UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Politècnica Superior d'Enginyeria de Manresa

Universitat Politècnica de Catalunya

GRAU EN ENGINYERIA DE SISTEMES TIC

PROJECT: CONFIGURABLE WI-FI AP

Communication Systems

Ivan Chamero & Manuel Ángel Román

January 10, 2022

Contents

1 INTRODUCTION

The objective of this course project is the realization of a configurable AP. This can be created in multiple ways (with specific software like *Hostapd*, *ESP32* or other technologies). The access point can be from a local network or with internet access and has to allow the connection to the clients that it knows by its *MAC* address The access to the clients has to be according to the hours that they have established to connect to the point of access, if they are out of their hours of use, they cannot have access to the network provided by the AP.

It is about designing a system that is easy to configure, robust and efficient with the technologies we have to create access points.

2 SYSTEM ARCHITECTURE

Our group has chosen to take advantage of the features offered by *Linux* systems to create access points from its graphical environment. In this way, clients connect to our access point on our computer and are assigned an IP address thanks to the *dnsmasq* service that includes a DHCP server.

The core of the system is a Python server that controls clients with an intelligent system of threads. The server is connected to a local database that contains the characteristics of the users (MAC, time of entry, time of exit and connection status (connected/disconnected)). To control the traffic within the network provided by our access point, we have a firewall connected to the intelligent control system that denies and allows traffic to users according to the hours they have established. We are aware of the devices connected to our system thanks to an *ARP* scanner that we have implemented and it is capable of checking the status of a client by sending ICMP control packets (ping) to know the status of its connection.

The server offers a web service so that we can dynamically reconfigure client times, create new ones, and delete them. The website also informs us of the list of devices that our AP accepts and the connection status of the clients. It also dynamically displays when a customer connects or disconnects on your front page. All components are interconnected so that a change to the website such as deleting a user who is currently logged in will be denied access by the firewall and removed from the database. The following diagram shows a complete schematic of our system. Subsequently, each component of the final system will be detailed.

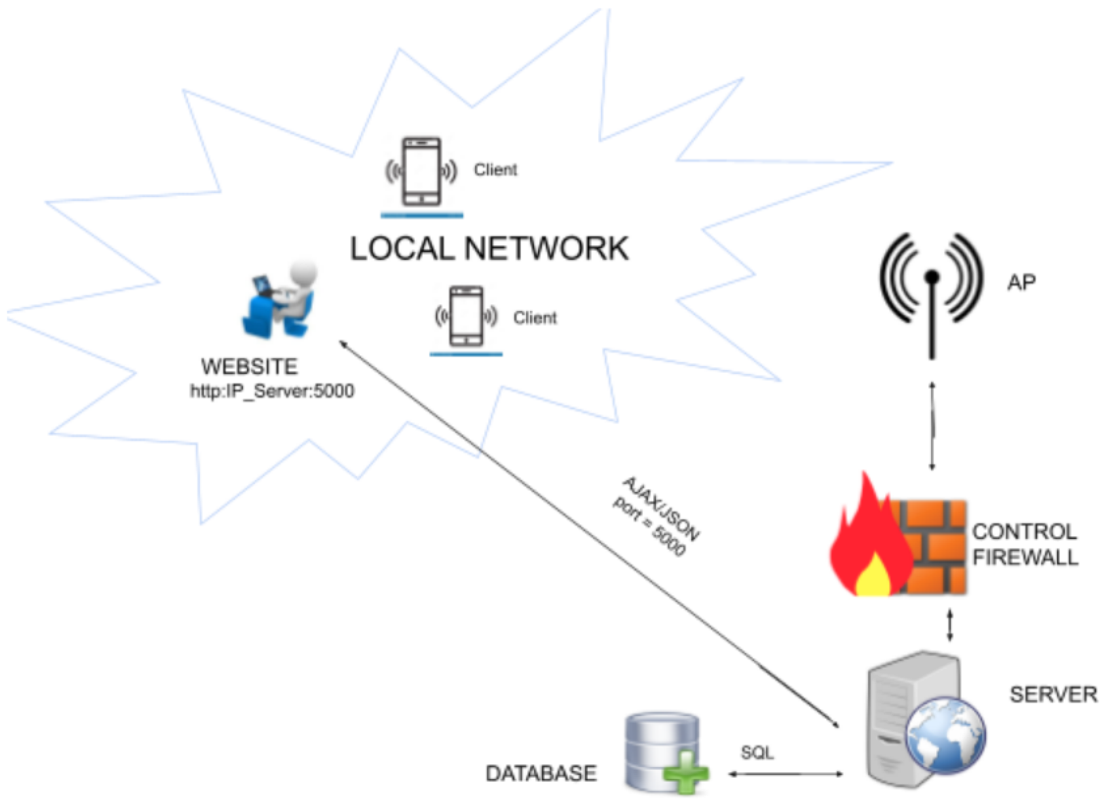


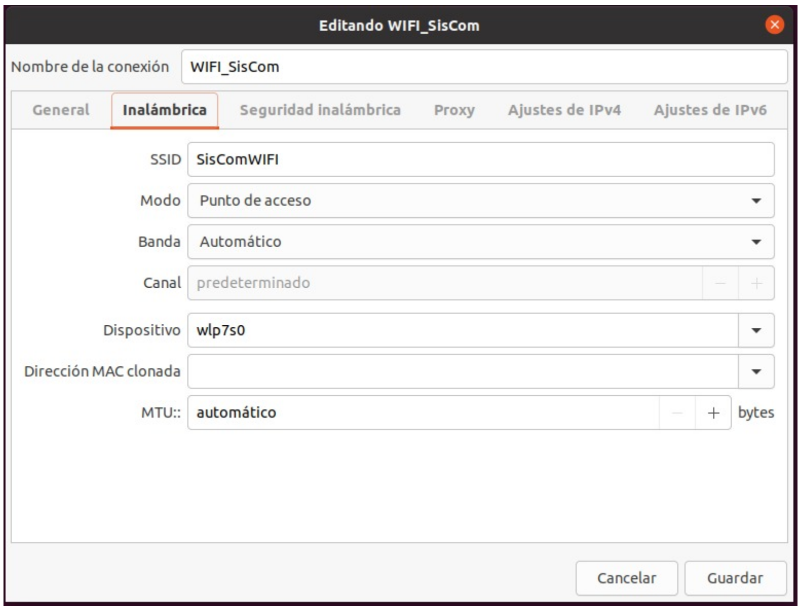
Figure 1: **System Architecture**

3 ACCES POINT

For the creation of the access point, we have used the tools provided by the Ubuntu operating system.

What we did was use the graphical application **nm-connection-editor**, which is already pre-installed in Ubuntu.

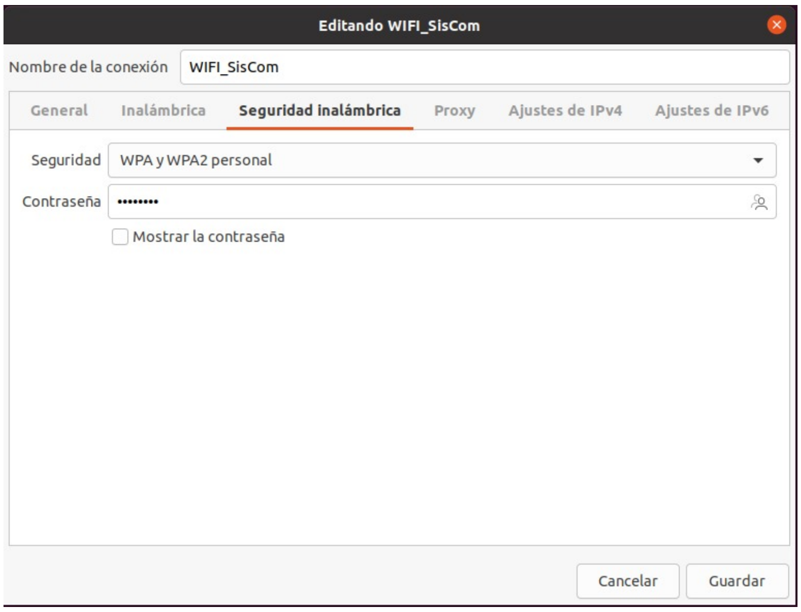
Once in the application, we add a new connection, where we define an SSID, select access point mode and determine my **wlp7s0** wi-fi network as the device. In wireless security, we select the type of security **WPA and WPA2** and we put a password.



Nombre de la conexión: WIFI_SisCom

SSID: SisComWIFI
Modo: Punto de acceso
Banda: Automático
Canal: predeterminado
Dispositivo: wlp7s0
Dirección MAC clonada:
MTU:: automático bytes

Figure 2: **AP configuration (1)**

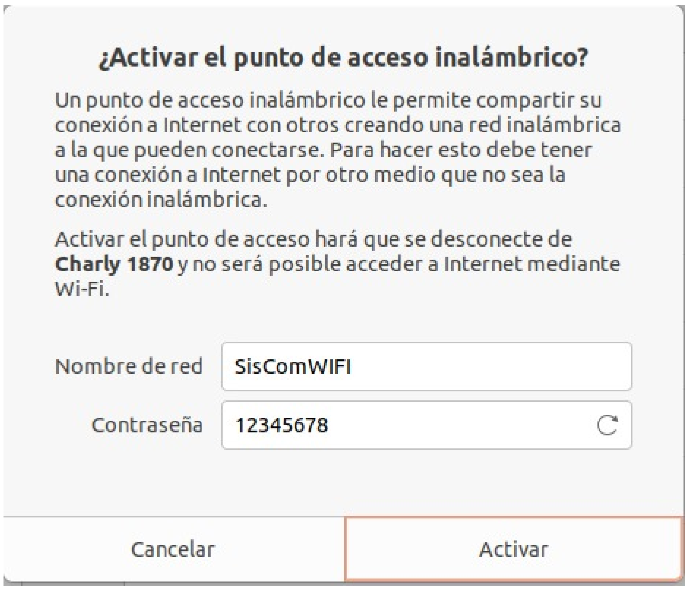


Nombre de la conexión: WIFI_SisCom

Seguridad: WPA y WPA2 personal
Contraseña: ••••••••
Mostrar la contraseña

Figure 3: **AP configuration (2)**

Once created, what we had to do was activate it, for this in the wireless configuration of Ubuntu, there is an option where it allows you to activate the wireless access point, the only thing we have to do is put the name of the connection and the password of the connection that we have created for the access point, and our pc will enter access point mode.



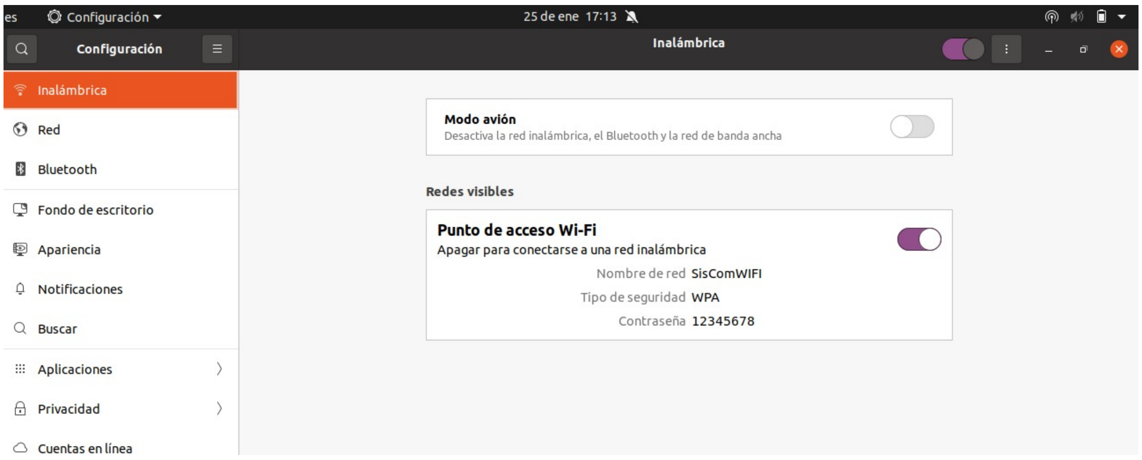
¿Activar el punto de acceso inalámbrico?

Un punto de acceso inalámbrico le permite compartir su conexión a Internet con otros creando una red inalámbrica a la que pueden conectarse. Para hacer esto debe tener una conexión a Internet por otro medio que no sea la conexión inalámbrica.

Activar el punto de acceso hará que se desconecte de **Charly 1870** y no será posible acceder a Internet mediante Wi-Fi.

Nombre de red: SisComWIFI
Contraseña: 12345678

Figure 4: **AP activation**



Modo avión
Desactiva la red inalámbrica, el Bluetooth y la red de banda ancha

Redes visibles

Punto de acceso Wi-Fi
Apagar para conectarse a una red inalámbrica
Nombre de red: SisComWIFI
Tipo de seguridad: WPA
Contraseña: 12345678

Figure 5: **AP activated**

In order to know who is connected to our device, we will use the operating system files that are in /proc/net/arp

4 SERVER

It is the core of our system. It is programmed in Python3 using the Flask framework that offers a web service with http. The server is in charge of managing the website and also controlling the devices at our access point through a firewall.

The server runs locally on the client that provides the access point on port 5000. It interacts with the database thanks to SQL statements hosted on Python3 thanks to the sqlite3 library. The module in charge of interacting with the database is the *models.py* file.

The server manages the website providing the information to the http requests made to it. Most requests are of the GET type to obtain a web resource such as the page to create, delete or modify the user. It also manages POST-type requests to collect the data from the forms, process them and return a response.

The server is structured in different modules where each one offers different functionalities. The following schematic shows a server module diagram:

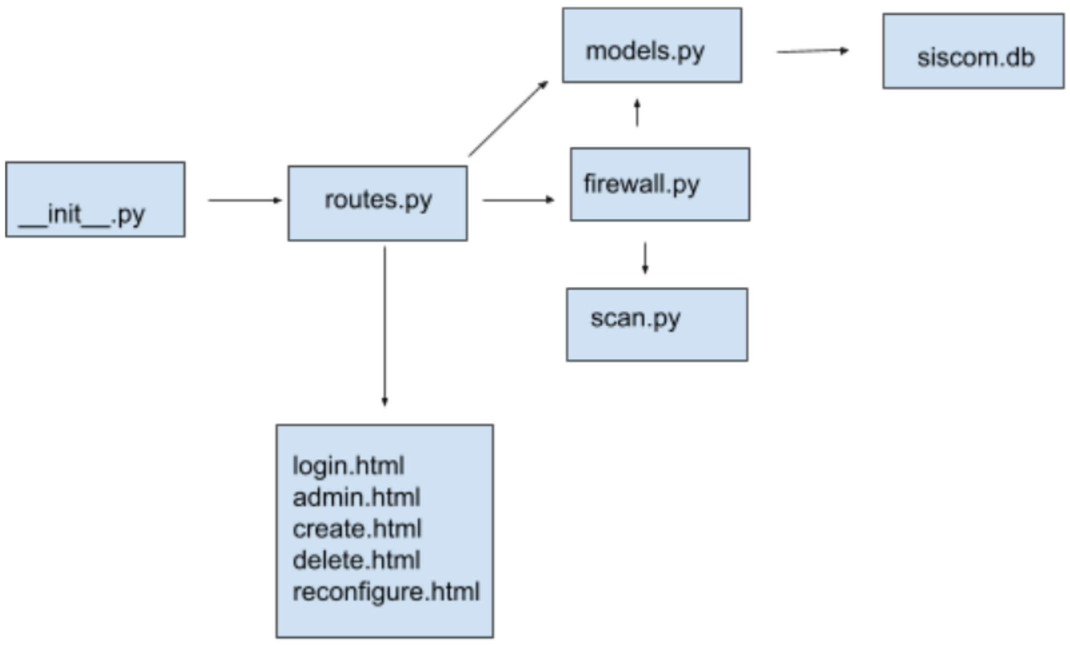


Figure 6: **Module diagram on the server**

As can be seen in the image, the main module is the init file and the routes file is the intermediary between the website and the firewall. It is responsible for rendering the HTML templates thanks to HTTTP requests of the GET and POST type when the forms are filled out. It interacts with the siscom.db database through the models file.

Once the server program is launched, the firewall system is activated (it will be explained in the Control System section) and you can interact with it thanks to a global variable ("firewall") that is an instance of the Control class of the server. firewall module in charge of all traffic management.

The routes file offers different routes with the methods it accepts. If the method is a GET, the html file will be rendered for the web interface and if it is a POST, it will update, delete or create new data in the database according to the user's wishes.

As we have mentioned, it is a dynamic system in such a way that it interacts dynamically with the control system and the firewall according to current characteristics.

The server also interacts with the user informing him of the users who connect, disconnect, reconnect within their hour, if their assigned time has run out, if traffic is allowed for the client specifying their MAC and if traffic is not allowed for the client with its MAC identifier.

In order to make the website more dynamic and to display the messages about which clients connect and disconnect to our access point, JavaScript is used, sending GET-type requests every second to the route http:ip:5000/api/changes. The way of exchanging messages between server and web client is in JSON format and the requests are made in AJAX, a JavaScript framework.

4.1 DATABASE

It is in charge of storing the information of the clients of our AP. The database is made in SQL language on python3 thanks to the sqlite3 library that it offers. The module to create the database is the file db.py (DATABASE directory) which creates an empty binary database file called *sis-com.db*.

It is a relational database with a single table 'USUARIS' that contains the attributes MAC, check-in time, check-out time and status. Status refers to the state of the client as connected or disconnected.

To interact with this database file, the models.py module is used, which allows creating, deleting and modifying parameters of the users of our system. It is important when starting the system to change the path of the variable *database* of the file *models.py* to interact with the local database of the device where it is located.

The *models.py* module is structured in functions to interact with the database. In this way, the server can interact with the database by calling the specific function of the interaction module.

The file has all the functions that our system needs, such as: create a new user with his entry and exit times, delete him and modify some parameters such as the entry and exit time and the connection status. The function names are intuitive and errors are handled by printing them to the screen and returning -1. There are also query functions to find out the parameters of one client or all clients.

Thanks to this module, the server routes file is capable of correctly rendering the information on the web portal and interacting with the database. On the other hand, not only does the route file interact with the database, but the firewall module also does it to change the status of a client, know the hours assigned to it, know the system's clients, etc.

4.2 ARP SCANNER

This module is in charge of knowing the clients connected to our AP and knowing the status of their connection. The program is written in python3 and corresponds to the file *scan.py*.

To know the list of devices that have connected or are connected to our AP, the Linux operating system functionality of the file *arp* found in the path */proc/net/arp* is used. The file contains the IP and MAC address of the client that has connected to our AP. We have opted for this option since we obtain the MAC and IP address of the clients and we can use them to search the database thanks to the MAC and carry out actions on a client (since in the database we identify it by the MAC) or to deny the traffic thanks to the IP address.

On the other hand, another of the features of this module is to know the status of a client. To do this, we send ICMP control packets in the form of *ping* thanks to the python *pyping* library. When we get a response from the client, we know that it is connected and if we do not receive a response or the host is not available (or does not exist) we know that it is offline. To send the ICMP control packets we use the IP address of the device that we have previously obtained from the *ARP* file.

Thanks to the functionalities of these modules, the control system knows the devices connected to our AP and the status of their connection, which will be seen by the shell when the server is executed and by the web portal. Thanks to obtaining the IP address of the device, we are able to deny the traffic in the firewall when the user is not at their assigned times.

4.3 CONTROL SYSTEM

It is in charge of allowing and denying traffic on our local network. The control module corresponds to the file firewall.py written in python3. This module is in charge of constantly consulting the connected devices and allowing or denying the traffic. It also allows you to dynamically delete and modify user parameters when a user is connected and is deleted from the system, deletes it.

To allow and deny the traffic we use the features of the *UFW* firewall through the operating system with the *subprocess* library offered by python3.

The system constantly consults the connections that exist in our AP thanks to the functionalities of the ARP scanner explained in the previous section. The database is also constantly consulted thanks to the models.py file to know the hours allowed for users, change the status of a user, etc.

The module is structured in classes since it is very easy to work with them and the final system is better structured. The 'Scheduler' class implements a task scheduler thanks to timer threads. It is used to program tasks such as executing a function that denies traffic to a client when its time has not expired. Thanks to timer threads we can create software interrupts that execute functions in a specific time. This class allows us to perform functions infinitely or once and cancel actions that are being executed as threads.

Thanks to this class we are also able to ensure that the firewall system is always activated at the time the server is launched. The server imports an instance of the Control class that is executed every second thanks to a timer thread to control the entire system. Thanks to that variable we can also interact with the firewall when there are changes to the website.

On the other hand we have the "Control" class. It is in charge of allowing traffic or not to clients according to their hours established in the database. Every second it executes the control function thanks to a timer thread of the Scheduler class. It is responsible for checking in the database if there is any device that currently has permission to access the local network. If it has it, it gives you permissions and activates a timer to deny you traffic when its end time is reached thanks to the timer threads of the previous class. To check the status of the devices, it uses the ARP scanner and if they are connected and have permission on the network, it modifies the status of the client in the database, showing the message that the client with a specific MAC has connected to the system. On the other hand, it also controls when a device is disconnected and displays the message that it has been disconnected. A client can have permission but not be connected and the timer thread for its hours is still active. It also controls the reconnection of clients in case they disconnect and reconnect to the network at their established times, displaying the client's reconnection message.

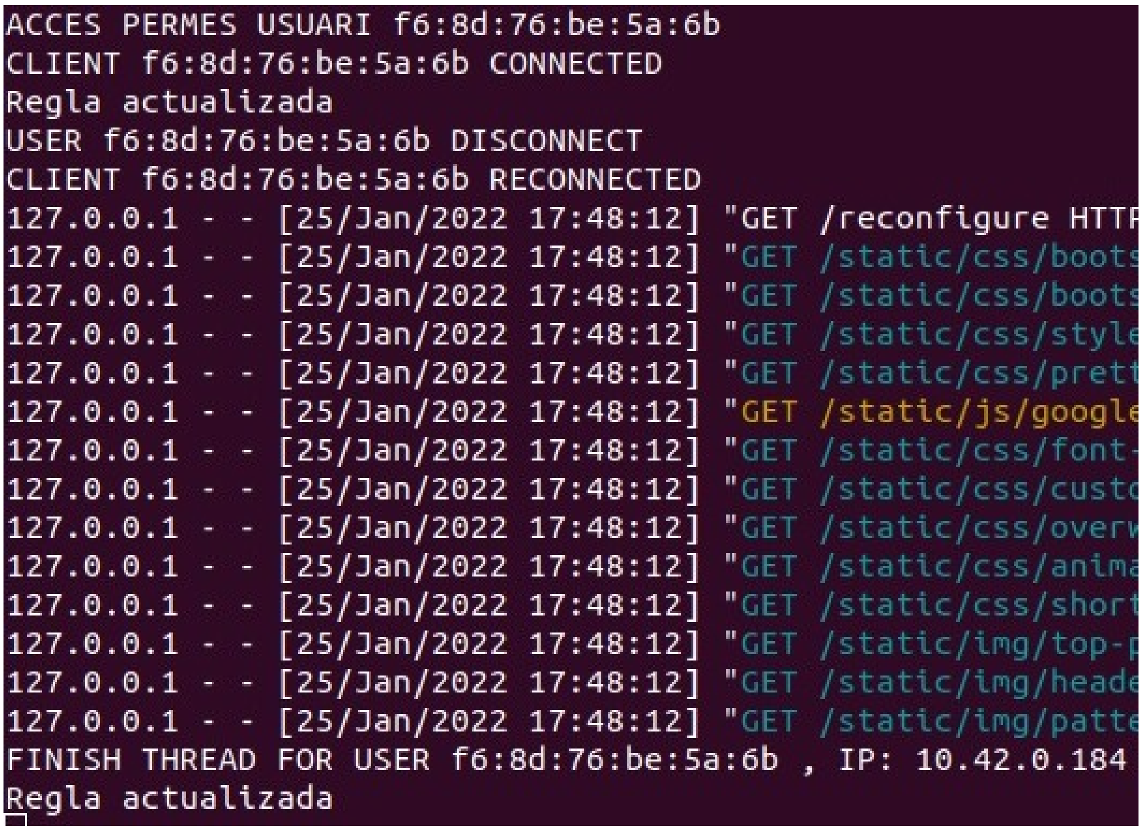
As we have mentioned before, it is a dynamic system so that from the web portal you can change the hours of a client, delete it or create it again. In the event that the data of a client is modified, the control system will be informed thanks to the global variable "firewall" in the routes file, which is an instance of the Control class. Once modified, the control system will look at the new hours assigned to it and if they correspond to the current hours, it will launch the termination thread again to deny access with the new hours, eliminating the old one. On the other hand, if the hours are changed and they are not the current ones, the system will deny access to the client with UFW. If a user is deleted from the web, the same procedure will be followed and traffic will be denied to the specific user.

The strategy we use is to collect the MAC address of the website that we want to make changes to. Once collected, we call the control class function to allow or deny access to that user.

The control class returns a dictionary with the currently connected devices that will be used in the routes file to inform the website when a client connects and disconnects compared to the ones it had previously.

Two dictionaries are used. One to save the connected clients and another that identifies each connected client with the corresponding timer thread. In this way, if we want to deny access to a user at a specific time (for example, by removing the user from the system while connected), knowing the user's MAC we can cancel the timer thread and deny access.

In the following image we can see the behavior of the system when a user connects, disconnects, reconnects and the assigned time runs out. In the image we can also see how the Firewall is activated and deactivated for that user when their hours are finished.



```
ACCES PERMES USUARI f6:8d:76:be:5a:6b
CLIENT f6:8d:76:be:5a:6b CONNECTED
Regla actualizada
USER f6:8d:76:be:5a:6b DISCONNECT
CLIENT f6:8d:76:be:5a:6b RECONNECTED
127.0.0.1 - - [25/Jan/2022 17:48:12] "GET /reconfigure HTTP
127.0.0.1 - - [25/Jan/2022 17:48:12] "GET /static/css/boots
127.0.0.1 - - [25/Jan/2022 17:48:12] "GET /static/css/boots
127.0.0.1 - - [25/Jan/2022 17:48:12] "GET /static/css/style
127.0.0.1 - - [25/Jan/2022 17:48:12] "GET /static/css/prett
127.0.0.1 - - [25/Jan/2022 17:48:12] "GET /static/js/google
127.0.0.1 - - [25/Jan/2022 17:48:12] "GET /static/css/font-
127.0.0.1 - - [25/Jan/2022 17:48:12] "GET /static/css/custo
127.0.0.1 - - [25/Jan/2022 17:48:12] "GET /static/css/overw
127.0.0.1 - - [25/Jan/2022 17:48:12] "GET /static/css/anima
127.0.0.1 - - [25/Jan/2022 17:48:12] "GET /static/css/short
127.0.0.1 - - [25/Jan/2022 17:48:12] "GET /static/img/top-p
127.0.0.1 - - [25/Jan/2022 17:48:12] "GET /static/img/heade
127.0.0.1 - - [25/Jan/2022 17:48:12] "GET /static/img/patte
FINISH THREAD FOR USER f6:8d:76:be:5a:6b , IP: 10.42.0.184
Regla actualizada
```

Figure 7: **Running system output**

5 USER INTERFACE

The web page is made with HTML and CSS with bootstrap, to obtain the data so that the page will be updated dynamically we have used JavaScrip and thanks to ajax that sends us the data in json format.

The first thing we see when we access the web is a login page where there is only one user created, admin, with the password also admin.

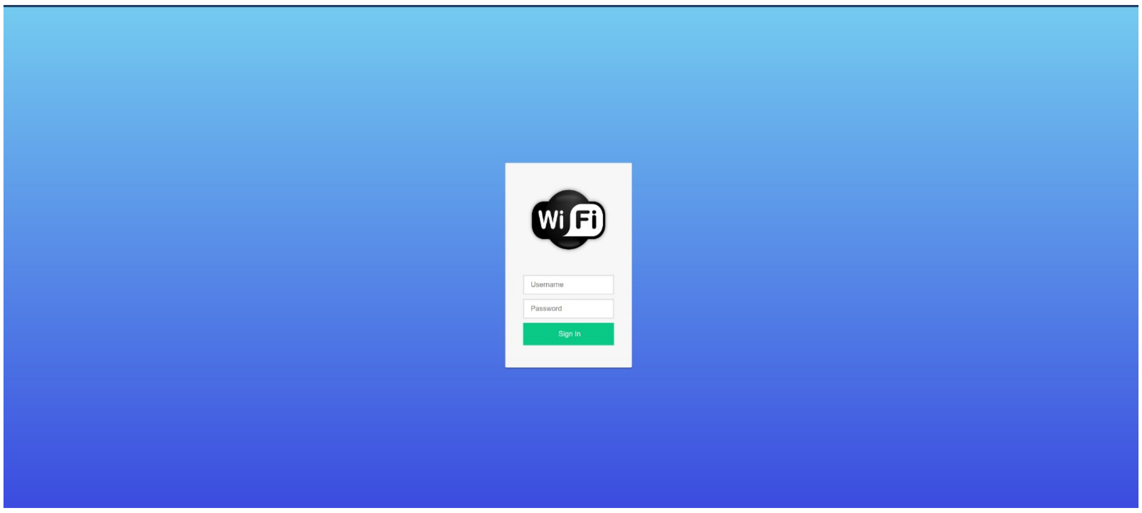
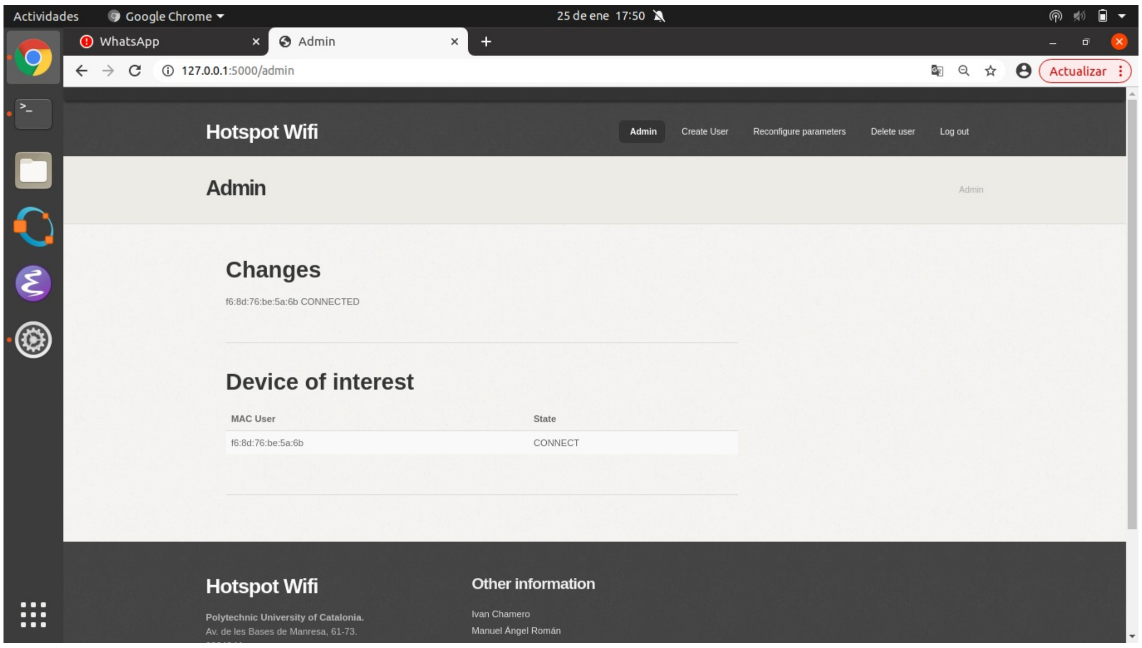


Figure 8: **Login page**

Once you enter you have several tabs, in the admin tab we will see the MAC and the status of the devices of interest, we will also see the changes that the devices make in the "Changes" section.



Admin

Changes

f6:8d:76:be:5a:6b CONNECTED

Device of interest

MAC User	State
f6:8d:76:be:5a:6b	CONNECT

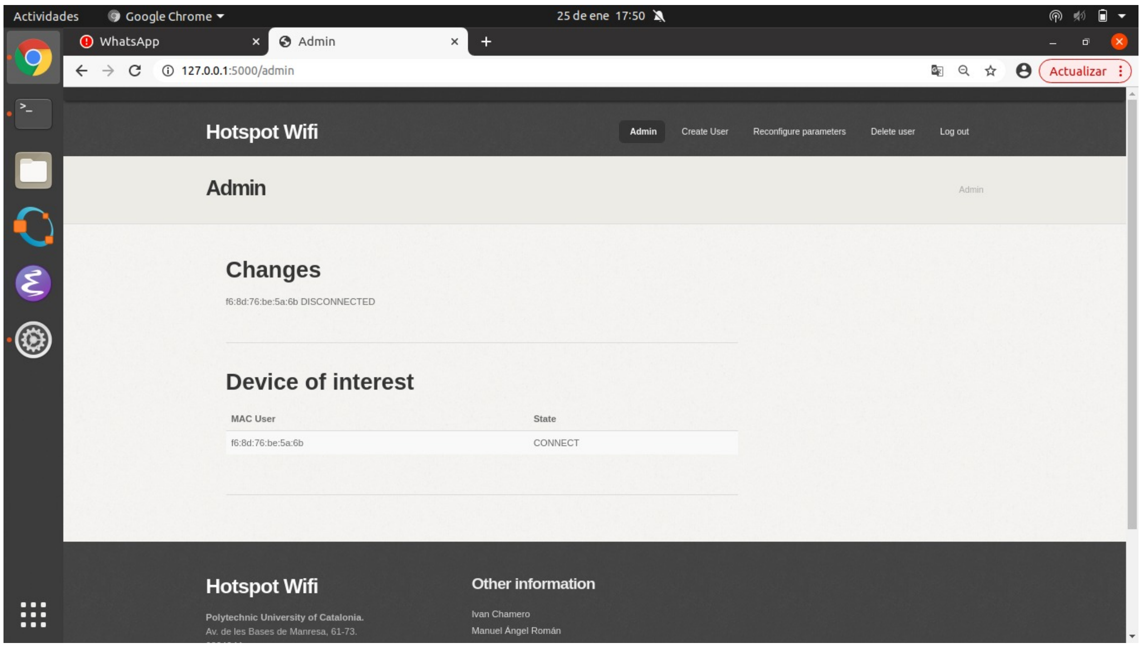
Hotspot Wifi

Polytechnic University of Catalonia.
Av. de les Bases de Manresa, 61-73.

Other information

Ivan Chamero
Manuel Ángel Román

Figure 9: **Admin page (1)**

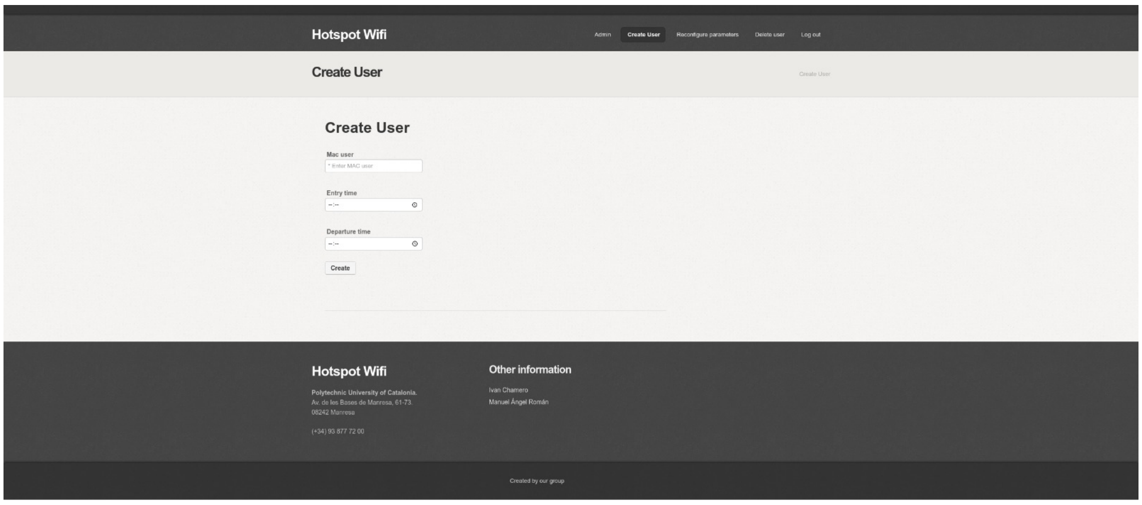


f6:8d:76:be:5a:6b DISCONNECTED

MAC User	State
f6:8d:76:be:5a:6b	CONNECT

Figure 10: **Admin page (2)**

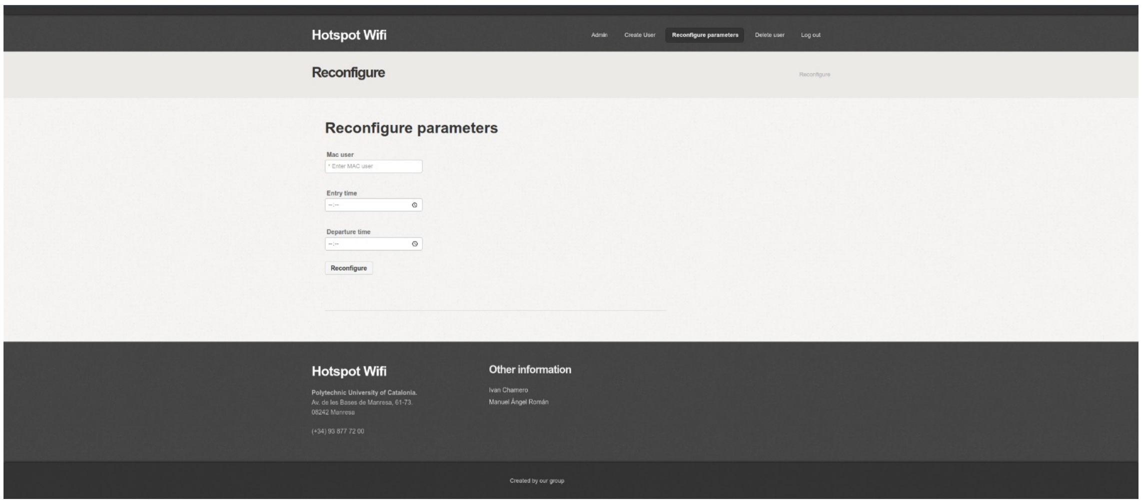
Then we have the add user tab, it is in this tab where we will add the devices of interest giving them a schedule to connect. Devices are added by MAC and two devices cannot be added with the same MAC.



Mac user: * Enter MAC user
Entry time: --:--
Departure time: --:--

Figure 11: **Add page**

Next we have the reconfigure user tab, in this tab we can change the hours at which we want the devices that are in the list of interest to connect. The hours of a device that has not been previously added cannot be modified.



Reconfigure parameters

Mac user: * Enter MAC user

Entry time: --:--

Departure time: --:--

Hotspot Wifi

Polytechnic University of Catalonia.
Av. de les Bases de Manresa, 61-73.
08242 Manresa

(+34) 93 877 72 00

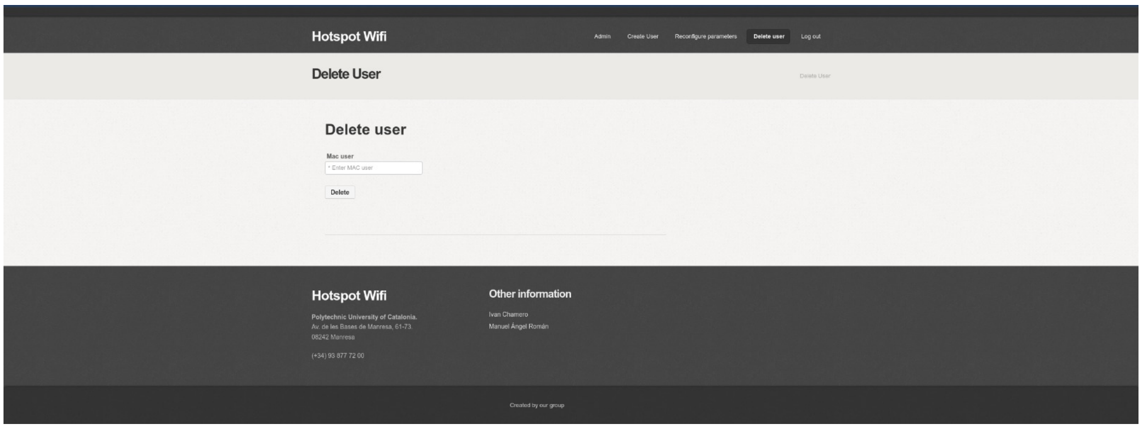
Other information

Ivan Chamero
Manuel Ángel Román

Created by our group

Figure 12: **Reconfigure page**

Finally we have the delete user tab, here we can delete a user from the list of users of interest. Users that do not exist in the list of users of interest cannot be deleted.



Delete user

Mac user: * Enter MAC user

Hotspot Wifi

Polytechnic University of Catalonia.
Av. de les Bases de Manresa, 61-73.
08242 Manresa

(+34) 93 877 72 00

Other information

Ivan Chamero
Manuel Ángel Román

Created by our group

Figure 13: **Delete page**

6 CONCLUSIONS

To conclude, we can say that we are very happy with the work carried out since we have achieved all the objectives that have been set for us. On the other hand, we have expanded our knowledge in the Linux environment both in firewall management and system file management and software timer threads.

The part that has been easier for us has been the database and the design of the user interface since we have elaborated more in other subjects. Even so, we have learned new functionalities.

As a negative point we must highlight the difficulties we had to start due to the incompatibility of Hostapd with our network cards. We also tried micropython on ESP32 but we had a lot of trouble getting the devices connected and ejecting them so we finally opted for this implementation. Regarding our implementation, the part that has cost us the most has been the intelligent control that it incorporates, interconnecting the web with a dynamic control system with timer threads, connection with a dynamic database, firewall management, etc.

Finally, we must emphasize that we are happy with our implementation since we have provided a functional and robust solution to the problem and we have applied our knowledge of the environment.