Supplementary Material

DLIB, YOLO, and Deep Learning-Based Motion and Anomaly Detection Systems for School Security Platform

S1. Software Architecture

This supplementary section provides additional technical details regarding the implementation of the proposed AI-supported school security platform.

The developed system consists of six primary Python modules:

File Name	Function
uygulama.py	Main application interface
kameralar.py	Multi-camera management
program.py	Facial recognition and object detection
yuz_ekle.py	User registration interface
egitim_programi.py	Face image acquisition
veriler.py	Data filtering and visualization

S2. User Interfaces

This section presents the graphical user interfaces developed for the proposed platform. The main interface enables users to:

- Start the monitoring system
- Register new individuals
- Access recorded data

Figure S2.1. Interface User

S3. Multi-Camera Monitoring System

The platform supports simultaneous monitoring through multiple camera streams. Users may:

- Add cameras
- Remove cameras
- Define monitored locations
- Configure notifications

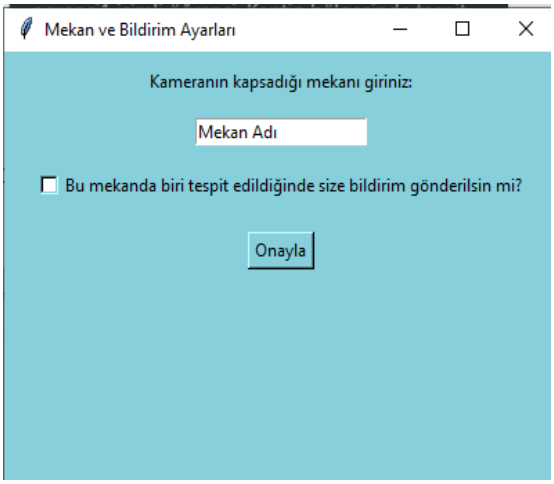


Figure S3.1. Camera Selection Interface

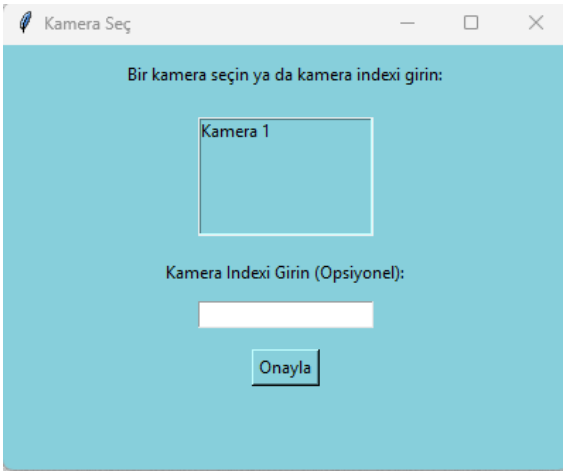


Figure S3.2. Location and Notification Settings

S4. Face Registration Module

The face registration module enables enrollment of students and staff members.
For each individual:

- Name information is entered
- Role/class information is entered
- Identification number is entered
- Five facial images are captured from different angles

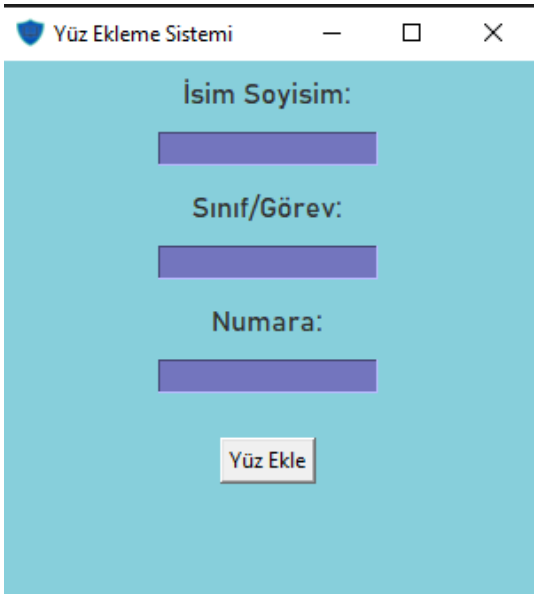


Figure S4. Face Registration Interface

S5. Data Recording Infrastructure

The system automatically stores monitoring data in CSV format.
Three main categories are used:

S5.1 Location-Based Records

Stored under:

Veriler/Mekanlar
Generated files:

- hareket.csv
- ogrenci.csv

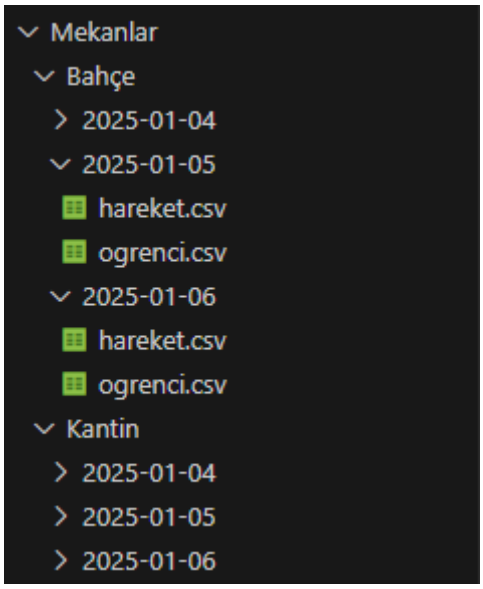


Figure S5.1. Location-Based Data Structure

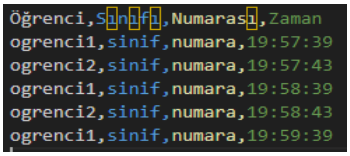


Figure S5.2. Example hareket.csv File

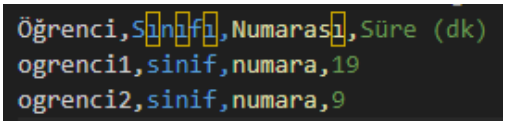


Figure S5.3. Example ogrenci.csv File

S5.2 Student-Based Records

Stored under:

Veriler/Ogrenciler

Generated files:

- cevre.csv
- hareket.csv
- mekan.csv

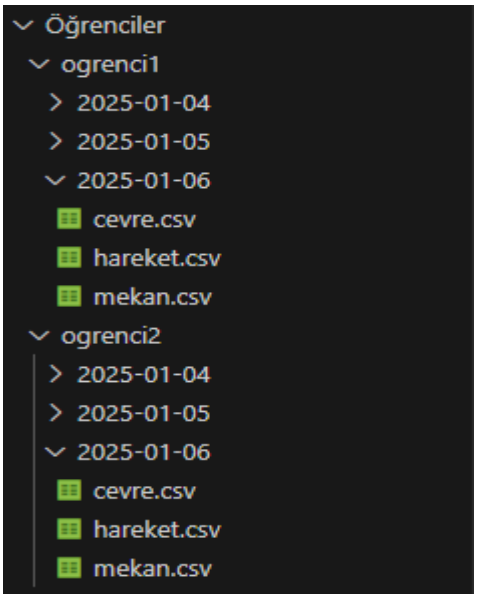


Figure S5.4. Student Data Structure

```
Öğrenci,Sınıf1,Numarası,Süre (dk)
ogrenci1,sınıf,numara,19
ogrenci2,sınıf,numara,12
```

Figure S5.5. Example cevre.csv File

```
Mekan,Sınıf1,Numarası,Zaman
Kantin,sınıf,numara,19:57:39
Kantin,sınıf,numara,19:58:39
Kantin,sınıf,numara,19:59:39
Bahçe,sınıf,numara,20:00:45
Bahçe,sınıf,numara,20:01:45
```

Figure S5.6.. Example hareket.csv File

```
Mekan,Sınıf1,Numarası,Süre (dk)
Kantin,sınıf,numara,19
Bahçe,sınıf,numara,13
```

Figure S5.7. Example mekan.csv File

S5.3 Potential Violation Records

Stored under:

Veriler/Potansiyel_Ihlal_Unsurlari

Generated file:

- ihlal_unsurlari.csv

Figure S5.8. Violation Data Structure

Figure S5.9. Example ihlal_unsurlari.csv File

S6. Data Visualization and Search Functions

The developed platform includes a dedicated data visualization module.
The module enables:

- Searching data
- Filtering data
- Displaying CSV content
- Reviewing historical records

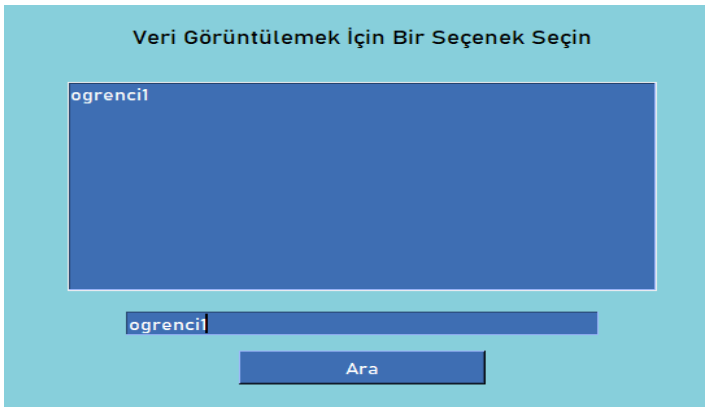


Figure S6.1. Search Interface

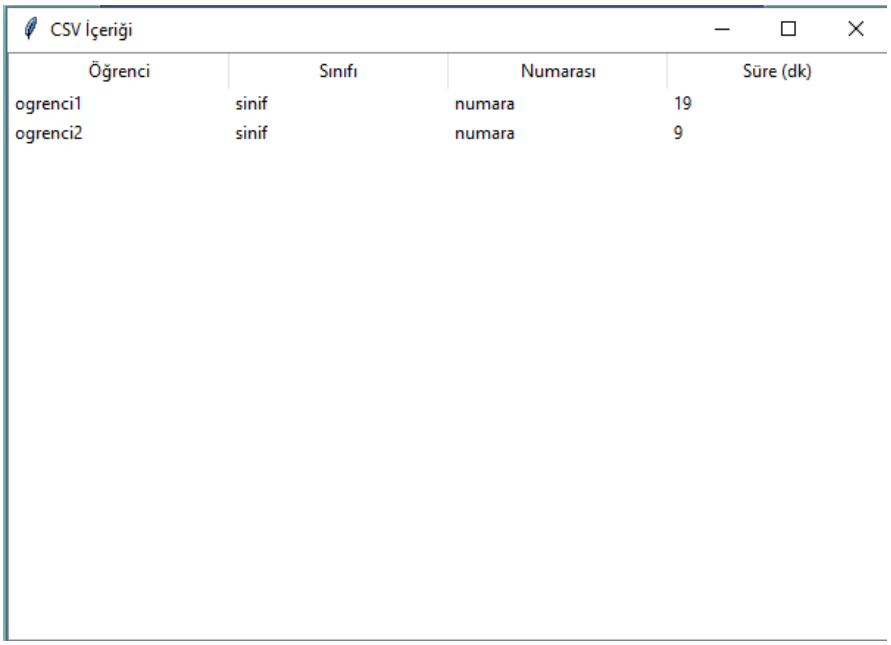


Figure S6.2. Table Visualization Interface

S7. YOLOv8 Training Details

The object detection model was trained using a combined dataset obtained from publicly available sources.
Datasets included:

- Guns-Knives Object Detection Dataset
- Smoking and Drinking Dataset for YOLO
- Smartphone Detection Dataset (Roboflow)

Training Configuration

Parameter	Value
Epochs	108
Model	YOLOv8
Classes	Mobile Phone, Knife, Cigarette
Training Samples	9,191

S8. Source Code Availability

The complete implementation consists of approximately six interconnected Python modules developed for educational security monitoring applications.

Source code can be made available by the authors upon reasonable request.

Project Codes and Files

Note: Each command line has been explained with comment lines, as shown in the examples below, describing its purpose and function.

application.py:

This file creates our main interface and enables other files to be called through this interface. It is the main file of our program.

main.py

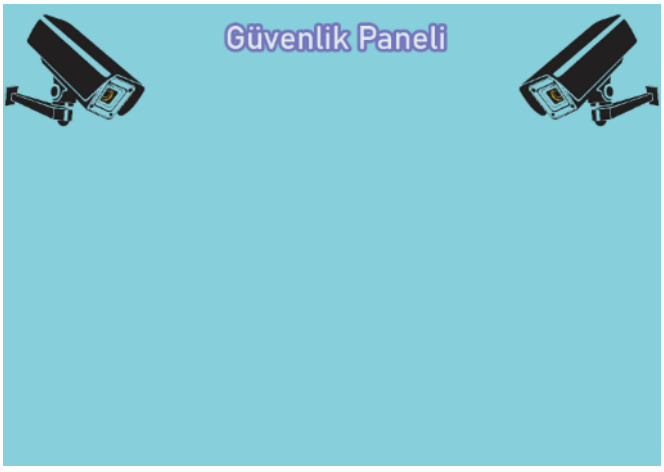
```
import tkinter as tk  # Tkinter, GUI oluşturmak için
import subprocess  # Diğer Python dosyalarını çalıştırmak için
import multiprocessing  # Çeşitli işlemleri paralel çalıştırmak için
from PIL import Image, ImageTk  # Görsel işlemler için PIL (Pillow) kütüphanesi

# Bu satır, multiprocessing'in 'spawn' başlatma yöntemini kullanmasını sağlar
multiprocessing.set_start_method('spawn', force=True)

# Pencere oluştur
window = tk.Tk()  # Tkinter pencere nesnesi oluşturuyoruz
window.title("Güvenlik Arayüzü")  # Pencerenin başlığı
window.geometry("1024×720")  # Pencerenin boyutları (1024×720)

# Pencerenin simgesi olarak bir ikon ekliyoruz
window.iconbitmap("logo.ico")

# Ekran boyutları alınıyor ve pencere ortalanıyor
screen_width = window.winfo_screenwidth()  # Ekranın genişliği
screen_height = window.winfo_screenheight()  # Ekranın yüksekliği

# Pencerenin ekran ortasında yer almasını sağlıyoruz
x = (screen_width - 1024) // 2
y = (screen_height - 720) // 2
window.geometry(f"1024×720+{x}+{y}")  # Ekranın ortasında olacak şekilde yerleştir

# Arka plan resmi yükleme
background_image = Image.open("Assets/arkaplan.png")  # Resmi aç
background_photo = ImageTk.PhotoImage(background_image)  # Tkinter ile uyumlu hale getir

# Arka plan için bir etiket oluştur
background_label = tk.Label(window, image=background_photo)  # Arka plan resmini ekle
background_label.place(relwidth=1, relheight=1)  # Arka planı tüm pencereye yay

# Butonların işlevlerini tanımlıyoruz
def button1_click():
    subprocess.Popen(['python', 'Programlar/kameralar.py'])  # Bu buton kameralar.py dosyasını çalıştırır

def button2_click():
    subprocess.Popen(['python', 'Programlar/yuz_ekle.py'])  # Bu buton yuz_ekle.py dosyasını çalıştırır

def button3_click():
    subprocess.Popen(['python', 'Programlar/veriler.py'])  # Bu buton veriler.py dosyasını çalıştırır

# Buton resimlerini yükleme
def load_image_with_transparency(image_path):
    image = Image.open(image_path).convert("RGBA")  # Resmi RGBA formatına dönüştür
    return ImageTk.PhotoImage(image)  # Tkinter ile uyumlu hale getir

# Buton resimlerini yükliyoruz
button1_photo = load_image_with_transparency("Assets/bÇalıştır.png")
button2_photo = load_image_with_transparency("Assets/bYüzekle.png")
button3_photo = load_image_with_transparency("Assets/bVeriler.png")

# Butonları oluşturuyoruz ve ortalıyoruz
button1 = tk.Button(window, image=button1_photo, command=button1_click, borderwidth=0, highlightthickness=0, bg='#87cfdb',
activebackground='#87cfdb')
button1.place(relx=0.5, rely=0.3, anchor='center')  # Pencerenin ortasına yerleştir

button2 = tk.Button(window, image=button2_photo, command=button2_click, borderwidth=0, highlightthickness=0, bg='#87cfdb',
activebackground='#87cfdb')
button2.place(relx=0.5, rely=0.5, anchor='center')  # Pencerenin ortasına yerleştir

button3 = tk.Button(window, image=button3_photo, command=button3_click, borderwidth=0, highlightthickness=0, bg='#87cfdb',
activebackground='#87cfdb')
button3.place(relx=0.5, rely=0.7, anchor='center')  # Pencerenin ortasına yerleştir

# Ana döngüyü başlat
window.mainloop()  # Tkinter pencere döngüsünü başlatır
```

Output of the Above Code: Security Panel

When the “Run” button is pressed, the “kameralar.py” file is executed, displaying multiple camera streams on the screen. When the “Add Face” button is pressed, the “yuz_ekle.py” file is executed. When the “Data” button is selected, the “veriler.py” file is executed.

logo.ico File:

background.png File:



bÇalıştır.png File:

bVeriler.png File:

bYüzEkle.png File:

kameralar.py:

This file enables the display of multiple camera streams on the screen. Camera addition and removal operations are performed through this interface. It displays the camera images processed by the “program.py” file.

main.py

```python
# Gerekli kütüphaneler import ediliyor
import cv2  # OpenCV: Görüntü işleme için
import os  # İşletim sistemi işlemleri için
import sys  # Sistemle ilgili işlemler için
import numpy as np  # NumPy: Matematiksel işlemler ve veri işlemleri için
import tkinter as tk  # Tkinter: GUI (grafiksel kullanıcı arayüzü) oluşturmak için
from tkinter import messagebox, simpledialog  # Tkinter: Kullanıcıya mesaj kutuları göstermek için
import multiprocessing  # Çoklu işlem çalıştırmak için
import program as program  # program.py dosyasındaki özel fonksiyonları kullanmak için
from PIL import Image, ImageTk  # PIL (Pillow): Görüntü işlemleri için
import face_recognition  # face_recognition: Yüz tanıma işlemleri için

# Kamera akışını göstermek ve yönetmek için ana sınıf
class CameraWindow:
    def __init__(self, root):
        # Ana pencere ayarları
        self.root = root
        self.root.title("Çoklu Kamera Akışı")  # Pencere başlığı
        self.root.state("zoomed")  # Pencereyi tam ekran yapar
        self.root.configure(bg='#87cfdb')  # Ana pencerenin arka plan rengini belirler
        self.root.iconbitmap("logo.ico")  # Pencerenin simgesi

        # Kamera verilerini ve ilişkili işlemleri saklamak için gerekli değişkenler
        self.frame_queues = {}  # Kamera görüntü kuyrukları
        self.cameras = []  # Kameraların bilgileri
        self.camera_labels = []  # Kameralar için oluşturulan label'lar

        # Menü çubuğu oluşturma
        self.menubar = tk.Menu(self.root)
        self.root.config(menu=self.menubar)

        # Kamera menüsü oluşturma
        self.camera_menu = tk.Menu(self.menubar, tearoff=0)  # "Kamera" menüsü
        self.menubar.add_cascade(label="Kamera", menu=self.camera_menu)  # Menü çubuğuna eklenir
        self.camera_menu.add_command(label="Kamera Ekle", command=self.add_camera)  # Kamera ekleme seçeneği
        self.camera_menu.add_command(label="Kamera Çıkar", command=self.remove_camera)  # Kamera çıkarma seçeneği

        # Kameraların görüntülerini göstermek için çerçeve
        self.canvas_frame = tk.Frame(self.root, bg='#87cfdb')  # Çerçeve arka plan rengi
        self.canvas_frame.pack(expand=True, fill=tk.BOTH)  # Çerçeve boyutunu genişletir

    # Yeni bir kamera eklemek için
    def add_camera(self):
        # Mevcut kameraları listeleme
        available_cameras = self.get_available_cameras()

        # Kamera seçimi için yeni pencere oluşturma
        camera_window = tk.Toplevel(self.root)
        camera_window.title("Kamera Seç")  # Kamera seç penceresi başlığı
        camera_window.configure(bg='#87cfdb')  # Arka plan rengi

        # Kamera seç penceresini ekranın ortasına yerleştirme
        self.center_window(camera_window, 400, 300)

        # Kullanıcıya kamera seçimi için bilgi veren metin
        label = tk.Label(camera_window, text="Bir kamera seçin:", bg='#87cfdb')
        label.pack(pady=10)

        # Kamera listesini gösteren liste kutusu
        camera_listbox = tk.Listbox(camera_window, bg='#87cfdb', selectbackground='#add8e6', height=5)
        for i in available_cameras:
            camera_listbox.insert(tk.END, f"Kamera {i + 1}")  # Mevcut kameraları ekler
        camera_listbox.pack(pady=10)
```

kameralar.py

```
# Kullanıcıya kamera indexi girmesi için bir etiket (textbox_label) oluşturulur.
        textbox_label = tk.Label(camera_window, text="Kamera Indexi Girin (Opsiyonel):", bg='#87cfdb')
        textbox_label.pack(pady=5)

        # Kullanıcıya index girebilmesi için bir metin kutusu (Entry) oluşturulur.
        camera_textbox = tk.Entry(camera_window, bg='#ffffff')
        camera_textbox.pack(pady=5)

        # on_confirm fonksiyonu, kullanıcı onay butonuna tıkladığında çalışacak olan fonksiyondur.
        def on_confirm():
            # Liste kutusundan seçilen öğe alınır.
            selected_index = camera_listbox.curselection()

            # Metin kutusundan (textbox) girilen değer alınır.
            entered_text = camera_textbox.get()

            # Eğer kullanıcı listeden bir kamera seçmişse, seçilen kameranın indexi alınır.
            if selected_index:
                camera_index = selected_index[0]  # Seçilen kamerayı al
                camera_window.destroy()  # Kamera seçim penceresi kapatılır
                self.show_location_and_notify_settings(camera_index)  # Seçilen kamerayla ilgili ayarlara geçilir
            # Eğer metin kutusuna bir sayı girildiyse
            elif entered_text.isdigit():
                # Girilen metin sayıya dönüştürülür.
                camera_index = int(entered_text)

                # Girilen kamera indexi geçerli bir indexse, ilgili işlemler yapılır.
                if 0 <= camera_index < len(available_cameras):
                    camera_window.destroy()  # Kamera seçim penceresi kapatılır
                    self.show_location_and_notify_settings(camera_index)  # Seçilen kamerayla ilgili ayarlara geçilir
                else:
                    # Eğer girilen index geçerli değilse, kullanıcıya hata mesajı gösterilir.
                    messagebox.showinfo("Hata", "Girilen kamera indexi geçerli değil.")
            else:
                # Ne listeden bir seçim yapılmış ne de geçerli bir index girilmişse, hata mesajı gösterilir.
                messagebox.showinfo("Hata", "Lütfen bir kamera seçin veya geçerli bir kamera indexi girin.")

        # "Onayla" butonu oluşturulur ve on_confirm fonksiyonu ile ilişkilendirilir.
        confirm_button = tk.Button(camera_window, text="Onayla", command=on_confirm, bg='#87cfdb',
activebackground='#87cfdb')
        confirm_button.pack(pady=10)

        # Kullanıcı, kamera seçim penceresini kapatana kadar beklenir.
        self.root.wait_window(camera_window)

    def show_location_and_notify_settings(self, camera_index):
        # Kamera ve bildirim ayarlarını yapmak için yeni bir pencere (camera_window) oluşturulur.
        camera_window = tk.Toplevel(self.root)

        # Pencereye başlık eklenir.
        camera_window.title("Mekan ve Bildirim Ayarları")

        # Pencerenin arka plan rengi '#87cfdb' olarak belirlenir.
        camera_window.configure(bg='#87cfdb')  # Arka plan rengi

        # center_window fonksiyonu çağrılarak pencere ekranın ortasında başlatılır.
        self.center_window(camera_window, 400, 300)

        # Kullanıcıya kameranın kapsadığı mekanı girmesi için bir etiket (location_label) eklenir.
        location_label = tk.Label(camera_window, text="Kameranın kapsadığı mekanı giriniz:", bg='#87cfdb')
        location_label.pack(pady=10)

        # Kullanıcıya mekan adını girmesi için bir metin kutusu (Entry) eklenir.
        location_entry = tk.Entry(camera_window)
        location_entry.pack(pady=5)

        # Bildirim alıp almayacağına dair bir onay kutusu (checkbox) oluşturulur.
        notify_var = tk.BooleanVar(value=False)  # Varsayılan olarak False, yani bildirim istemiyor.

        # Kullanıcıya bildirim almak isteyip istemediğini soran onay kutusu eklenir.
        notify_checkbox = tk.Checkbutton(
            camera_window,
            text="Bu mekanda biri tespit edildiğinde size bildirim gönderilsin mi?",
            variable=notify_var,
            bg='#87cfdb'
        )
        notify_checkbox.pack(pady=10)
```

main.py

```
# Ayarları onaylayan buton
        def confirm():
            location = location_entry.get()  # Kullanıcının girdiği mekan adı
            notify = notify_var.get()  # Bildirim ayarı
            camera_window.destroy()  # Ayar penceresini kapatır

            if location:  # Eğer mekan adı girilmişse
                # Kamera bilgilerini kaydetme
                self.cameras.append((camera_index, location, notify))

                # Kamera görüntüsü için yeni bir label oluşturma
                label = tk.Label(self.canvas_frame, bg='#87cfdb')
                label.grid(row=len(self.cameras)//4, column=len(self.cameras)%4, padx=10, pady=10)
                self.camera_labels.append(label)

                # Kamera için bir işlem kuyruğu oluştur ve başlat
                self.frame_queues[camera_index] = multiprocessing.Queue()
                camera_process = multiprocessing.Process(
                    target=program.run_task,
                    args=(self.frame_queues[camera_index], location, camera_index, notify)
                )
                camera_process.start()

                # Kamera akışını güncelleme
                self.update_frames()

        # Onay butonu
        confirm_button = tk.Button(camera_window, text="Onayla", command=confirm, bg='#87cfdb', activebackground='#87cfdb')
        confirm_button.pack(pady=10)

        # Mekan ayarları penceresini bekler
        self.root.wait_window(camera_window)

    # Kameraları kaldırmak için
    def remove_camera(self):
        if not self.cameras:  # Eğer kayıtlı kamera yoksa
            messagebox.showinfo("Kamera Yok", "Silinecek kamera yok.")  # Kullanıcıya bilgi ver
            return

        # Kullanıcıdan silmek istediği kamerayı seçmesini ister
        camera_to_remove = simpledialog.askinteger(
            "Kamera Seç",
            "Silmek için bir kamera seçin:\n" +
            "\n".join([f"{i + 1}: {self.cameras[i][1]}" for i in range(len(self.cameras))]),  # Kamera listesi
            parent=self.root
        )
        # Geçerli bir seçim yapıldıysa
        if camera_to_remove is not None and 1 <= camera_to_remove <= len(self.cameras):
            camera_to_remove -= 1  # Kullanıcı dostu seçim için 1 azaltılır (0 tabanlı indeksleme için)
            self.cameras.pop(camera_to_remove)  # Seçilen kamerayı listeden çıkarır
            self.camera_labels[camera_to_remove].destroy()  # İlgili label'ı yok eder
            self.camera_labels.pop(camera_to_remove)  # Label'ı listeden çıkarır
            self.update_frames()  # Kamera akışını günceller
```

main.py

```python
# Kameraların görüntülerini düzenli olarak güncellemek için
    def update_frames(self):
        for i, (camera_index) in enumerate(self.cameras):  # Kameralar üzerinde iterasyon
            try:
                # Kamera kuyruğundan bir çerçeve al
                frame = self.frame_queues[camera_index].get_nowait()
                # BGR'den RGB'ye dönüştür
                frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                # Çerçeveyi bir görüntüye çevir
                image = Image.fromarray(frame)
                # Tkinter için görüntüyü uygun bir formata dönüştür
                photo = ImageTk.PhotoImage(image=image)
                # Kamera label'ını güncelle
                self.camera_labels[i].config(image=photo)
                self.camera_labels[i].image = photo
            except:
                pass  # Kuyrukta görüntü yoksa bir şey yapma
        # Güncellemeleri belirli bir süre sonra tekrar çağır
        self.root.after(10, self.update_frames)

    # Mevcut kameraları algılamak için
    def get_available_cameras(self):
        available_cameras = []  # Algılanan kameraların listesi
        i = 0  # Kamera indekslerini kontrol etmek için başlangıç
        while True:
            cap = cv2.VideoCapture(i)  # Kamerayı açmayı dener
            if cap.isOpened():  # Eğer kamera açıksa
                available_cameras.append(i)  # Kamerayı listeye ekler
            else:
                cap.release()  # Kamera açılamıyorsa döngüyü sonlandırır
                break
            cap.release()  # Kamera açıldıktan sonra serbest bırakılır
            i += 1  # Bir sonraki kamerayı kontrol et
        return available_cameras  # Algılanan kameraları döndür

    # Verilen pencereyi ekranın ortasına yerleştirmek için
    def center_window(self, window, width, height):
        screen_width = self.root.winfo_screenwidth()  # Ekran genişliğini alır
        screen_height = self.root.winfo_screenheight()  # Ekran yüksekliğini alır
        x = (screen_width - width) // 2  # X koordinatını hesaplar
        y = (screen_height - height) // 2  # Y koordinatını hesaplar
        window.geometry(f"{width}x{height}+{x}+{y}")  # Pencereyi yerleştirir

# Program ana kısmı
if __name__ == "__main__":
    multiprocessing.freeze_support()  # Windows'ta çoklu işlem başlatma için gerekli
    root = tk.Tk()  # Tkinter ana pencere oluşturma
    window = CameraWindow(root)  # Ana pencereyi CameraWindow sınıfı ile başlatma
    root.mainloop()  # Ana döngüyü başlatır (uygulama çalışır)
```

program.py:

This file processes the image captured from the camera at the specified index and sends it to the “kameralar.py” file. Face recognition, object detection, and image processing operations are performed through this file.

main.py

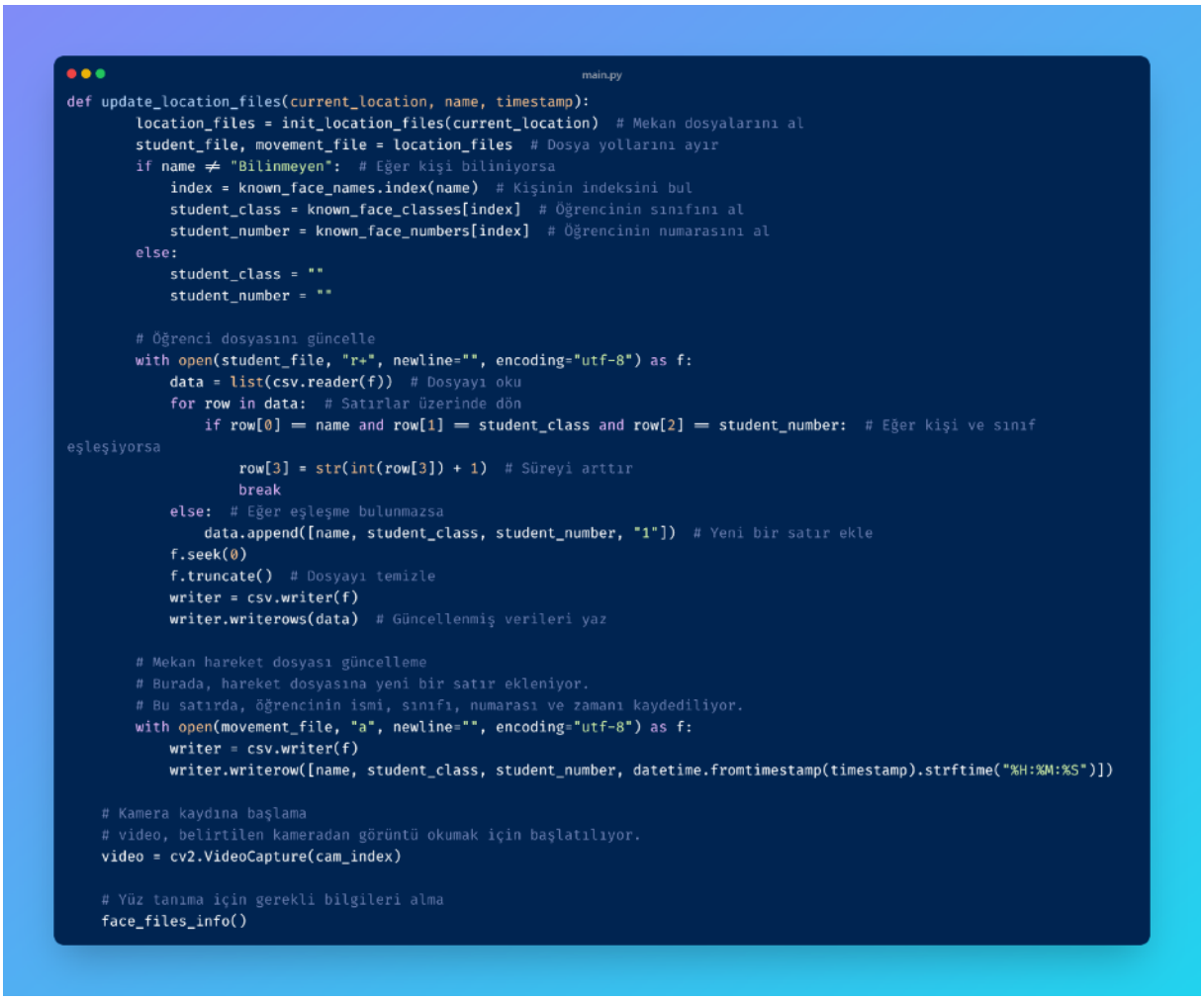
```python
import cv2  # OpenCV kütüphanesini import eder, video ve görüntü işleme için kullanılır.
import multiprocessing  # Çoklu işlem yapabilmek için multiprocessing modülünü kullanır.
import face_recognition  # Yüz tanıma işlemleri için face_recognition kütüphanesini import eder.
import numpy as np  # Numpy kütüphanesini import eder, matris ve matematiksel işlemler için kullanılır.
import os  # Dosya ve dizin işlemleri için os kütüphanesini import eder.
import csv  # CSV dosyalarıyla işlem yapabilmek için csv kütüphanesini import eder.
import time  # Zamanla ilgili işlemler için time kütüphanesini import eder.
from pathlib import Path  # Dosya ve klasör yolu işlemleri için Path kütüphanesini kullanır.
from datetime import datetime  # Tarih ve saat işlemleri için datetime modülünü import eder.
from ultralytics import YOLO  # YOLO (You Only Look Once) modelini kullanmak için ultralytics kütüphanesini import eder.
from plyer import notification  # Bildirim göndermek için plyer kütüphanesini kullanır.
from ultralytics.engine.results import Results  # YOLO sonuçlarını almak için Results sınıfını import eder.

# Ana klasörleri oluştur
base_folder = "Veriler"  # Veriler için ana klasör
place_folder = "Veriler\\Mekanlar"  # Mekanlar için klasör
student_folder = "Veriler\\Öğrenciler"  # Öğrenciler için klasör
os.makedirs(base_folder, exist_ok=True)  # Ana klasörü oluştur
os.makedirs(place_folder, exist_ok=True)  # Mekanlar klasörünü oluştur
os.makedirs(student_folder, exist_ok=True)  # Öğrenciler klasörünü oluştur
detected = []  # Algılanan yüzlerin listesini tutan bir değişken

# Kişilerin son kaydedilme zamanını takip etmek için bir sözlük
last_recorded_time = {}

# YOLO modelini yükle
model = YOLO("model.pt")  # YOLO modelini "model.pt" dosyasından yükle

# Yüz verilerini kaydedilen klasörden al
known_face_encodings = []  # Yüz verilerinin kodlarını tutar
known_face_names = []  # Yüzlere ait isimleri tutar
known_face_classes = []  # Yüzlerin sınıf bilgilerini tutar
known_face_numbers = []  # Yüzlerin numaralarını tutar

output_folder = "data/yuzler"  # Yüz verilerinin bulunduğu klasör
face_files = os.listdir(output_folder)  # Yüz verilerinin bulunduğu klasördeki dosyaların listesi

# Yüz verilerini yüklemek için fonksiyon
def face_files_info():
    for file in face_files:
        if file.endswith(".jpg"):  # Yalnızca ".jpg" uzantılı dosyaları işle
            # Dosya adını alt çizgilerle ayırarak öğrenci bilgilerini çıkar
            file_info = file.split('_')
            name = file_info[0]  # Öğrenci adı
            student_class = file_info[1]  # Öğrenci sınıfı
            student_number = file_info[2].split(".")[0]  # Öğrenci numarası

            # Yüz verisini yükle
            image_path = os.path.join(output_folder, file)  # Yüz verisinin tam yolu
            image = cv2.imread(image_path)  # Resmi oku
            rgb_image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  # Resmi RGB'ye dönüştür
            face_encodings = face_recognition.face_encodings(rgb_image)  # Yüzün kodlamalarını al

            if face_encodings:  # Eğer yüz bulunursa
                known_face_encodings.append(face_encodings[0])  # Yüz kodlamasını ekle
                known_face_names.append(name)  # Öğrenci adını ekle
                known_face_classes.append(student_class)  # Sınıf bilgisini ekle
                known_face_numbers.append(student_number)  # Öğrenci numarasını ekle
            else:
                print(f"Yüz bulunamadı: {file}")  # Eğer yüz bulunamazsa, hata mesajı ver
```

main.py

```python
# Kameradaki kareyi alıp queue üzerinden geri gönderen fonksiyon
def run_task(frame_queue, current_location, cam_index, notify):
    # Potansiyel İhlal Unsurları klasör ve dosya işlemleri
    def init_violation_files():
        today = datetime.now().strftime("%Y-%m-%d")  # Bugünün tarihi
        violations_folder = os.path.join(base_folder, "Potansiyel_Ihlal_Unsurlari", today)  # İhlaller için klasör
        os.makedirs(violations_folder, exist_ok=True)  # İhlal klasörünü oluştur

        violation_file = os.path.join(violations_folder, "ihlal_unsurlari.csv")  # İhlal dosyası
        if not os.path.isfile(violation_file):  # Dosya yoksa, başlıkları ile oluştur
            with open(violation_file, "w", newline="", encoding="utf-8") as f:
                writer = csv.writer(f)
                writer.writerow(["Tarih", "Unsurlar", "Mekan", "Zaman"])

        return violation_file  # İhlal dosyasının yolunu döndür

    # İhlal unsurlarını kaydetme fonksiyonu
    def log_violation_item(item, location, timestamp):
        violation_file = init_violation_files()  # İhlal dosyasını al

        with open(violation_file, "a", newline="", encoding="utf-8") as f:  # Dosyaya ekle
            writer = csv.writer(f)
            date = datetime.fromtimestamp(timestamp).strftime("%Y-%m-%d")  # Zamanı tarih formatında al
            time_str = datetime.fromtimestamp(timestamp).strftime("%H:%M:%S")  # Zamanı saat: dakika: saniye formatında al
            writer.writerow([date, item, location, time_str])  # Verileri dosyaya yaz

    # Zaman kontrol fonksiyonu
    def can_record_person(name):
        current_time = time.time()  # Şu anki zamanı al
        if name in last_recorded_time:  # Eğer kişi daha önce kaydedildiyse
            elapsed_time = current_time - last_recorded_time[name]  # Geçen süreyi hesapla
            if elapsed_time < 60:  # Eğer 60 saniye geçmemişse
                return False  # Kayıt yapma
        # Kayıt yapılabilir, zaman güncellenir
        last_recorded_time[name] = current_time  # Kişinin son kaydedilme zamanını güncelle
        return True  # Kayıt yapılabilir

    # Kişi klasörlerini ve dosyalarını oluşturma
    def init_person_files(person_name):
        today = datetime.now().strftime("%Y-%m-%d")  # Bugünün tarihi
        person_folder = os.path.join(student_folder, person_name, today)  # Kişiye özel klasör yolu
        os.makedirs(person_folder, exist_ok=True)  # Klasörü oluştur

        # Mekan dosyası
        location_file = os.path.join(person_folder, "mekan.csv")
        if not os.path.isfile(location_file):  # Dosya yoksa, başlıkları ile oluştur
            with open(location_file, "w", newline="", encoding="utf-8") as f:
                writer = csv.writer(f)
                writer.writerow(["Mekan", "Sınıfı", "Numarası", "Süre (dk)"])

        # Çevre dosyası
        environment_file = os.path.join(person_folder, "cevre.csv")
        if not os.path.isfile(environment_file):  # Dosya yoksa, başlıkları ile oluştur
            with open(environment_file, "w", newline="", encoding="utf-8") as f:
                writer = csv.writer(f)
                writer.writerow(["Öğrenci", "Sınıfı", "Numarası", "Süre (dk)"])
```

main.py

```
# Hareket dosyası
        movement_file = os.path.join(person_folder, "hareket.csv")
        if not os.path.isfile(movement_file):  # Dosya yoksa, başlıkları ile oluştur
            with open(movement_file, "w", newline="", encoding="utf-8") as f:
                writer = csv.writer(f)
                writer.writerow(["Mekan", "Sınıfı", "Numarası", "Zaman"])

        return location_file, environment_file, movement_file  # Dosya yollarını döndür

    # Mekan klasörlerini ve günlük dosyalarını oluşturma
    def init_location_files(location_name):
        today = datetime.now().strftime("%Y-%m-%d")  # Bugünün tarihi
        location_folder = os.path.join(place_folder, location_name, today)  # Mekan adıyla klasör yolu
        os.makedirs(location_folder, exist_ok=True)  # Klasörü oluştur

        # Mekan öğrenci dosyası
        student_file = os.path.join(location_folder, "ogrenci.csv")
        if not os.path.isfile(student_file):  # Dosya yoksa, başlıkları ile oluştur
            with open(student_file, "w", newline="", encoding="utf-8") as f:
                writer = csv.writer(f)
                writer.writerow(["Öğrenci", "Sınıfı", "Numarası", "Süre (dk)"])

        # Mekan hareket dosyası
        movement_file = os.path.join(location_folder, "hareket.csv")
        if not os.path.isfile(movement_file):  # Dosya yoksa, başlıkları ile oluştur
            with open(movement_file, "w", newline="", encoding="utf-8") as f:
                writer = csv.writer(f)
                writer.writerow(["Öğrenci", "Sınıfı", "Numarası", "Zaman"])

        return student_file, movement_file  # Dosya yollarını döndür

    # Veriyi dosyalara yazma
    def update_person_files(name, current_location, timestamp, detected_people):
        person_files = init_person_files(name)  # Kişiye ait dosya yollarını al
        location_file, environment_file, movement_file = person_files  # Dosya yollarını ayır

        index = known_face_names.index(name)  # Yüzün adını kullanarak indeksini bul
        student_class = known_face_classes[index]  # Öğrencinin sınıfını al
        student_number = known_face_numbers[index]  # Öğrencinin numarasını al

        # Mekan dosyası güncelleme
        with open(location_file, "r+", newline="", encoding="utf-8") as f:
            data = list(csv.reader(f))  # Dosyayı oku
            for row in data:  # Satırlar üzerinde dön
                if row[0] == current_location and row[1] == student_class and row[2] == student_number:  # Eğer kişi ve
mekan eşleşiyorsa
                    row[3] = str(int(row[3]) + 1)  # Süreyi bir dakika arttır
                    break
            else:  # Eğer eşleşme bulunmazsa
                data.append([current_location, student_class, student_number, "1"])  # Yeni bir satır ekle
            f.seek(0)
            f.truncate()  # Dosyayı temizle
            writer = csv.writer(f)
            writer.writerows(data)  # Güncellenmiş verileri yaz

        # Çevre dosyası güncelleme
        with open(environment_file, "r+", newline="", encoding="utf-8") as f:
            data = list(csv.reader(f))  # Dosyayı oku
            for person in detected_people:  # Algılanan kişiler üzerinde dön
                p_index = known_face_names.index(person)  # Kişinin indeksini bul
                p_class = known_face_classes[p_index]  # Sınıfını al
                p_number = known_face_numbers[p_index]  # Numarasını al
                for row in data:  # Satırlar üzerinde dön
                    if row[0] == person and row[1] == p_class and row[2] == p_number:  # Eğer kişi ve sınıf eşleşiyorsa
                        row[3] = str(int(row[3]) + 1)  # Süreyi arttır
                        break
                else:  # Eğer eşleşme bulunmazsa
                    data.append([person, p_class, p_number, "1"])  # Yeni bir satır ekle
            f.seek(0)
            f.truncate()  # Dosyayı temizle
            writer = csv.writer(f)
            writer.writerows(data)  # Güncellenmiş verileri yaz

        # Hareket dosyası güncelleme
        with open(movement_file, "a", newline="", encoding="utf-8") as f:
            writer = csv.writer(f)
            writer.writerow([current_location, student_class, student_number,
datetime.fromtimestamp(timestamp).strftime("%H:%M:%S")])
```


main.py
def update_location_files(current_location, name, timestamp):
location_files = init_location_files(current_location) # Mekan dosyalarını al
student_file, movement_file = location_files # Dosya yollarını ayır
if name ≠ "Bilinmeyen": # Eğer kişi biliniyorsa
index = known_face_names.index(name) # Kişinin indeksini bul
student_class = known_face_classes[index] # Öğrencinin sınıfını al
student_number = known_face_numbers[index] # Öğrencinin numarasını al
else:
student_class = ""
student_number = ""
# Öğrenci dosyasını güncelle
with open(student_file, "r+", newline="", encoding="utf-8") as f:
data = list(csv.reader(f)) # Dosyayı oku
for row in data: # Satırlar üzerinde dön
if row[0] == name and row[1] == student_class and row[2] == student_number: # Eğer kişi ve sınıf
eşleşiyorsa
row[3] = str(int(row[3]) + 1) # Süreyi arttır
break
else: # Eğer eşleşme bulunmazsa
data.append([name, student_class, student_number, "1"]) # Yeni bir satır ekle
f.seek(0)
f.truncate() # Dosyayı temizle
writer = csv.writer(f)
writer.writerows(data) # Güncellenmiş verileri yaz
Mekan hareket dosyası güncelleme
Burada, hareket dosyasına yeni bir satır ekleniyor.
Bu satırda, öğrencinin ismi, sınıfı, numarası ve zamanı kaydediliyor.
with open(movement_file, "a", newline="", encoding="utf-8") as f:
writer = csv.writer(f)
writer.writerow([name, student_class, student_number, datetime.fromtimestamp(timestamp).strftime("%H:%M:%S")])
# Kamera kaydına başlama
# video, belirtilen kameradan görüntü okumak için başlatılıyor.
video = cv2.VideoCapture(cam_index)
# Yüz tanıma için gerekli bilgileri alma
face_files_info()

main.py

```
# Sonsuz bir döngü başlatıyoruz, bu döngüde kameradan her frame alınacak.
    while True:
        ret, frame = video.read()  # Frame okuyarak alıyoruz.

        # Eğer kameradan görüntü alınamadıysa döngüden çıkıyoruz
        if not ret:
            print("Kameradan görüntü alınamadı.")
            break

        # Model sonuçlarını alıyoruz, yani kameradaki görüntüyü modelle analiz ediyoruz
        results = model.predict(source=frame, conf=0.5)

        # BGR formatındaki görüntüyü RGB'ye dönüştürme
        # OpenCV'nin kullandığı BGR formatını, yüz tanıma için kullanılan RGB formatına dönüştürüyoruz.
        rgb_frame = np.ascontiguousarray(frame[:, :, ::-1])

        # Yüz tespiti yapıyoruz, rgb_frame üzerinde yüzlerin yerlerini buluyoruz
        face_locations = face_recognition.face_locations(rgb_frame)
        face_encodings = face_recognition.face_encodings(rgb_frame, face_locations)

        # Algılanan kişiler listesi
        detected_people = []  # Bu döngüde algılanan kişiler

        # Yüzlerin bulunduğu her bölgeyi (kutuları) ve yüzleri analiz ediyoruz
        for (top, right, bottom, left), face_encoding in zip(face_locations, face_encodings):
            # Tanınan yüzler ile karşılaştırma yapıyoruz
            matches = face_recognition.compare_faces(known_face_encodings, face_encoding, tolerance=0.5)
            name = "Bilinmeyen"  # Eğer tanınan bir yüz yoksa 'Bilinmeyen' olarak ayarlıyoruz

            # Eğer eşleşme varsa, doğru kişinin ismini alıyoruz
        if True in matches:
            first_match_index = matches.index(True)
            name = known_face_names[first_match_index]

            # Tanınan yüzü ekranda gösteriyoruz
            cv2.putText(frame, name, (left, top - 15), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2)
            cv2.rectangle(frame, (left, top), (right, bottom), (50, 50, 255), 2)  # Yüz etrafına dikdörtgen çiziyoruz

            # Kişi tanındığında ve kayıt için uygun bir zaman varsa, bilgileri kaydediyoruz
            if name != "Bilinmeyen" and can_record_person(name):
                timestamp = time.time()  # Zaman damgasını alıyoruz
                detected_people.append(name)  # Algılanan kişiyi ekliyoruz

                # Bildirim gönderme
                if notify:
                    notification.notify(
                        title="Öğrenci Tespiti",
                        message=f'{name} isimli öğrenci, {current_location} bölgesinde tespit
edilmiştir.\n{datetime.fromtimestamp(timestamp).strftime("%H:%M:%S")}',
                        timeout=10  # Bildirimin ne kadar süre görüntüleneceğini ayarlıyoruz
                    )

                # Kişiyi ve hareketi güncelliyoruz
                update_person_files(name, current_location, timestamp, detected_people)
                update_location_files(current_location, name, timestamp)

            # Frame'i bir sonraki işlem için kuyruk sistemine gönderiyoruz
            frame_queue.put(frame)
```

main.py

```python
# Modelle işlenmiş sonucu ekranda gösteriyoruz
        for result in results:
            for box in result.boxes:
                x1, y1, x2, y2 = map(int, box.xyxy[0])  # Kutuların koordinatlarını alıyoruz
                conf = box.conf[0]  # Güven skoru
                cls = int(box.cls[0])  # Sınıf kimliğini alıyoruz

                # Eğer model "Telefon" tespit ederse
                if model.names[cls] == "Telefon" and conf > 0.5:
                    cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)  # Yeşil çerçeve çiziyoruz
                    cv2.putText(frame, "Telefon", (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

                    # Telefonu kaydediyoruz
                    timestamp = time.time()
                    log_violation_item("Telefon", current_location, timestamp)

                # Eğer model "Sigara" tespit ederse
                elif model.names[cls] == "Sigara" and conf > 0.6:
                    cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 255), 2)  # Kırmızı çerçeve çiziyoruz
                    cv2.putText(frame, "Sigara", (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)

                    # Sigara kaydını yapıyoruz
                    timestamp = time.time()
                    log_violation_item("Sigara", current_location, timestamp)

                # Eğer model "Bıçak" tespit ederse
                elif model.names[cls] == "Bıçak":
                    cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 255), 2)  # Kırmızı çerçeve çiziyoruz
                    cv2.putText(frame, "Bıçak", (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)

                    # Bıçak kaydını yapıyoruz
                    timestamp = time.time()
                    log_violation_item("Bıçak", current_location, timestamp)

            # Frame'i tekrar kuyruk sistemine gönderiyoruz
            frame_queue.put(frame)

        # Video kaydını sonlandırıyoruz
        video.release()
        cv2.destroyAllWindows()

    # Ana fonksiyon çalıştırıldığında
if __name__ == "__main__":
    # Parametreleri tanımlıyoruz
    frame_queue = multiprocessing.Queue()  # Frame kuyruk oluşturma
    current_location = "Kantin"  # Örnek bir mekan adı
    cam_index = 0  # Kameranın index'i
    notify = False  # Bildirim açma
    known_face_encodings = []  # Yüz tanımlamaları için örnek liste
    known_face_names = []  # Tanımlı yüzlerin isimleri
    known_face_classes = []  # Öğrencilerin sınıf bilgisi
    known_face_numbers = []  # Öğrencilerin numaraları

        # Process başlatma
    thread = multiprocessing.Process(target=run_task, args=(frame_queue, current_location, cam_index, notify))
    thread.start()
```

egitim_programi.py:

This file captures photos of individuals' faces through the camera and saves image files to train the program.

```python
import cv2
import face_recognition
import os
from PIL import Image
import numpy as np
import time

def train_face(name, Class, number, cam):
    # Video kaynağını başlat (kamera bağlantısını aç)
    video = cv2.VideoCapture(cam)

    # Yüz verilerini kaydetmek için bir klasör oluştur
    output_folder = "data/yuzler"
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)  # Klasör yoksa oluştur
        print(f"{output_folder} klasörü oluşturuldu.")

    # Eğitim verisi için yüz kaydetme
    i = 0  # Fotoğraf sayısı
    start_time = time.time()  # Zamanı başlat (fotoğraf çekmek için süreyi takip etmek için)
    countdown_start_time = None  # Geri sayımın başladığı zaman (ilk başta None)

    while True:
        ret, frame = video.read()  # Kameradan bir görüntü al
        height, width, _ = frame.shape  # Görüntünün boyutlarını al

        # Ekranın sağ üst köşesinde geri sayım gösterecek konum hesapla
        x = width - 50  # Sağdan 50 piksel
        y = 50  # Üstten 50 piksel
        if not ret:
            print("Kameradan görüntü alınamadı.")  # Kamera görüntüsü alınamazsa hata mesajı
            break

        # Görüntüyü RGB formatına dönüştür (OpenCV BGR kullanır, biz RGB formatına ihtiyacımız var)
        rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

        # Yüz algılama (face_recognition kütüphanesi kullanılarak)
        face_locations = face_recognition.face_locations(rgb_frame)

        # Yüz algılandığında, etrafına çerçeve koy
        if face_locations:
            for (top, right, bottom, left) in face_locations:
                # Yüzün etrafına bir dikdörtgen çizin
                cv2.rectangle(frame, (left, top), (right, bottom), (0, 255, 0), 2)
                # Yüz algılandığını belirten bir yazı ekle
                cv2.putText(frame, "Yuz algilandi!", (left, top - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 0, 0), 2)

        # Fotoğraf çekme işlemi
        elapsed_time = time.time() - start_time  # Geçen zamanı hesapla
        if elapsed_time >= 3 and i < 5:
            # 3 saniye geçtiğinde fotoğraf çekme işlemi başlat
            print("Bir sonraki fotoğraf için:")
            if countdown_start_time is None:
                countdown_start_time = time.time()  # Geri sayımı başlat
```

main.py

```python
# Kalan süreyi hesapla
            remaining_time = 3 - (time.time() - countdown_start_time)
            remaining_time_text = f"{int(remaining_time)}"  # Kalan süreyi metin olarak formatla

            # Geri sayım metnini sağ üst köşeye yerleştir
            cv2.putText(frame, remaining_time_text, (x, y), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 0, 0), 2)

            # Fotoğraf çekme işlemi
            if remaining_time <= 0:
                # Dosya adı formatı: "ad_sınıf_numara_index.jpg"
                filename = f"{name}_{Class}_{number}_{i}.jpg"
                file_path = os.path.join(output_folder, filename)

                try:
                    # OpenCV ile BGR'yi RGB'ye çevir, sonra Pillow ile kaydet
                    pil_image = Image.fromarray(rgb_frame)  # numpy array'ini Pillow görüntüsüne çevir
                    pil_image.save(file_path)  # Fotoğrafı kaydet
                    print(f"Yüz kaydedildi: {file_path}")
                except Exception as e:
                    print(f"Fotoğraf kaydedilemedi: {e}")  # Hata mesajı yazdır

                i += 1  # Fotoğraf sayısını artır
                start_time = time.time()  # Zamanı tekrar başlat
                countdown_start_time = None  # Geri sayımı sıfırla

        # Görüntüyü ekranda göster
        cv2.imshow("Yüz Ekleme Sistemi", frame)

        # 5 fotoğraf kaydedildiyse çık
        if i >= 5:
            print("Eğitim verileri kaydedildi.")
            break

        k = cv2.waitKey(1)  # 1 ms bekle, klavye tuşuna basılırsa devam et
        if k == ord("q"):  # "q" tuşuna basıldığında çık
            break

    # Video kaydını sonlandır
    video.release()
    cv2.destroyAllWindows()

# Eğer bu dosya doğrudan çalıştırılırsa, kullanıcıdan bilgi al
if __name__ == "__main__":
    name = input("Kişinin ismini ve soyismini giriniz: ")  # Kullanıcıdan isim bilgisi al
    Class = input("Kişinin sınıfını giriniz: ")  # Kullanıcıdan sınıf bilgisi al
    number = input("Kişinin numarasını giriniz: ")  # Kullanıcıdan numara bilgisi al
    train_face(name, Class, number, 0)  # Yüz kaydetme fonksiyonunu başlat (0, varsayılan kamera)
```

yuz_ekle.py:

This file provides the interface that requests the information of the individuals whose data will be saved for program training. Once all the data is entered, it calls the “egitim_programi.py” file.

main.py

```python
import tkinter as tk
from tkinter import simpledialog
from egitim_programi import train_face  # egitim_programi.py dosyasından train_face fonksiyonunu import et
import cv2

# Ana pencereyi oluştur
window = tk.Tk()  # Tkinter ana penceresi
window.title("Yüz Ekleme Sistemi")  # Pencerenin başlığı
window.geometry("300×300")  # Pencerenin boyutlarını ayarla
window.iconbitmap("logo.ico")  # Pencerenin ikonunu ayarla

# Ekran genişliği ve yüksekliği alınır
screen_width = window.winfo_screenwidth()
screen_height = window.winfo_screenheight()

# Pencerenin ekranın ortasında konumlanması için hesaplama yapılır
x = (screen_width - 300) // 2
y = (screen_height - 300) // 2
window.geometry(f"300×300+{x}+{y}")  # Pencereyi belirtilen konuma taşı

# Kullanıcıdan bilgi almak için etiketler ve giriş kutuları oluştur
label_name = tk.Label(window, text="İsim Soyisim:", bg='#87cfdb', font="Bahnschrift", fg="#2f2f2f")
label_name.pack(pady=5)  # Etiketi pencereye yerleştir
entry_name = tk.Entry(window, bg="#7375be")  # İsim bilgisi için giriş kutusu
entry_name.pack(pady=5)  # Giriş kutusunu pencereye yerleştir

label_class = tk.Label(window, text="Sınıf/Görev:", font="Bahnschrift", bg='#87cfdb', fg="#2f2f2f")
label_class.pack(pady=5)  # Etiketi pencereye yerleştir
entry_class = tk.Entry(window, bg="#7375be")  # Sınıf bilgisi için giriş kutusu
entry_class.pack(pady=5)  # Giriş kutusunu pencereye yerleştir

label_number = tk.Label(window, text="Numara:", font="Bahnschrift", bg='#87cfdb', fg="#2f2f2f")
label_number.pack(pady=5)  # Etiketi pencereye yerleştir
entry_number = tk.Entry(window, bg="#7375be")  # Numara bilgisi için giriş kutusu
entry_number.pack(pady=5)  # Giriş kutusunu pencereye yerleştir

# Kamera seçim fonksiyonu
def select_camera():
    # Kullanılabilir kameraları al
    available_cameras = get_available_cameras()

    # Kullanıcıya kamera seçimi ekranı göster
    camera_index = simpledialog.askinteger(
        "Kamera Seç",  # Pencere başlığı
        "Bir kamera seçin:\n" + "\n".join([f"{i + 1}: Kamera {i + 1}" for i in available_cameras]),  # Kamera listesi
        parent=window  # Ana pencereyi referans al
    )

    # Geçerli bir kamera seçildiyse, kamera indeksini döndür
    if camera_index is not None and 1 ≤ camera_index ≤ len(available_cameras):
        return available_cameras[camera_index - 1]
    else:
        return None  # Geçersiz seçim yapılırsa None döndür

# Kullanılabilir kameraları algılayan fonksiyon
def get_available_cameras():
    available_cameras = []  # Kullanılabilir kamera listesi
    for i in range(5):  # İlk 5 kamera portunu kontrol et
        cap = cv2.VideoCapture(i)  # Kamera bağlantısını dene
        if cap.isOpened():  # Kamera açıksa
            available_cameras.append(i)  # Kamera listesini ekle
        cap.release()  # Kamerayı serbest bırak
    return available_cameras  # Kullanılabilir kameraları döndür

# Pencere arka plan rengini ayarla
window.configure(bg='#87cfdb')
```

main.py

```python
# Eğitim sürecini başlatan fonksiyon
def start_training():
    # Kullanıcıdan alınan bilgileri giriş kutularından al
    name = entry_name.get()  # İsim bilgisi
    class_info = entry_class.get()  # Sınıf bilgisi
    number = entry_number.get()  # Numara bilgisi

    # Kamera seçimi ekranını başlat
    camera_index = select_camera()

    # Eğer geçerli bir kamera seçilmediyse fonksiyonu sonlandır
    if camera_index is None:
        print("Geçerli bir kamera seçilmedi.")  # Hata mesajı
        return

    # Yüz tanıma eğitimini başlat
    train_face(name, class_info, number, camera_index)  # train_face fonksiyonunu çağır

# Eğitim başlatma butonu oluştur
button_start = tk.Button(window, text="Yüz Ekle", command=start_training)  # Buton oluştur
button_start.pack(pady=20)  # Butonu pencereye yerleştir

# Ana döngüyü başlat
window.mainloop()  # Tkinter pencere döngüsünü başlat
```

veriler.py:

This file facilitates access to the ".csv" files kept in the records and displays this data on the screen in a table format.

main.py

```
def view_potential_violation_folders(self):
        """Potansiyel İhlal Unsurları verisini görüntülemek için klasörü ayarlar."""
        violation_folder = os.path.join(self.data_dir, "Potansiyel_Ihlal_Unsurlari")  # Potansiyel ihlal unsurları klasör
yolunu al
        self.show_violation_folders(violation_folder)  # Klasörü göstermek için ilgili fonksiyonu çağır

    def show_violation_folders(self, violation_folder):
        """Potansiyel İhlal Unsurları klasörlerini listeler."""
        self.listbox.delete(0, tk.END)  # Mevcut listeyi temizle

        # Alt klasörleri listele
        violations = self.list_folders(violation_folder)  # Alt klasörleri al
        if not violations:  # Eğer klasör yoksa
            messagebox.showwarning("Uyarı", "Potansiyel İhlal Unsurları klasörleri bulunamadı!")  # Uyarı mesajı göster
            return

        for violation in violations:  # Klasörleri listeye ekle
            self.listbox.insert(tk.END, violation)

        # Listbox tıklama olayını ayarla
        self.listbox.bind("<Double-1>", lambda event: self.show_csv_files_directly(event, violation_folder))
        self.current_folder = violation_folder  # Mevcut klasörü güncelle

    def show_csv_files_directly(self, event, parent_folder_path):
        """Seçilen klasördeki doğrudan CSV dosyalarını listeler."""
        selected_item = self.listbox.get(self.listbox.curselection())  # Seçilen klasörü al
        folder_path = os.path.join(parent_folder_path, selected_item)  # Tam yolunu oluştur

        # CSV dosyalarını listele
        csv_files = self.list_csv_files(folder_path)  # CSV dosyalarını al

        if not csv_files:  # Eğer CSV dosyası yoksa
            messagebox.showwarning("Uyarı", "Bu klasörde herhangi bir CSV dosyası bulunamadı!")  # Uyarı göster
            return

        self.listbox.delete(0, tk.END)  # Mevcut listeyi temizle
        for csv_file in csv_files:  # CSV dosyalarını listeye ekle
            self.listbox.insert(tk.END, csv_file)

        # CSV dosyasını açma fonksiyonunu bağla
        self.listbox.bind("<Double-1>", lambda event: self.open_csv_file(event, folder_path))  # Dosya açma olayını bağla

    def check_folders(self):
        """Veriler klasöründeki gerekli alt klasörlerin varlığını kontrol eder."""
        required_folders = ["Öğrenciler", "Mekanlar", "Potansiyel_Ihlal_Unsurlari"]  # Gerekli klasörlerin listesi
        for folder in required_folders:  # Her bir klasörü kontrol et
            folder_path = os.path.join(self.data_dir, folder)  # Klasör yolunu oluştur
            if not os.path.exists(folder_path):  # Klasör mevcut değilse
                messagebox.showerror("Hata", f"'{folder}' klasörü bulunamadı: {folder_path}")  # Hata mesajı göster
                self.root.quit()  # Uygulamayı kapat
                return

    def view_mechanics(self):
        """Mekanlar verisini görüntülemek için klasörü ayarlar."""
        mechanics_folder = os.path.join(self.data_dir, "Mekanlar")  # Mekanlar klasör yolunu al
        self.show_mechanics_folders(mechanics_folder)  # Klasörü göstermek için ilgili fonksiyonu çağır

    def show_mechanics_folders(self, mechanics_folder):
        """Mekanlar klasörlerini listeler."""
        self.listbox.delete(0, tk.END)  # Mevcut listeyi temizle

        mechanics = self.list_folders(mechanics_folder)  # Mekanlar klasörlerini al
        if not mechanics:  # Eğer mekan klasörü yoksa
            messagebox.showwarning("Uyarı", "Mekanlar klasörleri bulunamadı!")  # Uyarı mesajı göster
            return

        for mechanic in mechanics:  # Mekan klasörlerini listeye ekle
            self.listbox.insert(tk.END, mechanic)

        self.listbox.bind("<Double-1>", lambda event: self.show_date_folders(event, mechanics_folder))  # Tarih klasörlerini
gösterme olayını bağla
        self.current_folder = mechanics_folder  # Mevcut klasörü güncelle
```

main.py

```
def search_in_current_folder(self):
        """Mevcut klasördeki dosya ve klasörlerde arama yapar."""
        search_term = self.search_entry.get()  # Arama terimini al
        if not search_term or not self.current_folder:  # Geçersiz terim veya klasör seçilmemişse
            messagebox.showwarning("Uyarı", "Lütfen geçerli bir arama terimi girin ve geçerli bir klasör seçin.")
    def view_potential_violation_folders(self):
        """Potansiyel İhlal Unsurları verisini görüntülemek için klasörü ayarlar."""
        violation_folder = os.path.join(self.data_dir, "Potansiyel_Ihlal_Unsurlari")  # Potansiyel ihlal unsurları klasör
yolunu al
        self.show_violation_folders(violation_folder)  # Klasörü göstermek için ilgili fonksiyonu çağır

    def show_violation_folders(self, violation_folder):
        """Potansiyel İhlal Unsurları klasörlerini listeler."""
        self.listbox.delete(0, tk.END)  # Mevcut listeyi temizle

        # Alt klasörleri listele
        violations = self.list_folders(violation_folder)  # Alt klasörleri al
        if not violations:  # Eğer klasör yoksa
            messagebox.showwarning("Uyarı", "Potansiyel İhlal Unsurları klasörleri bulunamadı!")  # Uyarı mesajı göster
            return

        for violation in violations:  # Klasörleri listeye ekle
            self.listbox.insert(tk.END, violation)

        # Listbox tıklama olayını ayarla
        self.listbox.bind("<Double-1>", lambda event: self.show_csv_files_directly(event, violation_folder))
        self.current_folder = violation_folder  # Mevcut klasörü güncelle

    def show_csv_files_directly(self, event, parent_folder_path):
        """Seçilen klasördeki doğrudan CSV dosyalarını listeler."""
        selected_item = self.listbox.get(self.listbox.curselection())  # Seçilen klasörü al
        folder_path = os.path.join(parent_folder_path, selected_item)  # Tam yolunu oluştur

        # CSV dosyalarını listele
        csv_files = self.list_csv_files(folder_path)  # CSV dosyalarını al

        if not csv_files:  # Eğer CSV dosyası yoksa
            messagebox.showwarning("Uyarı", "Bu klasörde herhangi bir CSV dosyası bulunamadı!")  # Uyarı göster
            return

        self.listbox.delete(0, tk.END)  # Mevcut listeyi temizle
        for csv_file in csv_files:  # CSV dosyalarını listeye ekle
            self.listbox.insert(tk.END, csv_file)

        # CSV dosyasını açma fonksiyonunu bağla
        self.listbox.bind("<Double-1>", lambda event: self.open_csv_file(event, folder_path))  # Dosya açma olayını bağla

    def check_folders(self):
        """Veriler klasöründeki gerekli alt klasörlerin varlığını kontrol eder."""
        required_folders = ["Öğrenciler", "Mekanlar", "Potansiyel_Ihlal_Unsurlari"]  # Gerekli klasörlerin listesi
        for folder in required_folders:  # Her bir klasörü kontrol et
            folder_path = os.path.join(self.data_dir, folder)  # Klasör yolunu oluştur
            if not os.path.exists(folder_path):  # Klasör mevcut değilse
                messagebox.showerror("Hata", f"'{folder}' klasörü bulunamadı: {folder_path}")  # Hata mesajı göster
                self.root.quit()  # Uygulamayı kapat
                return

    def view_mechanics(self):
        """Mekanlar verisini görüntülemek için klasörü ayarlar."""
        mechanics_folder = os.path.join(self.data_dir, "Mekanlar")  # Mekanlar klasör yolunu al
        self.show_mechanics_folders(mechanics_folder)  # Klasörü göstermek için ilgili fonksiyonu çağır

    def show_mechanics_folders(self, mechanics_folder):
        """Mekanlar klasörlerini listeler."""
        self.listbox.delete(0, tk.END)  # Mevcut listeyi temizle

        mechanics = self.list_folders(mechanics_folder)  # Mekanlar klasörlerini al
        if not mechanics:  # Eğer mekan klasörü yoksa
            messagebox.showwarning("Uyarı", "Mekanlar klasörleri bulunamadı!")  # Uyarı mesajı göster
            return

        for mechanic in mechanics:  # Mekan klasörlerini listeye ekle
            self.listbox.insert(tk.END, mechanic)

        self.listbox.bind("<Double-1>", lambda event: self.show_date_folders(event, mechanics_folder))  # Tarih klasörlerini
gösterme olayını bağla
        self.current_folder = mechanics_folder  # Mevcut klasörü güncelle
```

main.py

```
def search_in_current_folder(self):
        """Mevcut klasördeki dosya ve klasörlerde arama yapar."""
        search_term = self.search_entry.get()  # Arama terimini al
        if not search_term or not self.current_folder:  # Geçersiz terim veya klasör seçilmemişse
            messagebox.showwarning("Uyarı", "Lütfen geçerli bir arama terimi girin ve geçerli bir klasör seçin.")  # Uyarı
mesajı göster
            return

        # Klasördeki klasör ve dosyaları listele
        items = self.list_folders(self.current_folder) + self.list_csv_files(self.current_folder)  # Klasör ve dosyaları
birleştir
        matched_items = [item for item in items if search_term.lower() in item.lower()]  # Arama terimine uyanları bul

        if not matched_items:  # Eşleşme yoksa
            messagebox.showinfo("Sonuç", "Arama sonucu bulunamadı.")  # Bilgi mesajı göster
        else:
            self.listbox.delete(0, tk.END)  # Mevcut listeyi temizle
            for item in matched_items:  # Eşleşenleri listeye ekle
                self.listbox.insert(tk.END, item)
    def list_folders(self, folder_path):
        """Belirtilen klasör yolundaki alt klasörleri listeler."""
        try:
            return [folder for folder in os.listdir(folder_path) if os.path.isdir(os.path.join(folder_path, folder))]  #
Sadece klasörleri döndür
        except Exception as e:
            messagebox.showerror("Hata", f"Klasörler listelenirken bir hata oluştu: {str(e)}")  # Hata mesajı göster
            return []

    def list_csv_files(self, folder_path):
        """Belirtilen klasör yolundaki CSV dosyalarını listeler."""
        try:
            return [file for file in os.listdir(folder_path) if file.endswith(".csv")]  # Sadece .csv uzantılı dosyaları
döndür
        except Exception as e:
            messagebox.showerror("Hata", f"CSV dosyaları listelenirken bir hata oluştu: {str(e)}")  # Hata mesajı göster
            return []

    def open_csv_file(self, event, folder_path):
        """Seçilen CSV dosyasını açar ve içeriğini görüntüler."""
        selected_item = self.listbox.get(self.listbox.curselection())  # Listeden seçilen dosya adını al
        file_path = os.path.join(folder_path, selected_item)  # Dosya yolunu oluştur

        try:
            # CSV dosyasını pandas ile oku
            df = pd.read_csv(file_path)
            # Yeni bir pencere aç ve veriyi göster
            self.show_data_in_new_window(df, selected_item)
        except Exception as e:
            messagebox.showerror("Hata", f"CSV dosyası açılırken bir hata oluştu: {str(e)}")  # Hata mesajı göster

    def show_date_folders(self, event, mechanics_folder):
        """Seçilen mekan klasöründeki tarih klasörlerini listeler."""
        selected_item = self.listbox.get(self.listbox.curselection())  # Seçilen mekan klasörünü al
        folder_path = os.path.join(mechanics_folder, selected_item)  # Mekan klasör yolunu oluştur

        self.listbox.delete(0, tk.END)  # Mevcut listeyi temizle

        # Tarih klasörlerini listele
        date_folders = self.list_folders(folder_path)
        if not date_folders:  # Eğer tarih klasörü yoksa
            messagebox.showwarning("Uyarı", "Bu mekanda herhangi bir tarih klasörü bulunamadı!")  # Uyarı göster
            return

        for date_folder in date_folders:  # Tarih klasörlerini listeye ekle
            self.listbox.insert(tk.END, date_folder)

        # Tarih klasörünü seçince CSV dosyalarını listeleme fonksiyonunu bağla
        self.listbox.bind("<Double-1>", lambda event: self.show_csv_files_directly(event, folder_path))
        self.current_folder = folder_path  # Mevcut klasörü güncelle
```

main.py

```
# Uygulamayı çalıştır
if __name__ == "__main__":
    root = tk.Tk()  # Ana pencereyi oluştur
    app = ProjectApp(root)  # Uygulama sınıfını başlat
    root.mainloop()  # Uygulama döngüsünü başlat
```