TED UNIVERSITY
LOW LEVEL DESIGN PROJECT

CMPE 492
SENIOR DESIGN PROJECT 2

Begüm AKDENİZ 16669037806
Umut YILDIRIM 10822781650
Ceren BÜYÜKGÜLLÜ 27142510204
İrem TAMAY 59701366816

Table of Contents

1 INTRODUCTION

Today, educational institutions need technological solutions to ensure the safety of students and staff. In order to meet this need, the "School Entrance with Facial Recognition System" project is being implemented in our school. This project aims to increase the security of the school and optimize operational efficiency by transforming traditional access control methods with a contemporary approach. In this project, a system is provided that will detect students' faces from the cameras at the entrances and can pass through the turnstiles if they match the school system. If the students' faces match 80%, the system will give approval. With the interfaces in our system, new registrations can be easily recorded and students' entry and exit times can be recorded. The main purpose of this project is to improve our school's admission process with modern technology. We will save time by accelerating the physical identity verification process by recognizing the faces of our students and staff. Additionally, we will be able to quickly detect potential security threats with the facial recognition system. In this way, we aim to maximize the safety of our students and staff by minimizing possible risks in our school.
The facial recognition system offers many advantages over traditional methods. With the introduction of the system, manual authentication processes will be reduced and therefore the login process will be accelerated. With this project, our school aims to increase student safety and the quality of the school environment by using technology.

1.1 Object Design Trade-Offs

In the context of designing a face recognition system, there are several object design trade-offs that need to be considered.

1.1.1 Accuracy vs. Efficiency:

- *Accuracy:* The system must identify individuals accurately. It must be verified under all circumstances. Therefore higher algorithms and higher calculations may be required.

- *Efficiency:* Balancing accuracy with computational speed is important. A highly accurate system might be computationally expensive, which could impact real-time performance.

1.1.2 Privacy vs. Functionality:

- *Privacy:* Implementing strict privacy measures, such as encryption, anonymization, and access controls, is crucial to protect users' sensitive data. This may introduce complexities in the design.

- *Functionality***:** Balancing privacy measures with the system's functionality is essential. Excessive privacy measures could potentially hinder the system's effectiveness.

1.1.3 Scalability vs. Resource Constraints:

- *Scalability***:** Designing the system to handle a growing number of users or an increasing volume of data is important for long-term viability.

- *Resource Constraints***:** Depending on the deployment environment, there may be limitations in terms of memory, processing power, or network bandwidth that need to be taken into account.

1.2 Interface Documentation Guidelines

This section serves as a cornerstone for laying down comprehensive guidelines on how to document the interfaces that link the diverse components and modules within the system. The primary objective is to ensure that these interfaces are meticulously defined, impeccably documented, and uphold a consistent standard across the entire system.

In adherence to these guidelines, all classes, attributes, and methods within this report are designated using camel case. Class titles are initiated with a capital letter, while others begin with lowercase letters. The prescribed structure for describing class interfaces is as follows:

class ClassName
Information about the class.
Attributes
typeOfAttribute   nameOfAttribute ..
Methods
returnType MethodName(parameters) If necessary, the method will be explained..

1.3 Engineering Standards

In our reports, we followed the UML guideline for class interfaces, subsystem compositions, diagrams and scenarios. The reason we used UML was that it was a good method and easy to prepare for us in creating these diagrams. Our reports are created in accordance with and following IEEE standards.

1.4 Definitions, Acronyms And Abbreviations

Abbreviations used in our report are given below. The most important reason for explaining this is to ensure correct readability of the report. This is to avoid any misunderstanding of the abbreviations used. The understandability and readability of our report is important.

- **FR:** Face recognition
- **UML:** Unified Modelling Language. It is a generic developmental modelling language used for analysis, design and implementation of software systems.
- **AI:** Artificial intelligence.
- **IEEE:** This means "Institute of Electrical and Electronics Engineers" . It is a global professional organization that is dedicated to advancing technology for the benefit of humanity.
- **ML:** Machine learning.
- **MVC:** Model View Controller architecture.
- **UI:** User Interface.

1 PACKAGES

Face Recognition Module: This module includes the face recognition system. It matches and recognizes faces by processing camera images.

Database Integration Module: This system stores the face systems of the entire school and the names, surnames and student numbers. Data of students and employees are stored in this module.

Access Control Module: This system is based on allowing or denying access through facial recognition. It is the process of opening or not opening the door by recognizing faces and then approving or rejecting them.
Camera integration Module: This is the package that will integrate the camera hardware and transmit the captured images to the Face Recognition module. This is a connection between the camera and the software.

.*User interface Module*:The user interface is included in this package. The necessary information is shown to the user in this package. New entries and exits are made through this system.
Communication Module: Provides communication with the Server. It is the package that will send face recognition results data to the server part. This package provides data flow.

MODEL
User: It is a data structure that contains basic information of users. Stores users' usernames and passwords.
User Controller: Communication is established with the user interface. Requests from the user are received, the relevant action is performed and the result is sent to the user.
Access Log: It is a model that keeps users' logins and logouts in time.

SERVICE
User Service: User operations are performed. The new user is registered in this section. Login or password also includes reset sections.
Facial Recognition Service: The image coming from the camera is processed. Face detection and recognition operations are performed. Facial recognition systems are implemented.
Access Log Service: User login and logout records are processed.

CONTROLLER.
User Controller: This is the section where user requests are received and directed to User Service.
Facial recognition controller: The image from the camera is taken and directed to the service section. The results are processed here
.Access Log Controller: User login and logout requests are managed. From here it is directed to the service.

CLIENT

VIEW:
Login View: The interface through which users log in is provided.
Registration View: The interface where new user registrations will be provided.
Facial Recognition View: Interface that presents facial recognition processes in a visual way.

CONTROLLER
Login Controller: Receives user login requests, performs the necessary verifications and manages the login process.
Registration Controller: Receives new user registration requests, performs the necessary verifications and manages the registration process.
Facial Recognition Controller: Receives the image from the camera, directs it to the Facial Recognition Service and processes the results.
Access Log Controller: Manages user input/output requests and directs them to Access Log Service.

Model (Client Side)

User Model (Client): Represents user data and is used in the user interface.

Facial Recognition Model (Client): Visually represents facial recognition processes.

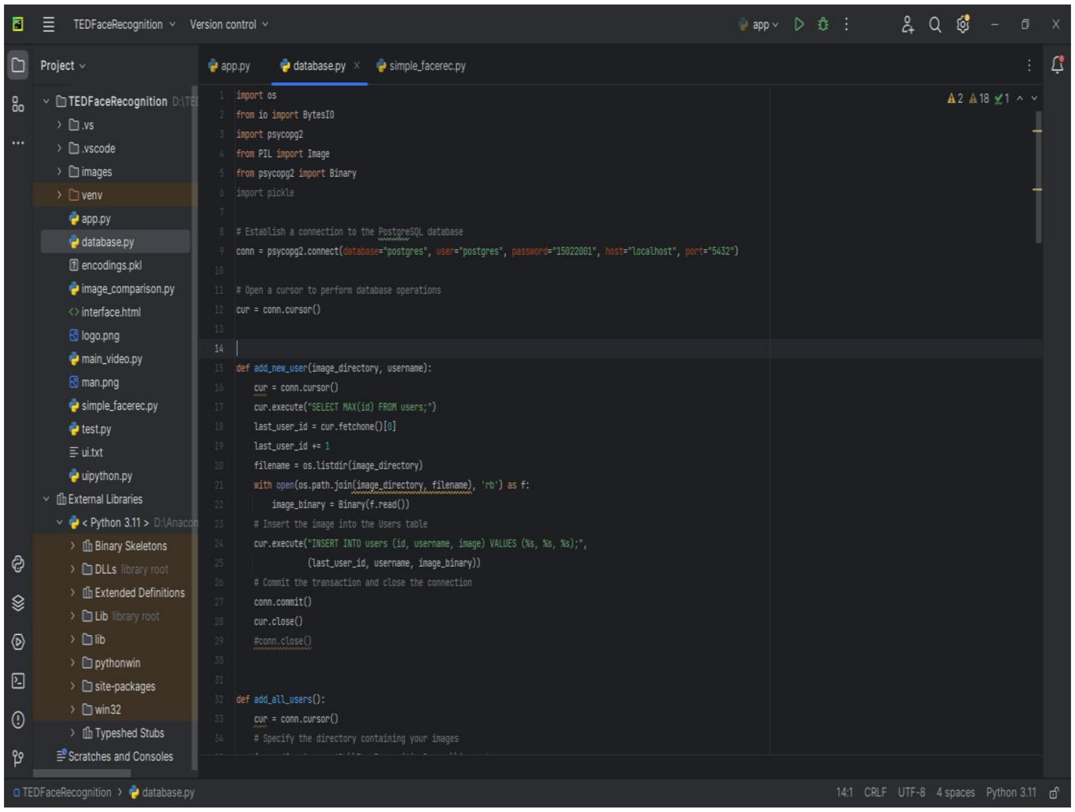
Access Log Model (Client): Represents user input/output records, used in the user interface.

2 CLASS INTERFACES

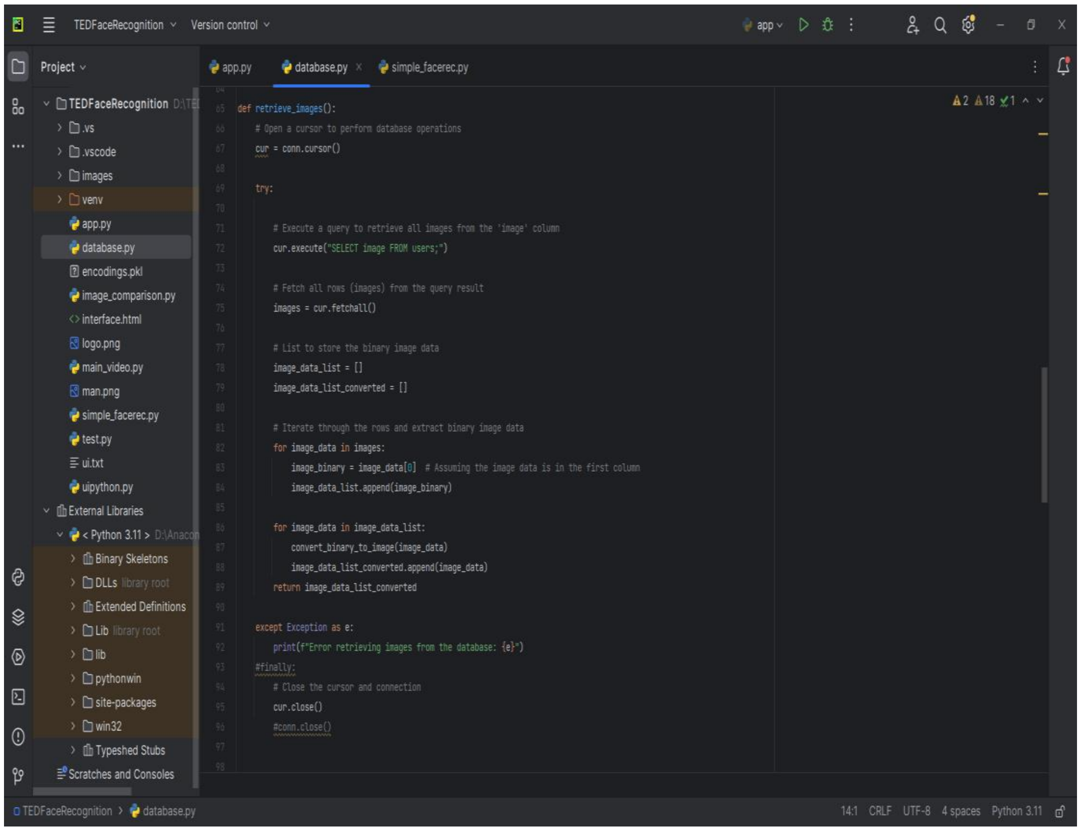
Class	DatabaseManager
Package	Authentication/Authorization

Methods	add_new_user(image_directory, username)
	add_all_users()
	retrieve_images()
	retrieve_users()
	retrieve_username(id)
	convert_binary_to_image(binary_data)

Description	Inserts a new user with an image into the database Inserts all users with images from a specified directory into the database. Retrieves and returns a list of binary image data from the database Retrieves and returns a list of usernames from the database. Retrieves and returns the username associated with the given user ID. Converts binary image data to an image object using Pillow (PIL).



```python
import os
from io import BytesIO
import psycopg2
from PIL import Image
from psycopg2 import Binary
import pickle

# Establish a connection to the PostgreSQL database
conn = psycopg2.connect(database="postgres", user="postgres", password="15022001", host="localhost", port="5432")

# Open a cursor to perform database operations
cur = conn.cursor()


def add_new_user(image_directory, username):
    cur = conn.cursor()
    cur.execute("SELECT MAX(id) FROM users;")
    last_user_id = cur.fetchone()[0]
    last_user_id += 1
    filename = os.listdir(image_directory)
    with open(os.path.join(image_directory, filename), 'rb') as f:
        image_binary = Binary(f.read())
    # Insert the image into the Users table
    cur.execute("INSERT INTO users (id, username, image) VALUES (%s, %s, %s);",
                (last_user_id, username, image_binary))
    # Commit the transaction and close the connection
    conn.commit()
    cur.close()
    #conn.close()


def add_all_users():
    cur = conn.cursor()
    # Specify the directory containing your images
```

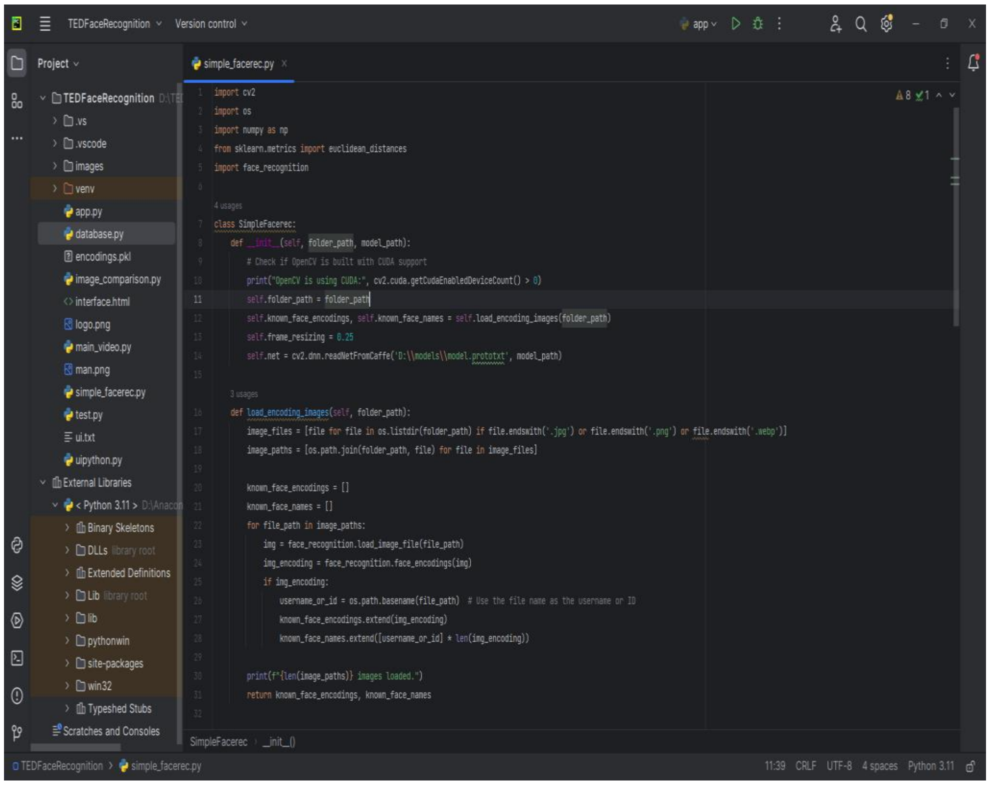


```python
def retrieve_images():
    # Open a cursor to perform database operations
    cur = conn.cursor()

    try:

        # Execute a query to retrieve all images from the 'image' column
        cur.execute("SELECT image FROM users;")

        # Fetch all rows (images) from the query result
        images = cur.fetchall()

        # List to store the binary image data
        image_data_list = []
        image_data_list_converted = []

        # Iterate through the rows and extract binary image data
        for image_data in images:
            image_binary = image_data[0]  # Assuming the image data is in the first column
            image_data_list.append(image_binary)

        for image_data in image_data_list:
            convert_binary_to_image(image_data)
            image_data_list_converted.append(image_data)
        return image_data_list_converted

    except Exception as e:
        print(f"Error retrieving images from the database: {e}")
    #finally:
        # Close the cursor and connection
        cur.close()
        #conn.close()
```

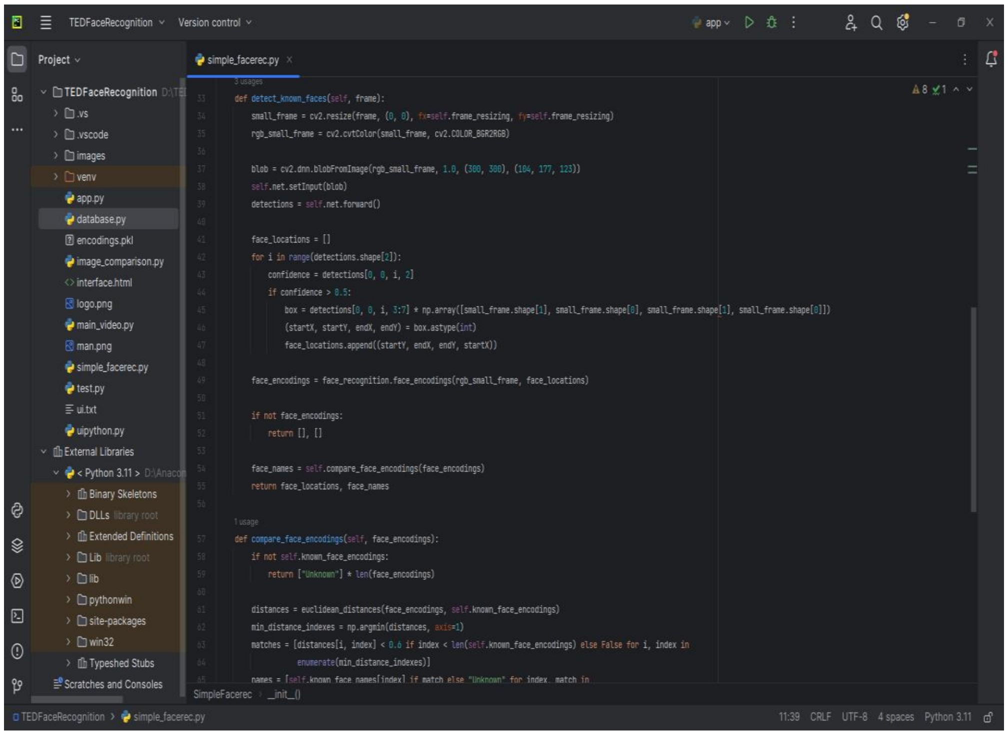
Class	SimpleFaceRecognition
Package	FaceRecognition

Methods	__init__(self, folder_path, model_path)
	load_encoding_images(folder_path)
	detect_known_faces(frame)
	compare_face_encodings(face_encodings)

Description	Initializes the SimpleFacerec class with the folder path and model path. Loads and encodes face images from the specified folder path. Detects known faces in the provided video frame using face recognition and deep learning. Compares face encodings with known face encodings and returns corresponding names.



```python
import cv2
import os
import numpy as np
from sklearn.metrics import euclidean_distances
import face_recognition

4 usages
class SimpleFacerec:
    def __init__(self, folder_path, model_path):
        # Check if OpenCV is built with CUDA support
        print("OpenCV is using CUDA:", cv2.cuda.getCudaEnabledDeviceCount() > 0)
        self.folder_path = folder_path
        self.known_face_encodings, self.known_face_names = self.load_encoding_images(folder_path)
        self.frame_resizing = 0.25
        self.net = cv2.dnn.readNetFromCaffe('D:\\models\\model.prototxt', model_path)

    3 usages
    def load_encoding_images(self, folder_path):
        image_files = [file for file in os.listdir(folder_path) if file.endswith('.jpg') or file.endswith('.png') or file.endswith('.webp')]
        image_paths = [os.path.join(folder_path, file) for file in image_files]

        known_face_encodings = []
        known_face_names = []
        for file_path in image_paths:
            img = face_recognition.load_image_file(file_path)
            img_encoding = face_recognition.face_encodings(img)
            if img_encoding:
                username_or_id = os.path.basename(file_path)  # Use the file name as the username or ID
                known_face_encodings.extend(img_encoding)
                known_face_names.extend([username_or_id] * len(img_encoding))

        print(f"{len(image_paths)} images loaded.")
        return known_face_encodings, known_face_names
```



```python
    def detect_known_faces(self, frame):
        small_frame = cv2.resize(frame, (0, 0), fx=self.frame_resizing, fy=self.frame_resizing)
        rgb_small_frame = cv2.cvtColor(small_frame, cv2.COLOR_BGR2RGB)

        blob = cv2.dnn.blobFromImage(rgb_small_frame, 1.0, (300, 300), (104, 177, 123))
        self.net.setInput(blob)
        detections = self.net.forward()

        face_locations = []
        for i in range(detections.shape[2]):
            confidence = detections[0, 0, i, 2]
            if confidence > 0.5:
                box = detections[0, 0, i, 3:7] * np.array([small_frame.shape[1], small_frame.shape[0], small_frame.shape[1], small_frame.shape[0]])
                (startX, startY, endX, endY) = box.astype(int)
                face_locations.append((startY, endX, endY, startX))

        face_encodings = face_recognition.face_encodings(rgb_small_frame, face_locations)

        if not face_encodings:
            return [], []

        face_names = self.compare_face_encodings(face_encodings)
        return face_locations, face_names

    def compare_face_encodings(self, face_encodings):
        if not self.known_face_encodings:
            return ["Unknown"] * len(face_encodings)

        distances = euclidean_distances(face_encodings, self.known_face_encodings)
        min_distance_indexes = np.argmin(distances, axis=1)
        matches = [distances[i, index] < 0.6 if index < len(self.known_face_encodings) else False for i, index in
                   enumerate(min_distance_indexes)]
        names = [self.known_face_names[index] if match else "Unknown" for index, match in
```

Dependencies:

Libraries: OpenCV, NumPy, scikit-learn, face_recognition, psycopg2, Pillow

External Models: Caffe Deep Learning Model (Used for face detection)

Notes:

- The SimpleFacerecognition class integrates face recognition capabilities, utilizing the face_recognition library for encoding and comparing faces, as well as a pre-trained deep learning model for face detection.
- Database operations, such as adding new users, retrieving user information, and converting binary data to images, are managed by the DatabaseManager class.
- The system requires the specified external dependencies and models to function properly.

CLASS	CameraApp
Package	There is no specific package.

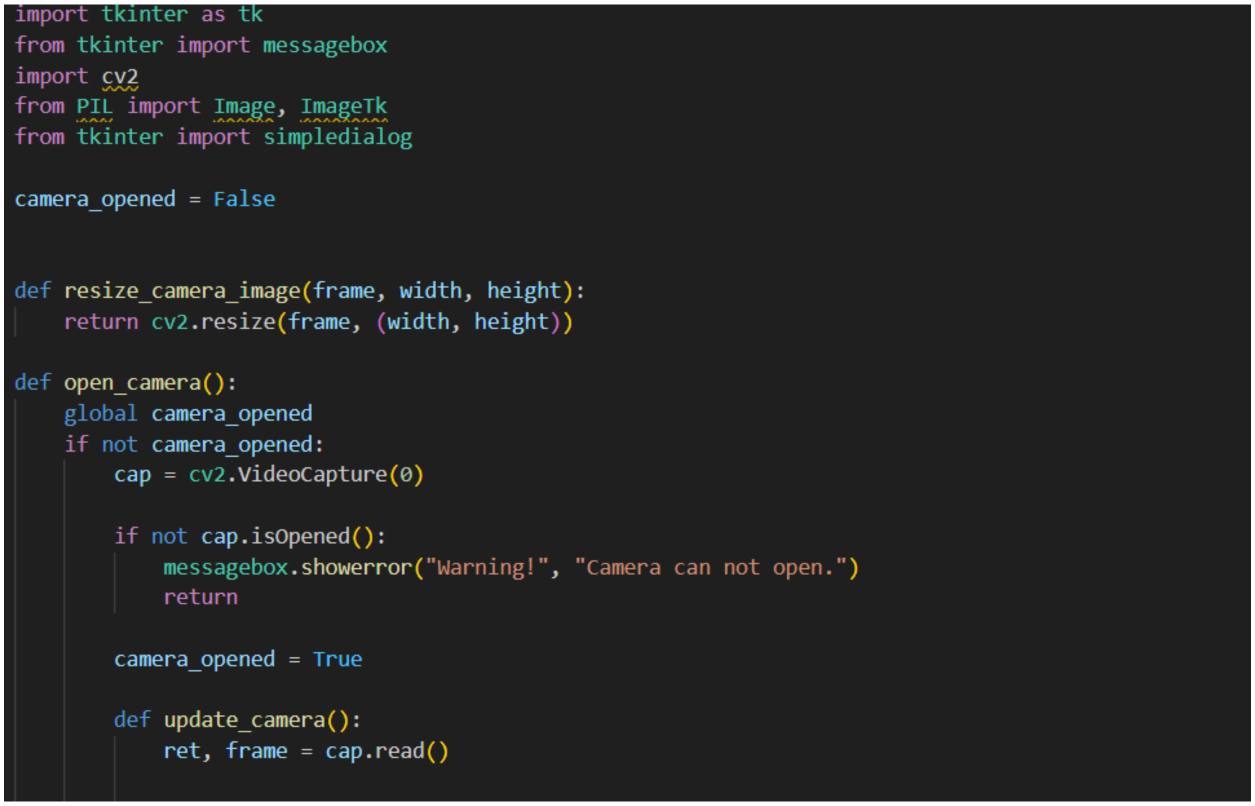
Methods	__init__(self, root)
	create_gui(self
	resize_camera_image(self, frame, width, height)
	open_camera(self)
	take_photo(self)
	quit_app(self)

Descriptions	Launches the application's GUI. It renders the main GUI and resizes the camera image size. It opens the camera and then updates it. When a photo is to be taken, a button appears on the screen. The application is exited.

CLASS	tkinter.Tk (root)
Package	tkinter

Methods	__init__(self)
	title(self, title)
	geometry(self, geometry)
	configure(self, **kwargs)

Descriptions	Launches the app's home screen and the title of the home screen. Adjusts the position and size of the screen.



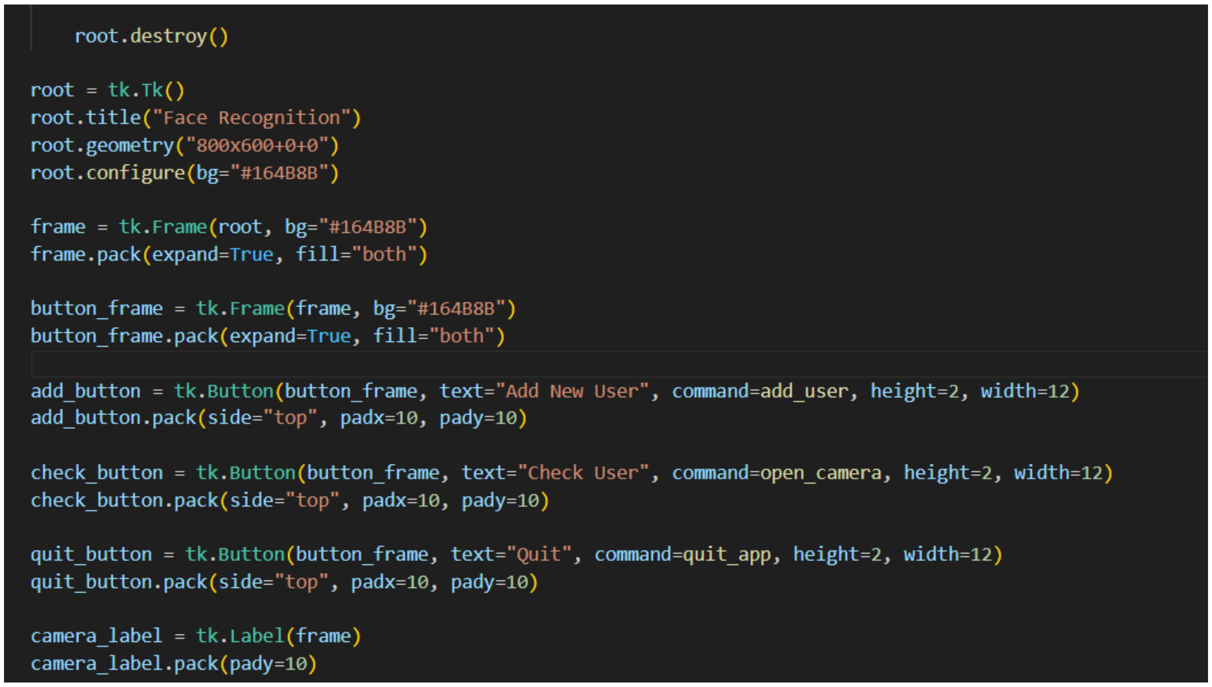
```python
import tkinter as tk
from tkinter import messagebox
import cv2
from PIL import Image, ImageTk
from tkinter import simpledialog

camera_opened = False


def resize_camera_image(frame, width, height):
    return cv2.resize(frame, (width, height))

def open_camera():
    global camera_opened
    if not camera_opened:
        cap = cv2.VideoCapture(0)

        if not cap.isOpened():
            messagebox.showerror("Warning!", "Camera can not open.")
            return

        camera_opened = True

        def update_camera():
            ret, frame = cap.read()
```

CLASS	tkinter.Frame (frame)
Package	tkinter

Methods	__init__(self, master, **kwargs)
	pack(self, **options)
	grid_rowconfigure(self, index, **kwargs)
	grid_columnconfigure(self, index, **kwargs)

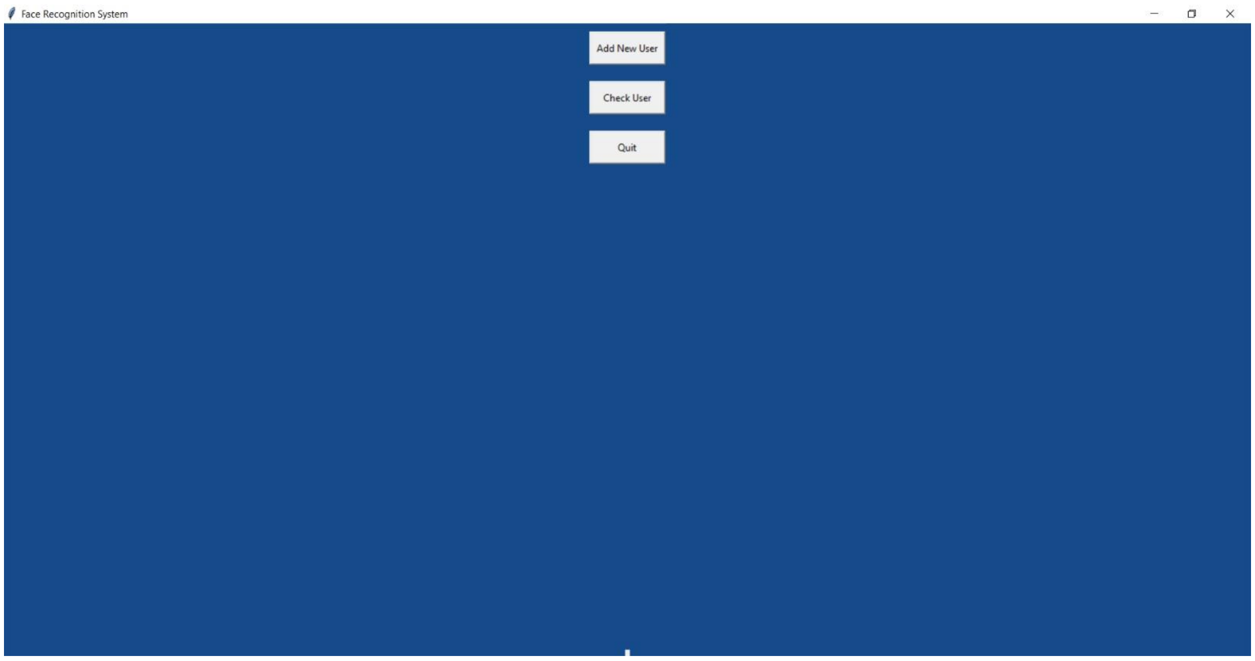
Descriptions	Sets and initializes the frames of the main screen. Packs the frame into the main window. Configures row and column weights.

CLASS	tkinter.Button (add_button, check_button, quit_button, take_photo_button)
Package	tkinter
Methods	__init__(self, master, text, command, height, width) pack(self, **options)
Descriptions	It initiates button and starts it. Insert the button into the frame.



```
    root.destroy()

root = tk.Tk()
root.title("Face Recognition")
root.geometry("800x600+0+0")
root.configure(bg="#164B8B")

frame = tk.Frame(root, bg="#164B8B")
frame.pack(expand=True, fill="both")

button_frame = tk.Frame(frame, bg="#164B8B")
button_frame.pack(expand=True, fill="both")

add_button = tk.Button(button_frame, text="Add New User", command=add_user, height=2, width=12)
add_button.pack(side="top", padx=10, pady=10)

check_button = tk.Button(button_frame, text="Check User", command=open_camera, height=2, width=12)
check_button.pack(side="top", padx=10, pady=10)

quit_button = tk.Button(button_frame, text="Quit", command=quit_app, height=2, width=12)
quit_button.pack(side="top", padx=10, pady=10)

camera_label = tk.Label(frame)
camera_label.pack(pady=10)
```

CLASS	tkinter.Label (camera_label)
Package	tkinter
Methods	pack(self, **options) __init__(self, master) |
Descriptions	Starts a tag to display images. Places the label in the frame.



3 GLOSSARY

Binary image: A binary image is one that consists of pixels that can have one of exactly two colors, usually black and white.

Encoded data: Data converted from one format to another for any purpose is called encoded data.

Facial Image: A photograph or video frame or other image that shows the visible physical structure of an individual's face.

Facial Recognition Software: Software designed to compare the visible physical structure of an individual's face with a stored facial image.

Frame: It is a type of widget used in Tkinter. It is used to group and organize other widgets.

GUI (Graphical User Interface): A graphical interface that allows the application to interact with the user.

Widget: They are graphical interface elements used in Tkinter. Examples are Button, Label, and Frame.

4 REFERENCES

https://www.innovatrics.com/facial-recognition-technology/

https://www.tutorialspoint.com/python/python_gui_programming.htm

https://realpython.com/python-gui-tkinter/