FACE RECOGNITION ATTENDENCE SYSTEM

TABLE OF CONTENTS

1. CHAPTER 1 INTRODUCTION

1.1 INTRODUCTION

The purpose of this document is to present a detailed description of the Face Recognition Attendance System. It will explain the purpose and features of the system, the interfaces of the system, what the system will do, the constraints under which it must operate and how the system will react to external stimuli. This document is intended for the Comm-IT Career Academy for its approval.

1.2 OBJECTIVE AND SCOPE

This software system will be a Face Recognition Attendance System. This system will be designed in such a way that everyone could easily find out the important things about college like- admission, training, placement, alumni, faculty, activity, events and many morefrom just few scrolls. The system will meet the visitors' needs while remaining easy to understand and use.

More specifically, this system is designed to allow all visitors, students and faculty to know information about college .In this website, a system administrator will manage the website.

1.3 PROCESS / SOFTWARE DEVELOPMENT METHODOLOGY

Agile Software Development Methodology:

The Attendance using Face Recognition System was developed using the Agile software development methodology, which emphasizes adaptive planning, iterative progress, and flexible responses to change.

Agile development allowed the project to evolve through multiple short development cycles or "iterations," each lasting between one to four weeks. During each iteration, specific features such as student registration, facial recognition, or attendance viewing were developed and refined.

Given the short timeframe for development and the evolving requirements of the system, Agile methodology was chosen to ensure flexibility. This approach allowed for quick releases, continuous feedback, and the ability to adapt the system based on changing needs while minimizing delays and risks.

2. CHAPTER 2 EFFORT AND COST ESTIMATION

Use Case Points

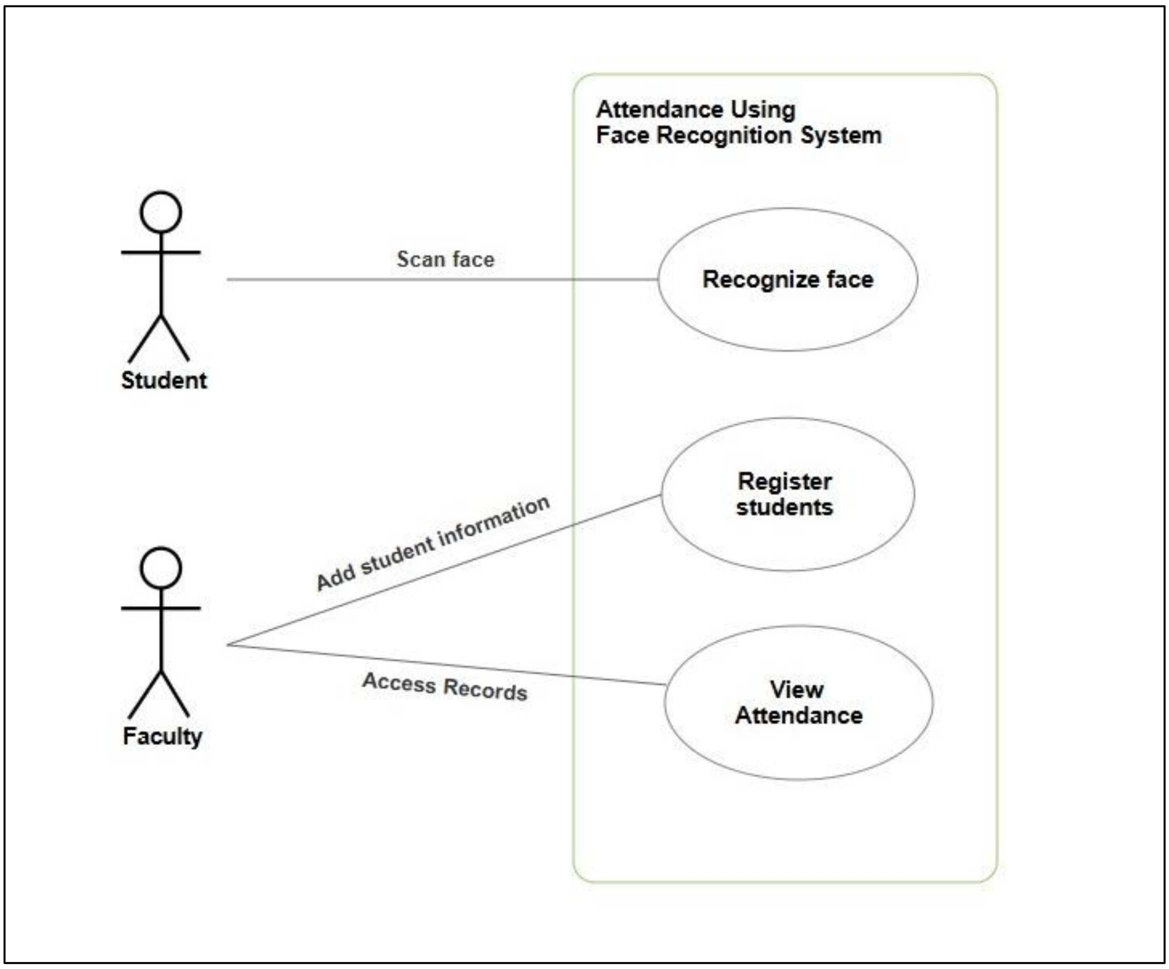


Figure 1

Unified Modeling Language (UML) made concept of “Use Cases” popular.IVAR JACOBSON developed technique of specifying Use Cases well before thedevelopment of UML in 1990 for which IVAR JACOBSON was one of the contributors. According to JONES,“A Use Case describes what happens to a software system when an actor (typically a user) sends a message or causes a software system to take some action.

Out of all Use Case based method of effort estimation UCP (USE CASE POINTS) is probably the most used, recognized and discussed method. Karner proposed following steps in counting Use Case points:

1. Unadjusted Actors Weights (UAW) Calculation. To any of these categories simple – API, average – protocol or terminal, complex – GUI, assign each of the actors, based on interaction of actor with the type of interface when communicating with the system. Multiply total number of actors assigned to each category with the weight assigned to respective category (simple = 1, average = 2, and complex = 3). UAW is the sum of products of number of actors assigned to categories and their weights.

	Simple –API	Average–Protocol, Terminal	Complex-GUI	
Student			3	
Faculty			3	

Table1

UAW= 3+3 = 6

UAW=6

2. Unadjusted Use Case Weights (UUCW) Calculation. Calculate number of transaction in each Use Case. Divide Use Cases in three categories based on number of transaction each Use Case has i.e. simple (T < 4), average (4 ≥T ≤ 7), and complex (T > 7). UUCW is the sum of products of number of Use Cases assigned to each category and their assigned weights (simple = 5, average = 10, and complex = 15).

	Simple T<4	Average 4>=T<=7	Complex T>7	
Recognize Face	5			
Maintain Student data		10		

View Attendance	5			

Table2

UUCW=5+10+5= 20

Table 2. Assess Technical Complexity Factors (TCF). Influence of each Technical Complexity Factor (Table 2.3) can be evaluated by assigning value from 0 to 5. TF_Prod is sum of products of the factors weights and their influence. TCF is given below in equation 2.2.

TCF = 0.6 + (0.01 x TF_Prod) …2.1

From Table 2.1

TCF=0.6+(0.01x22)

TCF=0.82

Table 2.3 Technical Complexity Factors and environmental factors [4]

Factor	**Description**	**Weight**	
Technical Complexity Factors			
T1	Distributed system	2	2x0=0
T2	Performance	1	1x5=5
T3	End user efficiency	1	1x5=5
T4	Complex processing	1	1x0=0
T5	Reusable code	1	1x0=0
T6	Easy to install	0.5	0.5x5=2.5
T7	Easy to use	0.5	0.5x5=2.5
T8	Por Table	2	2x0=0
T9	Easy to change	1	1x5=5
T10	Concurrent	1	1x0=0
T11	Security features	1	1x0=0
T12	Access to third parties	1	1x0=0
T13	Special training required	1	1x0=0
			TF_Prod =20
Environmental Factors			
F1	Familiarity with standard process	1.5	1.5x1=1.5
F2	Application experience	0.5	0.5x0=0
F3	Object Oriented experience	1	1x0=0
F4	Lead analyst capability	0.5	0.5x0=0
F5	Motivation	1	1x5=5
F6	Stable requirements	2	2x5=10
F7	Part Time workers	-1	-1x5=-5
F8	Difficult programming Language	-1	-1x5=-5
			EF_Prod=5

Table3

Table 2. Assess Environmental Factors (EF). Influence of each Environmental Factors (Table 2.3) can be evaluated by assigning value from 0 to 5. EF_Prod is sum of products of the factors weights and their influence. TCF is given below in equation 2.3.

EF = 1.4 + (-0.03 x EF_Prod) …2.2

EF=1.4-0.03x5=1.4-0.15=1.55

EF=1.

Table 2.3 Technical Complexity Factors and environmental factors [4]

Factor	Description	Weight
Technical Complexity Factors		
T1	Distributed system	2
T2	Performance	1
T3	End user efficiency	1
T4	Complex processing	1
T5	Reusable code	1
T6	Easy to install	0.5
T7	Easy to use	0.5
T8	PorTable	2
T9	Easy to change	1
T10	Concurrent	1
T11	Security features	1
T12	Access to third parties	1
T13	Special training required	1
Environmental Factors		
F1	Familiarity with standard process	1.5
F2	Application experience	0.5
F3	Object Oriented experience	1
F4	Lead analyst capability	0.5
F5	Motivation	1
F6	Table requirements	2
F7	Part Time workers	-1
F8	Difficult programming Language	-1

Table4

5. Unadjusted Use Case Points (UUCP) calculation: According to Eq. (2.4).

UUCP = UAW + UUCW ...2.3

UUCP=6+ 20=26

6. Use Case Points (UCP) Calculation: According to Eq. (2.4).

UCP = UUCP x TCF x EF ...2.4

UCP=26x0.82x1.55=33.046

7. UCP has to be multiplied by Productivity Factor (PF) to get effort estimation measured in man-hours. Karner proposed 20 hours per UCP as default value for PF. Schneider and winters proposed rule for choosing PF. They suggested "counting the

number of Environmental Factors F1–F6 which influence is predicted to be less than 3, and factors F7–F8 which influence is predicted to be greater than 3. If the counted total is equal to two or less, 20 [h/UCP] should be used; if the total is between 3 and 4, PF is set to 28 [h/UCP]; if the calculated number is greater than 4, the highest value of 36 [h/UCP] should be used."

8. Effort =33.046x20= 660.92 hours
   Cost = 660.92 x $30=$19827.

3. CHAPTER SOFTWARE REQUIREMENTS SPECIFICATION

3.1 INTRODUCTION

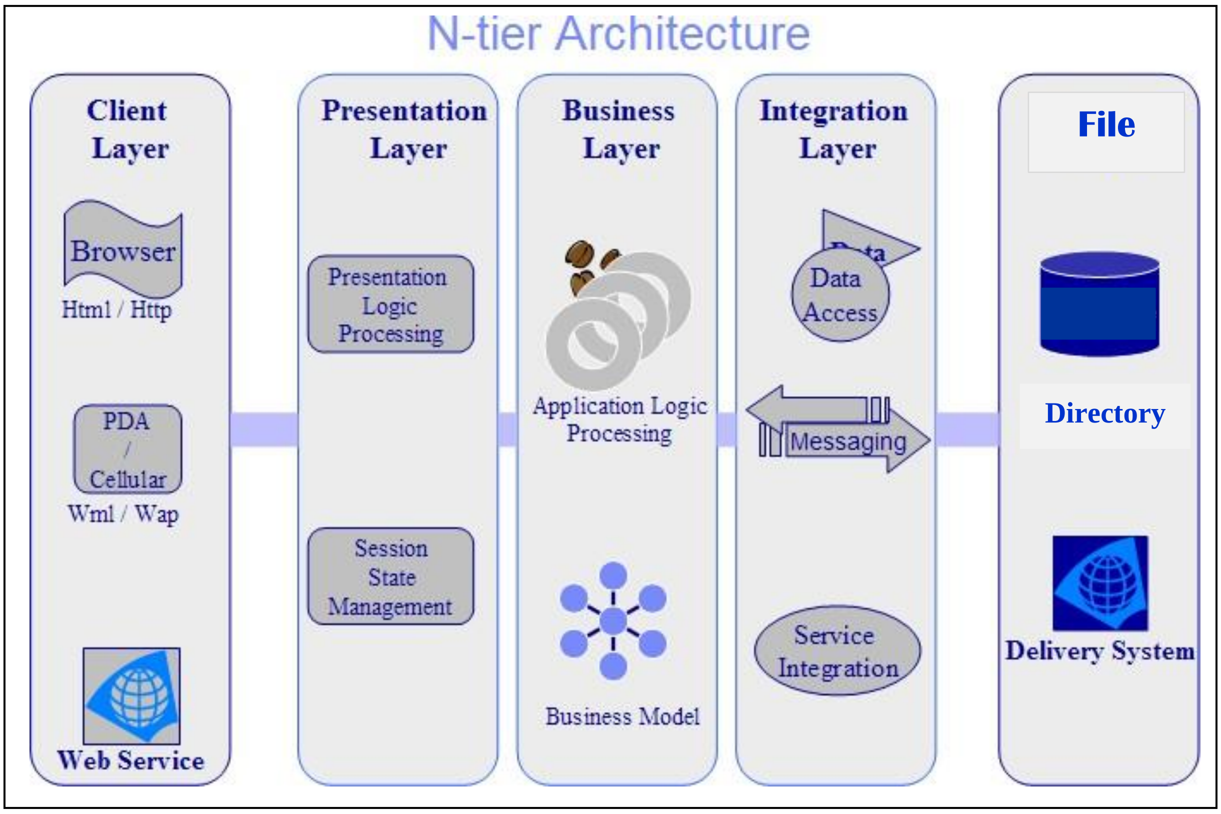
Software Requirements Specification presents a detailed description of the Face Recognition Attendance System It will explain the purpose and features of the system, the interfaces of the system, what the system will do, the constraints under which it must operate and how the system will react to external stimuli.

3.2 INTENDED AUDIENCE AND READING SUGGESTIONS

The document is intended for requirements engineer, domain expert, developers, project manager, stakeholders and in our case it is for Comm – IT Career Academy.

3.3 GENERAL ARCHITECTURE OF SOFTWARE

The Face Recognition Attendance System is divided into layers based on the N-tier architecture



Figure

The layering model of the Face Recognition Attendance System is based on a responsibility layeringstrategy that associates each layer with a particular responsibility.

This strategy has been chosen because it isolates various system responsibilities from one another, so that it improves both system development and maintenance.

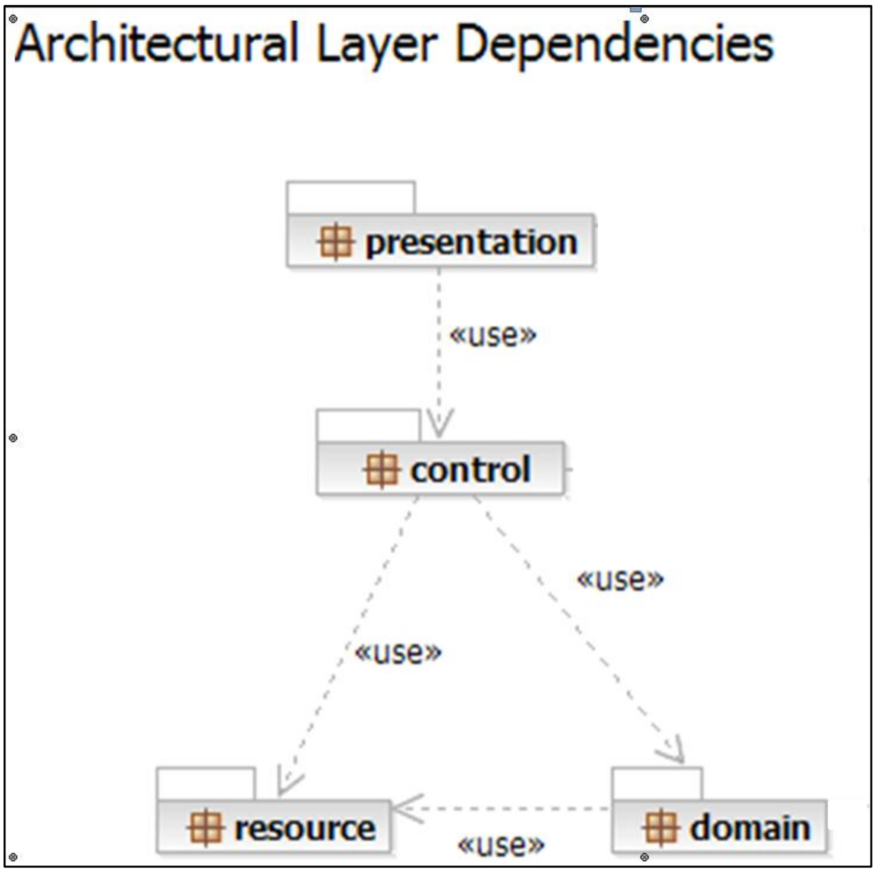


Figure 3

Each layer has specific responsibilities.

- The **presentation layer** deals with the presentation logic and the pages rendering
- The **control layer** manages the access to the domain layer
- The **resource layer** (integration layer) is responsible for the access to the enterprise information system (directory or other sources of information)
- The **domain layer** is related to the business logic and manages the accesses to the resource layer.

The Attendance using Face Recognition System application version 1.0 is quite simple and only contains twobasic features and screens.

3.4 REQUIREMENT SPECIFICATION

3.4.1 FUNCTIONAL REQUIREMENTS

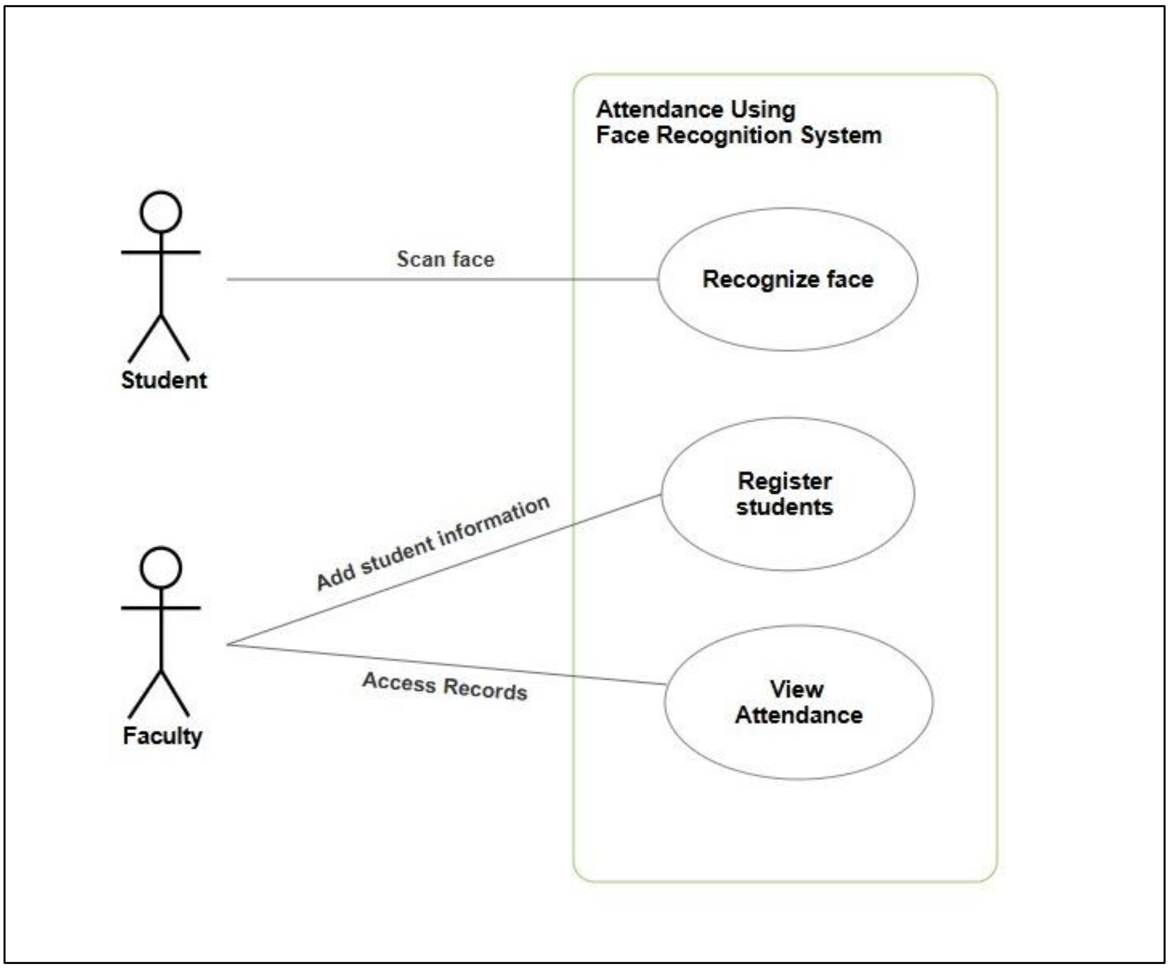


Figure 4

The diagram illustrates a use case model for an " Face Recognition Attendance System." This system enables students and faculty to interact with various features for registering, recognizing, and viewing attendance.

Actors :

Student: Represents individuals whose attendance will be tracked by the system.
Faculty: Represents teachers or administrators who manage student registrations and view attendance records.

Use Case: Recognize Face

Description

Allows a student to mark attendance by scanning their face.

Pre-conditions

The student must be registered in the system with their face data and personal information stored.
The system's camera and face recognition software must be functioning and accessible.

Post-conditions
If successful, the student's attendance for the current session is marked in the database or attendance record.
If unsuccessful, no attendance record is created, and the student may need to retry or contact the faculty.

Primary Flow
Student stands in front of the camera.
System scans the student's face.
If a match is found in the database, attendance is successfully marked.

Alternate Flow
AF1: Poor Lighting or Obstruction
If lighting is poor or the student’s face is partially obstructed, the system prompts the student to adjust their position or lighting.
Comment: This helps ensure accurate face recognition despite environmental factors.

Exceptional Flow
EF1: Face Not Recognized
If the system cannot recognize the face after multiple attempts, it displays an error message and suggests contacting faculty.
Comment: Helps in cases where the student is not in the database or face recognition fails due to changes in appearance.
EF2: Multiple Faces Detected
If multiple faces are detected, the system prompts only one person to stand in front of the camera.
Comment: Ensures accurate identification by isolating the individual face.
EF3: Database Connectivity Issue
If there’s a connectivity issue with the database, the system displays an error, and attendance is not marked.
Comment : This prevents incorrect or missing data due to technical issues.

Use Case: Register Students

Description
Allows faculty to add a new student to the system by capturing their details and face data.

Pre-conditions
The faculty must have login access to the system.
Camera and storage (database and Excel/CSV file) should be available and functional.

Post-conditions
The student’s information (name, enrollment number, semester, face data) is stored in the system database and Excel/CSV file.
If unsuccessful, the system does not store partial data, ensuring data integrity.

Primary Flow
Faculty enters the student’s details (name, enrollment number, semester).
System captures and stores the student’s face and information.

Alternate Flow
AF1: Student’s Face Already Registered

If the system detects that the face has already been registered, it notifies the faculty. Faculty can choose to update the record or ignore the duplicate entry.
Comment: Prevents redundant registrations while allowing updates if needed.
AF2: Partial Data Entry
If some required data fields are blank, the system prompts the faculty to complete all fields.
Comment: Ensures that all necessary information is recorded to avoid incomplete records.

Exceptional Flow
EF1: Face Capture Failure
If the camera fails to capture the face, the system displays an error message, and the faculty can retry.
Comment: Provides a chance to correct technical issues before proceeding.
EF2: File Write Error
If there’s an error while writing to the Excel/CSV file, the system displays an error and halts the registration process.
Comment: Protects data integrity by preventing partial or erroneous data storage.
EF3: Duplicate Enrollment Number
If the enrollment number is already registered, the system alerts the faculty, preventing duplication.
Comment :Ensures each student has a unique identifier.

<u>Use Case: View Attendance</u>

Description
Allows faculty to view attendance records by filtering by semester or student name.

Pre-conditions
The faculty must have login access to the system.
Attendance records must be stored in the system database and Excel/CSV file.

Post-conditions
If successful, attendance records matching the filter criteria are displayed.
If unsuccessful, no records are displayed, and the faculty may need to retry or adjust the criteria.

Primary Flow
Faculty selects the criteria for viewing attendance (e.g., by semester or student name).
System retrieves and displays the attendance records.

Alternate Flow
AF1: Date Filter
Faculty applies a date range to filter attendance records (e.g., for a specific week or month).
Comment: Allows flexible record viewing by time period.

Exceptional Flow
EF1: No Records Found
If no records match the selected criteria, the system displays a “No Records Found” message, allowing the faculty to adjust search parameters.
Comment: Prevents confusion if records aren’t available for the selected criteria.
EF2: Database Connectivity Issue
If there is a connectivity issue with the attendance database, the system displays an error message.
Comment: Prevents viewing partial or outdated records due to connectivity issues.

EF3: File Read Error
If there’s an issue reading from the Excel/CSV file (e.g., file corruption or access permission), the system displays an error.
Comment: Prompts the faculty to check file access issues before retrying.

3.4.2 NON-FUNCTIONAL REQUIREMENTS

Reliability
Availability - System will be available 99.99%of time.

Performance
Response time for a transaction is 5 second

Design Constraints
None

Security
None

3.5 FEASIBILITY STUDY

None

3.5.1 OPERATIONAL FEASIBILITY

Proposed solution will work because system is user friendly for end user.

3.5.2 TECHNICAL FEASIBILITY

This is a web based system which will be deployed on windows platform.

3.6 SYSTEM REQUIREMENTS STUDY

3.6.1 SOFTWARE REQUIREMENTS

It is a web based application and can be accessed using popular browsers like chrome or Microsoft edge.

3.6.2 HARDWARE REQUIREMENTS

1 GHz processor or faster 32-bit (x86) or 64-bit (x64).

1 GB of RAM for 32-bit or 2 GB of RAM for 64-bit or more the better

16 GB of hard drive space for 32-bit or 20 GB for 64-bit.

3.7 USER REQUIREMENT DOCUMENT (URD)

3.7.1 USE-CASE DIAGRAM

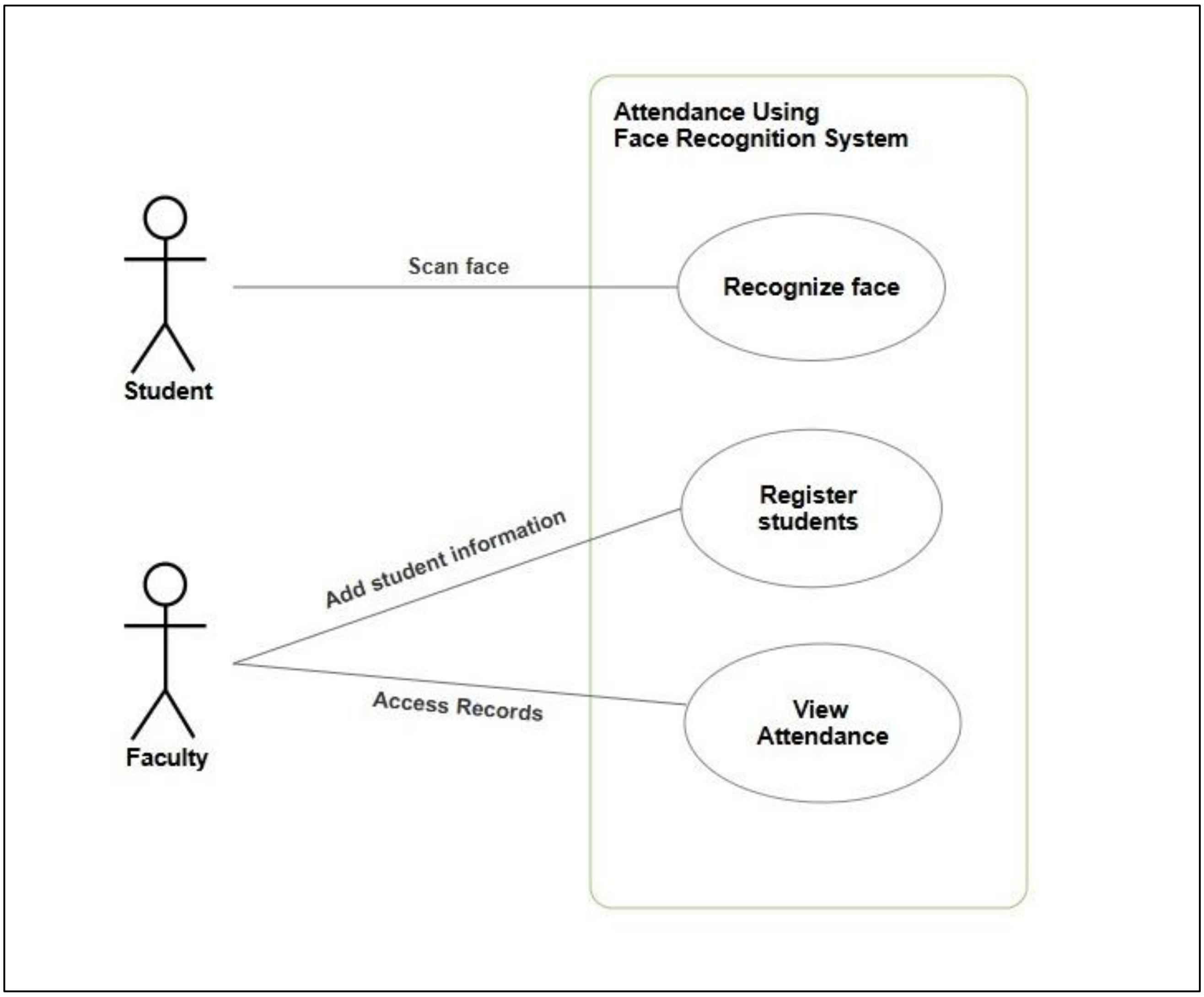


Figure 5

3.7.2 ACTIVITY DIAGRAM

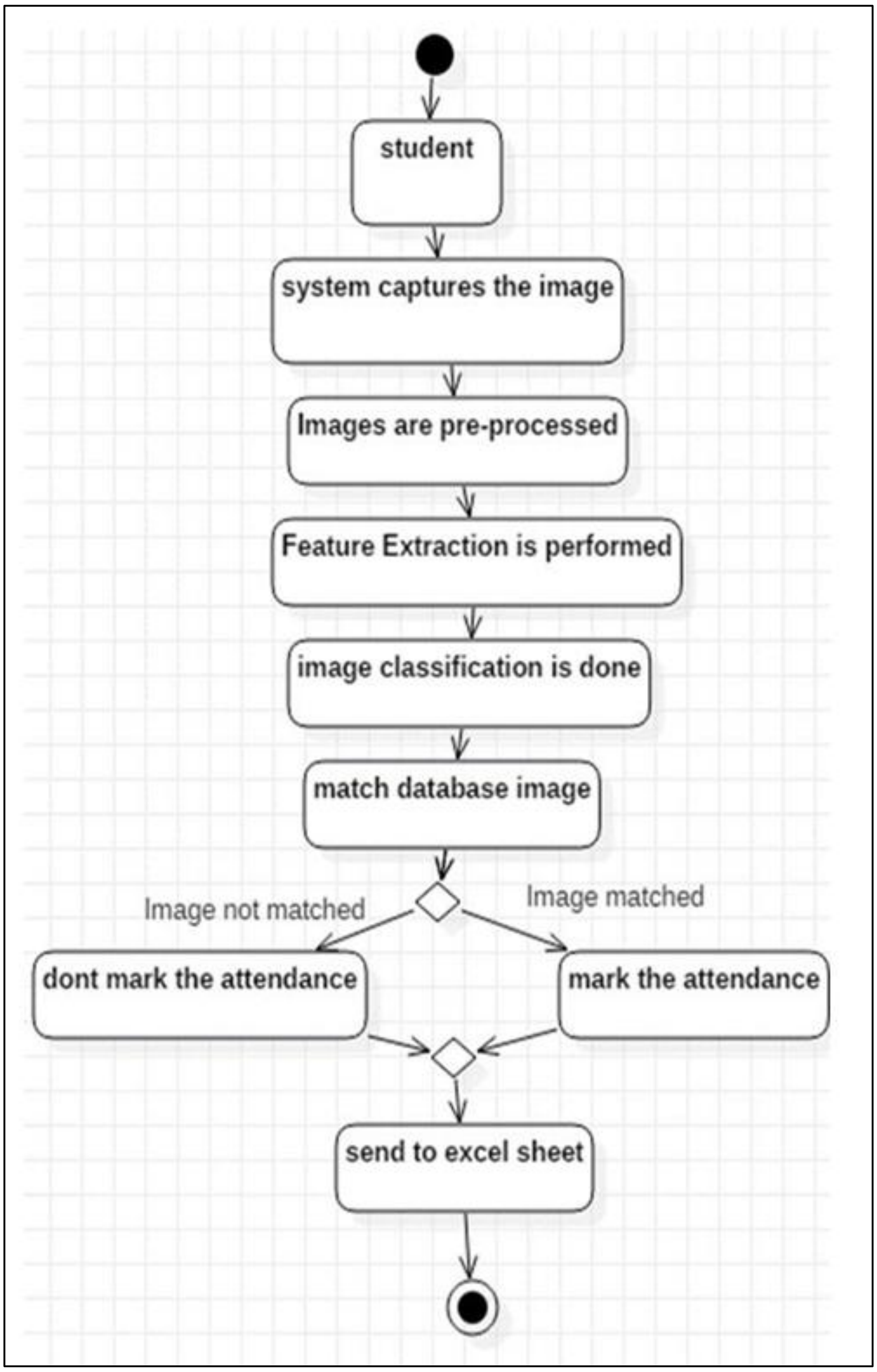


Figure 6

3.8 SYSTEM DESIGN

3.8.1 INTRODUCTION

Systems design is the process of defining elements of a system like modules, architecture, components and their interfaces and data for a system based on the specified requirements. It is the process of defining, developing and designing systems which satisfies the specific needs and requirements of a business or organization.

UML is used as a modeling language to express design aspects of the system. Unified Modeling Language (UML) describes software both structurally and behaviorally with graphical notation.

3.8.2 DATA FLOW DIAGRAM

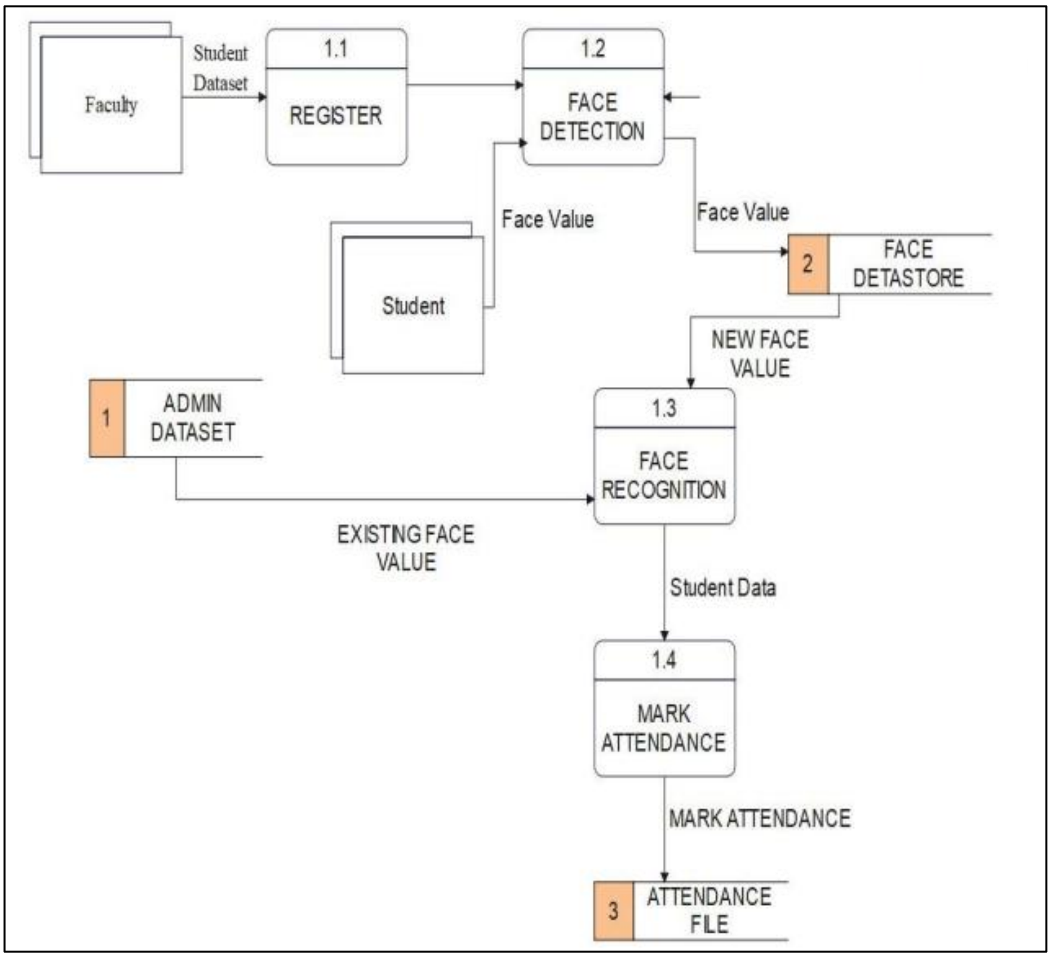


Figure 7

3.8.3 SEQUENCE DIAGRAM

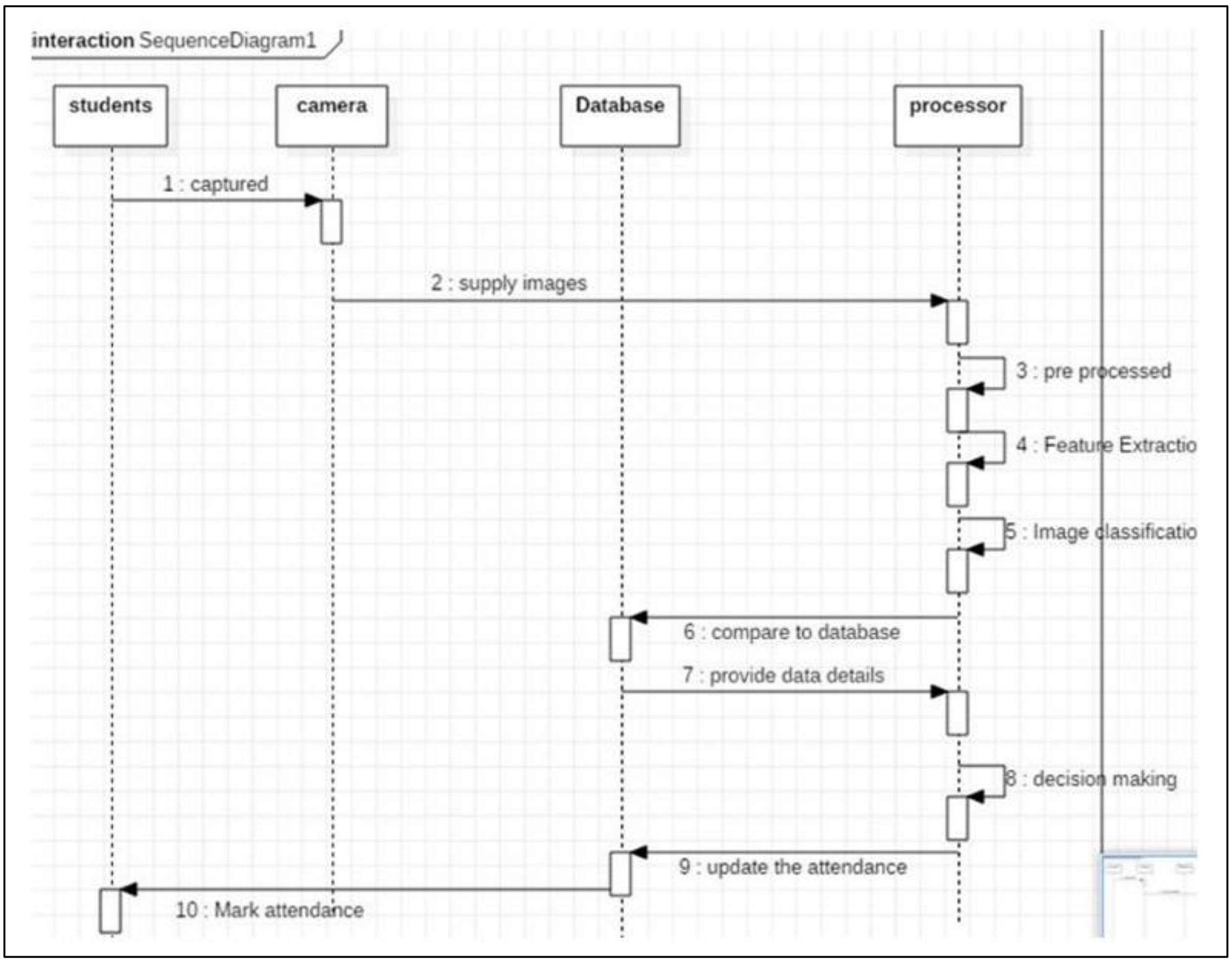


Figure 8

3.8.4 CLASS DIAGRAM

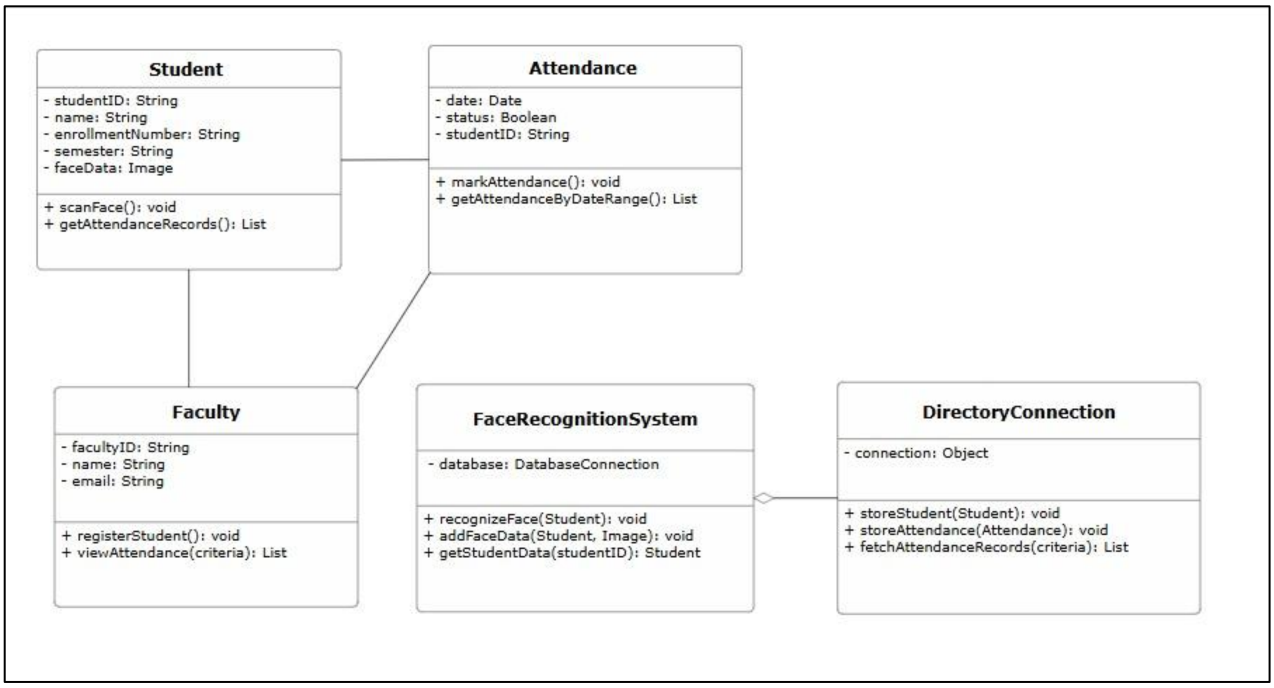
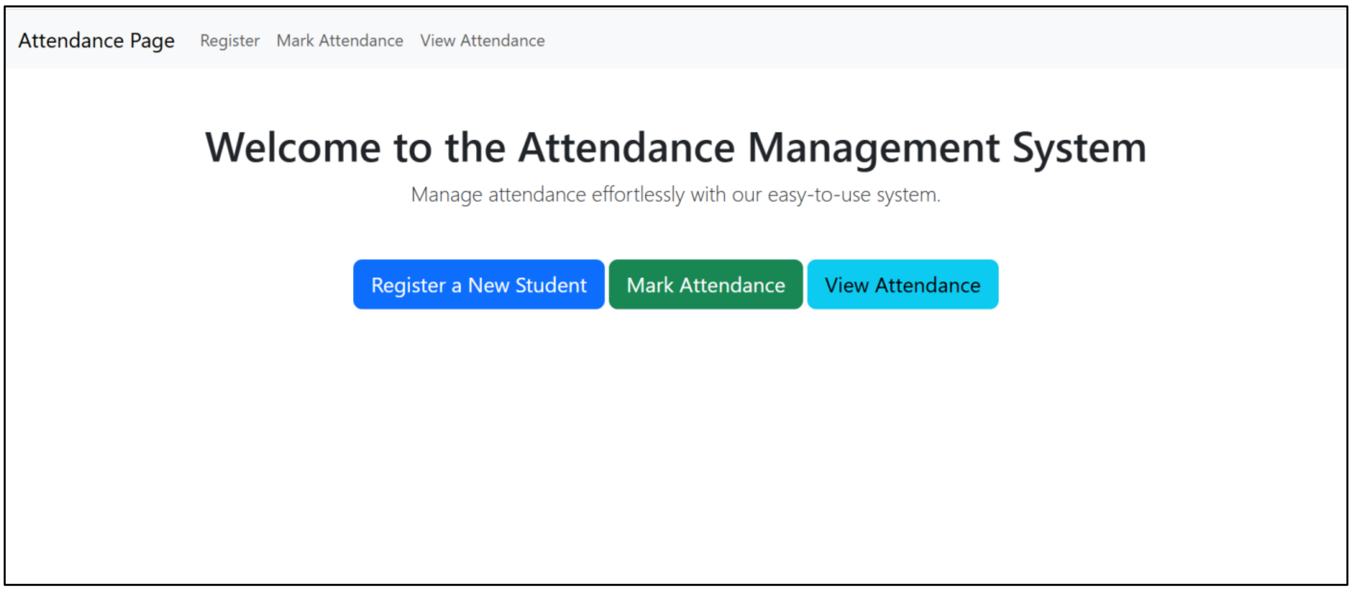

Student
- studentID: String
- name: String
- enrollmentNumber: String
- semester: String
- faceData: Image
+ scanFace(): void
+ getAttendanceRecords(): List
Attendance
- date: Date
- status: Boolean
- studentID: String
+ markAttendance(): void
+ getAttendanceByDateRange(): List
Faculty
- facultyID: String
- name: String
- email: String
+ registerStudent(): void
+ viewAttendance(criteria): List
FaceRecognitionSystem
- database: DatabaseConnection
+ recognizeFace(Student): void
+ addFaceData(Student, Image): void
+ getStudentData(studentID): Student
DirectoryConnection
- connection: Object
+ storeStudent(Student): void
+ storeAttendance(Attendance): void
+ fetchAttendanceRecords(criteria): List

Figure 9

4. CHAPTER SCREENSHOTS

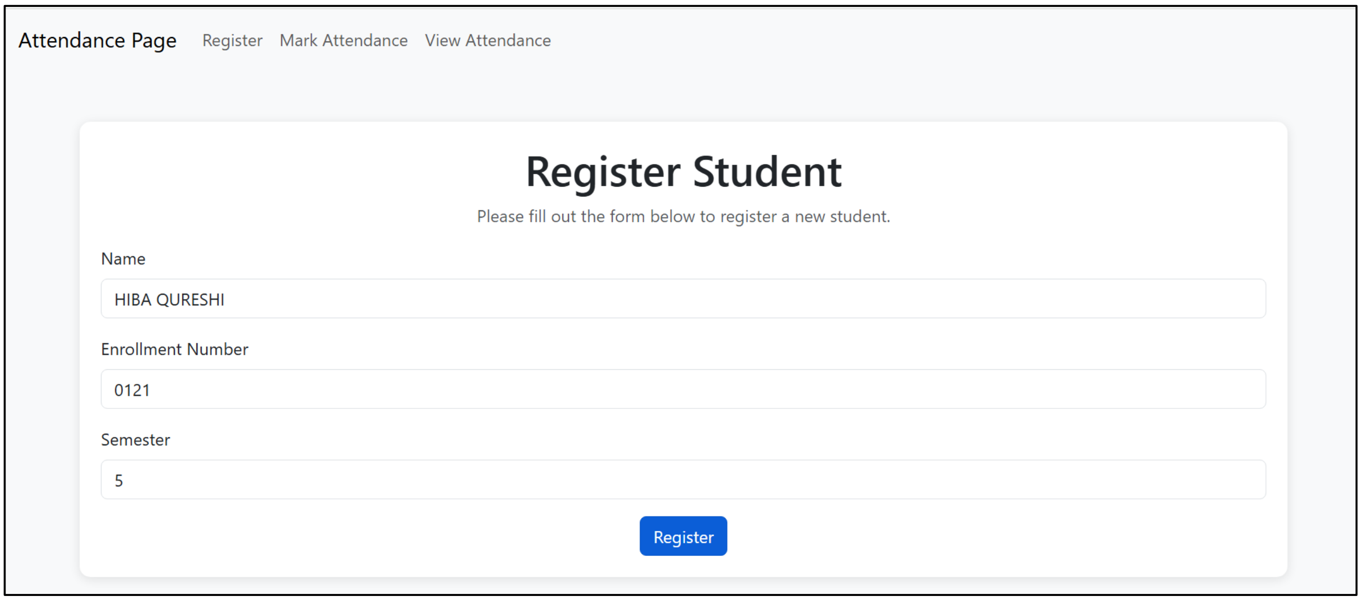
Screen 1



Welcome to the Attendance Management System

Manage attendance effortlessly with our easy-to-use system.

Screen 2



Register Student

Please fill out the form below to register a new student.

Name

HIBA QURESHI

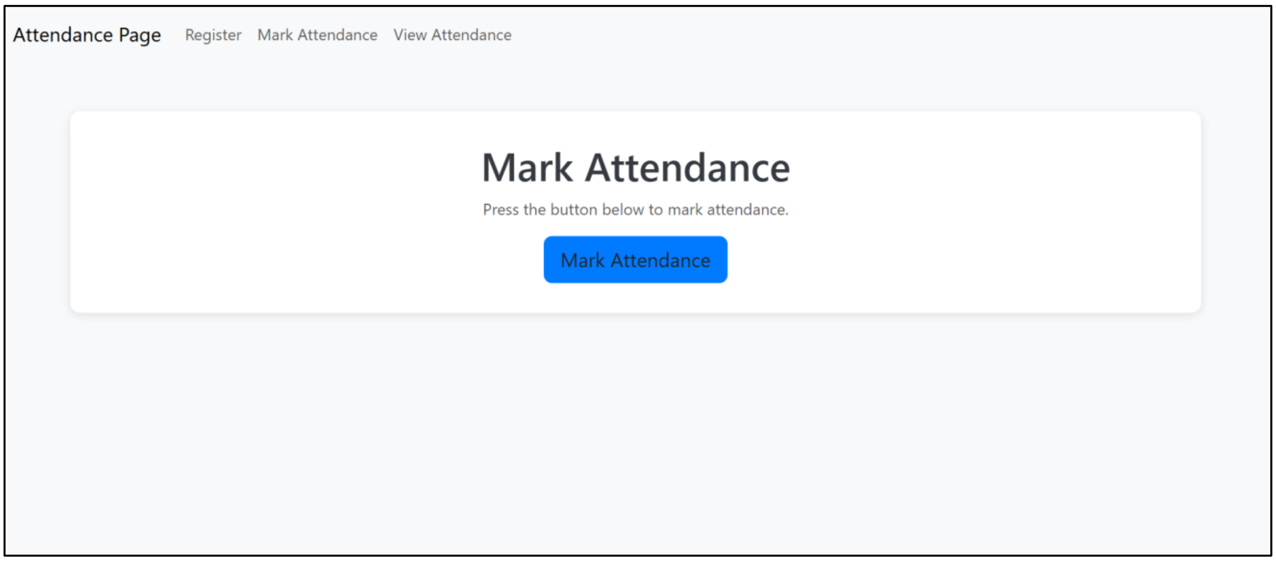
Enrollment Number

0121

Semester

5

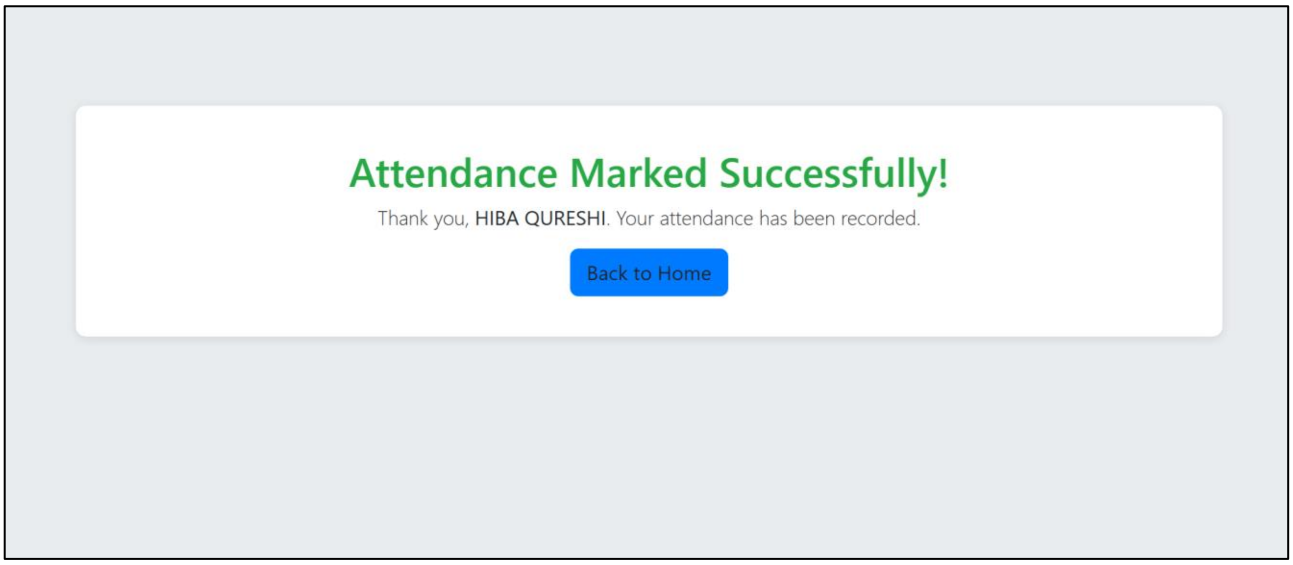
Screen 3



Mark Attendance

Press the button below to mark attendance.

Screen 4

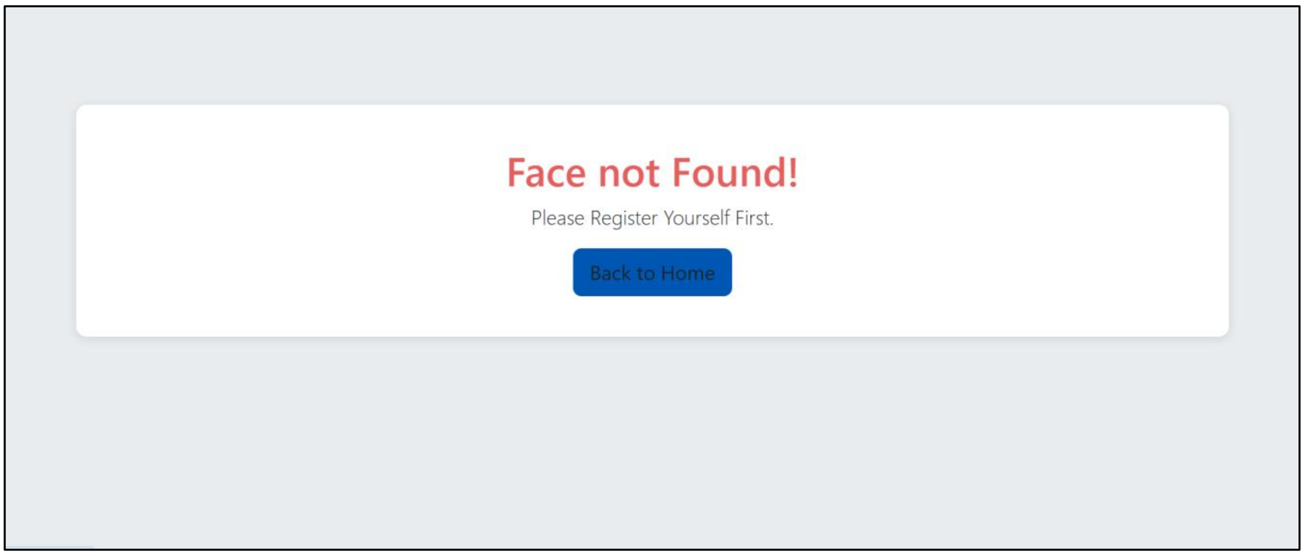


Attendance Marked Successfully!

Thank you, HIBA QURESHI. Your attendance has been recorded.

Screen 5

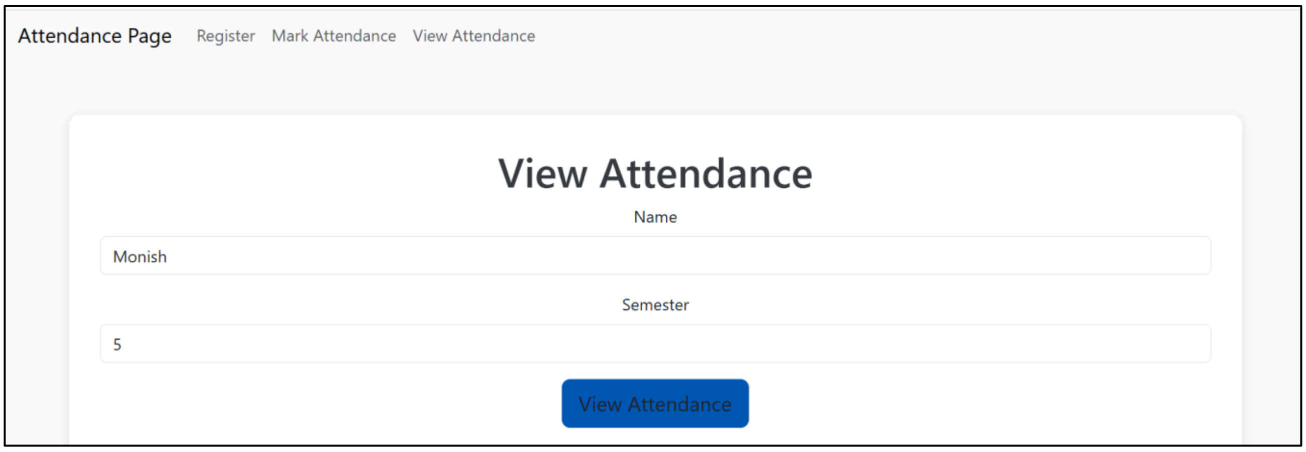
If an unregistered user marks his/her attendance or an unrecognized face is displayed.



Face not Found!

Please Register Yourself First.

Screen 6

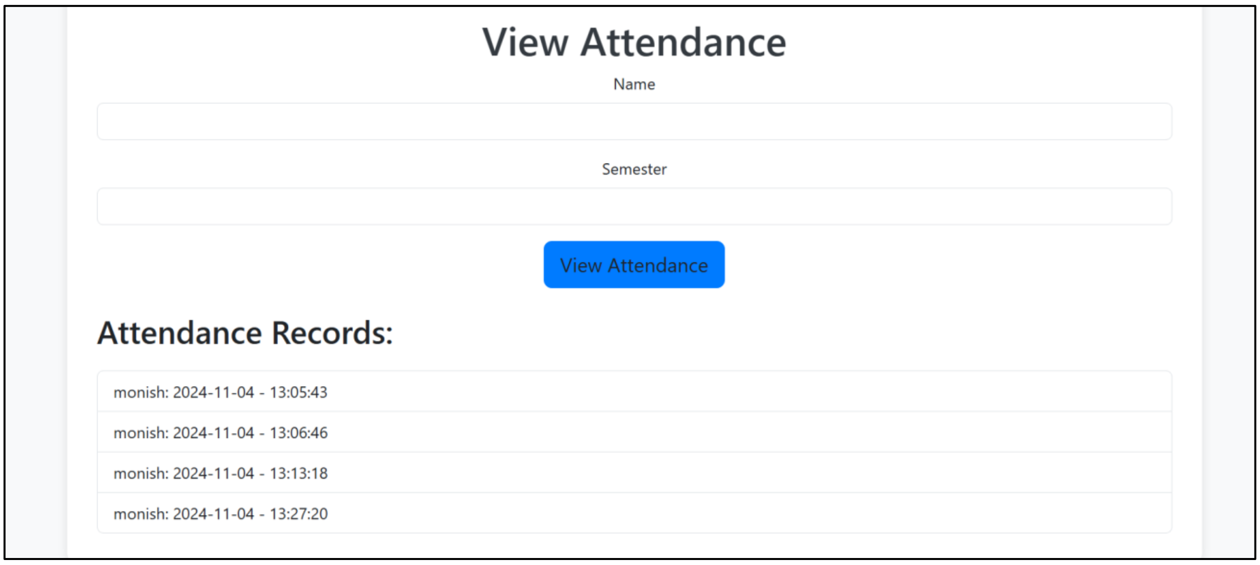


View Attendance

Name: Monish

Semester: 5

Screen 7

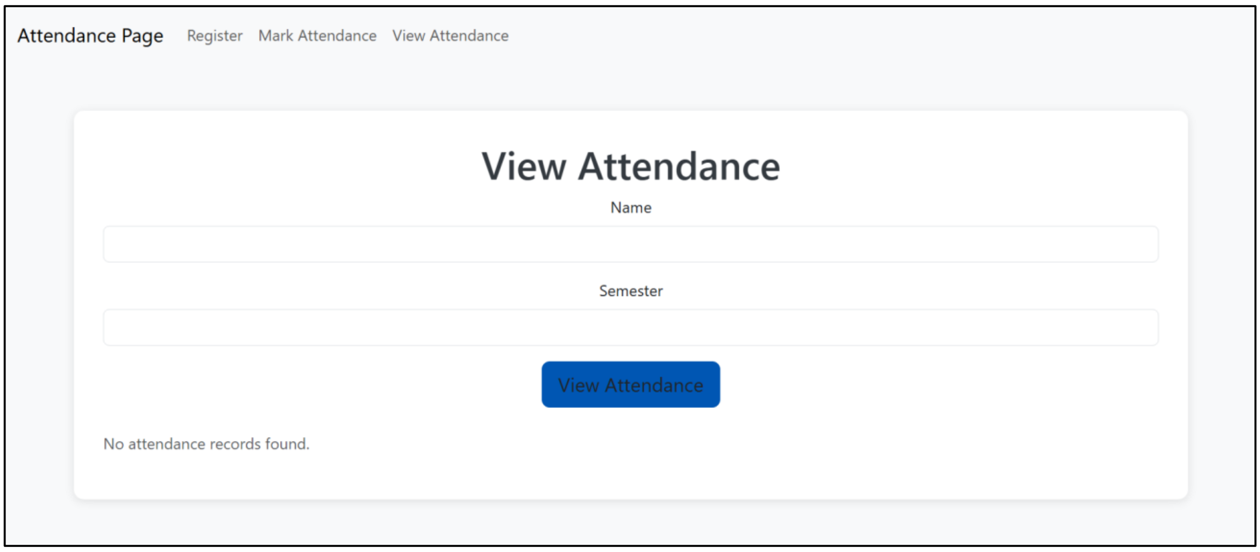


Attendance Records:

```
monish: 2024-11-04 - 13:05:43
monish: 2024-11-04 - 13:06:46
monish: 2024-11-04 - 13:13:18
monish: 2024-11-04 - 13:27:20
```

Screen 8

If wrong details are entered or no attendance record is found



No attendance records found.

5. CHAPTER TECHNOLOGY USED

HTML
CSS
Python

STANDARD CODING CONVENTIONS HAS BEEN USED

Python Coding Conventions (PEP 8): A Comprehensive Guide

Python's coding conventions, as outlined in PEP 8, ensure that code is readable, maintainable, and consistent across projects. Here's a detailed breakdown of essential conventions.

---

1. Indentation

- **Use 4 Spaces Per Level**: Indentation improves readability and is required by Python's syntax.
- **Avoid Tabs**: Mixing tabs and spaces can lead to errors. Consistency is critical; PEP 8 recommends using **spaces only**.
- **Example**:

```
python
Copy code
def my_function():
    if condition:
        do_something()
```

---

2. Maximum Line Length

- **79 Characters for Code**: This limit makes code easier to read, especially on standard-sized monitors.
- **72 Characters for Comments**: Keeps comments compact.
- **Breaking Long Lines**: Use parentheses for line continuation or the backslash (\) operator if needed.
- **Example**:

```
python
Copy code
# Example with line continuation
total = (first_variable + second_variable + third_variable
         + fourth_variable)
```

---

3. Blank Lines

- **Separate Functions and Classes**: Use two blank lines to separate top-level functions and classes.
- **Within Functions**: Separate logical sections with a single blank line for clarity.
- **Example**:

```
python
Copy code
class MyClass:
    def method_one(self):
        pass

    def method_two(self):
        pass
```

---

4. Imports

- **Order of Imports**: Group imports in the following order:
  1. **Standard library imports**
  2. **Related third-party imports**
  3. **Local application/library-specific imports**
- **One per Line**: Import one module per line for clarity.
- **Avoid Wildcard Imports**: Use `from module import specific_function` rather than `from module import *`.
- **Example**:

  ```
  python
  Copy code
  import os
  import sys

  import requests
  ```

---

5. Naming Conventions

- **Variables and Functions**: Use `lower_case_with_underscores`.
- **Constants**: Use `UPPERCASE_WITH_UNDERSCORES`.
- **Classes**: Use `CamelCase`.
- **Private Variables/Functions**: Prefix with a single underscore (`_var`), and use double underscores (`__var`) for strongly private.
- **Example**:

  ```
  python
  Copy code
  class ExampleClass:
      def __init__(self, value):
          self._private_value = value
  ```

---

6. Function and Variable Annotations

- **Type Hints**: Improve readability and help with static analysis tools.
- **Syntax**:

  ```
  python
  Copy code
  def function_name(param1: str, param2: int) -> bool:
      pass
  ```

- **Optional Annotations**: Use `Optional[Type]` or `Union[Type1, Type2]` for flexibility.
- **Example**:

  ```
  python
  Copy code
  from typing import Optional
  ```

```
def greet(name: Optional[str] = None) -> str:
    if name:
        return 'Hello ' + name
    else:
        return 'Hello, World!'
```

---

7. Comments

- **Explain Why, Not What**: Comments should clarify the "why" behind code decisions.
- **Inline Comments**: Place two spaces away from the code and use sparingly.
- **Block Comments**: Use full sentences, and align them with the indentation level of the code.
- **Example**:

```
python
Copy code
# Initialize variable for tracking score
score = 0  # Starting score for game
```

---

8. Docstrings

- **PEP 257 Guidelines**: Follow PEP 257 for consistent formatting.
- **Modules, Functions, and Classes**: Every public method, function, or class should have a docstring.
- **Multi-line Docstrings**: Include a summary line followed by a blank line and a more detailed explanation.
- **Example**:

```
python
Copy code
def add(a, b):
    """Add two numbers.

    Args:
        a (int): The first number.
        b (int): The second number.

    Returns:
        int: The sum of the two numbers.
    """
    return a + b
```

---

9. Whitespace in Expressions and Statements

- **Avoid Extraneous Whitespace**:
  - Inside parentheses, brackets, or braces.
  - Before commas, colons, and semicolons.
- **Binary Operators**: Use a single space on either side of binary operators (e.g., `a + b`).
- **Example**:

```
python
Copy code
x = (1 + 2) * (3 + 4)
```

---

10. Exception Handling

- **Specific Exceptions**: Catch specific exceptions instead of using a bare `except`.
- **Avoid Silent Failures**: If an exception is caught, log it or handle it in a way that preserves error information.
- **Example**:

```
python
Copy code
try:
    result = 1 / 0
except ZeroDivisionError as e:
    print(f"Error occurred: {e}")
```

---

11. Boolean Expressions

- **Implicitly Evaluated Conditions**: Avoid `== True` or `== False`. Instead, use implicit conditions.
- **Example**:

```
python
Copy code
if is_valid:  # instead of is_valid == True
    print("Valid")
```

---

12. Classes and Inheritance

- **New-Style Classes**: Use `class ClassName(object)` (in Python 2) or simply `class ClassName:` (Python 3).
- **Method Order**: Place `__init__`, then public methods, and private methods at the bottom.
- **Example**:

```
python
Copy code
class Animal:
    def __init__(self, name):
        self.name = name

    def speak(self):
        pass
```

---

13. Context Managers and `with` Statements

- **File Handling**: Use `with` statements to ensure files and resources are properly closed.
- **Example**:

```
python
Copy code
with open('file.txt', 'r') as file:
    content = file.read()
```

---

14. Testing Conventions

- **Naming Tests**: Use descriptive test names with prefixes like `test_` for easy identification.
- **Assertion and Error Messages**: Include clear messages in assertions.

- **Example**:

  ```
  python
  Copy code
  def test_add():
      assert add(2, 3) == 5, "Addition result incorrect"
  ```

---

15. Lambda Functions

- **Use Sparingly**: Only for short, simple expressions.
- **Example**:

  ```
  python
  Copy code
  squares = map(lambda x: x**2, range(10))
  ```

---

16. Function Arguments

- **Avoid Too Many Arguments**: Aim for clarity and simplicity; consider using data structures if arguments exceed 5.
- **Keyword Arguments**: Use them to improve readability in calls.
- **Example**:

  ```
  python
  Copy code
  def calculate_area(width, height, *, unit='cm'):
      return f"{width * height} {unit}²"
  ```

---

17. Global Variables

- **Avoid Globals**: Use constants or classes to encapsulate state where possible.
- **Example**:

  ```
  python
  Copy code
  GLOBAL_VALUE = 10
  ```

18. Dunder Methods (Double-underscore)

- **Usage**: Implement `__str__`, `__repr__`, and other special methods to customize object behavior.
- **Example**:

  ```
  python
  Copy code
  class Person:
      def __init__(self, name):
          self.name = name

      def __str__(self):
          return f"Person named {self.name}"
  ```

HTML Coding Conventions

1. HTML Document Structure and Formatting

- **Doctype Declaration**: Always start with the `<!DOCTYPE html>` declaration to ensure the document is rendered in standards mode.
- **HTML Tags**: Organize HTML tags in a logical order. A typical HTML document structure includes:

```
html
Copy code
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Page Title</title>
</head>
<body>
    <!-- Page content -->
</body>
</html>
```

- **Indentation**: Use 2 or 4 spaces per indentation level (4 is recommended). Avoid using tabs to ensure consistency across text editors.
- **Lowercase Tags**: Write all HTML tags, attributes, and values in lowercase. This ensures consistency and adheres to XHTML standards.

---

2. Naming Conventions for Classes and IDs

- **Descriptive Naming**: Use meaningful names for classes and IDs that clearly describe their function or purpose (e.g., `.navbar`, `.footer`, `#main-content`).
- **BEM (Block, Element, Modifier) Convention**: Use the BEM naming convention for classes to keep CSS modular and maintainable:
    - Block: `.navbar`
    - Element: `.navbar__link`
    - Modifier: `.navbar--large`

```
html
Copy code
<div class="navbar navbar--large">
    <a href="#" class="navbar__link">Home</a>
</div>
```

- **Avoid Generic Names**: Avoid using names like `container`, `header`, or `text` which are too generic and can lead to confusion.

---

3. Attribute Order

- **Attribute Order**: Follow a consistent attribute order to improve readability. Recommended order:
    - `id`, `class`, `name`, `data-*`, `src`, `for`, `type`, `href`, `alt`, `title`, `aria-*`
- **Quotes**: Use double quotes for attribute values (`"`), rather than single quotes.

- **Boolean Attributes**: For boolean attributes (like `disabled`, `checked`), omit the value and just write the attribute name:

  ```
  html
  Copy code
  <input type="checkbox" checked>
  ```

---

4. Comments and Documentation

- **HTML Comments**: Use comments to indicate sections, explain complex structures, or mark closing tags when they are nested.
- **Section Comments**: Add comments at the beginning and end of major sections for easy navigation.
- **Closing Tag Comments**: In deeply nested code, label closing tags to clarify which section they belong to.

Example:

```
html
Copy code
<!-- Main navigation -->
<nav class="navbar">
    <!-- Links go here -->
</nav>
<!-- End of Main navigation -->
```

---

5. Accessibility Standards

- **Semantic HTML**: Use semantic HTML elements (e.g., `<header>`, `<main>`, `<footer>`, `<article>`) to structure your content, enhancing readability and accessibility.
- **Alt Text**: Always add `alt` attributes to `<img>` tags, providing descriptive text for screen readers.
- **ARIA Attributes**: Use ARIA roles (`aria-label`, `aria-labelledby`, `role`) to enhance accessibility for screen readers when necessary.
- **Form Accessibility**: Use `<label>` elements for form inputs and link them with the `for` attribute.

  ```
  html
  Copy code
  <label for="username">Username:</label>
  <input type="text" id="username" name="username">
  ```

---

6. Semantic HTML and Structural Organization

- **Use Headings Hierarchically**: Headings should follow a logical hierarchy (`<h1>` to `<h6>`) for better SEO and accessibility.
- **Sectioning Elements**: Use `<header>`, `<nav>`, `<main>`, `<section>`, `<article>`, and `<footer>` elements to organize the document.
- **Avoid Using `<div>` and `<span>` Excessively**: These elements should only be used when no semantic element is suitable.

Example:

```
html
Copy code
<header>
```

```
    <h1>Website Title</h1>
    <nav>
        <ul>
            <li><a href="#home">Home</a></li>
            <li><a href="#about">About</a></li>
        </ul>
    </nav>
</header>
```

---

7. Forms and Input Elements

- **Use `<label>` Tags**: Every `<input>` field should be associated with a `<label>` tag for accessibility.
- **Placeholder Text**: Use placeholder text sparingly, as it can disappear once users start typing and may be insufficient for accessibility.
- **Validation**: Use HTML5 validation attributes (e.g., `required`, `type="email"`, `pattern`) to ensure data accuracy.
- **Button vs. Input**: Prefer `<button>` over `<input type="submit">` for better styling flexibility.

---

8. Best Practices for Links

- **Descriptive Text**: Link text should describe the destination, not just “click here.” This improves accessibility and SEO.
- **External Links**: For links that open in a new tab, add `target="_blank"` and `rel="noopener noreferrer"` for security.
- **Avoid Link Styling on Non-links**: Do not use link styles (like blue underlined text) on elements that are not links to prevent confusion.

Example:

```
html
Copy code
<a href="https://example.com" target="_blank" rel="noopener noreferrer">Visit
Example Site</a>
```

---

9. Image and Media Conventions

- **Use Appropriate File Types**: Use `.jpg` for photos, `.png` for graphics with transparency, and `.svg` for icons.
- **Responsive Images**: Use the `srcset` attribute for responsive images that adapt to different screen sizes.
- **Lazy Loading**: For performance, use `loading="lazy"` on images to load them only when they appear in the viewport.

Example:

```
html
Copy code
<img src="image.jpg" alt="Description of image" loading="lazy">
```

---

10. CSS and JavaScript Linking

- **External Files**: Link external CSS files in the `<head>` section and JavaScript files before the closing `</body>` tag to improve loading speed.
- **Minify and Combine**: Minify and combine CSS and JavaScript files to reduce file size and improve performance.
- **Avoid Inline Styles and JavaScript**: Inline styles and scripts can make the HTML harder to read and maintain.

Example:

```
html
Copy code
<link rel="stylesheet" href="styles.css">
<script src="scripts.js" defer></script>
```

---

11. Performance Optimization

- **Use HTML5 `async` and `defer`**: For non-essential JavaScript, use `async` or `defer` to improve load time without blocking HTML parsing.
- **Optimize Image Sizes**: Compress images for web to avoid loading large files.
- **Use a Content Delivery Network (CDN)**: For libraries (like Bootstrap, jQuery), use a CDN to improve loading times and take advantage of caching.

Example:

```
html
Copy code
<script src="https://code.jquery.com/jquery-3.6.0.min.js" defer></script>
```

---

12. SEO Best Practices

- **Meta Tags**: Add essential meta tags for SEO, including `meta description` and `meta keywords`.
- **Headings**: Use `<h1>` only once per page to represent the main heading, with descending levels for subheadings.
- **Structured Data**: Implement structured data (e.g., JSON-LD) where applicable to improve search engine understanding.

Example:

```
html
Copy code
<meta name="description" content="This is a sample website description">
```

---

13. Responsive Design

- **Viewport Meta Tag**: Always include the viewport meta tag for responsive design.
- **Media Queries**: Use CSS media queries for responsive layouts.
- **Flexbox and Grid**: Use Flexbox and CSS Grid for modern, responsive layouts instead of older techniques like floats.

Example:

```
html
Copy code
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

---

14. Security Best Practices

- **Avoid Inline JavaScript and CSS**: Inline scripts and styles can make the code more vulnerable to XSS attacks.
- **Sanitize User Input**: Ensure any input data displayed on the page is sanitized to prevent XSS.
- **Content Security Policy (CSP)**: Use a Content Security Policy to restrict sources of content.
- **rel="noopener noreferrer"**: When using `target="_blank"`, add `rel="noopener noreferrer"` to prevent the new page from accessing the original page.

---

15. Testing and Validation

- **HTML Validators**: Regularly validate HTML using tools like the W3C HTML Validator to catch syntax errors.
- **Cross-Browser Testing**: Test across multiple browsers to ensure consistency.
- **Accessibility Testing**: Use tools like WAVE or Axe to check for accessibility issues.

CSS Coding Conventions

CSS coding conventions standardize how we write CSS, making code consistent, readable, and maintainable. These conventions cover various aspects, including structure, naming, formatting, and organization.

1. General Principles

- **Consistency**: Apply the same rules and patterns throughout the codebase.
- **Readability**: Code should be easy to read and understand.
- **Modularity**: Structure code so it's easy to change, reuse, and test.
- **Maintainability**: Write code that is easy to debug and update in the future.

---

2. File Organization

- **Separate Files for Each Component**: Use separate files for layout, components, and utilities, e.g., `layout.css`, `header.css`, and `buttons.css`.
- **Single Responsibility**: Each file should address only one aspect or component.
- **Folder Structure**:
  - Organize files into folders, such as:

    ```
    bash
    Copy code
    /css
    ├── base/
    ├── layout/
    ├── components/
    ├── utilities/
    └── themes/
    ```

    - **Use Imports**: Use `@import` or `@use` (for SCSS) to combine styles into one main file.

---

3. Naming Conventions

- **Class Names**: Use meaningful, descriptive names.
    - Use lowercase and hyphenated names (kebab-case), e.g., `.primary-button`.
    - Avoid abbreviations, unless commonly understood (e.g., `.nav`, `.btn`).
    - **BEM (Block-Element-Modifier)**: A popular convention for structuring names:
        - **Block**: The main component (e.g., `.card`).
        - **Element**: A part of the block (e.g., `.card__title`).
        - **Modifier**: A variation or state of the block (e.g., `.card--highlighted`).

---

4. Formatting and Syntax

- **Indentation**: Use 2 or 4 spaces (not tabs) for consistency.
- **Line Length**: Limit to 80-100 characters per line for readability.
- **Spacing**:
    - Include one line between selectors and properties for readability.
    - Keep one space after the colon in property-value pairs.
- **Selector Order**: List classes before IDs and elements in selectors.
    - Example:

```
css
Copy code
.class, #id, element {
    color: blue;
}
```

---

5. Selectors

- **Avoid Overly Specific Selectors**: Keep selectors as simple as possible.
- **Limit Depth**: Don't nest selectors too deeply (recommended limit: 3 levels).
- **Avoid Tag Selectors for Reusable Components**: Use classes instead, as they are less specific.

---

6. Property Declaration Order

- Group properties by type, such as:
    1. **Positioning**: `position`, `top`, `left`, etc.
    2. **Box Model**: `width`, `height`, `padding`, `margin`, etc.
    3. **Typography**: `font`, `color`, `text-align`, etc.
    4. **Visual**: `background`, `border`, etc.
    5. **Animation**: `transition`, `transform`, etc.

Example:

```
css
Copy code
.box {
    /* Positioning */
    position: absolute;
    top: 0;

    /* Box Model */
    width: 100px;
    padding: 10px;

    /* Typography */
    font-size: 16px;
    color: #333;

    /* Visual */
    background-color: #f0f0f0;
    border: 1px solid #ddd;
}
```

---

7. Comments

- **Comment Sections**: Use comments to label sections within files.
- **Explain Complex Code**: Leave comments explaining non-obvious code.
- **Comment Formatting**: Use consistent format:

```
css
Copy code
/* === Layout Styles === */
```

---

8. Responsive Design

- **Media Queries**: Place media queries at the end of the respective selector block.
- **Mobile First**: Begin with the smallest screen sizes, then add breakpoints for larger screens.
- **Variable Breakpoints**: Use variables for breakpoints for consistency and readability.

---

9. CSS Variables

- Define colors, font sizes, and spacing using CSS variables for easier updates.
- Example:

```
css
Copy code
:root {
    --primary-color: #007bff;
    --font-size: 16px;
}

.button {
    background-color: var(--primary-color);
    font-size: var(--font-size);
}
```

---

10. Preprocessors and Tools

- **Use of SCSS or SASS**: Preprocessors like SASS allow nesting, variables, and mixins, improving modularity.
- **Autoprefixer**: Use autoprefixers to add browser-specific prefixes for better compatibility.
- **Linters**: Run linters, such as Stylelint, to enforce consistency and flag errors.

6. CHAPTER TESTING AND INTEGRATION

6.1 TEST CASE DESCRIPTION

Test Case 1: Register a New Student

Objective: Verify that the system can successfully register a new student by capturing their image, storing student details, and saving them in the students' database.

Preconditions:

- Ensure the camera is connected and accessible.
- The `known_faces` directory and `students_info.xlsx` file exist.

Test Steps:

1. Launch the application and select the **Register Student** option.
2. Enter the student's name, enrollment number, and semester when prompted.
3. Follow the on-screen instructions to position the student in front of the camera.
4. Press 's' to capture and save the image.
5. Check for the confirmation message indicating that the image has been saved.
6. Verify that the student's details are stored in `students_info.xlsx` with the correct name, enrollment number, semester, and image path.

Expected Result:

- The student's image is saved in the `known_faces` directory with the correct file name.
- The student's details are added to `students_info.xlsx` without any data loss or errors.

Test Case 2: Mark Attendance

Objective: Ensure that a student's attendance can be marked accurately by recognizing their face and recording the time and date in the attendance file.

Preconditions:

- The student is already registered in the system with their image and information in `students_info.xlsx`.
- The camera is functional.

Test Steps:

1. Launch the application and select the **Mark Attendance** option.
2. Stand in front of the camera and press 'Space' to capture the image.
3. Wait for the system to process the image and check for a recognition result.
4. Verify that the system recognizes the student's face and displays a message confirming attendance has been marked.

5. Open `attendance.xlsx` and confirm that a new record has been created with the student's name, enrollment number, semester, date, and time.

Expected Result:

- The system successfully matches the face with the registered image.
- Attendance is recorded in `attendance.xlsx` with the correct date and time.

Test Case 3: View Attendance Record for a Student

Objective: Verify that the system can retrieve and display the attendance records for a specific student based on their name and semester.

Preconditions:

- The `attendance.xlsx` file exists and contains attendance records for the student.
- The student's information (name and semester) matches the records in the file.

Test Steps:

1. Launch the application and select the **View Attendance** option.
2. Enter the student's name and semester when prompted.
3. Wait for the system to filter and display the attendance records.
4. Confirm that the displayed records are accurate and match the information in `attendance.xlsx`.

Expected Result:

- The system displays a list of attendance records (dates and times) for the specified student.
- The total attendance count is shown.

Test Case 4: Handle Unrecognized Student Attempting to Mark Attendance

Objective: Ensure the system can handle cases where a person whose face is not registered attempts to mark attendance.

Preconditions:

- An unregistered student (or a person not enrolled in the system) tries to mark attendance.
- The `students_info.xlsx` file is up to date and does not include this person.

Test Steps:

1. Launch the application and select the **Mark Attendance** option.
2. Have the unregistered person stand in front of the camera and press 'Space' to capture the image.
3. Wait for the system to process the image and attempt to match it with registered faces.
4. Check for any feedback message indicating that the student is not recognized.

Expected Result:

- The system fails to match the face and displays a message indicating that the student was not recognized.
- No attendance record is added to `attendance.xlsx`.

Test Case 5: Attempt Registration Without Camera Access

Objective: Verify the system's behavior when the camera is not accessible during student registration.

Preconditions:

- The camera is disconnected or inaccessible.

Test Steps:

1. Launch the application and select the **Register Student** option.
2. Enter the student's name, enrollment number, and semester when prompted.
3. Observe the system's response when it attempts to access the camera.

Expected Result:

- The system detects that the camera is not accessible and displays an error message to the user.
- Registration does not proceed without camera access.

6.2 TYPES OF TESTING

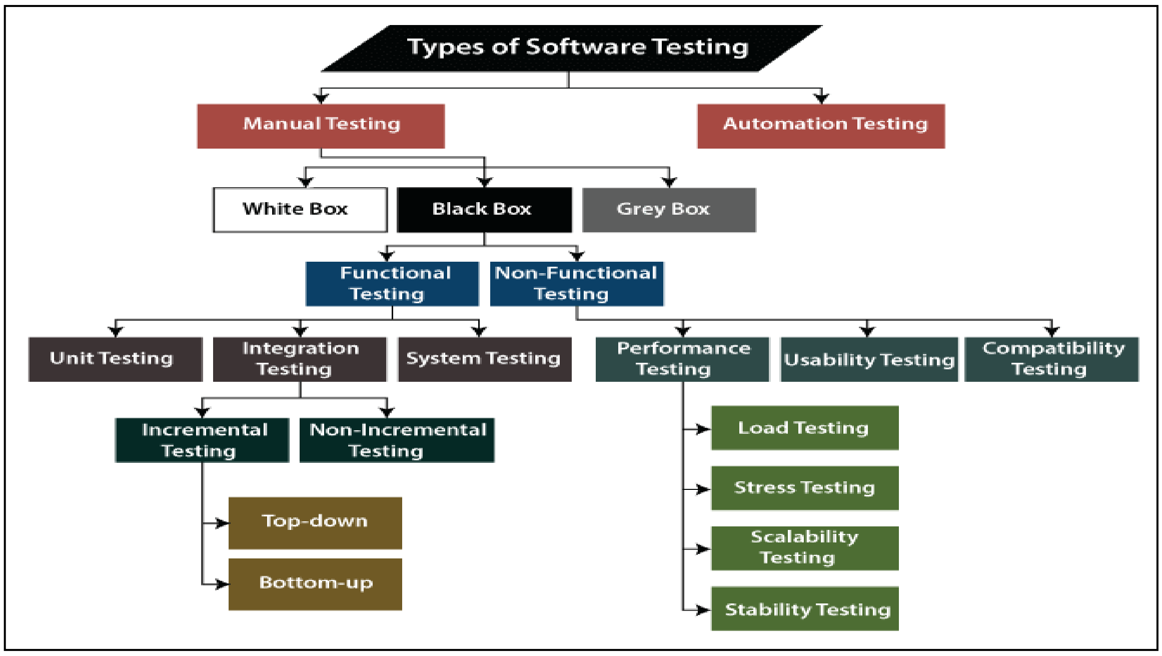


Figure 10

6.3 TEST CASES

Test Case Type	Description	Test Step	Expected Result	Result
Register	This test case describes the process by which a student can register themselves in the system.	Enter **Name : HIBA**, **Enrollment Number:012**, **Semester**. Capture image and save.	Successfully registered; details saved in `students_info.xlsx`	Pass
Mark Attendance	Use case starts when a registered student attempts to mark their attendance.	Capture the student's face using the camera.	System will recognize the student and mark attendance in `attendance.xlsx`	Pass
View Attendance	Use case starts when a user views attendance records for a specific student and semester.	Enter the **student's name : HIBA** and **semester : 5**.	System will display the attendance records for the specified student and semester.	Pass
Track Unrecognized Faces	This test case checks the system's response when an unregistered person attempts to mark attendance.	Unregistered user stands in front of the camera. Capture face.	System will display a message stating "Student not recognized" and will not mark attendance.	Pass
Error Handling - No Camera Access	This test case verifies that the system handles errors when the camera is not accessible.	Attempt to register a student or mark attendance without a connected camera.	System displays an error message indicating that the camera is not accessible.	Pass
Load Student Info	Use case to check if student information loads correctly from `students_info.xlsx`.	Open the system to view student information file.	System will load and display all student information accurately from `students_info.xlsx`.	Pass
Load Attendance Records	Use case to verify that attendance records are loaded and displayed correctly from `attendance.xlsx`.	Open `attendance.xlsx` to view attendance records.	System will display attendance records accurately.	Pass
Exit Application	This test case verifies that the application closes smoothly and without errors.	Select the **Exit** option from the main menu.	Application closes successfully.	Pass

7. RESOURCES, LIMITATION AND FUTURE ENHANCEMENT

Resources

1. **Libraries and Tools**:
    - **OpenCV**: Used for capturing images and handling video inputs.
    - **face_recognition**: A powerful library for face detection and recognition, simplifying the process of encoding and matching faces.
    - **Pandas**: Used for handling and manipulating attendance data, stored in Excel format for easy access and modification.
    - **Excel Files**: To store student information and attendance records, making it accessible for administrators or faculty.
2. **Hardware Requirements**:
    - **Camera**: A working camera (either integrated or external) to capture images and recognize faces in real-time.
    - **Computer System**: The program runs on a system capable of running Python and handling the necessary computational tasks associated with face recognition.

Limitations

1. **Lighting and Environmental Conditions**:
    - Face recognition accuracy can be impacted by poor lighting, camera quality, and environmental factors, which may lead to difficulty in detecting or recognizing faces accurately.
2. **Single Camera Dependency**:
    - The system currently relies on a single camera. If the camera fails or malfunctions, attendance cannot be marked until the camera is operational again.
3. **Limited Scalability**:
    - As the number of registered faces grows, the time required to load and process these faces for comparison also increases. This could slow down the attendance marking process in larger classrooms or organizations.
4. **Privacy Concerns**:
    - Storing images and face encodings may raise privacy issues. Proper permissions and data protection protocols are necessary to ensure compliance with privacy regulations.
5. **Absence of Mobile Integration**:
    - The system is designed for desktop use and relies on an attached camera, which limits flexibility. A mobile-compatible solution would allow more flexibility, particularly for outdoor or large campus environments.

Future Enhancements

1. **Improving Recognition Accuracy**:
    - Implement additional algorithms for pose correction and lighting adjustments to improve the recognition rate. This would allow the system to handle a broader range of lighting conditions and partial face views.
2. **Multi-Camera Support**:

    - Introducing support for multiple cameras could cover larger areas and improve the system’s robustness, especially in classrooms or entrances where multiple people are present at the same time.
3. **Real-Time Notifications and Alerts**:
    - Add functionality to send real-time attendance notifications to students and faculty via email or SMS. This feature could alert students if they were missed or remind them to ensure their presence is recorded.
4. **Automated Data Analysis and Reporting**:
    - Integrate analytical tools to provide attendance statistics and reports automatically. This could include attendance rates, trend analysis, and monthly summaries, helping faculty track student participation over time.
5. **Mobile and Web Application Integration**:
    - Expanding the project to support mobile and web platforms would allow remote access for administrators, faculty, and students. This would make attendance tracking possible outside the classroom, particularly useful for hybrid or remote learning environments.
6. **Enhanced Security and Privacy Features**:
    - Implement encryption for storing facial data and ensure compliance with data protection laws. Access control mechanisms can be added to restrict unauthorized access to sensitive student information.
7. **Cross-Institutional Use and Scalability**:
    - Develop the system to handle a larger database and multiple classes, making it adaptable for use in large institutions or organizations with high student enrollment numbers.

These future enhancements would increase the adaptability and usability of the system, making it more robust, scalable, and user-friendly across a variety of environments.

8. SOURCE CODE

MAIN CODE:

```
      from flask import Flask, render_template, request, redirect, url_for
import os
import cv2
import face_recognition
import pandas as pd
from datetime import datetime

app = Flask(__name__)

# Directories and files
known_faces_dir = "known_faces"
student_info_file = 'students_info.xlsx'
attendance_file = 'attendance.xlsx'

# Ensure directories exist
if not os.path.exists(known_faces_dir):
    os.makedirs(known_faces_dir)

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

@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        user_name = request.form['name']
        enrollment = request.form['enrollment']
        semester = request.form['semester']

        image_path = capture_image(user_name)
        if image_path:
            store_student_info(user_name, enrollment, semester, image_path)
            return redirect(url_for('home'))
    return render_template('register.html')

@app.route('/attendance', methods=['GET', 'POST'])
def attendance():
    if request.method == 'POST':
        student_df = load_student_info()
        if student_df is not None:
            known_faces, known_names = load_known_faces(student_df)  # This will
work now
            image = capture_image_for_attendance()
            if image is not None:
                recognized_data = recognize_face(image, known_faces, student_df)
                if recognized_data:
                    student_name, enrollment, semester = recognized_data
                    mark_attendance(student_name, enrollment, semester)
                    return render_template('success.html', name=student_name)
                else:
                    return render_template('failure.html', message="Student not 
recognized!")
    return render_template('attendance.html')


@app.route('/view', methods=['GET', 'POST'])
def view_attendance():
    if request.method == 'POST':
        student_name = request.form['name'].strip().lower()
        semester = request.form['semester'].strip()

        df = pd.read_excel(attendance_file)
        df['Name'] = df['Name'].str.strip().str.lower()
        df['Semester'] = df['Semester'].astype(str).str.strip()

        filtered_df = df[(df['Name'] == student_name) & (df['Semester'] == 
semester)]
        if filtered_df.empty:
            return render_template('view.html', result=None)
        else:
            return render_template('view.html', 
result=filtered_df.to_dict(orient='records'))
    return render_template('view.html', result=None)

# Helper functions (similar to your Python code)
def capture_image(user_name):
    cap = cv2.VideoCapture(0)
    ret, frame = cap.read()
    if ret:
        image_path = os.path.join(known_faces_dir, f"{user_name}.png")
        cv2.imwrite(image_path, frame)
        cap.release()
        return image_path
    return None

def store_student_info(user_name, enrollment, semester, image_path):
    student_data = {'Name': [user_name], 'Enrollment': [enrollment], 'Semester': 
[semester], 'ImagePath': [image_path]}
    df = pd.DataFrame(student_data)
    if os.path.exists(student_info_file):
        existing_df = pd.read_excel(student_info_file)
        df = pd.concat([existing_df, df], ignore_index=True)
    df.to_excel(student_info_file, index=False)

def load_student_info():
    if os.path.exists(student_info_file):
        return pd.read_excel(student_info_file)
    return None

def load_known_faces(student_df):
    known_faces = []
    known_names = []

    for index, row in student_df.iterrows():
        image_path = row['ImagePath']
        if os.path.exists(image_path):
            image = face_recognition.load_image_file(image_path)
            encodings = face_recognition.face_encodings(image)
            if encodings:  # Ensure there's at least one encoding
                known_faces.append(encodings[0])  # Only take the first encoding
                known_names.append(row['Name'])  # Add the corresponding name

    # Return only two lists: faces and names
    return known_faces, known_names

def capture_image_for_attendance():
    cap = cv2.VideoCapture(0)
    ret, frame = cap.read()
    cap.release()
    return frame if ret else None

def recognize_face(captured_image, known_faces, student_df):
    face_encodings = face_recognition.face_encodings(captured_image)
    if len(face_encodings) == 0:
        return None
    captured_encoding = face_encodings[0]
    matches = face_recognition.compare_faces(known_faces, captured_encoding)
    if True in matches:
        first_match_index = matches.index(True)
        student_name = student_df.iloc[first_match_index]['Name']
        enrollment = student_df.iloc[first_match_index]['Enrollment']
        semester = student_df.iloc[first_match_index]['Semester']
        return student_name, enrollment, semester
    return None

def mark_attendance(student_name, enrollment, semester, file='attendance.xlsx'):
    now = datetime.now()
    current_date = now.strftime("%Y-%m-%d")
    current_time = now.strftime("%H:%M:%S")

    try:
        df = pd.read_excel(file)
    except FileNotFoundError:
        df = pd.DataFrame(columns=["Enrollment", "Name", "Semester", "Date",
"Time"])
```

```
    # New record to be added
    new_record_df = pd.DataFrame({
        "Enrollment": [enrollment],
        "Name": [student_name],
        "Semester": [semester],
        "Date": [current_date],
        "Time": [current_time]
    })

    # Append using pd.concat instead of df.append()
    df = pd.concat([df, new_record_df], ignore_index=True)

    # Save the updated DataFrame back to the Excel file
    df.to_excel(file, index=False)

    print(f"Attendance marked for {student_name} (Enrollment: {enrollment},
Semester: {semester})")


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

TEMPLATES :

Attendance HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Mark Attendance</title>
    <!-- Bootstrap CSS -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-
alpha1/dist/css/bootstrap.min.css" rel="stylesheet">
    <style>
        body {
            background-color: #f8f9fa;
        }
        .container {
            margin-top: 50px;
            padding: 30px;
            background: white;
            border-radius: 10px;
            box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
        }
        h1 {
            color: #343a40;
        }
        .btn-custom {
            background-color: #007bff;
            border-color: #007bff;
        }
        .btn-custom:hover {
            background-color: #0056b3;
            border-color: #0056b3;
        }
    </style>
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
        <div class="container-fluid">
            <a class="navbar-brand" href="/">Attendance Page</a>
            <div class="collapse navbar-collapse" id="navbarNav">
                <ul class="navbar-nav">
                    <li class="nav-item">
                        <a class="nav-link" href="/register">Register</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="/attendance">Mark Attendance</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="/view">View Attendance</a>
                    </li>
                </ul>
            </div>
        </div>
    </nav>
    <div class="container">
        <h1 class="text-center">Mark Attendance</h1>
        <p class="text-center text-muted">Press the button below to mark
attendance.</p>

        <form method="POST" action="/attendance">
            <div class="text-center">
                <button type="submit" class="btn btn-custom btn-lg">Mark
Attendance</button>
            </div>
        </form>
    </div>

    <!-- Bootstrap JS -->
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-
alpha1/dist/js/bootstrap.bundle.min.js"></script>
</body>
</html>
```

Register HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Register Student</title>
    <!-- Bootstrap CSS -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-
alpha1/dist/css/bootstrap.min.css" rel="stylesheet">
    <style>
        body {
            background-color: #f8f9fa;
        }
        .container {
            margin-top: 50px;
            padding: 20px;
            background: white;
            border-radius: 10px;
            box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
        }
    </style>
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
        <div class="container-fluid">
            <a class="navbar-brand" href="/">Attendance Page</a>
            <div class="collapse navbar-collapse" id="navbarNav">
                <ul class="navbar-nav">
                    <li class="nav-item">
                        <a class="nav-link" href="/register">Register</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="/attendance">Mark Attendance</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="/view">View Attendance</a>
                    </li>
                </ul>
            </div>
        </div>
    </nav>
    <div class="container">
        <h1 class="text-center">Register Student</h1>
        <p class="text-center text-muted">Please fill out the form below to
register a new student.</p>

        <form method="POST" action="/register">
            <div class="mb-3">
```

```
                <label for="name" class="form-label">Name</label>
                <input type="text" class="form-control" id="name" name="name"
required>
            </div>
            <div class="mb-3">
                <label for="enrollment" class="form-label">Enrollment
Number</label>
                <input type="text" class="form-control" id="enrollment"
name="enrollment" required>
            </div>
            <div class="mb-3">
                <label for="semester" class="form-label">Semester</label>
                <input type="text" class="form-control" id="semester"
name="semester" required>
            </div>
            <div class="text-center">
                <button type="submit" class="btn btn-primary">Register</button>
            </div>
        </form>
    </div>

    <!-- Bootstrap JS -->
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-
alpha1/dist/js/bootstrap.bundle.min.js"></script>
</body>
</html>
```

View HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>View Attendance</title>
    <!-- Bootstrap CSS -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-
alpha1/dist/css/bootstrap.min.css" rel="stylesheet">
    <style>
        body {
            background-color: #f8f9fa;
        }
        .container {
            margin-top: 50px;
            padding: 30px;
            background: white;
            border-radius: 10px;
            box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
        }
        h1 {
            color: #343a40;
        }
        .btn-custom {
            background-color: #007bff;
            border-color: #007bff;
        }
        .btn-custom:hover {
            background-color: #0056b3;
            border-color: #0056b3;
        }
        .result-list {
            margin-top: 20px;
        }
    </style>
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
        <div class="container-fluid">
            <a class="navbar-brand" href="/">Attendance Page</a>
            <div class="collapse navbar-collapse" id="navbarNav">
                <ul class="navbar-nav">
                    <li class="nav-item">
                        <a class="nav-link" href="/register">Register</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="/attendance">Mark Attendance</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="/view">View Attendance</a>
                    </li>
                </ul>
            </div>
        </div>
    </nav>
    <div class="container">
        <h1 class="text-center">View Attendance</h1>
        <form method="POST" action="/view" class="text-center">
            <div class="mb-3">
                <label for="name" class="form-label">Name</label>
                <input type="text" class="form-control" name="name" required>
            </div>
            <div class="mb-3">
                <label for="semester" class="form-label">Semester</label>
                <input type="text" class="form-control" name="semester" required>
            </div>
            <button type="submit" class="btn btn-custom btn-lg">View
Attendance</button>
        </form>
        {% if result %}
            <h2 class="mt-4">Attendance Records:</h2>
            <ul class="list-group result-list">
                {% for record in result %}

                    <li class="list-group-item">{{ record['Name']}}: {{
record['Date'] }} - {{ record['Time'] }}</li>
                {% endfor %}
            </ul>
        {% elif result is none %}
            <p class="mt-4 text-muted">No attendance records found.</p>
        {% endif %}
    </div>

    <!-- Bootstrap JS -->
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-
alpha1/dist/js/bootstrap.bundle.min.js"></script>
</body>
</html>
```

Index HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Welcome to Attendance Page</title>
    <!-- Bootstrap CSS -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-
alpha1/dist/css/bootstrap.min.css" rel="stylesheet">
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
        <div class="container-fluid">
            <a class="navbar-brand" href="#">Attendance Page</a>
            <div class="collapse navbar-collapse" id="navbarNav">
                <ul class="navbar-nav">
                    <li class="nav-item">
                        <a class="nav-link" href="/register">Register</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="/attendance">Mark Attendance</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="/view">View Attendance</a>
                    </li>
                </ul>
            </div>
        </div>
    </nav>

    <div class="container text-center mt-5">
        <h1>Welcome to the Attendance Management System</h1>
        <p class="lead">Manage attendance effortlessly with our easy-to-use
system.</p>
        <div class="mt-5">
            <a href="/register" class="btn btn-primary btn-lg">Register a New
Student</a>
            <a href="/attendance" class="btn btn-success btn-lg">Mark
Attendance</a>
            <a href="/view" class="btn btn-info btn-lg">View Attendance</a>
        </div>
    </div>

    <!-- Bootstrap JS -->
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-
alpha1/dist/js/bootstrap.bundle.min.js"></script>
</body>
</html>
```

Failure HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Failure Page</title>
    <!-- Bootstrap CSS -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-
alpha1/dist/css/bootstrap.min.css" rel="stylesheet">
    <style>
        body {
            background-color: #e9ecef;
        }
        .container {
            margin-top: 100px;
            padding: 40px;
            background: white;
            border-radius: 10px;
            box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
        }
        h1 {
            color: rgb(230, 95, 95);
        }
        .btn-custom {
            background-color: #97c6f9;
            border-color: #afcff1;
        }
        .btn-custom:hover {
            background-color: #0056b3;
            border-color: #0056b3;
        }
    </style>
</head>
<body>
    <div class="container text-center">
        <h1>Face not Found!</h1>
        <p class="lead">Please Register Yourself First.</p>
        <a href="/" class="btn btn-custom btn-lg">Back to Home</a>
    </div>

    <!-- Bootstrap JS -->
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-
alpha1/dist/js/bootstrap.bundle.min.js"></script>
</body>
</html>
```

9. CONCLUSION

The project that was done will accomplish the following benefits:

- Maintains Overall Records
- Get rid of pen and paper system
- Proxy attendance is eliminated
- Saves time
- Less mistakes
- Virtual classroom

This Face Recognition Attendance System successfully integrates face recognition technology to automate the process of recording attendance in an efficient and reliable manner. By leveraging Python libraries such as OpenCV and face_recognition, we've created a user-friendly interface that simplifies student registration, attendance marking, and attendance review. The system not only minimizes human error but also enhances security and accuracy by ensuring that only registered students are marked present.

Incorporating features like real-time image capture and automatic attendance logging into Excel allows for seamless data management, making it easy to track and analyze attendance records over time. This approach not only saves administrative effort but also supports transparent record-keeping, benefiting both students and faculty.

With further enhancements, such as adding real-time notifications, improving the accuracy of recognition under various conditions, and generating analytical reports, this system can be an indispensable tool in educational environments. Overall, this project demonstrates the potential of combining facial recognition with data management to create a practical, scalable solution for attendance tracking.

10. REFERENCE

1. https://towardsdatascience.com/face- recognition-how-lbph-works-90ec258c3d6b
2. https://www.pyimagesearch.com/2018/07/19/op encv-tutorial-a-guide-to-learn-opencv/
3. https://www.geeksforgeeks.org/working-csv- files-python/
4. https://www.tutorialspoint.com/python/os_file_ methods.htm
5. https://becominghuman.ai/face-detection-using- opencv-with-haar-cascade-classifiers-941dbb25177
6. https://ieeexplore.ieee.org/document/7916753/
7. https://www.semanticscholar.org/paper/Automa tic-Attendance-System-Using-Facial-Choudhary- Tripathi/fcc208fbf92faba8705d3fe89cb4bcfafb0 97c57
8. https://www.researchgate.net/publication/32767 1423_Automated_Student_Attendance_Manage ment_System_Using_Face_Recognition
9. https://www.youtube.com/watch?v=helH- oa2Tw4
10. https://www.youtube.com/watch?v=FeNasBaXd hg
11. https://www.slideshare.net/Nikyaa7/automatic- attendance-system-using-facial-recognition
12. https://StarUML.io