
            student_id = request.form.get("StudentID")
            roll_number = request.form.get("rollNumber")
            student_name = request.form.get("StudentName")
            student_dob = request.form.get("StudentDOB")
            student_class = request.form.get("StudentClass")
            student_division = request.form.get("StudentDivision")
            email = request.form.get("email")
        phone = request.form.get("phone")

        conn = sqlite3.connect(DATABASE)
        cursor = conn.cursor()
        cursor.execute(
            """
            INSERT INTO students_record (
                student_id, student_name, student_dob, roll_number,
                student_class, student_division, email, phone
            ) VALUES (?, ?, ?, ?, ?, ?, ?, ?)
            """,
            (
                student_id,
                student_name,
                student_dob,
                roll_number,
                student_class,
                student_division,
                email,
                phone,
            ),
        )

        conn.commit()
        conn.close()

        return jsonify(
            {"status": "success", "message": "Student added successfully"}
        )

    except sqlite3.IntegrityError as e:
        return jsonify(
            {
                "status": "error",
                "message": "Duplicate entry for Student ID or Roll Number",
            }
        )
    except Exception as e:
        return jsonify({"status": "error", "message": str(e)})
```

```
@app.route("/webcam_capture_app/")
def webcam_capture_app():
    return render_template("Capture.html")


@app.route("/update_records")
def update_records():
    return render_template("UpdateRecords.html")


@app.route("/student_records")
def student_records():
    return render_template("StudentRecords.html")


@app.route("/attendance_records")
def attendance_records():
    return render_template("AttendanceRecord.html")


@app.route("/export_records")
def export_records():
    return render_template("ExportRecords.html")


if __name__ == "__main__":
    app.run(debug=True)
```

attendance.py

```
from flask import Blueprint, render_template, request
import cv2
import sqlite3
import face_recognition
from datetime import datetime
from flask_cors import CORS
from plyer import notification
import numpy as np
```

```
attendance_bp = Blueprint(
    "attendance", __name__
)
CORS(attendance_bp)

CORS(attendance_bp, resources={r"/attendance/take_attendance": {"origins": "*"}})


recognized_students_session = set()


recognized_students_current_session = set()


def take_attendance(subject, teacher_name):
    conn = sqlite3.connect("snapattendance.db")
    cursor = conn.cursor()

    cursor.execute("SELECT student_id, photo_path FROM students_photo")
    rows = cursor.fetchall()

    known_face_encodings = []
    known_student_ids = []

    for student_id, photo_path in rows:
        img = face_recognition.load_image_file(photo_path)

        face_encodings = face_recognition.face_encodings(img)
        if face_encodings:
            face_encoding = face_encodings[0]
            known_face_encodings.append(face_encoding)
            known_student_ids.append(student_id)


    video_capture = cv2.VideoCapture(0)  # Use 0 for the default webcam
    video_capture.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)  # Set the frame width
    video_capture.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)  # Set the frame height

    timer_start = None
```

```
while True:
    ret, frame = video_capture.read()

    current_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    font = cv2.FONT_HERSHEY_SIMPLEX
    cv2.putText(frame, current_time, (10, 30), font, 0.6, (0, 255, 0), 1)

    status = "Absent"

    face_locations = face_recognition.face_locations(frame)
    face_encodings = face_recognition.face_encodings(frame, face_locations)

    for (top, right, bottom, left), face_encoding in zip(
        face_locations, face_encodings
    ):
        distances = face_recognition.face_distance(
            known_face_encodings, face_encoding
        )

        min_distance_index = np.argmin(distances)
        min_distance = distances[min_distance_index]

        if min_distance < 0.5:
            student_id = known_student_ids[min_distance_index]

            if student_id not in recognized_students_current_session:

                recognized_students_current_session.add(student_id)

                if student_id not in recognized_students_session:

                    recognized_students_session.add(student_id)


                    entry_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
                    cursor.execute(
                        """
                        INSERT INTO attendance_entry
                        (student_id, entry_time, status, recognize, subject, teacher_name)
    VALUES (?, ?, ?, ?, ?, ?)
""",
    (
        student_id,
        entry_time,
        "Present",
        "Recognized",
        subject,
        teacher_name,
    ),
)

notification_title = f"Student {student_id} Attendance"
notification_message = (
    f"Attendance recorded for Student ID: {student_id}"
)
notification.notify(
    title=notification_title,
    message=notification_message,
    app_icon=None,
    timeout=10,
)


rectangle_color = (255, 0, 0)
cv2.rectangle(
    frame, (left, top), (right, bottom), rectangle_color, 2
)

cv2.putText(
    frame,
    f"Student ID: {student_id}",
    (left + 5, bottom + 20),
    font,
    0.6,
    rectangle_color,
    1,
)
```

```
            status = "Present"

            timer_start = datetime.now()

    else:
        rectangle_color = (0, 0, 255)  # Red color for unknown faces
        cv2.rectangle(frame, (left, top), (right, bottom), rectangle_color, 2)
        cv2.putText(
            frame,
            "Unknown",
            (left + 5, bottom + 20),
            font,
            0.6,
            rectangle_color,
            1,
        )

cv2.putText(frame, f"Status: {status}", (1000, 50), font, 0.6, (255, 0, 0), 1)
cv2.putText(
    frame, "Recognition: Recognized", (1000, 80), font, 0.6, (255, 0, 0), 1
)
cv2.putText(
    frame, f"Student ID: {student_id}", (1000, 110), font, 0.6, (255, 0, 0), 1
)

if timer_start:

    elapsed_time = (datetime.now() - timer_start).total_seconds()

    if elapsed_time >= 3:
        recognized_students_current_session.clear()
        timer_start = None

conn.commit()

cv2.imshow("Attendance", frame)

if cv2.waitKey(1) & 0xFF == ord("q"):
    break
    video_capture.release()
    cv2.destroyAllWindows()

    recognized_students_session.clear()

    conn.close()


@attendance_bp.route("/attendance")
def index():
    return render_template("TakeAttendance.html")

@attendance_bp.route("/take_attendance", methods=["POST"])
def take_attendance_route():
    print("Take Attendance route reached.")
    subject = request.form.get("subject")
    teacher_name = request.form.get("teacher")
    print(f"Subject: {subject}, Teacher: {teacher_name}")

    take_attendance(subject, teacher_name)

    return "Attendance taken successfully!", 200
```

AddStudent.py

```
from flask import Blueprint, render_template, request, jsonify, redirect, url_for
import sqlite3


AddStudent = Blueprint("AddStudent", __name__)

DATABASE = "snapattendance.db"

def create_table():
    conn = sqlite3.connect(DATABASE)
    cursor = conn.cursor()

    cursor.execute(
        """
        CREATE TABLE IF NOT EXISTS students_record (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            student_id TEXT NOT NULL UNIQUE,
            student_name TEXT NOT NULL,
            student_dob DATE NOT NULL,
            roll_number TEXT NOT NULL UNIQUE,
            student_class TEXT NOT NULL,
            student_division TEXT NOT NULL,
            email TEXT NOT NULL,
            phone TEXT NOT NULL
        )
    """
    )

    conn.commit()
    conn.close()

@AddStudent.route("/")
def index():
    return render_template("AddStudent.html")
```

webcam_capture.py

```
import cv2
import os
import sqlite3
from flask import Blueprint, render_template, request, jsonify
import face_recognition

webcam_capture_app = Blueprint("webcam_capture_app", __name__)

def capture_images(student_id, num_images=15):
    cap = cv2.VideoCapture(0)
    if not cap.isOpened():
        print("Error: Unable to access the webcam")
        return False, {"error": "Unable to access the webcam"}

    folder_path = create_student_folder(student_id)
    for image_count in range(1, num_images + 1):
        ret, frame = cap.read()
        if not ret:
            print("Error: Failed to capture frame from the webcam")
            return False, {"error": "Failed to capture frame from the webcam"}

        image_path = os.path.join(folder_path, f"{student_id}_{image_count}.jpg")
        try:
            cv2.imwrite(image_path, frame)
            print(f"Image {image_count} captured for student ID: {student_id}")

            db_connection = sqlite3.connect("snapattendance.db")
            db_cursor = db_connection.cursor()

            db_cursor.execute(
                "INSERT INTO students_photo (student_id, photo_path) VALUES (?, ?)",
                (student_id, image_path),
            )
            db_connection.commit()
            db_connection.close()

        except Exception as e:
            print(f"Error capturing image: {e}")
            return False, {"error": f"Error capturing image: {e}"}

    cap.release()
    cv2.destroyAllWindows()

    return True, {"success": f"Images captured and stored for student ID: {student_id}"}


def create_student_folder(student_id):
    folder_path = os.path.join("static", "students_photos", student_id)
    os.makedirs(folder_path, exist_ok=True)
    return folder_path


@webcam_capture_app.route("/")
def index():
    return render_template("Capture.html")
```

```
@webcam_capture_app.route("/capture", methods=["POST"])
def capture_images_route():
    student_id = request.form.get("studentId")

    db_connection = sqlite3.connect("snapattendance.db")
    db_cursor = db_connection.cursor()
    db_cursor.execute(
        "SELECT * FROM students_record WHERE student_id = ?", (student_id,)
    )
    existing_student = db_cursor.fetchone()
    db_connection.close()

    if existing_student:
        success, message = capture_images(student_id)
        if success:
            return jsonify(message)
        else:
            return jsonify(message)
    else:
        return jsonify({"error": "Student not found"})
```

ExportRecords.py

```
from flask import Blueprint, render_template, request, Response
import sqlite3
import csv
from io import StringIO

export_records_app = Blueprint("export_records", __name__)

@export_records_app.route("/")
def export_records():
    return render_template("ExportRecords.html")

@export_records_app.route("/export_csv", methods=["POST"])
def export_csv():
    from_id = request.form.get("from")
    to_id = request.form.get("to")
    records = fetch_records_from_db(from_id, to_id)

    csv_data = generate_csv(records)

    return Response(
        csv_data,
        mimetype="text/csv",
        headers={"Content-disposition": "attachment; filename=studentrecords.csv"},
    )


def fetch_records_from_db(from_id, to_id):
    conn = sqlite3.connect("snapattendance.db")
    cursor = conn.cursor()

    cursor.execute(
        "SELECT * FROM students_record WHERE student_id BETWEEN ? AND ?",
        (from_id, to_id),
    )
    records = cursor.fetchall()

    conn.close()

    return records

def generate_csv(records):
    csv_data = StringIO()
    csv_writer = csv.writer(csv_data)

    csv_writer.writerow(
        [
            "id",
            "student_id",
            "student_name",
            "student_dob",
            "roll_number",
            "student_class",
            "student_division",
            "email",
            "phone",
        ]
    )

    csv_writer.writerows(records)

    return csv_data.getvalue()


@export_records_app.route("/export_attendance_csv", methods=["POST"])
def export_attendance_csv():
    student_id = request.form.get("studentID")
    attendance_date = request.form.get("attendanceDate")

    attendance_records            =            fetch_attendance_records_from_db(student_id,
attendance_date)

    csv_data = generate_attendance_csv(attendance_records)

    return Response(
        csv_data,
        mimetype="text/csv",
        headers={"Content-disposition": "attachment; filename=attendance_records.csv"},
    )


def fetch_attendance_records_from_db(student_id, attendance_date):
    conn = sqlite3.connect("snapattendance.db")
    cursor = conn.cursor()

    cursor.execute(
        "SELECT * FROM attendance_entry WHERE student_id = ? AND date(entry_time)
= ?",
        (student_id, attendance_date),
    )
    attendance_records = cursor.fetchall()

    conn.close()

    return attendance_records
```

```
def generate_attendance_csv(attendance_records):
    csv_data = StringIO()
    csv_writer = csv.writer(csv_data)

    csv_writer.writerow(
        [
            "id",
            "student_id",
            "entry_time",
            "status",
            "recognize",
            "subject",
            "teacher_name",
        ]
    )

    csv_writer.writerows(attendance_records)

    return csv_data.getvalue()


@export_records_app.route("/export_class_div_csv", methods=["POST"])
def export_class_div_csv():
    from_id = request.form.get("from2")
    to_id = request.form.get("to2")
    selected_date = request.form.get("dateSelect")

    records = fetch_class_div_records_from_db(from_id, to_id, selected_date)

    csv_data = generate_attendance_csv(records)

    return Response(
        csv_data,
        mimetype="text/csv",
        headers={"Content-disposition": "attachment; filename=class_div_records.csv"},
    )

def fetch_class_div_records_from_db(from_id, to_id, selected_date):
    conn = sqlite3.connect("snapattendance.db")
    cursor = conn.cursor()
    cursor.execute(
        "SELECT * FROM attendance_entry WHERE student_id BETWEEN ? AND ? AND date(entry_time) = ?",
        (from_id, to_id, selected_date),
    )
    records = cursor.fetchall()

    conn.close()

    return records
```

StudentRecords.py

```
from flask import Blueprint, render_template, request, redirect, url_for
import sqlite3

student_records_app = Blueprint("student_records", __name__)

def get_student_records(search_term=None):
    try:
        connection = sqlite3.connect("snapattendance.db")
        cursor = connection.cursor()

        query = "SELECT * FROM students_record"
        if search_term:
            query += f" WHERE student_id LIKE '{search_term}%'"

        cursor.execute(query)

        records = cursor.fetchall()

        cursor.close()
        connection.close()

        return records

    except Exception as e:
        print("Error:", str(e))
        return []
```

```
@student_records_app.route("/student_records", methods=["GET", "POST"])
def student_records():
    if request.method == "POST":
        search_term = request.form.get("search_term")
        return redirect(
            url_for("student_records.student_records", search_term=search_term)
        )

    search_term = request.args.get("search_term")
    records = get_student_records(search_term)
    return render_template(
        "StudentRecords.html", records=records, search_term=search_term
    )
```

AttendanceRecords.py

```
from flask import Blueprint, render_template, request
import sqlite3

attendance_records_app = Blueprint("attendance_records", __name__)

def fetch_all_attendance_records():
    conn = sqlite3.connect("snapattendance.db")
    cursor = conn.cursor()

    query = """
    SELECT * FROM attendance_entry
    """

    cursor.execute(query)
    records = cursor.fetchall()

    conn.close()
    return records


def fetch_attendance_records(student_id):
    conn = sqlite3.connect("snapattendance.db")
    cursor = conn.cursor()
    query = """
    SELECT * FROM attendance_entry
    WHERE student_id = ?
    """

    cursor.execute(query, (student_id,))
    records = cursor.fetchall()

    conn.close()
    return records

@attendance_records_app.route("attendance_records", methods=["GET", "POST"])
def attendance_records():
    if request.method == "POST":
        search_term = request.form.get("search_term")
        records = fetch_attendance_records(search_term)
        return render_template(
            "AttendanceRecord.html", records=records, search_term=search_term
        )

    records = fetch_all_attendance_records()
    return render_template("AttendanceRecord.html", records=records,
search_term=None)
```

UpdateRecords.py

```
from flask import Blueprint, render_template, request, jsonify
import sqlite3

update_records_app = Blueprint("update_records_app", __name__)


def fetch_student_data(student_id):
    connection = sqlite3.connect("snapattendance.db")
    cursor = connection.cursor()

    cursor.execute("SELECT * FROM students_record WHERE student_id = ?",
(student_id,))
```

```
    student_data = cursor.fetchone()

    connection.close()

    return student_data

def update_student_record_in_db(data):
    connection = sqlite3.connect("snapattendance.db")
    cursor = connection.cursor()

    cursor.execute(
        """
        UPDATE students_record
        SET student_name=?, student_dob=?, roll_number=?, student_class=?,
            student_division=?, email=?, phone=?
        WHERE student_id=?
    """,
        (
            data["student_name"],
            data["student_dob"],
            data["roll_number"],
            data["student_class"],
            data["student_division"],
            data["email"],
            data["phone"],
            data["student_id"],
        ),
    )

    connection.commit()
    connection.close()

def delete_student_record_from_db(student_id):
    connection = sqlite3.connect("snapattendance.db")
    cursor = connection.cursor()

    cursor.execute("DELETE FROM students_record WHERE student_id = ?",
(student_id,))

    connection.commit()
    connection.close()


@update_records_app.route("/update_records")
def update_records_form():
    return render_template("UpdateRecords.html")


@update_records_app.route("/fetch_student_data", methods=["POST"])
def fetch_student_data_route():
    student_id = request.form.get("student_id")
    student_data = fetch_student_data(student_id)

    return jsonify(student_data)


@update_records_app.route("/update_student_record", methods=["POST"])
def update_student_record():
    try:
        data = request.get_json()

        if not all(
            key in data
            for key in [
                "student_id",
                "student_name",
                "student_dob",
                "roll_number",
                "student_class",
                "student_division",
                "email",
                "phone",
            ]
        ):
            raise ValueError("Invalid data format")

        update_student_record_in_db(data)

        return jsonify({"message": "Student record updated successfully"})
    except Exception as e:
        return jsonify({"error": str(e)}), 500



@update_records_app.route("/delete_student_record", methods=["POST"])
def delete_student_record():
    student_id = request.form.get("student_id")
    delete_student_record_from_db(student_id)

    return jsonify({"message": "Student record deleted successfully"})
```

Slogin.py

```
from flask import Blueprint, render_template, request, redirect, url_for
import sqlite3

slogin = Blueprint('slogin', __name__)


@slogin.route("/", methods=["GET", "POST"])
def index():
    if request.method == "POST":
        return redirect(url_for("student_login"))
    return render_template("Slogin.html", message="")

def get_connection():
    try:
        connection = sqlite3.connect("snapattendance.db", check_same_thread=False)
        return connection
    except sqlite3.Error as e:
        print(e)
    return None


def check_student_login(connection, student_id, roll_number):
    try:
        cursor = connection.cursor()
        query = "SELECT * FROM students_record WHERE student_id = ? AND
roll_number = ?"
```

```
        cursor.execute(query, (student_id, roll_number))
        student = cursor.fetchone()
        return student
    except sqlite3.Error as e:
        print(f"Error checking login credentials: {e}")
        return None


@slogin.route("/student-login", methods=["GET", "POST"])
def student_login():
    if request.method == "POST":
        connection = get_connection()

        if connection:
            student_id = request.form.get("sid")
            roll_number = request.form.get("roll")

            student = check_student_login(connection, student_id, roll_number)

            if student:
                return redirect(
                    url_for(
                        "slogin.student_dashboard",
                        success_message="Login successful",
                        student_id=student_id,
                    )
                )
            else:
                return render_template("Slogin.html", message="Invalid credentials")

        return render_template("Slogin.html")

    return render_template("Slogin.html")


@slogin.route("/student-dashboard")
def student_dashboard():
    success_message = request.args.get("success_message", None)
    student_id = request.args.get("student_id", None)
    if success_message and student_id:
        connection = get_connection()
        cursor = connection.cursor()
        student_query = "SELECT * FROM students_record WHERE student_id = ?"
        cursor.execute(student_query, (student_id,))
        student_details = cursor.fetchone()

        recent_records_query = "SELECT * FROM attendance_entry WHERE student_id =
? ORDER BY entry_time DESC LIMIT 8"
        cursor.execute(recent_records_query, (student_id,))
        recent_records = cursor.fetchall()

        total_lectures = 8
        attendance_count = sum(1 for record in recent_records if record[3] == 'Present')

        attendance_percentage = (attendance_count / total_lectures) * 100 if total_lectures
> 0 else 0

        connection.close()

        if student_details:
            student_name, roll_number, student_class, student_division = (
                student_details[2],
                student_details[4],
                student_details[5],
                student_details[6],
            )

            return render_template(
                "Sdashboard.html",
                success_message=success_message,
                student_id=student_id,
                student_name=student_name,
                roll_number=roll_number,
                student_class=student_class,
                student_division=student_division,
                recent_records=recent_records,
                attendance_count=attendance_count,
                attendance_percentage=attendance_percentage,
            )
        else:
            return redirect(url_for("student_login"))
    else:
        return redirect(url_for("student_login"))

@slogin.route("/logout")
def logout():
    return redirect(url_for("student_login"))
```

TakeAttendance.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Take Attendance</title>
    <style>
      body {
        font-family: "Arial", sans-serif;
        margin: 0;
        padding: 0;
        background-color: #f4f4f4;
        text-align: center;
      }

      .container {
        max-width: 70%;
        margin: 5% 13%;
        background-color: #fff;
        padding: 20px;
        border-radius: 20px;
        box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
      }

      h1 {
        color: #333;
      }
```

```
select {
  padding: 10px;
  font-size: 16px;
  margin-bottom: 20px;
  margin-right: 1%;
}

#subject,
#teacher {
  border-radius: 15px;
}

#takeAttendanceButton {
  padding: 15px 30px;
  font-size: 18px;
  font-weight: bold;
  background-color: #09158f;
  color: #fff;
  border: none;
  border-radius: 25px;
  cursor: pointer;
  margin-top: 2%;
  margin-right: 1%;
}

.back-button {
  position: absolute;
  top: 10px;
  font-size: 16px;
  font-weight: bold;
  left: 10px;
  background-color: #231a6f;
  color: white;
  border: none;
  padding: 10px 20px;
  cursor: pointer;
  transition: background-color 0.3s;
}

.back-button:hover {
  background-color: #0f054c;
}

table {
  width: 90%;
  border-collapse: collapse;
  margin-top: 5%;
  margin-left: 5%;
}

th,
td {
  border: 1px solid #ddd;
  padding: 8px;
  text-align: left;
}

th {
  background-color: #f2f2f2;
}

@media only screen and (max-width: 600px) {
  .container {
    padding: 20px;
    margin: 20% 5%;
    max-width: 100%;
  }

  select {
    width: 83%;
    margin-top: 2%;
  }

  #takeAttendanceButton {
    width: 65%;
    margin-bottom: 3%;
  }

  .back-button {
    width: 30%;
```

```
        height: 6%;
        position: absolute;
        font-size: 16px;
        font-weight: bold;
      }
    }
  </style>
</head>

<body>
  <button class="back-button" onclick="goBack()">Back</button>
  <div class="container">
    <h1>Take Attendance</h1>

    <label for="subject">Subject :</label>
    <select id="subject" name="subject">
      <option value="Python" selected>Python</option>
      <option value="DBMS">DBMS</option>
      <option value="Web Designing">Web Designing</option>
      <option value="Java Programming">Java Programming</option>
    </select>
    <label for="teacher">Faculty Name :</label>
    <select id="teacher" name="teacher">
      <option value="Prof. Nisha M." selected>Prof. Nisha M.</option>
      <option value="Prof. Dhaval T.">Prof. Dhaval T.</option>
      <option value="Prof. Hitesh S.">Prof. Hitesh S.</option>
      <option value="Prof. Rahul K.">Prof. Rahul K.</option>
    </select>

    <button id="takeAttendanceButton" onclick="startFaceRecognition()">
      Take Attendance
    </button>
  </div>

  <script src="https://code.jquery.com/jquery-3.6.4.min.js"></script>

  <script>
    function goBack() {
      window.history.back();
    }
```

```
      function startFaceRecognition() {
        console.log("Start Face Recognition button clicked.");
        var subject = document.getElementById("subject").value;
        var teacher = document.getElementById("teacher").value;
        console.log("Subject:", subject);
        console.log("Teacher:", teacher);

        $.ajax({
          type: "POST",
          url: "/attendance/take_attendance",
          data: { subject: subject, teacher: teacher },
          success: function (response) {
            console.log("Attendance taken successfully!");
            alert("Attendance taken successfully!");
          },
          error: function (error) {
            console.error("Error taking attendance:", error);
            alert("Error taking attendance!");
          },
        });
      }
    </script>
  </body>
</html>
```

Capture.html

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-
scale=1.0, user-scalable=no">
    <title>Capture Image</title>
    <script src="https://code.jquery.com/jquery-3.6.4.min.js"></script>
    <style>
        body {
```

```
        font-family: Arial, sans-serif;
        text-align: center;
        margin: 0;
        padding: 0;
        background-image: linear-gradient(25deg, #cf4633, #be5368, #a35f9c,
#706bd2);
        height: 82.3vh;
        background-repeat: no-repeat;
        background-size: cover;
     }

     .container {
        max-width: 700px;
        margin: 8% auto;
        padding: 20px;
        background-color: white;
        box-shadow: 0 0 30px rgba(255, 255, 255, 0.5);
        border-radius: 25px;
     }

     h1 {
        color: #333;
     }

     label {
        display: block;
        margin-bottom: 20px;
        font-size: 18px;
        color: #333;
        text-align: left;
        font-weight: bold;
     }

     input {
        width: 100%;
        padding: 10px;
        margin-bottom: 20px;
        box-sizing: border-box;
        border-bottom: 2px solid #25BE84;
        font-size: 16px;
```

```
    border-top: none;
    border-left: none;
    border-right: none;
    transition: border-bottom 0.5s ease;
}

input:hover {
    border-bottom: 2px solid #0f054c;
}

button {
    background-color: #231a6f;
    color: white;
    padding: 12px 20px;
    font-size: 18px;
    font-weight: bold;
    border: none;
    border-radius: 10px;
    cursor: pointer;
    transition: background-color 0.3s;
}

button:hover {
    background-color: #0f054c;
}

.back-button {
    position: absolute;
    top: 10px;
    left: 10px;
    padding: 10px 20px;
    font-size: 16px;
    font-weight: bold;
    cursor: pointer;
    color: white;
    background-color: #231a6f;
    border: none;
    transition: background-color 0.3s;
    text-decoration:none;
}
```

```
        .back-button:hover {
            background-color: #0f054c;
        }

        @media screen and (max-width: 600px) {
            body{
                height: 90vh;
            }
            .container {
                margin: 30% 5%;
                padding: 30px;
                box-shadow: 0 0 15px rgba(255, 255, 255, 0.8);
            }

            h1 {
                font-size: 24px;
            }

            label {
                font-size: 16px;
            }

            input {
                font-size: 14px;
            }

            button {
                font-size: 16px;
                padding: 15px;
            }

            .back-button {
                padding: 15px 25px;
                font-size: 16px;
            }
        }
    </style>
</head>
```

```
<body>
    <a class="back-button" href="{{ url_for('teacher_dashboard')}}">Back</a>
    <div class="container">
        <h1>Capture Student Image</h1>
        <label for="studentId">Student ID:</label>
        <input type="text" id="studentId" placeholder="Enter Student ID" required>
        <button onclick="captureImage()">Capture Image</button>
    </div>

    <script>
        function captureImage() {
            var studentId = document.getElementById('studentId').value;

            $.post("/webcam_capture_app/capture", { studentId: studentId }, function (data) {
                alert(data.success || data.error);
            });
        }
        function goBack() {
            window.history.back();
        }
    </script>
</body>

</html>
```

AddStudent.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta
      name="viewport"
      content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-
scalable=no"
    />
    <title>SnapAttendance</title>
    <script src="https://code.jquery.com/jquery-3.6.4.min.js"></script>
    <style>
```

```
body {
  background: wheat;
  font-family: "Open Sans", sans-serif;
  margin: 0;
  padding: 0;
}

.container {
  max-width: 960px;
  margin: 8% auto;
  padding: 20px;
}

h1 {
  font-size: 28px;
  text-align: center;
  margin-bottom: 3%;

  small {
    display: block;
    font-size: 15px;
    padding-top: 8px;
    color: gray;
  }
}

.back-button {
  position: absolute;
  top: 10px;
  left: 10px;
  padding: 10px 20px;
  font-size: 16px;
  font-weight: bold;
  cursor: pointer;
  color: white;
  background-color: #231a6f;
  border: none;
  transition: background-color 0.3s;
}
```

```
.back-button:hover {
  background-color: #0f054c;
}

.transparent-form {
  padding: 20px;
  border-radius: 10px;
  text-align: center;
  margin-top: -2%;
}

.transparent-form input {
  width: 40%;
  padding: 10px;
  font-size: 16px;
  background: transparent;
  margin-bottom: 15px;
  border-bottom: 2px solid #515e5e;
  border-top: none;
  border-left: none;
  border-right: none;
  outline: none;
}

.transparent-form input:hover {
  border-bottom: 2px solid #0659f4;
}

.transparent-form button {
  font-size: 24px;
  color: white;
  padding: 1%;
  font-family: inherit;
  font-weight: 600;
  cursor: pointer;
  position: relative;
  border: none;
  background: none;
  background-color: black;
  text-transform: capitalize;
  transition-timing-function: cubic-bezier(0.25, 0.8, 0.25, 1);
  transition-duration: 200ms;
  transition-property: color;
  border-radius: 5px;
}

.transparent-form button:focus,
.transparent-form button:hover {
  color: black;
  background: none;
  transition: 0.3s;
}

.transparent-form button:focus:after,
.transparent-form button:hover:after {
  width: 100%;
  left: 0%;
}

.transparent-form button:after {
  content: "";
  pointer-events: none;
  bottom: -2px;
  left: 50%;
  position: absolute;
  width: 0%;
  height: 2px;
  background-color: #0659f4;
  transition-timing-function: cubic-bezier(0.25, 0.8, 0.25, 1);
  transition-duration: 400ms;
  transition-property: width, left;
}

@media (max-width: 600px) {
  .transparent-form input {
    width: 100%;
    font-size: 16px;
  }

  .transparent-form button {
        padding: 12px 20px;
      }

      .transparent-form div {
        display: block;
        /* Show the div on smaller screens */
        margin-right: 67%;
        font-size: 16px;
        font-weight: bold;
      }
    }

    @media (min-width: 601px) {
      .transparent-form div {
        display: none;
      }
    }
  </style>
</head>

<body>
  <div class="container">
    <h1>Add Student</h1>
    <button class="back-button" onclick="goBack()">Back</button>
    <div class="transparent-form">
      <form
        onsubmit="return submitForm()"
        id="myForm"
        enctype="multipart/form-data"
        action="/submit"
        method="POST"
      >
        <input
          type="text"
          id="StudentID"
          placeholder="Enter Student ID"
          name="StudentID"
        />
        <input
          type="text"
  id="StudentName"
  placeholder="Enter Student Name"
  name="StudentName"
/>
<div style="margin-bottom: 5px">Date of Birth</div>
<input
  type="date"
  id="StudentDOB"
  placeholder="Select DOB"
  name="StudentDOB"
/>
<input
  type="text"
  id="rollNumber"
  placeholder="Enter Roll Number"
  name="rollNumber"
/>
<input
  type="text"
  id="StudentClass"
  placeholder="Enter Student Class"
  name="StudentClass"
/>
<input
  type="text"
  id="StudentDivision"
  placeholder="Enter Student Division"
  name="StudentDivision"
/>
<input
  type="email"
  id="email"
  placeholder="Enter Student Email ID"
  name="email"
/>
<input
  type="tel"
  id="phone"
  placeholder="Enter Phone Number"
  name="phone"
```

```
      />
      <br />
      <button type="submit">Submit</button>
    </form>
  </div>
</div>
<script>
  function submitForm() {
    // 1. Add "STUD" prefix to Student ID
    var studentID = "STUD" + $("#StudentID").val();
    if (!studentID) {
      alert("Student ID is compulsory.");
      return false;
    }

    // 2. Capitalize the first letter of Student Name
    var studentName = $("#StudentName")
      .val()
      .replace(/\b\w/g, (c) => c.toUpperCase());
    if (!studentName) {
      alert("Student Name is compulsory.");
      return false;
    }

    // 3. Add DOB label as a placeholder in the date input
    var studentDOB = $("#StudentDOB").val();
    if (!studentDOB) {
      alert("Date of Birth is compulsory.");
      return false;
    }

    // 4. Validate Roll Number (9 digits)
    var rollNumber = $("#rollNumber").val();
    if (!/^\d{9}$/.test(rollNumber)) {
      alert("Roll number must be 9 digits.");
      return false;
    }

    // 5. Convert Division to uppercase
    var studentDivision = $("#StudentDivision").val().toUpperCase();
```

```
if (!studentDivision) {
  alert("Division is compulsory.");
  return false;
}

// 6. Validate Email
var email = $("#email").val();
if (!/^\S+@\S+\.\S+$/.test(email)) {
  alert("Invalid email address.");
  return false;
}

// 7. Validate Phone Number (10 digits)
var phone = $("#phone").val();
if (!/^\d{10}$/.test(phone)) {
  alert("Phone number must be 10 digits.");
  return false;
}

// 8. Submit the form using AJAX
var formData = new FormData();
formData.append("StudentID", studentID);
formData.append("rollNumber", rollNumber);
formData.append("StudentName", studentName);
formData.append("StudentDOB", studentDOB);
formData.append("StudentClass", $("#StudentClass").val().toUpperCase());
formData.append("StudentDivision", studentDivision);
formData.append("email", email);
formData.append("phone", phone);

$.ajax({
  type: "POST",
  url: "/submit",
  data: formData,
  contentType: false,
  processData: false,
  success: function (response) {
    alert(response.message);
    document.getElementById("myForm").reset();
  },
          error: function (xhr, status, error) {
            alert(JSON.parse(xhr.responseText).message);
          },
        });

        return false;
      }

      function goBack() {
        window.history.back();
      }
    </script>
  </body>
</html>
```

ExportRecords.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta
      name="viewport"
      content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-
scalable=no"
    />
    <title>Export Records</title>
    <style>
      body {
        font-family: "Arial", sans-serif;
        margin: 0;
        padding: 0;
        background-image: linear-gradient(
          25deg,
          #cf4633,
          #be5368,
          #a35f9c,
          #706bd2
        );
  display: flex;
  align-items: flex-start;
  justify-content: center;
  min-height: 100vh;
}

.export-container,
.export-container2,
.export-container3 {
  text-align: center;
  padding: 20px;
  background-color: #fff;
  box-shadow: 0 0 30px rgba(0, 0, 0, 0.7);
  border-radius: 50px;
  width: 80%;
  max-width: 400px;
  margin: 35px;
  margin-top: 7%;
}

h2 {
  color: black;
}

label {
  font-weight: bold;
}

input[type="text"],
input[type="date"] {
  width: 100%;
  padding: 10px;
  margin: 10px 0;
  font-weight: bold;
  font-size: 14px;
  box-sizing: border-box;
  border: 1px solid #ccc;
  border-radius: 10px;
}
```

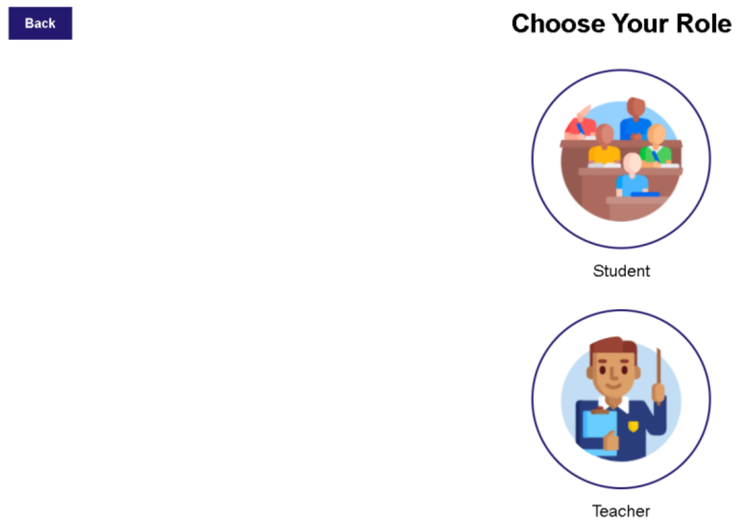
```
select {
  width: 100%;
  padding: 10px;
  margin: 10px 0;
  font-weight: bold;
  font-size: 14px;
  box-sizing: border-box;
  border: 1px solid #ccc;
  border-radius: 10px;
}

.export1,
.export2,
.export3 {
  background-color: #4d65cc;
  color: #fff;
  padding: 10px 20px;
  border: none;
  border-radius: 10px;
  cursor: pointer;
  font-size: 16px;
  font-weight: bold;
}

.export1:hover {
  background-color: #1c2dc7;
}

.export2:hover {
  background-color: #1c2dc7;
}

.export3:hover {
  background-color: #1c2dc7;
}
.back-button {
  position: absolute;
  top: 10px;
  font-size: 16px;
  font-weight: bold;
```

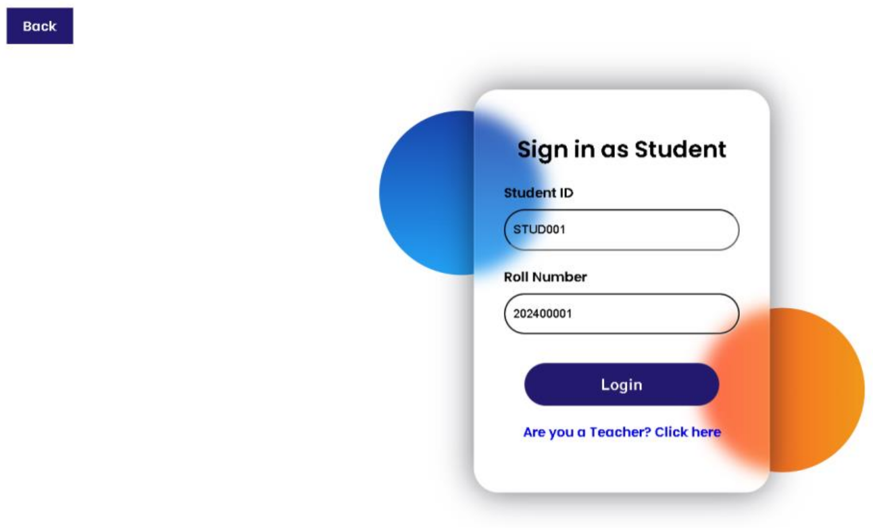
```
    left: 10px;
    background-color: #231a6f;
    color: white;
    border: none;
    padding: 10px 20px;
    cursor: pointer;
    transition: background-color 0.3s;
  }

  .back-button:hover {
    background-color: #0f054c;
  }

  @media screen and (max-width: 600px) {
    body {
      flex-direction: column;
      margin-top: -12%;
      padding: 0;
      background-size: cover;
      background-repeat: no-repeat;
      min-height: 150vh;
    }

    .export-container,
    .export-container2,
    .export-container3 {
      width: 80%;
      margin: 4% 5%;
    }
    .back-button {
      width: 30%;
      height: 6%;
      position: absolute;
      font-size: 16px;
      font-weight: bold;
    }
  }
</style>
<script>
  function validateForm1() {
      var from = document.getElementById("from").value;
      var to = document.getElementById("to").value;

      if (from === "" || to === "") {
        alert("Please enter values for both 'From' and 'To' fields.");
        return false;
      }

      return true;
    }

    function validateForm2() {
      var studentID = document.getElementById("studentID").value;
      var attendanceDate = document.getElementById("attendanceDate").value;

      if (studentID === "" || attendanceDate === "") {
        alert("Please enter values for both 'Student ID' and 'Date' fields.");
        return false;
      }

      return true;
    }

    function validateForm3() {
      var from2 = document.getElementById("from2").value;
      var to2 = document.getElementById("to2").value;
      var dateSelect = document.getElementById("dateSelect").value;

      if (from2 === "" || to2 === "" || dateSelect === "") {
        alert("Please enter values for all fields.");
        return false;
      }

      return true;
    }
    function goBack() {
      window.history.back();
    }
  </script>
</head>
```

```
<body>
  <button class="back-button" onclick="goBack()">Back</button>
  <div class="export-container">
    <h2>Students Records</h2>
    <form
      action="{{ url_for('export_records.export_csv') }}"
      method="POST"
      onsubmit="return validateForm1()"
    >
      <label for="from">From</label>
      <input
        type="text"
        id="from"
        name="from"
        placeholder="Enter Student ID"
      />
      <br />
      <label for="to">To</label>
      <input type="text" id="to" name="to" placeholder="Enter Student ID" />
      <br />
      <button type="submit" class="export1">Export CSV</button>
    </form>
  </div>
  <div class="export-container2">
    <h2>Student Attendance Record</h2>
    <form
      action="{{ url_for('export_records.export_attendance_csv') }}"
      method="POST"
      onsubmit="return validateForm2()"
    >
      <label for="studentID">Student ID:</label>
      <input
        type="text"
        id="studentID"
        name="studentID"
        placeholder="Enter Student ID"
      />
      <br />
      <label for="attendanceDate">Date:</label>
```

```
        <input type="date" id="attendanceDate" name="attendanceDate" />
        <br />
        <button type="submit" class="export2">Export CSV</button>
      </form>
    </div>
    <div class="export-container3">
      <h2>Attendance Data</h2>
      <form
        action="{{ url_for('export_records.export_class_div_csv') }}"
        method="POST"
        onsubmit="return validateForm3()"
      >
        <label for="from2">From</label>
        <input
          type="text"
          id="from2"
          name="from2"
          placeholder="Enter Student ID"
        />
        <br />
        <label for="to2">To</label>
        <input type="text" id="to2" name="to2" placeholder="Enter Student ID" />
        <br />
        <label for="dateSelect">Select Date:</label>
        <input type="date" id="dateSelect" name="dateSelect" />
        <br />
        <button type="submit" class="export3">Export CSV</button>
    </form>
    </div>
</body>
</html>
```

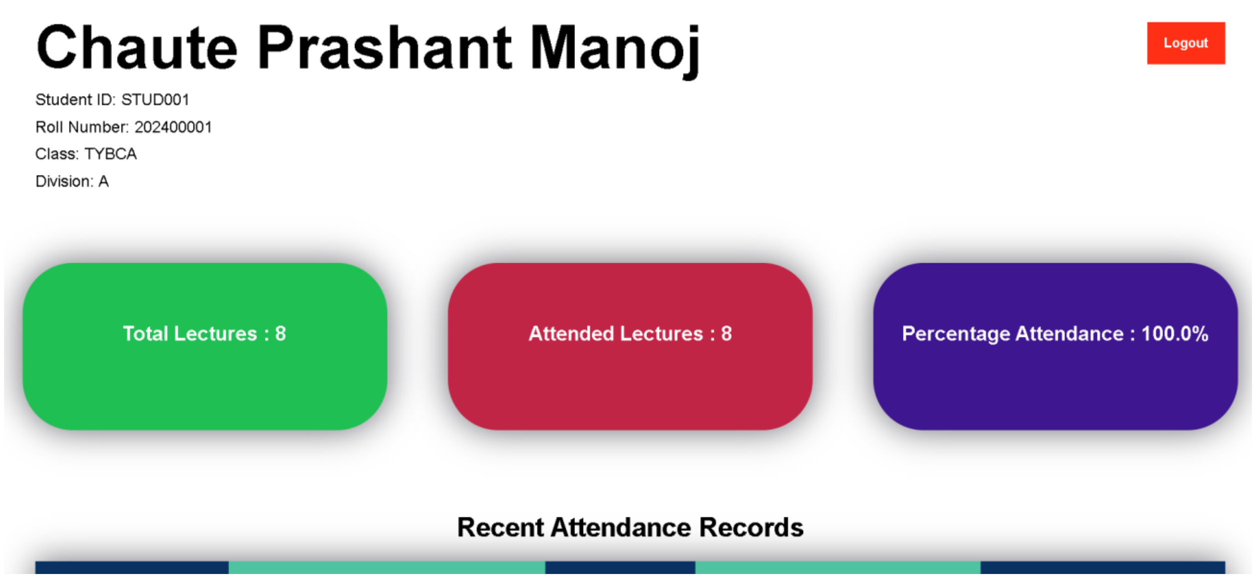
6.5 Output Screenshot





STUD001

202400001



Chaute Prashant Manoj

Logout

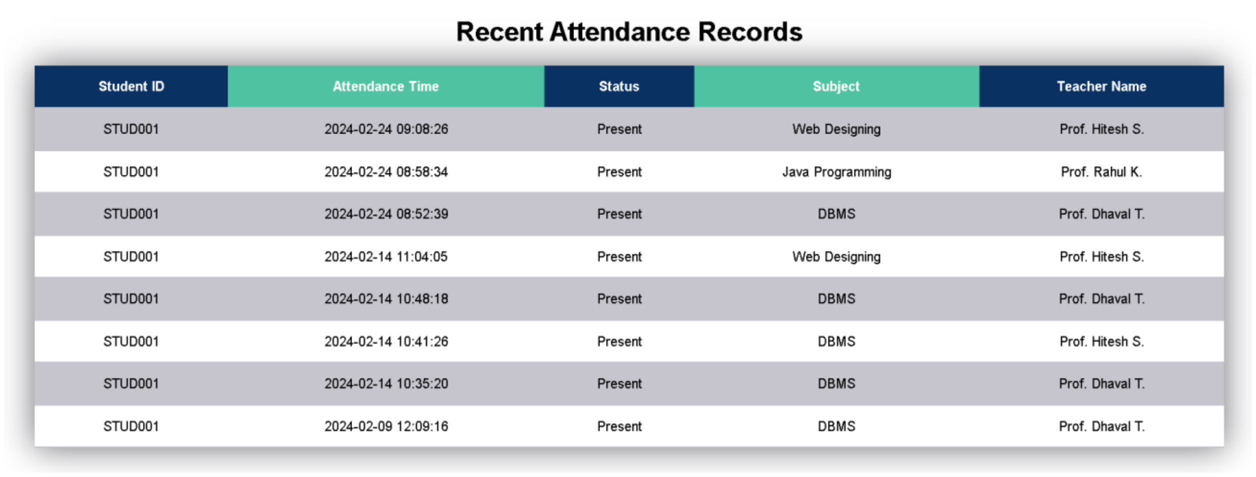
Student ID: STUD001
Roll Number: 202400001
Class: TYBCA
Division: A

Total Lectures : 8

Attended Lectures : 8

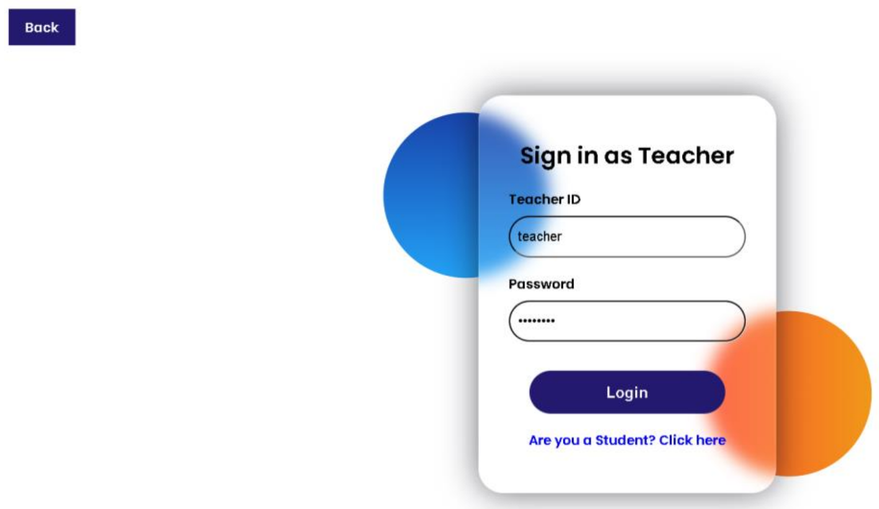
Percentage Attendance : 100.0%

Recent Attendance Records



Recent Attendance Records

Student ID	Attendance Time	Status	Subject	Teacher Name
STUD001	2024-02-24 09:08:26	Present	Web Designing	Prof. Hitesh S.
STUD001	2024-02-24 08:58:34	Present	Java Programming	Prof. Rahul K.
STUD001	2024-02-24 08:52:39	Present	DBMS	Prof. Dhaval T.
STUD001	2024-02-14 11:04:05	Present	Web Designing	Prof. Hitesh S.
STUD001	2024-02-14 10:48:18	Present	DBMS	Prof. Dhaval T.
STUD001	2024-02-14 10:41:26	Present	DBMS	Prof. Hitesh S.
STUD001	2024-02-14 10:35:20	Present	DBMS	Prof. Dhaval T.
STUD001	2024-02-09 12:09:16	Present	DBMS	Prof. Dhaval T.

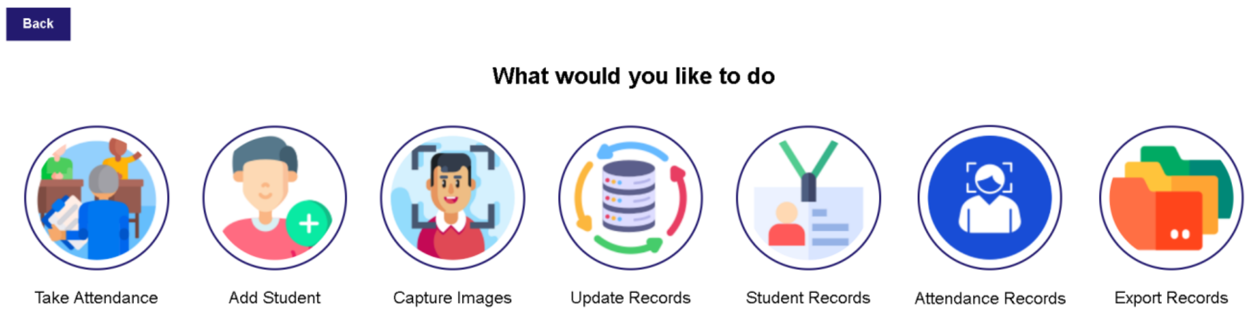


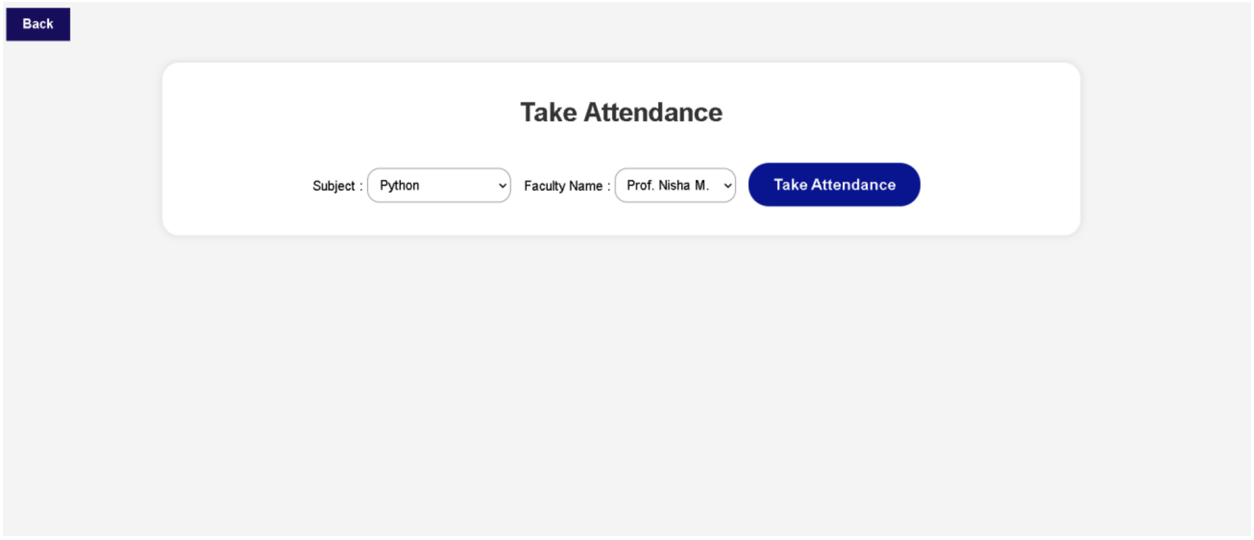
Back

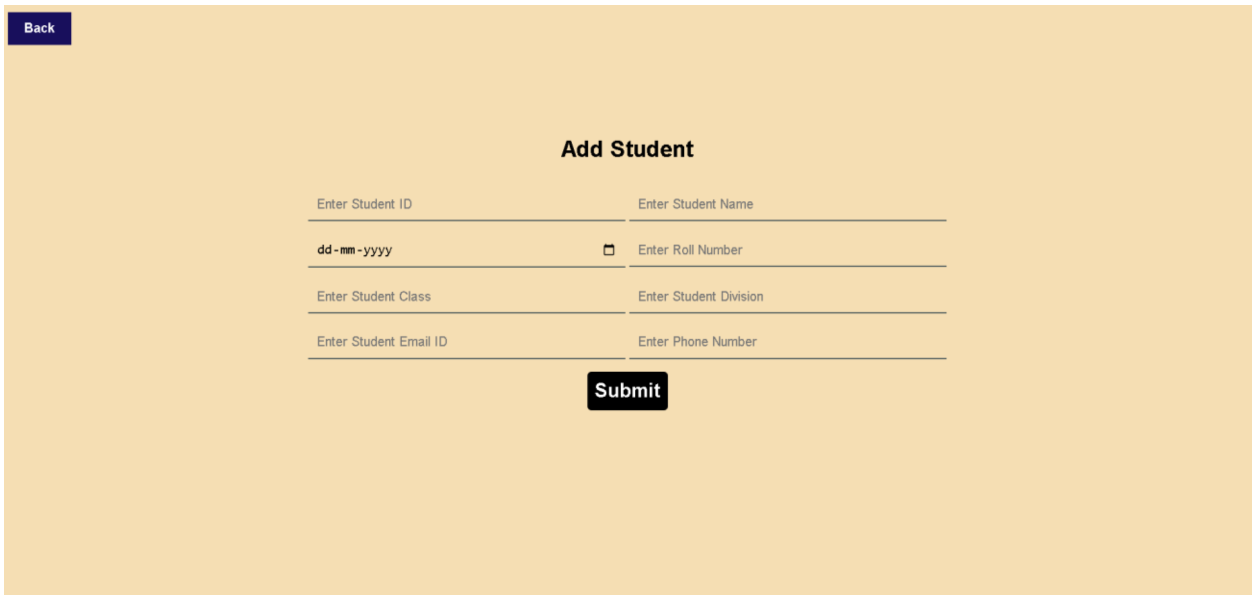
Sign in as Teacher

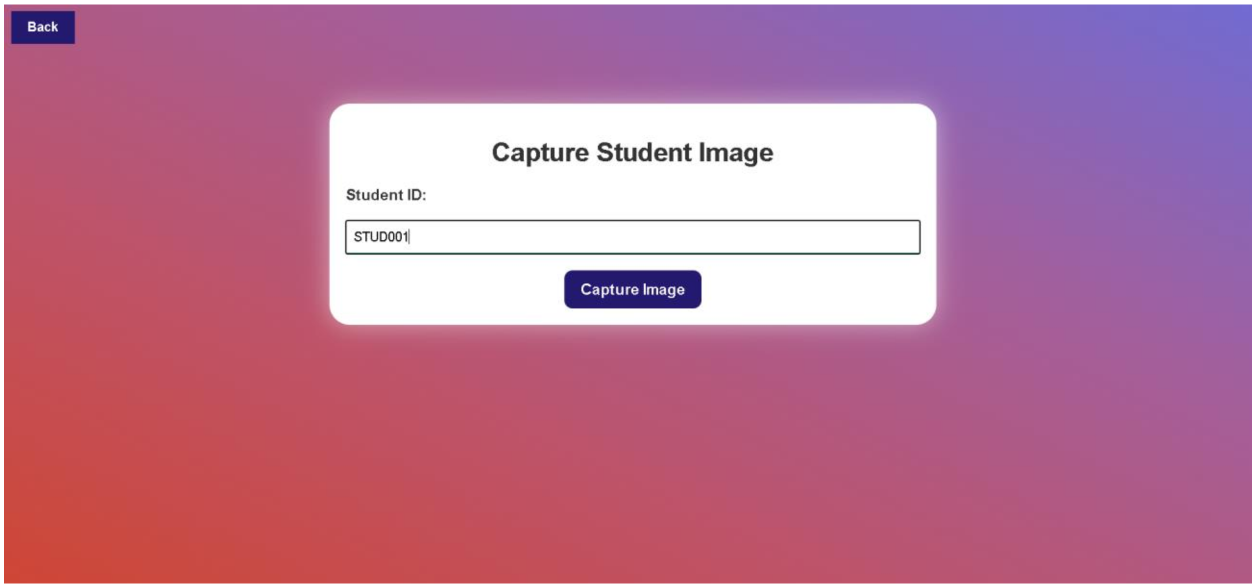
Teacher ID: teacher

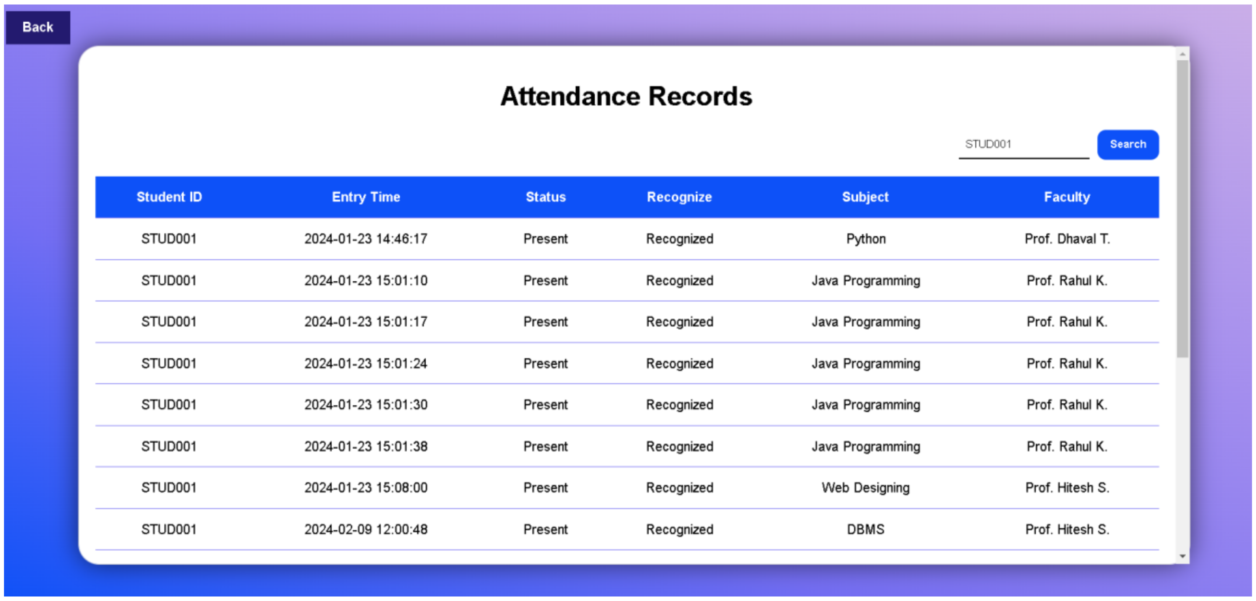
Password: ••••••••


Back
What would you like to do
Take Attendance
Add Student
Capture Images
Update Records
Student Records
Attendance Records
Export Records


Back
Take Attendance
Subject : Python
Faculty Name : Prof. Nisha M.
Take Attendance


Back
Add Student
Enter Student ID
Enter Student Name
dd-mm-yyyy
Enter Roll Number
Enter Student Class
Enter Student Division
Enter Student Email ID
Enter Phone Number
Submit


Back
Capture Student Image
Student ID:
STUD001
Capture Image

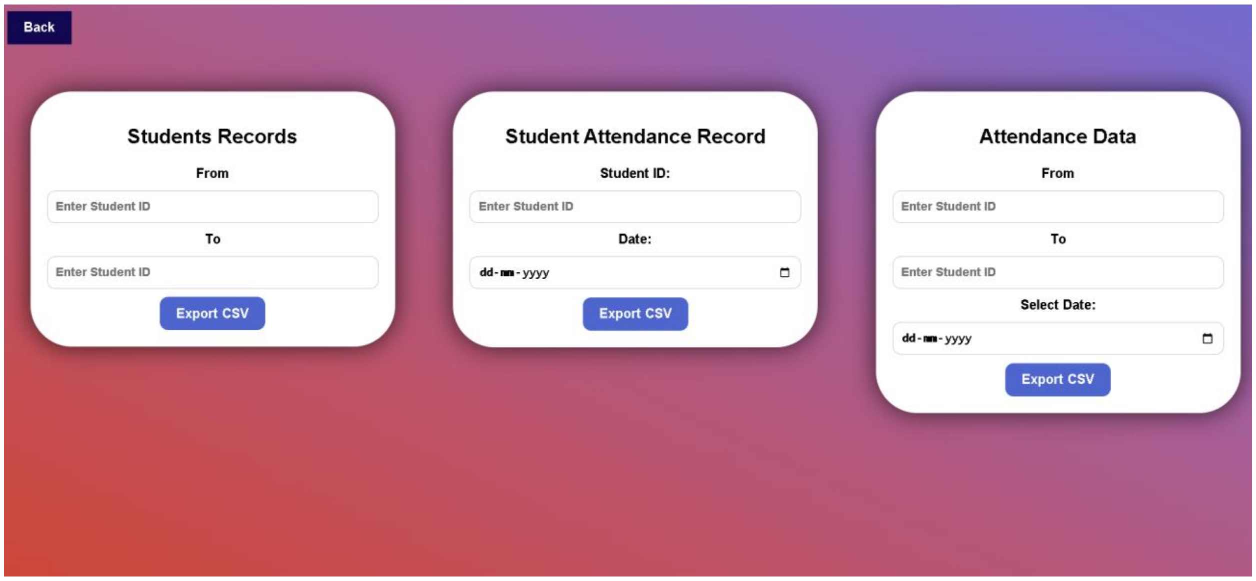


Back

Attendance Records

STUD001 Search

Student ID	Entry Time	Status	Recognize	Subject	Faculty
STUD001	2024-01-23 14:46:17	Present	Recognized	Python	Prof. Dhaval T.
STUD001	2024-01-23 15:01:10	Present	Recognized	Java Programming	Prof. Rahul K.
STUD001	2024-01-23 15:01:17	Present	Recognized	Java Programming	Prof. Rahul K.
STUD001	2024-01-23 15:01:24	Present	Recognized	Java Programming	Prof. Rahul K.
STUD001	2024-01-23 15:01:30	Present	Recognized	Java Programming	Prof. Rahul K.
STUD001	2024-01-23 15:01:38	Present	Recognized	Java Programming	Prof. Rahul K.
STUD001	2024-01-23 15:08:00	Present	Recognized	Web Designing	Prof. Hitesh S.
STUD001	2024-02-09 12:00:48	Present	Recognized	DBMS	Prof. Hitesh S.



Back

Students Records

From: Enter Student ID

To: Enter Student ID

Export CSV

Student Attendance Record

Student ID: Enter Student ID

Date: dd-mm-yyyy

Export CSV

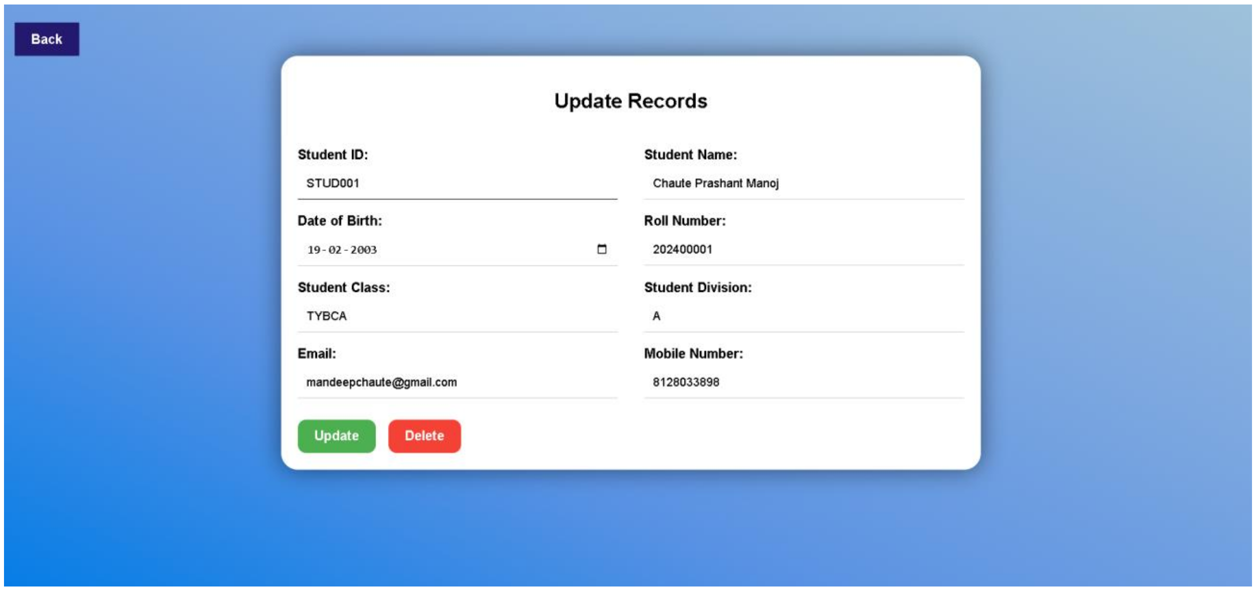
Attendance Data

From: Enter Student ID

To: Enter Student ID

Select Date: dd-mm-yyyy

Export CSV



Update Records

Student ID: STUD001	Student Name: Chaute Prashant Manoj
Date of Birth: 19-02-2003	Roll Number: 202400001
Student Class: TYBCA	Student Division: A
Email: mandeepchaute@gmail.com	Mobile Number: 8128033898

7. Software Testing

Testing Experience with SnapAttendance

During the development of SnapAttendance, rigorous testing was conducted to ensure the reliability, functionality, and performance of the application. The testing process involved various functionalities and modules, with a focus on capturing student images, which posed a significant challenge initially but was successfully addressed through persistent efforts.

1. Tested Functionality - Capture Images

- Capturing student images was a crucial aspect of SnapAttendance. Although challenging initially, thorough testing and refinement led to the successful implementation of this feature.

2. Testing Environment

- The testing environment included a laptop with a webcam, various operating systems, browsers, and databases. Additionally, mobile devices were utilized to preview the application in different contexts.

3. Encountered Bugs and Errors

- Throughout the development phase, numerous bugs and errors were encountered. One particularly complex issue involved the installation of Python libraries using terminal commands, which required extensive troubleshooting and problem-solving.

4. Testing Approach

- All modules of the project were tested manually, with a systematic step-by-step approach to ensure thorough coverage of functionalities and features.

5. Performance Issues

- Performance issues, particularly related to webcam functionality during attendance sessions, were identified and addressed to enhance the application's responsiveness and reliability.

6. Security Concerns

- No significant security concerns or vulnerabilities were identified during testing, ensuring the application's robustness and data integrity.

7. Testing Tools and Techniques

- A combination of manual testing and automated testing techniques was employed, with different modules requiring varying testing approaches based on their complexity and functionality.

8. Result Tracking

- Testing results were continuously monitored and tracked by running the application in various environments, enabling real-time feedback and iterative improvements.

9. Collaborative Testing

- Collaboration with friends and colleagues facilitated problem-solving and logic development to address complex errors and challenges encountered during testing.

10. Overall Conclusion

- Despite the challenges faced during testing, dedicated effort and perseverance led to the successful development of SnapAttendance. The comprehensive testing process ensured the application's quality, reliability, and readiness for deployment, reflecting the culmination of hard work and determination.

8. Limitations and Future Scope of Enhancements

Despite its success, SnapAttendance has certain limitations that warrant consideration. Additionally, there are opportunities for future enhancements to further improve the functionality, usability, and performance of the application.

Limitations

1. Hardware Dependence

SnapAttendance relies on hardware components such as webcams for capturing student images, which may limit its accessibility in environments lacking compatible devices.

2. Performance Issues

Performance issues, particularly during webcam usage and heavy load conditions, may affect the responsiveness and reliability of SnapAttendance, especially in large-scale deployment scenarios.

3. Security Concerns

While efforts have been made to ensure data security, SnapAttendance may still be vulnerable to security threats such as unauthorized access or data breaches, necessitating ongoing vigilance and updates to mitigate risks.

4. Mobile Compatibility

SnapAttendance may not be fully optimized for mobile devices, potentially limiting its usability and functionality on smaller screens or in mobile environments.

Future Scope of Enhancements

1. Enhanced Performance Optimization

Implementing performance optimization techniques, such as caching mechanisms and server-side optimizations, can enhance SnapAttendance's responsiveness and scalability, ensuring seamless performance under varying load conditions.

2. Mobile Application Development

Developing dedicated mobile applications for SnapAttendance can enhance its accessibility and usability on mobile devices, providing students and teachers with a convenient platform for attendance management.

3. Biometric Authentication

Integrating biometric authentication mechanisms, such as fingerprint or facial recognition, can enhance the security and user experience of SnapAttendance, ensuring secure access and authentication for users.

4. Data Analytics and Insights

Implementing data analytics tools and algorithms can provide valuable insights into attendance patterns, trends, and anomalies, enabling administrators to make informed decisions and interventions to improve attendance management processes.

5. Integration with Learning Management Systems (LMS)

Integrating SnapAttendance with existing Learning Management Systems (LMS) can streamline attendance management processes for educational institutions, enabling seamless data exchange and synchronization between platforms.

6. Continuous Security Updates

Regular security audits and updates are essential to address evolving security threats and vulnerabilities, ensuring the ongoing protection of sensitive student data and maintaining compliance with data privacy regulations.

7. User Interface (UI) Enhancements

Enhancing the user interface with intuitive design elements and user-friendly features can improve the overall user experience of SnapAttendance, making it more accessible and engaging for students and teachers alike.

9. References

1. Face Recognition Library

- Website: face-recognition · PyPI

2. Flask Documentation

- Website: Flask

3. OpenCV Documentation

- Website: OpenCV

4. SQLite Documentation

- Website: SQLite

5. HTML, CSS, JavaScript Tutorials

- Website: W3Schools

6. Python Documentation

- Website: Python Docs

7. Youtube

- Website : YouTube

8. Perplexity AI

- Website : Perplexity