

Petre-Florin STAN

## PROIECT DE DIPLOMĂ

Sistem de control al accesului cu recunoaștere facială și  
detectarea obiectelor

SPECIALIZAREA ELECTRONICĂ APLICATĂ

FACULTATEA DE ELECTRONICĂ, TELECOMUNICAȚII  
ȘI TEHNOLOGIA INFORMAȚIEI  
UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

2025

**UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA**  
**FACULTATEA DE ELECTRONICĂ**  
**TELECOMUNICAȚII ȘI TEHNOLOGIA INFORMAȚIEI**  
**Specializarea: Electronică Aplicată**

**Sistem de control al accesului cu recunoaștere facială și  
detectarea obiectelor**

**Proiect de diplomă**

**Absolvent,  
Petre-Florin STAN**

*Stm*

**Decan,  
Prof.dr.ing. Ovidiu POP**

**Președinte comisie,  
Prof.dr.ing. Dorin PETREUŞ**

**2025**



**UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA**  
**FACULTATEA DE ELECTRONICĂ**  
**TELECOMUNICAȚII ȘI TEHNOLOGIA INFORMAȚIEI**

**Departamentul Electronică Aplicată**

**Titlul proiectului de diplomă:**

**Sistem de control al accesului cu recunoaștere facială și detectarea obiectelor**

**Descrierea temei:**

Tema propune un sistem de control al accesului, care utilizează recunoașterea facială și detecția obiectelor, având ca scop principal creșterea nivelului de securitate. Sistemul este bazat pe microcontrolerul ESP32-CAM, ales pentru integrarea camerei OV2640 și conectivitatea Wi-Fi. Din punct de vedere software, implementarea a presupus dezvoltarea unei aplicații Python, care utilizează biblioteci precum OpenCV, face\_recognition și YOLOv3 pentru procesarea imaginilor în timp real. În plus, a fost realizată o interfață web dedicată, care permite controlul și monitorizarea sistemului local sau de la distanță. ESP32-CAM a fost utilizat pentru achiziția imaginilor și comanda releului, buzzer-ului și LCD-ului. Performanțele obținute în urma testelor efectuate au confirmat capacitatea sistemului de a detecta rapid fețele și obiectele, cu semnalizare vizuală și sonoră corespunzătoare. În final, proiectul a fost complet integrat hardware pe un prototip funcțional și testat practic în scenarii reale, confirmând stabilitatea și eficiența sistemului.

**Locul de realizare:**

Facultatea de Electronică, Telecomunicații și Tehnologia Informației, Sala 320, Strada George Barițiu, nr. 26, Cluj-Napoca, România

**Data emiterii temei:** 28.10.2024

**Data predării temei:** 08.07.2025

**Absolvent,**

**Petre-Florin STAN**

*Stan*

**Director departament,**

**Prof.dr.ing. Dorin PETREUŞ**

**Conducător,**

**Ş.I.dr.ing. Elena ȘTEȚCO**



**Absolvent: Petre-Florin STAN**

**Conducător: S.I.dr.ing. Elena ȘTEȚCO**

## **SINTEZA PROIECTULUI DE DIPLOMĂ**

### **Sistem de control al accesului cu recunoaștere facială și detectarea obiectelor**

Proiectul propune realizarea unui sistem de control al accesului, bazat pe recunoaștere facială și detecția obiectelor, folosind ESP32-CAM și algoritmi de inteligență artificială. Activarea sistemului este realizată prin intermediul unui senzor PIR, iar camera integrată capturează imagini care sunt transmise prin Wi-Fi către aplicația Python pentru procesare ulterioară. Elementul de noutate al proiectului constă în integrarea, în timp real, a tehnologiilor de identificare facială cu clasificarea obiectelor, ceea ce contribuie semnificativ la sporirea nivelului de securitate. Datorită acestei abordări, sistemul prezintă o utilitate practică ridicată, fiind potrivit pentru implementare în instituții, birouri, precum și în locuințe inteligente. Contribuția personală a constat în proiectarea arhitecturii hardware-software, programarea modulelor, implementarea aplicației Python cu OpenCV, face\_recognition, YOLOv3 și Flask, precum și integrarea bazei de date MySQL și a unei interfețe web dedicată. Sistemul a fost realizat pe prototip alimentat la 12V folosind un convertor LM2596 și a fost testat practic. Rezultatele obținute au demonstrat o acuratețe bună, detecție rapidă și funcționalitate stabilă. Totodată, sistemul este prevăzut cu afișaj LCD, avertizare sonoră cu buzzer și comandă pentru încuietoarea electromagnetică, atingând astfel obiectivele propuse.

### **Access control system with facial recognition and object detection**

The project proposes the development of an access control system based on facial recognition and object detection, using the ESP32-CAM microcontroller and artificial intelligence algorithms. System activation is achieved via a PIR sensor, while the integrated camera captures images that are transmitted over Wi-Fi to a Python application for further processing. The innovative aspect of this project lies in the real-time integration of facial identification technologies with object classification, which significantly enhances security levels. Due to this approach, the system has high practical applicability, making it suitable for implementation in institutions, offices, as well as smart homes. The personal contribution consisted of designing the hardware-software architecture, programming the modules, implementing the Python application using OpenCV, face\_recognition, YOLOv3, and Flask, as well as integrating the MySQL database and a dedicated web interface. The system was built on a prototype powered at 12V using an LM2596 buck converter and was tested in practice. The results obtained demonstrated good accuracy, fast detection, and stable functionality. Additionally, the system is equipped with an LCD display, audible alerts via a buzzer, and control of the electromagnetic lock, thus achieving the proposed objectives.

*Avizul conducătorului*

Conducător,

Absolvent,

*Stan*



# Cuprins

<b>1</b>	<b>Summary.....</b>	<b>3</b>
	State of the Art .....	3
	Theoretical Fundamentals .....	3
	Implementation.....	5
	Experimental Results.....	7
<b>2</b>	<b>Planificarea activității.....</b>	<b>9</b>
<b>3</b>	<b>Stadiul actual.....</b>	<b>10</b>
<b>4</b>	<b>Fundamentare teoretică .....</b>	<b>13</b>
4.1	Machine Learning (învățarea automată) și Deep Learning (învățarea profundă) .....	13
4.2	Rețele neuronale artificiale .....	13
4.3	Recunoaștere facială utilizând metode de învățare profundă.....	15
4.3.1	Preprocesarea imaginilor faciale .....	15
4.3.2	Procesarea imaginilor faciale .....	16
4.3.3	Antrenarea modelului.....	16
4.3.4	Testarea modelului.....	16
4.4	Modelul YOLO (You Only Look Once).....	16
4.5	Microcontrolere.....	17
4.5.1	Tipuri de microcontrolere .....	18
4.5.2	Diferențe dintre microprocesoare și microcontrolere .....	18
4.5.3	ESP32-CAM .....	19
4.6	Protocole.....	20
4.6.1	Protocolul Universal Asynchronous Receiver Transmitter (UART).....	20
4.6.2	Serial Peripheral Interface (SPI) .....	21
4.6.3	Inter-Integrated Circuit (I2C).....	22
4.7	Senzori IR .....	22
4.7.1	Senzori de temperatură.....	22
4.7.2	Senzori ultrasonici.....	23
4.7.3	Senzori PIR .....	23
<b>5</b>	<b>Implementarea soluției adoptate .....</b>	<b>24</b>
5.1	Implementarea folosind modulului ESP32CAM .....	24
5.2	Implementarea aplicației în Python.....	27
5.2.1	Implementarea algoritmului de recunoaștere facială .....	28
5.2.2	Implementarea algoritmului de detecție a obiectelor .....	29

5.3	Implementarea bazei de date.....	31
5.4	Periferice hardware integrate în sistem .....	33
5.4.1	Modulul buzzer .....	33
5.4.2	Modulul senzor PIR .....	34
5.4.3	Încuietoarea electromagnetică controlată de releu .....	35
5.4.4	Afișaj LCD 2004 .....	36
5.5	Circuitul final al sistemului.....	37
5.6	Interfața web.....	38
<b>6</b>	<b>Rezultate experimentale .....</b>	<b>41</b>
<b>7</b>	<b>Concluzii .....</b>	<b>52</b>
<b>Bibliografie .....</b>		<b>53</b>
<b>Anexe .....</b>		<b>55</b>

# 1 Summary

## State of the Art

The human face is a unique and relevant feature for identifying individuals. Although faces are common objects, they are frequently used in recognition processes. Facial recognition has become increasingly important over the years, now being one of the most used biometric methods for quick and secure access.

In the 1960s, the first results in facial recognition were obtained when facial features were added, and in 1988 the Eigenface method used linear algebra for recognition. In 1991, the first real-time automatic system appeared. After 2000, databases such as FERET supported development, and from 2011 neural networks improved accuracy. In 2014, Facebook launched DeepFace, and in 2017 Apple introduced Face ID on the iPhone X [1] [2] [3] [4] [5].

FaceNet, developed by Google in 2015, uses convolutional neural networks to transform faces into numerical vectors in a Euclidian space, achieving an accuracy of 99.63% on LFW. In 2018, ArcFace improved facial recognition by applying an angular margin, increasing the separation between different faces and reaching an accuracy of 99.82% on LFW. Recently, in 2024, RobFaceNet was introduced, extracting fine facial details effectively and quickly on resource-limited devices. It achieved 99.95% accuracy on CA-LFW and 92.23% on CP-LFW [6] [7] [8] [9].

Currently, facial recognition is widely used due to the security it offers in commercial, banking, law enforcement, and public safety domains. It replaces traditional passwords and is used to unlock phones and other devices. Banks use this technology for identity verification and fast account opening without physical presence. In airports, it allows quick passage through controls without physical documents. US federal agencies use facial recognition to identify suspects [11] [12] [13] [14] [15] [16] [21] [22].

Although facial recognition has evolved greatly, issues remain regarding image quality, poor lighting, wrong angles, and accessories like glasses. According to NIST, there are racial and gender biases, with false positive rates for people from East Asia and West Africa, and higher false negative rates for women. Performance also decreases for children and the elderly. There are concerns about the privacy of collecting biometric data. The EDPB prohibits mass surveillance, requiring clear justification for data collection. Operators must reduce error risks, ensure transparency and respect for the rights of the individuals concerned [13] [17] [18].

## Theoretical Fundamentals

Machine learning is a branch of artificial intelligence that develops systems capable of learning from data without explicit programming for each task. These are four main categories: supervised learning that uses labeled data to make predictions, unsupervised learning that works with unlabeled data to discover patterns and groupings, semi-supervised learning which combines both methods for improving results and reinforcement learning that involves an agent that interacts with the environment to make optimal decisions. Deep learning is a subcategory of machine learning, based on artificial neural networks inspired by how the human brain processes information. Deep learning models automatically extract relevant features and perform learning and classification simultaneously without human intervention [23] [24].

Artificial neural networks are an essential part of machine learning and form the foundation of deep learning, being structurally inspired by biological neurons. They consist of numerous interconnected simple units and model nonlinear relationships between inputs and outputs, used in applications like facial recognition and handwriting recognition. Their structure includes an input layer, hidden layers, and an output layer. Feed-forward networks pass data in one direction only, with the perceptron as a classic example. Recurrent networks have feedback connections and can

recognize temporal patterns, while convolutional networks classify images by extracting local features. Recursive networks process data structures of varying sizes and are used in natural language processing [25].

Modern facial recognition systems use deep learning to extract relevant features from images with high accuracy. The process begins with preprocessing, which involves detecting face in a image, aligning it for standardization, and anti-spoofing detection to prevent fraud. Image processing includes one-to-many techniques, generating multiple versions from a single image, and many-to-one normalization, converting several images into a single standardized frontal image. Models are trained using backbone networks like AlexNet or ResNet and composite networks, with loss functions based on Euclidean distance, angular margin or center loss to improve accuracy. During testing, new images are preprocessed, feature embeddings are extracted, and Euclidean distances and cosine similarities are compared for identification [26] [27].

YOLO is an object detection algorithm that converts an image directly into object coordinates and classes in a single pass. The image is divided into a grid, each cell detecting objects whose centers falls within it, predicting coordinates, dimensions and confidence scores. Class probabilities are also estimated. YOLO is implemented as an end-to-end convolutional neural network inspired by GoogLeNet, adapted for real-time detection. Its architecture includes 24 convolutional layers for feature extraction and two fully connected layers for final predictions. YOLO's advantages are high speed and a global view of the entire image during training and testing [28].

A microcontroller is an integrated microcomputer on a single chip, also called an embedded system, fabricated using VSLI technology. It includes a CPU, memory, I/O ports, communication interfaces, A/D and D/A converters, clock generator, reset, Brown-Out detector, watchdog, timers and interrupts. The CPU executes instructions, memory can be RAM, ROM and Flash, and I/O ports connect to peripherals. Interfaces like UART, SPI and I2C enable serial communications. The clock generator synchronizes operations, A/D converters transform analog signals into digital, and D/A converts produce analog outputs. The reset restores the initial state, Brown-Out detector monitors voltage, watchdog prevents software lock-up, timers manage counting and interrupts ensure fast responses to events. Microcontrollers are classified by bus width and program memory. Von Neumann architecture uses for the same memory for code and data, being slower, while Harvard architecture separates them for simultaneous access, increasing efficiency. Instruction sets can be CISC or RISC [29] [30].

Microcontrollers integrate CPU, memory and peripherals on the same chip, being compact, energy-efficient, and easy to implement with single power supply. Microprocessors include only CPU and require external memory and peripherals, being bulkier and consuming more power [31].

ESP32-CAM is a compact module with an integrated camera, developed by Essence, used in IoT applications such as smart homes, wireless monitoring and facial recognition. It has a microcontroller with 512KB SRAM, 4MB PSRAM, 32 Mbit Flash, classic Bluetooth and BLE, supporting UART, SPI, I2C, PWM and TF card interfaces. It includes Wi-Fi 802.11 b/g/n with advanced security and operates at 5V, with low sleep mode consumption, ideal for battery-powered systems. It operates from -20°C to 85°C and has 16 pins, with 9 usable as I/O ports. Its dimensions are 27 x 40.5 x 4.5 mm [32] [33].

UART, SPI and I2C are protocols used for serial communication between microcontrollers and peripherals. UART is asynchronous, using two wires, TX and RX, without a clock signal, transmitting data as packets with start bit, data frame, parity bit and stop bit. It is simple but has limited speed and data frame size. SPI is synchronous, full-duplex, controlled by a master device that generates the clock signal, using four lines: SS, MOSI, MISO and SCK, enabling fast data transfer in embedded and IoT systems. I2C is synchronous, using two lines, SDA and SCL, to connect multiple slave devices to one or more masters. Communication involves messages with addressing, read/write bit, and ACK/NACK bit. I2C is efficient for connecting multiple devices but has lower transfer rates than SPI and requires more complex hardware implementation [35] [36] [37] [38] [39].

Infrared sensors are electric devices detecting emitted or reflected infrared (IR) radiation from objects, used for presence, motion and temperature variation detection. Passive sensors (PIR) detect thermal radiation naturally emitted by bodies, while active sensors use an emitter, like an IR LED and a detector sensitive to emitted light. IR sensors include temperature sensors that measure temperature from a distance using laser targeting without contact. Ultrasonic sensors determine object distance by emitting sound waves and measure reflection time, though inclined surface may not reflect properly. PIR sensors detect change in IR radiation due to human movement, having two differently arranged pyroelectric zones and a Fresnel lens for 180-degree field of view. Without the lens, the detection area is narrow. PIR sensors are widely used for motion detection due to their high sensitivity and low cost [40] [41].

## Implementation

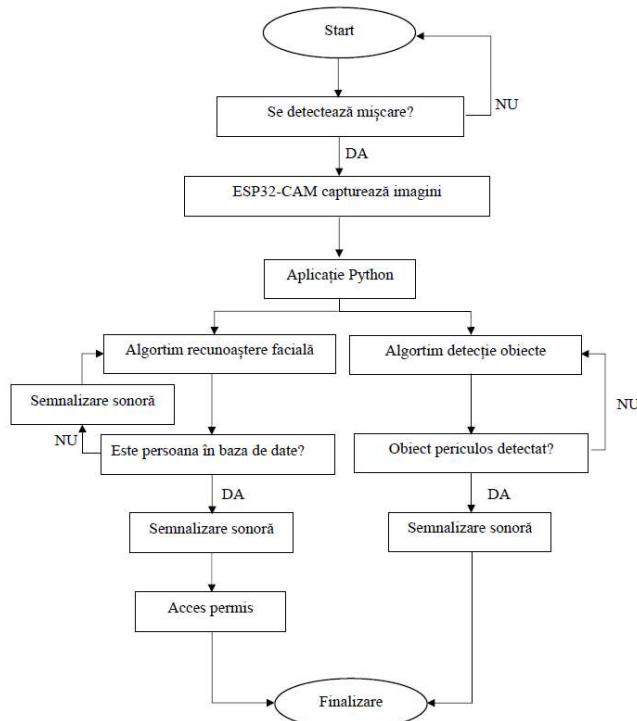


Figure 8. Block diagram of the system.

This project presents an access control based on facial recognition and dangerous object detection. Activation is performed by a PIR sensor that triggers the ESP32-CAM camera to capture images. These are analyzed by a Python application that identifies the detected person and objects. Depending on the results, access is granted with a corresponding sound signal, or a warning is issued in case of a dangerous object or an unknown person.

The system uses the ESP32-CAM microcontroller due to its high performance, low cost and integrated OV2640 camera, making it suitable for IoT projects requiring image transmission over Wi-Fi. Programming requires an FT232RL serial adaptor, converting USB to UART signals, connecting TX, RX, GND and 5V, with IO0 pulled to GND for upload mode. The code was developed in Arduino IDE using the “AI Thinker ESP32-CAM” board, C++ language and libraries such as “WiFi.h” for connectivity and “esp32cam.h” for camera configuration. The web server, created with “WebServer.h” on port 80, handles HTTP requests, remaining active in the loop for continuous processing.

The software application was developed in Python using Spyder IDE Anaconda, due to its high-performance libraries such as OpenCV for image processing and NumPy for matrix manipulation. The Flask framework was used to create the web control interface, running locally

on port 5000. The system retrieves real-time images from the ESP32-CAM’s OV2640 camera, connected to the network, via separate thread that downloads, decodes, and processes JPEG frames. Images are analyzed by dedicated facial recognition and object detection functions, while in the absence of a connection, an error message is displayed.

Facial recognition was implemented using the *face\_recognition* library, based on Dlib, an open-source C++ package supporting image processing and deep learning. In Python, it is installed rapidly via pip without complex configurations. The main function receives image captured by ESP32-CAM, converted from BGR to RGB. Faces are identified using HOG model, then facial feature vectors are extracted. Each vector is compared for those in the database using Euclidean distance, with a match verified under a 0.6 threshold. Based on the results, a green rectangle with a recognized name is drawn on the image or a red rectangle with the “Unknown” label.

Object detection was implemented with YOLOv3, using Darknet-53 neural network with 53 convolutional layers to extract detailed visual features. The model requires yolov4.cfg, yolov3.weights and coco.names files. In the application, the model is initialized, and images are processed in the main function. The image is transformed into a blob, resized to  $416 \times 416$  px, converted to RBG, and passed through the network. Classes with maximum probability are extracted and if the score exceeds 0.5, object coordinates are converted to top-left format for display. Non-Maximum Suppression is applied to remove redundant detections, keeping only relevant boxes. Object boxes are displayed on the image, blue for regular objects and red for dangerous ones.

For database implementation, the XAMMP software package was used, including MySQL server and the phpMyAdmin interface accessible via Apache. The created database, named “licenta” contains two tables: *users* and *logs*. The *users* table stores person names and facial encodings as numeric vectors, structured with id, name and encoding fields. The *logs* table records detected accesses with id, name and timestamp fields. The Python application connects to the databases using “pymysql” library to load facial data for recognition and save each detected access.

The MH-FMD passive piezoelectric buzzer module is used for sound signaling. It operates by applying a PWM signal on GPIO14, emitting different tones based on context: a continuous 1000Hz tone for recognition persons and three short 500Hz beeps for unknown ones or dangerous objects. Control is implemented in Arduino IDE via the tone() function and communication between Python and ESP32-CAM is performed via HTTP routes.

The SEN0018 PIR module sensor detects motion based on IR radiation emitted by warm bodies, with a detection distance of 9m and a  $110^\circ$  angle. It is powered at 5V, GND and the digital pin connects to GPIO13 configurated as input. It includes a jumper for retriggerable mode and a potentiometer for adjusting pulse duration. In code, the sensor is continuously monitoring uand if no motion is detected for 60s, the ESP32-CAM enters *deep sleep*, resuming via external wake-up.

For physical access control, an electromagnetic lock powered at 12V is used, controlled via a relay module with an optocoupler. The relay connects pin COM to 12V, NO to the lock, and is activated on GPIO15 with high logic. There are two functions implemented who activated the lock and automatically closed it after 10s.

The 2004 LCD with I2C module adapter is used for displaying messages, connected to GPIO4, for SDA, and GPIO 2, for SCL, powered at 5V. The “LiquidCrystal\_I2C” library manages display functions in Arduino IDE. There is a default message which changes if a person is recognized or unrecognized.

These peripherals integrate visual audio, and access control functionalities, ensuring complete system operability. Power is supplied by a 12V DC external source required for the lock, while other components operate at 5V via an LM2596 buck converter, efficiently reducing voltage to a stable, adjustable level, providing 3A for safe operation of all modules.

The web interface was developed with the Flask framework, defining specific routes for each action. The main route handles login, returning login.html, and upon validation requires the user to the dashboard. The dashboard displays the ESP32-CAM video feed via the “/video” route as a JPEG stream. Adding new users is managed by the “/add\_user” route, processing images with

*face\_recognition* and saving data to the database. The “/users” route displays all users and allows deletion, while “/logs” shows the latest 50 access events. Logout clears the session and redirects the user to the login page.

## Experimental Results

In the experimental results, it is observed that the system correctly detects motion using the PIR sensor, activating the ESP32-CAM camera to capture images.

In Figure X, the person from the database is correctly identified, displaying a green bounding box and the detected name, and the electromagnetic lock opens as programmed.

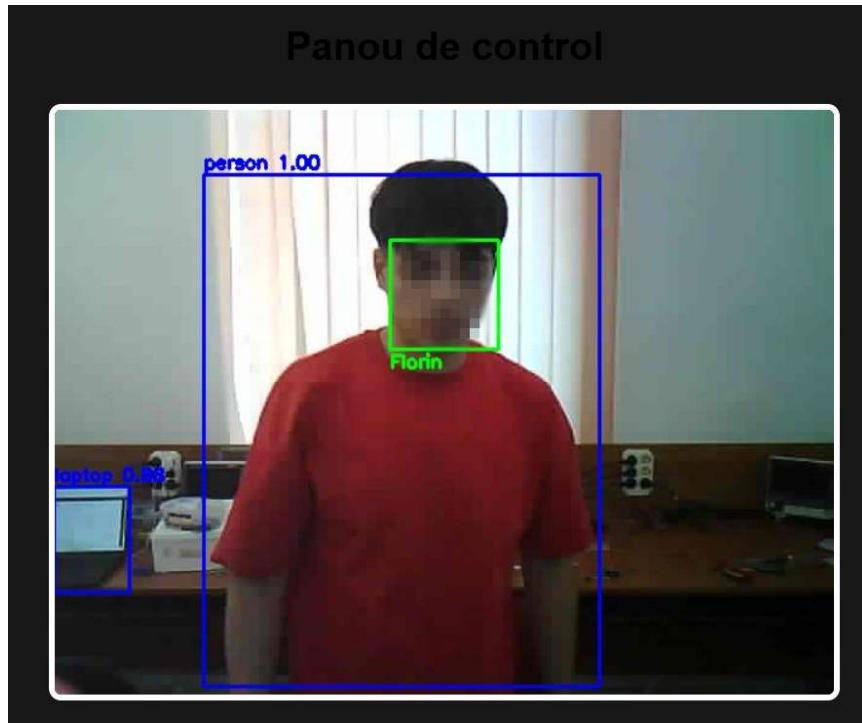


Figure 49. The result of the person's identification from the database.

In Figure X, an unknown person is detected, highlighted with a red bounding box and the “Unknown” label, while the buzzer generates the corresponding sound alert.

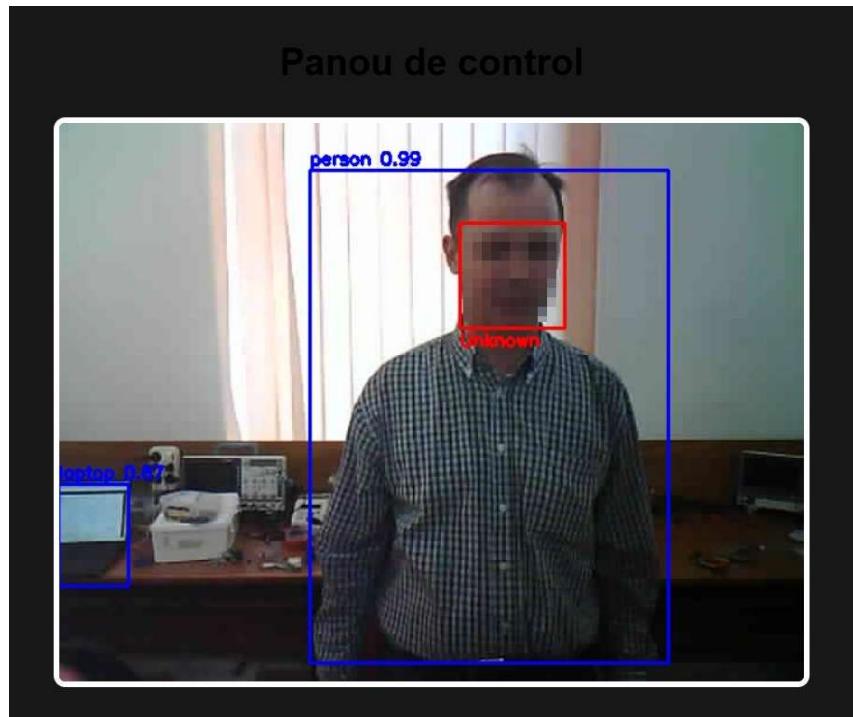


Figure 53. The result of the recognition of a person who is not in the database.

In Figure 57, the detection of a dangerous object, specifically a knife, is illustrated, framed with a red bounding box and a probability of 82%, confirming that the YOLOv3 algorithm classifies objects and generates the specific warning.



Figure 57. The result of the detection of a dangerous object.

Additionally, the LCD displays different messages depending on the results, and the web interface allows real-time visualization of these processes, demonstrating that the system operates according to the implemented hardware and software design.

## 2 Planificarea activității

<b>Activitate</b>	<b>Data</b>	<b>octombrie 2024</b>	<b>noiembrie 2024</b>	<b>decembrie 2024</b>	<b>ianuarie 2025</b>	<b>februarie 2025</b>	<b>martie 2025</b>	<b>aprilie 2025</b>	<b>mai 2025</b>	<b>iunie 2025</b>	<b>iulie 2025</b>
Alegerea temei											
Documentare teoretică generală											
Studiul componentelor hardware/software											
Achiziția componentelor											
Testarea individuală a modulelor											
Implementarea codului pe module											
Integrarea hardware completă											
Dezvoltarea aplicației software în Python											
Testarea sistemului integrat											
Redactarea documentației											
Revizuire și corectură finală											
Predarea lucrării și susținerea prezentării finale											

### 3 Stadiul actual

Fața umană oferă una dintre cele mai relevante și unice trăsături care poate fi folosită pentru identificarea persoanei. Deși unice, fețele alcătuiesc cea mai comună clasă de obiecte care poate servi procesării unei recunoașteri ulterioare. De-a lungul anilor, recunoașterea facială a căptănat o atenție din ce în ce mai mare, ajungând una dintre cele mai utilizate funcții biometrice de acces rapid și sigur.

Primele rezultate în domeniul recunoașterii faciale au fost obținute în anii 1960, de către Woody Bledsoe, Helen Chan Wolf și Charles Bisson. Lucrarea acestora a constat în marcarea manuală a celor mai importante trăsături ale feței, folosite ca repere, cum ar fi: ochii, gura, nasul etc., fiind apoi calculate distanțele dintre aceste repere printr-un sistem automat și comparate cu imagini de referință folosite pentru identificare. Această muncă a fost ulterior preluată de Goldstein, Harmon și Lesk, în anii '70, care au extins procesul de recunoaștere facială reușind să includă noi repere, cum ar fi: culoarea părului, grosimea buzelor etc. Însă, deși a crescut precizia, marcarea reperelor și calculul distanțelor trebuiau făcute manual. Spre finalul anilor 1980, începutul anilor 1990 apar primele progrese remarcabile. În 1988, Sirovich și Kirby au folosit algebra liniară în aplicații de recunoaștere facială, cel mai cunoscut sistem devenind Eigenface, având la bază analiza caracteristicilor unor colecții de imagini. În 1991, Turk și Pentland au continuat proiectul Eigenface, reușind să creeze primul sistem automat de detectare facială în timp real [1].

Începutul anilor 2000 aduce un pas important pentru a încuraja această industrie, astfel că Defense Advanced Research Projects Agency (DARPA) dezvoltă Face Recognition Technology (FERET), prima bază de date complexă, compusă din 2400 de imagini pentru 850 de persoane [2]. Totodată în 2006 se lansează Face Recognition Grand Challenge (FRGC), care avea ca scop dezvoltarea domeniului, pornind de la concepte deja existente [3]. Din 2011, tehnologia de recunoaștere facială căptă amplioare odată cu apariția rețelelor neuronale, deep learning machine, care au la bază antrenarea cu cât mai multe seturi de imagini pentru a accurația recunoașterii cât mai mare [2]. În anul 2014, compania Facebook revoluționează tehnologia, propusă până atunci, prin intermediul sistemului DeepFace, captând atenția pe rețelele de socializare, devenind rapid un subiect mai controversat. Sistemul propus a reprezentat la momentul respectiv cea mai precisă tehnologie de detectare facială, cu o accurație de 97.35%. Sistemul contribuie prin: dezvoltarea și antrenarea rețelei neuronale la un nivel mai profund, cu o scară mai largă de imagini și modelarea 3D a fețelor [4]. Odată cu apariția noului model de telefon de la Apple în 2017, iPhone X, se lansează și prima funcție de deblocare a telefonului pe baza recunoașterii faciale, Face ID, înlăturând funcția de deblocare pe baza amprentei, Touch ID, îmbunătățind astfel securitatea [5]. În următorii ani, recunoașterea facială a devenit una dintre cele mai utilizate tehnologii biometrice prezintă în domenii vaste.

Printre cele mai importante sisteme de recunoaștere facială se numără FaceNet, propus de Google, în anul 2015. Algoritmul utilizează o rețea neuronală convolutională, care spre deosebire de metodele anterioare care comparați trăsături extrase din mai mulți pași intermediari, este antrenată să optimizeze direct imaginea unei fețe și să transforme trăsăturile într-un vector numeric. Practic imaginile cu fețe sunt transformate într-un spațiu euclidian, unde fețele care aparțin aceluiași persoană sunt apropiate una de alta, iar fețele care aparțin unor persoane diferite sunt foarte departe. Pentru antrenare se folosesc aşa numite triple. Sistemul este considerat eficient, obținându-se rezultate semnificative, folosind doar 128 de octeți pentru o față și o accurație record la momentul respectiv, 99,63% pe setul de date LFW (Labeled Faced in the Wild) [6] [7].

ArcFace, unul din cei mai puternici algoritmi, propus în 2018, folosește o rețea neuronală convolutională pentru extragerea trăsăturilor faciale dintr-o imagine pe baza unei reprezentări numerice unice, numit vector embedding. Acest algoritm a îmbunătățit problema cu ajutorul funcției de pierderi propusă în lucrarea „SphereFace” și aplică o marjă unghiulară asupra unghiului dintre vectorul de ponderi și vectorul de trăsături, care practic mărește distanța dintre fețele diferite și micșorează distanța dintre fețele asemănătoare. Datorită acestui fapt, sistemele bazate pe ArcFace

ating o acuratețe de 99,82%, pe setul de date LFW (Labeled Faced in the Wild) și s-a dovedit a fi eficient și ușor de implementat [8].

Mai recent, pentru a trata problema complexității arhitecturilor ce utilizează rețele neuronale convoluționale, care creează dificultăți dispozitivelor cu resurse limitate, în 2024 se introduce RobFaceNet. Acest sistem reușește să extragă inclusiv detalii fine ale feței, îmbunătățind reprezentarea, fără a sacrifica eficiența și viteza. Rezultatele arată o performanță superioară, obținându-se 95,95% pe CA-LFW (Cross-Age Labeled Faced in the Wild) și 92,23% pe CP-LFW (Cross-Pose Labeled Faced in the Wild) [9].

NEC Corporation anunță, în aprilie 2025, că algoritmul lor de recunoaștere facială a obținut primul loc mondial în urma evaluării FRTE (Face Recognition Technology Evaluation) 1:N, sponsorizat de NIST (National Institute of Standards and Technology), reușind să obțină cea mai mare performanță, cu o eroare de doar 0,07% la autentificare. Tehnologia NEC este prezentă în peste 50 de țări, folosită în aproximativ 80 de aeroporturi din lume, de asemenea și în transportul public, fabrici, hoteluri, bănci, etc. Pe viitor, NEC urmărește să înlocuiască cat mai mult metodele de identificare fizice și să dezvolte noi soluții, concentrându-se pe confidențialitate și etică [10].

Datorită securității asigurate, aplicabilitatea tehnologiei de recunoaștere facială este vastă, folosindu-se atât în domenii comerciale, cât și în cel al legii și a siguranței publice. [13]

Recunoașterea facială folosită pentru identificarea utilizatorilor este o soluție care îmbunătățește considerabil securitatea și reduce vulnerabilitatea, fără a mai depinde de tradiționalele parole pentru autentificare, care adesea sunt ținte ale hackerilor [12]. De asemenea, tehnologia este foarte populară în sistemele de control destinate accesului, organizațiile optând pentru recunoașterea facială, în detrimentul cheilor sau a cardurilor [11].

Dispozitive precum telefoane, tablete sau laptopuri sunt cel mai frecvent deblocate folosind tehnologia de recunoaștere facială. În locul parolelor sau codurilor PIN tradiționale, această funcție biometrică oferă mult mai multă siguranță, dar și o modalitate mai rapidă de acces a dispozitivelor utilizate, cum ar fi iPhone-urile, iPad-urile sau diferite PC-uri cu Windows care au implementată această tehnologie [13].

Conform unui articol publicat în 2024 de CyberLink, tot mai multe instituții bancare optează pentru serviciile bancare inteligente. Odată cu apariția funcției de Face ID, lansată de Apple, utilizatorii s-au familiarizat rapid cu tehnologia, devenind una dintre cele mai sigure și rapide metode de a oferi servicii fără riscuri, fiind preluată și de bănci. În consecință, inovațiile în inteligență bancară au făcut serviciile pentru utilizatori mult mai ușor de utilizat, în combinație cu captura facială, fără a mai fi nevoie de prezență fizică și accelerând deschiderea de noi conturi. Așteptările tot mai mari ale consumatorilor pot fi satisfăcute prin tehnologii moderne cum ar fi verificarea KYC (Know Your Customer). Mai mult, în zilele noastre serviciile la cerere sunt la modă, dorind efectuarea diferitelor achiziții fără a părăsi locuința, ceea ce face mersul fizic la bancă depășit în acest context. Metodele de plată au evoluat semnificativ. Acum se pot face tranzacții rapid și simplu folosind doar telefonul mobil și recunoașterea facială [14].

Industria aeroportuară folosește tot mai mult tehnologii moderne pentru a îmbunătăți siguranța și eficiența în deplasarea pasagerilor, printre care se numără și recunoașterea facială. Aeroportul Internațional Dubai a fost unul din primele aeroporturi care a implementat această tehnologie, fiind considerat și cel mai aglomerat. În 2020 a început să folosească recunoașterea facială și a irisului, sub forma unui sistem biometric integrat care a făcut posibilă trecerea pasagerilor prin mai multe puncte de control din aeroport fără să mai fie nevoie de prezentarea fizică a documentelor [15]. Conform unui articol publicat de NEC Corporation, pentru a îmbunătăți fluxul de pasageri, Asociația Internațională de Transport Aerian (IATA) susține inițiativa Fast Travel, care dorește să ofere pasagerilor mai multe opțiuni utilizând recunoașterea facială. Mai multe aeroporturi internaționale au sprijinit acest program prin introducerea de proiecte pilot [16].

Un raport publicat în 2023 de Government Accountability Office (GAO) arată că în Statele Unite, șapte agenții federale, utilizează tehnologia de recunoaștere facială pentru a sprijini identificarea suspectilor și trei agenții dețin astfel de sisteme [21]. Există foarte multe studii care demonstrează că tehnologia de recunoaștere facială s-a dovedit a fi foarte folositoare pentru

siguranța publică. Conform unui studiu publicat de Security Industry Association, în 2020, autoritățile locale din Baltimore, Maryland, ajutați de camerele de supraveghere au reușit să identifice agresorul în urma unui atac violent. Au cerut asistență și au comparat imaginile capturate cu o bază de date folosind tehnologia de recunoaștere facială [22].

Cercetările recente în recunoaștere facială, pe lângă extinderea utilizării în mai multe domenii, se concentrează foarte mult pe protecția datelor biometrice. Lucrarea de cercetare „Advancements in detecting Deepfakes: AI algorithms and future prospects – a review” explorează diferite abordări pentru a identifica conținutul DeepFake în diferite tipuri de media, cum ar fi videoclipuri, imagini și audio. Concluzia la care s-a ajuns este că inteligența artificială și metodele deep learning analizează modificările vocale și faciale din media pentru a identifica conținuturi DeepFake, prin care se recurge la fraude [19]. Mai multe metode de protecție a datelor au fost dezvoltate în cazul unor probleme de securitate, potrivit unei cercetări recente, „Toward a Privacy-Preserving Face Recognition System: A Survey of Leakages and Solutions”, publicată în 2025. S-au examinat soluții avansate, cum ar fi Secure Multiparty Cryptography (SMC) și Homomorphic Encryption (HE). Aceste soluții permit realizarea tehniciilor de recunoaștere facială, fără a expune efectiv datele. Federated Learning (FL) reprezintă o nouă abordare care permite modelelor să fie antrenate direct pe dispozitivele utilizatorului, limitând transferul datelor sensibile. Modelul Differential Privacy (DP) garantează cea mai mare confidențialitate prin reducerea probabilității de identificare a informațiilor personale, cu ajutorul unor mecanisme de generare a zgomotului, cele mai comune fiind Laplace și Gaussian [20].

Cu toate că recunoașterea facială a făcut progrese remarcabile în ultimii ani, tehnologia are încă limitări care pot afecta în special acuratețea algoritmilor. Calitatea imaginilor capturate este una din cele mai frecvente probleme, determinată de iluminare slabă, umbre sau unghiuri inadecvate. De asemenea, purtarea păturii pe față sau a ochelarilor poate, la rândul său, genera probleme [13]. Conform unui articol publicat de National Institute of Standards and Technology (NIST), tehnologia nu exclude prejudecățile și rasismul predominant în societate. Persoanele din Asia de Est, cu câteva excepții, și Africa de Vest au cele mai mari rate de fals pozitiv, identificând fețe diferite ca fiind ale aceleiași persoane, iar oamenii din Europa de Est au în general rate mici de fals pozitiv. Apar discriminări inclusive legate de sexe, erorile de fals negativ, în care nu se recunosc corect aceleiași persoane, au un procent mai ridicat în rândul femeilor decât a bărbaților. Performante mai slabe apar și în funcție de vîrstă, la copii și persoanele mai în vîrstă.[17]

Deoarece implică colectarea și analizarea datelor biometrice, apar preocupări legate de confidențialitate folosind recunoașterea facială. Utilizatorii pot fi sceptici cu privire la modul în care sunt colectate, stocate și utilizate datele, care pot duce la folosirea neautorizată a lor [13].

Pentru a împiedica utilizarea neautorizată a tehnologiei de recunoaștere facială, European Data Protection Board (EDPB) stabilește, în documentul publicat în 2023, reguli pentru utilizarea în scop legal a acestei funcții biometrice. Pentru a utiliza datele biometrice trebuie să existe obiective legitime și o justificare clară a colectării datelor. Este interzisă supravegherea în masă, colectarea datelor în timp real în spațiile publice este incompatibilă cu dreptul de viață privată și nu poate fi considerată, cu excepția unor situații extrem de restrictive și autorizare specială. Sunt necesare metode de reducere a riscului de erori prin generarea rezultatelor de fals pozitiv a persoanelor de culoare și a femeilor pentru împiedicarea prejudecăților în ceea ce privește rasa și genul. Prin urmare, operatorii sunt responsabili pentru asigurarea transparenței, limitarea prelucrării datelor și respectarea drepturilor persoanelor vizate [18].

## 4 Fundamentare teoretică

### 4.1 Machine Learning (învățarea automată) și Deep Learning (învățarea profundă)

Învățarea automată (eng. machine learning) reprezintă o ramură a inteligenței artificiale care are ca scop dezvoltarea unor sisteme capabile să învețe din date, fără a fi programate explicit pentru fiecare sarcină. Algoritmii de învățare automată sunt împărțiți în patru mari categorii: învățare supervizată, învățare nesupervizată, învățare semi-supervizată și învățare prin întărire. Învățarea supervizată se bazează pe utilizarea unui set de date de antrenament care conține atât valorile de intrare, cât și ieșirile corespunzătoare, permășând astfel modelului să stabilească o relație funcțională între aceste două componente. Învățarea nesupervizată lucrează cu date fără etichetă, cu scopul de a descoperi tipare, structuri sau grupări ascunse în date. Învățarea semi-supervizată combină avantajele metodelor supervizate și nesupervizate, cu scopul de a îmbunătății performanțele predicției. Învățarea prin întărire implică un agent care interacționează cu mediul și învață să ia decizii optime [23].

Învățarea profundă (deep learning) reprezintă o subcategorie a machine learning-ului, inspirată de modul în care creierul uman procesează informația. Modelele de deep learning sunt construite utilizând rețele neuronale artificiale, fiecare având rolul de a extrage și transforma progresiv reprezentările datelor. Spre deosebire de abordările clasice, modelele deep learning au capacitatea de a învăța automat trăsăturile relevante, eliminând nevoia de intervenție umană pentru selecția trăsăturilor. Astfel, procesele de învățare și clasificare au loc simultan, fiind integrate într-o singură etapă [24].

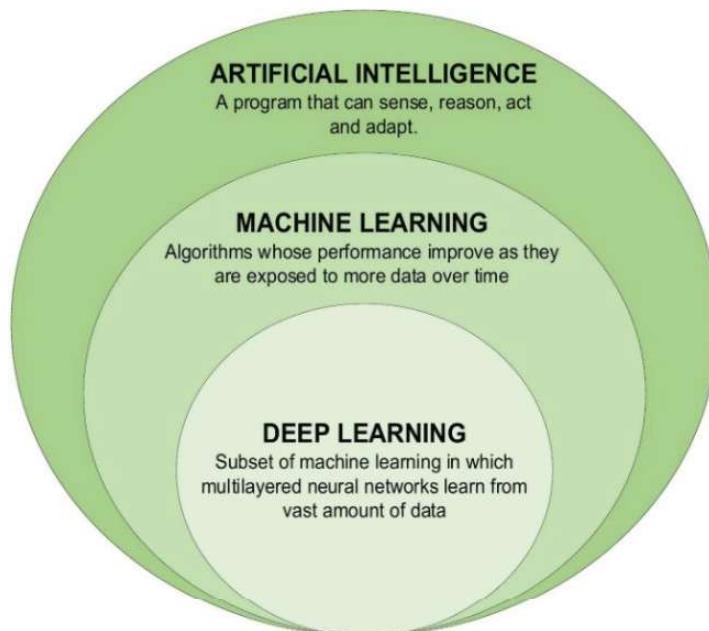


Figura 1. Legătura dintre inteligență artificială, machine learning și deep learning [24].

### 4.2 Rețelele neuronale artificiale

Rețelele neuronale artificiale reprezintă o ramură esențială a machine learning-ului (învățării automate) și constituie fundamentele metodelor avansate de deep learning. Acestea sunt inspirate din punct de vedere structural de modul în care funcționează creierul uman, mai exact a comunicării dintre neuronii biologici. Rețelele neuronale sunt sisteme paralele, compuse dintr-un

număr mare de unități de procesare elementare interconectate. Rețeaua poate fi definită ca un sistem de modelare neliniară a datelor, în care relațiile neliniare dintre sursele de intrări și ieșiri sunt captate și reprezentate. Deoarece au capacitate remarcabile de învățare, rețelele neuronale sunt utilizate frecvent în sisteme precum recunoașterea facială sau a scrisului de mână [25].

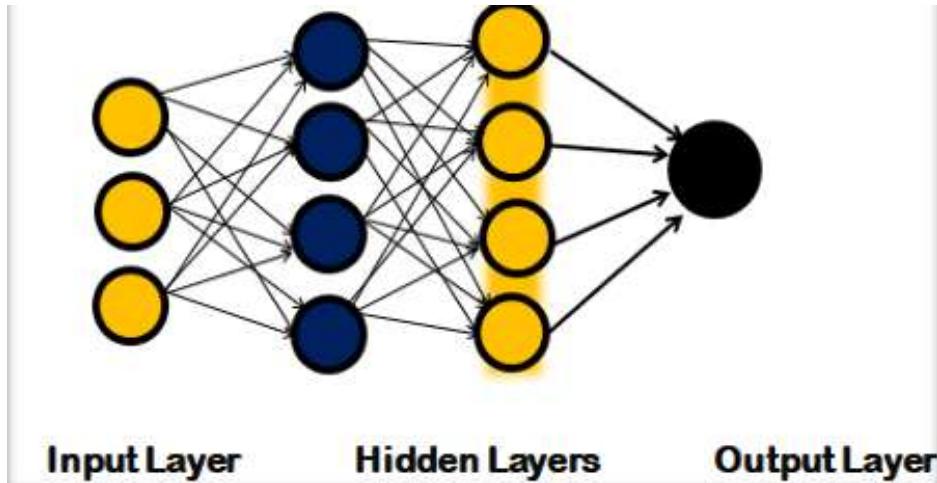


Figura 2. Structura rețelelor neuronale artificiale [25].

Din punct de vedere structural, o rețea neuronală artificială este compusă din trei straturi de bază: stratul de intrare, straturi ascunse și stratul de ieșire. Stratul de intrare este compus din neuroni de intrare artificiali, care primesc datele de intrare și le transmit spre procesare în rețea. Straturile ascunse sunt alcătuite din straturi de intrare și ieșire, unde fiecare conexiune este ponderată în funcție de numărul intrărilor. Stratul de ieșire generează răspunsul final al rețelei.

Rețelele neuronale feed-forward reprezintă cel mai simplu tip de rețea neuronală, în care datele circulă într-o singură direcție, de la stratul de intrare, apoi prin stratul ascuns până la stratul de ieșire. Acest tip de arhitectură asigură un flux unidirecțional al datelor, facilitând procesarea simplă și rapidă. Perceptronul, apărut prima dată în anii 1940 este un exemplu clasic de rețea de acest tip, utilizat în probleme de clasificare binară.

Rețelele neuronale cascadeante au o structură similară cu cea a structurii de tip feed-forward, însă conexiunile includ legături directe între stratul de intrare și restul straturilor, inclusiv cel de ieșire. Această arhitectură permite captarea relațiilor neliniare dintre intrări și ieșiri, oferind un grad de flexibilitate mai ridicat al procesului de învățare.

Rețelele neuronale recurente (eng. Recurrent Neural Networks - RNN) se caracterizează prin existența unor conexiuni de feedback, care permit ieșirilor să influențeze în mod direct intrările viitoare. RNN pot opera cu vectori de intrare de dimensiuni fixe și pot învăța modele complexe care recunosc tipare ce apar în timp.

Rețelele neurale convoluționale (eng. Convolutional Neural Network - CNN) sunt rețele neuronale adânci (deep neural networks), utilizate mai ales în clasificarea imaginilor. Arhitectura are la bază mai multe tipuri de straturi: straturi convoluționale, straturi de pooling, straturi de activare neliniară și straturi complet conectate. Straturile convoluționale și cele de pooling se ocupă de extragerea informațiilor relevante ale datelor de intrare, iar straturile complet conectate transformă aceste caracteristici în ieșiri. Straturile convoluționale aplică operații matematice, precum convinguția, care permit extragerea și identificarea caracteristicilor locale ale unei imagini, cum ar fi marginile sau forme distințe.

Rețelele neuronale recursive (eng. Recursive Neural Network - RvNN) funcționează prin aplicarea în mod repetat a aceleiași funcții sau aceleiași ponderi asupra unor structuri de intrare cu dimensiuni variabile, fiind organizate adesea sub formă de arbori (structuri ierarhice). Sunt utilizate pentru învățarea reprezentărilor semantice ale propozițiilor, și sunt frecvent folosite în prelucrarea limbajului natural pentru analiza și reprezentarea structurilor logice și sintactice [25].

## 4.3 Recunoaștere facială utilizând metode de învățare profundă

Sistemele moderne de recunoaștere facială se bazează pe metode de învățare profundă (deep learning), care utilizează rețele neuronale adânci pentru a extrage automat trăsăturile relevante direct din imagini. Această abordare permite optimizarea procesului de recunoaștere și crește considerabil precizia. [26].

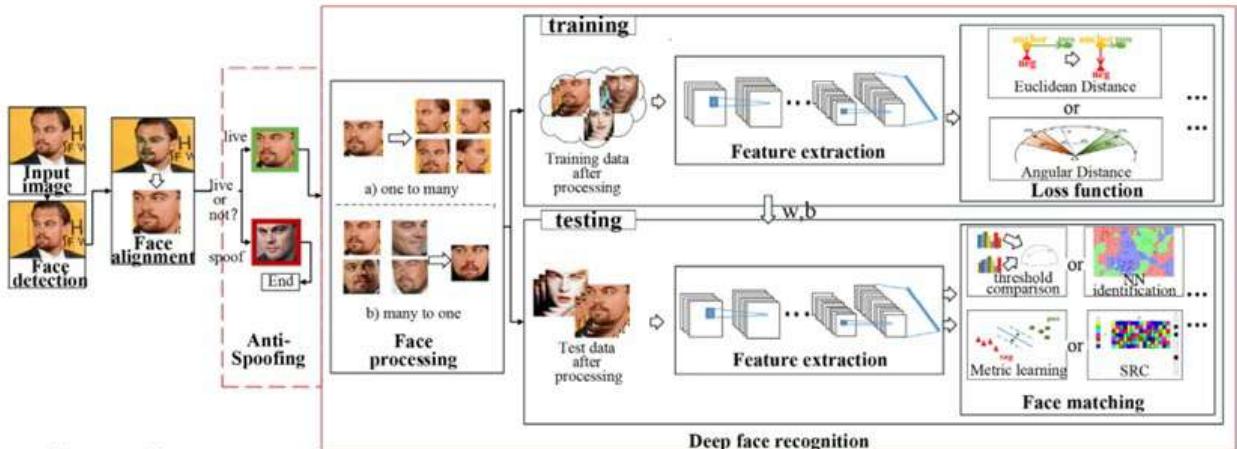


Figura 3. Sistem de recunoaștere facială utilizând metode de învățare profundă [27].

### 4.3.1 Preprocesarea imaginilor faciale

Recunoașterea facială automată implică detectarea și extragerea feței din imagini sau secvențe video complexe, ca etapă preliminară necesară procesului de identificare sau verificare a identității. Pentru realizarea acestui proces, se aplică inițial algoritmi specializați de detecție facială, care identifică poziția feței în imagine sau în cadrul video. Ulterior, sunt extrase trăsăturile relevante, iar, optional, regiunea facială este aliniată într-un format standardizat pentru a optimiza precizia procesului de recunoaștere.

Datorită complexității ridicate a trăsăturilor faciale, sunt utilizate arhitecturi neuronale profunde, care au ca scop îmbunătățirea acurateței detectării faciale în diferite condiții, cum ar fi unghiuri necorespunzătoare, expresii faciale sau condiții de iluminare slabă. Un exemplu eficient este modelul MTCNN (eng. Multi-task Cascaded Convolutional Neural Network), care utilizează o arhitectură în cascadă formată din trei rețele neuronale convoluționale, capabile să detecteze simultan poziția feței și a trăsăturilor esențiale. Modelul NMS (eng. Non-Maximum Supression) este folosit pentru a elimina detecțiile multiple redundante, păstrând doar predicția cea mai relevantă.

După detecție, imaginile faciale pot prezenta variații de formă din cauza poziției capului sau a altor factori externi, care pot afecta acuratețea recunoașterii. Pentru a reduce din aceste variații se folosesc tehnici de aliniere facială, standardizând poziția și unghiul feței din imagine. Această tehnică de corecție se realizează prin transformări 2D, sau 3D în modelele mai avansate precum DeepFace.

Detectarea anti-spoofing este esențială în domeniile unde securitatea este prioritara, prevenind tentativele de fraudă. Prin metodele utilizate se regăsesc atât abordări clasice, cum ar fi Histogram of Oriented Gradients (HOG), cât și soluții moderne bazate pe rețele neuronale convoluționale [26].

### **4.3.2 Procesarea imaginilor faciale**

Pentru a diminua efectele variațiilor naturale între imaginile faciale, sunt aplicate tehnici de procesare de tip one-to-many și normalizare de tip many-to-one. One-to-many este o tehnică prin care dintr-o singură imagine sunt generate mai multe versiuni ale feței din poziții diferite, antrenând astfel rețea. Many-to-one este o tehnică de normalizare unde mai multe imagini ale aceleiași persoane sunt transformate într-o reprezentare frontală, standardizată, facilitând recunoașterea ulterioară [27].

### **4.3.3 Antrenarea modelului**

Modelele de recunoaștere facială utilizează două arhitecturi de rețele neuronale: rețele backbone și rețele compuse. Rețelele backbone, cum sunt AlexNet, GoogleNet, ResNet, sunt folosite ca bază pentru extragerea caracteristicilor. Rețelele compuse sunt structuri mai complexe care combină mai multe rețele pentru a integra mai multe surse de informație din surse diferite, care cresc performanța generală.

În procesul de antrenare, rețea este ghidată de funcții de pierderi, care reduc eroarea de clasificare. Funcția standard Softmax Loss utilizată în clasificare nu poate separa eficient clasele în recunoașterea facială. Din această cauză au fost dezvoltate multiple variante de funcții îmbunătățite. Funcțiile de pierderi pe bază de distanță Euclidiană regleză direct distanțele dintre vectorii trăsăturilor asociate aceleiași persoane. Funcțiile de pierderi cu marjă unghiulară sau cosinus adaugă penalizări unghiulare între vectori din clase diferite, forțând rețea să genereze reprezentări bine delimitate. Funcția de pierdere de tip centru reduce dispersia vectorilor pentru aceeași reprezentare facială, atrăgând vectorii către centrul comun al clasei respective [27].

### **4.3.4 Testarea modelului**

După antrenarea modelului, acesta este utilizat pentru a evalua imagini noi, necunoscute anterior de rețea. Imaginea de test este supusă aceleiași preprocesări ca și cele din setul de antrenare, apoi introdusă în rețea neuronală antrenată. Aceasta generează un vector de trăsături, numit vector embedding, de dimensiune fixă, în care sunt codificate informațiile faciale esențiale pentru fiecare persoană. După extragerea acestui vector, sistemul compară această reprezentare cu alți vectori embedding deja existenți în rețea. Distanța Euclidiană este o metrică care calculează distanța geometrică directă dintre vectorii embedding, o distanță mică sugerând o probabilitate mare ca cele două imagini să reprezinte aceeași persoană. Similitudinea cosinus evaluatează unghiul dintre doi vectori pentru a evalua cât de asemănătoare sunt direcțiile acestora, însă este o metodă mai sensibilă la scalare vectorială.

Pe baza acestor comparații, sistemul stabilește dacă imaginea testată corespunde sau nu unei fețe cunoscute din rețea [27].

## **4.4 Modelul YOLO (You Only Look Once)**

YOLO reprezintă un algoritm de detecție a obiectelor care primește o imagine de intrare și transformă direct acea imagine, într-o singură trecere, în coordonatele obiectelor detectate și probabilitățile claselor din care fac parte.

Modelul împarte imaginea de intrare într-o grilă de dimensiune  $S \times S$ . Fiecare celulă a grilei este responsabilă pentru detectarea obiectelor al căror centru se încadrează în acea celulă. Pentru fiecare celulă se prezic un număr fix de cutii delimitatoare, B, iar pentru fiecare cutie sunt estimate coordonatele centrului ( $x, y$ ), lățimea ( $w$ ), înălțimea ( $h$ ) și un scor de încredere. În plus, sunt generate

și probabilități condiționate pentru apartenența obiectelor la clase specifice, permitând maparea directă a pixelilor imaginii de intrare către un set de predicții structurale [28].

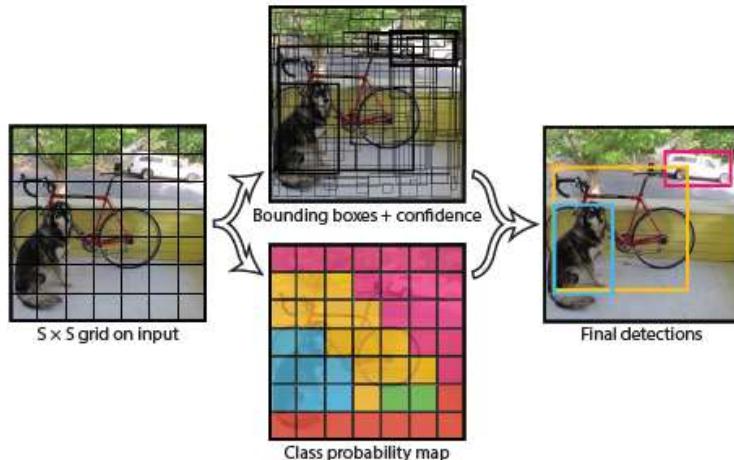


Figura 4. Etapele de detecție a obiectelor ale modelului YOLO [28].

Modelul YOLO este implementat sub forma unei rețele neuronale conoluționale, proiectat pentru a funcționa într-un regim end-to-end, de la imaginea brută până la predicția cutiilor de detecție și a claselor de obiecte. Arhitectura este inspirată din modelul GoogLeNet utilizat pentru clasificarea imaginilor, dar cu adaptări care o fac potrivită sarcina de detecție în timp real.

Rețeaua YOLO standard este alcătuită din 24 de straturi conoluționale, urmate de două straturi complet conectate. Straturile conoluționale sunt responsabile de extragerea automată a caracteristicilor vizuale relevante din imagine, cum ar fi margini sau forme, în timp ce straturile complet conectate realizează predicția finală, coordonatele cutiilor delimitatoare și probabilitățile claselor.

YOLO oferă avantaje fundamentale. Este extrem de rapid, datorită faptului că totul se calculează într-un singur pas și, de asemenea, are o viziune globală asupra întregii imagini, atât în timpul antrenării, cât și în cel al testării [28].

## 4.5 Microcontrolere

Microcontrolerul este un microcalculator integrat într-un singur chip, fabricat prin tehnologia VLSI (Very Large Scale Integration), adesea întâlnit sub denumirea de sistem integrat (eng. embedded system). Acesta conține atât unitatea centrală de procesare (eng. CPU-Central Processing Unit), cât și un set de componente periferice. Structura generală a unui microcontroler include următoarele componente:

- CPU – reprezintă componenta care preia instrucțiuni din memorie, le decodează și apoi le execută, fiind adesea numit „creierul” microcontrolerului.
- Memoria – utilizată pentru stocarea codului de program și a datelor în timpul execuției. Memoria poate fi RAM, ROM (EPROM, EEPROM) sau Flash, în funcție de natura aplicației.
- Porturi de intrare/ieșire (I/O) – porturi digitale care permit conectarea microcontrolerului la dispozitive externe.
- Interfețe de comunicare serială – permit transferul de date între microcontroler și periferice prin protocole ca UART (Universal Asynchronous Receiver/Transmitter), SPI (Serial Peripheral Interface), I2C (Inter Integrated Circuit), etc.
- Generatorul de ceas (Clock) – generează semnalul de sincronizare al microcontrolerului, care poate fi realizat cu cristal de quarț, un rezonator ceramic sau un ceas intern bazat pe un circuit RC.

- Convertoare analog-digitale (A/D) – transformă semnalele analogice în semnale digitale pentru a putea fi procesate intern.
- Convertoare digital-analogice (D/A) - generează ieșiri analogice pe baza unor valori digitale.
- Reset-ul – reduce microcontrolerul în starea inițială de funcționare.
- Detector Brown-Out – are rol de a monitoriza și reseta microcontrolerul dacă tensiunea de alimentare a acestuia scade sub valoarea nominală.
- Watchdog-ul – circuit de siguranță care previne blocarea programului, forțând o resetare în caz de comportament anormal.
- Timere – sunt folosite pentru a controla operațiile de numărare și temporizare ale microcontrolerului.
- Întreruperi – reprezintă mecanismul prin care un microcontroler poate opri un program pentru a răspunde unui eveniment important [29] [30].

#### **4.5.1 Tipuri de microcontrolere**

Microcontrolerelor pot fi clasificate în funcție de lățimea magistralei de date, fiind disponibile în versiuni de 4, 8, 16 sau 32 de biți în funcție de câți biți pot procesa simultan la nivelul unității de date.

În funcție de memoria de program, microcontrolerelor pot fi clasificate în mai multe categorii. Există microcontrolere care nu au memorie de program integrată, fiind dependente de o memorie externă pentru a stoca codul executabil. O altă categorie sunt microcontrolerelor OTP (eng. One Time Programmable), care pot fi programate o singură dată. Dacă programul nu este corect, acestea nu mai pot fi utilizate. Microcontrolerelor cu memorie ROM programată direct de producător sunt o soluție eficientă și mai ieftină, fiind adecvate în aplicații cu volume mari de producție. O variantă mai flexibilă reprezintă microcontrolerelor cu memorie de tip Flash, care au devenit tot mai populare pe piață datorită capacitatea de a fi reprogramate de mai multe ori. Aceste microcontrolere au un cost mai ridicat față de celelalte tipuri, însă sunt mult mai versatile și eficiente în procesul de dezvoltare.

În structura internă a microprocesoarelor și microcontrolerelor se regăsesc două tipuri de arhitecturi de organizare a memoriei: arhitectura Von Neumann și arhitectura Harvard. Arhitectura Von Neumann presupune utilizarea spațiului de memorie atât pentru codul de program, cât și pentru date. Această arhitectură este întâlnită în sistemele clasice de calcul, cum ar fi arhitectura PC, dar și în unele microcontrolere ca cele din seria HC05 de la Motorola sau a familiei C16X de la Infineon. Arhitectura Harvard se diferențiază prin separarea memoriei de program de cea de date. Această separare permite procesorului să acceseze simultan instrucțiuni de date. De asemenea, permite proiectarea în care dimensiunea unității de cod este diferită de lungimea unității de date. Majoritatea microcontrolerelor moderne adoptă arhitectura Harvard, fiind considerată mai eficientă.

Setul de instrucțiuni al unui procesor poate fi de tip CISC (Complex Instruction Set Computer), unde instrucțiunile pot executa operații multiple, reducând numărul liniilor de cod, sau de tip RISC (Reduced Instruction Set Computer), care utilizează un set redus de instrucțiuni simple, optimizate pentru execuție rapidă [29] [30].

#### **4.5.2 Diferențe dintre microprocesoare și microcontrolere**

Între un microcontroler și un microprocesor, deși par termini similari, există diferențe fundamentale. Deoarece microprocesorul are doar unitatea centrală (CPU), el necesită o conectare separată a memoriei și a perifericelor. Pe de altă parte, microcontrolerul integrează pe același cip atât CPU, cât și memoria și perifericele necesare.

Din punct de vedere structural, microcontrolerul deja integrează, pe lângă CPU, memoria și perifericele, fiind o soluție simplă, compactă, din punct de vedere a proiectării unor sisteme, dar

și eficientă din punct de vedere al alimentării, necesitând de regulă o singură alimentare. Însă proiectând un sistem cu un microprocesor, deoarece are nevoie de periferice externe, acesta poate să devină mai voluminos, fiind necesare mai multe linii de alimentare diferite pentru periferice.

Din punct de vedere funcțional, microprocesoarele nu dispun de memorie RAM integrată și folosesc pentru procesarea datelor memorie DRAM externă pentru execuția semnalului de la intrare, iar apoi intervine memoria ROM internă. Ca urmare, microprocesorul necesită mai mult timp până oferă un rezultat. Însă posibilitatea extinderii memoriei RAM reprezintă un avantaj al microprocesorului. Pe de altă parte, microcontrolerele dispun de memorie flash integrată, asigurând o pornire și execuție rapidă a aplicațiilor datorită timpului de acces redus, singurul dezavantaj fiind capacitatea limitată a memoriei.

Din punct de vedere al conectivității, microprocesoarele necesită conectivități precum USB, Bluetooth, Wi-Fi putând gestiona transferuri mari de date sau să ruleze sisteme de operare complexe. Microcontrolerele nu pot, de regulă, integra interfețe suplimentare, prin urmare numărul de intrări/ieșiri și viteza de transfer sunt reduse, astfel flexibilitatea sistemelor care au la bază microcontrolere este mai redusă.

Din punct de vedere al consumului energetic, microprocesoarele au nevoie de mai multă energie datorită componentelor externe, iar microcontrolerele au un consum redus de energie datorită integrării tuturor funcțiilor pe un singur chip [31].

#### 4.5.3 ESP32-CAM

ESP32-CAM este un modul care integrează o cameră video, dezvoltat de Essence. Acesta poate funcționa ca un sistem embedded, având dimensiuni de doar 27\*40.5\*4.5mm. Acesta este un microcontroler capabil să proceseze imagini fiind utilizat pe scară largă în diverse aplicații IoT, cum ar fi dispozitive smart house, control și monitorizare wireless, identificare facială și multe altele [32].



Figura 5. Modulul ESP32-CAM [32]

ESP32-CAM oferă o memorie RAM DE 512 KB SRAM și 4 MB PSRAM și o memorie Flash de 32 Mbit. Are conexiune Bluetooth clasica BR/EDR, dar și conexiune cu consum redus, BLE. Suportă interfețe UART, SPI, I2C, PWM și card TF cu suport maxim de 4G. Viteza de transmitere a datelor prin UART este implicit 115200 bps. Dispune de Wi-Fi 802.11 b/g/n pentru conectarea la rețele wireless și o securitate WPA/WPA2/WPA2-Enterprise/WPS. Necesită o tensiune de alimentare de 5V, iar consumul de energie al dispozitivului variază în funcție de starea în care se află. Astfel că, atunci când lampa flash este oprită, consumă un curent de 180mA și tensiune de alimentare de 5V, iar când lampa flash este pornită cu luminozitate maximă, modulul consumă 310mA și 5V. În starea de deep-sleep are un consum minim de 6mA și 5V, în starea de modern-sleep consumă minim 20mA și 5V, iar în starea de light-sleep are un consum de 6.7mA și

5V, ceea ce îl face perfect pentru sisteme cu alimentare limitată. De asemenea, poate opera într-un domeniu larg de temperatură, de la -20°C până la aproximativ 85°C.

ESP32-CAM dispune de 16 pini de conexiune, dintre care 9 sunt porturi de intrare/ieșire care pot servi unor funcții externe [32] [33].

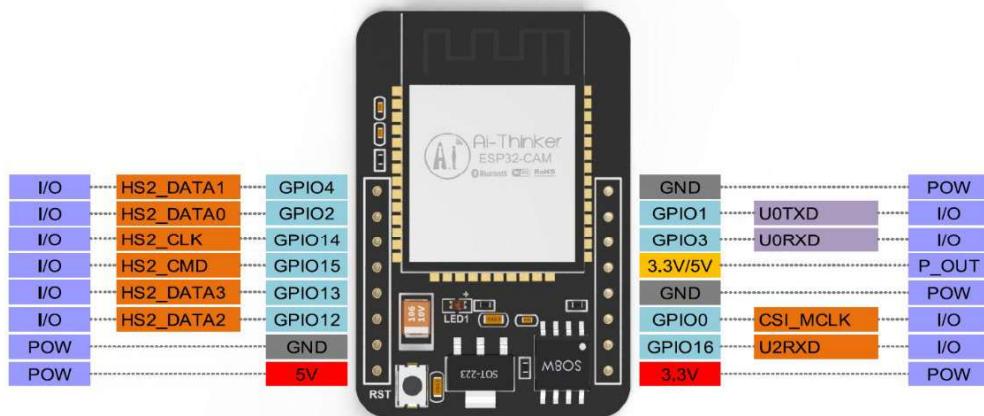


Figura 6. Diagrama pinilor de conectare pentru modulul ESP32-CAM [32].

Modulul ESP32-CAM include senzorul de imagine OV2640, un senzor CMOS de joasă tensiune, încorporat într-un format cu dimensiuni reduse, care oferă o rezoluție de 1632x1232 pixeli, corespunzătoare standardului UXGA, precum și un procesor de imagine. Reușește să opereze cu până la 15 cadre pe secundă (fps) la această rezoluție, cu ajutorul unei matrice de imagini. Utilizatorul are control complet asupra calității imaginii, formatării și transferului de date la ieșire, de asemenea poate programa toate funcțiile necesare procesării imaginii, cum ar fi luminozitatea, contrastul, tonul culorii, balansul de alb, corecția gama, reducerea de zgomot și multe altele, prin interfața Serial Camera Control Bus (SCCB). Controlat prin interfața SCCB, OV2640 oferă imagini full-frame, eșantionate și scalate la 8 sau 10 biți într-o varietate de formate. Formatele de ieșire suportate sunt: Raw RGB, RGB (555/565), GBR422, YUV (422/420) și YCbCr (4:2:2), inclusiv JPEG și greyscale. OV2640 utilizează o tensiune de 1,2V pentru nucleul de procesare (core), între 2,5V și 3V pentru circuitele analogice și aproximativ 1,7-3,3V pentru interfețele de date de intrare/ieșire [34].

## 4.6 Protocoale

### 4.6.1 Protocolul Universal Asynchronous Receiver Transmitter (UART)

UART reprezintă un protocol de comunicație serială asincronă, fiind utilizat în sistemele embedded pentru transferul de date între două dispozitive digitale. Reprezintă un protocol universal, permite configurarea datelor și a vitezelor de transmisie și nu necesită semnal de ceas pentru transferul de date.

Protocolul UART utilizează două fire, unul pentru transmisie (TX) și unul pentru receptie (RX). Transmiterea datelor se realizează prin conectarea liniei TX a unui dispozitiv la linia RX a celuilalt dispozitiv, ceea ce permite implementarea comunicării seriale un număr redus de conexiuni fizice. Acest lucru reprezintă un avantaj față de comunicația paralelă care necesită un număr mai mare de conexiuni pentru liniile de date [35].

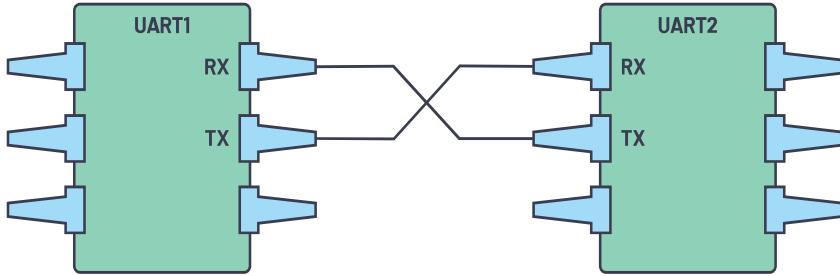


Figura 7. Comunicarea UART dintre două dispozitive [36].

Modul de transmitere a datelor se face sub forma unor pachete seriale. Pachetul transmis prin UART este alcătuit din: bitul de start, cadrul de date, bitul de paritate și bitul de stop.

Bitul de start reprezintă informația de sincronizare care anunță începutul transmiterii unui nou cadrul de date, care comută linia de date a emițătorului de la un nivel logic ridicat, la un nivel logic scăzut. Receptorul detectează tranziția, iar procesul de citire a bițiilor ce urmează se realizează în funcție de viteza de transmisie configurată.

Cadrul de date conține datele transferate. Poate avea o dimensiune cuprinsă între 5 și 8 biți când este folosit bitul de paritate, iar dacă nu, poate conține un număr de 9 biți. Datele se transmit în general începând cu cel mai nesemnificativ bit.

Bitul de paritate are rol de detecție a modificărilor din cadrul transmisiei, optional însă pentru protocolul UART. Dacă numărul total al bițiilor de 1 este un număr par, paritatea este pară, iar dacă numărul bițiilor de 1 este un număr impar, paritatea este impară.

Bitul de stop marchează finalul transmisiei datelor, menținând linia de transmisie a datelor la o tensiune joasă. Poate avea o lungime de la unu, până la doi biți [36].

Baut rate-ul este viteza cu care datele sunt transmise în cadrul comunicației UART. Se exprimă în biți pe secundă, iar emițătorul și receptorul trebuie configurate la aceeași valoare pentru un transfer de date corect.

Principalul avantaj al protocolului UART este că necesită doar două fire pentru transmisia datelor. De asemenea, transmisia asincronă elimină necesitatea unui semnal de ceas sau a altor semnale de sincronizare suplimentare.

Printre dezavantaje se numără dimensiunea limitată a cadrului de date și viteza de transmisie mai redusă comparativ cu cea a comunicației paralele. Pe lângă asta, transmițătorul și receptorul trebuie configurate corespunzător, cu aceleași setări de comunicație, inclusive a vitezei de transmisie [37].

#### 4.6.2 Serial Peripheral Interface (SPI)

SPI reprezintă un protocol de comunicație serială, utilizat pentru realizarea transferului de date între microcontroler și diverse dispozitive periferice. Deoarece comunicația se realizează sincron, datele transmise și receptate sunt realizate sincron coordonate de un semnal de ceas.

Protocolul permite transmisia bidirectională simultană de date, full-duplex. Dispozitivul master este componenta care controlează fluxul de date, generează semnalul de ceas și inițiază transferul de informații, această funcție fiind preluată în majoritatea cazurilor de microcontroler. Dispozitivele slave reprezintă perifericele conectate la magistrală și răspund comenziilor inițiate de master. SPI utilizează patru linii principale pentru realizarea comunicației: slave select (SS), master out slave in (MOSI), master in slave out (MISO) și serial clock (SCK). Pentru ca transferul informațiilor să fie sincronizat, fiecare bit de date este trimis la fiecare impuls de ceas generat de master. Viteza de transfer a datelor poate varia în funcție de capacitatea dispozitivului master, fiind exprimată de regulă în mega biți pe secundă (Mbps) sau megahertz (MHz). Parametrii Clock polarity (CPOL) și Clock Phase (CPHA) sunt utilizati pentru a determina relația dintre semnalele de date și semnalul de ceas.

Protocolul SPI este utilizat în sisteme embedded, aplicații IoT (Internet of Things), unde sunt necesare conexiuni cu senzori și interfațare cu module de comunicații wireless, cum ar fi Wi-Fi, Bluetooth sau alte echipamente care necesită transmisii de date rapide [38].

#### 4.6.3 Inter-Integrated Circuit (I2C)

I2C este un protocol de comunicație serială sincronă, utilizat pentru conectarea mai multor dispozitive periferice la unul sau mai multe dispozitive master. Protocolul îmbină atât avantajele oferite de SPI, adică poate controla mai multe dispozitive slave, cât și avantajele oferite de UART, deoarece realizează transmisia datelor utilizând două linii. Serial Data (SDA) reprezintă linia utilizată pentru transmiterea și recepționarea datelor. Serial Clock (SCL) este linia care transmite semnalul de ceas generat de dispozitivul master.

Comunicația în cadrul protocolului se face sub forma unor mesaje, împărțite în cadre. Mesajul include: condiție de start, cadru de adresă, bit de citire/scrivere, bit ACK/NACK, unul sau mai multe cadre de date și condiție de stop.

Bitul de start semnalează inițierea unei transmisii, unde linia SDA trece de la nivel logic ridicat la unul scăzut, iar linia SCL rămâne la nivel logic ridicat. Cadrul de adresă reprezintă o secvență de șapte sau zece biți prin care dispozitivul master specifică adresa binară a celui slave, care compară adresa primită cu cea proprie, iar dacă există corespondențe răspunde printr-un bit ACK. Dacă adresarea nu corespunde, dispozitivul rămâne inactiv și menține linia SDA la nivel logic ridicat. Bitul de scriere/citire specifică dacă se trimit sau se cer date de la master. Fiecare cadru de date este format din opt biți, transmiși începând cu cel mai semnificativ bit. După fiecare cadru urmează un bit ACK/NACK, care verifică dacă cadrul a fost recepționat corect. Transmisia continuă până la trimiterea condiției de stop de către master, care semnalează încheierea comunicării.

I2C reprezintă o metodă eficientă de comunicație hardware, unde pot exista mai mulți masteri într-un sistem, însă are o rată de transfer mai scăzută în comparație cu protocolul SPI, iar implementarea protocolului presupune o complexitate hardware mai ridicată pentru a gestiona adresarea și controlul magistralei [39].

### 4.7 Senzori IR

Senzorul infraroșu (IR) reprezintă un dispozitiv electronic capabil să detecteze anumite caracteristici ale mediului înconjurător, fie prin detectarea radiației infraroșu, fie prin emiterea ei. Acești senzori pot fi utilizati pentru a detecta prezența sau mișcarea obiectelor, precum și pentru a identifica variații de temperatură. Circuitele bazate pe senzori infraroșu sunt printre cele mai întâlnite și utilizate în domeniul electronicii.

Senzorii de tip pasiv (PIR) detectează radiația infraroșie emisă de corpuri, sub forma de radiație termică. Indiferent de temperatură, toate obiectele emit o cantitate de radiație în spectrul infraroșu, invizibilă ochiului uman, detectată cu ajutorul acestor senzori. Senzorii activi funcționează pe baza unui emițător, de regulă o diodă LED care emite lumina infraroșie și unui detector, o fotodiодă, sensibilă la radiația emițătorului, care-și modifică rezistență în funcție de intensitatea luminii detectate.

În funcție de aplicațiile în care sunt utilizati, cele mai comune tipuri de senzori infraroșu sunt: senzori de temperatură, senzori ultrasonici și senzori PIR [40].

#### 4.7.1 Senzori de temperatură

Senzorii de temperatură cu infraroșu sunt dispozitive care, prin măsurarea radiației termice, determină temperatura unui obiect. Acești senzori utilizează un fascicul laser cu care pot indica

precis zona de măsurare, de asemenea nu necesită atingerea pentru a măsura temperatura corpului respectiv, reușind să determine temperatura de la distanță. Din acest motiv, sunt senzori utili în situații în care este dificilă utilizarea sau măsurarea cu senzorii de contact [40].

#### 4.7.2 Senzori ultrasonici

Senzorii ultrasonici sunt dispozitive, care prin emiterea de unde sonore, sunt capabili să determine distanța față de un obiect. Detecția se realizează prin transmiterea unei unde sonore, la o frecvență specifică și măsurarea timpului necesar până ce această undă ajunge pe obiect, se reflectă și se întoarce înapoi la senzor. Totuși unele obiecte nu pot fi detectate corect de senzorii ultrasonici datorită formei sau poziționării, care nu permit ca unda sonoră să fie reflectată la senzor, ci în alte direcții [40].

#### 4.7.3 Senzori PIR

Senzorii PIR detectează variația radiației infraroșie, mai exact măsoară modificările nivelului de radiație provenită din mediul înconjurător. Cantitatea de variație a radiației măsurate variază în funcție de temperatură și proprietățile obiectelor aflate în câmpul detectat. Acest principiu de funcționare este mai complex decât al altor tipuri de senzori, deoarece semnalul lor este influențat de mai mulți factori externi.

Atunci când o persoană ajunge în raza senzorului, acesta detectează temperatura corpului uman, iar dacă persoana părăsește zona, temperatura detectată de senzor scade înapoi la cea a mediului ambiental. Senzorul reacționează atunci când un corp cald intră în aria de detecție și generează inițial un impuls diferențial pozitiv, iar dacă corpul se deplasează din aria activă, senzorul generează un impuls diferențial negativ. În absența diferențelor de temperatură, senzorul detectează în mod constant radiații de la corpurile statice mediului, însemnând absența mișcării.

Un senzor PIR are două zone active piroelectrice de detecție, realizate din materiale speciale, cum ar fi ceramice piroelectrice, dispuse diferențial, ambele sensibile la variația rapidă de radiație infraroșie. În fața acestor zone se montează o lentilă specială care să mărească câmpul de detecție al senzorului. În majoritatea cazurilor, senzorii PIR sunt dotati cu lentile Fresnel, fabricate din plastic, care permit detecția într-un unghi de 180 de grade, sesizând mișcare pe o distanță mai mare. Fără aceste lentile senzorul ar percepe doar radiația corespunzătoare celor două zone, pe o zonă restrânsă de detecție [41].

## 5 Implementarea soluției adoptate

Lucrarea propune un sistem care să permită accesul unui utilizator folosind recunoașterea facială, cu avertizare în cazul detecției unui obiect periculos. Procesul începe în momentul în care este detectată mișcare de către un senzor PIR, apoi se activează camera modului ESP32-CAM pentru a captura imagini. Aceste imagini sunt analizate de aplicația Python, unde sunt aplicăti algoritmi de recunoaștere și detecție de obiecte. Dacă persoana este recunoscută în baza de date, este generată o semnalizare sonoră și se permite accesul. Dacă persoana nu este recunoscută sau se detectează un obiect periculos, se generează un alt tip de avertizare sonoră.

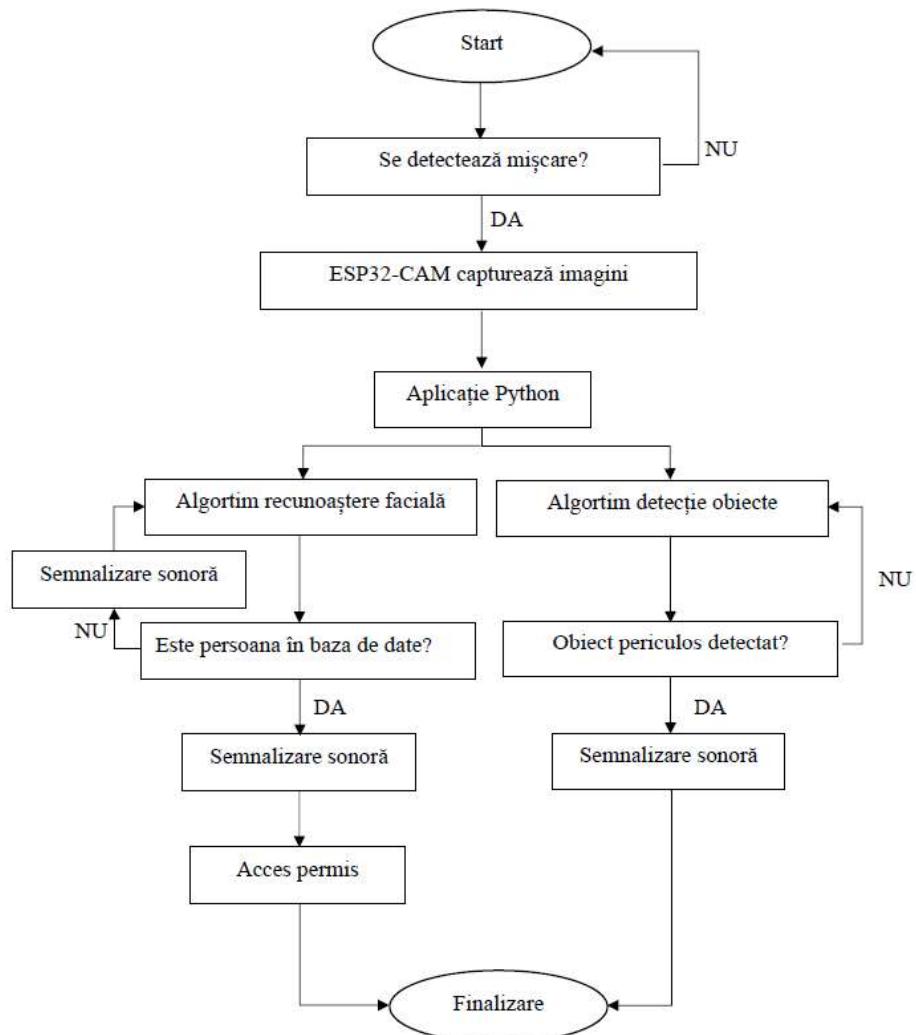


Figura 8. Diagrama bloc a sistemului

### 5.1 Implementarea folosind modulului ESP32CAM

Microcontrolerul ales pentru implementarea hardware este ESP32-CAM. Este un modul care combină performanță, costul redus și funcționalitate integrată, utilizat frecvent în proiecte embedded. Dispune de un modul de cameră OV2640 deja integrat și oferă conexiuni Wi-Fi și Bluetooth, fiind potrivit pentru proiecte care implică transmisie de imagini, video sau date prin rețea, fară a necesita o cameră externă suplimentară. Consumul de energie este unul redus, iar prin activarea modului de *deep sleep* poate atinge un consum de doar 6mA.



Figura 9. ESP32-CAM

Arduino Uno reprezintă o soluție simplă, ideală pentru aplicații de control digital sau achiziția de date de la senzori. Nu are însă conectivitate wireless în mod implicit și nici nu suportă direct camere video. De asemenea, este mai limitat în putere de procesare și memorie internă.

Raspberry Pi este un microcontroler capabil totuși să ruleze un sistem de operare complet, fiind potrivit pentru aplicații complexe. Are un procesor performant și o memorie mult mai mare, însă nu dispune direct de un modul de cameră, are un consum energetic mai ridicat, dimensiuni mai mari și costul mult mai ridicat față de ESP32-CAM.

Modulul ESP32-CAM nu dispune de interfață USB, de aceea pentru programarea lui a fost utilizat un adaptor serial de tip FT232RL. Acest adaptor permite conversia semnalelor USB în semnale UART compatibile cu ESP32-CAM.

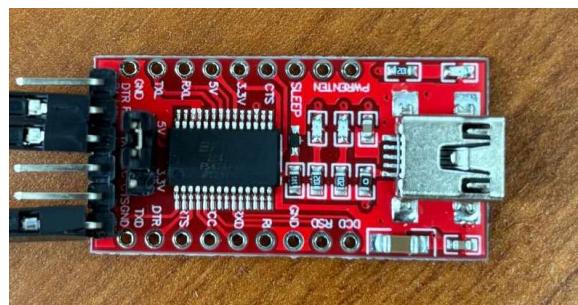


Figura 10. Modul FT232RL

Pentru a programa placa se conectează pinii TX, RX, GND și 5V la placa ESP32-CAM, iar pentru inițializarea modului în modul de lucru *programare*, se conectează pinul IO0 la GND printr-un jumper. Odată încărcat codul, se scoate jumperul și se apasă butonul de reset de pe ESP32-CAM ca să se repornească în mod normal de execuție, rulând codul încărcat.

Pentru dezvoltarea codului care rulează pe placa ESP32-CAM, a fost utilizat Arduino IDE, un mediu de dezvoltare integrat simplu și accesibil, dedicat microcontrolerelor compatibile cu ecosistemul Arduino. Oferă suport pentru limbajul C/C++ și o bibliotecă extinsă de funcții pentru controlul componentelor hardware. În cadrul proiectului, a fost selectată placa „AI Thinker ESP32-CAM”, esențială în compilarea adecvată a codului.

Pentru a permite ca modulul să interacționeze cu alte dispozitive prin rețea, a fost necesară conexiunea acestuia la Wi-Fi. ESP32-CAM se conectează la o rețea wireless și acționează în modul client, conectat la un router deja existent. Pentru asta a fost necesară includerea bibliotecii „WiFi.h” în fișierul programului. S-au definit două variabile pentru SSID și Password care stochează numele și parola rețelei Wi-Fi și s-au creat funcții care inițiază procesul. Prinț-o buclă infinită se repetă execuția până la realizarea conexiunii, afișând puncte la fiecare 500ms.

```

void initWiFi() {
    WiFi.persistent(false);
    WiFi.mode(WIFI_STA);
    WiFi.begin(WIFI_SSID, WIFI_PASS);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.printf("\nhttp://%s%s\n", WiFi.localIP().toString().c_str(), URL);
}

```

Figura 11. Funcția care permite conectarea modulului la Wi-Fi.

Configurarea corectă a camerei a fost necesară pentru a obține performanțe bune în ceea ce privește calitatea imaginii. A fost inclusă librăria „esp32cam.h”, care permite controlul camerei OV2640, pentru definirea pinilor de conectare, ajustarea rezoluției sau a calității imaginilor, captarea cadrelor sau gestionarea memoriei. Rezoluția aleasă a imaginii este de 640x480, deoarece oferă atât claritate bună a imaginii, cât și un consum redus de memorie RAM. Pentru a gestiona cadrele captureate, numărul de buffere pentru transmisia unui flux video este de două. Compresia JPEG setată la valoarea de 75 oferă un echilibru bun între calitate și performanță. Odată ce parametrii au fost configurați se realizează inițializarea camerei, care returnează true sau false în funcție de succes sau eșec.

```

void initCamera() {
    using namespace esp32cam;
    Config cfg;
    cfg.setPins(pins::AiThinker);
    cfg.setResolution(RES);
    cfg.setBufferCount(2);
    cfg.setJpeg(75);
    bool ok = Camera.begin(cfg);
    Serial.println(ok ? "CAMERA OK" : "CAMERA FAIL");
}

```

Figura 12. Funcția care inițializează camera modulului.

Inițializarea serverului web are un rol important în cazul modulului ESP32-CAM pentru vizualizarea camerelor sau a altor dispozitive conectate. Un server web este responsabil de gestionarea cererilor HTTP primite de la clienți și returnarea de răspunsuri care pot include un flux de imagini. Pentru a crea un server HTTP pe ESP32-CAM, se include biblioteca „WebServer.h”. Serverul web este ales să funcționeze pe portul 80, un port standard utilizat de protocolul HTTP, care permite utilizatorului să accese interfața web direct din browser, fără a specifica un port personalizat. După ce se definesc toate rutele necesare, se pune serverul în funcțiune pentru a fi disponibil cererilor din rețea. Serverul este menținut activ în cadrul unei funcții loop pentru a verifica dacă există cereri noi de la clienti.

Funcția serveJpg() are rolul de a prelua o imagine de la camera integrată ESP32-CAM. Aceasta este apelată în momentul în care serverul web primește o cerere pentru imagine. Dacă operația de capturare a unui cadru de la cameră eșuează, se afișează un mesaj de eroare pe portul serial și se transmite către client un cod HTTP 503. Dacă imaginea este capturată cu succes, funcția trimite un răspuns HTTP 200 cu imaginea.

```

void serveJpg() {
    auto frame = esp32cam::capture();
    if (frame == nullptr) {
        Serial.println("CAPTURE FAILED");
        server.send(503, "text/plain", "Camera indisponibila");
        return;
    }
    Serial.printf("CAPTURE OK %dx%d %db\n", frame->getWidth(), frame->getHeight(), static_cast<int>(frame->size()));
    server.setContentLength(frame->size());
    server.send(200, "image/jpeg");
    WiFiClient client = server.client();
    frame->writeTo(client);
}

```

Figura 13. Funcția care servește imagini de la ESP32-CAM.

Pentru a vizualiza toate rezultatele pe monitorul serial, se stabilește conexiunea serială la 115200 baud.



Figura 14. Serverul modulului afișat pe monitorul serial

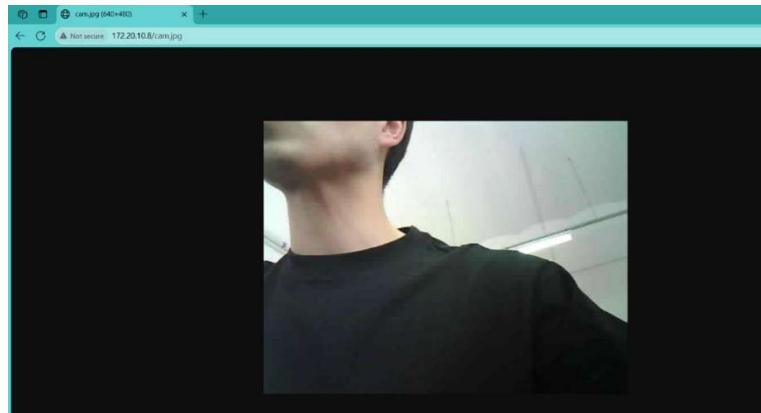


Figura 15. Testarea serverului care servește imagini de la camera modulului.

## 5.2 Implementarea aplicației în Python

Pentru dezvoltarea aplicației care include partea de procesare de imagini, comunicare cu baza de date și gestionarea interfeței web s-a utilizat limbajul de programare Python. Codul a fost configurat în mediul de dezvoltare Spyder, care este inclus în distribuția Anaconda.

Python este un limbaj de programare interpretat, de nivel înalt, cu sintaxă clară și ușor de învățat, care a devenit foarte popular în ultimii ani în domenii precum inteligența artificială sau dezvoltare web. Python oferă o gamă largă de biblioteci specializate, ceea ce permite dezvoltarea rapidă a aplicațiilor complexe.

OpenCV este o bibliotecă open-source, esențială aplicațiilor de viziune computerizată și procesare de imagini. În cadrul proiectului, OpenCV este utilizată pentru capturarea și decodarea imaginilor provenite de la ESP32-CAM, pentru conversii de culoare, pentru trasarea de elemente vizuale pe imagini, dar și pentru lucrul cu rețelele neuronale. NumPy este o altă bibliotecă, esențială pentru procesarea numerică în Python. Oferă structuri de date eficiente pentru lucrul cu matrici și vectori, precum și un set extins de funcții matematice. În cadrul aplicației, NumPy este folosit pentru conversia datelor binare din imagini în vectori faciali și pentru manipularea imaginilor sub formă de matrici. Astfel, combinația dintre OpenCV și NumPy permite prelucrarea performantă a datelor vizuale, compatibil cu cerințele metodelor de inteligență artificială.

Pentru a dezvolta aplicația la nivel web s-a utilizat microframework-ul web Flask. Acesta are un control complet asupra arhitecturii aplicației. Flask oferă suport pentru gestionarea adreselor URL, integrare cu HTML și alte funcționalități necesare unui sistem web. În aplicația dezvoltată, Flask a fost utilizat pentru crearea interfeței web de administrare, preluarea și afișarea în timp real a imaginilor procesate. Aplicația rulează local, independent de un server extern, fiind pornită din terminalul Python și accesibilă printr-un browser local, la o adresă de tipul <http://localhost:5000>, unde 5000 este portul implicit utilizat de Flask.

În cadrul aplicației, imaginile utilizate pentru recunoașterea facială și detecția de obiecte sunt obținute în timp real de la camera OV2640, montată pe ESP32-CAM, conectată în rețea locală. Dispozitivul expune constant un flux de cadre JPEG, disponibil printr-o adresă IP locală, configurată în rețea. În aplicația implementată, această adresă este stocată într-o variabilă globală

ESP32\_CAMERA\_URL= http://172.20.10.8/cam.jpg. Preluarea imaginilor este realizată într-un thread separat, printr-o funcție numită camera\_worker(). Aceasta se ocupă de descărcarea, decodarea și prelucrarea fiecărui cadru furnizat de cameră. Se trimit periodic o cerere HTTP către ESP32-CAM utilizând biblioteca *urllib.request*, cu un timeout de două secunde pentru a evita blocarea în cazul unei deconectări. Imaginea primită sub formă de flux binar este transformată într-o matrice de octeți și apoi decodificată într-un obiect imagine OpenCV, folosind funcția cv2.imdecode(). Imediat ce imaginea este reconstruită, aceasta este trimisă funcțiilor recognize\_faces(), pentru recunoașterea facială și detect\_objects() pentru detecția obiectelor.

```
def camera_worker():
    while True:
        try:
            img_resp = urllib.request.urlopen(ESP32_CAMERA_URL, timeout=2)
            imgnp = np.array(bytearray(img_resp.read()), dtype=np.uint8)
            frame = cv2.imdecode(imgnp, -1)

            #procesare imagini
            recognize_faces(frame)
            detect_objects(frame)
```

Figura 16. Funcția care procesează imaginile primite de la ESP32-CAM.

În cazul în care transmisia nu este disponibilă, din cauza unor erori de rețea sau din cauza că ESP32-CAM este în starea deep-sleep, este utilizată o imagine de rezervă. Aceasta este creată ca imagine neagră, utilizând funcția np.zeros(). Cu ajutorul funcției cv2.putText() din biblioteca OpenCV, pe aceasă imagine este afișat mesajul „Camera indisponibilă”, scris cu font de culoare roșie.

### 5.2.1 Implementarea algoritmului de recunoaștere facială

Pentru implementarea recunoașterii faciale s-a utilizat biblioteca *face\_recognition*, construită pe baza modelului de învățare profundă Dlib. Biblioteca Dlib reprezintă un pachet software open-source dezvoltat în limbajul de programare C++. Oferă mai multe componente independente, care includ funcții pentru verificarea și depanarea codului. Arhitectura Dlib este una modulară, adică fiecare parte poate fi utilizată fără a depinde de restul bibliotecii. De-a lungul timpului, biblioteca a fost extinsă semnificativ și include în prezent componente pentru rețelistică, gestionarea firelor de execuție, interfețe grafice, procesarea imaginilor, optimizare numerică etc. Biblioteca Dlib este concepută pentru a fi ușor de integrat în proiecte, fără a avea nevoie de o instalare sau configurare specială. Pentru utilizarea din Python aceasta se poate instala rapid cu ajutorul instrumentului de instalare pip, fără pași suplimentari de compliere sau configurare manuală.

Se creează o funcție recognize\_faces(frame), responsabilă pentru procesarea fiecărei imagini capturate de ESP32-CAM. Funcția primește ca parametru un obiect frame de tip NumPy array, adică o imagine în format BGR cum este furnizat de OpenCV. Biblioteca face\_recognition necesită imagini în format RGB, de accea primul pas reprezintă conversia culorilor din BGR în RGB. După conversie imaginea este procesată cu funcția face\_recognition.face\_locations(rgb\_frame), care utilizează implicit un model HOG pentru a identifica locația fețelor prezente în poză.

```
def recognize_faces(frame):
    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    face_locations = face_recognition.face_locations(rgb_frame)
    face_encs = face_recognition.face_encodings(rgb_frame, face_locations)

    for face_encoding, loc in zip(face_encs, face_locations):
        matches = face_recognition.compare_faces(known_encodings, face_encoding, tolerance=0.6)
```

Figura 17. Funcția care realizează algoritmul de recunoaștere facială.

Pe baza acestor locații, se extrag vectorii trăsăturilor faciale prin funcția `face_recognition.face_encodings(rgb_frame, face_locations)`, care aplică o rețea neuronală profundă pre-antrenată pentru a genera un vector de 128 dimensiuni ce descrie fiecare față. După aceea, se parcurge cu o bucla `for` fiecare vector de caracteristici împreună cu pozițiile corespunzătoare în imagine, pentru a permite ca fiecare față detectată să fie procesată împreună cu locația sa geometrică, ca mai apoi să fie desenat chenarul de imagine și să se afișeze rezultatul recunoașterii. În cadrul buclei se folosește o funcție care compară vectorul extras cu toți vectorii din baza de date, folosind distanța Euclidiană și returnează o listă de valori booleene, `true` sau `false`, care indică dacă există sau nu vreo potrivire în funcție de un prag de toleranță, setat 0.6.

În final, se extrag coordonatele feței, `y1`, `x2`, `y2`, `x1`, pentru a desena un chenar de formă dreptunghiulară, folosind funcția `cv2.rectangle()`, de culoare verde dacă persoana a fost recunoscută în baza de date, pentru a oferi un răspuns vizual. De asemenea se extrage numele persoanei, folosind funcția `cv2.putText()`, plasat sub chenar, tot de culoare verde dacă aceasta este recunoscută. Dacă persoana nu este recunoscută în baza de date, culoarea chenarului din jurul feței o să aibă culoarea roșie, iar numele, „Unknown”, tot culoarea roșie. Aceste funcții permit operatorului sistemului să observe identificarea în timp real.

### 5.2.2 Implementarea algoritmului de detecție a obiectelor

Pentru realizarea detecției obiectelor se implementează algoritmul YOLOv3, versiunea a treia a algoritmului YOLO. Este un algoritm capabil să identifice obiecte în imagini și secvențe video. Acesta se bazează pe caracteristici de învățare pe o rețea neuronală conoluțională profundă.

Versiunea a treia a algoritmului aduce îmbunătățiri semnificative în ceea ce privește viteza de execuție și precizia detecției comparative cu versiunile anterioare YOLOv1 și YOLOv2. Utilizează o rețea neuronală numită Darknet-53 ca bază pentru extragerea trăsăturilor. Această rețea conține 53 de layer convoluționale. Datorită acestui număr mare de straturi convoluționale, rețeaua este capabilă să învețe automat informații vizuale detaliate, fără a fi nevoie de extragerea trăsăturilor manual. Prin urmare, modelul este capabil să recunoască obiecte chiar și în condiții de iluminare slabă, suprapunerii sau fundaluri variabile. Un element important introdus în YOLOv3 este folosirea cutiilor de ancorare cu diferite dimensiuni și proporții, care permit algoritmului să recunoască obiecte de forme și dimensiuni variate. De asemenea, include o structură de tip Feature Pyramid Network (FPN), care permite atât detecția obiectelor mari, cât și a obiectelor mici, într-o singură imagine.

În implementarea practică a algoritmului YOLOv3, sunt necesare trei fișiere esențiale: `yolov3.cfg`, `yolov3.weights` și `coco.names`. Fișierul `yolov3.cfg` este fișierul de configurare care definește în detaliu arhitectura rețelei YOLOv3. Acesta conține informații despre ordinea și tipul straturilor utilizate, inclusiv cele convoluționale, de pooling și reziduale. De asemenea sunt specificați parametrii precum dimensiunea matricelor kernel, funcțiile de activare, numărul de filtre și alte setări pentru fiecare strat. Acest fișier funcționează ca o schiță completă a structurii interne a modelului. Fișierul `yolov3.weights` conține valorile numerice, cu parametri antrenați, învățate de rețeaua YOLOv3 în timpul procesului de antrenare pe un set de date, cum sunt cele COCO. Acești parametri antrenați determină modul în care rețeaua interpretează informațiile vizuale și recunoaște obiecte în imagine. La rularea aplicației, parametrii sunt încărcați automat în model și permit utilizarea rețelei fără a mai fi necesar un proces de antrenare suplimentar. Fișierul `coco.names` este de tip text care conține lista completă a claselor de obiecte pe care modelul YOLOv3 le poate recunoaște. Modelul antrenat pe setul de date COCO are un fișier care conține 80 de clase, cum sunt: person, bottle, dog, knife etc. Aceste denumiri sunt utilizate pentru a afișa rezultatele peste imaginea procesată.

```

# initializare YOLOv3
yolo_weights = "./YOLO/yolov3.weights"
yolo_config = "./YOLO/yolov3.cfg"
yolo_names = "./YOLO/coco.names"
net = cv2.dnn.readNet(yolo_weights, yolo_config)
with open(yolo_names, "r") as f:
    classes = [line.strip() for line in f.readlines()]
out_layers = [net.getLayerNames()[i - 1] for i in net.getUnconnectedOutLayers().flatten()]
colors = np.random.uniform(0, 255, size=(len(classes), 3))

```

Figura 18. Inițializarea algoritmului YOLOv3 în aplicație.

În aplicație se încarcă cele trei fișiere pentru inițializarea modelului, apoi se creează un obiect net prin funcția cv2.dnn.readNet(), care încarcă modelul de rețea neuronală YOLOv3, mai exact a structurii rețelei din fișierul yolov3.cfg și a parametrilor pre-antrenați din fișierul yolov3.weights. Se citește fișierul coco.names linie cu linie și se salvează numele claselor într-o listă numită classes, unde fiecare element corespunde unei clase de obiecte. Pentru a identifica layerele de ieșire ale rețelei, se extrage lista completă a layerelor, iar din aceasta se selectează indexurile acestora care contribuie direct la predicția finală, adică cele care nu mai au alt layer după ele, deoarece nu toate sunt relevante. Pentru asta se folosește funcția net.getUnconnectedOutLayers().

```

def detect_objects(frame):
    height, width = frame.shape[2]
    blob = cv2.dnn.blobFromImage(frame, 1/255.0, (416, 416), swapRB=True, crop=False)
    net.setInput(blob)
    outputs = net.forward(out_layers)

    boxes, confidences, class_ids = [], [], []
    for output in outputs:
        for detection in output:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > 0.5:
                cx, cy, w, h = (detection[0:4] * [width, height, width, height]).astype(int)
                x, y = int(cx - w/2), int(cy - h/2)
                boxes.append([x, y, int(w), int(h)])
                confidences.append(float(confidence))
                class_ids.append(class_id)

```

Figura 19. Funcția care realizează algoritmul de detecție a obiectelor.

Odată încărcată structura rețelei YOLOv3 și totodată au fost identificate layerele de ieșire responsabile pentru generarea predicțiilor de detecție ale obiectelor, se definește funcția detect\_objects(frame), unde se vor utiliza aceste informații pentru a procesa fiecare cadru. Se extrag dimensiunile imaginii, *height* și *width*, iar apoi aceasta este transformată într-un *blob* utilizând funcția cv2.dnn.blobFromImage(). Un *blob* este o formă de reprezentare a unei imagini necesară pentru a fi folosită ca intrare într-o rețea neuronală. În acest caz, funcția folosită normează valorile pixelilor în intervalul [0.0, 1.0], redimensionează imagini de dimensiune 416×416, convertește canalul de culoare din BGR în RGB și menține imaginea fără decupare. Acest blob este setat ca intrare pentru rețea, iar detecțiile sunt obținute apelând funcția net.forward(out\_layers), unde out\_layers reprezintă layerele de ieșire ale rețelei, configurate anterior. Pentru fiecare ieșire generată se extrag vectorii de detecție, unde primii patru reprezintă centru x, centru y, lățime și înălțime, iar restul sunt scorurile de probabilitate pentru fiecare clasă. Se calculează clasa cu cea mai mare probabilitate folosind funcția np.argmax(), iar detecția este păstrată doar dacă încrederea depășește pragul de 0.5. Fiecare detecție returnează coordonatele centrului obiectului, împreună cu lățimea și înălțimea chenarului de detecție, exprimate în format centrat. Pentru a putea reprezenta grafic aceste obiecte pe imagine, coordonatele sunt transformate în format de tip colț stânga-sus prin calcularea poziției: (x,y)=(cx-w/2, cy-h/2). Astfel se obține punctul de plecare pentru desenarea dreptunghiului de delimitare. Ulterior aceste coordonate, împreună cu valoarea scorului de încredere și indexul clasei prezise sunt stocate în liste intermediare. Pentru a elimina detecțiile multiple care se suprapun același obiect, se aplică funcția cv2.dnn.NMSBoxes() care implementează algoritmul de suprimare a maximelor non-corelate (Non-Maximum Supression). Această metodă analizează toate cutiile de delimitare și reține doar acele detecții care au cel mai

mare scor de încredere. Astfel se asigură că pentru fiecare obiect este afișată o singură detecție relevantă. Fiecare detecție relevantă este desenată pe cadru sub forma unui chenar dreptunghiular, împreună cu numele clasei și scorul de încredere. În funcție de tipul obiectului, culoarea chenarelor și a numerelor poate să difere. Astfel că pentru obiecte periculoase, cum sunt cuțitele, culoarea chenarului este roșie, iar pentru restul obiectelor s-a ales culoarea albastră, oferind o semnalizare vizuală operatorului sistemului.

### 5.3 Implementarea bazei de date

Pentru implementarea și gestionarea bazei de date utilizate în sistem se folosește pachetul software XAMPP. Acesta oferă un mediu de dezvoltare local complet, care permite rularea unui server MySQL și accesarea acestuia printr-o interfață web, phpMyAdmin.

XAMPP este o multiplatformă care integrează mai multe servicii pentru dezvoltarea de aplicații web. Dintre acestea, în cadrul proiectului se utilizează doar două componente esențiale: Apache și MySQL.

Apache este un server HTTP, folosit în etapa de configurare pentru accesarea interfeței web phpMyAdmin, care rulează în browser la adresa <http://localhost/phpmyadmin>. Această aplicație PHP oferă o interfață grafică pentru interacțiuni cu serverul MySQL.

MySQL este sistemul de gestiune a bazei de date utilizată în proiect. Reprezintă un sistem care permite organizarea și manipularea datelor în tabele relaționale. MySQL a fost ales pentru compatibilitatea cu PHP și pentru că este deja inclus în pachetul XAMPP.

Astfel, pentru a crea și administra baza de date este necesară pornirea simultană a serviciilor Apache și MySQL din XAMPP Control Panel.

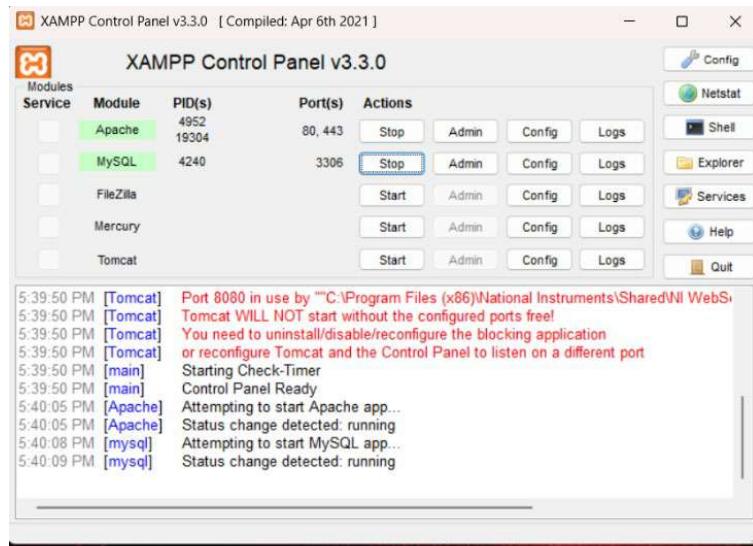


Figura 20. Interfața aplicației XAMPP.

După accesarea phpMyAdmin, se selectează opțiunea de creare a unei noi baze de date din bara laterală. Se introduce numele bazei de date, iar apoi se confirmă crearea, cu setări implicate, „collation:utf8\_general\_ci”.

Baza de date „licenta” are o structură simplă, adaptată cerințelor sistemului. Conține două tabele: users și logs.

Tabelul users este utilizat pentru stocarea utilizatorilor cunoscuți, împreună cu encodările faciale, care reprezintă vectori numerică extrași din imagini. Structura tabelului este compusă din câmpurile: *id*, *name*, *encoding*. Câmpul *id* este de tip întreg și este configurat ca auto-increment, adică valorile acestuia se incrementează automat la fiecare înregistrare nouă. Acest câmp asigură identificarea unică fiecărui utilizator stocat în tabel. Câmpul *name* este de tip varchar și nu permite

valori nule. Acesta conține numele utilizatorului introdus de administrator în momentul adăugării persoanei în sistem. Câmpul *encoding* este de tip longblob, nu permite valori nule și stochează codarea facială a utilizatorului sub formă de date binare.

Tabelul logs este utilizat pentru a păstra o evidență cronologică a fiecărei detecții realizate de aplicație. Structura tabelului este formată din câmpurile: *id*, *name*, *timestamp*. Câmpul *id* este de tip întreg și reprezintă cheia primară a tabelului, configurat ca auto-increment. Aceasta asigură identificarea unică a fiecărei înregistrări. Câmpul *name* este de tip varchar și nu permite valori nule. Acest câmp reține numele persoanei detectate de sistem, iar în cazul în care persoana nu se găsește în baza de date, se inserează automat valoarea „Unknown”. Câmpul *timestamp* este de tip datetime, nu permite valori nule și are valoarea implicită la *current\_timestamp()*. Acest câmp salvează momentul exact când o persoană este identificată.

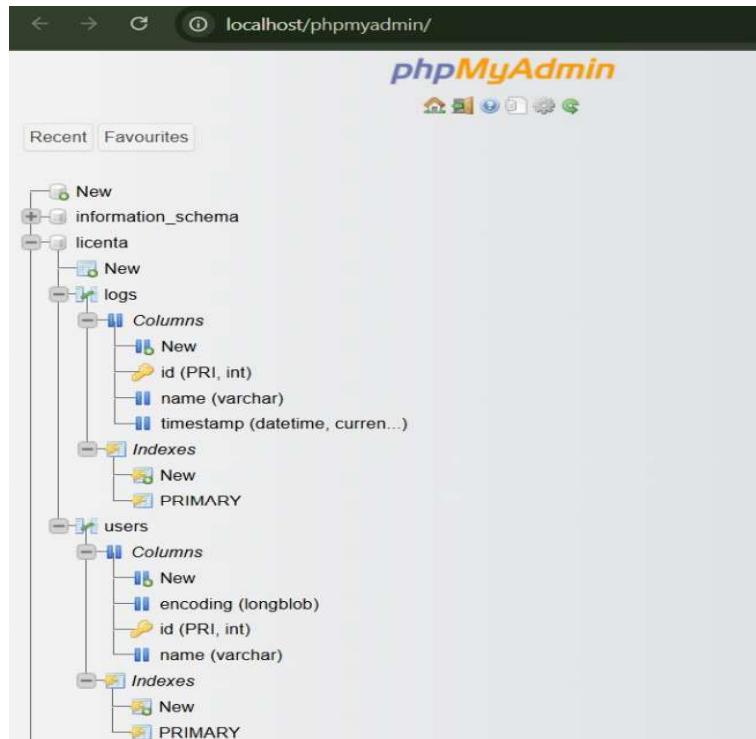


Figura 21. Structura bazei de date.

După crearea bazei de date, aplicația din Python se conectează la aceasta pentru a putea utiliza în cadrul procesului de recunoaștere facială. Conectarea la baza de date se realizează prin intermediul bibliotecii *pymysql*, care permite interacțiunea cu servere MySQL.

```
# baza de date MySQL
db = pymysql.connect(host="localhost", user="root", password="", database="licenta")

#funcție de încarcare a encodărilor faciale
def load_face_encodings():
    cursor = db.cursor()
    cursor.execute("SELECT name, encoding FROM users")
    rows = cursor.fetchall()
    encodings, names = [], []
    for name, encoding_blob in rows:
        encoding = np.frombuffer(encoding_blob, dtype=np.float64)
        encodings.append(encoding)
        names.append(name)
    return encodings, names
```

Figura 22. Liniile de cod care realizează conexiunea dintre aplicația Python și baza de date.

În cadrul aplicației, conexiunea se stabilește prin specificarea parametrilor de acces: adresa serverului, utilizatorul MySQL, parola și numele bazei de date. Astfel, în momentul în care aplicația pornește, se execută automat o funcție care încarcă din tabela users toate înregistrările existente, care sunt formate dintr-un nume și o encodare facială. Encodările sunt extrase sub formă binară din baza de date și transformate în vectori numerici, ca mai apoi să fie utilizați în comparare de către algoritmul de recunoaștere facială.

## 5.4 Periferice hardware integrate în sistem

### 5.4.1 Modulul buzzer

Pentru a integra un efecte sonore în sistem în funcție de rezultatul procesului de recunoaștere facială sau de detecție de obiecte, a fost integrat un modul MH-FMD, echipat cu un buzzer piezoelectric și un mic circuit de comandă.



Figura 23. Modulul buzzer MH-FMD.

Un buzzer pasiv este un dispozitiv, care generează sunete atunci când este aplicat un semnal electric. Pentru a genera sunete necesită un semnal extern de la un microcontroler, deoarece nu are un oscilator extern, de aici denumirea de pasiv. Acesta poate produce tonuri între 1,5kHz și 2,5kHz, iar pentru a fi pornit sau opus la frecvențe diferite sunt necesare fie întârzieri, fie semnale PWM. Buzzerul piezoelectric funcționează pe baza efectului piezoelectric. Aceasta conține un element realizat de obicei dintr-un material ceramic sau cristalin, care își modifică forma atunci când este aplicat un curent electric. Această deformare mecanică produce unde sonore.

Acest modul se conectează la placa ESP32-CAM prin intermediul a trei pini:

- VCC – conectat la alimentarea de 5V
- GND – conectat la masă
- I/O – legat la pinul GPIO14 de pe placă, utilizat pentru controlul semnalului.

Codul se implementează în Arduino IDE, în cadrul același sketch în care a fost inițializată placa ESP32-CAM. În codul sursă, buzzerul se controlă cu ajutorul funcției tone(pin, frecvență, durată). În implementare, sunt generate tonuri diferite în funcție de context. Atunci când se recunoaște o persoană din baza de date, se produce un sunet continu de 1000Hz, implementat în funcția handleBuzzerKnown(). Când persoana detectată nu este identificată în baza de date sau este detectat un obiect periculos, se produce o succesiune de tonuri scurte la 500Hz, prin funcția handleBuzzerUnknown().

```
void handleBuzzerKnown() {
    Serial.println("Buzzer: Persoana cunoscută");
    tone(BUZZER_PIN, 1000, 1000);
```

Figura 24. Implementarea funcției care semnalizează sonor o persoană recunoscută.

```

void handleBuzzerUnknown() {
    Serial.println("Buzzer: Persoana necunoscuta");
    for (int i = 0; i < 3; ++i) {
        tone(BUZZER_PIN, 500, 200);
        delay(300);
    }
}

```

Figura 25. Implementarea funcției care semnalizează sonor o persoană necunoscută.

În cadrul funcției initServer() se definesc rutele HTTP, de tip „/buzzer/know” și „/buzzer/unknown” pe care serverul găzduit de ESP32-CAM le va recunoaște și gestiona. Aceste comenzi stabilesc faptul că atunci când ESP32-CAM primește o solicitare de tip GET către una din aceste adrese, va apela funcția corespunzătoare. Acest mecanism permite interacțiunea în timp real între aplicația Python, care face recunoașterea facială și detecția obiectelor, și modulul hardware.

#### 5.4.2 Modulul senzor PIR

Modulul SEN0018 este un senzor de mișcare PIR, utilizat în acest sistem pentru a detecta prezența unei persoane în fața camerei și de a activa sistemul doar atunci când este necesar, în vederea optimizării consumului de energie. Acest senzor funcționează pe baza detecției radiației infraroșii pasive, captând variații emise de corpuși calde aflate în mișcare. Elementul sensibil din senzor este un detector cu două zone, acoperit cu o lentilă Fresnel, care concentrează radiația IR pe suprafața sa. Semnalul analogic creat de diferența de potențial dintre cele două zone este apoi amplificat și trecut prin un comparator de tip trigger Schmitt, rezultând un semnal logic digital stabil.

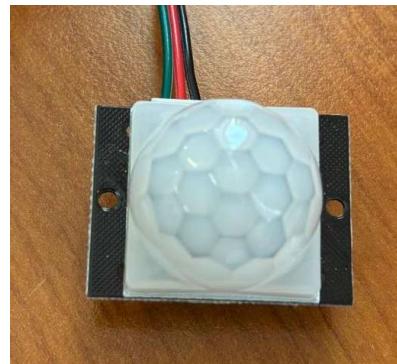


Figura 26. Modulul SEN0018

Modulul operează la o tensiune de alimentare între 3V-5V, are un consum de curent de  $50\mu\text{A}$  și funcționează într-un interval de temperatură cuprins între  $[0^\circ\text{C}, +70^\circ\text{C}]$ . Poate detecta mișcare pe o distanță de până la 7 metri, într-un unghi de aproximativ  $110^\circ$ . Dispune de trei fire: unul pentru ieșirea semnalului digital, unul pentru alimentare și unul pentru masă. De asemenea, prezintă un jumper care permite selectarea funcționării în două moduri: H, retriggerable și L, non-retriggerable. În modul H semnalul de ieșire este menținut activ și se reînnoiește la fiecare detecție succesivă de mișcare, iar pentru modul L senzorul trimite un singur impuls pentru fiecare ciclu de detecție, indiferent dacă mișcarea continuă sau nu. În plus durata semnalului de ieșire poate fi ajustată prin intermediul unui potențiometru aflat pe modul, în funcție de aplicație.

În implementarea practică, firul de alimentare al senzorului este conectat la 5V, masa la GND, iar pinul digital este conectat la GPIO13 al placii ESP32-CAM, configurat ca intrare și se selectează jumperul pe modul H.

Partea de programare a senzorului se implementează în Arduino IDE. În codul sursă, în cadrul buclei loop(), senzorul este monitorizat constant, iar în absența activității pentru mai mult de 60 de secunde, dispozitivul intră automat în stare de *deep sleep*. Starea de *deep sleep* asigură un

consum semnificativ de energie, de doar 6mA. În cadrul funcției `setup()`, se activează funcționalitatea de wake-up prin întrerupere externă, ceea ce permite revenirea din starea *deep sleep* când se detectează mișcare. La reactivare, dispozitivul reinițializează componentele, asigurând continuitatea funcționalității.

```
if (millis() - lastMotion > activeDuration) {
    Serial.println("Inactivitate detectata. Intru in deep sleep");
    esp_deep_sleep_start();
}
```

Figura 27. Liniile de cod implementate pentru modul *deep sleep*.

#### 5.4.3 Încuietoarea electromagnetică controlată de releu

Pentru asigurarea unui control fizic doar utilizatorilor autorizați, sistemul integrează o încuietoare electromagnetică. Această încuietoare funcționează pe baza unui mecanism electromagnetic. În lipsa tensiunii de alimentare, pistonul rămâne extins și blochează mecanic accesul, iar când se aplică tensiune, mai exact 12V, pistonul este retras în interior, ceea ce duce la deschiderea ușii. Încuietoarea este prevăzute cu două fire, unul pentru tensiunea pozitivă de 12V și un fir de masă.



Figura 28. Încuietoare electromagnetică

Comanda încuietorii se face indirect, fiind necesară folosirea unui circuit de comutare conectat între sursa de 12V și încuietoare. Pentru aceasta se folosește un modul releu cu optocupluri, cu rol în izolare electrică între circuitul de control și cel de putere.

Releul poate fi configurat atât pe partea de intrare, cât și pe partea de ieșire, în funcție de aplicația dorită. Astfel că pe pinul de intrare, releul poate fi declanșat fie la un nivel logic scăzut, fie la un nivel logic ridicat. În ceea ce privește ieșirea, releul oferă două opțiuni de conexiune, contact normal deschis (NO) și contact normal închis (NC). În modul de contact normal deschis circuitul se închide doar când releul este activ, iar în modul de contact închis circuitul este închis și se deschide doar în timpul activării releului.



Figura 29. Modul releu

Pentru implementarea practică se conectează pinul de alimentare pozitivă al modulului de releu la 5V, iar cel de alimentare negativă la masă. Semnalul de comandă este generat pe pinul GPIO15 al plăcii ESP32-CAM, fiind conectat la pinul de intrare al modulului de releu. Pe partea de putere, pinul COM al modulului se conectează la sursa de 12V, iar ieșirea NO asigură alimentarea pozitivă a încuietorii. Firul de masă al încuietorii se leagă la borna negativă a sursei de 12V. Modulul a fost configurat să funcționeze în mod High Level Trigger.

Codul sursă care controlează deschiderea încuietorii se implementează în Arduino IDE. Funcția triggerUnlock() activează releul, trimițând un semnal logic pe nivel high care permite alimentarea cu 12V a încuietorii. În acel moment se pornește un timer care controlează durata în care rămâne deschisă încuietoarea. După zece secunde, cu ajutorul funcției handleAutoLock(), apelată în loop, pinul este setat pe nivel low, ceea ce dezactivează releul și închide încuietoarea.

În cadrul funcției initServer() se definește ruta HTTP „/lock/unlock”, asociată funcției handleUnlock(). Atunci când serverul primește o solicitare de tip GET către una din aceste adrese, va apela funcția și va determina deschiderea încuietorii, semn că persoana recunoscută se află în baza de date.

```
void triggerUnlock() {
    Serial.println(">> Încuietoare deschisa");
    digitalWrite(LOCK_RELAY_PIN, HIGH);
    lockIsOpen = true;
    lockOpenTime = millis();
}

void handleAutoLock() {
    if (lockIsOpen && millis() - lockOpenTime >= lockOpenDuration) {
        digitalWrite(LOCK_RELAY_PIN, LOW);
        lockIsOpen = false;
        Serial.println(">> Încuietoare inchisa");
    }
}
```

Figura 30. Funcțiile care permit deschiderea și închiderea încuietorii.

#### 5.4.4 Afișaj LCD 2004

Pentru afișarea informațiilor textuale în cadrul sistemului, a fost utilizat un ecran LCD cu 4 linii și 20 de coloane, capabil să afișeze până la 80 de caractere simultan. Acesta este construit pe baza controlerului clasic HD44780 și permite afișarea de caractere alfanumerice prin semnale digitale.



Figura 31. LCD 2004

În forma sa clasică, afișajul necesită conectarea a cel puțin șase pini pentru control și alți doi pentru alimentare, însă această complexitate a fost eliminată prin utilizarea unui modul adaptor I2C.

Adaptorul I2C, montat pe spatele ecranului LCD, conține un circuit integrat PCF8574 care transformă comenziile seriale transmise prin două fire, SDA și SCL, în semnale paralele compatibile cu controlerul LCD-ului. De asemenea, modulul dispune de un potențiometru pentru reglarea contrastului caracterelor afișate. Afișajul LCD se conectează la placa ESP32-CAM prin modulul I2C, utilizând pinul GPIO4 pentru linia de date, SDA și pinul GPIO2 pentru ceas, SCL. Modulul este alimentat la 5V, cu conexiunea de masă potrivită.

Implementarea software a LCD-ului se realizează în mediul Arduino IDE. Integrarea afișajului în codul proiectului se realizează cu ajutorul bibliotecii LiquidCrystal\_I2C, care oferă metode pentru inițializarea ecranului, setarea poziției corsoului și scrierea de text. Pentru comunicarea prin magistrala I2C dintre placa ESP32-CAM și afișajul LCD, a fost inclusă biblioteca „Wire.h”, responsabilă de gestionarea transmisiei de date pe liniile SDA și SCL.

În codul sursă se instantiază un obiect lcd(0x27, 20, 4), unde 0x27 reprezintă adresa I2C a modulului de afișaj, iar valorile 20 și 4 indică numărul de coloane și rânduri ale ecranului. Pentru gestionarea eficientă a informației afișate a fost implementată o funcție safeLcdPrint(), care actualizează textul de pe ecran atunci când conținutul afișat se modifică pentru a evita rescrieri inutile. Scrierea textului se face prin poziționarea cursorului, cu ajutorul funcției lcd.setCursor(col, row), unde col reprezintă coloana, iar row este linia dorită. De asemenea, se definește funcția displayDefaultMessage(), utilizată pentru afișarea unui mesaj implicit atunci când nu sunt detectate evenimente.

În cadrul funcției setup(), inițializarea comunicației I2C se realizează prin comanda Wire.begin(4,2), specificând pinii SDA și SCL pe placa ESP32-CAM. Ulterior, prin apelul funcției lcd.init() afișajul este configurat și pregătit pentru utilizare.

Astfel, afișajul LCD este folosit pentru a furniza utilizatorului mesaje de stare relevante, cum ar fi confirmarea accesului în cazul recunoașterii faciale sau refuzul în cazul unei persoane necunoscute.

## 5.5 Circuitul final al sistemului

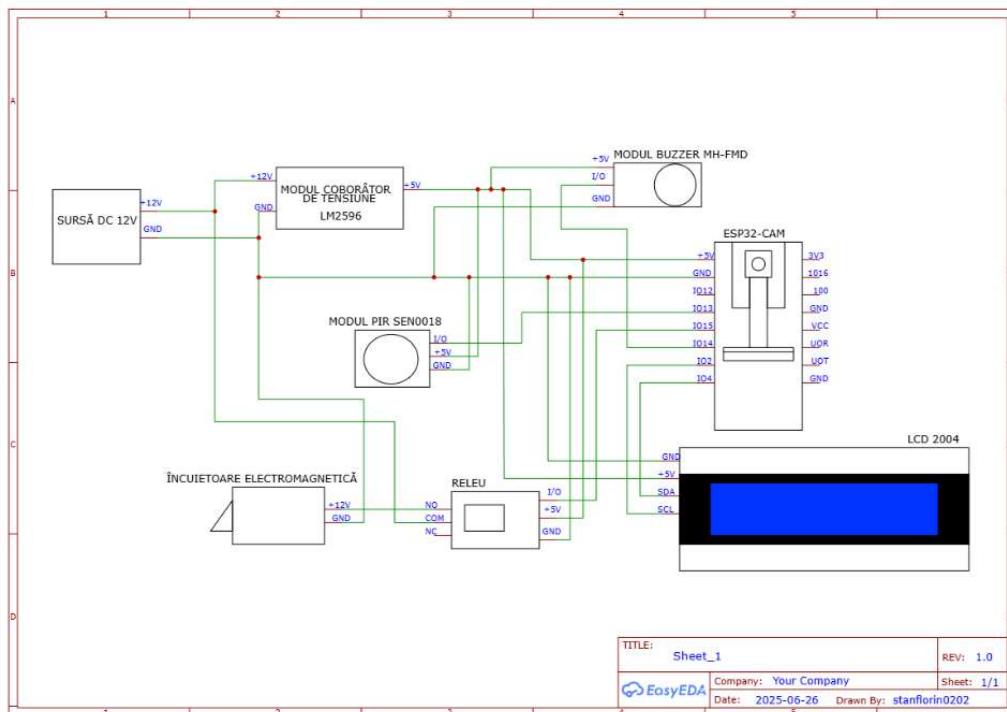


Figura 32. Schema electrică a circuitului sistemului,

Pentru alimentarea întregului circuit se utilizează o sursă de tensiune externă de 12V DC, necesară acționării încuietorii electromagnetice. Restul componentelor din sistem funcționează la o tensiune de alimentare de 5V, motiv pentru care a fost necesară utilizarea unui modul coborâtor de tensiune.

Modulul coborâtor de tensiune LM2596 este un convertor DC-DC de tip buck, capabil să reducă tensiunea de între mai mare la o tensiune mai mică, stabilă, ajustabilă. Acest modul este eficient și stabil, oferă curent de până la 3A, ideal pentru alimentarea mai multor componente de joasă tensiune dintr-un sistem. Tensiunea de ieșire se ajustează cu ajutorul unui potențiometru.

## 5.6 Interfața web

Pentru a facilita interacțiunea utilizatorului cu aplicația, am dezvoltat o interfață web folosind framework-ul Flask. Acest framework permite definirea unor rute corespunzătoare fiecărei pagini sau acțiuni, iar pentru transmiterea datelor între browser și server se folosesc metodele HTTP standard, GET și POST.

Ruta principală, care este asociată cu pagina de autentificare este definită: `@app.route("/", methods=["GET", "POST"])`. Când utilizatorul accesează pagina pentru prima dată, serverul returnează fișierul HTML login.html care conține formularul de autentificare. După autentificare, browserul trimite datele către server folosind metoda POST. Aceste date sunt apoi preluate din `request.form`, care verifică dacă numele de utilizator și parola sunt corecte. Dacă acestea sunt corecte se setează o variabilă de sesiune `session["logged_in"]` pentru a indica faptul că utilizatorul este conectat și redirecționat către ruta `"/dashboard"`. Dacă datele sunt greșite se returnează aceeași pagină login.html.

```
@app.route("/")
def login():
    if request.method == "POST":
        if request.form["username"] == "florinstan" and request.form["password"] == "parola123":
            session["logged_in"] = True
            return redirect("/dashboard")
        else:
            return render_template("Login.html", error="Date incorecte!")
    return render_template("login.html")
```

Figura 33. Implementarea rutei pentru pagina de logare.

Ruta `@app.route("/dashboard")` este activată imediat după autentificare și reprezintă interfața principală a aplicației. În interiorul funcției se verifică dacă utilizatorul este conectat, iar dacă nu, este redirecționat către pagina login.

```
@app.route("/dashboard")
def dashboard():
    if not session.get("logged_in"): return redirect("/")
    return render_template("dashboard.html")
```

Figura 34. Implementarea rutei pentru pagina dashboard.

HTML-ul acestei pagini include linkuri către celelalte secțiuni, conform Figurii 35.

```

<a href="#" onclick="openStream()">Vizualizați imaginile live</a>
<br><br>
<a href="/add_user">Adăugați persoană nouă</a>
<br><br>
<a href="/users">Administrați utilizatorii</a>
<br><br>
<a href="/logs">Vizualizați istoricul accesărilor</a>
<br><br>
<a href="/logout">Ieșire</a>
<br><br>

```

Figura 35. Implementarea unor butoane către alte pagini.

Ruta `@app.route("/video")` este responsabilă pentru transmiterea în timp real a imaginilor video de la ESP32-CAM. Serverul returnează un răspuns de tip stream, în format mime. Conținutul este generat de funcția `stream_video()`, care citește câte un cadru la un moment dat, îl codifică în JPEG și il transmite către client.

```

@app.route("/video")
def video():
    if not session.get("Logged_in"):
        return redirect("/")
    return Response(stream_video(), mimetype="multipart/x-mixed-replace; boundary=frame")

def stream_video():
    while True:
        frame = frame_queue.get()
        _, jpeg = cv2.imencode('.jpg', frame)
        yield (b'--frame\r\nContent-Type: image/jpeg\r\n\r\n' + jpeg.tobytes() + b'\r\n')

```

Figura 36. Implementarea rutei de vizualizare stream live.

În `dashboard.html`, fluxul video este afișat într-un element `` plasat sub forma unei suprapuneri pe ecran, controlată de două funcții JavaScript.

Pentru adăugarea de noi utilizatori în baza de date am creat ruta `@app.route("/add_user")`, `methods=["GET", "POST"]`. Pentru metoda de tip GET, serverul returnează formularul HTML care permite introducerea numelui persoanei și selectarea imaginii. Metoda POST procesează datele introduce de server, imaginea este salvată, apoi analizată cu ajutorul bibliotecii `face_recognition` pentru extragerea vectorului de caracteristici. Dacă se detectează față, datele sunt stocate în baza de date.

```

@app.route("/add_user", methods=["GET", "POST"])
def add_user():
    if not session.get("Logged_in"):
        return redirect("/")
    if request.method == "POST":
        name = request.form["name"]
        file = request.files["file"]
        if file:
            img_path = f"faces/temp.jpg"
            file.save(img_path)
            image = face_recognition.load_image_file(img_path)
            face_encs = face_recognition.face_encodings(image)
            if len(face_encs) > 0:
                enc = face_encs[0].astype(np.float64).tobytes()
                cursor = db.cursor()
                cursor.execute("INSERT INTO users (name, encoding) VALUES (%s, %s)", (name, enc))
                db.commit()
                global known_encodings, known_names
                known_encodings, known_names = load_face_encodings()
                flash("Persoana adăugată cu succes!")
            else:
                flash("Nicio față detectată!")
    return render_template("add_user.html")

```

Figura 37. Implementarea rutei pentru adăugarea unor noi utilizatori în baza de date

Ruta `@app.route("/users")` permite extragerea utilizatorilor din baza de date și îi afișează într-un tabel HTML. Pentru fiecare utilizator există un buton care permite ștergerea acestuia din baza de date, care face trimitere către ruta `@app.route("/delete_user/<int:user_id>")`.

```
@app.route("/users")
def users():
    if not session.get("Logged_in"):
        return redirect("/")
    cursor = db.cursor()
    cursor.execute("SELECT id, name FROM users")
    users = cursor.fetchall()
    return render_template("users.html", users=users)
```

Figura 38. Implementarea rutei de vizualizare a utilizatorilor din baza de date.

Ruta `@app.route("/logs")` permite vizualizarea evenimentelor recente de recunoaștere facială, care realizează o interogare asupra tablei logs din baza de date și returnează o listă cu ultimele 50 de înregistrări, fiecare conținând numele persoanei recunoscute și timpul corespunzător.

```
@app.route("/Logs")
def logs():
    if not session.get("Logged_in"):
        return redirect("/")
    cursor = db.cursor()
    cursor.execute("SELECT name, timestamp FROM Logs ORDER BY timestamp DESC LIMIT 50")
    logs = cursor.fetchall()
    return render_template("logs.html", logs=logs)
```

Figura 39. Implementarea rutei care permite vizualizarea istoricului de accesări.

Ultima rută realizează închiderea sesiunii curente. Prin apelarea `@app.route("/logout")` se elimină toate datele din sesiunea utilizatorului prin comanda `session.clear()`, după care se face redirecționarea către pagina principală de login.

```
@app.route("/Logout")
def logout():
    session.clear()
    return redirect("/")
```

Figura 40. Implementarea rutei de închidere a sesiunii de logare.

## 6 Rezultate experimentale

Pentru implementarea practică a sistemului, circuitul electric final a fost realizat pe o placă de prototip, unde au fost montați pini pentru conectarea modulelor necesare funcționării sistemului. Conexiunile au fost realizate prin lipire direct pe traseele plăcii, asigurând contact electric corespunzător. În Figura 41.b se observă firele trase pe partea inferioară pentru legături între pini și module.

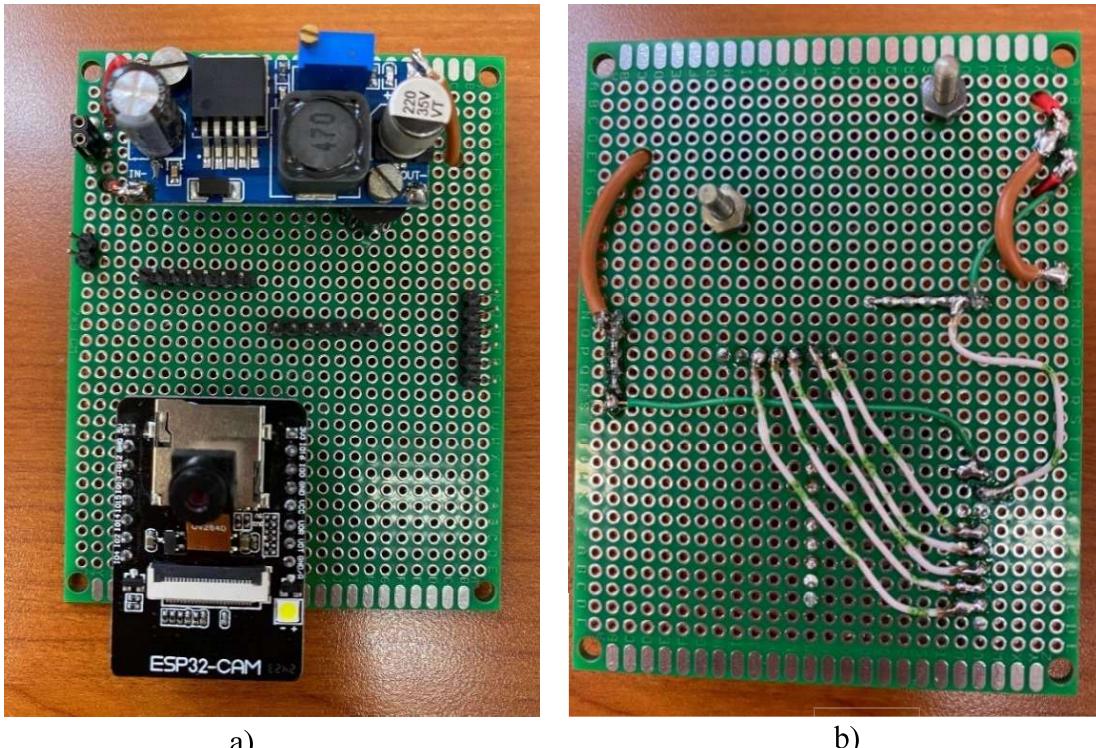


Figura 41. a) Partea superioară a plăcii. b) Partea inferioară a plăcii.

În Figura 42 se observă circuitul final încapsulat într-o carcăsă dedicată, care facilitează integrarea într-un sistem real. De asemenea, perifericele precum senzorul PIR, încuietoarea electromagnetică și LCD-ul au fost montate pe un panou de prezentare, împreună cu o ușă demonstrativă.

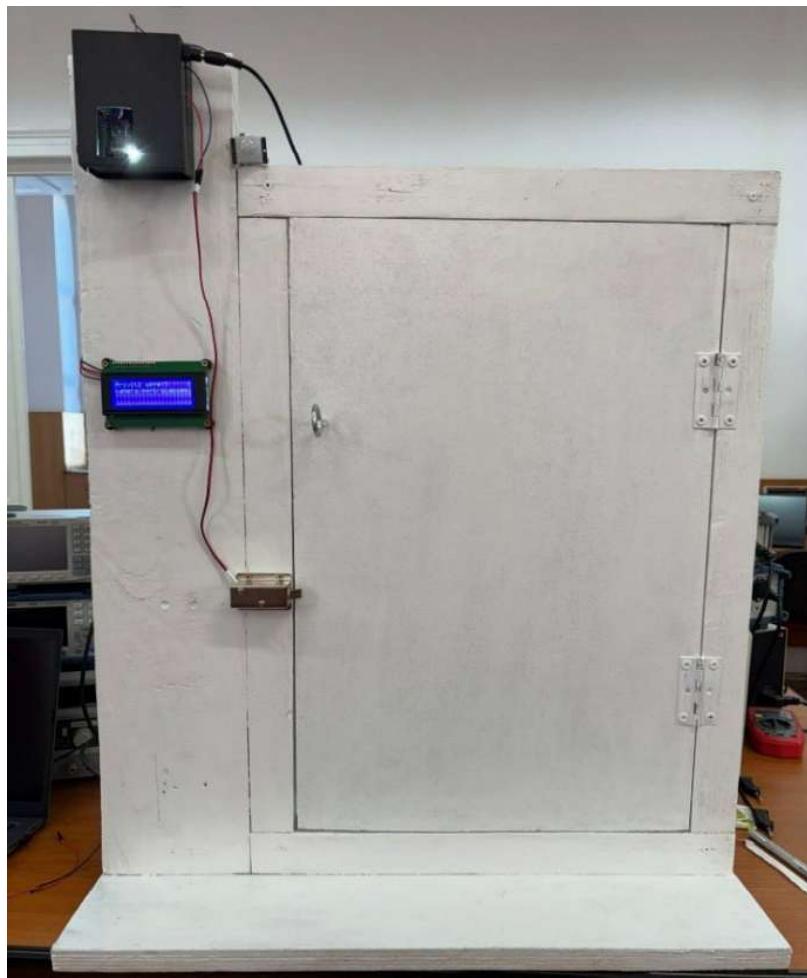


Figura 42. Montajul de prezentare al sistemului.

În momentul în care este rulată baza de date MySQL, împreună cu aplicația din Python, aplicația Flask pornește serverul web local. În consolă se afișează adresele IP pe care aplicația rulează și poate fi accesată, conform Figurii 43.

```
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://192.168.0.105:5000
Press CTRL+C to quit
```

Figura 43. Adresele IP pe care rulează aplicația.

Adresa 127.0.0.1:5000 permite utilizarea aplicației direct de pe calculatorul care rulează, iar adresa 127.168.0.105:5000 permite accesarea aplicației de pe alte dispozitive conectate la aceeași rețea locală. Astfel, interfața web devine disponibilă și pentru controlul de la distanță al sistemului, cu condiția ca dispozitivul să fie conectat la aceeași rețea locală.

În momentul în care utilizatorul accesează adresa IP a serverului, aplicația afișează pagina de autentificare, confirmând funcționarea corectă a serverului Flask și a interfeței web, conform Figurii 44.

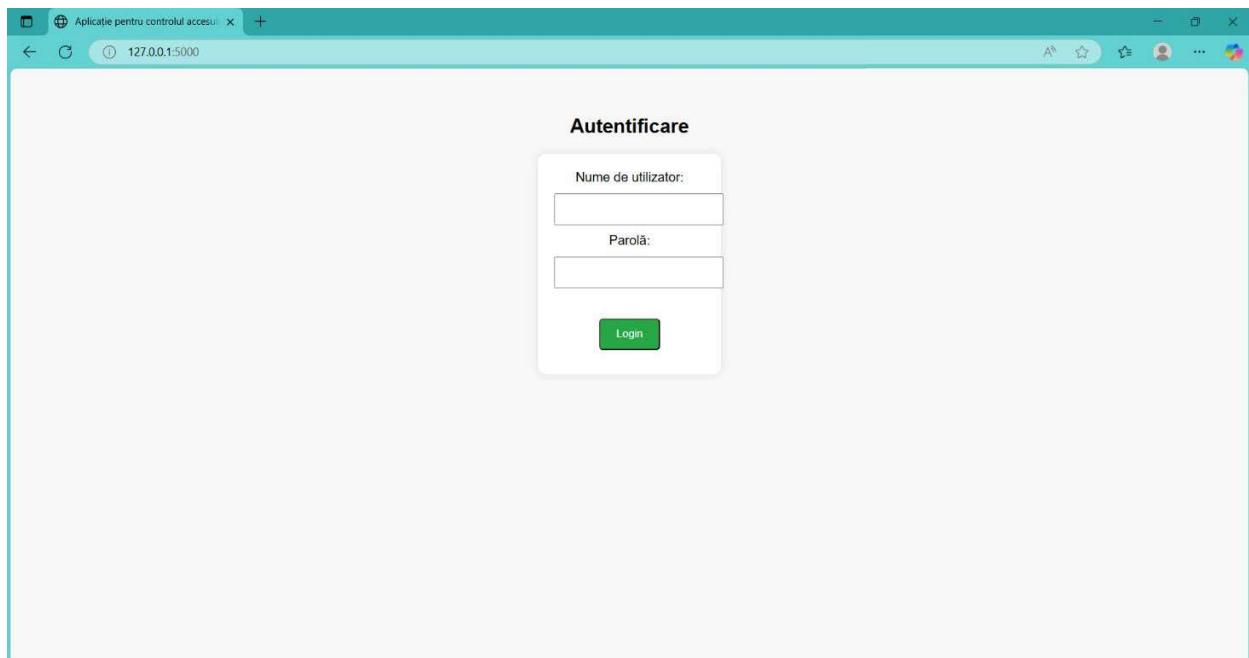


Figura 44. Pagina de autentificare.

După autentificarea cu succes a utilizatorului în aplicație, acesta este redirecționat către pagina principală unde este afișat un panou de control, conform Figurii 45. Acesta conține butoane care permit accesul rapid la diferite funcționalități ale aplicației, cum ar fi: vizualizarea imaginilor live transmise de ESP32-CAM, adăugarea de noi persoane în baza de date, administrarea utilizatorilor existenți, vizualizarea istoricului accesărilor sau ieșire pentru încheierea sesiunii curente de autentificare.

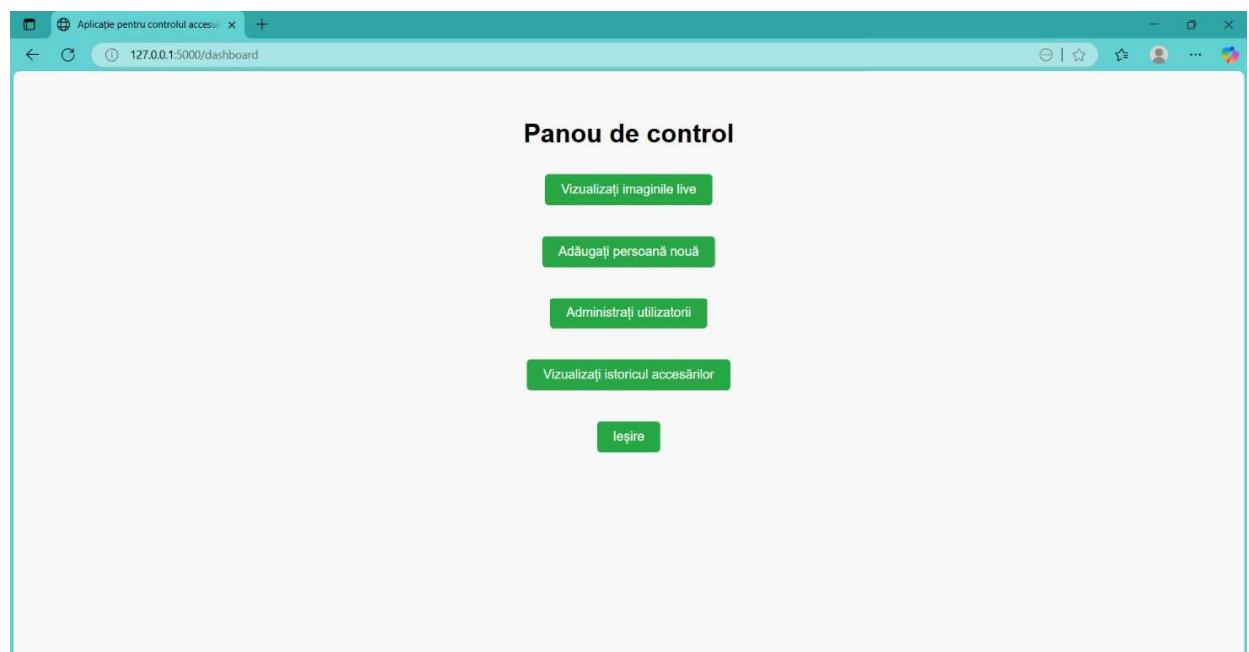


Figura 45. Pagina dashboard.

În Figura 46 este prezentată situația în care imaginea capturată de ESP32-CAM nu este disponibilă. Mesajul cu font de culoare roșie apare când modulul este în stare de *deep sleep*, conexiunea wireless este instabilă sau dispozitivul nu este alimentat corespunzător, informând utilizatorul despre imposibilitatea afișării imaginilor în timp real.

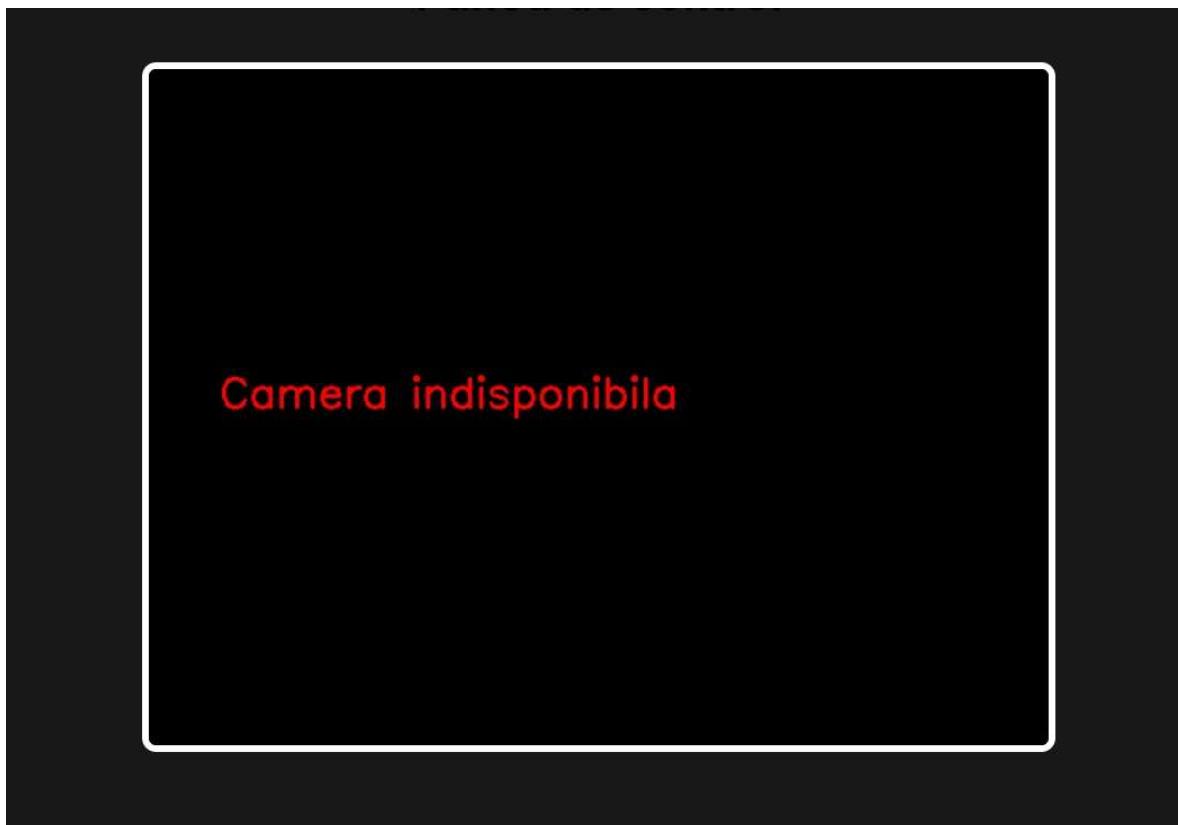


Figura 46. Mesajul care semnalizează că nu este disponibilă imaginea capturată.

În Figura 47 este prezentată forma de undă a semnalului generat de modul cu senzorul PIR, capturată cu ajutorul osciloscopului. Se observă că semnalul de ieșire rămâne constant la nivel logic *sus* în timpul detecției de mișcare, indicând prezența unei persoane. Tensiunea măsurată a fost de aproximativ 3.3V, care corespunde nivelului logic necesar pentru a putea fi interpretat corect de către pinul GPIO 13.



Figura 47. Forma de undă a semnalului generat de senzorul PIR.

Figura 48 ilustrează afişajul LCD al sistemului, pe care este vizualizat mesajul implicit, afişat în mod automat când senzorul PIR detectează mişcare, indicând că dispozitivul este activ și pregătit pentru capturarea imaginilor și procesarea necesară accesului.



Figura 48. Mesajul implicit afișat pe LCD.

În Figura 49 este prezentat rezultatul procesului de recunoaștere facială. Persoana detectată, încadrată într-un chenar albastru a fost identificată corect în baza de date, fiind afișat și numele acesteia (chenar verde) cu font de culoare verde.

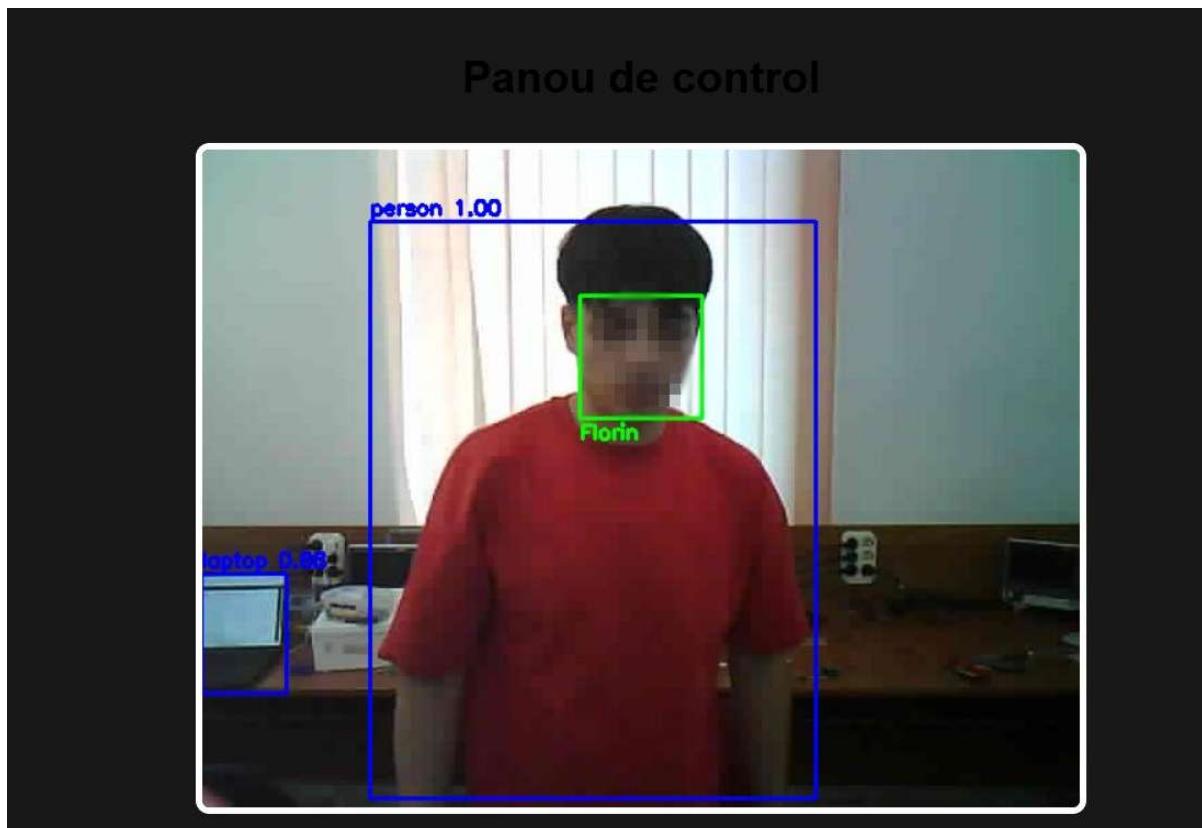


Figura 49. Rezultatul recunoașterii persoanei din baza de date.



Figura 50. Mesajul afișat pe LCD în momentul în care este recunoscută o persoană.

În momentul în care persoana identificată este recunoscută în baza de date, se va afișa un mesaj pe LCD, care confirmă vizual accesul permis, conform Figurii 50. De asemenea, în Figura 51 se evidențiază activarea încuietorii electromagnetice, care să permită deschiderea ușii pentru accesul persoanei autorizate.



Figura 51. Încuietoarea activată după recunoașterea persoanei.

În Figura 52 este ilustrat semnalul preluat de la ieșirea buzzerului în timpul funcționării, când este recunoscută o persoană în baza de date. Se observă un semnal dreptunghiular generat la o frecvență de 1000Hz, corespunzător comenzi implementate în codul sursă.

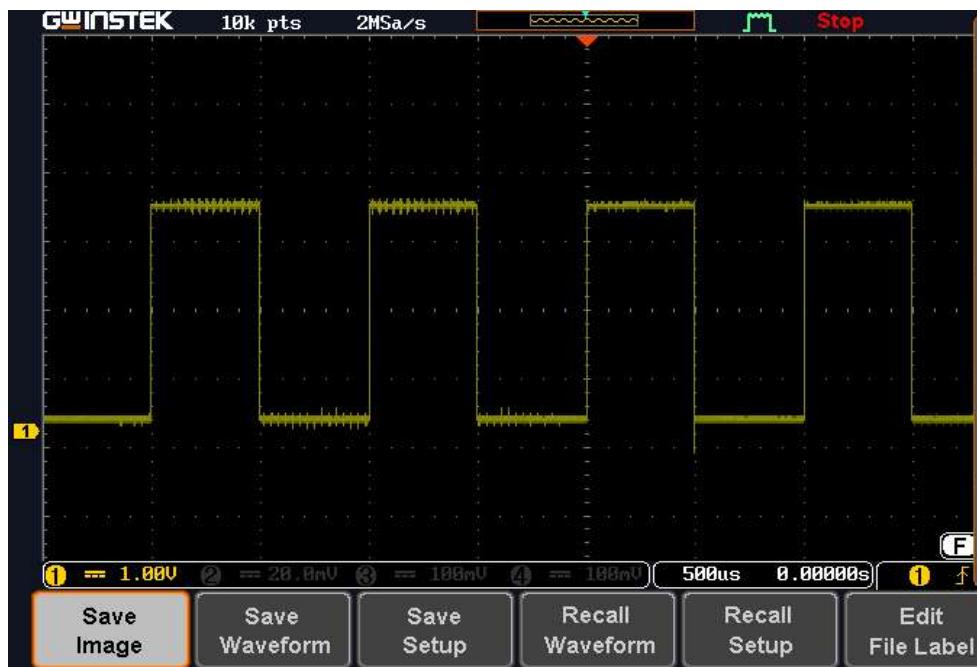


Figura 52. Forma de undă a buzzerului care semnalizează o persoană cunoscută.

În Figura 53 se observă că algoritmul de recunoaștere facială a detectat o persoană (chenar albastru) care nu este în baza de date și se afișează eticheta „Unknown” (chenar roșu) de culoare roșie. În momentul în care persoana identificată nu este recunoscută în baza de date, se va afișa un mesaj pe LCD, care confirmă vizual că accesul este respins, conform Figurii 54. Acest mesaj afișat este completat de alertă acustică emisă de buzzer, conform Figurii 55. Oscilograma ilustrează semnalul generat pe pinul buzzerului, unde se observă o succesiune de impulsuri dreptunghiulare corespunzătoare unui ton de 500Hz, în perioada de 200ms, cu pauze de 300ms între tonuri.

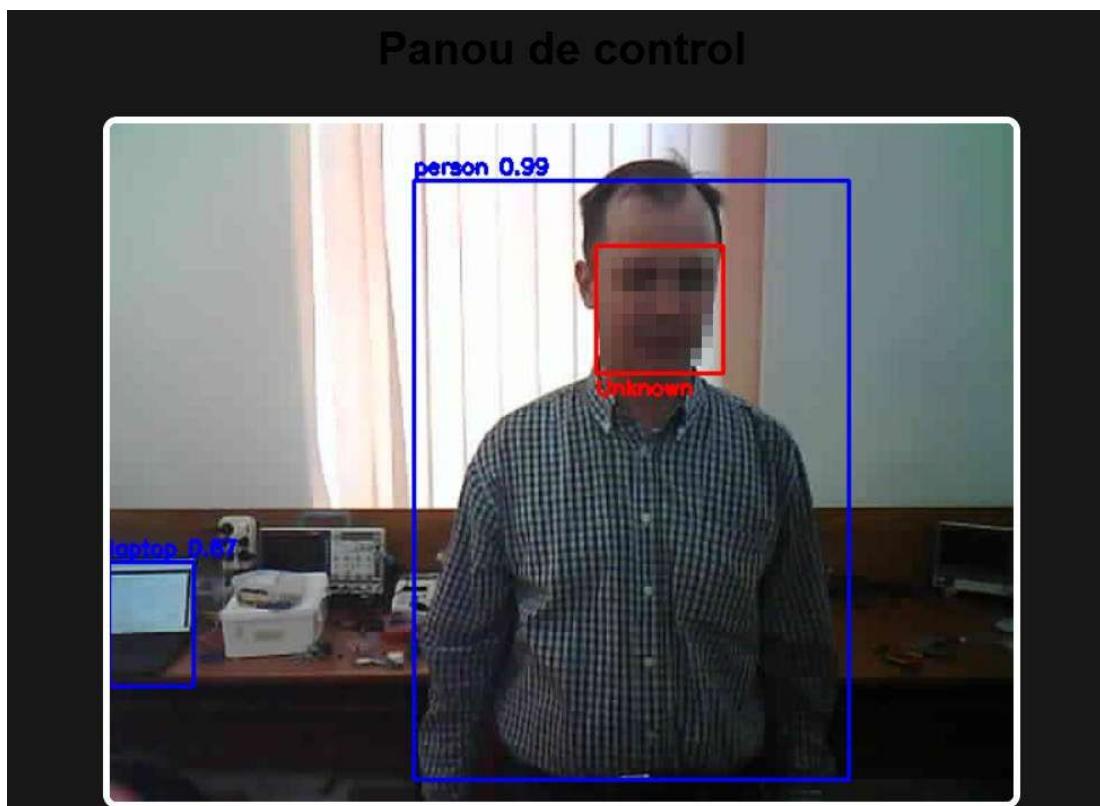


Figura 53. Rezultatul recunoașterii unei persoane care nu este în baza de date.



Figura 54. Mesajul afișat pe LCD în momentul în care nu este recunoscută o persoană.

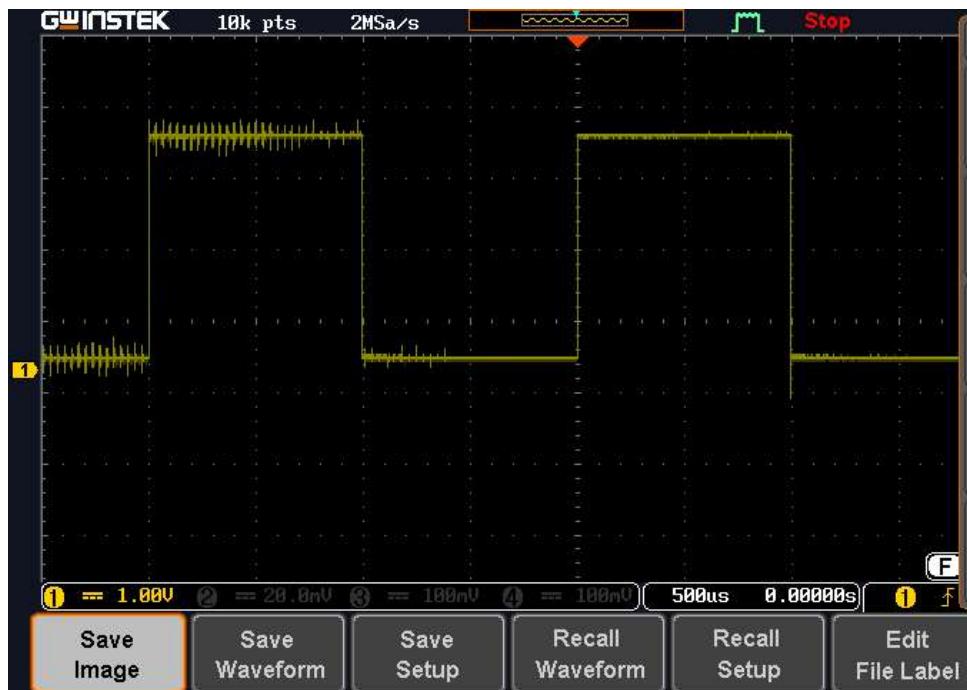


Figura 55. Forma de undă a buzzerului care semnalizează o persoană necunoscută sau un obiect periculos.

În Figura 56 este prezentat rezultatul procesului de detecție a obiectelor implementat în Python. Sistemul a identificat corect mai multe obiecte din imagine, cum ar fi un laptop, cu probabilitate de 99% și o sticlă (eng. bottle), cu o probabilitate de 100%. Detectarea multiplă demonstrează funcționalitatea algoritmului YOLOv3 de a identifica și clasa simultan mai multe obiecte în timp real.

În cazul detectiei unui obiect periculos, cum este cuțitul din Figura 57, acesta este încadrat într-un chenar, cu etichetă roșie pentru a evidenția nivelul de pericol, cu o probabilitate de 82%. De asemenea, sistemul activează semnalizarea sonoră produsă de buzzer, similară cu cea pentru persoane necunoscute.

## Panou de control

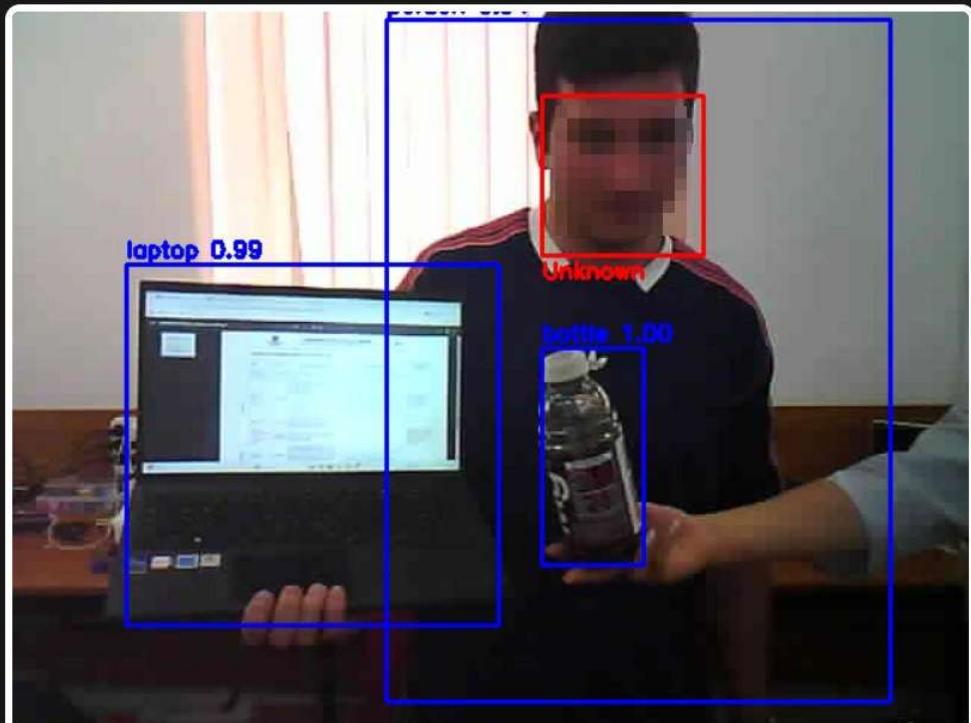


Figura 56. Rezultatul detecției obiectelor.

## Panou de control



Figura 57. Rezultatul detecției unui obiect periculos.

## Gestionarea bazei de date

În contextul detecției persoanelor și identificării multiplelor obiecte, următorul pas constă în gestionarea informațiilor despre persoanele detectate. Prin apăsarea butonului „Adăugați persoană nouă” din panoul de control, utilizatorul este redirecționat către o pagină dedicată introducerii de noi persoane în baza de date, după cum se observă în Figura 58. În această interfață se completează câmpul pentru nume și se selectează imaginea asociată pentru a se extrage setul de date utilizat în recunoașterea facială.

The screenshot shows a web browser window titled 'Aplicație pentru controlul accesului' with the URL '127.0.0.1:5000/add\_user'. The main content is a form titled 'Adăugați persoană nouă'. It contains two input fields: 'Nume:' and 'Selectează imagine:', with a 'Choose File' button and a message 'No file chosen'. Below the form is a green 'Adaugă persoană' button and a green 'Înapoi la panoul de control' button at the bottom.

Figura 58. Pagina de adăugare a noi persoane în baza de date.

Prin accesarea butonului „Administrați utilizatorii” din panoul de control, aplicația afișează lista persoanelor înregistrate în baza de date, împreună cu ID-ul și numele acestora, precum și un buton dedicat ștergerii fiecărui utilizator, după cum se observă în Figura 59. Această funcționalitate permite gestionarea eficientă și rapidă a persoanelor autorizate în sistem.

The screenshot shows a web browser window titled 'Aplicație pentru controlul accesului' with the URL '127.0.0.1:5000/users'. The main content is a table titled 'Lista utilizatorilor din baza de date'. The table has columns 'ID', 'Nume', and 'Acțiune'. It contains two rows: one for Florin (ID 1) and one for Elena (ID 2). Each row has a green 'Șterge' button in the 'Acțiune' column. A green 'Înapoi la panoul de control' button is located above the table.

ID	Nume	Acțiune
1	Florin	<button>Șterge</button>
2	Elena	<button>Șterge</button>

Figura 59. Pagina de administrare a persoanelor din baza de date.

Prin accesarea opțiunii „Vizualizați istoricul accesărilor” din panoul de control, este afișat un tabel cu toate accesările înregistrate, care includ și numele persoanei detectate, data și ora evenimentului, conform Figurii 60. Această funcționalitate permite monitorizarea completă a tuturor accesărilor din sistem.

Nume	Data/Ora
Florin	2025-06-25 14:24:31
Florin	2025-06-25 14:23:11
Florin	2025-06-25 14:17:19
Florin	2025-06-25 14:16:38
Florin	2025-06-25 14:16:09
Florin	2025-06-25 14:15:51
Florin	2025-06-25 14:15:30
Florin	2025-06-25 14:04:21
Florin	2025-06-25 14:04:15
Florin	2025-06-25 14:04:04
Florin	2025-06-25 14:03:36
Unknown	2025-06-25 14:03:30
Florin	2025-06-25 14:03:20

Figura 60. Pagina de accesare a istoricului accesărilor.

## 7 Concluzii

Lucrarea realizată a avut ca scop implementarea unui sistem de control cu acces intelligent bazat pe modulul ESP32-CAM, integrat cu algoritmi de recunoaștere facială și detecție a obiectelor, capabil să semnalizeze corespunzător și să comande o încuietoare electromagnetică. Sistemul a fost proiectat astfel încât să detecteze prezența unei persoane cu ajutorul unui senzor PIR, să captureze imaginea acesteia prin camera ESP32-CAM și să o proceseze ulterior într-o aplicație Python care determină identitatea persoanei sau tipul obiectului detectat.

Rezultatele experimentale au confirmat că toate funcționalitățile propuse inițial au fost implementate cu succes. Modulul ESP32-CAM a reprezentat o alegere potrivită datorită integrării sale compacte a unei camere și a capabilităților de conectivitate Wi-Fi, la un cost redus și un consum energetic relativ mic, putând intra în mod de *deep sleep* pentru optimizarea consumului atunci când nu este detectată mișcare.

De asemenea, sistemul beneficiază de o interfață web intuitivă care permite vizualizarea imaginilor live, administrarea utilizatorilor și consultarea istoricului accesărilor, demonstrând astfel posibilitatea controlului de la distanță, atât timp cât dispozitivul și stația de lucru se află în aceeași rețea locală. Totuși, s-a observat o limitare legată de conexiunea wireless, întrucât funcționarea stabilă depinde de calitatea rețelei și de acoperirea acesteia în zona unde este montat modulul ESP32-CAM.

Pentru a asigura o acuratețe optimă a recunoașterii faciale și a detecției de obiecte, este necesar ca zona monitorizată să fie bine iluminată, deoarece senzorul camerei nu are o performanță ridicată în condiții de iluminare scăzută, care poate afecta claritatea imaginilor capturate și implicit precizia clasificării.

Din punct de vedere al aplicațiilor practice, sistemul poate fi utilizat cu succes în controlul accesului în incinte cu securitate ridicată, precum laboratoare, birouri administrative sau spații rezidențiale, oferind un plus de siguranță prin validarea vizuală a persoanelor autorizate și semnalizare sonoră corespunzătoare fiecărei situații.

Pentru dezvoltări ulterioare, se propune realizarea unui PCB dedicat, care să înlocuiască montajul actual pe protoboard pentru a asigura fiabilitatea și rezistență mecanică sporită. De asemenea, se poate extinde funcționalitatea aplicației web pentru a include control extins asupra componentelor hardware, cum ar fi activarea manuală a încuietorii sau a camerei și integrarea unor funcționalități suplimentare de analiză a datelor colectate. Totodată, se propune accent pe modulul de detecție a obiectelor, prin antrenarea algoritmilor pentru identificarea unui spectru mai larg de obiecte periculoase, crescând astfel aplicabilitatea sistemului în scenarii complexe de securitate.

În concluzie, proiectul a demonstrat fezabilitatea unui astfel de sistem integrat, bazat pe ESP32-CAM, care combină eficient tehnologia de recunoaștere vizuală cu control practic al accesului și oferă perspective viitoare promițătoare pentru implementări mai avansate în domeniul securității inteligente.

## Bibliografie

- [1] NEC Corporation (2022), *A brief history of Facial Recognition*; <https://www.nec.co.nz/market-leadership/publications-media/a-brief-history-of-facial-recognition/>
- [2] I. Adjabi, A. Ouahabi, A. Benzaoui, A. Taleb-Ahmed, *Past Present, and Future of Face Recognition: A Review*, Electronics 2020, 9(8); <https://doi.org/10.3390/electronics9081188>
- [3] National Institute of Standards and Technology (2010), *Face Recognition Grand Challenge (FRGC)*; <https://www.nist.gov/programs-projects/face-recognition-grand-challenge-frgc>
- [4] Yaniv Taigman, Ming Yang, Marc' Aurelio Ranzato, Lior Wolf (2014), *DeepFace: Closing the Gap to Human-Level Performance in Face Verification*, IEEE Conference on Computer Vision and Pattern Recognition (CVPR).
- [5] SimplyMac (2024), *First iPhone with Face ID*; <https://www.simplymac.com/ios/first-iphone-with-face-id>
- [6] Florian Schroff, Dmitry Kaleichenko, James Philbin (2015), *FaceNet: A Unified Embedding for Face Recognition and Clustering*, arXiv:1503.03832v3.
- [7] Analytics Vidhya (2019), *Introduction to FaceNet: A Unified Embedding for Face Recognition and Clustering*; <https://medium.com/analytics-vidhya/introduction-to-facenet-a-unified-embedding-for-face-recognition-and-clustering-dbdac8e6f02>
- [8] Analytics Vidhya (2021), *ArcFace: Facial Recognition Model*; <https://medium.com/analytics-vidhya/arcface-facial-recognition-model-2eb77080aa80>
- [9] Aly Khalifa, Ahmed A. Abdelrahman, Thorsten Hempel, Ayoub Al-Hamadi (2024), *Towards efficient and robust face recognition through attention-integrated multi-level CNN*, Multimedia Tools and Applications, 84, 12715-12737. <https://doi.org/10.1007/s11042-024-19521-0>
- [10] NEC Corporation (2025), *NEC Face Recognition Ranks First in NIST Accuracy Testing*; [https://www.nec.com/en/press/202504/global\\_20250409\\_01.html](https://www.nec.com/en/press/202504/global_20250409_01.html)
- [11] Facia.ai (2025), *Top Uses of Facial Recognition System in 2025*; <https://facia.ai/blog/top-10-uses-of-facial-recognition-system-in-2023/>
- [12] Yasmin M. Mohialden, Nadia M. Hussien, Doa M. Abd Ali (2023), *Enhancing User Authentication with Facial Recognition and Feature-Based Credentials*, Journal La Multiapp, 4(6).
- [13] Keyless (2024), *Facial Recognition applications, benefits and challenges*; <https://keyless.io/blog/post/facial-recognition-applications-benefits-and-challenges>
- [14] CyberLink (2024) *Facial Recognition for smart banking*; <https://www.cyberlink.com/faceme/insights/articles/599/facial-recognition-for-smart-banking>
- [15] Intexus (2024), *The future of airports: Passenger facial recognition for faster, safer boarding*; <https://intexus.la/en/blog-en/the-future-of-airports-passenger-facial-recognition-for-faster-safer-boarding>
- [16] NEC Corporation, *Fast Travel: Using Face Recognition to Improve Airport Services with a View towards Wide scale Implementation*; <https://www.nec.com/en/global/techrep/journal/g20/n01/200109.html>
- [17] Patrick Grother, Mei Ngan, Kayee Hanaoka, *Face Recognition Vendor Test (FRVT) Part 3: Demographic Effects*, NISTIR 8280, National Institute of Standards and Technology. <https://doi.org/10.6028/NIST.IR.8280>
- [18] European Data Protection Board (2023), *Orientările nr. 5/2022 privind utilizarea tehnologiei de recunoaștere facială în domeniul aplicării legii*, Versiunea 2.0.
- [19] Laishram H. Singh, Panem Chranarur, Naveen K. Chaudhary, *Advancements in detecting Deepfakes: AI algorithms and future prospects – a review*. *Discov Internet Things* 5, 53 (2025). <https://doi.org/10.1007/s43926-025-00154-0>

- [20] L. Laishram, M. Shaheryar, J. T. Lee, S. Ki Jung, Toward a Privacy-Preserving Face Recognition System: A Survey of Leakages and Solutions. *ACM Computing Surveys* 57, 6 (2025). <https://doi.org/10.1145/3673224>
- [21] U.S. Government Accountability Office (2024), *Facial Recognition Technology: Federal Law Enforcement Agency Efforts Related to Civil Rights and Training*; <https://www.gao.gov/products/gao-24-107372>
- [22] Security Industry Association (2020), *Facial Recognition Success Stories Showcase Positive Use Cases of the Technology*; <https://www.securityindustry.org/2020/07/16/facial-recognition-success-stories-showcase-positive-use-cases-of-the-technology/>
- [23] Sarler, I. H. Machine Learning: Algorithm, Real-World Applications and Research Directions. *SN COMPUT. SCI.* 2, 160 (2021). <https://doi.org/10.1007/s42979-021-00592-x>.
- [24] L. Alzubaidi, J. Zhang, A. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. Fadhel, M. Al-Amidie, L. Farhan. Review of deep learning concepts: CNN architecture, challenges, applications, future directions. *Journal of Big Data*, 8(1) (2021) [10.1186/s40537-021-00444-8](https://doi.org/10.1186/s40537-021-00444-8)
- [25] R. Qamar, B. Zardari. Artifical Neural Networks: An Overview. *Mesopotamian Journal of Computer Science* 2023: 130-139 (2023) [10.58496/MJCSC/2023/015](https://doi.org/10.58496/MJCSC/2023/015)
- [26] X. Wang, J. Peng, S. Zhang, B. Chen, Y. Wang, Y. Guo. A Survey of Facial Recognition. (2022) [10.48550/arXiv.2212.13038](https://arxiv.org/abs/2212.13038)
- [27] M. Wang, W. Deng. Deep Face Recognition: A Survey. *Neurocomputing*, 429 (2021) [10.1016/j.neucom.2020.10.081](https://doi.org/10.1016/j.neucom.2020.10.081)
- [28] J. Redmon, S. Divvala, R. Girshick, A. Farhadi. You Only Look Once: Unified, Real-Time Object Detection. (2016) <https://doi.org/10.48550/arXiv.1506.02640>.
- [29] Platforma Microsoft Teams, Prof. Dr. Ing. Doris Petreus, *Introducere in microcontrolere*.
- [30] ElectronicsHub, "Microcontroller Types and Applications", (2024). <https://www.electronicshub.org/microcontrollers/>
- [31] S. K. Srivastava, S. K. Sharma. A Comparative Study on Microprocessor and Microcontroller, International Journal of Research and Analytical Reviews (IJRAR), vol. 1, nr. 4.
- [32] Ai-Thinker, ESP32-CAM camera development board (2020), <https://docs.ai-thinker.com/en/esp32-cam>
- [33] Ai-Thinker, "ESP32-CAM module datasheet"
- [34] OmniVision Technologies, OV2640 datasheet
- [35] GreeksforGeeks. Universal Asynchronous Receiver Transmitter (UART) Protocol. <https://www.geeksforgeeks.org/computer-networks/universal-asynchronous-receiver-transmitter-uart-protocol/>
- [36] Analog Devices. UART: A Hardware Communication Protocol. <https://www.analog.com/en/resources/analog-dialogue/articles/uart-a-hardware-communication-protocol.html>
- [37] Electronics Hub . Basics of UART Communication. <https://www.electronicshub.org/basics-uart-communication/>
- [38] GreeksforGeeks. What is Serial Peripheral Interface (SPI). <https://www.geeksforgeeks.org/electronics-engineering/what-is-serial-peripheral-interface-spi/>
- [39] Circuit Basics. Basics of the I2C Communication Protocol. <https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/>
- [40] P. Ajmera. A review Paper on Infrared Sensor. International Journal of Engineering Research and Technology (IJERT). (2017). <https://www.ijert.org/>
- [41] Y. Pawar, A. Chopde, M. Nandre. Motion Detection Using PIR Sensor. International Journal of Engineering Research and Technology (IJERT), 5(4) (2017). <https://www.ijert.org/>

## Anexe

### Anexa 1 – Cod sursă Arduino IDE

```
#include "WebServer.h" //biblioteca dedicata serverului web
#include "WiFi.h"      //biblioteca dedicata conectivitatii WiFi
#include "esp32cam.h"  //biblioteca pentru functiile camerei
#include <Wire.h>       //biblioteca dedicata comunicatiei I2C
#include <LiquidCrystal_I2C.h> //biblioteca dedicata LCD-ului cu interfata I2C

LiquidCrystal_I2C lcd(0x27, 20, 4); //creeare obiect LCD

const char* WIFI_SSID = "iPhone - Florin"; //numele retelei WiFi
const char* WIFI_PASS = "24002400";      //parola retelei WiFi
const char* URL = "/cam.jpg";           //endpoint-ul URL pentru imaginea camerei

static auto RES = esp32cam::Resolution::find(640, 480); //rezolutia camerei
WebServer server(80); //obiect server web e portul 80

const int BUZZER_PIN = 14; //pinul pentru buzzer
const gpio_num_t PIR_PIN = GPIO_NUM_13; //pinul pentru senzorul PIR
const int LOCK_RELAY_PIN = 15; //pinul pentru releul incuietorii

unsigned long lastMotion = 0; //memoreaza momentul ultimei detectii de miscare
const unsigned long activeDuration = 60000; //durata pana la intrarea in deep sleep
bool lockIsOpen = false; //flag care indica daca incuietoarea este deschisa
unsigned long lockOpenTime = 0; //memoreaza momentul deschiderii incuietorii
const unsigned long lockOpenDuration = 10000; //durata activarii incuietorii
unsigned long messageStartTime = 0; //momentul afisarii mesajului pe LCD
bool messageActive = false; //flag pentru mesajul activ pe LCD
const unsigned long lcdMessageDuration = 5000; //durata de afisare mesaj in functie de acces pe LCD

String currentLcdLine1 = ""; //textul curent linia 1 LCD
String currentLcdLine2 = ""; //textul curent linia 2 LCD

//actualizeaza LCD doar daca textul s-a modificat
void safeLcdPrint(const String& line1, const String& line2) {
    if (currentLcdLine1 != line1 || currentLcdLine2 != line2) {
        lcd.setCursor(0, 0);
        lcd.print("          ");
        lcd.setCursor(0, 0);
        lcd.print(line1);
        lcd.setCursor(0, 1);
        lcd.print("          ");
        lcd.setCursor(0, 1);
        lcd.print(line2);
        currentLcdLine1 = line1;
        currentLcdLine2 = line2;
        Serial.printf("LCD Updated: '%s' '%s'\n", line1.c_str(), line2.c_str());
    }
}
```

```

}

//afiseaza mesajul implicit pe LCD cand sistemul e inactiv
void displayDefaultMessage() {
    safeLcdPrint("Priviti spre", "camera pentru acces");
}

//activeaza releul pentru deschiderea incuietorii
void triggerUnlock() {
    Serial.println("Incuietoare deschisa");
    digitalWrite(LOCK_RELAY_PIN, HIGH); //seteaza pinul releului pe high
    lockIsOpen = true; //seteaza starea incuietorii ca deschisa
    lockOpenTime = millis(); //salveaza timpul deschiderii
}

//inchide automat incuietoarea dupa durata setata
void handleAutoLock() {
    if (lockIsOpen && millis() - lockOpenTime >= lockOpenDuration) {
        digitalWrite(LOCK_RELAY_PIN, LOW); //seteaza pinul releului pe low pentru inchidere
        lockIsOpen = false; //seteaza starea incuietorii ca inchisa
        Serial.println("Incuietoare inchisa");
    }
}

//alerta buzzer pentru persoana recunoscuta
void handleBuzzerKnown() {
    Serial.println("Persoana cunoscuta");
    tone(BUZZER_PIN, 1000, 1000); //emite ton de 1000Hz timp de 1s
    server.send(200, "text/plain", "Acces permis: poarta deschisa"); //trimite raspuns HTTP
    safeLcdPrint("Persoana cunoscuta", "Acces permis"); //afiseaza pe LCD
    messageStartTime = millis(); //salveaza momentul afisarii
    messageActive = true; //seteaza mesajul ca activ
}

//alerta buzzer pentru persoana necunoscuta sau obiect periculos
void handleBuzzerUnknown() {
    Serial.println("Buzzer: Persoana necunoscuta");
    for (int i = 0; i < 3; ++i) { //3 bipuri scurte
        tone(BUZZER_PIN, 500, 200); //ton de 500Hz pentru 200ms
        delay(300); //pauza 300ms intre bipuri
    }
    server.send(200, "text/plain", "Persoana necunoscuta"); //trimite raspuns HTTP
    safeLcdPrint("Persoana necunoscuta", "Acces respins"); //afiseaza pe LCD
    messageStartTime = millis(); //salveaza momentul afisarii
    messageActive = true; //seteaza mesajul ca activ
}

//functie pentru activarea incuietorii
void handleUnlock() {
    triggerUnlock();
    server.send(200, "text/plain", "Incuietoare deschisa");
}

```

```

//captureaza imagine si o trimit ca raspuns HTTP
void serveJpg() {
    auto frame = esp32cam::capture();
    if (frame == nullptr) { //verifica daca captura a esuat
        Serial.println("CAPTURE FAILED");
        server.send(503, "text/plain", "Camera indisponibila"); //trimit cod eroare
        return;
    }
    Serial.printf("CAPTURE OK %dx%d %db\n", frame->getWidth(), frame->getHeight(),
static _cast<int>(frame->size()));
    server.setContentLength(frame->size()); //seteaza lungimea continutului
    server.send(200, "image/jpeg"); //trimit headerul HTTP
    WiFiClient client = server.client(); //obtine clientul
    frame->writeTo(client); //trimit datele imaginii
}

//se obtine endpoint-ul cam.jpg
void handleJpg() {
    if (!esp32cam::Camera.changeResolution(RES)) { //se schimba rezolutia camerei daca e nevoie
        Serial.println("CAN'T SET RESOLUTION!");
    }
    serveJpg(); //se trimit imaginea
}

// functie care initializeaza camera cu configuratia dorita
void initCamera() {
    using namespace esp32cam;
    Config cfg;
    cfg.setPins(pins::AiThinker); //seteaza pinii pentru modulul AI Thinker
    cfg.setResolution(RES); //seteaza rezolutia
    cfg.setBufferCount(2); //seteaza numarul de buffere
    cfg.setJpeg(75); //seteaza calitatea JPEG la 75
    bool ok = Camera.begin(cfg); //porneste camera
    Serial.println(ok ? "CAMERA OK" : "CAMERA FAIL"); //afiseaza statusul
}

//functie care conecteaza la reteaua WiFi modulul
void initWiFi() {
    WiFi.persistent(false); //nu salveaza credentialele in flash
    WiFi.mode(WIFI_STA); //seteaza modul station
    WiFi.begin(WIFI_SSID, WIFI_PASS); //incepe conectarea
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.printf("\nhttp://%s%s\n", WiFi.localIP().toString().c_str(), URL); //afiseaza IP-ul local
}

//initializeaza rutele serverului web
void initServer() {
    server.on(URL, handleJpg); // ruta pentru imaginea camerei
}

```

```

server.on("/buzzer/known", handleBuzzerKnown); // ruta pentru buzzer known
server.on("/buzzer/unknown", handleBuzzerUnknown); // ruta pentru buzzer unknown
server.on("/lock/unlock", handleUnlock); // ruta pentru activarea incuietorii
server.begin(); // porneste serverul
}

void setup() {
    Serial.begin(115200); // initializeaza comunicarea seriala la 115200
    delay(500);
    Serial.println("ESP32CAM START");

    // se verifica daca trezirea a fost din cauza senzorului PIR
    if (esp_sleep_get_wakeup_cause() == ESP_SLEEP_WAKEUP_EXT0) {
        esp32cam::Camera.end(); // opreste camera pentru reinitializare curata
    }

    pinMode(BUZZER_PIN, OUTPUT); // seteaza pinul buzzer ca output
    pinMode(PIR_PIN, INPUT); // seteaza pinul PIR ca input
    pinMode(LOCK_RELAY_PIN, OUTPUT); // seteaza pinul releu ca output
    digitalWrite(LOCK_RELAY_PIN, LOW); // initializeaza incuietoarea ca inchisa

    Wire.begin(4, 2); // initializeaza comunicarea I2C cu SDA=4 si SCL=2
    lcd.init(); // initializeaza LCD-ul
    lcd.backlight(); // aprinde backlight-ul
    displayDefaultMessage(); // afiseaza mesajul implicit

    esp_sleep_enable_ext0_wakeup(PIR_PIN, 1); // activeaza wakeup extern pe pinul PIR
    initWiFi(); // conecteaza la WiFi
    initCamera(); // initializeaza camera
    initServer(); // porneste serverul
    lastMotion = millis(); // salveaza timpul curent
}

void loop() {
    server.handleClient(); // gestioneaza cererile clientilor web

    // se verifica daca senzorul PIR detecteaza miscare
    if (digitalRead(PIR_PIN) == HIGH) {
        Serial.println("Miscare detectata");
        lastMotion = millis(); // actualizeaza timpul ultimei miscari
    }

    handleAutoLock(); // verifica daca trebuie inchisa incuietoarea

    // verifica daca a expirat durata mesajului pe LCD
    if (messageActive && (millis() - messageStartTime > lcdMessageDuration)) {
        Serial.println("Mesaj LCD expirat, revenim la default.");
        displayDefaultMessage(); // afiseaza mesajul implicit
        messageActive = false; // dezactiveaza mesajul in functie de recunoastere
    }

    // daca nu e detectata miscare mai mult timp, intra in deep sleep
}

```

```

if (millis() - lastMotion > activeDuration) {
    Serial.println("Inactivitate detectata. Intru in deep sleep...");
    lcd.clear(); //sterge LCD-ul
    lcd.noBacklight(); //stinge backlight-ul
    delay(100);
    esp_deep_sleep_start(); //intra in modul deep sleep
}

delay(20); //scurt delay pentru stabilitate
}

```

## Anexa 2 – Cod sursă aplicație Python

```

#librariile necesare aplicatiei web Flask
from flask import Flask, render_template, request, redirect, session, flash, Response
import pymysql #biblioteca necesara conectarii la baza de date MySQL
import face_recognition #biblioteca pentru recunoasterea faciala
import numpy as np #biblioteca pentru procesare numerica (vectori, matrici)
import urllib.request #biblioteca pentru cereri HTTP catre ESP32-CAM
import cv2 #biblioteca OpenCV pentru procesarea imaginilor
import requests #biblioteca pentru a trimite cereri HTTP
import time #biblioteca pentru temporizari
import threading #biblioteca pentru rularea functiilor in thread separat
import queue #biblioteca pentru cozi de date intre threaduri

#initializare instanta aplicatiei Flask
app = Flask(__name__)
app.secret_key = "secretkey123"

#URL-ul local al camerei ESP32-CAM de unde sunt preluate imaginile
ESP32_CAMERA_URL = "http://172.20.10.8/cam.jpg"

#se creeaza conexiunea la baza de date MySQL
db = pymysql.connect(host="localhost", user="root", password="", database="licenta")

#functie care incarca encodari din baza de date
def load_face_encodings():
    cursor = db.cursor() #se creeaza cursor pentru interogari
    cursor.execute("SELECT name, encoding FROM users") #se selecteaza numele si codarea
    din tabela users
    rows = cursor.fetchall() #preia toate rezultatele
    encodings, names = [], [] #listele pentru codari si nume
    for name, encoding_blob in rows:
        encoding = np.frombuffer(encoding_blob, dtype=np.float64) #se realizeaza conversia din
        blob in vector numpy
        encodings.append(encoding)
        names.append(name)
    return encodings, names #se returneaza listele

known_encodings, known_names = load_face_encodings() #se incarca codurile si numele in
memorie

```

```

#se configureaza calea catre fisierele YOLOv3
yolo_weights = "./YOLO/yolov3.weights"
yolo_config = "./YOLO/yolov3.cfg"
yolo_names = "./YOLO/coco.names"
net = cv2.dnn.readNet(yolo_weights, yolo_config) #se incarca reteaua YOLOv3
with open(yolo_names, "r") as f:
    classes = [line.strip() for line in f.readlines()] #se incarca liste de clase COCO
out_layers = [net.getLayerNames()[i - 1] for i in net.getUnconnectedOutLayers().flatten()]
#identifica layerele de iesire

frame_queue = queue.Queue(maxsize=1) #coada cu capacitate maxima 1 pentru frame-ul procesat

#functie detectie obiecte
def detect_objects(frame):
    height, width = frame.shape[:2] #se obtine dimensiunea imaginii
    blob = cv2.dnn.blobFromImage(frame, 1/255.0, (416, 416), swapRB=True, crop=False) #se creeaza blob pentru YOLO
    net.setInput(blob) #seteaza inputul pentru retea
    outputs = net.forward(out_layers) #se calculeaza predictiile

    boxes, confidences, class_ids = [], [], [] #liste pentru cutii, incredere si clase
    for output in outputs:
        for detection in output:
            scores = detection[5:] #scorurile claselor
            class_id = np.argmax(scores) #se identifica clasa cu scor maxim
            confidence = scores[class_id]
            if confidence > 0.5: #filtrare in functie de scorul de incredere > 50%
                cx, cy, w, h = (detection[0:4] * [width, height, width, height]).astype(int) #se calculeaza coordonatele
                x, y = int(cx - w/2), int(cy - h/2)
                boxes.append([x, y, int(w), int(h)])
                confidences.append(float(confidence))
                class_ids.append(class_id)

    indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4) #elimina suprapunerilor multiple
    if len(indexes) > 0:
        for i in indexes.flatten():
            x, y, w, h = boxes[i]
            label = f"{classes[class_ids[i]]} {confidences[i]:.2f}"

            danger_classes = ["knife"] #clase periculoase
            if classes[class_ids[i]] in danger_classes:
                box_color = (0, 0, 255) #rosu pentru obiecte periculoase
            else:
                box_color = (255, 0, 0) #albastru pentru obiecte normale

            cv2.rectangle(frame, (x, y), (x + w, y + h), box_color, 2) #se deseneaza chenarul
            cv2.putText(frame, label, (x, y - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5, box_color, 2)
#se scrie numele clasei

```

```

#trimite alerta sonora catre ESP32-CAM daca obiectul detectat este periculos
if classes[class_ids[i]] in danger_classes:
    try: requests.get("http://172.20.10.8/buzzer/unknown", timeout=0.5)
    except: pass

#functie recunoastere faciala
def recognize_faces(frame):
    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB) #se converteste imaginea in RGB
    face_locations = face_recognition.face_locations(rgb_frame) #se detecteaza fetele
    face_encs = face_recognition.face_encodings(rgb_frame, face_locations) #se calculeaza
    encodarile

    #compara codarea cu baza de date si trimite comenzi buzzerului si incuietorii in functie de
    rezultat
    for face_encoding, loc in zip(face_encs, face_locations):
        matches = face_recognition.compare_faces(known_encodings, face_encoding,
        tolerance=0.6) #se compara cu baza de date
        name = "Unknown"
        if True in matches:
            name = known_names[matches.index(True)]
            try:
                requests.get("http://172.20.10.8/buzzer/known", timeout=0.5)
                requests.get("http://172.20.10.8/lock/unlock", timeout=0.5)
            except: pass
        else:
            try: requests.get("http://172.20.10.8/buzzer/unknown", timeout=0.5)
            except: pass

    #salveaza inregistrarea in tabela logs
    cursor = db.cursor()
    cursor.execute("INSERT INTO logs (name) VALUES (%s)", (name,))
    db.commit()

    #deseneaza chenarul si afiseaza numele detectat in functie de rezultat
    y1, x2, y2, x1 = loc
    color = (0, 255, 0) if name != "Unknown" else (0, 0, 255) #verde pentru persoana cunoscuta
    si rosu pentru necunoscuta
    cv2.rectangle(frame, (x1, y1), (x2, y2), color, 2)
    cv2.putText(frame, name, (x1, y2 + 15), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

def camera_worker():
    #se creeaza un frame negru cu mesaj de eroare
    fallback_frame = np.zeros((480, 640, 3), dtype=np.uint8)
    cv2.putText(fallback_frame, "Camera indisponibila", (50, 240),
    cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)

    error_counter = 0
    max_errors = 5 #tolereaza 5 frame-uri ratate

    #se preia incontinuu imagini de la ESP32-CAM
    while True:

```

```

try:
    img_resp = urllib.request.urlopen(ESP32_CAMERA_URL, timeout=2)
    imgnp = np.array(bytearray(img_resp.read()), dtype=np.uint8)
    frame = cv2.imdecode(imgnp, -1)

    #procesare recunoastere si detectie
    recognize_faces(frame)
    detect_objects(frame)

    if not frame_queue.full():
        frame_queue.put(frame)

    error_counter = 0
except:
    error_counter += 1
    if error_counter >= max_errors:
        if not frame_queue.full():
            frame_queue.put(fallback_frame)
    time.sleep(0.1)

#strat thread o singura data
threading.Thread(target=camera_worker, daemon=True).start()

#ruta principala pentru autentificare
@app.route("/", methods=["GET", "POST"])
def login():
    if request.method == "POST":
        if request.form["username"] == "florinstan" and request.form["password"] == "parola123":
            session["logged_in"] = True
            return redirect("/dashboard")
        else:
            return render_template("login.html", error="Date incorecte!")
    return render_template("login.html")

#ruta pentru pagina principala de control dupa login
@app.route("/dashboard")
def dashboard():
    if not session.get("logged_in"):
        return redirect("/")
    return render_template("dashboard.html")

#ruta care returneaza fluxul video in browser
@app.route("/video")
def video():
    if not session.get("logged_in"):
        return redirect("/")
    return Response(stream_video(), mimetype="multipart/x-mixed-replace; boundary=frame")

#functia care genereaza stream video
def stream_video():
    while True:
        frame = frame_queue.get()
        _, jpeg = cv2.imencode('.jpg', frame)
        yield (b"--frame\r\nContent-Type: image/jpeg\r\n\r\n" + jpeg.tobytes() + b"\r\n")

```

```

#ruta care permite adaugarea unui nou utilizator in baza de date
@app.route("/add_user", methods=["GET", "POST"])
def add_user():
    if not session.get("logged_in"): return redirect("/")
    if request.method == "POST":
        name = request.form["name"]
        file = request.files["file"]
        if file:
            img_path = f"faces/temp.jpg"
            file.save(img_path)
            image = face_recognition.load_image_file(img_path)
            face_encs = face_recognition.face_encodings(image)
            if len(face_encs) > 0:
                enc = face_encs[0].astype(np.float64).tobytes()
                cursor = db.cursor()
                cursor.execute("INSERT INTO users (name, encoding) VALUES (%s, %s)", (name, enc))
                db.commit()
                global known_encodings, known_names
                known_encodings, known_names = load_face_encodings()
                flash("Persoana adaugata cu succes!")
            else:
                flash("Nicio fata detectata!")
    return render_template("add_user.html")

#ruta care permite vizualizarea istoricului accesului
@app.route("/logs")
def logs():
    if not session.get("logged_in"): return redirect("/")
    cursor = db.cursor()
    cursor.execute("SELECT name, timestamp FROM logs ORDER BY timestamp DESC LIMIT 50")
    logs = cursor.fetchall()
    return render_template("logs.html", logs=logs)

#ruta care permite administrarea utilizatorilor din baza de date
@app.route("/users")
def users():
    if not session.get("logged_in"): return redirect("/")
    cursor = db.cursor()
    cursor.execute("SELECT id, name FROM users")
    users = cursor.fetchall()
    return render_template("users.html", users=users)

#ruta care permite stergerea utilizatorilor din baza de date
@app.route("/delete_user/<int:user_id>")
def delete_user(user_id):
    if not session.get("logged_in"): return redirect("/")
    cursor = db.cursor()
    cursor.execute("DELETE FROM users WHERE id = %s", (user_id,))
    db.commit()

```

```

global known_encodings, known_names
known_encodings, known_names = load_face_encodings()
flash("Utilizator sters cu succes!")
return redirect("/users")

#ruta pentru inchidere sesiune de autentificare
@app.route("/logout")
def logout():
    session.clear()
    return redirect("/")

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000, threaded=True) #se porneste serverul Flask pe portul 5000

```

### **Anexa 3 – Cod sursă *login.html***

```

<!DOCTYPE html>
<html lang="ro">
<head>
    <meta charset="UTF-8">
    <title>Aplicație pentru controlul accesului</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
    <h2>Autentificare</h2>
    <form method="post">
        <label>Nume de utilizator:</label><br>
        <input type="text" name="username" required><br>
        <label>Parolă:</label><br>
        <input type="password" name="password" required><br><br>
        <input type="submit" value="Login">
    </form>
    {% if error %}
        <p class="error">{{ error }}</p>
    {% endif %}
</body>
</html>

```

### **Anexa 4 – Cod sursă *dashboard.html***

```

<!DOCTYPE html>
<html lang="ro">
<head>
    <meta charset="UTF-8">
    <title>Aplicație pentru controlul accesului</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
    <h1>Panou de control</h1>

    <a href="#" onclick="openStream()">Vizualizați imaginile live</a>

```

```

<br><br>
<a href="/add_user">Adăugați persoană nouă</a>
<br><br>
<a href="/users">Administrați utilizatorii</a>
<br><br>
<a href="/logs">Vizualizați istoricul accesărilor</a>
<br><br>
<a href="/logout">Ieșire</a>
<br><br>

<div id="streamOverlay">
  <span id="closeStreamBtn" onclick="closeStream()">x</span>
  
  <br><br>
</div>

<script>
  function openStream() {
    document.getElementById("streamOverlay").style.display = "flex";
  }

  function closeStream() {
    document.getElementById("streamOverlay").style.display = "none";
  }
</script>
</body>
</html>

```

## Anexa 5 – Cod sursă *add\_user.html*

```

<!DOCTYPE html>
<html lang="ro">
<head>
  <meta charset="UTF-8">
  <title>Aplicație pentru controlul accesului</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
  <h1>Adăugați persoană nouă</h1>

  <form method="post" enctype="multipart/form-data">
    <label>Nume:</label><br>
    <input type="text" name="name" required><br><br>

    <label>Selectează imagine:</label><br>
    <input type="file" name="file" accept="image/*" required><br><br>

    <input type="submit" value="Adaugă persoană">
  </form>

  {{% with messages = get_flashed_messages() %}}

```

```

{%- if messages %}
    <ul>
        {% for message in messages %}
            <li>{{ message }}</li>
        {% endfor %}
    </ul>
{%- endif %}
{%- endwith %}

<br>
<a href="/dashboard">Înapoi la panoul de control</a>
</body>
</html>

```

## Anexa 6 – Cod sursă *users.html*

```

<!DOCTYPE html>
<html>
<head>
    <title>Aplicație pentru controlul accesului</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>

<h1>Lista utilizatorilor din baza de date</h1>

<a href="/dashboard">Înapoi la panoul de control</a>
<br><br>

<table border="1" cellpadding="10">
    <tr>
        <th>ID</th>
        <th>Nume</th>
        <th>Acțiune</th>
    </tr>

    {% for user in users %}
        <tr>
            <td>{{ user[0] }}</td>
            <td>{{ user[1] }}</td>
            <td>
                <a href="/delete_user/{{ user[0] }}" onclick="return confirm('Sigur vrei să ștergi?')">Șterge</a>
            </td>
        </tr>
    {% endfor %}
</table>
</body>
</html>

```

## Anexa 7 – Cod sursă *logs.html*

```
<!DOCTYPE html>
<html lang="ro">
<head>
<meta charset="UTF-8">
<title>Aplicație pentru controlul accesului</title>
<link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
<h1>Istoricul accesărilor</h1>
<a href="/dashboard">Înapoi la panoul de control</a>
<br><br>

<table border="1">
<tr><th>Nume</th><th>Data/Ora</th></tr>
{%
    for name, timestamp in logs %
}
<tr>
<td>{{ name }}</td>
<td>{{ timestamp }}</td>
</tr>
{%
    endfor %
}
</table>

</body>
</html>
```

## Anexa 8 – Cod sursă *style.css*

```
body {
    font-family: Arial, sans-serif;
    background-color: #f8f8f8;
    text-align: center;
    padding: 30px;
}

form {
    background: #ffffff;
    padding: 20px;
    border-radius: 10px;
    display: inline-block;
    box-shadow: 0 0 10px rgba(0,0,0,0.1);
}

input[type="text"], input[type="password"], input[type="file"] {
    width: 100%;
    padding: 10px;
    margin: 10px 0;
}

input[type="submit"], a {
    padding: 10px 20px;
    background: #28a745; /* Verde */
    color: white;
```

```
text-decoration: none;
border-radius: 5px;
margin: 10px;
display: inline-block;
}
a:hover, input[type="submit"]:hover {
background: #1e7e34; /* Verde închis */
}
table {
margin: auto;
border-collapse: collapse;
width: 80%;
}
th, td {
padding: 10px;
border: 1px solid #aaa;
background: white;
}
#streamOverlay {
display: none;
position: fixed;
top: 0;
left: 0;
width: 100vw;
height: 100vh;
background: rgba(0,0,0,0.9);
justify-content: center;
align-items: center;
z-index: 9999;
}
#streamOverlay video,
#streamOverlay img {
max-width: 90%;
max-height: 90%;
border: 5px solid white;
border-radius: 10px;
}
#closeStreamBtn {
position: absolute;
top: 20px;
right: 30px;
font-size: 30px;
color: white;
cursor: pointer;
background: transparent;
border: none;
}
```