object detection from images

CS-412 Visual Programming

A robot typing on a computer

Description automatically generated with low confidence

Submitted to:

Sir Ahsan Raza Satar

**Team Members Profile**

Name:

Muhammad Qasim



Reg no:

2021-ag-7873

Degree:

Bs computer science

Semester:

4th

Section:

A

Role:

Facial Expression Recognition, Attendance System , GUI

Name:



Wali Arshad

Reg no:

2021-ag-7739

Degree:

Bs computer science

Semester:

4th

Section:

A

Role:

Documentation

Name:

Ayaan Ali



Reg no:

2021-ag-7810

Degree:

Bs computer science

Semester:

4th

Section:

A

Role:

Database Manager

Contents

[Team overview: 4](#_Toc137631940)

[ABSTRACT: 4](#_Toc137631941)

[Methodology: 5](#_Toc137631942)

[1-Project Management: 5](#_Toc137631943)

[2-Study and Analysis: 5](#_Toc137631944)

[3-Preprocessing and Data Collection: 5](#_Toc137631945)

[4-Module 1: Mood Recognition: 5](#_Toc137631946)

[5-Module 2: Database Administration: 5](#_Toc137631947)

[9-Integration and testing of the system: 5](#_Toc137631948)

[10-Deployment and documentation: 6](#_Toc137631949)

[Code overview: 6](#_Toc137631950)

[1-Module for Detecting Mood: 6](#_Toc137631951)

[2-Module for Database Management: 6](#_Toc137631952)

[Requirements: 7](#_Toc137631953)

[1-Hardware Prerequisites: 7](#_Toc137631954)

[2-Software Prerequisites: 7](#_Toc137631955)

[3-Dataset: 7](#_Toc137631956)

[4-Libraries and Dependencies: 8](#_Toc137631957)

[5-Documentation: 8](#_Toc137631958)

[6-Validation and testing: 8](#_Toc137631959)

[7-Additional Conditions: 8](#_Toc137631960)

[Specifications: 8](#_Toc137631961)

[Module 1: Mood Recognition 8](#_Toc137631962)

[Module 2: Database Storage 8](#_Toc137631963)

[Module 3: Integration and User Interface 9](#_Toc137631964)

[Use Cases: 9](#_Toc137631965)

[Use Case 1: Detect Mood 9](#_Toc137631966)

[1-Actors: User 9](#_Toc137631967)

[2-Use Case: Save Data 9](#_Toc137631968)

[Data Flow Diagram: 10](#_Toc137631969)

[Algorithm: 10](#_Toc137631970)

[Module 1: Mood Detection 10](#_Toc137631971)

[Module 2: Database Storage 10](#_Toc137631972)

[Module 3: Integration and User Interface 10](#_Toc137631973)

[User guideline: 11](#_Toc137631974)

[1-System Requirements: 11](#_Toc137631975)

[2-Tool Setup: 11](#_Toc137631976)

[3-Launching the Tool: 11](#_Toc137631977)

[4-User Interface: 11](#_Toc137631978)

[5-Selecting an Image File: 12](#_Toc137631979)

[6-Displaying the Mood Result: 12](#_Toc137631980)

[7-Saving the Mood Data: 12](#_Toc137631981)

[8-Viewing Stored Mood Data: 12](#_Toc137631982)

[9-Exiting the Tool: 12](#_Toc137631983)

[Actual code : 12](#_Toc137631984)

[Appendix: Mood Detection Tool 22](#_Toc137631985)

[System Requirements: 22](#_Toc137631986)

[Tool Structure: 22](#_Toc137631987)

[Installation Instructions: 22](#_Toc137631988)

[Usage Instructions: 22](#_Toc137631989)

[Troubleshooting: 23](#_Toc137631990)

[Limitations: 23](#_Toc137631991)

[Future Improvements: 23](#_Toc137631992)

[Contributors: 23](#_Toc137631993)

[References: 23](#_Toc137631994)

[GUI 24](#_Toc137631995)

# Team overview:

Welcome to our Visual Programming project team profile! We are a committed team made up of Qasim, Ayaan, and Wali who are all passionate about learning about the intriguing field of visual programming and its many applications.

Our team is dedicated to using state-of-the-art tools to identify and analyze items in photos. With experience in software development, computer vision, and machine learning, our team is motivated to produce cutting-edge solutions that close the gap between visual data and useful insights.

Our team's success is mostly a result of collaboration. Ayaan's experience in machine learning algorithms, Wali's in-depth knowledge of image processing methods, and Qasim's prowess in software development and user interface design all contribute to the team. Together, we make a well-rounded team that is able to take on difficult problems and provide solid solutions.

Our study aims to create sophisticated algorithms and systems that can precisely recognize and categorize items in photos. We work to increase the accuracy and effectiveness of our detection models through research, experimentation, and an iterative development approach.

# ABSTRACT:

The proposed project seeks to create a system comprised of three Python modules that identify a person's mood, save the recorded data in a database, and give a full solution for mood analysis. The first module analyzes facial expressions in photos to detect mood, specifically discriminating between pleasure and melancholy.

This lesson extracts relevant information from facial photographs and classifies them using image processing techniques and machine learning algorithms. The second module is in charge of storing the recorded data in a database, ensuring secure and dependable storage of mood-related data. A suitable relational database management system (DBMS), such as MySQL or SQLite, can be used to design the database.

The last module connects the previous modules, creating a unified workflow. It allows users to enter an image or take one with a camera, process the image to detect the person's mood, and then save the pertinent information in a database. The system provides an easy-to-use interface for interaction, allowing for the easy input and retrieval of mood data.

Overall, this project combines image processing, machine learning, and database management techniques to develop a strong solution for mood detection and data storage, providing valuable insights into mood analysis and potentially finding applications in fields such as mental health monitoring, market research, and social analysis.

# Methodology:

## 1-Project Management:

* Set project objectives and goals.
* Determine the project's scope and its deliverables.
* Establish a schedule and allot resources appropriately.
* Identify the team members' roles and duties.

## 2-Study and Analysis:

* Perform thorough literature research on strategies for database administration, and face expression analysis.
* Investigate Python modules and frameworks that fit the project's needs.
* Examine each module's technological viability and any potential difficulties.

## 3-Preprocessing and Data Collection:

* Collect a broad variety of facial photos with happy and sorrowful expressions.
* Preprocessing the dataset includes operations like image cropping, scaling, and normalization.
* Label the photographs with the appropriate mood (happy or sad).

## 4-Module 1: Mood Recognition:

* Create a facial recognition algorithm with Python modules such as OpenCV and dlib.
* Using the labeled dataset, train a machine learning model (such as Convolutional Neural Networks).
* Fine-tune the model to correctly classify happy and sad facial emotions.
* Utilize proper evaluation metrics to validate the model's performance.

## 5-Module 2: Database Administration:

* Choose an appropriate database management system (e.g., MySQL, PostgreSQL) to store the collected data.
* Create a database structure to store mood detection findings and related data.
* Create Python code to connect to the database and store data in a systematic manner.
* Implement data retrieval and management functions that are both secure and efficient.

## 9-Integration and testing of the system:

* Combine all three parts into a unified system.
* Test the system's operation and performance, making sure that each module works well with the others.
* Conduct extensive testing to evaluate the mood detection, data storage, and uniform verification features' accuracy.
* Collect user feedback and make appropriate changes based on their suggestions.

## 10-Deployment and documentation:

* Complete documentation, including system architecture, module specifications, and usage recommendations, should be prepared.
* Package the project code and dependencies for simple deployment across several environments.
* Check that the system can be deployed on the target platform, taking into account hardware and software requirements.
* Provide post-deployment user assistance and maintenance to address any issues that may emerge.

# Code overview:

This project's code is divided into modules that correspond to the three primary functionalities: mood detection, database management, and uniform verification. Let us take a quick look at each module:

## 1-Module for Detecting Mood:

* This module uses face recognition algorithms and machine learning techniques to determine a person's mood based on facial expressions.
* For face identification and extraction, it makes use of Python modules such as OpenCV and dlib.
* On a labeled dataset of happy and sad facial photos, a machine learning model, such as Convolutional Neural Networks (CNN), is trained.
* The trained model is then applied to real-time or input image classification of face emotions.
* The code provides functions for image preparation, training the model, and mood recognition.

## 2-Module for Database Management:

* This module is concerned with storing and maintaining the collected data, which includes mood detection findings and pertinent information.
* For data storage, a suitable database management system, such as MySQL or PostgreSQL, is chosen.
* The code connects to the database by utilizing Python libraries or modules relevant to the database system of choice.
* In order to insert and retrieve data from the database in a systematic manner, functions are implemented.
* To protect the integrity and confidentiality of the stored data, data validation and security methods may be implemented.

Aside from these individual modules, the general codebase may include shared utility functions, configuration files, and primary execution scripts that coordinate the functionality of several modules.

Python, a versatile and popular programming language for machine learning, computer vision, and database operations, was used to create the code. Python's rich libraries and frameworks provide the tools required to properly implement the project's needs.

To increase readability and maintainability, the code is documented using best practices. It may contain inline comments, docstrings, and README files to help developers and users navigate the source and its features.

Overall, the codebase shows a modular and systematic approach to implementing the project's capabilities, ensuring flexibility, reusability, and ease of maintenance.

# Requirements:

## 1-Hardware Prerequisites:

* Python and the associated libraries must be installed on a computer or device.
* If you intend to capture live video for mood detection and uniform verification, use a webcam or camera.

## 2-Software Prerequisites:

* Python programming language (compatibility with project dependencies).
* PyCharm, Visual Studio Code, or Jupyter Notebook are examples of integrated development environments (IDEs).
* Python frameworks and libraries:
* OpenCV: image and video processing library.
* dlib: for extracting facial features.
* TensorFlow or PyTorch: for deep learning and machine learning.
* Database system library (for example, mysql-connector-python for MySQL).
* Other libraries required for supporting functionalities.

## 3-Dataset:

* For training the mood detection algorithm, a labeled dataset of happy and sad facial photos was used.
* Dataset containing whole and incomplete uniform images for training the uniform verification model.
* For accurate detection and verification, a broad and representative dataset is required.

## 4-Libraries and Dependencies:

* To manage dependencies, use a virtual environment manager such as Anaconda or pipenv.
* Keep a requirements.txt or an environment.yml file that lists all of the required libraries and their versions.
* Install the essential libraries with your preferred package manager.

+-

## 5-Documentation:

* Create documentation or a README file that explains the project's goal, functionality, and setup and usage instructions.
* For better comprehension and maintainability, document the codebase with inline comments, docstrings, and any additional explanations.

## 6-Validation and testing:

* Create test cases and execute unit testing to ensure that individual modules and functionality are correct.
* Validate the mood detection and uniform verification models' accuracy and performance using appropriate evaluation metrics and approaches.

## 7-Additional Conditions:

* If applicable, access to a database for storing and retrieving records.
* For testing and demonstration reasons, image and video samples are provided.

# Specifications:

## Module 1: Mood Recognition

* For mood analysis, the module should use an input image as a parameter.
* Image processing techniques should be used to detect and extract facial features from the input image.
* Based on the retrieved facial features, use machine learning methods such as Convolutional Neural Networks (CNNs) or Support Vector Machines (SVMs) to identify the mood as joyful or sad.
* Ascertain that the module produces accurate and dependable mood detection results.
* Handle numerous face expressions and lighting circumstances to improve the algorithm's robustness.

## Module 2: Database Storage

* To save mood-related data, the module should connect to a relational database management system (e.g., MySQL, SQLite).
* Create a database schema to record the mood detection findings, containing attributes such as image ID, mood label (happy/sad), timestamp, and any other relevant information.
* Implement functions to store studied mood data in the database while maintaining data integrity and confidentiality.
* Handle any problems or exceptions that may arise during database operations by providing proper error handling techniques.

## Module 3: Integration and User Interface

* Create an easy-to-use interface that allows users to engage with the system and give input images for mood detection.
* Integrate the mood detection and database storage modules to create a unified workflow.
* Allow users to submit an image file or capture an image with a camera for mood analysis.
* After the analysis, display the detected mood (happy or sad) to the user.
* Create a user interface that allows users to see and retrieve mood data recorded in the database.
* For ease of use, ensure a visually appealing and intuitive user interface design.

# Use Cases:

## Use Case 1: Detect Mood

## 1-Actors: User

* **Preconditions:** The User is interacting with the Mood Detection module.
* **Flow of Events:** The User expresses emotions, the system processes these emotions, detects the mood, and produces Mood Data.
* **Postconditions:** The Mood Data is ready to be saved in the database.

## 2-Use Case: Save Data

* **Actors:** Mood Detection module
* **Preconditions:** Mood Data is available from the Mood Detection module.
* **Flow of Events:** The Mood Detection module passes the Mood Data to the Data Saving module, which then processes and stores the Mood Data in the database.
* **Postconditions:** The Mood Data is stored in the database for future reference

# Data Flow Diagram:

A diagram of a flowchart

Description automatically generated with low confidence

# Algorithm:

## Module 1: Mood Detection

1. Load the input image for mood analysis.
2. Apply image processing techniques to detect and extract facial features from the image.
3. Preprocess the extracted facial features (e.g., resize, normalize) to prepare them for mood classification.
4. Feed the preprocessed features into a machine learning model (e.g., CNN, SVM) trained on a labeled dataset of happy and sad facial expressions.
5. Obtain the predicted mood label (happy or sad) from the model.
6. Return the predicted mood label as the output of the module.

## Module 2: Database Storage

1. Establish a connection with the chosen relational database management system (e.g., MySQL, SQLite).
2. Define the database schema to store mood-related data, including fields like image ID, mood label, timestamp, etc.
3. Implement a function to save the mood detection results in the database, taking the relevant information as input (e.g., image ID, mood label, timestamp).
4. Connect to the database and execute the necessary SQL queries to insert the mood data into the appropriate table.
5. Handle any errors or exceptions that may occur during the database operations and provide appropriate error handling.

## Module 3: Integration and User Interface

1. Display the user interface to interact with the system and provide input images for mood detection.
2. If the user selects to capture an image:
   * Capture an image using the camera.
   * Invoke the Mood Detection module by passing the captured image.
3. If the user selects to input an image file:
   * Allow the user to browse and select an image file.
   * Invoke the Mood Detection module by passing the selected image file.
4. Retrieve the predicted mood label from the Mood Detection module.
5. Display the predicted mood label to the user.
6. Connect to the database.
7. Save the mood detection results in the database by executing the necessary SQL queries with the relevant information (e.g., image ID, mood label, timestamp).
8. Handle any errors or exceptions that may occur during the database operations and provide appropriate error handling.
9. If the user wants to view stored mood data:
   * Retrieve the mood data from the database by executing the necessary SQL queries.
   * Display the retrieved mood data to the user.

# User guideline:

Welcome to the Mood Detection Tool! This guide will walk you through the steps to effectively use the tool and analyze the mood of a person. Please follow the instructions below:

## 1-System Requirements:

* Make sure your system meets the following requirements:
* Python installed (version X.X or higher).
* Required libraries and dependencies installed (e.g., OpenCV, TensorFlow, SQLAlchemy).

## 2-Tool Setup:

* Download and install the Mood Detection Tool from the provided source.
* Install any necessary dependencies or libraries mentioned in the project documentation.

## 3-Launching the Tool:

* Open the command prompt or terminal and navigate to the directory where the Mood Detection Tool is located.
* Run the tool using the appropriate command (e.g., python mood\_detection\_tool.py).

## 4-User Interface:

* Once the tool is launched, a user interface will be displayed.
* The user interface will provide options for capturing an image or selecting an image file for mood analysis.
* Choose the desired option to proceed.

## 5-Selecting an Image File:

* If you select the "Select Image File" option:
* Click on the "Browse" button to choose an image file from your system.
* Navigate to the location where the desired image is stored and select it.
* The tool will process the selected image to determine the person's mood.

## 6-Displaying the Mood Result:

* After the image is processed, the tool will display the detected mood (happy or sad) on the user interface.
* Take note of the displayed mood result for further analysis or record-keeping.

## 7-Saving the Mood Data:

* The tool provides an option to save the mood detection results in a database.
* Click on the "Save" button to store the mood-related information, including the image ID, mood label, and timestamp.
* The mood data will be securely saved in the database for future reference.

## 8-Viewing Stored Mood Data:

* If you wish to view previously stored mood data:
* Click on the "View Data" button to access the stored mood information in the database.
* The tool will display the retrieved mood data, allowing you to review and analyze it.

## 9-Exiting the Tool:

* Once you have finished using the Mood Detection Tool, click on the "Exit" or "Close" button to exit the application.

# Actual code :

import sys ,  datetime

from PyQt5 import QtCore

from PyQt5.QtGui import QIcon, QPixmap, QFont

from PyQt5 import QtGui

from PyQt5.QtWidgets import QApplication, QMainWindow, QWidget, QVBoxLayout, QHBoxLayout, QLabel, QLineEdit, QPushButton, QTableWidget, QTableWidgetItem

from PyQt5.QtWidgets import QStackedWidget

from PyQt5.QtWidgets import QFileDialog

from PyQt5.QtWidgets import QApplication, QMainWindow, QWidget, QVBoxLayout, QLabel, QLineEdit, QPushButton, QSizePolicy

import sqlite3

class SideNavMenu(QMainWindow):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.setWindowTitle("Side Navbar Menu")

        self.setGeometry(100, 100, 900, 700)

        self.sidebar = QWidget(self)

        self.sidebar.setGeometry(0, 0, 200, self.height())

        self.sidebar.setStyleSheet("background-color: #232931;")

        self.layout = QVBoxLayout(self.sidebar)

        self.layout.setContentsMargins(0, 0, 0, 0)

        self.add\_button("Records")

        self.add\_button("Portal")

        self.add\_button("Attendance")

        self.central\_widget = QStackedWidget(self)

        self.central\_widget.setGeometry(200, 0, self.width() - 200, self.height())

        self.page\_home = QWidget()

        self.page\_form = QWidget()

        self.page\_about = QWidget()

        self.setup\_home\_page()

        self.setup\_form\_page()

        self.setup\_about\_page()

        self.central\_widget.addWidget(self.page\_home)

        self.central\_widget.addWidget(self.page\_form)

        self.central\_widget.addWidget(self.page\_about)

        self.show()

    def add\_button(self, text):

        button = QPushButton(text, self.sidebar)

        button.setStyleSheet(

            """

            QPushButton {

                background-color: #232931;

                color: white;

                border: none;

                padding-top: 10px;

                padding-bottom: 10px;

                padding-left: 15px;

                text-align: center;

                font-size: 20px;

            }

            QPushButton:hover {

                background-color: #393e46;

            }

            """

        )

        button.clicked.connect(lambda: self.handle\_button\_click(text))

        self.layout.addWidget(button)

    def handle\_button\_click(self, button\_text):

        if button\_text == "Records":

            self.central\_widget.setCurrentWidget(self.page\_home)

        elif button\_text == "Portal":

            self.central\_widget.setCurrentWidget(self.page\_form)

        elif button\_text == "Attendance":

            self.central\_widget.setCurrentWidget(self.page\_about)

    def setup\_home\_page(self):

        layout = QVBoxLayout(self.page\_home)

        layout.setAlignment(QtCore.Qt.AlignTop)

        # Create the heading

        heading\_label = QLabel("Students Record")

        heading\_label.setStyleSheet("font-size: 20px; font-weight: bold;")

        layout.addWidget(heading\_label)

        # Create the search option

        search\_layout = QHBoxLayout()

        search\_label = QLabel("Search:")

        search\_label.setStyleSheet("font-size: 16px; margin-right: 5px;")

        self.search\_input\_students = QLineEdit()

        self.search\_input\_students.setStyleSheet("font-size: 16px; padding: 5px;")

        search\_button = QPushButton("Search")

        search\_button.setStyleSheet(

            """

            QPushButton {

                padding: 5px 10px;

                background-color: #4CAF50;

                color: white;

                font-size: 16px;

                border: none;

            }

            QPushButton:hover {

                background-color: #45a049;

            }

            """

        )

        refresh\_button = QPushButton("Refresh")

        refresh\_button.setStyleSheet(

            """

            QPushButton {

                padding: 5px 10px;

                background-color: #f44336;

                color: white;

                font-size: 16px;

                border: none;

            }

            QPushButton:hover {

                background-color: #d32f2f;

            }

            """

        )

        search\_layout.addWidget(search\_label)

        search\_layout.addWidget(self.search\_input\_students)

        search\_layout.addWidget(search\_button)

        search\_layout.addWidget(refresh\_button)

        layout.addLayout(search\_layout)

        # Create the table to display student records

        self.table\_students = QTableWidget()

        self.table\_students.setColumnCount(6)

        self.table\_students.setHorizontalHeaderLabels(["Image", "ID No", "Name", "Class", "Contact Info", "Address"])

        # Populate the table with data

        self.update\_student\_table()

        layout.addWidget(self.table\_students)

        # Connect the search button to the search functionality

        search\_button.clicked.connect(self.search\_students)

        refresh\_button.clicked.connect(self.refresh\_student\_table)

    def extract\_student\_record1(self):

        # Connect to the SQLite database

        conn = sqlite3.connect('example.db')

        cursor = conn.cursor()

        # Execute the query to retrieve student records

        cursor.execute("SELECT \* FROM students\_record")

        student\_records = cursor.fetchall()

        # Close the connection

        conn.close()

        return student\_records

    def update\_student\_table(self, student\_records=None):

        # Clear the table

        self.table\_students.clearContents()

        # Get the student records if not provided

        if student\_records is None:

            student\_records = self.extract\_student\_record1()

        # Set the number of rows in the table

        self.table\_students.setRowCount(len(student\_records))

        # Define the desired width and height for the image

        medium\_width = 100

        medium\_height = 100

        # Populate the table with data

        for row, record in enumerate(student\_records):

            for column, value in enumerate(record):

                if column == 0:

                    # Display the image in the table cell (first column)

                    image\_path = value

# Assuming the image path is stored in the first column

                    pixmap = QtGui.QPixmap(image\_path).scaled(medium\_width, medium\_height, QtCore.Qt.AspectRatioMode.KeepAspectRatio)

                    label = QLabel()

                    label.setPixmap(pixmap)

                    label.setScaledContents(True)

                    self.table\_students.setCellWidget(row, column, label)

                else:

                    # Display the other data in the table cells

                    item = QTableWidgetItem(str(value))

                    self.table\_students.setItem(row, column, item)

        # Set the row height

        self.table\_students.verticalHeader().setDefaultSectionSize(medium\_height)

    def search\_students(self):

        # Get the search query

        query = self.search\_input\_students.text()

        # Retrieve all student records

        student\_records = self.extract\_student\_record1()

        # Filter the records based on the search query

        filtered\_records = []

        for record in student\_records:

            if query.lower() in str(record).lower():

                filtered\_records.append(record)

        # Update the table with the filtered records

        print(filtered\_records)

        self.update\_student\_table(filtered\_records)

    def refresh\_student\_table(self):

        # Clear the search input

        self.search\_input\_students.clear()

        # Retrieve all student records

        student\_records = self.extract\_student\_record1()

        # Update the table with all student records

        self.update\_student\_table(student\_records)

    def setup\_about\_page(self):

        layout = QVBoxLayout(self.page\_about)

        layout.setAlignment(QtCore.Qt.AlignTop)

        # Create the heading

        heading\_label = QLabel("Attendance Record")

        heading\_label.setStyleSheet("font-size: 20px; font-weight: bold;")

        layout.addWidget(heading\_label)

        # Create the search option

        search\_layout = QHBoxLayout()

        search\_label = QLabel("Search:")

        search\_label.setStyleSheet("font-size: 16px; margin-right: 5px;")

        self.search\_input = QLineEdit()

        self.search\_input.setStyleSheet("font-size: 16px; padding: 5px;")

        search\_button = QPushButton("Search")

        search\_button.setStyleSheet(

            """

            QPushButton {

                padding: 5px 10px;

                background-color: #4CAF50;

                color: white;

                font-size: 16px;

                border: none;

            }

            QPushButton:hover {

                background-color: #45a049;

            }

            """

        )

        refresh\_button = QPushButton("Refresh")

        refresh\_button.setStyleSheet(

            """

            QPushButton {

                padding: 5px 10px;

                background-color: #f44336;

                color: white;

                font-size: 16px;

                border: none;

            }

            QPushButton:hover {

                background-color: #d32f2f;

            }

            """

        )

        search\_layout.addWidget(search\_label)

        search\_layout.addWidget(self.search\_input)

        search\_layout.addWidget(search\_button)

        search\_layout.addWidget(refresh\_button)

        layout.addLayout(search\_layout)

        # Create the table to display record details

        self.table = QTableWidget()

        self.table.setColumnCount(5)

        self.table.setHorizontalHeaderLabels([ "ID No", "Time", "Name", "Expression", "Uniform"])

        # Example data (replace with your actual data)

        self.records =extract\_attendance\_records()

        # Populate the table with data

        self.update\_table()

        layout.addWidget(self.table)

        # Connect the search button to the search functionality

        search\_button.clicked.connect(self.search\_records)

        refresh\_button.clicked.connect(self.refresh\_table)

    def update\_table(self):

        self.table.setRowCount(len(self.records))

        for row, record in enumerate(self.records):

            for col, data in enumerate(record):

                self.table.setItem(row, col, QTableWidgetItem(str(data)))

    def search\_records(self):

        search\_text = self.search\_input.text().lower()

        filtered\_records = []

        for record in self.records:

            if search\_text in str(record).lower():

                filtered\_records.append(record)

        self.records = filtered\_records

        self.update\_table()

    def refresh\_table(self):

        self.search\_input.clear()

        self.records = extract\_attendance\_records()

        self.update\_table()

    def setup\_form\_page(self):

        layout = QVBoxLayout(self.page\_form)

        layout.setAlignment(QtCore.Qt.AlignCenter)

        heading\_label = QLabel("File Selection")

        heading\_label.setFont(QFont("Arial", 20, QFont.Bold))

        layout.addWidget(heading\_label)

        self.file\_path\_label = QLabel("Selected File: No file selected")

        layout.addWidget(self.file\_path\_label)

        button\_select\_file = QPushButton("Select File")

        button\_select\_file.clicked.connect(self.select\_file)

        layout.addWidget(button\_select\_file)

        button\_submit = QPushButton("Submit")

        button\_submit.clicked.connect(submit\_function)

        layout.addWidget(button\_submit)

        # Apply styling using style sheets

        self.setStyleSheet(

            """

            QLabel {

                font-size: 18px;

                padding: 10px;

            }

            QPushButton {

                font-size: 16px;

                padding: 10px 20px;

                background-color: #4CAF50;

                color: white;

                border: none;

            }

            QPushButton:hover {

                background-color: #45a049;

            }

            """

        )

        # Set layout margins and spacing

        layout.setContentsMargins(50, 50, 50, 50)

        layout.setSpacing(20)

        self.show()

    def select\_file(self):

        file\_dialog = QFileDialog()

        file\_path, \_ = file\_dialog.getOpenFileName(self, "Select File")

        if file\_path:

            self.file\_path\_label.setText(f"Selected File: {file\_path}")

        else:

            self.file\_path\_label.setText("Selected File: No file selected")

def submit\_function():

    file\_path = window.file\_path\_label.text().split(": ")[-1]

    print("Selected File:", file\_path)

    # Process the file further or call other functions for processing

    import cv2

    from deepface import DeepFace

    img = cv2.imread(file\_path)

    import matplotlib.pyplot as plt

    predictions =  DeepFace.analyze(img,

        actions = [ 'emotion']

    )

    expression = str((predictions[0]['dominant\_emotion']))

    dfs = DeepFace.find(

        img,

        db\_path = "employees")

    print(dfs[0])

    file\_path = str((dfs[0]["identity"][0]))

    print(file\_path)

    # Extract the substring after the last '/' and before the '.'

    extracted\_string = file\_path[file\_path.rindex('/') + 1:file\_path.rindex('.')]

    add\_attendance\_record(extracted\_string, expression)

    print("data passed")

def extract\_attendance\_records():

    import sqlite3

    conn = sqlite3.connect('example.db')

    cursor = conn.cursor()

    # Execute the SELECT query to retrieve all rows from the table

    cursor.execute("SELECT \* FROM attendance\_record")

    rows = cursor.fetchall()

    # Print or process the retrieved data

    for row in rows:

        print(row)  # Replace with your desired processing

    cursor.close()

    conn.close()

    return rows if rows else []

def add\_attendance\_record(id\_no, expression):

    conn = sqlite3.connect('example.db')

    cursor = conn.cursor()

    # Get user input for time, expressions, and uniform\_status

    time = datetime.datetime.now().strftime('%H:%M:%S')

    expressions = expression

    uniform\_status = "complete"

    name = extract\_name\_by\_emp\_id(id\_no)

    # Insert the data into the table

    cursor.execute("""

        INSERT INTO attendance\_record (id\_no, time, name, expressions, uniform\_status)

        VALUES (?, ?, ?, ?, ?)

    """, (id\_no, time, name, expressions, uniform\_status))

    conn.commit()

    cursor.close()

    conn.close()

    print("data saved successfully")

def extract\_name\_by\_emp\_id(emp\_id):

    # Connect to the SQLite database

    conn = sqlite3.connect('example.db')

    cursor = conn.cursor()

    # Execute the query to retrieve the name based on emp\_id

    cursor.execute("SELECT name FROM students\_record WHERE emp\_id=?", (emp\_id,))

    result = cursor.fetchone()

    # Close the connection

    conn.close()

    if result:

        return result[0]  # Extract the name from the result

    else:

        return None  # No name found for the given emp\_id

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

    app = QApplication(sys.argv)

    window = SideNavMenu()

    sys.exit(app.exec\_())

# Appendix: Mood Detection Tool

## System Requirements:

* Python (version X.X or higher)
* Required libraries and dependencies:
* OpenCV
* TensorFlow
* SQLAlchemy

## Tool Structure:

* mood\_detection\_tool.py: Main script that launches the Mood Detection Tool.
* mood\_detection.py: Contains the implementation of the mood detection algorithm.
* database.py: Handles the database connection and operations.
* user\_interface.py: Manages the user interface of the tool.
* utils.py: Contains utility functions used throughout the project.

## Installation Instructions:

* Download the Mood Detection Tool from the provided source.
* Install Python on your system (if not already installed).
* Install the required libraries and dependencies using the following commands:
* pip install opencv-python
* pip install tensorflow
* pip install sqlalchemy

## Usage Instructions:

* Launch the Mood Detection Tool by executing the following command:
* python mood\_detection\_tool.py
* Follow the instructions on the user interface to capture an image or select an image file for mood analysis.
* View the detected mood (happy or sad) displayed on the user interface.
* Save the mood data to the database using the "Save" button.
* Retrieve stored mood data from the database using the "View Data" button.

## Troubleshooting:

* If you encounter any issues while using the Mood Detection Tool, consider the following:
* Ensure that the required libraries and dependencies are installed correctly.
* Check that your camera is properly connected and functioning.
* Verify that the database connection details are correct.
* Refer to the project documentation for more troubleshooting tips.

## Limitations:

* The Mood Detection Tool may have limitations that include:
* Accuracy of mood detection is dependent on the training data and the chosen machine learning algorithm.
* The tool may not accurately detect moods in certain lighting conditions or with obscured facial features.
* Performance may vary depending on the hardware capabilities of the system.

## Future Improvements:

* The Mood Detection Tool can be enhanced in the following ways:
* Incorporating a wider range of facial expressions and moods for more comprehensive mood analysis.
* Implementing real-time mood detection for continuous monitoring and analysis.
* Adding additional features such as age and gender estimation to provide more context to the mood analysis.

## Contributors:

* List the names and roles of the individuals who contributed to the development of the Mood Detection Tool.

## References:

* Include any references to external resources, libraries, or research papers used during the development of the tool.

# GUI

