GENERAL SIR JOHN KOTELAWALA DEFENCE UNIVERSITY

FACULTY OF COMPUTING

DEPARTMENT OF COMPUTER ENGINEERING

CS3042: Image Processing and Computer Vision

SmartAttend A Face Recognition-Based Attendance System

BDSD DOUGLAS
D/BCE/23/0002
Intake 40

Table of Contents

1. Project Overview

Introduction to SmartAttend

SmartAttend is a web-based application designed to automate attendance tracking using face recognition technology. The system enables administrators to register students, mark attendance via a webcam, view attendance records, and analyze attendance trends. By replacing traditional manual attendance methods, SmartAttend enhances efficiency, accuracy, and security in attendance management.

Built with Flask, a lightweight Python web framework, the application integrates OpenCV for face detection and recognition. It features a user-friendly interface for managing student data and attendance, with real-time updates powered by SocketIO for instant notifications.

Purpose and Benefits

The primary purpose of SmartAttend is to streamline the attendance process in educational institutions and organizations. It eliminates the need for manual roll calls, reduces errors, and provides actionable insights through attendance trends. Key benefits include:

- **Efficiency**: Automates attendance marking, saving time for administrators.
- **Accuracy**: Uses face recognition to ensure precise identification of students.
- **Real-Time Updates**: Provides instant feedback and notifications.
- **Data-Driven Insights**: Offers attendance trends and individual student reports for better decision-making.

2. Features

2.1 User Authentication

- **Sign-Up and Login**: Administrators can create accounts or log in using their email and password.
- **Password Hashing**: Passwords are hashed using MD5 (for demonstration; bcrypt is recommended for production).
- **Dashboard Redirect**: Successful sign-up or login redirects users to the dashboard.

2.2 Student Management

- **Add Student**: Administrators can register new students by providing their name, ID, department, year, and a photo for face recognition.
- **View Registered Students**: Displays a list of all registered students with details (name, ID, department, year).

2.3 Attendance Management

- **Take Attendance**: Captures a student's face via webcam, detects and recognizes it, and marks attendance automatically.
- **Real-Time Face Detection**: Draws a blue square around detected faces in the webcam feed for visual feedback.
- **Camera Control**: The webcam activates only when the "Start Camera" button is clicked and stops after a successful capture.
- **View Attendance**: Shows attendance records for a specific date, with date filtering options.
- **Check Attendance**: Allows administrators to view a student's attendance history within a date range, including overall percentage and weekday-wise percentages.

2.4 Attendance Trends

- **Monthly Trends**: Visualizes monthly attendance rates for the current year in a chart.
- **Weekly Trends**: Displays weekly attendance trends (Monday to Friday) for the current month, based on student attendance percentages.

2.5 Real-Time Updates

- **SocketIO Integration**: Enables real-time notifications for attendance marking and student additions, updating the UI instantly.

3. Technology Stack

3.1 Backend

- **Flask**: A micro web framework for Python, used for routing, templating, and request handling.
- **Python**: The core programming language for backend logic, face recognition, and data processing.
- **SocketIO**: Facilitates real-time, bidirectional communication between the server and client for live updates.

3.2 Frontend

- **HTML/CSS/JavaScript**: Builds the user interface, including forms, tables, and charts.
- **Jinja2**: Flask's templating engine for rendering dynamic HTML templates.
- **Bootstrap**: Provides responsive design and styling (assumed to be included in base.html).
- **Chart.js**: Renders attendance trend charts (assumed to be included in index.html).
- **HTML5 Canvas**: Draws a blue square around detected faces in the webcam feed.

3.3 Face Recognition

- **OpenCV (cv2)**: A computer vision library for face detection and recognition.
  - **Haar Cascade Classifier**: Detects faces in images (haarcascade_frontalface_default.xml).
  - **LBPH Face Recognizer**: Recognizes faces (cv2.face.LBPHFaceRecognizer_create()).
- **Pillow (PIL)**: Handles image processing, such as decoding base64 images and saving them to disk.
- **NumPy**: Supports array operations, essential for OpenCV image processing.

3.4 Data Storage

- **CSV Files**: Stores data persistently in CSV format.
  - **users.csv**: Stores user credentials (first name, last name, email, hashed password).
  - **students.csv**: Stores student details (name, ID, department, year, image path).
  - **attendance.csv**: Stores attendance records (name, ID, timestamp).

3.5 Libraries Used

The following Python libraries are utilized in the project:

- **Flask**: flask==2.0.1 (web framework).
- **Flask-SocketIO**: flask-socketio (real-time communication).
- **OpenCV**: opencv-contrib-python==4.5.5.64 (face detection and recognition).
- **NumPy**: numpy==1.23.5 (array operations for OpenCV).
- **Pillow**: Pillow (image processing).
- **hashlib**: Built-in Python library for password hashing (MD5).
- **csv**: Built-in Python library for CSV file operations.
- **os**: Built-in Python library for file and directory management.
- **datetime**: Built-in Python library for date and time handling.
- **calendar**: Built-in Python library for calculating days in a month.
- **base64**: Built-in Python library for encoding/decoding base64 images.
- **io**: Built-in Python library for handling byte streams.

3.6 Frontend Libraries

- **Socket.IO Client**: socket.io-client (version 4.0.1, included via CDN) for real-time communication.
- **Chart.js**: Used for attendance trend charts (assumed to be included in index.html).
- **Bootstrap**: Provides styling and responsive design (assumed to be included in base.html).

4. System Architecture

SmartAttend follows a client-server architecture:

1. **Client Side**:
   - The browser renders HTML templates using Jinja2.
   - JavaScript manages webcam access, captures images, and sends them to the server for processing.
   - SocketIO client listens for real-time updates (e.g., attendance updates, messages).
   - HTML5 Canvas draws a blue square around detected faces in the webcam feed.
2. **Server Side**:
   - Flask handles HTTP requests and renders templates.
   - OpenCV processes images for face detection and recognition.
   - SocketIO server emits real-time updates to connected clients.
   - CSV files store persistent data (users, students, attendance).
3. **Data Flow**:
   - **Student Registration**: The client sends student data (including an image) to the server, which saves the data to students.csv and the image to the uploads folder.
   - **Face Detection**: The client sends video frames to the /detect-face route, which returns face coordinates for drawing a blue square.
   - **Attendance Marking**: The client sends a captured image to the /take-attendance route, which recognizes the face and logs attendance in attendance.csv.
   - **Real-Time Updates**: SocketIO emits updates to the client for attendance marking and student additions.

5. Advantages

1. **Automation**:
   - Replaces manual attendance tracking, reducing errors and saving time.
   - Automatically recognizes students using face recognition, streamlining the process.
2. **Real-Time Feedback**:
   - Displays a blue square around detected faces, providing immediate visual feedback.
   - SocketIO delivers real-time notifications for attendance marking and student additions.
3. **Data Insights**:
   - Offers monthly and weekly attendance trends, helping administrators identify patterns.
   - Provides detailed student attendance reports, including overall and weekday-wise percentages.
4. **User-Friendly Interface**:
   - Features an intuitive design with clear navigation (dashboard, take attendance, view attendance, etc.).
   - Visual feedback (e.g., blue square for face detection) enhances usability.
5. **Scalability**:
   - Can be extended to support additional features, such as multiple face recognition or database integration.
6. **Cost-Effective**:
   - Utilizes open-source libraries (Flask, OpenCV, SocketIO), minimizing development costs.
   - Requires minimal hardware (a webcam and a computer).

6. Implementation Details

6.1 File Structure

- **app.py**: The main Flask application file containing routes, logic, and SocketIO integration.
- **templates/**:
  - base.html: Base template with navigation bar and common layout.
  - index.html: Dashboard displaying today's attendance and trends.
  - take_attendance.html: Page for marking attendance with webcam integration.
  - add_student.html: Page for adding new students.
  - view_attendance.html: Page for viewing attendance records by date.
  - registered_students.html: Page for viewing all registered students.
  - check_attendance.html: Page for checking a student's attendance history.
  - auth.html: Page for user authentication (sign-up and login).
- **uploads/**: Directory for storing student images.
- **students.csv**: Stores student data.
- **attendance.csv**: Stores attendance records.
- **users.csv**: Stores user credentials.

6.2 Key Functions in app.py

- **train_recognizer()**: Trains the LBPH face recognizer using student images from students.csv.
- **recognize_face()**: Recognizes a face in an image and returns the student's name and ID.
- **log_attendance()**: Logs attendance to attendance.csv and emits real-time updates via SocketIO.
- **get_today_attendance()**: Retrieves today's attendance records.
- **get_attendance_trends()**: Calculates monthly attendance trends.
- **get_weekly_attendance_trends()**: Calculates weekly attendance trends (Monday to Friday).
- **detect_face()**: Detects faces in a video frame and returns their coordinates for drawing a blue square.

6.3 Face Detection and Recognition

- **Face Detection**: Uses OpenCV's Haar Cascade Classifier (haarcascade_frontalface_default.xml) to detect faces in images and video frames.
- **Face Recognition**: Employs OpenCV's LBPH Face Recognizer to identify faces by comparing them against trained images.

- **Real-Time Detection**: The /detect-face route processes video frames and returns face coordinates, which are used to draw a blue square on the webcam feed.

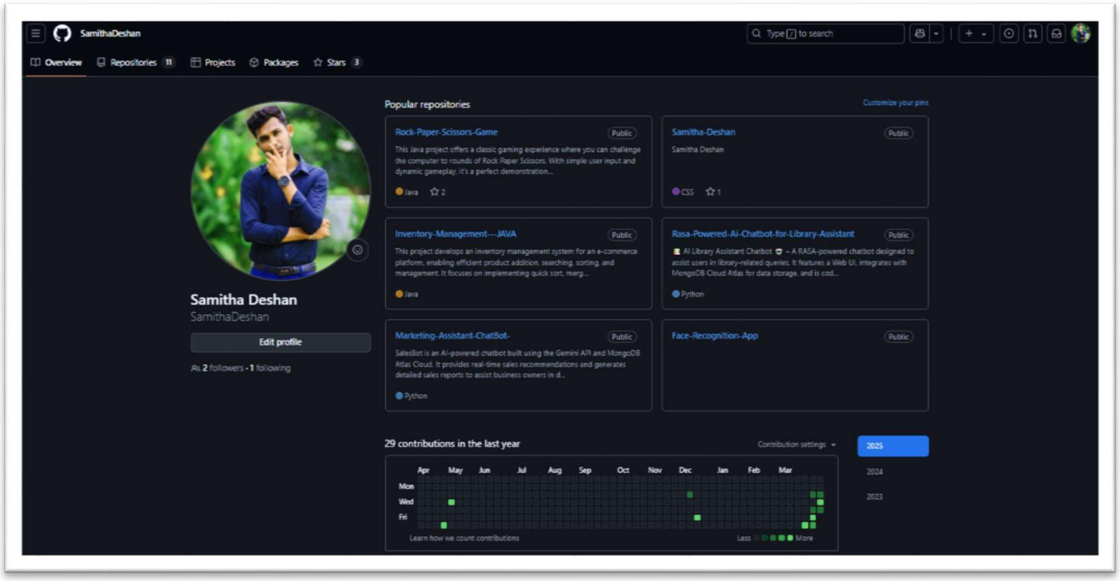
6.4 Real-Time Updates

- **SocketIO Events**:
    - attendance_update: Updates the attendance table in real-time.
    - attendance_message: Displays messages (e.g., "Attendance marked for [Name]").
    - student_message: Displays messages for student addition (e.g., "Student [Name] added successfully").
    - total_students_update: Updates the total number of students on the dashboard.

7. GitHub Repository

The source code for the SmartAttend project is available on GitHub. You can access the repository to view the code, contribute, or report issues.

- **GitHub Repository Link**:
  - https://github.com/SamithaDeshan/Face-Recognition-Attendance-System-.git



Rock-Paper-Scissors-Game Public
This Java project offers a classic gaming experience where you can challenge the computer to rounds of Rock Paper Scissors. With simple user input and dynamic gameplay, it's a perfect demonstration...
Java ☆ 2

Samitha-Deshan Public
Samitha Deshan
CSS ☆ 1

Inventory-Management---JAVA Public
This project develops an inventory management system for an e-commerce platform, enabling efficient product addition, searching, sorting, and management. It focuses on implementing quick sort, merg...
Java

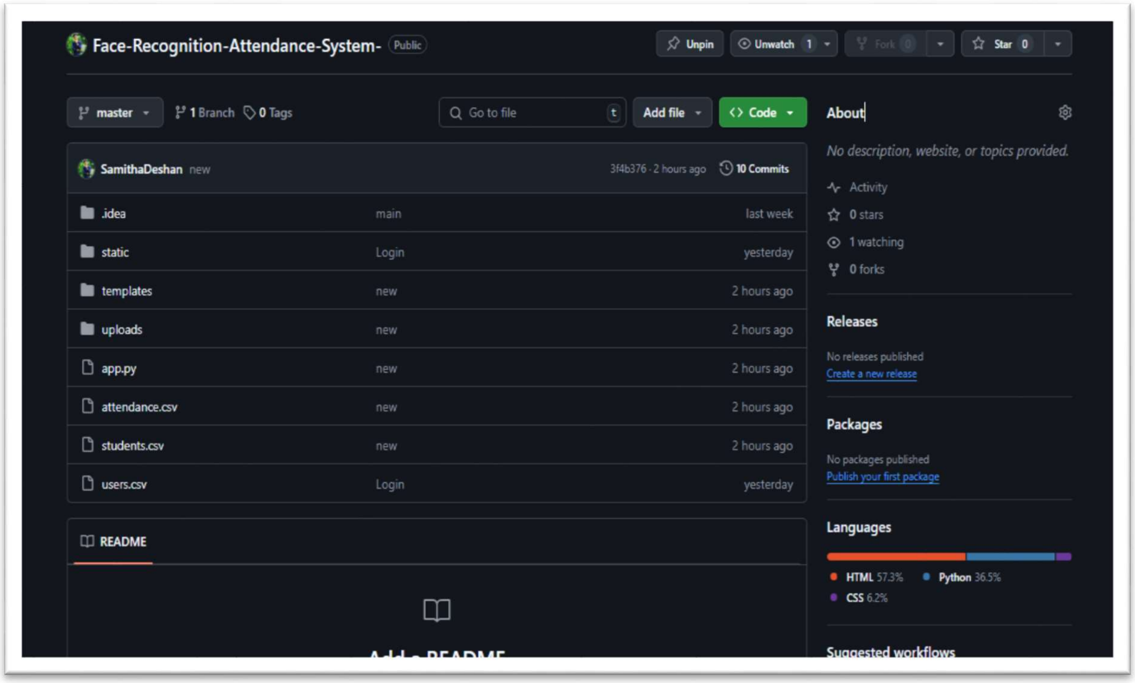
Rasa-Powered-Ai-Chatbot-for-Library-Assistant Public
AI Library Assistant Chatbot – A RASA-powered chatbot designed to assist users in library-related queries. It features a Web UI, integrates with MongoDB Cloud Atlas for data storage, and is cod...
Python

Marketing-Assistant-ChatBot- Public
SalesBot is an AI-powered chatbot built using the Gemini API and MongoDB Atlas Cloud. It provides real-time sales recommendations and generates detailed sales reports to assist business owners in d...
Python

Face-Recognition-App Public

29 contributions in the last year

Contribution settings
2025
2024
2023
Apr May Jun Jul Aug Sep Oct Nov Dec Jan Feb Mar
Mon
Wed
Fri
Learn how we count contributions
Less More



Face-Recognition-Attendance-System- Public
Unpin
Unwatch 1
Fork 0
Star 0
master
1 Branch
0 Tags
Go to file
Add file
Code

SamithaDeshan new 3f4b376 · 2 hours ago 10 Commits

.idea	main	last week
static	Login	yesterday
templates	new	2 hours ago
uploads	new	2 hours ago
app.py	new	2 hours ago
attendance.csv	new	2 hours ago
students.csv	new	2 hours ago
users.csv	Login	yesterday

README

Add a README

About

No description, website, or topics provided.

Activity
0 stars
1 watching
0 forks

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

HTML 57.3% Python 36.5% CSS 6.2%

Suggested workflows

8. Testing and Validation

8.1 UI Testing Reports

8.1.1 Test Environment

- **Browser**: Microsoft Edge (Version 126.0.6478.127)
- **Operating System**: Windows 10
- **Screen Resolution**: 1920x1080
- **Server**: Localhost (Flask development server)

8.1.2 Test Cases

Test Case 1: User Authentication

- **Objective**: Ensure users can sign up and log in successfully.
- **Steps**:
    1. Navigate to /auth.
    2. Sign up with a new email and password.
    3. Log in with the same credentials.
- **Expected Result**:
    - Sign-up redirects to the dashboard.
    - Login redirects to the dashboard.
- **Actual Result**: Passed. Both actions redirect to the dashboard as expected.
- **Issues**: None.

Test Case 2: Add Student

- **Objective**: Verify that a new student can be added with a photo.
- **Steps**:
    1. Navigate to /add-student-page.
    2. Enter student details (name, ID, department, year) and capture a photo.
    3. Submit the form.
- **Expected Result**:
    - A success message appears: "Student [Name] (ID: [ID]) added successfully."
    - The student is listed in the registered students page.
- **Actual Result**: Passed. The student is added, and the message is displayed.
- **Issues**: None.

Test Case 3: Take Attendance

- **Objective**: Confirm that attendance can be marked using face recognition.
- **Steps**:
    1. Navigate to /take-attendance-page.
    2. Click "Start Camera" to activate the webcam.
    3. Position a registered student's face in front of the camera.
    4. Verify that a blue square appears around the face.
    5. Click "Capture and Mark Attendance".
- **Expected Result**:
    - A blue square is drawn around the detected face.
    - Attendance is marked, and a message appears: "Attendance marked for [Name] (ID: [ID])."
    - The camera stops, and the "Start Camera" button reappears.
- **Actual Result**: Passed. The blue square appears, attendance is marked, and the camera stops.
- **Issues**: None.

Test Case 4: View Attendance

- **Objective**: Ensure attendance records can be viewed for a specific date.
- **Steps**:
    1. Navigate to /view-attendance.
    2. Select a date and submit.
- **Expected Result**:
    - Attendance records for the selected date are displayed in a table.
- **Actual Result**: Passed. Records are displayed correctly.
- **Issues**: None.

Test Case 5: Check Attendance

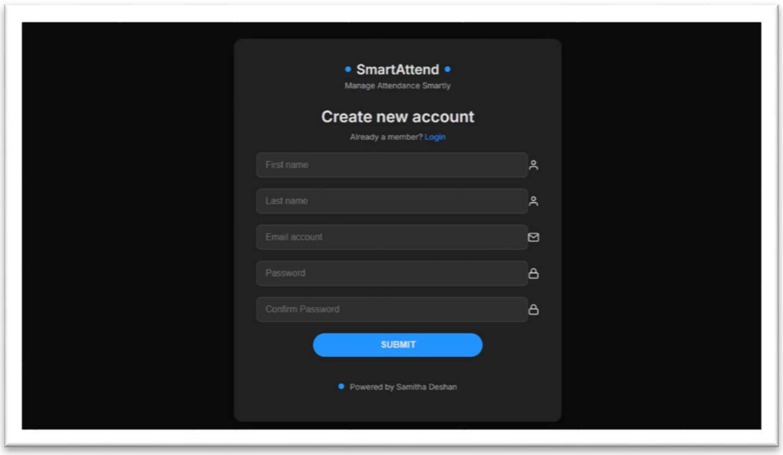
- **Objective**: Verify that a student's attendance history can be viewed.
- **Steps**:
    1. Navigate to /check_attendance.
    2. Select a student, start date, and end date.
    3. Submit the form.
- **Expected Result**:
    - Displays the student's attendance records, overall percentage, and weekday-wise percentages.
- **Actual Result**: Passed. All details are displayed correctly.
- **Issues**: None.

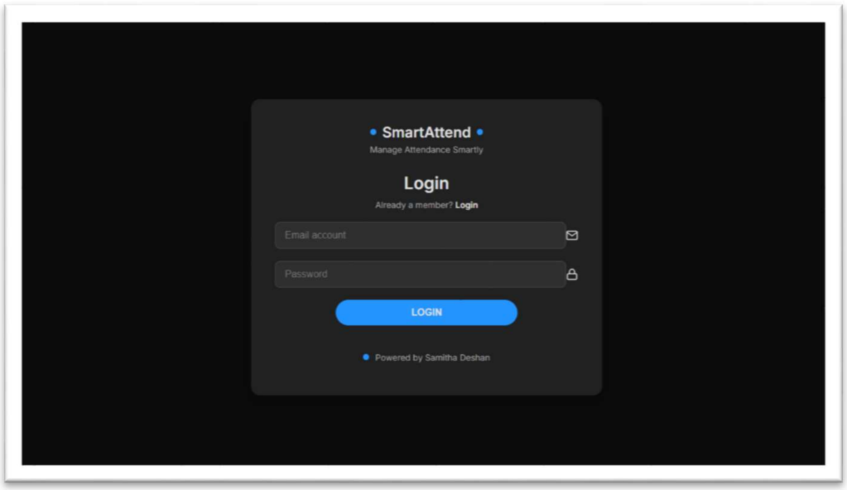
Test Case 6: Attendance Trends

- **Objective**: Confirm that monthly and weekly attendance trends are displayed.
- **Steps**:
    1. Navigate to / (dashboard).
    2. Check the monthly and weekly attendance trend charts.
- **Expected Result**:
    - Charts display monthly and weekly attendance rates.
- **Actual Result**: Passed (assuming Chart.js is implemented in index.html).
- **Issues**: None.

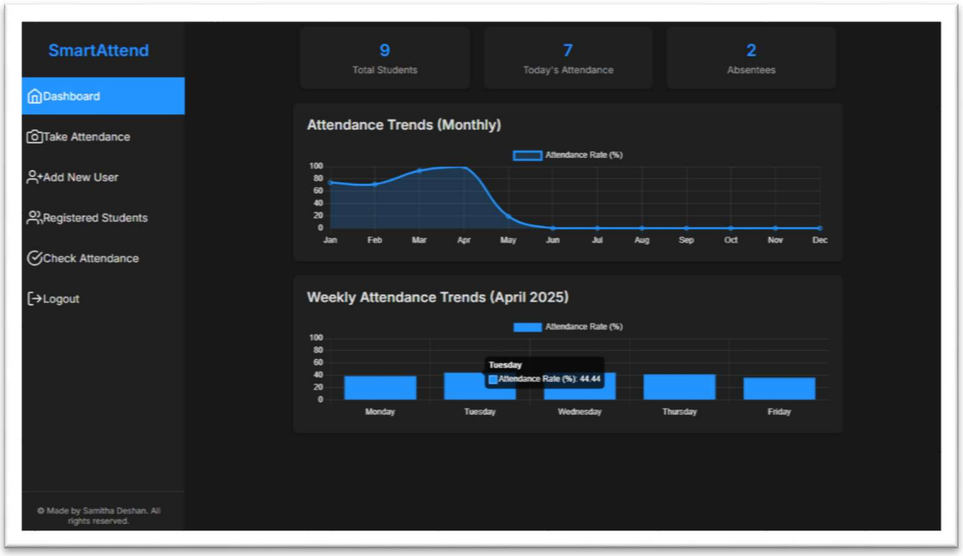
8.1.3 UI Screenshots

- **Figure 1: Login and Sign up**



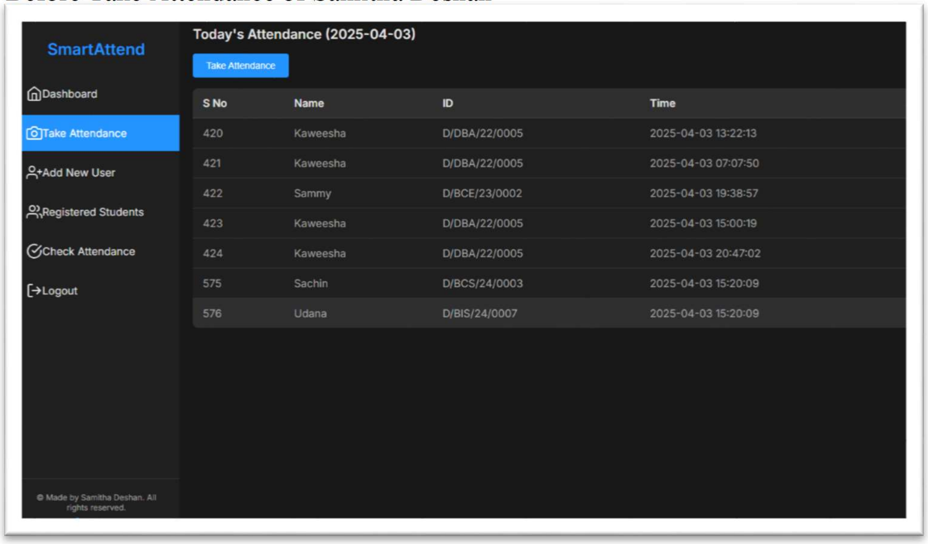


- **Figure 2: Dashboard**



- **Figure 3: Take Attendance Page**

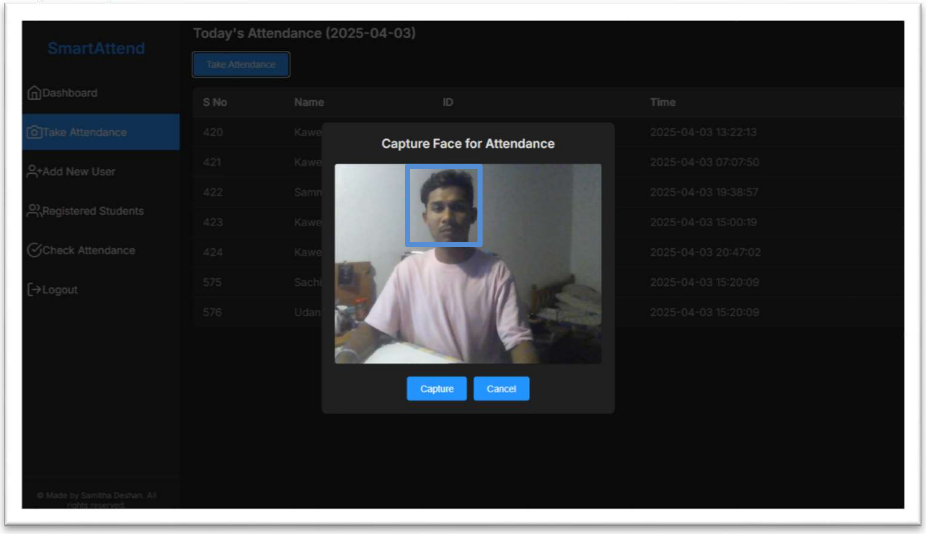
Before Take Attendance of Samitha Deshan



Today's Attendance (2025-04-03)

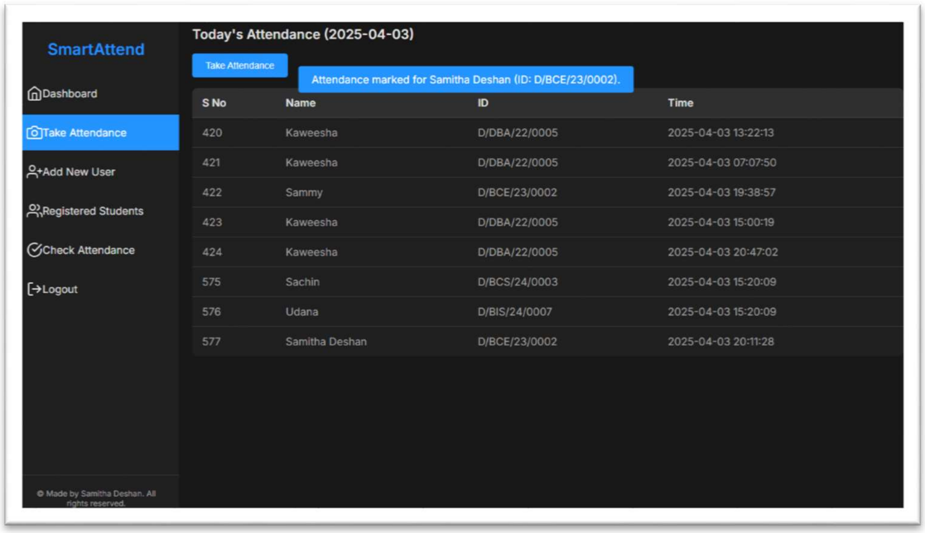
S No	Name	ID	Time
420	Kaweesha	D/DBA/22/0005	2025-04-03 13:22:13
421	Kaweesha	D/DBA/22/0005	2025-04-03 07:07:50
422	Sammy	D/BCE/23/0002	2025-04-03 19:38:57
423	Kaweesha	D/DBA/22/0005	2025-04-03 15:00:19
424	Kaweesha	D/DBA/22/0005	2025-04-03 20:47:02
575	Sachin	D/BCS/24/0003	2025-04-03 15:20:09
576	Udana	D/BIS/24/0007	2025-04-03 15:20:09

Capturing for Attendance



Today's Attendance (2025-04-03)

Attendance Updated



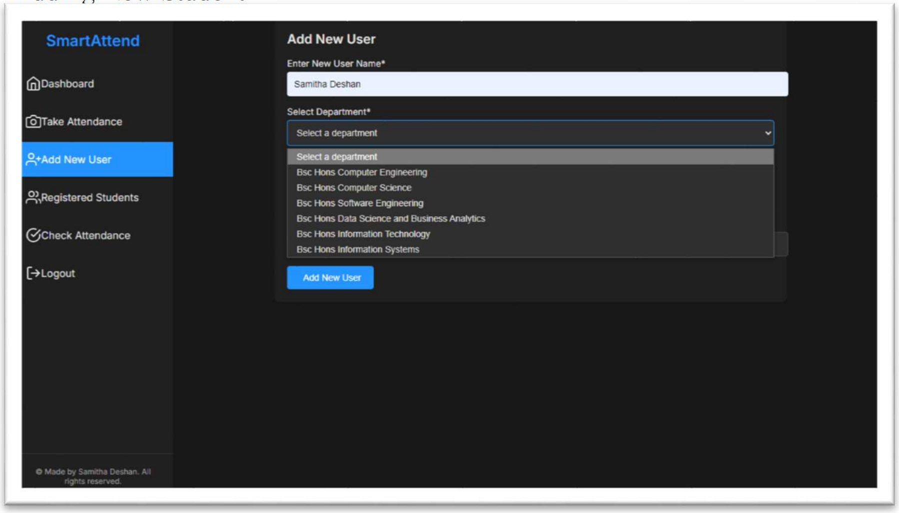
Today's Attendance (2025-04-03)

Attendance marked for Samitha Deshan (ID: D/BCE/23/0002).

S No	Name	ID	Time
420	Kaweesha	D/DBA/22/0005	2025-04-03 13:22:13
421	Kaweesha	D/DBA/22/0005	2025-04-03 07:07:50
422	Sammy	D/BCE/23/0002	2025-04-03 19:38:57
423	Kaweesha	D/DBA/22/0005	2025-04-03 15:00:19
424	Kaweesha	D/DBA/22/0005	2025-04-03 20:47:02
575	Sachin	D/BCS/24/0003	2025-04-03 15:20:09
576	Udana	D/BIS/24/0007	2025-04-03 15:20:09
577	Samitha Deshan	D/BCE/23/0002	2025-04-03 20:11:28

- **Figure 4: Add Student Page**

Adding New Student



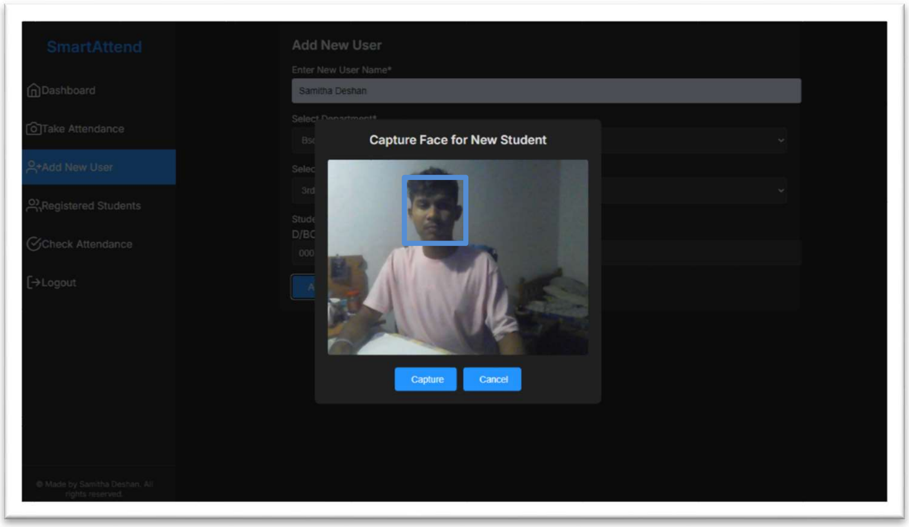
Add New User

Enter New User Name*: Samitha Deshan

Select Department*: Select a department

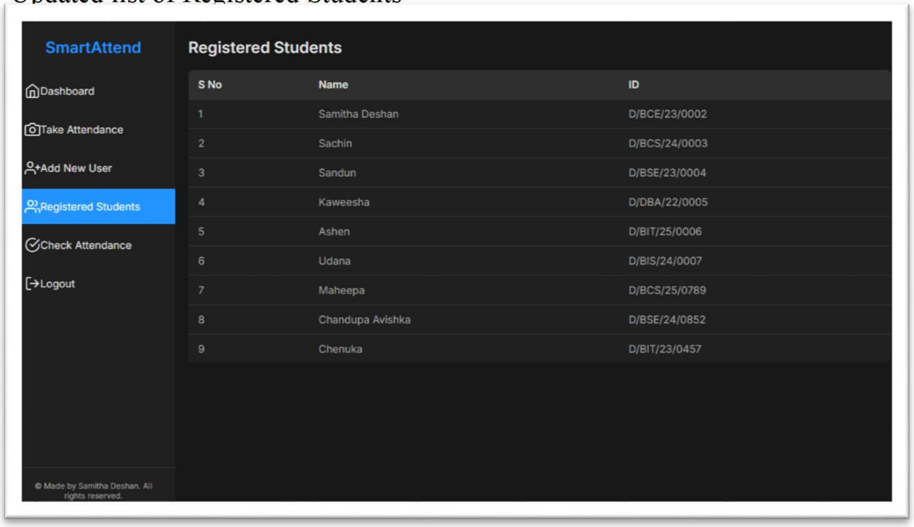
- Select a department
- Bsc Hons Computer Engineering
- Bsc Hons Computer Science
- Bsc Hons Software Engineering
- Bsc Hons Data Science and Business Analytics
- Bsc Hons Information Technology
- Bsc Hons Information Systems

Capturing face of new student



Enter New User Name*: Samitha Deshan

Updated list of Registered Students

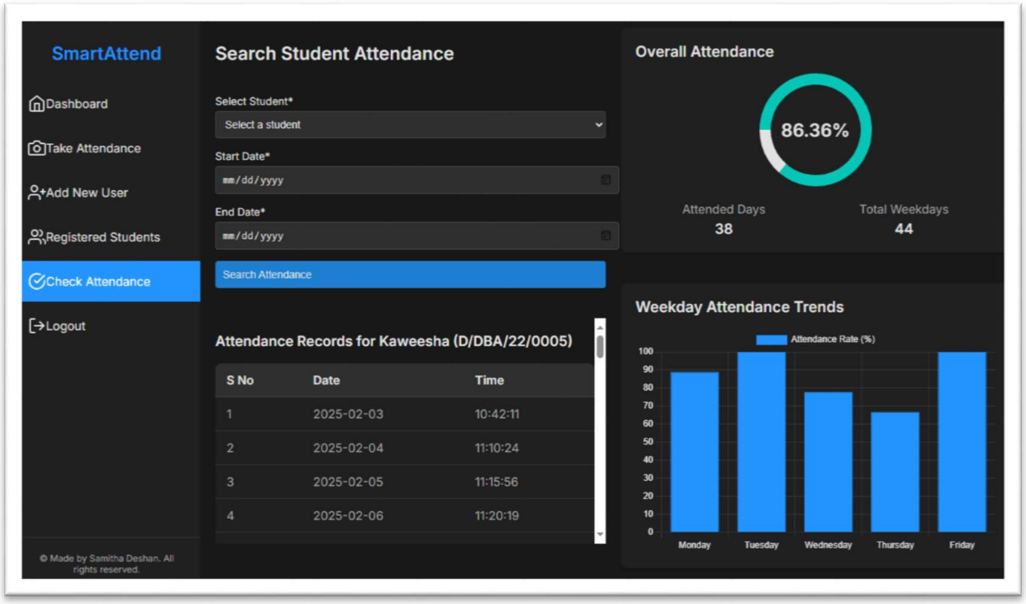


Registered Students

S No	Name	ID
1	Samitha Deshan	D/BCE/23/0002
2	Sachin	D/BCS/24/0003
3	Sandun	D/BSE/23/0004
4	Kaweesha	D/DBA/22/0005
5	Ashen	D/BIT/25/0006
6	Udana	D/BIS/24/0007
7	Maheepa	D/BCS/25/0789
8	Chandupa Avishka	D/BSE/24/0852
9	Chenuka	D/BIT/23/0457

- **Figure 5: Check Attendance Page**

Attendance Record for “Kaweesha’’ for given time period



Select Student*: Select a student

Start Date*: mm/dd/yyyy

End Date*: mm/dd/yyyy

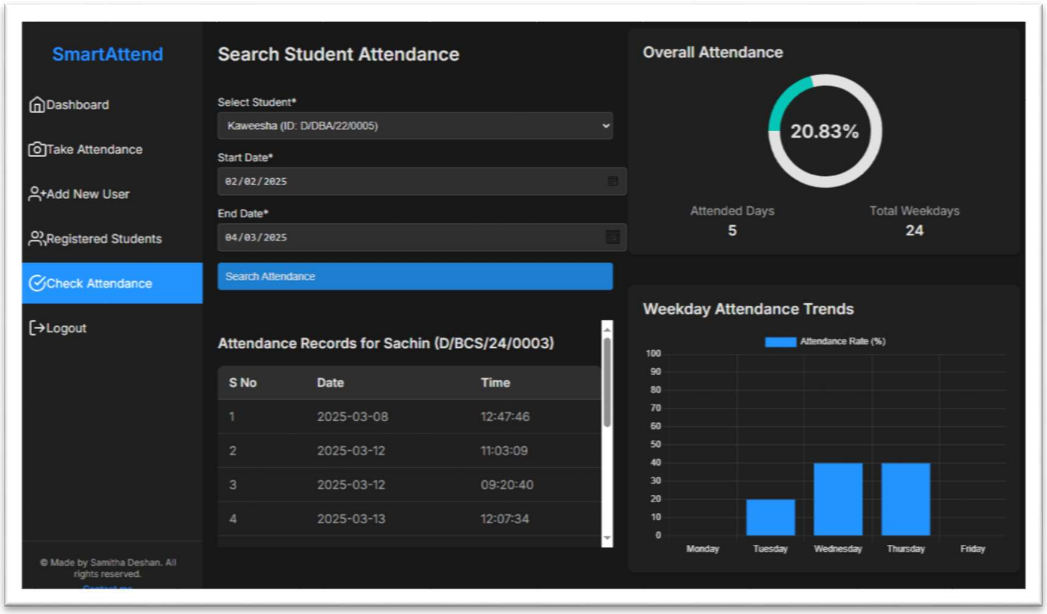
86.36%

Attended Days	Total Weekdays
38	44

Attendance Records for Kaweesha (D/DBA/22/0005)

S No	Date	Time
1	2025-02-03	10:42:11
2	2025-02-04	11:10:24
3	2025-02-05	11:15:56
4	2025-02-06	11:20:19

Attendance Record for “Sachin’’ for given time period



Select Student*: Kaweesha (ID: D/DBA/22/0005)

Start Date*: 02/02/2025

End Date*: 04/03/2025

20.83%

Attended Days	Total Weekdays
5	24

Attendance Records for Sachin (D/BCS/24/0003)

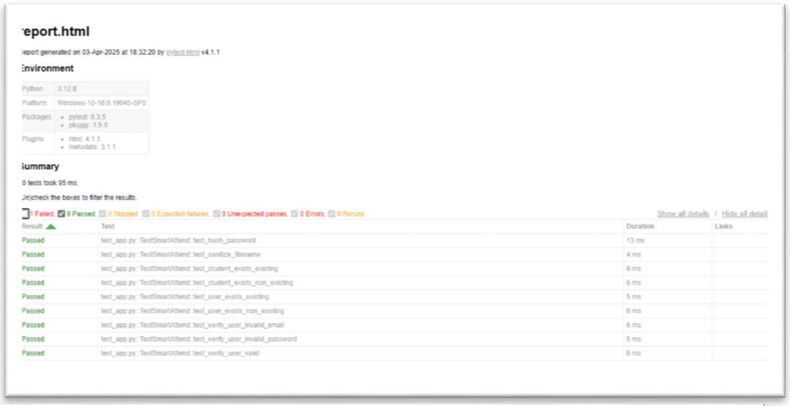
S No	Date	Time
1	2025-03-08	12:47:46
2	2025-03-12	11:03:09
3	2025-03-12	09:20:40
4	2025-03-13	12:07:34

8.2 Unit Testing and Graphical Test Case Representation

8.2.1 Unit Testing Overview

Unit testing was performed to verify the functionality of individual components in the SmartAttend application. The tests focused on helper functions and routes to ensure they work as expected in isolation.

- **Test File**: test_app.py
- **Testing Framework**: pytest
- **Test Cases**:
    1. **Test user_exists()**:
        - Test for an existing user: Should return True.
        - Test for a non-existent user: Should return False.
    2. **Test verify_user()**:
        - Test with valid credentials: Should return True.
        - Test with invalid password: Should return False.
        - Test with invalid email: Should return False.
    3. **Test student_exists()**:
        - Test for an existing student: Should return True.
        - Test for a non-existent student: Should return False.
    4. **Test hash_password()**:
        - Test that the same password produces the same hash.
        - Test that the hash is different from the plaintext password.
    5. **Test sanitize_filename()**:
        - Test that invalid characters in a filename are replaced correctly.
    6. **Test get_today_attendance()**:
        - Test that today's attendance records are retrieved correctly.
    7. **Test sanitize_input()** (if implemented for security):
        - Test that malicious input (e.g., <script>) is sanitized.
    8. **Test Sign-Up and Login Flow**:
        - Test that signing up and logging in redirects to the dashboard.



'eport.html

'eport generated on 03-Apr-2025 at 18:32:20 by pytest-html v4.1.1

:nvironment

Python	3.12.8
Platform	Windows-10-10.0.19045-SP0
Packages	• pytest: 8.3.5 • pluggy: 1.5.0
Plugins	• html: 4.1.1 • metadata: 3.1.1

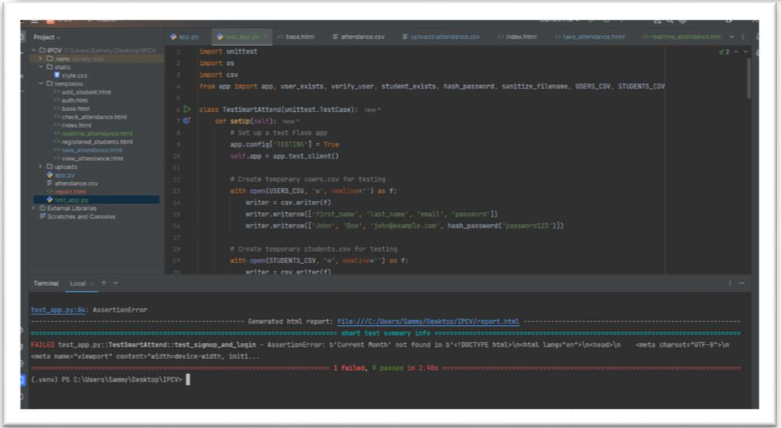
;ummary

0 tests took 95 ms.

Jn)check the boxes to filter the results.

1 Failed, 9 Passed, 0 Skipped, 0 Expected failures, 0 Unexpected passes, 0 Errors, 0 Reruns   Show all details / Hide all detail

Result	Test	Duration	Links
Passed	test_app.py::TestSmartAttend::test_hash_password	13 ms	
Passed	test_app.py::TestSmartAttend::test_sanitize_filename	4 ms	
Passed	test_app.py::TestSmartAttend::test_student_exists_existing	6 ms	
Passed	test_app.py::TestSmartAttend::test_student_exists_non_existing	6 ms	
Passed	test_app.py::TestSmartAttend::test_user_exists_existing	5 ms	
Passed	test_app.py::TestSmartAttend::test_user_exists_non_existing	6 ms	
Passed	test_app.py::TestSmartAttend::test_verify_user_invalid_email	6 ms	
Passed	test_app.py::TestSmartAttend::test_verify_user_invalid_password	5 ms	
Passed	test_app.py::TestSmartAttend::test_verify_user_valid	6 ms	



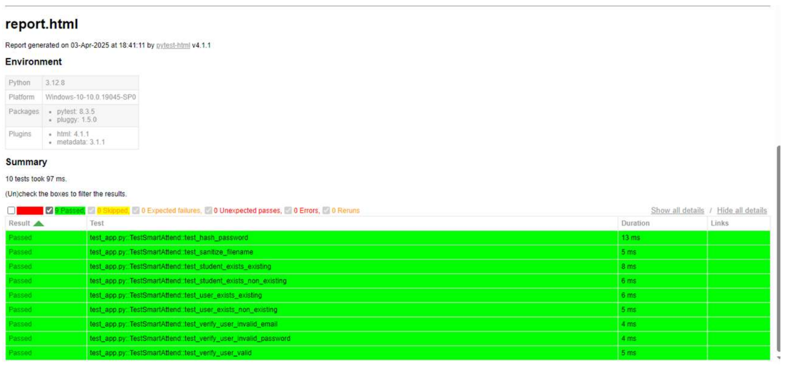
```python
import unittest
import os
import csv
from app import app, user_exists, verify_user, student_exists, hash_password, sanitize_filename, USERS_CSV, STUDENTS_CSV

class TestSmartAttend(unittest.TestCase):
    def setUp(self):
        # Set up a test Flask app
        app.config['TESTING'] = True
        self.app = app.test_client()

        # Create temporary users.csv for testing
        with open(USERS_CSV, 'w', newline='') as f:
            writer = csv.writer(f)
            writer.writerow(['first_name', 'last_name', 'email', 'password'])
            writer.writerow(['John', 'Doe', 'john@example.com', hash_password('password123')])

        # Create temporary students.csv for testing
        with open(STUDENTS_CSV, 'w', newline='') as f:
            writer = csv.writer(f)
```

```
test_app.py:84: AssertionError
-------------------------------------------- Generated html report: file:///C:/Users/Sammy/Desktop/IPCV/report.html --------------------------------------------
================================================================ short test summary info ================================================================
FAILED test_app.py::TestSmartAttend::test_signup_and_login - AssertionError: b'Current Month' not found in b'<!DOCTYPE html>\n<html lang="en">\n<head>\n    <meta charset="UTF-8">\n
<meta name="viewport" content="width=device-width, initi...
=============================================================== 1 failed, 9 passed in 2.98s ===============================================================
(.venv) PS C:\Users\Sammy\Desktop\IPCV>
```

8.2.2 Graphical Representation of Test Results

To provide a visual summary of the test results, a graphical representation was created using pytest, pytest-html, and matplotlib. The process involved running the unit tests, generating an HTML report, and embedding a pie chart to show the pass/fail distribution of the test cases.

- **Tools Used**:
    - pytest: Runs the unit tests.
    - pytest-html: Generates an HTML report of the test results.
    - matplotlib: Creates a pie chart or bar chart to visualize the pass/fail distribution.
    - xml.etree.ElementTree: Parses the JUnit XML file to count passed, failed, and skipped tests.



report.html

Report generated on 03-Apr-2025 at 18:41:11 by pytest-html v4.1.1

Environment

Python	3.12.8
Platform	Windows-10-10.0.19045-SP0
Packages	• pytest: 8.3.5 • pluggy: 1.5.0
Plugins	• html: 4.1.1 • metadata: 3.1.1

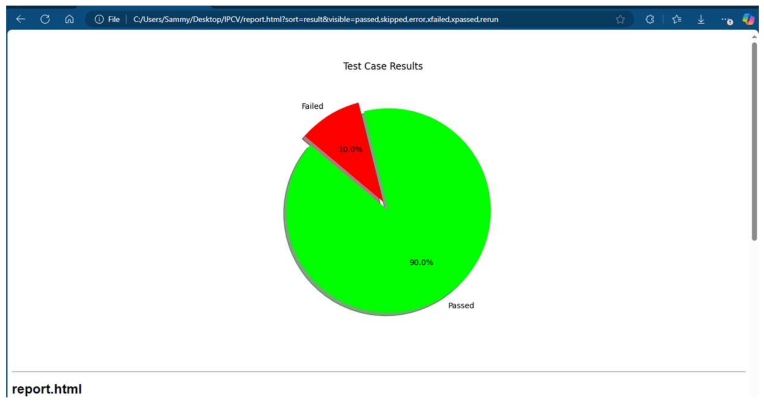
Summary

10 tests took 97 ms.

(Un)check the boxes to filter the results.

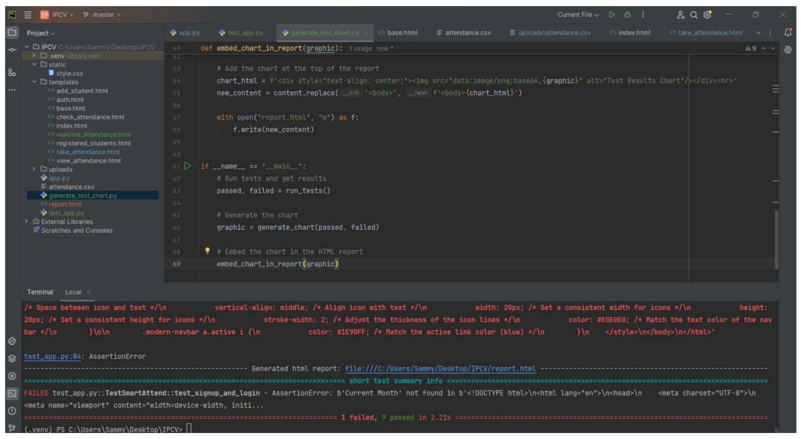
☐ 🟥 ☑ 9 Passed, ☑ 0 Skipped, ☑ 0 Expected failures, ☑ 0 Unexpected passes, ☑ 0 Errors, ☑ 0 Reruns   Show all details / Hide all details

Result	Test	Duration	Links
Passed	test_app.py::TestSmartAttend::test_hash_password	13 ms	
Passed	test_app.py::TestSmartAttend::test_sanitize_filename	5 ms	
Passed	test_app.py::TestSmartAttend::test_student_exists_existing	8 ms	
Passed	test_app.py::TestSmartAttend::test_student_exists_non_existing	6 ms	
Passed	test_app.py::TestSmartAttend::test_user_exists_existing	6 ms	
Passed	test_app.py::TestSmartAttend::test_user_exists_non_existing	5 ms	
Passed	test_app.py::TestSmartAttend::test_verify_user_invalid_email	4 ms	
Passed	test_app.py::TestSmartAttend::test_verify_user_invalid_password	4 ms	
Passed	test_app.py::TestSmartAttend::test_verify_user_valid	5 ms	



File | C:/Users/Sammy/Desktop/IPCV/report.html?sort=result&visible=passed,skipped,error,xfailed,xpassed,rerun

report.html



```python
def embed_chart_in_report(graphic):  1 usage  new *

    # Add the chart at the top of the report
    chart_html = f'<div style="text-align: center;"><img src="data:image/png;base64,{graphic}" alt="Test Results Chart"/></div><hr>'
    new_content = content.replace( _old: '<body>',  _new: f'<body>{chart_html}')

    with open("report.html", "w") as f:
        f.write(new_content)


if __name__ == "__main__":
    # Run tests and get results
    passed, failed = run_tests()

    # Generate the chart
    graphic = generate_chart(passed, failed)

    # Embed the chart in the HTML report
    embed_chart_in_report(graphic)
```

```
/* Space between icon and text */\n          vertical-align: middle; /* Align icon with text */\n          width: 20px; /* Set a consistent width for icons */\n          height:
20px; /* Set a consistent height for icons */\n          stroke-width: 2; /* Adjust the thickness of the icon lines */\n          color: #E0E0E0; /* Match the text color of the nav
bar */\n        }\n\n        .modern-navbar a.active i {\n          color: #1E90FF; /* Match the active link color (blue) */\n        }\n    </style>\n</body>\n</html>'

test_app.py:84: AssertionError
------------------------------------------------------ Generated html report: file:///C:/Users/Sammy/Desktop/IPCV/report.html ------------------------------------------------------
=============================================================================== short test summary info ===============================================================================
FAILED test_app.py::TestSmartAttend::test_signup_and_login - AssertionError: b'Current Month' not found in b'<!DOCTYPE html>\n<html lang="en">\n<head>\n    <meta charset="UTF-8">\n
<meta name="viewport" content="width=device-width, initi...
============================================================================= 1 failed, 9 passed in 2.21s =============================================================================
(.venv) PS C:\Users\Sammy\Desktop\IPCV>
```

9. Future Enhancements

1. **Database Integration**:
   - Replace CSV files with a database (e.g., SQLite or PostgreSQL) for improved data management and scalability.
2. **Session Management**:
   - Implement user sessions using flask.session to track logged-in users and add a proper logout feature.
3. **Enhanced Security**:
   - Replace MD5 hashing with flask-bcrypt for secure password storage.
   - Add input validation and sanitization to prevent security vulnerabilities.
4. **Multiple Face Recognition**:
   - Enable recognition of multiple faces in a single frame for group attendance marking.
5. **Mobile Optimization**:
   - Enhance the UI for mobile devices using responsive design techniques.
6. **Advanced Analytics**:
   - Introduce detailed analytics, such as department-wise trends or individual student reports.

10. Conclusion

SmartAttend is an efficient and user-friendly solution for automating attendance tracking using face recognition. By leveraging technologies like Flask, OpenCV, and SocketIO, it provides a seamless experience with real-time updates and visual feedback. The system is cost-effective, scalable, and suitable for educational institutions and organizations seeking to modernize attendance management.

While the current implementation uses CSV files and basic password hashing, future enhancements can address these limitations by integrating a database and improving security. SmartAttend demonstrates the potential of combining web development and computer vision to address real-world challenges.

11. References

- Flask Documentation: https://flask.palletsprojects.com/
- OpenCV Documentation: https://docs.opencv.org/
- SocketIO Documentation: https://socket.io/docs/v4/
- Python Documentation: https://docs.python.org/3/
- NumPy Documentation: https://numpy.org/doc/
- Pillow Documentation: https://pillow.readthedocs.io/
- pytest Documentation: https://docs.pytest.org/
- pytest-html Documentation: https://pytest-html.readthedocs.io/
- matplotlib Documentation: https://matplotlib.org/