Command and Control Systems for Video Sensors Communications Based on Python Application for Mobile Smart Devices

Laurențiu-Bogdan DUDU, Florin POPESCU, and Petrică CIOTÎRNAE

Abstract—**In this paper it is proposed a command and control system for video communications based on Python application for mobile smart devices. There are presented the features for integration and development in a mobile communication device. Section I is concerned with the proposed objectives, Section II presents the most important concepts of the application used for developement: Python programming language, Flask (Python-based microframework), HTML (Hypertext Markup Language), HTTP (Hypertext Transfer Protocol) and the modules used to ensure a good behavior for the application. In Section III three the command and control center design steps, hardware configurations along with the software applications and explanation of how to implement the code are presented. The command and control of technical resources will be done by using Python, JavaScript and HTML scripts along with the Flask microframework on a Windows operating system. Section IV presents the conclusions, the problems encountered, whether the objectives and ways of improving the project have been achieved.**

Index Terms—**Flask; HTML; Python; HTTP, video communications for mobile devices.**

I. Introduction

A command and control system has the ability to provide features to monitor the state of operation of the equipment or services, and upon the data generated by the process, decisions are made at the network management level for the purpose of maximizing the systems performance.

In this paper we will create a system that will be able to manage a multiple video communications form video sensors using smart mobile devices and a 4G network infrastructure. All video communications will be sent to a mobile command center. The mobile command center will be able to make timely detection of evolving problems, prioritization, global or local influences and it will solve them in a short period of time. At the same time, an important aspect is identifying of the sensors and communication infrastructure alerts.

Automation is a critical requirement for the system, thus avoiding the manual configuration for each equipment in the network, that will translate into waste time. With this resource saved, the attention will be placed on real-time monitoring and control in the same control center. A necessary step is to carry out the processes exactly as in the design to deliver the same results.

L.-B. DUDU is with the Communication Department, Military Technical Academy “Ferdinand I”, Bucharest, Romania (e-mail: bogdan0695@gmail.com).

F. POPESCU is with the Communication Department, Military Technical Academy “Ferdinand I”, Bucharest, Romania.

P. CIOTÎRNAE is with the Communication Department, Military Technical Academy “Ferdinand I”, Bucharest, Romania (e-mail: petrica.ciotirnae@mta.ro).

Continuous monitoring from a specialized team, which will manually check the availability of system functions and the state of the equipment, will be solved with temporary solutions and delays. A control center that integrates these processes will immediately signal the problems and their severity and will decide more effectively what team to be sent to resolve it.

II. Defining Elements and Modules of the Proposed Application

For software implementation of the system we choose the Python suite application.

A. Python

Its versatility lies in various application areas (databases, web applications, and math), the ability to run on different platforms (Windows, Mac, Linux, Raspberry pi, etc.) and simple syntax.

B. Flask

The flexibility thus generated allows the development of applications on different equipment versions.

A framework is a platform for developing software that uses Python. It provides the foundation on which software developers can build programs specific to a particular platform type. It can include predefined classes and functions that can process incoming information, interconnect hardware, and interact with software systems. With these in mind, the process of developing a program is simplified [1].

Flask is a small-sized framework that provides the functionality and versatility of a regular development platform. Additionally, access to a vast number of libraries provides the framework for creating complex applications.

The virtual environment generated by Flask is a copy of Python in which the libraries can be installed separately without affecting the system globally. The advantage of a virtual environment is that it prevents the conflict between libraries and different versions of Python so the app only has access to what lies within it [2].

C. HTML

HTML is a language used by the browser to manipulate text, images and other content in a way that displays the desired format [3].

D. HTTP

Global communication is done between web clients and the web server where customers can represent browsers (Explorer, Chrome, etc.) or any type of program or terminal while servers are most often cloud computing.

TCP (Transmission Control Protocol) is a set of rules and communication procedures used to interconnect network terminals. It specifies how information is exchanged over the Internet to facilitate end-to-end communication. It also defines how communication channels are created and how information is fragmented on packets, transmitted and reassembled at the reception [4].

E. SocketIO

Sockets are a technology that allows two-way instantaneous communication using a single TCP (Transmission Control Protocol) connection, thus eliminating the need for customers to call an Application Programming Interface (API) for new content. The main advantage is the reduction of network load and the dissemination of information for a much larger number of customers.

SocketIO is a JavaScript library for web applications that provide real-time functionality. It is composed of two parts:

- The client's library running in the browser;
- Server library for "node.js".

Node.js is a platform built on the Google Chrome processing engine for a smooth integration of applications using a data network. It uses an event-based model, a key element at the heart of access and gives it the ideal features to develop effective applications centered on real-time communication [5].

Sockets are the most common solution in real-time applications. This allows the server to send messages to customers. Each time an event is called the server will receive it and send it to all the connected customers in that session.

F. OpenCV

Python is a programming language that is noticeable by the simplicity and readability of the code. It allows the programmer to express ideas in fewer lines of code. Compared with programming languages like C / C ++, it has lower capabilities. The main advantage is the ability to develop complex scripts in C / C ++ embedded in Python modules or libraries and its availability for different platforms: Windows, Linux, IOS, Android [6].

III. Practical Implementation

Taking into account the theoretical notions presented above, an application has been implemented that will simulate an equipment command system, see Fig. 1.

In order to meet the requirements, the following equipment was used:

- 1 TP-LINK wireless router TL-WR820N (it has the role of interconnecting the equipment in a wireless local area network);
- 1 PC (it has a dual functionality: on one hand it will host the Flask framework within which the virtual environment is generated and on the other hand it will contain some of the functions of the command center);
- 1 webcam (will substitute the perimeter monitoring part where the control center is located);
- 2 mobile terminals (are designed to enhance the versatility of the application by accessing any web browser and entering only the IP address of the server).

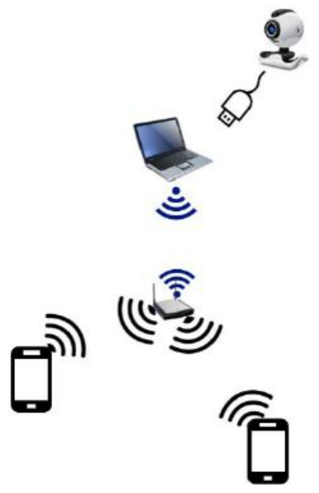


Figure 1. Test architecture

Both versatility and application flexibility allowed the development of the following scenarios:

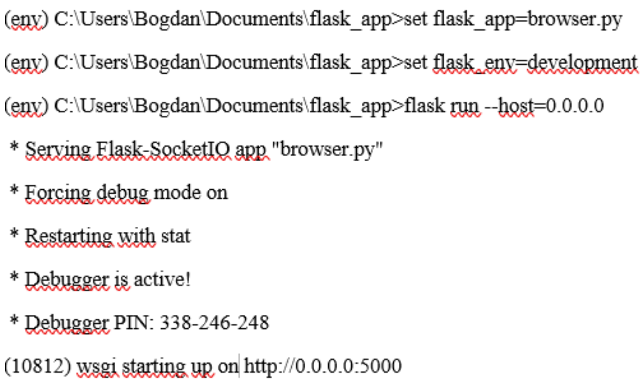
- When we want to restrict unauthorized access, we have a first obstacle represented by the router (uses AES-Advance Encryption Standard) and a second represented by a login form when accessing the main page;
- After completing the two security forms, it will open an HTML page with two buttons and a search box.

A. Initialization of the virtual environment

It will start by installing the framework using the "pip install flask" command. Being a closed virtual environment, the app will only have access to what's in the framework folder ("flask_app").

In the command line, we will position ourselves in the abovementioned folder and forward with the creation and activation of the environment and expect confirmation.

Next, the Python file that Flask will run at each server initialization will be set. Subsequently, debugging will be enabled, a useful function that reloads the source code whenever changes occur, then run the application with the command "flask run -host = 0.0.0.0". By default, the server is only visible from the terminal on which Flask is located. Setting the host with the value "0.0.0.0" allows the server to be externally visible (on the local network) and will tell the operating system to listen to all public IP addresses. In this case, all IPv4 addresses are searched locally, and network terminals can access the server on any of those IP addresses, see Fig. 2.


```
(env) C:\Users\Bogdan\Documents\flask_app>set flask_app=browser.py
(env) C:\Users\Bogdan\Documents\flask_app>set flask_env=development
(env) C:\Users\Bogdan\Documents\flask_app>flask run --host=0.0.0.0
 * Serving Flask-SocketIO app "browser.py"
 * Forcing debug mode on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 338-246-248
(10812) wsgi starting up on http://0.0.0.0:5000
```
Figure 2. Virtual server activation

B. Application development

Firstly, we'll create a Python script called "browser.py" which will include both the libraries and the routes on which the application depends.

A new route is accessed when the "/" (and optionally a name when at least two such functions are present) is added to the web browser along with the IP address of the server.

So, the following are defined, implementation example:

1) Login route

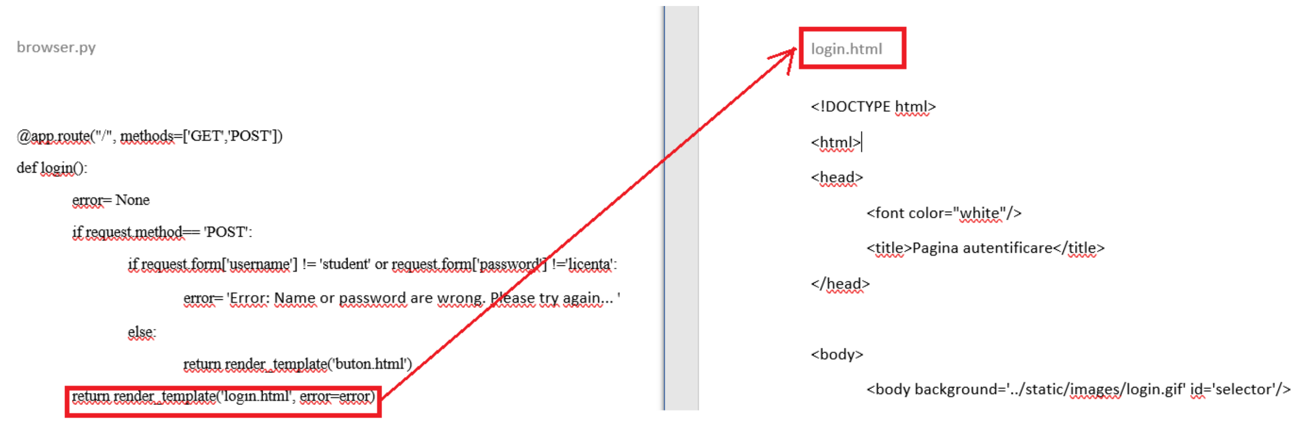
The script will check whether the entered information is valid and allow/deny access to the next page.

Initially, when you enter this route, an HTML document called "login.html" will be compiled.

When the client accesses the server URL, the subprogram for authentication is called. Without an error in the "error" variable and the default method being "GET" will open the web page "login.html".

Within the HTML file mentioned above, windows will be created to enter the user name and password. At the same time, window sizing has been pursued so that data entry is easily accomplished from any terminal.

Using the "POST" method, the name and password will be stored as "request.form" variables and will be interpreted in the "browser.py" source file, see Fig. 3.



browser.py

```
@app.route("/", methods=['GET','POST'])
def login():
        error= None
        if request.method== 'POST':
                if request.form['username'] != 'student' or request.form['password'] !='licenta':
                        error= 'Error: Name or password are wrong. Please try again... '
                else:
                        return render_template('buton.html')
        return render_template('login.html', error=error)
```

login.html

```
<!DOCTYPE html>
<html>
<head>
        <font color="white"/>
        <title>Pagina autentificare</title>
</head>

<body>
        <body background='../static/images/login.gif' id='selector'/>
```

Figure 3. Login procedure

Within the web page:

- Two value input windows will be created, one for the name, the other for the password;
- Add a button that has the function of memorizing the two values, sending them back to the previously mentioned route by the "POST" method and being checked in the script;
- A valid verification method will redirect your client to the "buton.html" web page. The title of the window and the page, the buttons, the background image and the page layout will be set here;
- Returning a wrong login form will display the message: "Error: Name or password is wrong".

2) Message route

The messaging route will begin by running the "chat.html" document when the corresponding web button is triggered.

To establish a secure communication path, this function will be called:

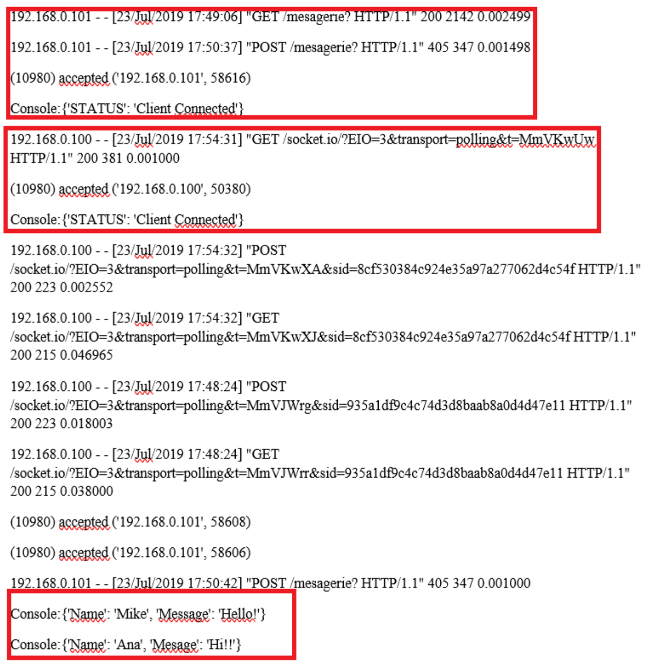
"app.config ['SECRET_KEY'] = '********'".

To receive messages from a client through WebSocket, the application will define events using the "socket.on" decorator and send messages using the "send ()" and "emit ()" functions.

At the moment of initiating the "current event" sequence, the branch "process current event" receives json objects (format used to store and transport information from the server to the web page), displays them in the Command Prompt console and then sends them to the next event called "response". This technique tells you that the message was sent successfully to the server, see Fig. 4.

It will set up the background image, page layout, and two boxes to enter a name and message in the open session. A CSS file will finalize the implementation of a GIF file and the centering of all text content in the web window. In the absence of a chat session, the message "Conversation didn't start!" is displayed on the screen.

The SocketIO library in Flask initiates a conversation with another client. SocketIO will encapsulate the web server when initializing it. From the command prompt window, all sent messages, when a user is logged in, and their IP address are displayed, see Fig. 5.

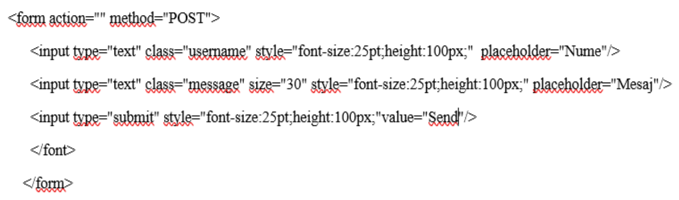


```
192.168.0.101 - - [23/Jul/2019 17:49:06] "GET /mesagerie? HTTP/1.1" 200 2142 0.002499

192.168.0.101 - - [23/Jul/2019 17:50:37] "POST /mesagerie? HTTP/1.1" 405 347 0.001498

(10980) accepted ('192.168.0.101', 58616)

Console:{'STATUS': 'Client Connected'}

192.168.0.100 - - [23/Jul/2019 17:54:31] "GET /socket.io/?EIO=3&transport=polling&t=MmVKwUw
HTTP/1.1" 200 381 0.001000

(10980) accepted ('192.168.0.100', 50380)

Console:{'STATUS': 'Client Connected'}

192.168.0.100 - - [23/Jul/2019 17:54:32] "POST
/socket.io/?EIO=3&transport=polling&t=MmVKwXA&sid=8cf530384c924e35a97a277062d4c54f HTTP/1.1"
200 223 0.002552

192.168.0.100 - - [23/Jul/2019 17:54:32] "GET
/socket.io/?EIO=3&transport=polling&t=MmVKwXJ&sid=8cf530384c924e35a97a277062d4c54f HTTP/1.1"
200 215 0.046965

192.168.0.100 - - [23/Jul/2019 17:48:24] "POST
/socket.io/?EIO=3&transport=polling&t=MmVJWrg&sid=935a1df9c4c74d3d8baab8a0d4d47e11 HTTP/1.1"
200 223 0.018003

192.168.0.100 - - [23/Jul/2019 17:48:24] "GET
/socket.io/?EIO=3&transport=polling&t=MmVJWrr&sid=935a1df9c4c74d3d8baab8a0d4d47e11 HTTP/1.1"
200 215 0.038000

(10980) accepted ('192.168.0.101', 58608)

(10980) accepted ('192.168.0.101', 58606)

192.168.0.101 - - [23/Jul/2019 17:50:42] "POST /mesagerie? HTTP/1.1" 405 347 0.001000

Console:{'Name': 'Mike', 'Message': 'Hello!'}

Console:{'Name': 'Ana', 'Mesage': 'Hi!!'}
```

Figure 4. Message notification in Command Prompt

In the next step, include the “jquery” and “socket.js” scripts in the HTML file.

Windows were created to enter the client's name and message and will be linked to the “socket.js” module using element classes. At the same time, the sizing of shapes was pursued so that data entry can be easily achieved from any terminal.



```
<form action="" method="POST">
   <input type="text" class="username" style="font-size:25pt;height:100px;"  placeholder="Nume"/>
   <input type="text" class="message" size="30" style="font-size:25pt;height:100px;" placeholder="Mesaj"/>
   <input type="submit" style="font-size:25pt;height:100px;"value="Send"/>
   </font>
  </form>
```

Figure 5. Defining classes for the chat windows

Next, the “io.connect()” command is used to initiate connections and create a session by connecting all clients to the same URL:

“http://192.168.0.100:5000”.

The first argument in the “socket.on()” function is the name of the event. The ones called “connect”, “disconnect”, “message” and “json” are generated by SocketIO and cannot be changed.

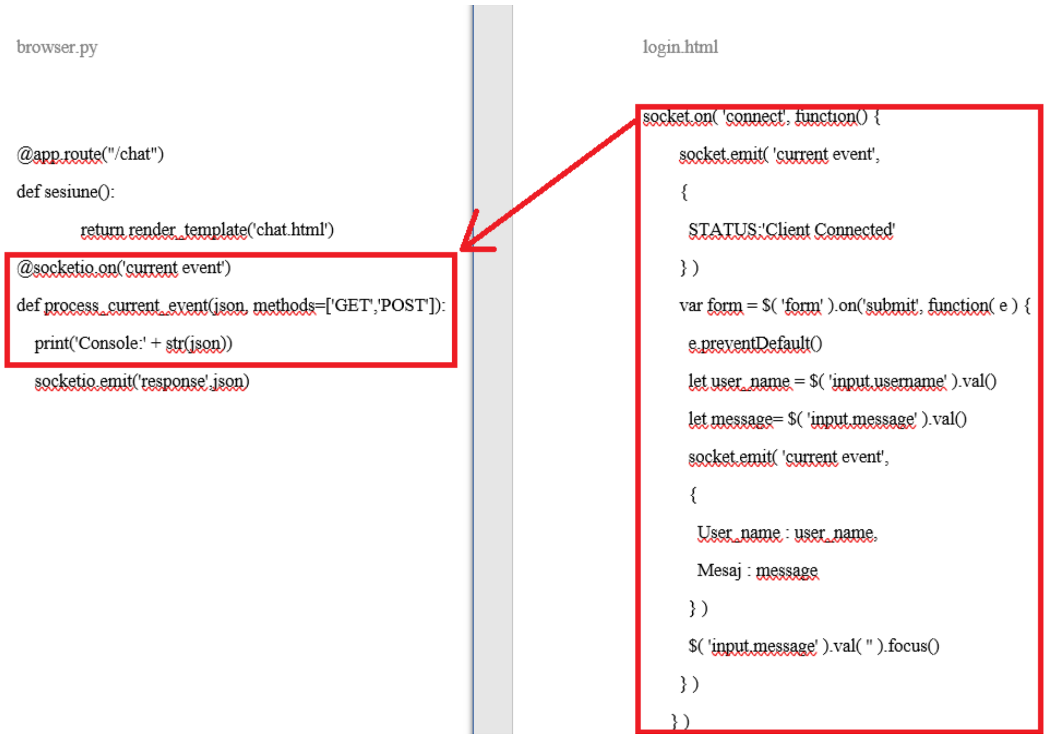
The “Message” variable sends the content as a string while “json” and customizable events transmit JSON content as a Python dictionary.

To send events, the server can use the “emit ()” and “send()” commands. The “emit ()” function sends a message as a custom event called “current event” along with the content.

Using the “POST” method, the “e” event is passed as an argument and the “preventDefault()” function preserves the name entered by the client so that it does not need to be reintroduced. Collected data is sent via the “emit” function to the custom event “current event” defined in the “browser.py” file. The “let” command allows entering values, see Fig. 6.

The message once sent via an event will be processed and sent to the HTML page.

After the first message in a session is sent, it must be displayed on the web page. In response, when the message “Conversation didn’t start....” will be deleted using the command: “$(`h3`).remove()”, see Fig. 7, Fig. 8.



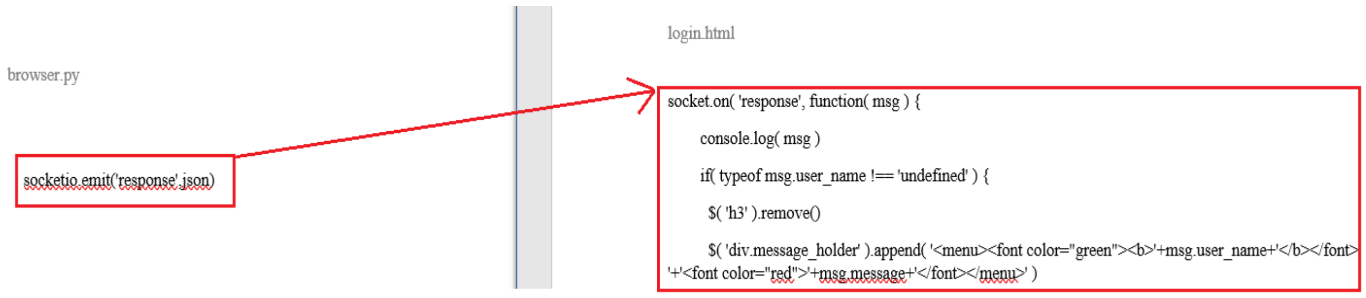
browser.py

```
@app.route("/chat")
def sesiune():
        return render_template('chat.html')
@socketio.on('current event')
def process_current_event(json, methods=['GET','POST']):
   print('Console:' + str(json))
   socketio.emit('response',json)
```

login.html

```
socket.on( 'connect', function() {
     socket.emit( 'current event',
     {
       STATUS:'Client Connected'
     } )
     var form = $( 'form' ).on('submit', function( e ) {
       e.preventDefault()
       let user_name = $( 'input.username' ).val()
       let message= $( 'input.message' ).val()
       socket.emit( 'current event',
       {
         User_name : user_name,
         Mesaj : message
       } )
       $( 'input.message' ).val( '' ).focus()
     } )
   } )
```

Figure 6. Authorization procedure



browser.py

```
socketio.emit('response',json)
```

login.html

```
socket.on( 'response', function( msg ) {
     console.log( msg )
     if( typeof msg.user_name !== 'undefined' ) {
       $( 'h3' ).remove()
       $( 'div.message_holder' ).append( '<menu><font color="green"><b>'+msg.user_name+'</b></font>
'+'<font color="red">'+msg.message+'</font></menu>' )
```

Figure 7. Message verification

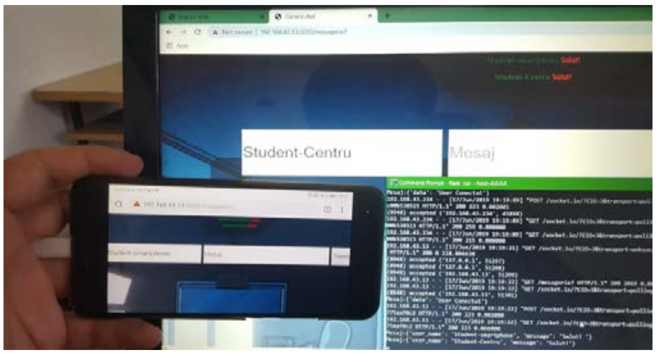
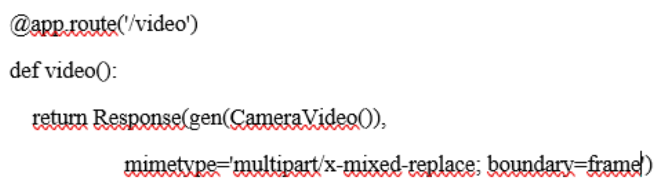


Figure 8. Communication between two clients

3) *Video route*

Forwarding this route will be done when the video button is operated and will return a webcam frame cascade in real time. The route will call the "video" function and will notify this action in the Command Prompt, see Fig. 9.



```
@app.route('/video')
def video():
   return Response(gen(CameraVideo()),
            mimetype='multipart/x-mixed-replace; boundary=frame')
```

Figure 9. Video route trigger

The MIME (Multi-purpose Internet Mail Extension) will be used to allow the web browser to display content in the desired mode. The server inserts a set of instructions at the beginning of the video in the following format: "Mimetype=multipart/x-mixed-replace; boundary = frame".

To create a stream from a sensor, an object will be responsible for the capture. The function argument is the device index or the name and path of a video file. The index is defined by a number to specify which sensor to use. If the source is internal, use "0" and for an external source "1":

```
"class CameraVideo(object):
        def __init__(sensor):
            sensor.video= cv2.VideoCapture(0)"
```

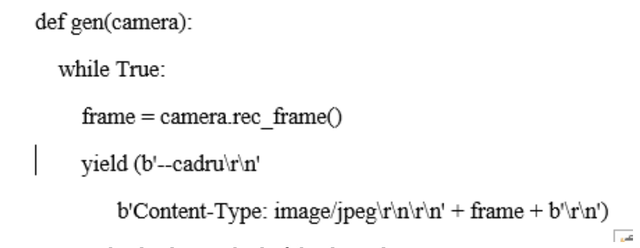
Frame capture requires software and then hardware release:

```
"def __del__(sensor):
        sensor.video.release()"
```

It will then proceed with the storage of that frame. For easier processing, the binary image will be converted into a matrix. This string will be converted to a coded image in "jpg" format. The "sensor.video.read()" function generates a Boolean variable (True / False):

```
"def rec_frame(sensor):

        succes, imagine= sensor.video.read()
        ret, jpeg = cv2.imencode(`.jpg`, imagine)
        return jpeg.tobytes()"
```

In the main script, the Boolean variable will serve as the threshold for displaying frames generated by the camera. Flask provides a native support for streaming using generator functions called "yield" (can be paused and resumed). Working with strings, there are "\ r" (Carriage Return) and "\ n" (Line Feed) to properly display the desired content. In Windows, they dictate the end of a line in the string, see Fig. 10.

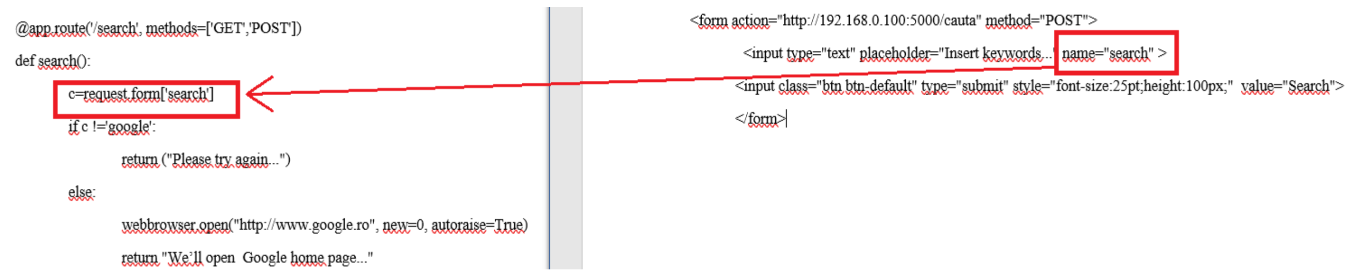


```
def gen(camera):
   while True:
      frame = camera.rec_frame()
      yield (b'--cadru\r\n'
         b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')
```

Figure 10. The display method of the desired image

4) *Search route*

The purpose of this route is to compare user-entered keywords to those linked to functions. The value entered is assigned to a variable using: "c = request.form[`search`]" which is then compared and returns an action in the system, see **Fig. 11**.



```
@app.route('/search', methods=['GET','POST'])
def search():
    c=request.form['search']
    if c !='google':
        return ("Please try again...")
    else:
        webbrowser.open("http://www.google.ro", new=0, autoraise=True)
        return "We'll open  Google home page..."
```

```
<form action="http://192.168.0.100:5000/cauta" method="POST">
    <input type="text" placeholder="Insert keywords..." name="search" >
    <input class="btn btn-default" type="submit" style="font-size:25pt;height:100px;"  value="Search">
    </form>
```

Figure 11. Correlation between keywords and search function

IV. Conclusion

The purpose of this project was to develop a command control system for smart devices able to transmit video information from remote sensors. Information is send to a mobile communications center to automate and facilitate different actions:

- The ability to view remotely and in real time what is happening in the control center;
- Rapid communication between intervention teams and authorized personnel;
- Power to perform remote actions.

The command center functionality puts a lot of emphasis on application resident in a smart device. Despite its limitations, the platform provides an ideal frame for testing hardware and software, and also provides real-time information about the different requests it receives from connected clients. Technical limitation has been used as a way to streamline and improve the code.

The versatility and flexibility of the Python programming language has helped to integrate HTML files that support the functionality of the application through a friendly interface. One of the development priorities is that any authorized terminal connected to the local network can access the application using only a web browser.

One of the difficulties encountered was communicating the data stored in variables in the HTML file to the source code in Python.

The existence of a closed virtual environment, even if it has a major advantage of locking actions locally, has also involved the installation of various libraries for a proper use of functions.

In the video function, getting raw information was easy, but processing and displaying was a challenge because a browser to interpret the information flow type needs a Multi-purpose Internet Mail Extension command. Also, due to the reduced bandwidth, it was found that when calling multiple users, the server will serve only the last client.

The messaging feature has no such bandwidth limitation, so messages could be sent simultaneously in a multi-session.

Correctly declaring the used methods in functions allowed the implementation of an authentication page that prevents unauthorized users from entering the system.

Despite the fact that the Python programming language is still not used to its full potential, a lot of automation in a few lines of code can be achieved, which is why it has become very popular in recent years.

ACKNOWLEDGMENT

This work was supported by a grant of the Ministry of Innovation and Research, UEFISCDI, project number 9SOL/12.04.2019 within PNCDI III.

REFERENCES

[1] M. Grinberg, *Flask Web Development: Developing Web Applications with Python*, 2nd ed. Sebastopol, CA: O'Reilly Media, 2018.

[2] P. Teixeira, *Instant Node.js Starter*. Birmingham, UK: Packt Publishing, 2013.

[3] K. Jamsa, *Introduction to Web Development Using HTML 5*. Burlington, MA: Jones & Bartlett Learning, 2014.

[4] R. Nixon, *Learning PHP, MySQL & JavaScript: With jQuery, CSS & HTML5*, 4th ed. Sebastopol, CA: O'Reilly Media, 2014.

[5] R. Rai, *Socket.IO Real-Time Web Application Development*. Birmingham, UK: Packt Publishing, 2013.

[6] R. Droms, "Dynamic Host Configuration Protocol," RFC 2131, Mar. 1997.