Escuela Politécnica Superior de Linares

Máster en Ingeniería de Telecomunicación

Universidad de Jaén

Escuela Politécnica Superior de Linares

ADVANCED TELEPRESENCE SYSTEM SUPPORTED BY WEBRTC

Alumno: Juan Núñez Lerma

Tutor: Prof. D. José Manuel Pérez Lorenzo
Prof. D. Raquel Viciana Abad

Depto.: Ingeniería de Telecomunicaciones

Septiembre, 2022

UNIVERSIDAD DE JAÉN

ESCUELA POLITÉCNICA SUPERIOR DE LINARES

Trabajo Fin de Máster

Curso 2021 – 2022

Advanced telepresence system supported by WebRTC

Alumno: Núñez Lerma, Juan

Tutor: Prof. D. José Manuel Pérez Lorenzo

Prof. D. Raquel Viciana Abad

Firma del autor

[illegible]

Firma del tutor

PEREZ LORENZO JOSE MANUEL - 23807798S
Firmado digitalmente por PEREZ LORENZO JOSE MANUEL - 23807798S
Fecha: 2022.09.07 19:17:01 +02'00'

Firma del tutor

VICIANA ABAD RAQUEL - 45595778A
Firmado digitalmente por VICIANA ABAD RAQUEL - 45595778A
Fecha: 2022.09.07 19:42:17 +02'00'

TABLE OF CONTENTS

INDEX OF FIGURES

INDEX OF TABLES

ABBREVIATIONS

AoIP: Audio over IP

API: Application Programming Interface

Asynchronous Transfer Mode: ATM

CBR: Constant Bit Rate

CSS: Cascading Style Sheet

DB: Database

DTSL: Datagram Transport Layer Security

GCP: Google Cloud Platform

GPL: General Public License

HTML: HyperText Markup Language

IETF: Internet Engineering Task Force

ICE: Interactive Connectivity Establishment

IDE: Integrated Development Environment

I/O: Input/Output

IP: Internet Protocol

IPTV: Internet Protocol Television

ISDN: Integrated Service Digital Network

MVC: Model – View – Controller

NAT: Network Address Translation

P2P: Peer-To-Peer

PBX: Private Branch Exchange

RTC: Real-Time Communication

RTSP: Real Time Streaming Protocol

RTP: Real Time Transport Protocol

RTCP: Real Time Control Protocol

rt-VBR: Real-Time Variable Bit Rate

SDP: Session Description Protocol

SIP: Session Initiation Protocol

SRTCP: Secure Real-Time Control Protocol

SRTP: Secure Real–Time Transport Protocol

STUN: Session Traversal Utilities for NAT

TCP: Transmission Control Protocol

TFM: Masther's Thesis

TURN: Traversal Using Relays around NAT

UDP: User Datagram Protocol

VoIP:	Voice over IP
VM:	Virtual Machine
W3C:	World Wide Web Consortium
WebRTC:	Web Real-Time Communication

1 RESUMEN

En este Trabajo se desarrolla y analiza atendiendo a prestaciones de servicio y red un sistema en la nube que por un lado permite la realización de videoconferencias múltiples en un entorno sanitario para la comunicación entre pacientes, familiares y personal sanitario, y por otro la gestión y reservas de cada una de las videollamadas a realizar. En particular, se ha diseñado un sistema web, basado en WebRTC, el cuál es capaz de establecer una comunicación multiusuario y ofrecer las características típicas de una videollamada a los usuarios finales.

Se ha utilizado Javascript y Node.js como lenguajes de programación para implementar un servicio web con una interfaz que permite la interacción entre los usuarios y la aplicación. Dicha interfaz está formada por dos secciones:

- Una sección destinada a los usuarios finales o pacientes/familiares, en el que pueden conectar a una videollamada que previamente ha sido dada de alta y es validada en el sistema.
- Y otra sección destinada al especialista o médico, donde se pueden dar de alta usuarios finales o residentes en la aplicación, al mismo tiempo que permite la modificación y gestión de las reservas a realizar.

2 ABSTRACT

This thesis focuses on a cloud system that was developed for the realization of multiple videoconferences in a healthcare environment for the communication among patients, relatives and staff. And on the other hand, it manages and reserves each video call in queue. Technically, a web system has been designed, based on WebRTC, which is capable of establishing multi-user communication and offering the typical characteristics of a video call to end users.

Javascript and Node.js were used to as programming languages to implement a cloud web service with an interface that allows interaction between users and the application. This interface can be parted into two sections:

- Patient’s interface intended for end users or patients/family member, in which they can connect to a video call that was previously been registered and is validated in the system.
- Doctor’s interface intended for the specialist or doctor, where end users or residents can be registered in the application, at the same time that it allows the modification and management of the reservations to be made.

3 INTRODUCTION

This chapter presents the background in which this work was framed and the circumstances that have motivated its development.

Nowadays, more and more telecommunication services are needed to support people's lives. In particular, videoconferencing services have evolved from being a tool used almost exclusively in the business environment to gradually, with the evolution of social networks and web-support technologies to real-time protocols, become a common tool for users in their daily lives. After the worldwide impact of the last pandemic, videoconferencing services are now part of the inherent services not only in large companies but also in SMEs to allow teleworking for example, in the educational field to allow a process of distance learning, and in the medical field. In the hospital environment, it is not only limited to remote consultations with patients, but has also been very useful in enabling communication between patients and their families during prolonged admissions where isolation was the norm. In this sense, more and more robotic platforms designed to work in these areas are incorporating telepresence as a necessary additional service.

In this sense, this work implements a cloud-based video call application and analyses the performance and deployment capabilities of WebRTC which is the real-time communication, protocol typically used for video calls. On the other hand, it is necessary to analyse the demands of the support servers associated with ICE in secure environments where it can even be deployed for security and privacy issues in the hospital's own facilities without resorting to external servers versus its use with a deployment in the cloud. This work has carried out a deployment taking into account the two scenarios and analysing not only the requirements but also the performance in terms of quality and service parameters.

As for the different analyzes that are going to be carried out, among which the limitations of bandwidth/jitter will be included, they are motivated by two fundamental reasons:

- In the event that no user needs to use TURN because all their NATs allow discovering pairs of public IP addresses/accessible ports, they are those imposed by the pairs and their respective connections and bandwidths. In this case the server, both the web server and the signaling server intervene little and only for communication with websockets for chat text, and user input and output.
- In the case where your networks block direct access, the use of TURN as a proxy will be required. In this situation, communication between these

peers is marked by the use of TURN. TURN can act as a bottleneck and this idea may also not be liked in the environment where the application will be used, since even if the flow is encrypted, it will go through the TURN server.

In this situation, there are two scenarios:

- Scenario 1: Develop everything in equipment that is located in the hospital or residence if it has a dedicated server or domain.
- Scenario 2: I have the servers in the same equipment but the TURN-proxy is external, either free or paid.

In addition, the structure of the current document is described, detailing the content of each of the chapters that comprise it.

3.1 Background

According to the RAE (Royal Spanish Academy) defines "video call" as: "simultaneous communication through a telecommunications network between two or more people, who can be heard and seen on the screen of an electronic device, such as a smartphone or a computer".

Depending on the criteria used to classify it, there are different options (depending on the information it carries), following the study by Javier Luque [1]:

1. Video conference: data, video or audio.
2. Shared spaces: where there is only data.
3. Audio conference: where there are only audios.

Likewise, depending on the characteristics and functionalities, the following types of videoconferences can be classified:

1. Specific videoconferences (carried out for a specific purpose).
2. Personal video conferences (or also desktop calls).
3. “Rollabout” room videoconferences (or also group calls).

Starting from this brief introduction, the origin of video calls will be summarized [2] [3].

It is considered that the First Videoconference did not arrive until April 20, 1964 at the Queens World's Fair in New York. Thanks to Bell Laboratories as it was marketed by AT&T (originally AT&T Bell Telephone Laboratories), despite there being multiple attempts of it.

When first experimented with televisions, it was clear that they could not only serve as audio or telegraph transmitters, but that they are also capable of displaying long-distance video. Not ignoring the necessary interrelated components that completes a video are:

1. Audio transmission equipment,
2. Cable channels,
3. An image capture device (a camera) and
4. A monitor

In 1927 when the first attempt was made by AT&T Bell Telephone Laboratories, during the Great Depression the development of communications was paralyzed. It was already in Germany around 1930 and began to give it impetus. That was how Georg Schubert managed to launch the first operating system, an ideal closer to what we know as modern telephony for commercial purposes.

This first prototype was named as the Gegensehn-Fernsprechanlagen. The post offices were filled with video call booths, but again, with the outbreak of the Second World War in 1939, its expansion stopped and it would not be until 1950 when its development began again.

Despite all these hurdles, AT&T continued to work on this communication system and thanks to the first commercial implementation of Picturephone Mod I in 1959. It was presented at the Queens World's Fair in New York on April 20, 1964.

One could name a series of problems that this new system presented, such as the parties that wanted to communicate had to be punctual in their respective booths, in addition to the high prices that they presented.

But all this did not stop the idea of reforming better and AT&T continued with more improvements such as the creation of a compact communication system.

This work and the urge to improve made other companies interested in this new way of communicating, such as the APO laboratory in the United Kingdom or the interest also shown by the USSR (Union of the Soviet Socialist Republics), although in this case, they did not establish a very accessible communication.

And it is here that the first video call attempt in a medical environment is known. The USSR intended to create an invention called Medical Video Phone, whose objective was none other than to ensure that visitors could communicate with those patients who could not have direct contact and therefore were in isolation, all this would be carried out carried out around the 70s. Another of the countries interested in the issue was Japan to create a network between Tokyo and Osaka for corporate use.

In 1980, when digital telephone networks appeared, when videoconferencing began to take off, especially when PC conferencing increased. Big labs including, again, AT&T, Compression Labs, Mitsubishi, IBM, and PictureTel, Connectix, etc., started making more powerful PCs, built-in color LCD compatible video phones, portable video phones, they created commercial mole webcams (QuickCam).

At this time the world begins to change thanks to all these advances in telecommunications and begins to notice the need for a development for multimedia delivery in real time.

It begins to talk about what ISDN is, the original Saxon term is ISDN (Integrated Service Digital Network), coined in 1972 by Japan and approved in 1984 by CCITT. Being a network, generally evolved from an integrated digital telephone network, which provides end-to-end digital connectivity, supporting a wide range of services, whether voice or other, and to which users can access through devices or multi-purpose interfaces [4].

Asynchronous Transfer Mode (ATM) was then developed to meet the needs of the Integrated Services Digital Network, Red Digital de Servicios Integrados (RDSI) in spanish, as defined in the late 1980s and designed to integrate telecommunications networks. It was designed for networks that must handle traditional high-performance data traffic (for example, file transfers) and real-time, low-latency content such as voice and video.

ATM is a set of standards of the telecommunications section of the ITU for arrangements in “cells”; in which information from many types of services, such as voice, video, or data, is contained in small "cells" of information the size variable. ATM networks are connection-oriented. It was originally conceived as a fast transfer technology for voice, video, and data over public networks [5].

Some characteristics of these networks that make them suitable for voice and video applications are: Constant Bit Rate (CBR) and Real-Time Variable Bit Rate (rt-VBR).

It is almost in the year 2000, with the birth of SIP/RTP when video calls take a qualitative leap and begin to become what we know them today.

Real-time transport protocol, or RTP, is an application-level protocol used for real-time transmission of information, such as audio and video in a video conference. It is developed by the IETF Audio and Video Transport Working Group, first published as a standard in 1996 [6].

In this same year, on February 22, 1996, Mark Handley and Eve Schooler presented to the IETF a draft of the Session Invitation Protocol v1 but it is v2 that is published as RFC 2543 in 1999.

It is complemented by the RTP (Real-time Transport Protocol), bearer of the voice and video content exchanged by the participants in a session established by SIP. SIP works as the standard for the initiation, modification and termination of interactive user sessions involving multimedia elements [7].

At that time, Asterisk became known, which is the project of Mark Spencer, who, in 1999, when he was a computer engineering student at the University of Alabama, founded a small company dedicated to providing technical support to Linux users [8].

Asterisk arises thanks to the SIP protocol. It is a free software program (under GPL license) that provides functions of a telephone exchange (PBX), supporting many VoIP protocols.

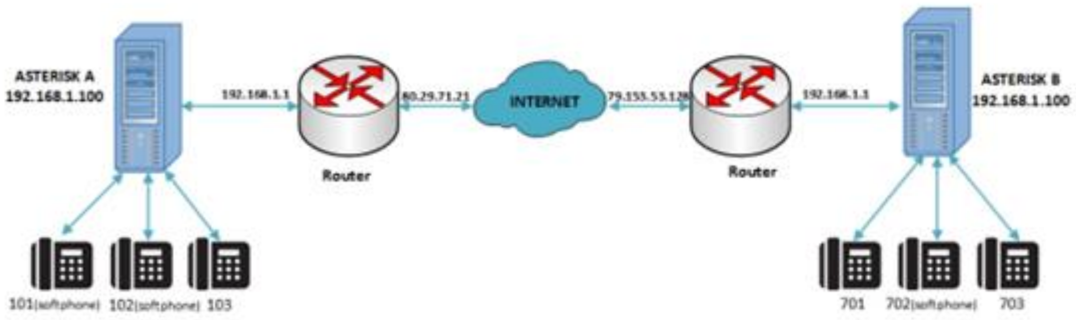


Figure 3-1. Asterisk scheme

From then on, VoIP softphones began to appear, such as Linphone, Zoiper...

They are free VoIP softphone to make phone calls with VoIP.

Some features are [9]:

- They allow free communication of voice, video and instant messaging.
- It works by the SIP protocol, which makes it compatible with any SIP server, even they have a SIP server to work with.
- Multi-platform.
- Supports IPv6.
- Support multi-call, call waiting and call transfer.

In 2003, two young university students, Jan Friis and Niklas Zenntrom, created a proprietary software to make calls over the Internet called Skype. This software is characterized by using a proprietary protocol that allows compression to improve voice quality. The company began as a voice calling service over the Internet, but its services gradually increased taking the first step towards what today would become one of the most widely used means of communication in any situation and part of the world [10].

On June 1, 2011, Google announced the launch of WebRTC, a technology that enables real-time communication (RTC) on the web.

In their statement, Google's Rian Linderberg and Jan Linden said: “Until now, real-time communications have required the use of proprietary technologies commonly acquired through plugins or by downloading clients. With WebRTC we are open-sourcing the audio and video technology engine we want from GIPS, giving developers access to cutting-edge technology." in signal processing under the free BSD license. This will allow developers to create audio and video chat applications through simple HTML code and JavaScript API" [11]. WebRTC is now a W3C and IETF standard, and one of the most important standards during the COVID-19 pandemic.

In a world in which we live frequently in communication and in which it is rare that we do not have a mobile device within our reach, there are many cases in which we are connected to WebRTC without being aware of them.

The increasingly advanced Smartphones, with mobile networks or Wi-Fi, as well as the various multimedia platforms, have managed to incorporate this new form of communication into our daily lives. The appearance of the Covid-19 has further boosted the use of video calls. As a result of the global pandemic, they have experienced a real boom, where the form of communication that we knew has completely changed, increasing its use both at work, in study or in personal use.

3.2 Justification

One of the main motivations for the development of this work has been the arrival of COVID-19. This virus emerged at the end of 2019, with a case of pneumonia, whose origin was unknown, in the city of Wuhan (China). It was from then on, that the health system began to count a series of pneumonias that began to have a great and easy expansion. All this alerted the World Health Organization (WHO) where they asked the Chinese health authorities to assess the real risk of this epidemic. What was initially thought was not going to go beyond the counted cases, on March 11, 2020, the WHO decreed a state of pandemic worldwide.

This situation overwhelmed all Health Systems, although this had already happened throughout history with other epidemics. But if something has made this brutal expansion unique, it has been through the contagion route, the respiratory route, which caused the rapid exchange between people.

This led to drastic measures, measures such as isolation, which lasted several months, being at the moment the most feasible way to prevent the virus from spreading. The world was not prepared for this home isolation for so long, which led to taking measures practically day by day, jobs and educational centers were closed, only the minimum services remained open. But then what about all those people who couldn't go to their jobs? What happened to all those students who couldn't go to their classes now? Or how could they communicate more closely with family or friends, other than a simple call or text message? And as far as we are concerned, how could family members communicate with patients who were admitted to hospitals?

Here a battle is born that would go against the clock to be able to continue in the most normal way possible. Platforms such as Messenger, Skype or Zoom would be among the most used to be able to telework or to be able to attend virtual classes (this last application has been one of the most used during all this time) and today, this method of work has remained implanted in many places to facilitate the work.

In the case of hospitals, it was necessary to create platforms where patients could communicate with health workers, without having to attend the centers and thus avoid, to a large extent, being exposed to this virus, with the additional difficulty that, as a general rule, patients couldn’t use their personal items, including mobile devices, during the first days in the hospital as a restriction rule.

Technological development has not only helped on this occasion, it had already been used in medicine for other problems, but in this project a technology will be developed that helps in medical practice, in a small part to what is called telemedicine (since doctors will be able to participate in video calls) but mainly it will contribute to the development of a robot that has been built to facilitate the communication of hospital patients with their familiar members via using network services.

The University of Jaén collaborates in a research project related to the development of a social robot for this type of applications, and the system implemented in this work will be used in order to establish communication at the patient's end without the need for active interaction of the patient on it.

To mention some precursor developments of telemedicine:

- Stethoscope (1890): created by Loeth (Chicago), it was a kind of stethoscope and a telephone, which allowed to capture the noises coming from the larynx, starting from a transmitter and a receiver.
- Biotelemetry (1883): it was based on the monitoring of vital signs, it was used by Marey, a French doctor.
- Einthoven (1903): the first remote transmission of an electrocardiogram, which carried the information from the hospital to a laboratory several miles away; work done by Willem Einthoven.

The first works considered within telemedicine were based on the transmission of videos, images or medical data, around 1960, but it will be from the 70s when telemedicine gains momentum thanks to the research work of NASA (since the astronauts could not travel to be able to attend a consultation with the health professional) [12].

3.3 Document structure

This document consists of a series of chapters detailing the process of designing and implementing a web application for video calls in a healthcare environment. Specifically, the distribution of the content is as follows:

- **Chapter 1**. Summary. This chapter summarizes the objectives of this work (in Spanish)
- **Chapter 2**. Summary. This chapter summarizes the objectives of this work
- **Chapter 3**. Introduction. This chapter establishes the context in which the need to be satisfied and the solutions adopted to solve the initial problem are framed.
- **Chapter 4**. Objectives. The objectives to be achieved with this Master's Thesis will be detailed
- **Chapter 5**. State of art. Provide the appropriate context for the Master’s Thesis by analyzing the environment in which it is developed.
- **Chapter 6**. Materials and Methods. It gathers all the information related to the design process of the work: the technologies and techniques used, architecture models adopted and relevant code fragments about the implementation.
- **Chapter 7**. Development. Implementation and design process of the web application.
- **Chapter 8**. Results and Discussion. The different results obtained after the completion of this work will be specified, as well as the execution times obtained with the applications used.
- **Chapter 9**. Conclusions. This chapter presents the conclusions drawn during the development of this work. It also describes the knowledge and skills acquired during the process.
- **Chapter 10**. Future lines of work. Some future lines of work will be mentioned.
- **Chapter 11**. Bibliographical References. Reviews of books, online documents, journals and websites that have been consulted during the completion of the dissertation.
- **Chapter 12**. Annexes. A series of annexes have been incorporated as a reference where some additional services implemented in the development of this work will be explained and an introduction to the use of Node.js will be made. In addition, a user manual for the web application will be added.

4 OBJECTIVES

In this Master's Thesis (TFM) the intention is to develop a system based on WebRTC technology that allows not only to establish a video conference for multiple users, but also to include additional functionality associated with the existence of a text chat, the invitation of users via email, the booking of common appointments, etc. The system must allow for the interconnection of more than one user connected from different networks and should incorporate security mechanisms. The performance associated with the analysis of the behavior of protocols such as the Interactive Connectivity Establishment (ICE) and WebRTC for multimedia data distribution must therefore be analyzed.

For this, the web application will be divided in two sections, one for the specialist or doctor and the other for the patient/family member. The module with the most functionalities will be the specialist, in which you can:

- Define each one of the system administrators, being able to register each one of them in their corresponding part.
- Register residents with all their corresponding data, which will be linked to the familiar member by email.
- Create and modify reservations to make a video call on a specific date and time for a patient and their familiar member.
- Possibility of enter and/or participate in a video call that has been registered by the specialist.

While in the patient section (video call interface), only the typical actions or operations offered in a conventional video call can be carried out. This video call needs to have been previously registered by the specialist.

For the development of this application the following objectives have been defined:

- Review of the specific bibliography that allows knowing the current situation of the state of the art.
- Study of the characteristics and implementation possibilities of web and signaling servers.
- Study and choice of technologies to be used.
- Study of a method that allows the creation of secure rooms and the method by which different users have access to the information.
- Integration between the video call system and the management system.
- Design and implementation of a method for checking, validating and classifying video calls.

- Analyze the possible differences between deploying on an internal server of an organization with all the necessary resources (including TURN) or through the use of external ICE servers (TURN/STUN).
- Analyze with typical connections in a home (case of a doctor or relative who joins the videoconference), the maximum number of connected users compared to one of course payment.
- Analyze the possible signaling server load when this can be used in a real situation.

5 STATE OF ART

WebRTC is rapidly gaining ground and is set to revolutionize standards of communications, for that reason it has been of great interest to carry out this Project.

The main objective will be to use WebRTC technology, along with some others, to create a multiplatform application in which videoconferences can be held, instant messaging between relatives and patients in a hospital.

After an exhaustive analysis, there are not many systems that provide these services in the medical field, for that reason this project is going to focus on this field, providing an extra resource to patients, family members and doctors in any hospital.

Obviously at this point there are platforms similar to the one that has been proposed, not in the medical field but in general, but they have some disadvantages which will be discussed below.

For example, we can speak of several such as:

- Google Hangouts, a Google conference call service that operates on the web. It was the first developed by Google that has later been divided into: Google Chat and Google Meet.
- Google Chat, previously Hangouts Chat. It includes a direct message exchange service, group chats and advanced features such as spaces, chat bots and a very powerful search function.
- Google Meet, previously Hangouts Meet. Includes group video calls with high-fidelity sound, automatic subtitles, and screen sharing. Depending on the edition of Google Workspace you have, Meet may also include advanced features, such as the ability to host meetings with up to 500 participants, record meetings, and broadcast live within a domain [13].
- Big BlueButton is an open-source web application for video conferencing and eLearning or distance education. It is a program distributed under the GNU license and has arisen from the reuse of various projects such as Asterisk, Flex SDK, Red5, MySQL and others.
- OpenTok provides a free API that allows any web developer to introduce group chat and video features to their websites. The platform is based on Adobe Flash technology and allows users to conduct video sessions with chat and live video for up to 20 participants. Allows you to perform invitations to use the service through accounts in the social networks Twitter, Facebook or MySpace.

Regarding the area that concerns us, we can say that with the arrival of the global COVID-19 pandemic, many hospitals tried to develop the video call service through different means to be able to connect family members and hospitalized patients.

To name some of the projects that were carried out, we can cite these:

- SMS To Video: The Information Technology Area of the Castilla-La Mancha Health Service (SESCAM) makes this project effective thanks to the use of mobile devices, smartphones and tablets and specific videoconferencing software, with the support of the health professionals themselves.

  The Danish company “Incendium” offered its “SMS To Video” software to enable this communication. Its ease of use and security in communication makes it possible to send an SMS from a SESCAM device to any family mobile, with a link to videoconference with patients [14].
- Unidos: The Son Llàtzer University Hospital launched the “United” project, which consists of bringing isolated patients due to COVID-19 closer to their families and professionals from different fields.

  According to Javier Giménez, head of audiovisuals at Son Llàtzer, “the project consists of providing the patient with a tablet or a telephone with an internet connection (4G) so that they can make video calls to their family or friends and to professionals from various fields. The system uses two different channels depending on the purpose of the call: if it is to contact family or friends, we use the usual means of connection, since these calls do not violate the Data Protection Law; but if the call is addressed to a professional, it is made through an encrypted system that meets all legal requirements.” [15]
- Visita virtual: It is an application designed during the pandemic that requires tablets or smartphones where family members have to choose the hospital, register and dial the extension of the room. It was launched in several Spanish hospitals such as Infanta Elena de Valdemoro Hospital, Villalba General Hospital, Rey Juan Carlos de Móstoles, La Moraleja University Hospital, Virgen del Mar or La Zarzuela University Hospital [16].

All these projects, born against the clock after the outbreak of the pandemic, depend for the most part on third-party infrastructure and hardware that today's hospitals are not prepared for. In addition, as a general rule, in healthcare environments (nursing homes, hospitals) are against using services such as Google Meet, Skype, Zoom ... as they use third-party servers, which means that the privacy of the data is put into question as they go through Google and can be stored in any way, therefore, they prefer (or give them more confidence) to use an own application.

This project arises with this motivation, in order to provide a robot with its own system that is capable of working by itself and without the need for any additional hardware by the patient, since the robot already incorporates a screen, while the family member with a simple smartphone can access the videocall via web browser.

Moreover, in the case of a strict policy in terms of security/privacy requirements, the system could be completely deployed in the health facilities IT infrastructures, since, as will be seen in the following sections, three private servers will be developed that will allow the implementation of this casuistry: Signaling Server, Web Server and Support Server (TURN/STUN).

Another aspect to be taken into account and which is of vital importance in the development of this project is the final end of the communication. This end user can be, as in our case, a robot, so it is necessary to take into account the specific policy to allow new participants in the videoconference, the time slots to be able to start it, the management of a historical record for its evaluation, etc. In other words, the interface on this side must be adapted to the requirement of the platform in aspects such as the audio/video interfaces to be used, the procedure to manage the start/end process of a video call and also the control of the signaling process allowing or not the access of a user to the video call system.

6 MATERIALS AND METHODS

This chapter relates the different technologies used during the development of this TFM and the reasons that have led to its choice. Next, the deployed infrastructure, the structure of the database (DB) and the distribution of the source code in classes, files and functions are described. Finally, a series of tests carried out on the deployed infrastructure are detailed.

Since this project is made up of many parts, figure 6-1 has been designed with the objective of synthesizing in a diagram the functional parts that make up the system and to show more clearly the behavior of each of the parts that make possible the development of this web application.

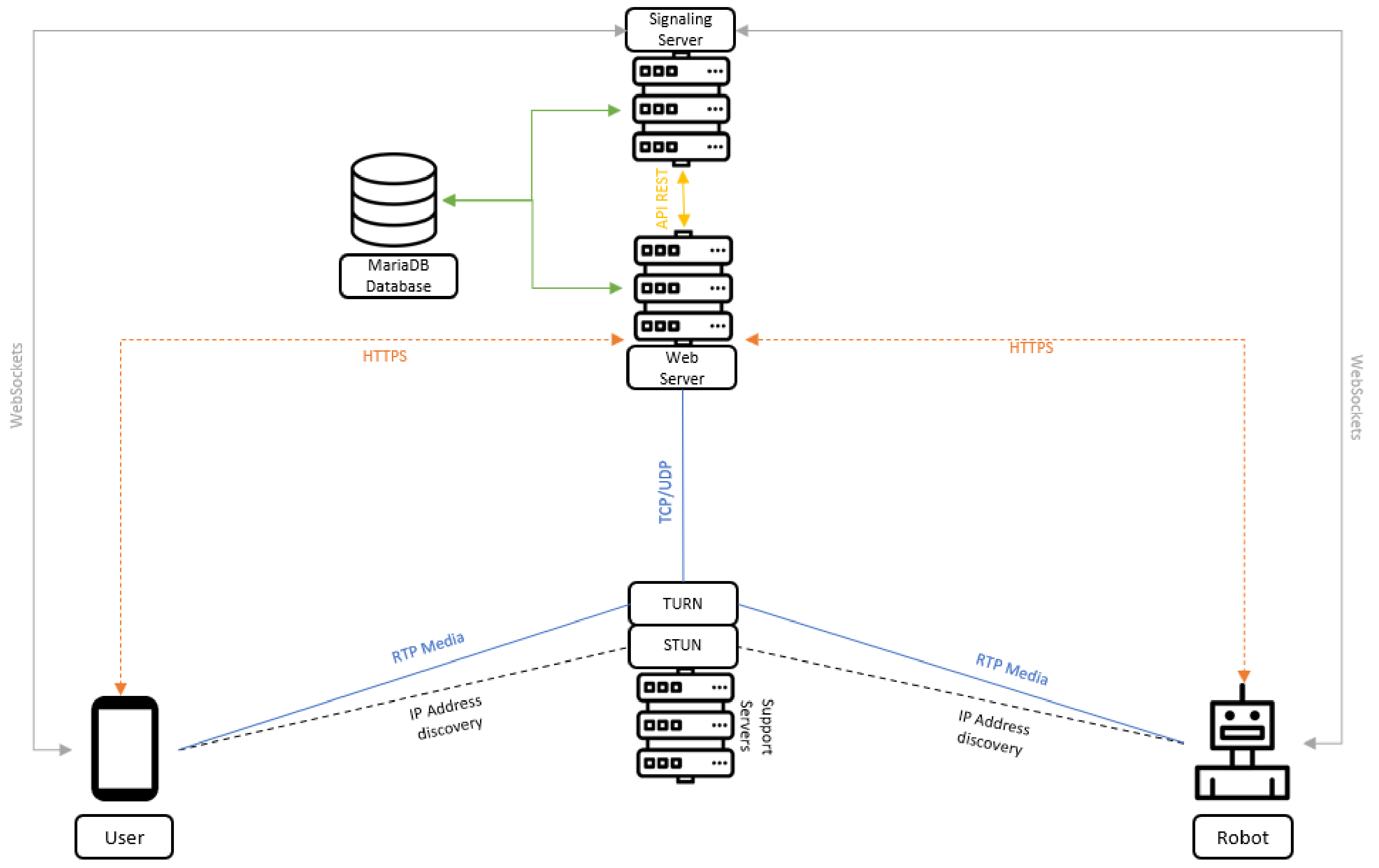


Figure 6-1. Diagram of the functional parts of the system

As a summary of this diagram, it is possible to comment on how each of the elements communicate and what they are used for.

The end users are represented by the mobile phone icon and the robot, they will be the participants of the video call.

The STUN server is part of the support servers for the discovery of the IP (IP Address discovery) of the endpoints, allowing a data flow between them directly, and in the event that STUN fails, the TURN server will come into operation, which it will create directly a tunnel and the multimedia data that is exchanged will pass through it, which will be in charge of distributing it to the end users (RTP Media).

The web server will be directly in charge of providing the different services to users through requests over HTTPS. The signaling server will make the video call possible and will be in charge of controlling the entry and exit of users of a video call. This communication where users inform the signaling server of the entry/exit of the video call, sending messages, etc. are in charge of the communication with WebSockets. These two servers supported by a database will be in charge of controlling web access and video calls. Finally, the communication between the web server and the back-end using API-REST.

6.1 Used technologies

Below, the main technologies and applications used during the implementation of the infrastructure are presented: architecture patterns, programming languages, frameworks, development environment, etc.

6.1.1 System architecture

The application has been designed following the Model – View – Controller (MVC) architecture pattern. This pattern was introduced by Trygve Reenskaug in Smalltalk-76 in the 1970s (1979) [17] and can be used in multiple frameworks. Its main virtue is the separation of the business logic from the user interface, which facilitates independent development of both parties and provides great flexibility to the developer.

6.1.2 Programming language

In this project we will use JavaScript as programming language.

JavaScript is an interpreted programming language, that is, programs do not need to be compiled to run. It is defined as object-oriented and despite its name, it is not directly related to the Java programming language. It is generally a language that is used on the client side although there are forms of JavaScript on the server side [18]. It appeared for the first time in 1995 by Brendan Einch under the name of LiveScript and is mainly used to create dynamic web pages [19].

6.1.3 Framework

Express has been used, a framework for the development of minimalist and flexible web applications with Nodejs [20]. Among other features, it offers route management (addressing), static files, use of a template engine, integration with databases, etc. among other.

6.1.4 Markup language

The different pages of the web application have been created using HTML5 pages and CSS style sheets.

A minimalist web design has been chosen, following the current trend of most web pages, choosing two main color tones for the web and several simple icons to represent the different functions.

The particularity for which the use of these CSS style sheets has been chosen is another feature of recent years in web page design, which is none other than adaptive design or “Responsive Design”, referring to the ability to adapt web content to the size of the screen of the device where it is being viewed.

Handlebars, a simple Javascript template system based on Mustache Templates, has been used as template engine. Handlebars is used to generate HTML from JSON formatted data objects.

Today, Handlebars is considered one of the most widely used templating systems in JavaScript. It allows you to create and format HTML code in a very simple way. Instead of doing tedious operations in libraries like jQuery to touch the DOM by inserting elements independently with "append" or "prepend", it allows you to create blocks of HTML code, written directly with HTML that you will populate with data coming from a JSON. It is as simple as writing HTML and so powerful that it allows you to perform structure traversal operations found in other templating systems you may have used [21].

6.1.5 Database management system

In order to ensure that the application is as light as possible and minimize its costs, it has been decided to choose a database management system that allows free use, both commercial and private. Among all the available options, MariaDB has been chosen, due to its ease of integration with Nodejs and its great data access speed.

6.1.6 Node.js

JavaScript, as a rule, has been a client-side programming language that runs in the browser. Node.js now offers the ability to run JavaScript on the server side. It's built on Chrome's V8 JavaScript engine, running your code at incredible speeds [22]. It is an event-driven and therefore asynchronous I/O (input/output) library and execution environment, and one of the many advantages it presents is sufficient consistency to create a large number of connections simultaneously with the server.

6.1.7 Development environment

When implementing the application in JavaScript under node.js, it is not necessary to use a specific IDE and a free text and source code editor has been used with support for various programming languages such as Notepad++ and Visual Studio Code.

6.1.8 Google Cloud Platform

Google Cloud Platform (GCP) is a set of modular cloud-based services that allow to create from simple websites to complex applications. In short, it is the solution offered by Google to provide cloud computing services.

Within this platform, this project uses Google Compute Engine (GCE), which it can be described as the platform on top of Google Cloud that allows users to launch virtual machines on demand [23] and that has been used for the deployment of the system in a cloud scenario (See Section 7.5).

6.2 WebRTC

WebRTC is becoming a very important standard since it allows videoconferences and, in general, data transmission directly through a Javascript API, which is why its use in web browsers is so important.

It is designed for real-time communications, as its name suggests: Web Real Time Communications.

The protocol was primarily promoted by Google in the IETF.

The first implementation is from Ericsson in 2011. Although browsers have come a long way in recent years, if the browser used by the end user is slightly recent, it will include the necessary API to be able to run applications under WebRTC.

Thanks to its success in browsers, it has been added to other types of applications such as WhatsApp, for example. The protocol is generic, so due to this success it is expected that its use will be extended to applications in other areas [24].

6.2.1 API WebRTC

The idea is to be able to use the browser to exchange data in real time, and in particular to do multimedia transmission. This transmission must be done between peers, instead of client-server oriented, to avoid delay problems that can occur in a real-time communication where you have to send the data to the server and wait for it to return it to the user.

We need peer-to-peer communication; we must allow endpoints to communicate with each other even though they are behind devices that are designed so that no one on another network can communicate with them. For this reason, we have used PeerJS library that simplifies WebRTC peer-to-peer data, video and audio calls. PeerJS wraps the browser's WebRTC implementation to provide a complete, configurable, and easy-to-use peer-to-peer connection API. Equipped with nothing but an ID, a peer can create a P2P data or media stream connection to a remote peer.

In the same direction of communications, it should be noted that websockets have been used to communicate between the clients and the signaling server. WebSockets is an advanced technology that makes it possible to open an interactive communication session between the user's browser and a server. With this API, you can send messages to a server and receive event-driven responses without having to query the server for a response [25].

On the other hand, the REST API, which relies on the Router middleware explained in section 7.1.4, to establish web access to the various associated services of the application.

Also, it includes security in order to avoid listening to the communication that is being established by an unauthorized third party. For this it uses SRTP and DTSL.

Finally, it solves the problem of universality since the only thing that is needed is a web browser.

6.3 WebRTC protocol stack

The function and characteristics of some of the main protocols used by WebRTC are described below.

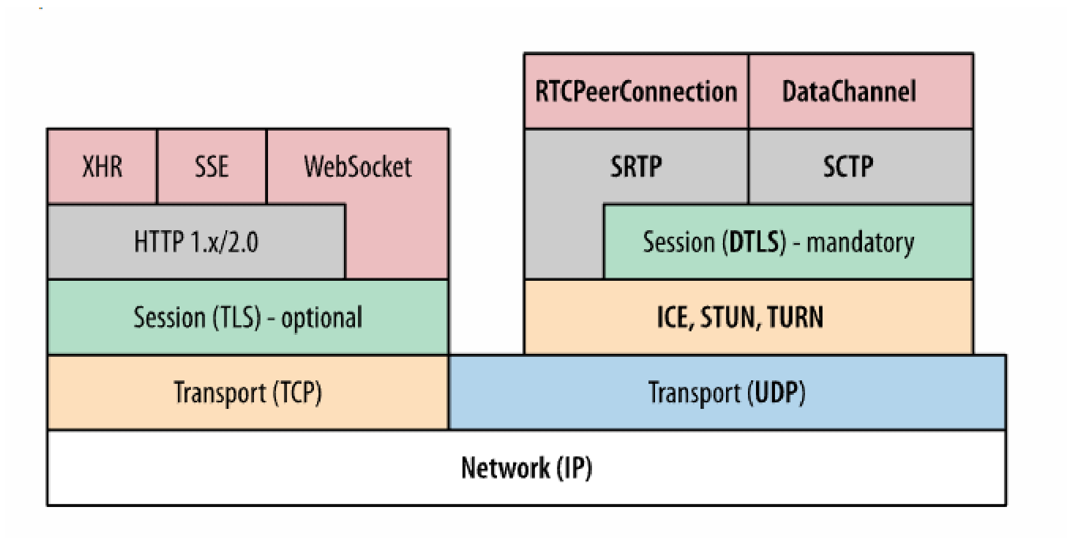


Figure 6-2. WebRTC protocol stack

Figure extracted from: https://hpbn.co/webrtc/

6.3.1 RTP

The Real Time Transport Protocol (RTP) is an application layer protocol designed to transport audio and video over IP networks [26]. This protocol is widely used in the communications and entertainment industry, some examples of use are multimedia applications such as Voice over IP (VoIP), Audio over IP (AoIP), WebRTC and Internet Protocol Television (IPTV).

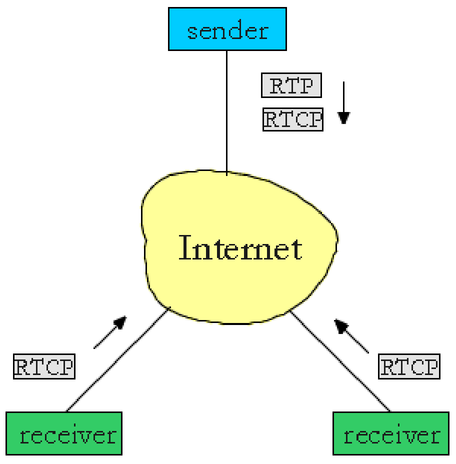


Figure 6-3. RTP operation

Figure extracted from: https://www.pbxdom.com/blog/engineering/how-rtp-real-time-transport-protocol-works-in-voips.

It is encapsulated in UDP datagrams. Although TCP can also be used, it is not recommended because this protocol prioritizes reliability over delivery time. It is used in conjunction with the Real Time Control Protocol (RTCP), while RTP is used to transport audio and video multimedia data, the second is responsible for tasks such as synchronization, collection of statistics and quality of service during the session.

Both RTP and RTCP have a secure variant called Secure Real-Time Transport Protocol (SRTP) and Secure Real-Time Control Protocol (SRTCP), which provide encryption, authentication, message integrity, and protection against replay attacks.

6.3.2 SDP

Session Description Protocol (SDP) is a protocol that is responsible for describing multimedia sessions in order to announce sessions, invite sessions and describe other forms of multimedia session initiation [27].

It is widely used in VoIP multimedia applications and video conferencing. This protocol does not deliver any multimedia data, rather it is used at the ends to negotiate session profiles, which are a set of properties such as network metrics, media types, etc.

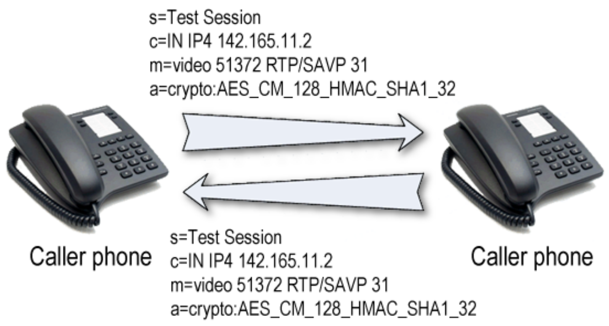

s=Test Session
c=IN IP4 142.165.11.2
m=video 51372 RTP/SAVP 31
a=crypto:AES_CM_128_HMAC_SHA1_32
Caller phone
s=Test Session
c=IN IP4 142.165.11.2
m=video 51372 RTP/SAVP 31
a=crypto:AES_CM_128_HMAC_SHA1_32
Caller phone

Figure 6-4. SDP operation

Figure extracted from: https://voip-sip-sdk.com/p_7233-session-description-protocol.html

This protocol is extensible so that new media formats can be included in it. It is a protocol that can act independently to describe multimedia sessions or together with other protocols such as RTP, RTSP or SIP.

6.3.3 ICE

The Interactive Connectivity Establishment (ICE) protocol is a protocol that allows traversal of Network Address Translation (NAT) in UDP multimedia sessions [28].

This technique makes it possible to find a way for two computers to communicate as directly as possible in a peer-to-peer (P2P) network. It is widely used in VoIP applications, peer-to-peer communications, instant messaging, etc.

This protocol allows communication to be done without going through a server. Its creation is a cause of the widespread use of network address translation, a technique that makes communication difficult in P2P networks.

While it's true that network translations helped slow IPv4 address exhaustion, they also added problems for certain use cases. An example of these drawbacks is that NAT breaks many IP applications and makes it difficult to deploy new applications. Although rules have been created for the creation of "NAT friendly" protocols, in many cases these rules cannot be applied, an example of these cases is multimedia applications or file sharing.

ICE allows peers to discover your public IP address in order to make use of protocols such as Session Initiation Protocol (SIP). In addition, it is in charge of choosing the best mechanism so that the peers can communicate, either using the STUN or TURN protocols.

6.3.4 STUN

Session Traversal Utilities for NAT (STUN) is a protocol that allows other protocols to deal with network address translation (NAT) [29].

This protocol can be used to determine the IP address and port assigned by NAT for a connection, determine the type of NAT it is in, check connectivity between 2 endpoints, and maintain associations established by NAT. It is often used as a complement to other protocols such as SIP.

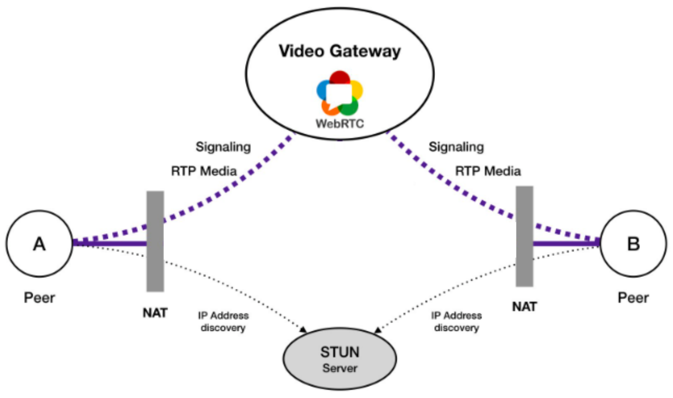


Figure 6-5. STUN operation

Figure extracted from: https://blog.ivrpowers.com/post/technologies/what-is-stun-turn-server/

STUN supports three of the four types of NAT, Full cone, Restricted cone and Port Restricted cone, however it does not support Symmetric NAT or bidirectional NAT. In a Full cone type NAT any end can start the connection while in the rest both ends must start the connection at the same time.

It should be noted that STUN does not provide a complete solution to go through NAT, since it does not work in scenarios with bidirectional NAT and in order to have such a solution it would be necessary for a client to be able to obtain an address from which it could receive data from any end that sends data to Internet. This can only be achieved by making use of servers that act as intermediaries, through which the data must pass. To solve this deficiency, the TURN protocol was defined.

6.3.5 TURN

Traversal Using Relays around NAT (TURN) is a protocol that allows a client located behind a NAT or a firewall to communicate directly with other clients using the service of an intermediate node that acts as an intermediary [30].

This protocol allows the client to control the operation of the intermediary node and the exchange of packets with the other end using its service. The main difference of this protocol with respect to other protocols that use intermediaries is that it allows the client to communicate with multiple ends using only one intermediary address (relay). It is particularly useful in scenarios where Symmetric NAT or bidirectional NAT exists.

Customer data will travel from the issuer to the TURN intermediary node, which will send it to the corresponding receiver. Although it is true that TURN is more robust than STUN, the latter is only used to discover the public IP of the client, so the multimedia communication does not go through any server, it is direct between the peers.

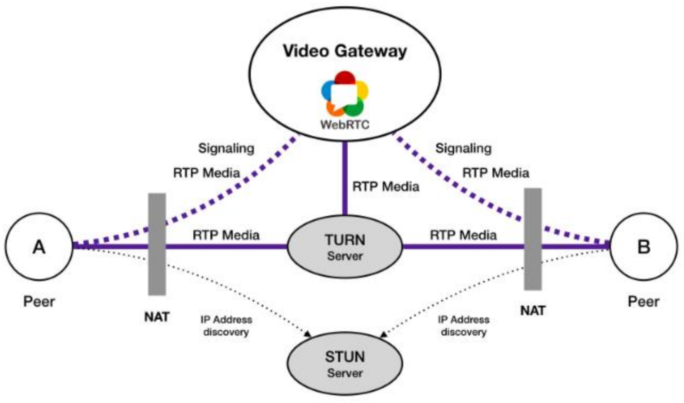


Figure 6-6. TURN operation

Figure extracted from: https://blog.ivrpowers.com/post/technologies/what-is-stun-turn-server/

This protocol was designed to support the ICE protocol, but can also be used without it.

6.4 Analysis and design

In the following sub-sections, an analysis of the designed web applications is carried out. To do this, first, the actors that can somehow interact with the web application are identified, the requirements that they must meet are defined, especially those related to the behavior of the system and the different scenarios of its use. Next step is to implement a solution described in terms of source code, database structure and scheme of the deployed infrastructure. Finally, the methodology used to carry out a test bench on this infrastructure is reviewed.

6.4.1 User identification

Depending on who accesses the web application, the following system users can be identified:

- End user (familiar member and robot), for whom the web application is intended, in which they can connect to the web application and access to make a video call.
- Administrator (specialist/doctor), whose role is to manage reservations in the system and register users or residents.

6.4.2 Definition of requirements

This section specifies the requirements that the implemented applications must meet, with respect to the services provided by the system (Functional Requirements), as well as its emergent properties (non-functional requirements).

6.4.2.1 Functional requirements

- FR1. Possibility of making a multi-user video call.
- FR2. Possibility of making several independent video calls at the same time.
- FR3. Possibility of managing video calls, residents and users registered in the system.
- FR4. Service that is capable of consulting the obligation or not to notify the family member of an upcoming video call.
- FR5. Sending email with the connection information by the server to the family member if necessary.
- FR6. Automatic disconnection closure of the last person left in the room once the video call has been made.
- FR7. Service capable of closing all the connections in progress of a video call and canceling it.
- FR8. Web application under JavaScript.

6.4.2.2 Non-Functional requirements

- NFR1. Whenever possible, free software should be used to minimize the costs associated with the project.
- NFR2. The graphical interface should be as friendly as possible.
- NFR3. The application must be developed following the established rules and standards in order to make it easier for future developers to understand.

6.4.3 Use cases

The different cases of use identified for the web application destined to carry out this TFM are those that are listed below and are shown in the context diagram of Figure 6.6.

- **Supervisor's Management**: Register new administrator users of the platform (specialists or doctors) in order to access the system as well as generate their own video call reservations for their patients with their family members.
- **Resident's Management:** Register new residents on the platform, in order to have a personal record of them, and one of the most important aspects of the application, to be able to link their familiar members through an email address. This email address will be used to notify the specific link of the room that will be used to make the video call a few minutes before it is made. It also allows the editing and modification of some data that have been saved.
- **Reservation's Management:** Schedule a new video call between patients, family members and specialists/doctors. It also allows the editing and modification of some data that have been saved.
- **Enter in a videocall:** Enter the reserved room to make the video call.
  Family members will receive the link by email and by clicking on the link they will be redirected directly to the room.
  The specialist or doctor can access in three different ways:
  1. From the link sent by email
  2. From the reservation link where you can see all the reservations you have
  3. From the text box that you have in your profile where you have to indicate the token of the room.

An additional use case has been implemented, so that the web application adapts as best as possible to the demands of this project. In this use case, a service is required, in this case hidden in the interface, which allows the general closure of any video call and its resources at a given time.

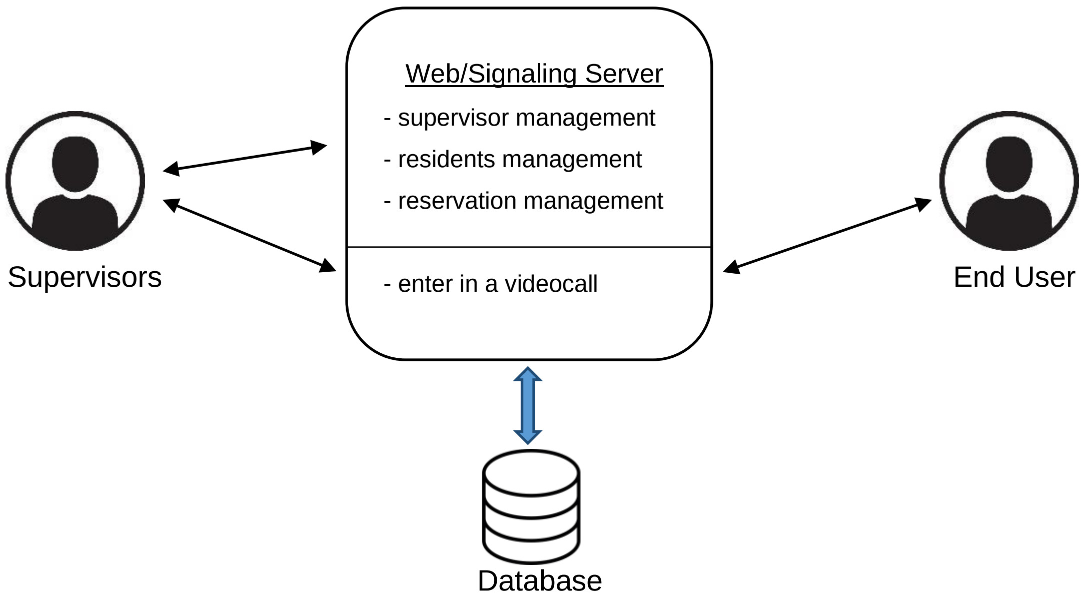


Figure 6-7. Use case context diagram

The following sub-sections establish the flow of information that occurs when a user/administrator uses the web application for the uses shown in the context diagram of Figure 6.6.

6.4.3.1 Supervisor management

Flow of information that is presented when a supervisor uses the VideoConf web application to register a new administrator user (member of the medical staff, nurse or doctor).

Main Actor: Supervisor
Preconditions: Login
Basic Flow:

1. The supervisor accesses the application through the web interface.
2. The supervisor accesses the user registration section.
3. The system requests the data of the new administrator, which are mandatory: username, password, name and email.
4. The web application returns the profile view of the administrator with a pop-up on the screen indicating that the registration has been carried out successfully.

Alternative Flow 1:

1. The supervisor accesses the application through the web interface.
2. The supervisor accesses the user registration section.
3. The system requests the data of the new administrator, which are mandatory: username, password, name and email.
4. The web application returns the view of the administrator's record indicating that some data is missing to complete.

Alternative Flow 2:

1. The supervisor accesses the application through the web interface.
2. The supervisor accesses the user registration section.
3. The system requests the data of the new administrator, which are mandatory: username, password, name and email.
4. The web application returns the view of the administrator's record indicating that username must be unique.

6.4.3.2 Residents management

Flow of information that is presented when an Administrator uses the VideoConf web application to register a new resident at the system.

Main Actor: Administrator

Preconditions: Login

Basic Flow 1:

1. The user accesses the application through the web interface.
2. The user accesses the resident registration section.
3. The system requests the data of the new resident, which are mandatory: name, room, age, password, familiar name, and familiar email.
4. The web application returns the view of the list with all the residents together with the data that is stored in the database and a pop-up on the screen indicating that the registration has been carried out correctly.

Basic Flow 2:

1. The user accesses the application through the web interface.
2. The user accesses the residents list section.
3. The web application returns the view of the list with all the residents together with the data that is stored in the database.

Alternative Flow:

1. The user accesses the application through the web interface.
2. The user accesses the resident registration section.

3. The system requests the data of the new user, which are mandatory: username, password, name and email.
4. The web application returns the view of the resident record indicating that some data is missing to complete.

6.4.3.3 Reservations management

Flow of information that is presented when a user uses the VideoConf web application to schedule a new videocall reservation.

Main Actor: Administrator

Preconditions: Login

Basic Flow 1:

1. The user accesses the application through the web interface.
2. The user accesses the videocall schedule section.
3. The system requests the data of the new reservation, which are mandatory: Patient's name, Familiar's email (it's completed automatically when selecting the patient), Description and Date-Time.
4. The web application returns the view of the list with all the scheduled videocalls together with the information of each of them and a pop-up on the screen indicating that the videocall has been registered correctly.

Basic Flow 2:

1. The user accesses the application through the web interface.
2. The user accesses the videocall schedule section.
3. The web application returns the view of the list with all the scheduled videocalls together with the information of each of them

Alternative Flow:

1. The user accesses the application through the web interface.
2. The user accesses the videocall schedule section.
3. The system requests the data of the new reservation, which are mandatory: Patient's name, Familiar's email (it's completed automatically when selecting the patient), Description and Date-Time.
4. The web application returns the view of the videocall schedule section indicating that some data is missing to complete.

6.4.3.4 Enter in a videocall

Flow of information that is presented when a user uses the VideoConf web application to enter in a videocall

Main Actor: End User

Preconditions: None

Basic Flow:

1. The user accesses the application through the web interface.
2. The user accesses the videocall section with the room created previously.
3. The system asks for access to the camera and microphone the first time and the videocall is initiated.
4. The system automatically redirects to the home page.

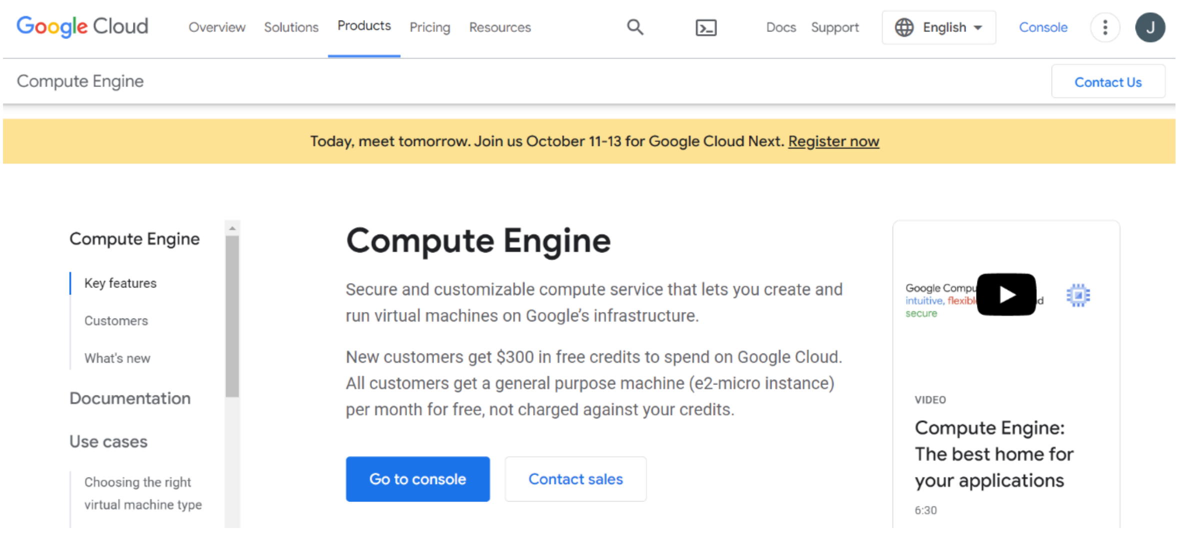
Alternative Flow:

1. The user accesses the application through the web interface.
2. The user accesses the videocall section with the room created previously.
3. The system automatically redirects to the home page with a pop-up alert on the screen indicating that the room ID is invalid.

6.5 Cloud server

As already mentioned in this work, Google Cloud Platform has been used for its development and use in a realistic environment.

Within this platform, this project uses Google Compute Engine (GCE), specifically, a VM (Virtual Machine) instance has been used. Virtual machines in Google Cloud Platform are highly configurable to run workloads on Google's infrastructure.



Today, meet tomorrow. Join us October 11-13 for Google Cloud Next. Register now

Compute Engine

Secure and customizable compute service that lets you create and run virtual machines on Google's infrastructure.

New customers get $300 in free credits to spend on Google Cloud. All customers get a general purpose machine (e2-micro instance) per month for free, not charged against your credits.

Compute Engine: The best home for your applications

Figure 6-8. Google Compute Engine

The choice to use GCE is given by several advantages after analyzing the different possibilities that were available to carry out and deploy this project.

These advantages are:

- More competitive prices compared to the competition. With a not very high cost you can set up a server without physically depending on any type of hardware.
- High level of security and data protection.
- Real-Time Data migration to virtual machines.
- Platform dedicated to expansion and scalability.

Google defines GCE as “secure and customizable compute service that lets you create and run virtual machines on Google’s infrastructure”.

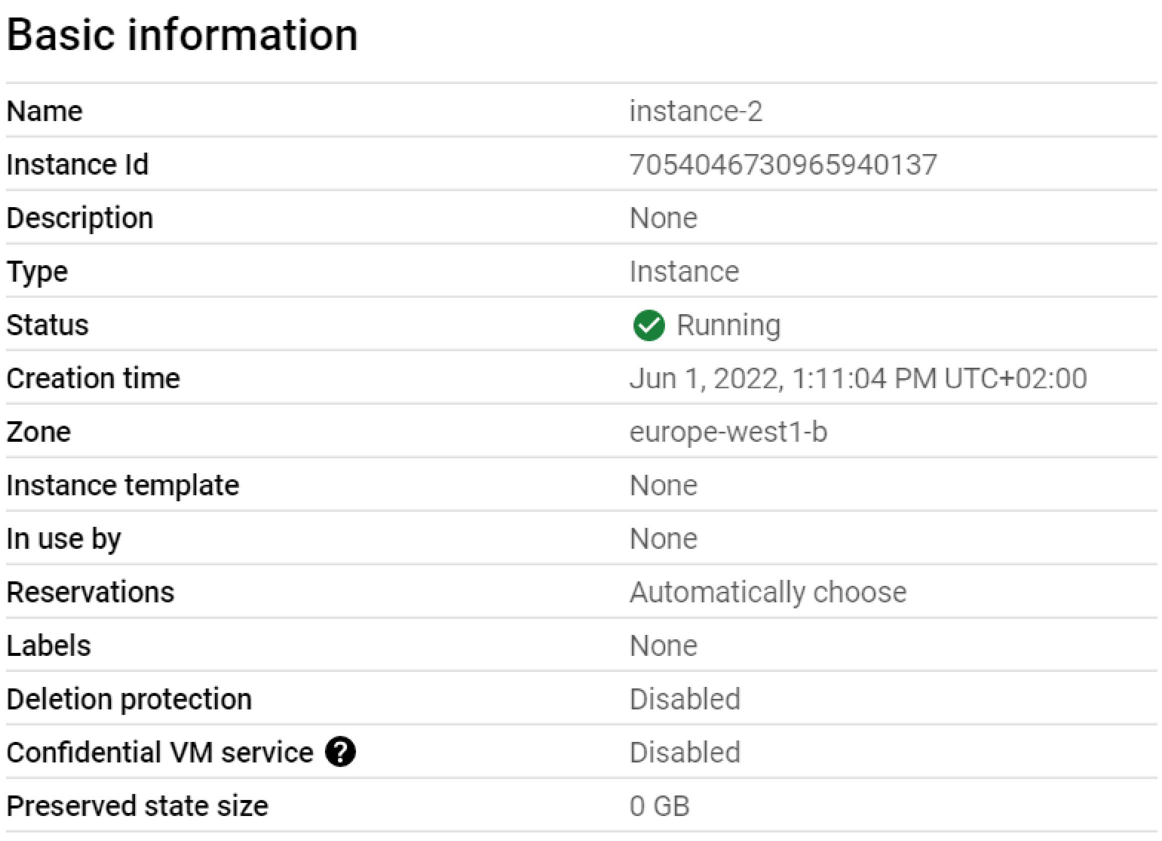
New customers get $300 in free credits to spend on Google Cloud. All customers get a general-purpose machine (e2-micro instance) per month for free, not charged against your credits.

It comes with different types of options. It has the possibility of creating machines with predefined parameters, it means, that there are ready-to-use machine models without the need to configure any parameters or features. Therefore, it also has a custom configuration parameter where the user can specify what type of resources they need, which allows us to balance the costs of our project. In our case, it is the option that has been used, mainly because the application to be developed does not require an excessive amount of resources and also allows us to keep the cost insignificant.

These virtual machines are themselves very affordable compute instances, ideal for batch tasks and fault-tolerant workloads.

Another important feature of these machines is what Google calls Confidential Computing, that is, it encrypts your most sensitive data while it is being processed. As a last point to note, these machines have an automatic resource tracking system, so if it detects that the virtual machine needs a reconfiguration, it notifies the user with that recommendation.

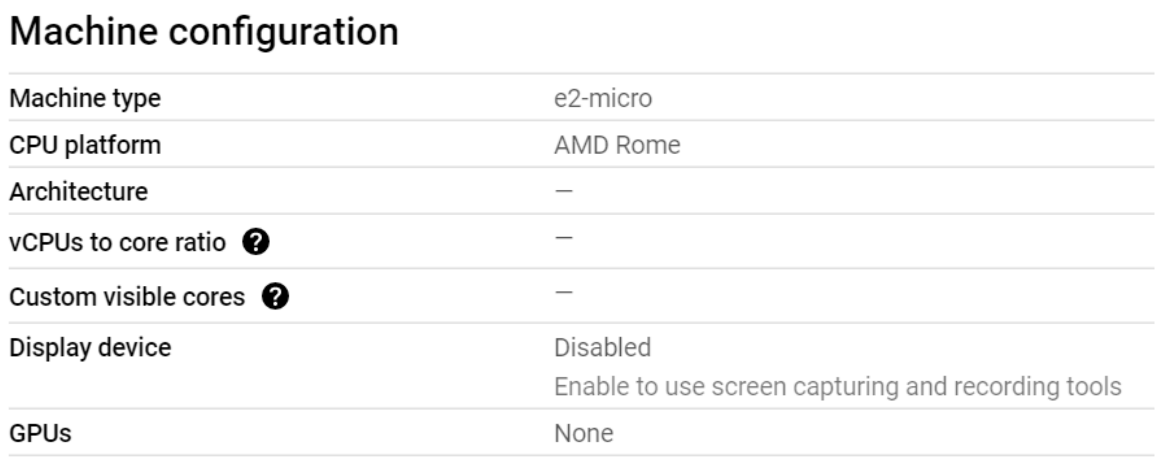
6.5.1 Characteristics of the chosen server



Basic information

Name	instance-2
Instance Id	7054046730965940137
Description	None
Type	Instance
Status	✓ Running
Creation time	Jun 1, 2022, 1:11:04 PM UTC+02:00
Zone	europe-west1-b
Instance template	None
In use by	None
Reservations	Automatically choose
Labels	None
Deletion protection	Disabled
Confidential VM service	Disabled
Preserved state size	0 GB

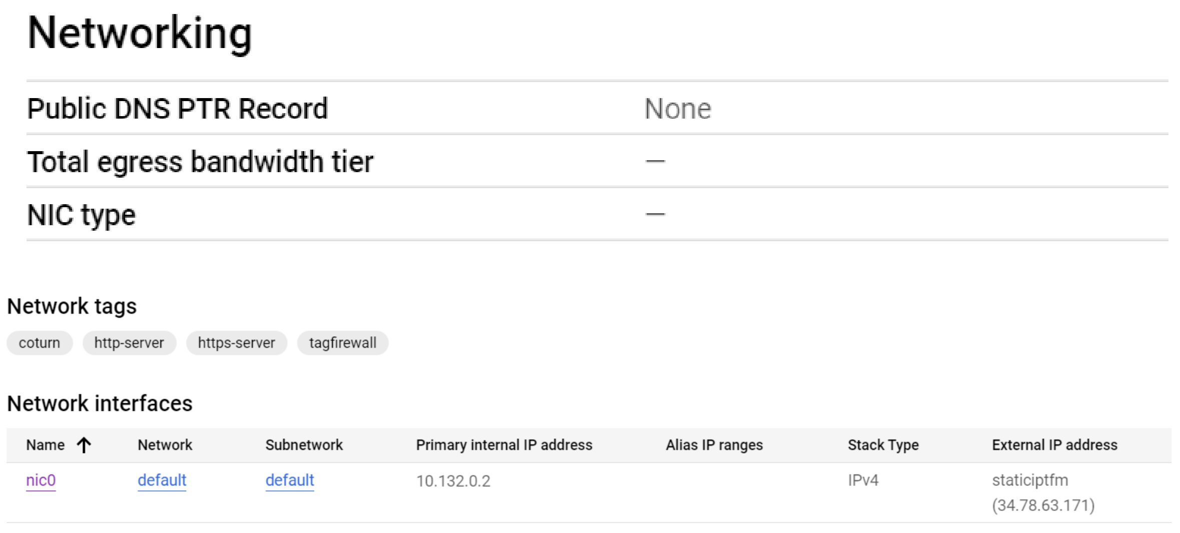
Table 6-1. Server basic information



Machine configuration

Machine type	e2-micro
CPU platform	AMD Rome
Architecture	—
vCPUs to core ratio	—
Custom visible cores	—
Display device	Disabled Enable to use screen capturing and recording tools
GPUs	None

Table 6-2. Server machine configuration



Networking

Public DNS PTR Record	None
Total egress bandwidth tier	—
NIC type	—

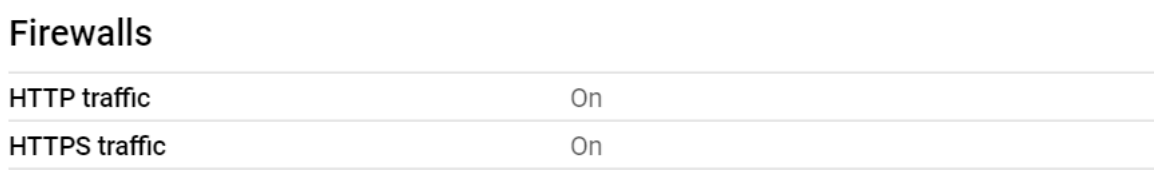
Network tags

coturn http-server https-server tagfirewall

Network interfaces

Name ↑	Network	Subnetwork	Primary internal IP address	Alias IP ranges	Stack Type	External IP address
nic0	default	default	10.132.0.2		IPv4	staticiptfm (34.78.63.171)

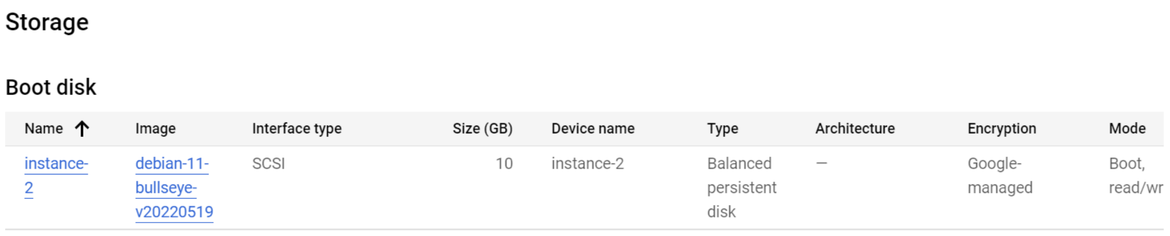
Table 6-3. Server networking



Firewalls

HTTP traffic	On
HTTPS traffic	On

Tabla 6-4. Server firewalls

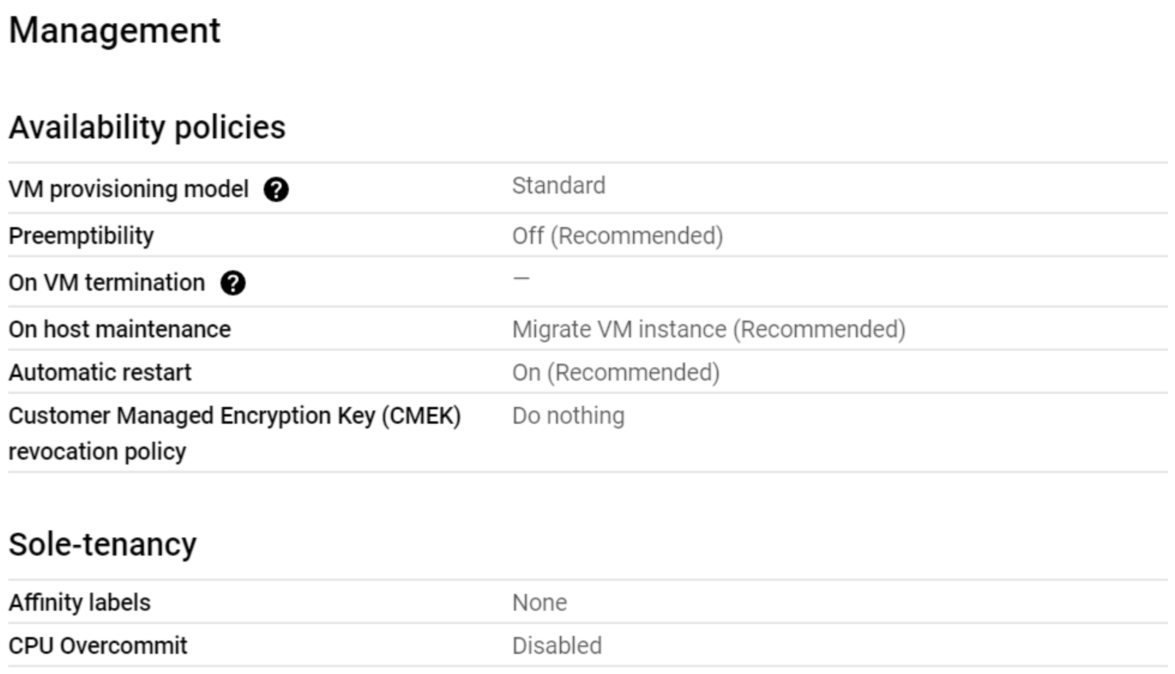


Storage

Boot disk

Name ↑	Image	Interface type	Size (GB)	Device name	Type	Architecture	Encryption	Mode
instance-2	debian-11-bullseye-v20220519	SCSI	10	instance-2	Balanced persistent disk	—	Google-managed	Boot, read/wr

Tabla 6-5. Server storage



Management

Availability policies

VM provisioning model ?	Standard
Preemptibility	Off (Recommended)
On VM termination ?	—
On host maintenance	Migrate VM instance (Recommended)
Automatic restart	On (Recommended)
Customer Managed Encryption Key (CMEK) revocation policy	Do nothing

Sole-tenancy

Affinity labels	None
CPU Overcommit	Disabled

Tabla 6-6. Server management

7 DEVELOPMENT

This chapter describes the process of designing the scalability of the videoconferencing service, its implementation and problems encountered throughout the life cycle.

We start from an application that was used in the practices of the Advanced Multimedia Services subject course developed in Javascript with React framework that allow making a video call through a browser between just two users in a local network. Summarizing, it was a peer-to-peer video and audio web application for a local network.

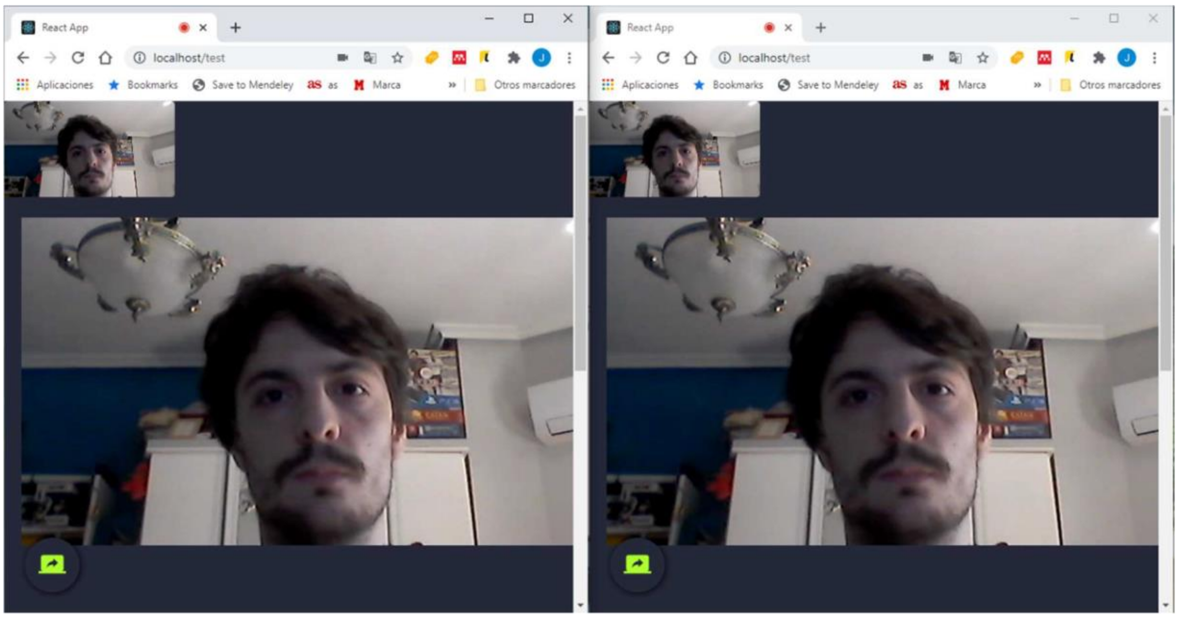


Figure 7-1. Screenshot of the starting application

Using as a starting point this application, the final system was designed and programmed to fulfill all the requirements described in section 6.4.2. However, new requirements such as multiparty videoconference, the inclusion of chat capabilities, the need of an administrator/supervisor portal for managing and scheduling sessions lead to the development of a completely new application, thus by using the initial one just for learning support.

List of features included in the application:

- User logging and user session maintenance.
- Registration of administrator users in the system
- Resident registration and listing of residents.
- Registration and validation of a video call and its management. Listing of all of them.
- Mute/unmute audio in video calls.
- Turn on/off the camera in video calls.

- Invite another user to the video call.
- Text chat
- Choice of audio and video source.

7.1 VideoConf web application design classes model

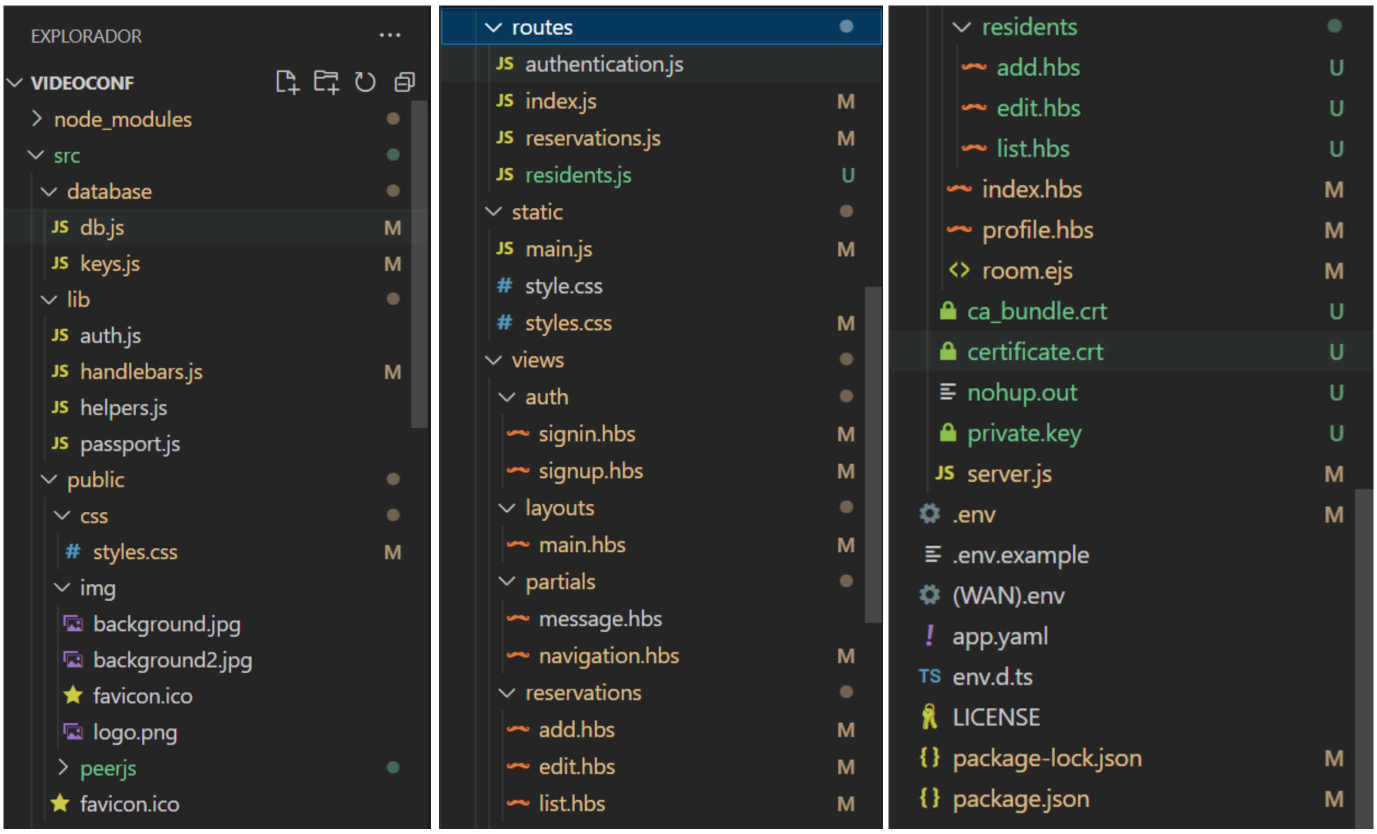
The VideoConf web application has been designed following the client-server paradigm. The server will offer a Web Interface for access through a browser and consists of two fundamental parts, the front-end and back-end, which integrate two functionalities: the call manager/reservation manager, and the access view to the videoconferencing system by creating a multipurpose room managed through communication with a signaling server.

- Back-end: It refers to all the services on which the front-end is based. It is the part in charge of everything working correctly and it uses the JavaScript programming language and the MariaDB DBMS. This is the part that implements access to services through a REST API interface.
- Front-end: Refers to the web interface and the different graphic components that will be displayed visually. It is in this layer where HTML pages, CSS style sheets, JavaScript functions and views based on HBS are used.

The video call is possible between the ends because this communication is done by using the WebRTC Peer.js API, already described in section 6.2 and discussed in more depth in this section. The establishment of this communication is done with the signaling server. The server will be in charge of controlling the events that may occur in the video call through the use of websockets: the start or close of the video call, the incorporation or exit of users or the sending of messages through the chat.

The web application follows the MVC model and is made up of a series of JavaScript files, JSON, and HBS/EJS views, and has been designed in such a way as to maximize system flexibility and facilitate component reuse and replacement.

The structure of the project is the one shown in Figure 7-2. In the following subsections, the different files and classes that compose it will be discussed.



```
EXPLORADOR
VIDEOCONF
  node_modules
  src
    database
      db.js                M
      keys.js              M
    lib
      auth.js
      handlebars.js        M
      helpers.js
      passport.js
    public
      css
        styles.css         M
      img
        background.jpg
        background2.jpg
        favicon.ico
        logo.png
      peerjs
      favicon.ico
```

```
    routes
      authentication.js
      index.js             M
      reservations.js      M
      residents.js         U
    static
      main.js              M
      style.css
      styles.css           M
    views
      auth
        signin.hbs         M
        signup.hbs         M
      layouts
        main.hbs           M
      partials
        message.hbs
        navigation.hbs     M
      reservations
        add.hbs            M
        edit.hbs           M
        list.hbs           M
```

```
      residents
        add.hbs            U
        edit.hbs           U
        list.hbs           U
      index.hbs            M
      profile.hbs          M
      room.ejs             M
    ca_bundle.crt          U
    certificate.crt        U
    nohup.out              U
    private.key            U
    server.js              M
  .env                     M
  .env.example
  (WAN).env
  app.yaml
  env.d.ts
  LICENSE
  package-lock.json        M
  package.json             M
```

Figure 7-2. Structure of the VideoConf web application

7.1.1 System files

To implement the web application, twelve JavaScript files, thirteen HBS views and one EJS have been created. Next, each of these files is listed, the classes it contains and the function of each of them are indicated.

7.1.2 Patient and call management system.

7.1.2.1 Front-End

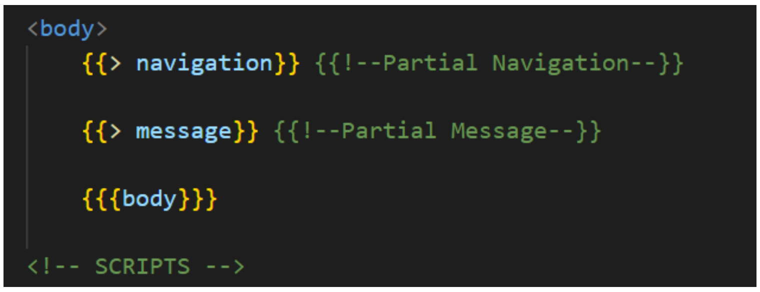
The front-end of our application refers to the static part based on a single page web application that uses the handlersbars framework to dynamically generate information.

All this information that is generated about our main view (main.hbs) is inside the folder that is called "views" in the project. This folder contains all the handlebar views that make up the application. Handlebars, as mentioned in section 6.1.4.

These are the fundamental parts of each of the HBS files:

- **auth/signin.hbs:** HTML view file supported on HBS that contains several div elements to give it a friendly visual format with the help of boostrap. In each div element a different class defined in bootstrap is used. Contains a form defined with the POST method that is used for user login.

- **auth/signup.hbs:** HTML view file supported on HBS that contains several div elements to give it a friendly visual format with the help of boostrap. In each div element a different class defined in bootstrap is used. Contains a form defined with the POST method that is used for user registration.
- **layouts/main.hbs:** HTML view file supported on HBS that contains the main view of the web application. It can be seen that it has a very basic HTML code, this is due to the fact that handlebars allow us to modulate all the views and index a fragment of HBS code that we want in parts. As we can see in the figure 7-3, two types of partials and the body itself have been implemented. Handlebars allows to reuse a template through partials. Partials are normal Handlebars templates that may be called directly by other templates. Calling the partial is done through the partial call syntax: “{{> myPartial }}”. For the case of the body, since we need the use of pure HTML, we use the expression of triple keys “{{{}}}”.



```
<body>
    {{> navigation}} {{!--Partial Navigation--}}

    {{> message}} {{!--Partial Message--}}

    {{{body}}}

<!-- SCRIPTS -->
```

Figure 7-3. HBS partial call syntax

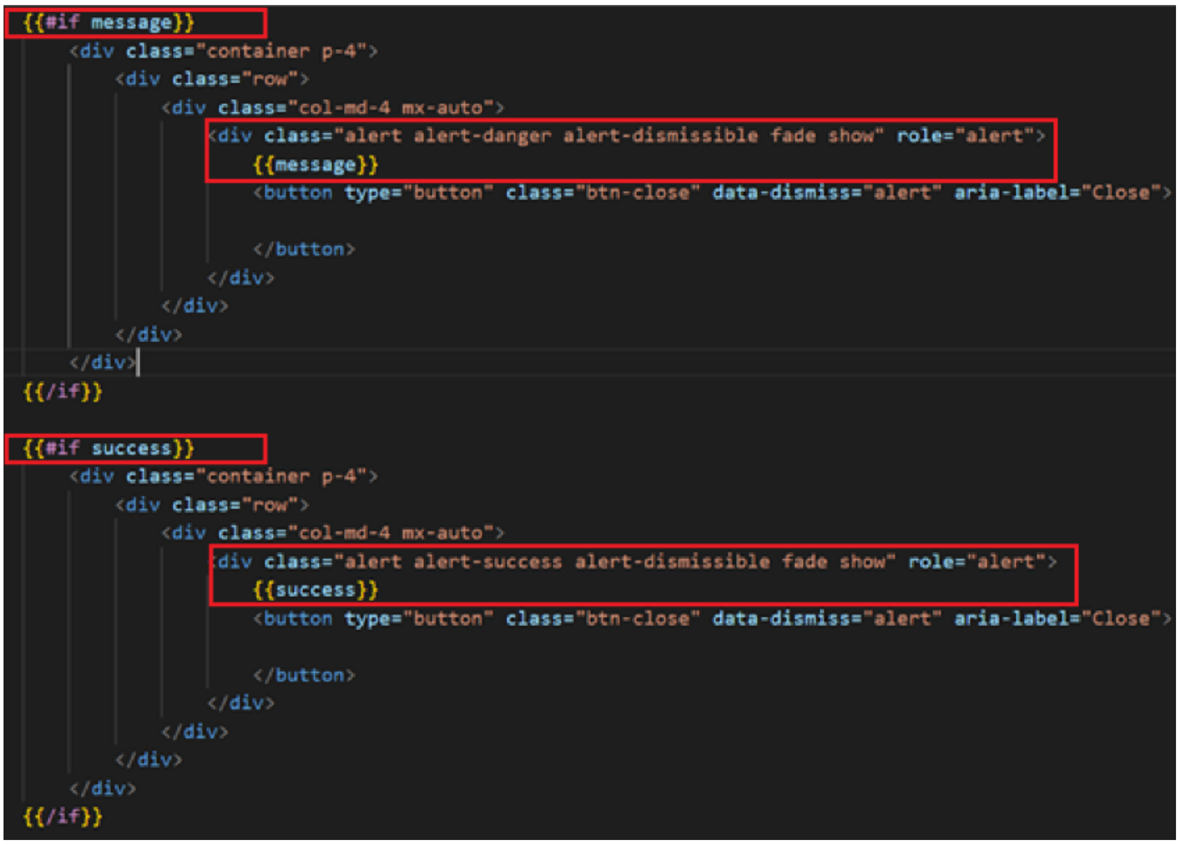
In addition, several functions have been implemented:

- **enablePath():** Function that enables and disables the email text box when scheduling a video call. This makes the system dependent on the patient that is selected so as not to mislead. This function is needed because if it is disabled in the select, the form does not send the data of the disabled fields.
- **getEmails():** Function that autocompletes the email in the text field of the familiar member of patient.
- **alertDelete():** Function to notify the user that a scheduled video call or a resident is going to be deleted. This way we prevent any data from being deleted by an erroneous click.

- **partials/message.hbs:** The versatility of handlebars allows us to execute code sentences and, based on the result, display one HTML code or another. Two types of pop-up have been defined in this file where, depending on the type of message received, one type of notice or another is displayed (Figure 7-4).

  The operation is as follows:

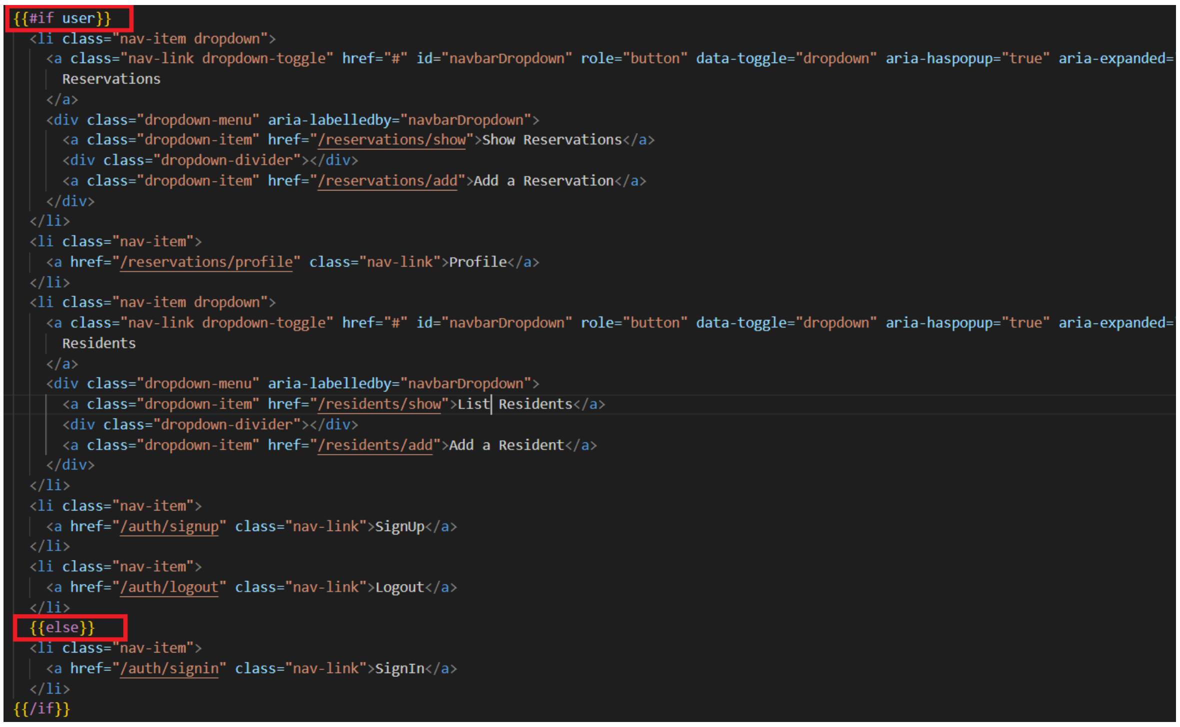
  If it receives "message" it shows an alert pop-up and, in the expression, {{message}} we index the message we want to report.If "success" is received, it shows a success pop-up and in {{success}} we index the message we want to report.



```
{{#if message}}
    <div class="container p-4">
        <div class="row">
            <div class="col-md-4 mx-auto">
                <div class="alert alert-danger alert-dismissible fade show" role="alert">
                    {{message}}
                    <button type="button" class="btn-close" data-dismiss="alert" aria-label="Close">

                    </button>
                </div>
            </div>
        </div>
    </div>
{{/if}}

{{#if success}}
    <div class="container p-4">
        <div class="row">
            <div class="col-md-4 mx-auto">
                <div class="alert alert-success alert-dismissible fade show" role="alert">
                    {{success}}
                    <button type="button" class="btn-close" data-dismiss="alert" aria-label="Close">

                    </button>
                </div>
            </div>
        </div>
    </div>
{{/if}}
```

Figure 7-4. Conditionals at HBS file

- **partials/navigation.hbs:** It is the file in charge of displaying the navigation bar. Following the same logic as the previous one, we make the navigation bar show some options or others depending on whether the user is logged in or not (Figure 7-5).

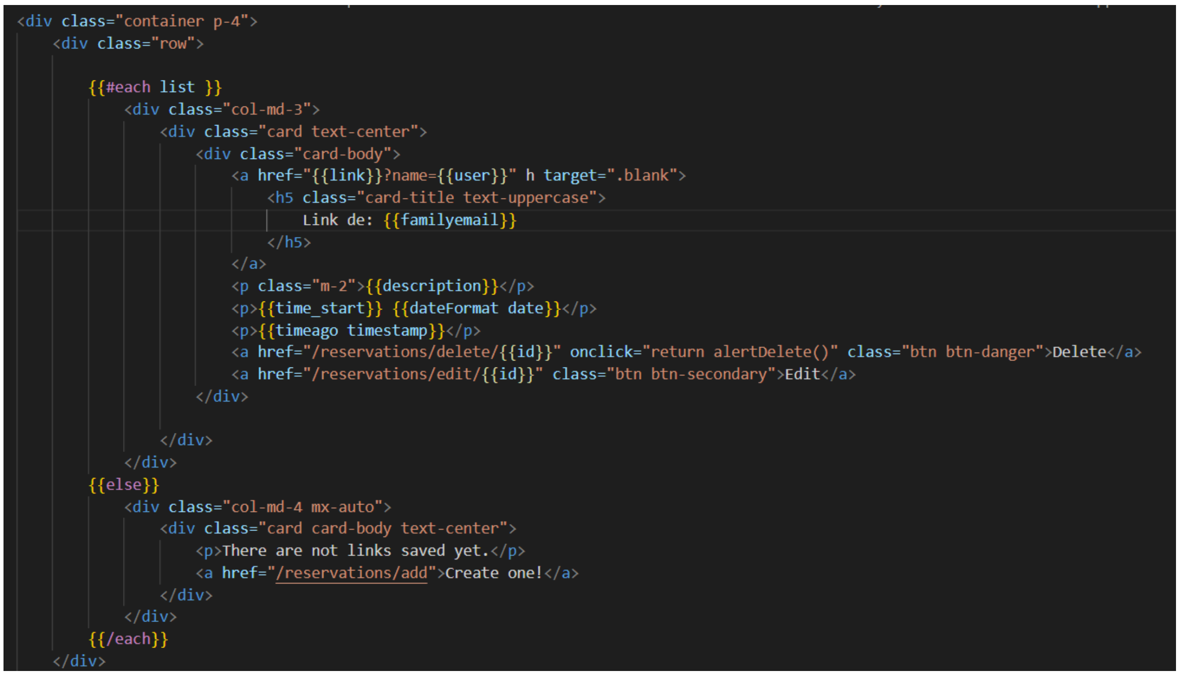


```
{{#if user}}
  <li class="nav-item dropdown">
    <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-toggle="dropdown" aria-haspopup="true" aria-expanded=
      Reservations
    </a>
    <div class="dropdown-menu" aria-labelledby="navbarDropdown">
      <a class="dropdown-item" href="/reservations/show">Show Reservations</a>
      <div class="dropdown-divider"></div>
      <a class="dropdown-item" href="/reservations/add">Add a Reservation</a>
    </div>
  </li>
  <li class="nav-item">
    <a href="/reservations/profile" class="nav-link">Profile</a>
  </li>
  <li class="nav-item dropdown">
    <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-toggle="dropdown" aria-haspopup="true" aria-expanded=
      Residents
    </a>
    <div class="dropdown-menu" aria-labelledby="navbarDropdown">
      <a class="dropdown-item" href="/residents/show">List Residents</a>
      <div class="dropdown-divider"></div>
      <a class="dropdown-item" href="/residents/add">Add a Resident</a>
    </div>
  </li>
  <li class="nav-item">
    <a href="/auth/signup" class="nav-link">SignUp</a>
  </li>
  <li class="nav-item">
    <a href="/auth/logout" class="nav-link">Logout</a>
  </li>
  {{else}}
  <li class="nav-item">
    <a href="/auth/signin" class="nav-link">SignIn</a>
  </li>
{{/if}}
```

Figure 7-5. Conditionals at HBS file

- **reservations/add.hbs:** HTML view file supported on HBS that contains several divs to give it a friendly visual format with the help of boostrap. In each div a different class defined in bootstrap is used. Contains a form defined with the POST method that is used for schedule a video call.
- **reservations/edit.hbs:** HTML view file supported on HBS that contains several divs to give it a friendly visual format with the help of boostrap. In each div a different class defined in bootstrap is used. Contains a form defined with the POST method that is used for edit a schedule video call.
- **reservations/list.hbs:** HBS compliant HTML view file containing several divs to give it a friendly visual format with the help of boostrap. In each div a different class defined in bootstrap is used. This view goes through all the scheduled video calls and shows them on the screen to the user.

Handlebars in this context makes the implementation much easier since just by indicating {{#each list}} it goes through all the objects in the list and shows them as indicated for each one of them (Figure 7.6).



```
<div class="container p-4">
    <div class="row">

        {{#each list }}
            <div class="col-md-3">
                <div class="card text-center">
                    <div class="card-body">
                        <a href="{{link}}?name={{user}}" h target=".blank">
                            <h5 class="card-title text-uppercase">
                                Link de: {{familyemail}}
                            </h5>
                        </a>
                        <p class="m-2">{{description}}</p>
                        <p>{{time_start}} {{dateFormat date}}</p>
                        <p>{{timeago timestamp}}</p>
                        <a href="/reservations/delete/{{id}}" onclick="return alertDelete()" class="btn btn-danger">Delete</a>
                        <a href="/reservations/edit/{{id}}" class="btn btn-secondary">Edit</a>
                    </div>

                </div>
            </div>
        {{else}}
            <div class="col-md-4 mx-auto">
                <div class="card card-body text-center">
                    <p>There are not links saved yet.</p>
                    <a href="/reservations/add">Create one!</a>
                </div>
            </div>
        {{/each}}
    </div>
```

Figure 7-6. "each" syntax at HBS file

- **residents/add.hbs:** HTML view file supported on HBS that contains several divs to give it a friendly visual format with the help of boostrap. In each div a different class defined in bootstrap is used. Contains a form defined with the POST method that is used for resident registration.
- **residents/edit.hbs:** HTML view file supported on HBS that contains several divs to give it a friendly visual format with the help of boostrap. In each div a different class defined in bootstrap is used. Contains a form defined with the POST method that is used for edit a resident registry.
- **residents/list.hbs:** HBS compliant HTML view file containing several divs to give it a friendly visual format with the help of boostrap. In each div a different class defined in bootstrap is used. This view loops through all the residents and displays them on the screen to the user.
- **index.hbs:** Application home page. It only contains a title along with a button.
- **profile.hbs:** View showing the user profile page. It is made up of a first part where it shows us the data of the logged in user and a second that has a form that is executed through POST with which the user can access a video call by filling in the form data (name and meeting ID).

- **room.ejs:** View where the video call is made. There is nothing to highlight because the behavior of video calls is not defined here as has already been seen. All the divs are defined and the placement of each of the parts of the application such as buttons, chat, video... each button that appears calls the functions described in section 7.2.7.

These views need a public folder to which they can access where the files are hosted. In the public directoriy, we can find the css style sheet files that give the visual appearance to the web application, the background used by the application and its web logo.

7.1.2.2 Back-End

The back-end of the patient and call management system is mainly formed by the controller and the communication templates with the database.

Four very similar .js files have been implemented that make use of the react-router-dom module to do the controller function, therefore, they will process GET and POST requests received by the web server.

In the following lines we will detail the requests they control and we will describe in detail some of the most representative and important ones, commenting on their behavior and connection with the DB.

- **authentication.js**

Contains functions to process the GET and POST requests for registration, login, and logout.

- **index.js**

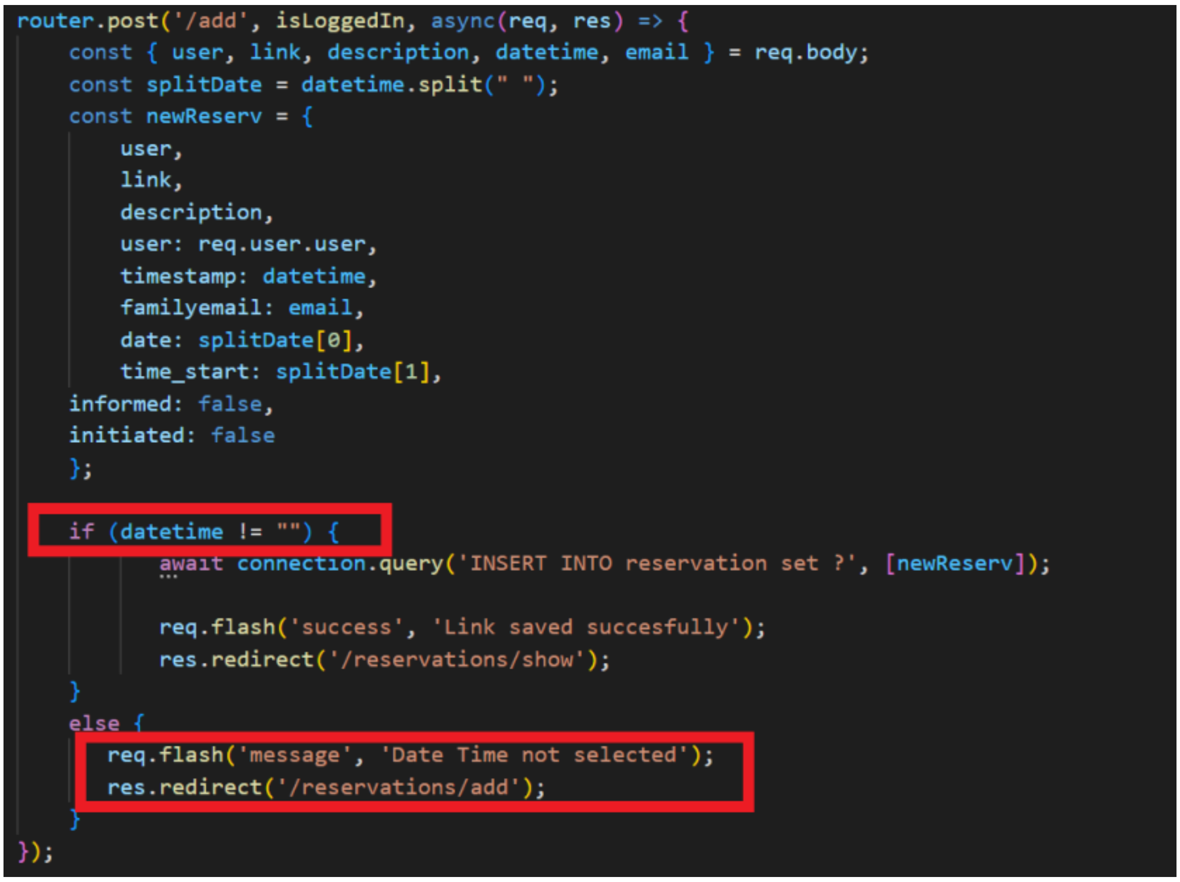
It contains the functions to process the requests:

- GET request to enter the homepage.

- **reservations.js**

It contains the functions to process the requests:

- GET request to delete a reservation and show the list of them.
- GET and POST to add or edit a scheduled reservation. It should be noted that the form has a special field where a calendar with a clock is displayed to select the date and time, for this reason, we cannot configure this field as required in the view so that it is mandatory to fill it out. To solve this a query before registering it in the database and if it has not been filled in, it is redirected to the same view informing the user that it is mandatory to select a date and time (Figure 7-7).



```
router.post('/add', isLoggedIn, async(req, res) => {
    const { user, link, description, datetime, email } = req.body;
    const splitDate = datetime.split(" ");
    const newReserv = {
        user,
        link,
        description,
        user: req.user.user,
        timestamp: datetime,
        familyemail: email,
        date: splitDate[0],
        time_start: splitDate[1],
    informed: false,
    initiated: false
    };

    if (datetime != "") {
        await connection.query('INSERT INTO reservation set ?', [newReserv]);

        req.flash('success', 'Link saved succesfully');
        res.redirect('/reservations/show');
    }
    else {
        req.flash('message', 'Date Time not selected');
        res.redirect('/reservations/add');
    }
});
```

Figure 7-7. POST request to add a scheduled reservation

- **residents.js**

  It contains the functions to process the requests:

  - GET request to delete a resident and show the list of them.
  - GET and POST requests to add or edit a resident.

In addition, several exportable functions have been implemented in the .js files in the lib directory to support the front-end of the application:

- **auth.js**

File containing two exportable functions used to check if a user is logged into the system or not. With this we achieve that a non-logged user cannot access certain parts of the application.

- **handlebars.js**

File containing two exportable functions that can be used from .hbs files with the idea of being able to format dates in two different ways:

  - "32 seconds ago", "5 hours ago", "2 days ago", "In 5 minutes".
  - "DD/MM/YYYY"

- **helpers.js**

File containing two exportable functions that are used to encrypt and decrypt login passwords that are stored and read from the database.

- **passport.js**

It has four important functions. The first (local.signin) is responsible for verifying that the login details entered by the user are correct, reporting if the user does not exist or if the password is incorrect. The second (local.signup) receives the registration data of a new administrator user and saves it in the database. And the last two are used to serialize and deserialize the user data.

7.1.3 *Video conference system*

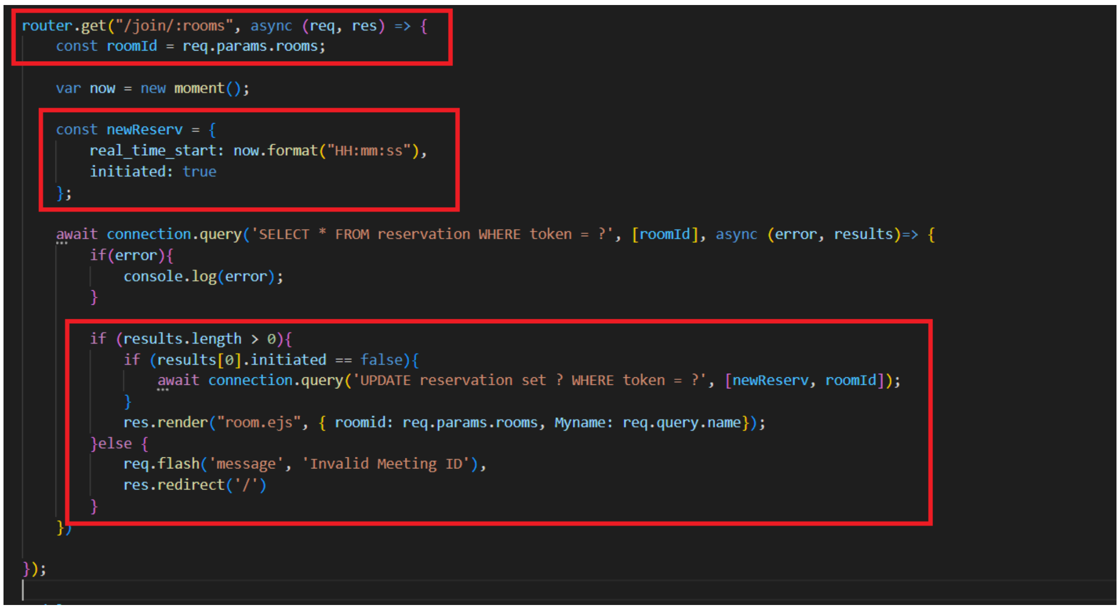
It has two parts:

1. The web interface for accessing the videoconferencing service, which is accessed through the web server that is already providing the access pages of the reservation system or through a link.
2. The signaling server.

7.1.3.1 *Web server*

The only page accessible through the management system or link, where the form is located, which, once passed through the signaling server, allows access to room.ejs, view that is controlled by main.js file. It is one of the most important part in the project since main.js is in charge of controlling the behavior of the video call and the flow of information exchanged among the endpoints and among them and the server. In general, it is the JavaScript file that the web browser will execute and therefore will define the behavior of each of the video calls that are made. In order not to make this part too long, only the most important developed and deployed parts of this JavaScript file are going to be commented.

The controller of this page we have it in the Routes folder index.js file. This file contains the functions to control the GET and POST request to join a scheduled video call. The POST method is used by doctors/specialists after logging in through their profile form. It should be noted that when we receive the GET request to enter the video call, we use the current time to enter the actual start time of the call in the database. As can be seen in Figure 7-8, when a GET request is made to the address /join/"roomID" we obtain this room ID from the request itself with the parameter “req.param.rooms” and the user name (if specified) by “using req.query.name”. For example in the case of a GET request to https://34.78.63.171:3030/join/aac37f-befb-d?name=Juan we obtain as req.param.rooms the value “aac37f-befb-d” and as req.query.name “Juan”. In addition, at this point it is validated, making a query to the database with the roomID obtained and checking with the field "initiated" if it is a video call already finished since a value of this parameter in true will indicate that the call has finished.



```
router.get("/join/:rooms", async (req, res) => {
    const roomId = req.params.rooms;

    var now = new moment();

    const newReserv = {
        real_time_start: now.format("HH:mm:ss"),
        initiated: true
    };

    await connection.query('SELECT * FROM reservation WHERE token = ?', [roomId], async (error, results)=> {
        if(error){
            console.log(error);
        }

        if (results.length > 0){
            if (results[0].initiated == false){
                await connection.query('UPDATE reservation set ? WHERE token = ?', [newReserv, roomId]);
            }
            res.render("room.ejs", { roomid: req.params.rooms, Myname: req.query.name});
        }else {
            req.flash('message', 'Invalid Meeting ID'),
            res.redirect('/')
        }
    })

});
```

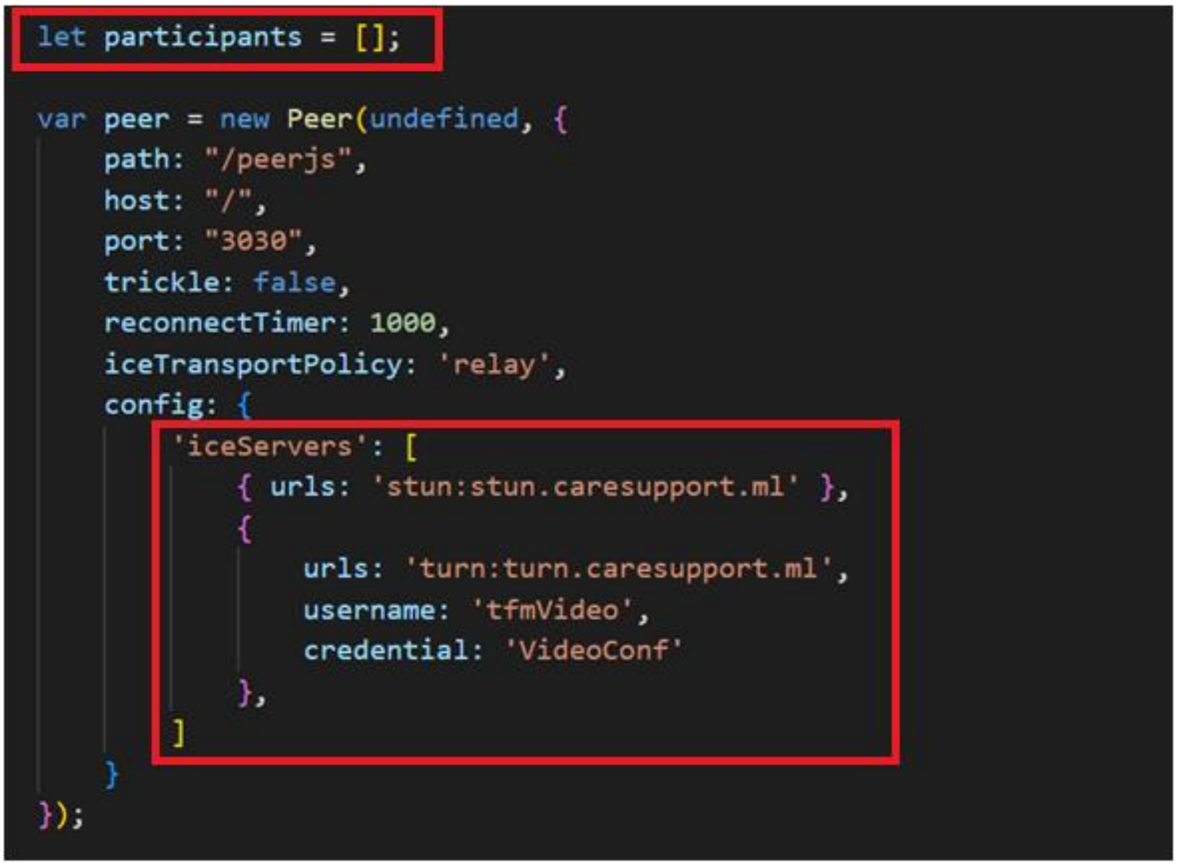
Figure 7-8. GET request to join a video call

Once the room has been created and the local peer associated with the user has been created, communication through the signaling server takes place:

1. Communication between signaling server and first client (browser) by redirection to room.ejs with websockets passing the info from the local peer, and it stays waiting for remote peers.
2. With peer.js, once another client is connected and indicated by the signaling server, communication at data level (the signaling is still done in the signaling server), it looks for the way to connect at data level (this is also part of the peer.js configuration – TURN / STUN, or directly if they are in the same network), to send the audio/video/chat stream, in a transparent way to the user.
3. Using WebRTC API and audio and video interfaces (hardware) are accessed to send data and display what arrives from remote peers. As well as the interface control, audio or video mute, device change…

The first one has created a participant’s variable with the aim of having the name of each of the participants in the video call to be able to view them at any time.

The first major change comes with the construction of the peers. As can be seen in figure 7-9, a dedicated STUN and TURN server has been created with COTURN with the aim of not depending on Google’s public servers, since on the one hand they are sometimes saturated and do not work and on the other hand, the information exchanged by the ends does not leave our own machine, providing more security and privacy to the data exchanged.

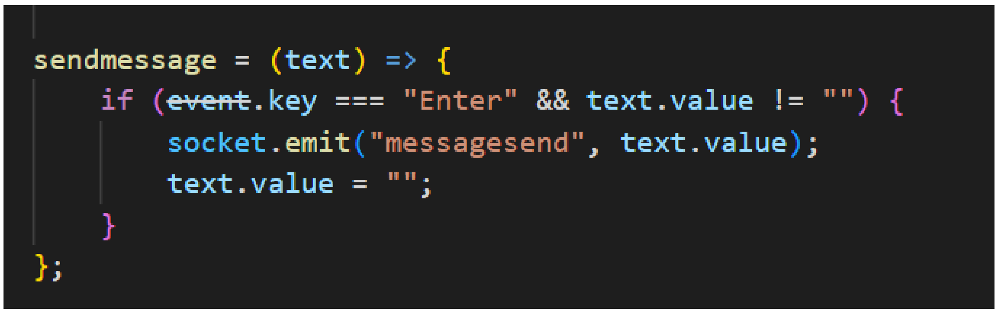


```
let participants = [];

var peer = new Peer(undefined, {
    path: "/peerjs",
    host: "/",
    port: "3030",
    trickle: false,
    reconnectTimer: 1000,
    iceTransportPolicy: 'relay',
    config: {
        'iceServers': [
            { urls: 'stun:stun.caresupport.ml' },
            {
                urls: 'turn:turn.caresupport.ml',
                username: 'tfmVideo',
                credential: 'VideoConf'
            },
        ]
    }
});
```

Figure 7-9. “participants” variable and peer constructor

The creation of dedicated STUN and TURN servers will be covered in a later section. In the peer constructor, we can specify the addresses of the STUN and TURN servers, and even concatenate several of them. In our case we have set the server that we have implemented with Coturn.

The next feature that has been implemented has to do with text chat. In this case, the sendmessage function is implemented, which has a very basic operation: once the user presses Enter and the text box is not empty, we send a message to the server (websockets) "messagesend" together with the text that has been written for later make the text box blank again. (Figure 7-10).

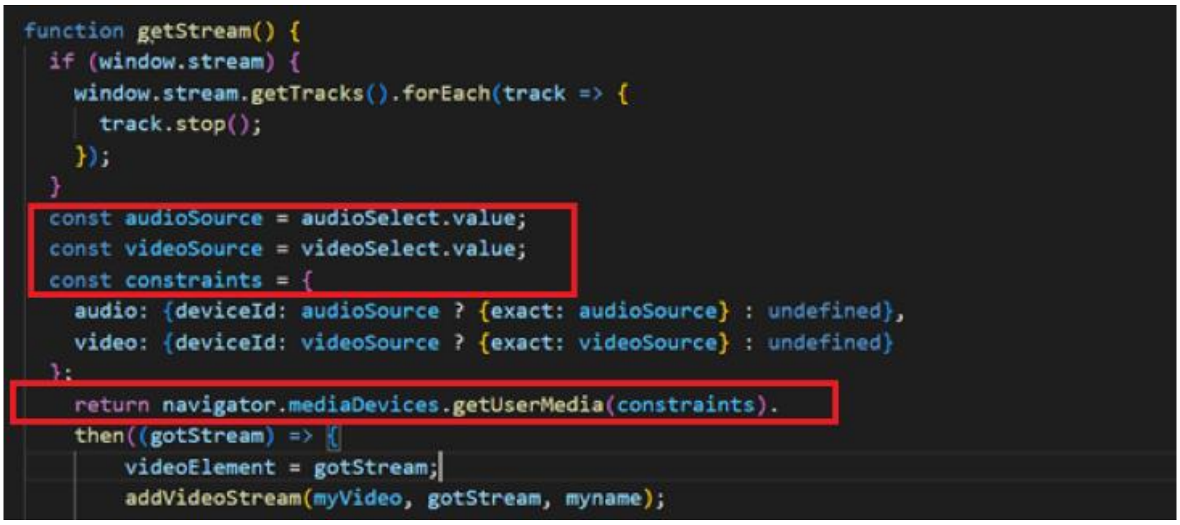


```
sendmessage = (text) => {
    if (~~event~~.key === "Enter" && text.value != "") {
        socket.emit("messagesend", text.value);
        text.value = "";
    }
};
```

Figure 7-10. “sendmessage” function

The following functions also have an important role since the robot that will have the system has several cameras and several microphones, so the user needs to be able to select or predefine a camera and a microphone from which they obtain audio and video. that is transmitted to the other participants of the video call. Besides, it incorporates more flexibility when being used from a mobile phone or a computer with several audio and video sources.

For this, the variables audioSelect and videoSelect have been defined along with a function to get the media from the system in question. Then they are assigned the user's choice and added to the stream that the user will share with the rest. It can be seen in the figure 7-11.

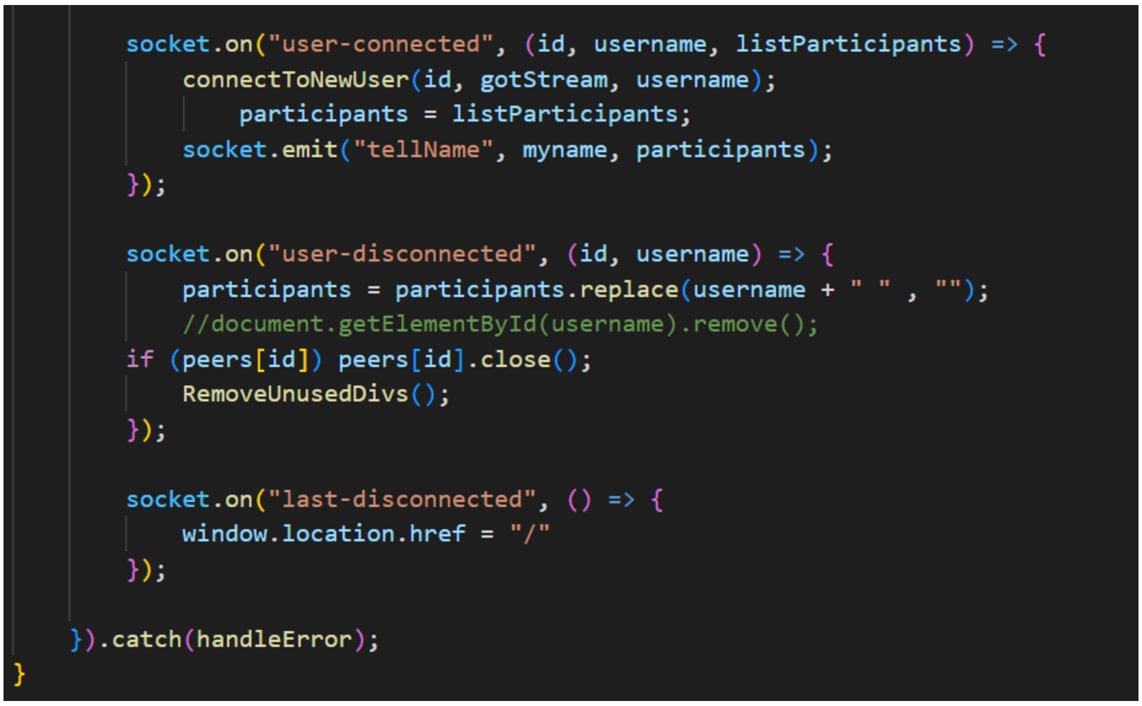


```
function getStream() {
  if (window.stream) {
    window.stream.getTracks().forEach(track => {
      track.stop();
    });
  }
  const audioSource = audioSelect.value;
  const videoSource = videoSelect.value;
  const constraints = {
    audio: {deviceId: audioSource ? {exact: audioSource} : undefined},
    video: {deviceId: videoSource ? {exact: videoSource} : undefined}
  };
    return navigator.mediaDevices.getUserMedia(constraints).
    then((gotStream) => {
        videoElement = gotStream;
        addVideoStream(myVideo, gotStream, myname);
```

Figure 7-11. Audio/video input device selection

In the following functions, related with the websockets communication among the server and the end-users, we are waiting to be able to receive 3 types of messages (Figure 7-12):

- **user-connected**, where we connect with the other user who started the room, and we also send a message to the server indicating our name to be able to be identified in the chat and in the list of participants.
- **user-disconnected**, upon receiving this message we delete the participant from the list and also look for the peer to close their connection and delete the div element where their camera is represented in the view.
- **last-disconnected**, is waiting to receive the message indicating that the last user in the room has left so that they can automatically disconnect (this is designed so that it is not necessary to physically touch the robot screen when the video call ends).

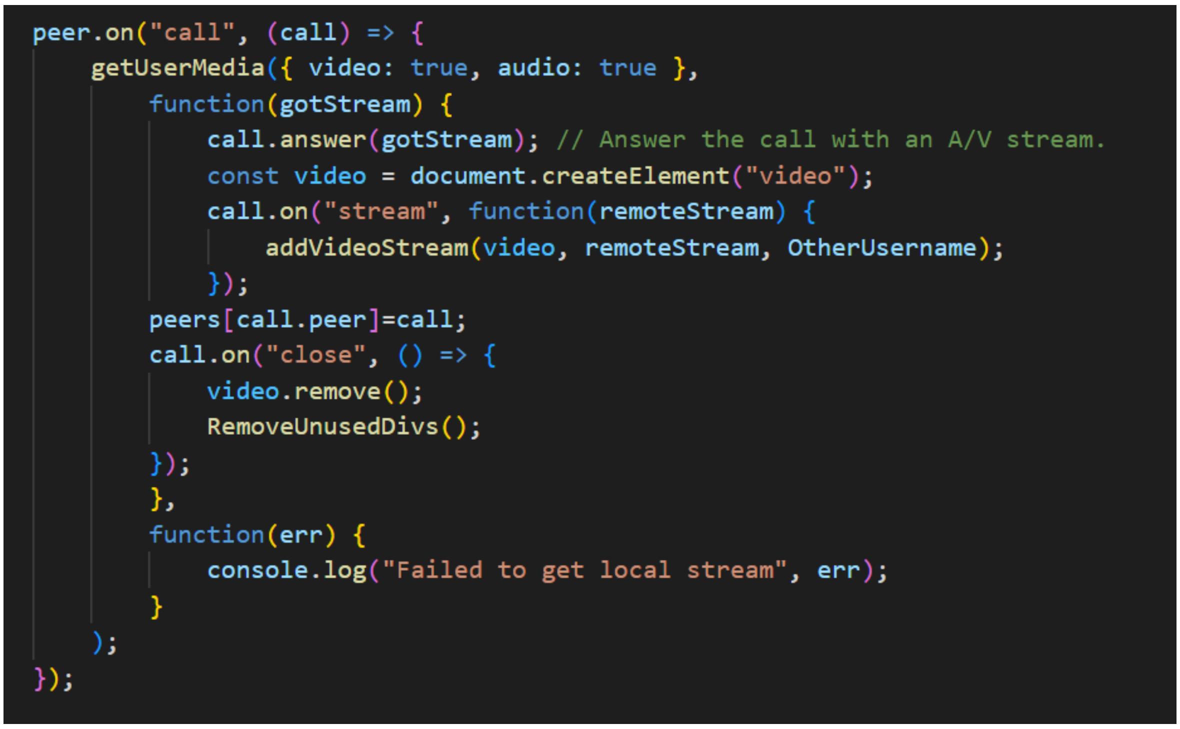


```
        socket.on("user-connected", (id, username, listParticipants) => {
            connectToNewUser(id, gotStream, username);
                participants = listParticipants;
            socket.emit("tellName", myname, participants);
        });

        socket.on("user-disconnected", (id, username) => {
            participants = participants.replace(username + " " , "");
            //document.getElementById(username).remove();
        if (peers[id]) peers[id].close();
            RemoveUnusedDivs();
        });

        socket.on("last-disconnected", () => {
            window.location.href = "/"
        });

    }).catch(handleError);
}
```

Figure 7-12. Connection handler messages

This part concerns the call response, when it receives the call message it replies with its stream to it.

In addition, a connection handler has also been added so that when a user closes the video call, it is also closed at the other end (Figure 7-13).

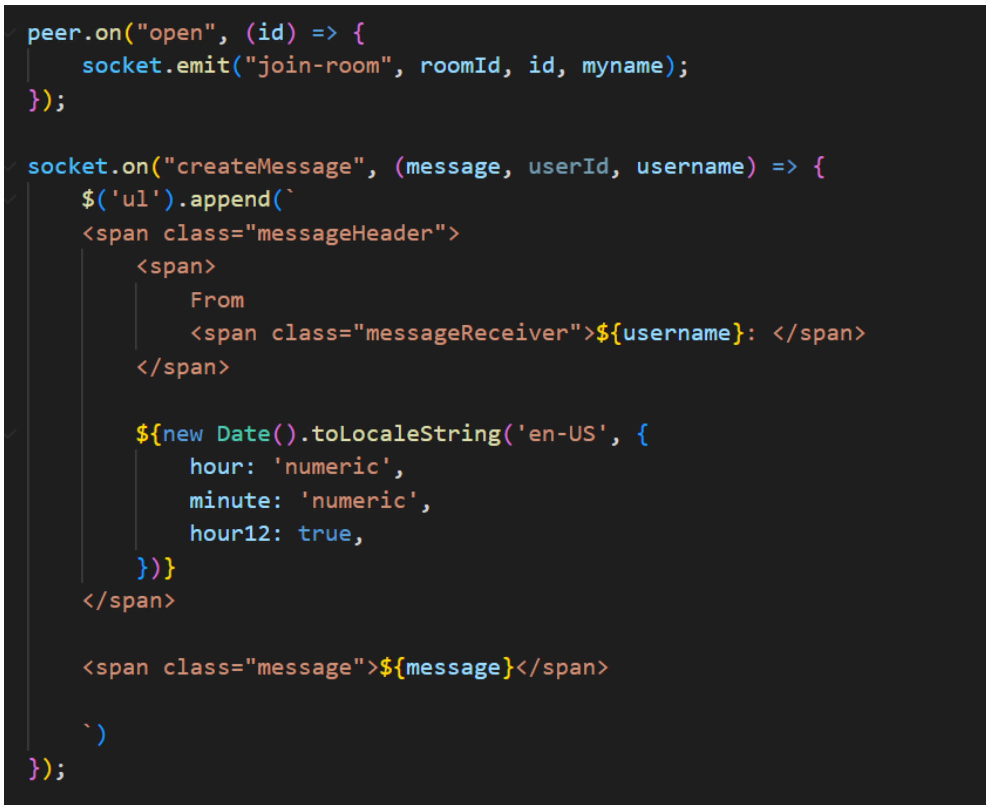


```
peer.on("call", (call) => {
    getUserMedia({ video: true, audio: true },
        function(gotStream) {
            call.answer(gotStream); // Answer the call with an A/V stream.
            const video = document.createElement("video");
            call.on("stream", function(remoteStream) {
                addVideoStream(video, remoteStream, OtherUsername);
            });
        peers[call.peer]=call;
        call.on("close", () => {
            video.remove();
            RemoveUnusedDivs();
        });
        },
        function(err) {
            console.log("Failed to get local stream", err);
        }
    );
});
```

Figure 7-13. Answer call function

The following two functions are relatively simple (Figure 7-14):

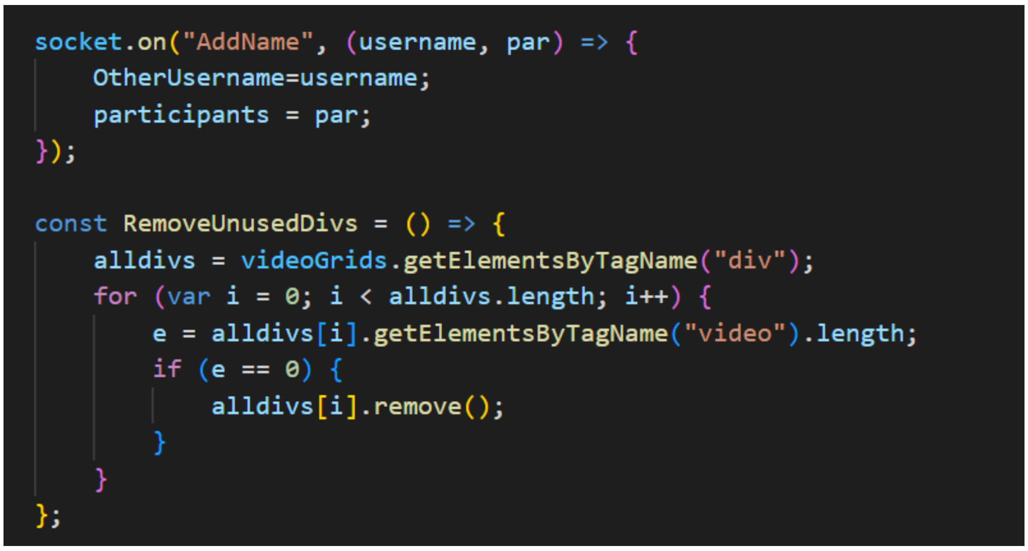
- **peer.on(open).** Send to the server that the first user has joined the room
- **socket.on(createMessage).** We get to format the message sent by a user.



```
peer.on("open", (id) => {
    socket.emit("join-room", roomId, id, myname);
});

socket.on("createMessage", (message, userId, username) => {
    $('ul').append(`
    <span class="messageHeader">
        <span>
            From
            <span class="messageReceiver">${username}: </span>
        </span>

        ${new Date().toLocaleString('en-US', {
            hour: 'numeric',
            minute: 'numeric',
            hour12: true,
        })}
    </span>

    <span class="message">${message}</span>

    `)
});
```

Figure 7-14. Message creation function

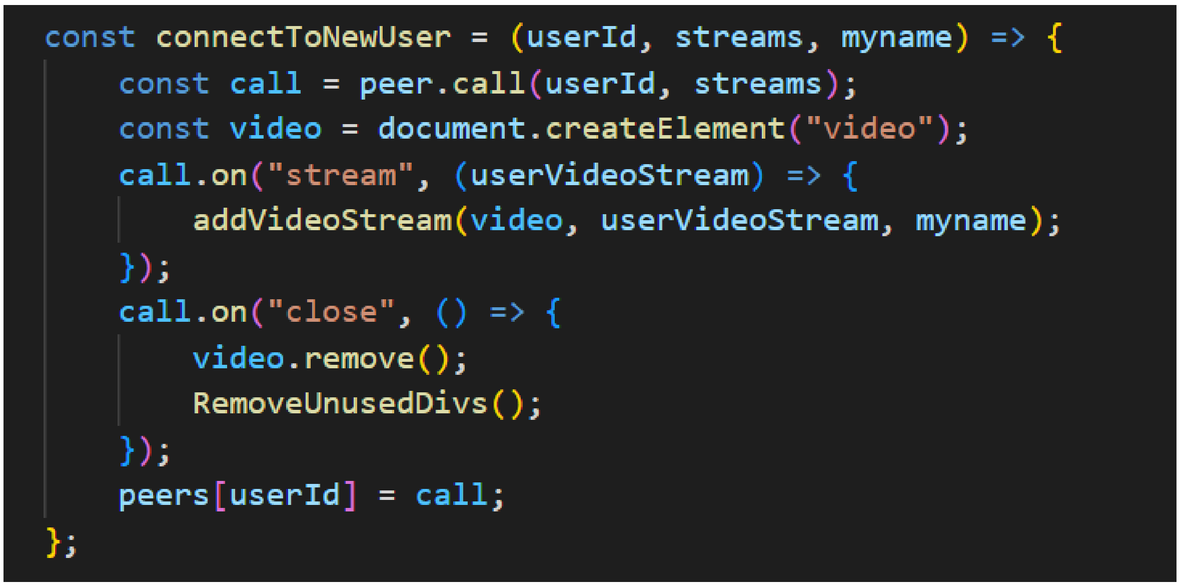
- **socket.on(AddName):** This function is used on the one hand to save the user name to write in the chat and on the other hand to save it in the list of participants
- **RemoveUnusedDivs:** It has been necessary to implement this function because once the user ends the call it is necessary to delete the camera from the screen of the other users. Its operation is simple, it obtains all the div elements that the users' cameras have and looks for the one we need to delete it.



```
socket.on("AddName", (username, par) => {
    OtherUsername=username;
    participants = par;
});

const RemoveUnusedDivs = () => {
    alldivs = videoGrids.getElementsByTagName("div");
    for (var i = 0; i < alldivs.length; i++) {
        e = alldivs[i].getElementsByTagName("video").length;
        if (e == 0) {
            alldivs[i].remove();
        }
    }
};
```

Figure 7-15. AddName and RemoveUnusedDivs functions

- **connectToNewUser:** function that provides the connection function with the rest of the users. It is also necessary in this part to have a connection manager to control the moment in which the users leave the video call and can remove their audio/video from the screens (Figure 7-16).

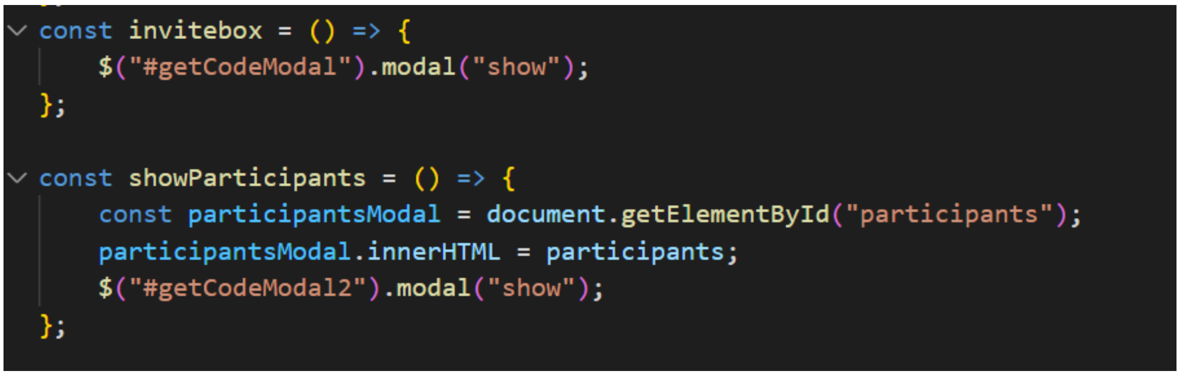


```
const connectToNewUser = (userId, streams, myname) => {
    const call = peer.call(userId, streams);
    const video = document.createElement("video");
    call.on("stream", (userVideoStream) => {
        addVideoStream(video, userVideoStream, myname);
    });
    call.on("close", () => {
        video.remove();
        RemoveUnusedDivs();
    });
    peers[userId] = call;
};
```

Figure 7-16. "connectToNewUser" function

The last functions of the file that we will describe have a similar behavior among them. They show/hide the different options of the video call application as well as are able to mute/unmute or turn on/off the camera.

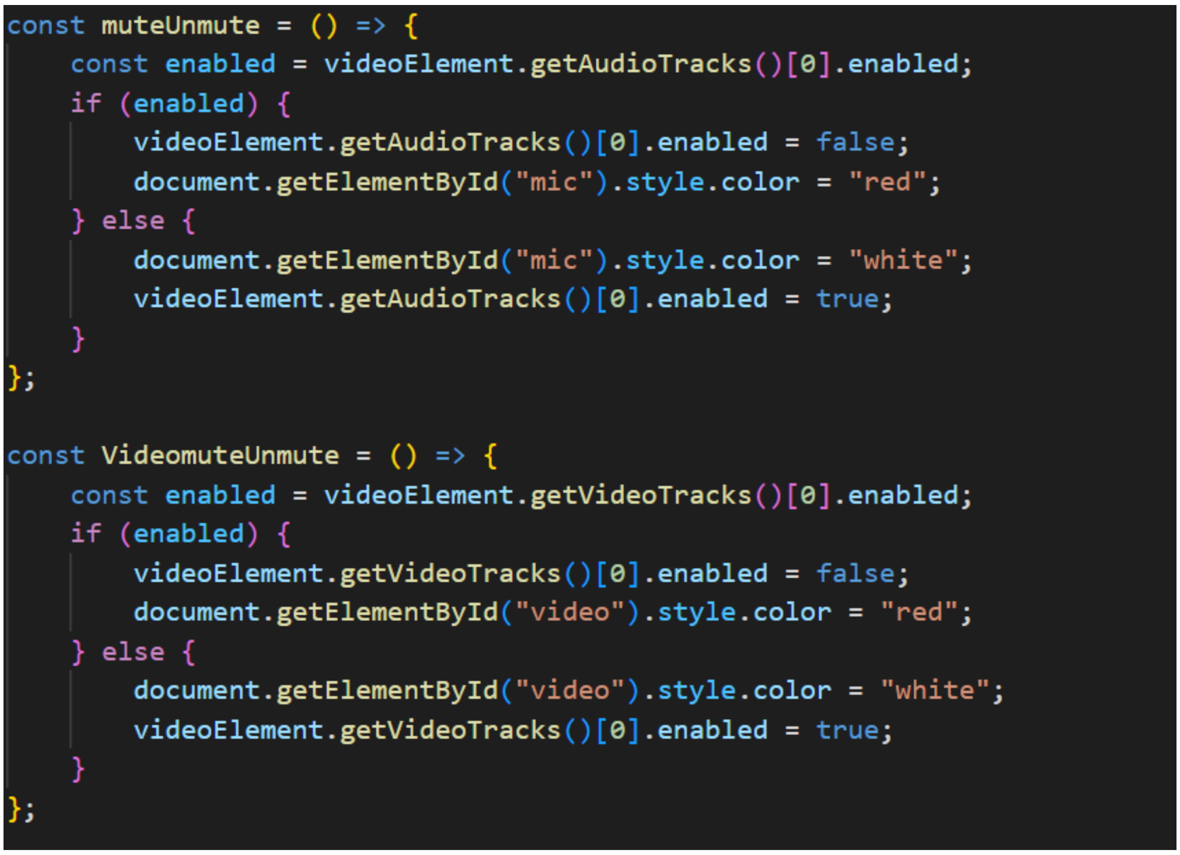
- **invitebox:** It shows us the modal that contains the meeting ID of the video call to be able to copy or share it.
- **showParticipants:** It shows us the modal that contains the list of participants in the video call.



```
const invitebox = () => {
    $("#getCodeModal").modal("show");
};

const showParticipants = () => {
    const participantsModal = document.getElementById("participants");
    participantsModal.innerHTML = participants;
    $("#getCodeModal2").modal("show");
};
```

Figure 7-17. "invitebox" and "showParticipants" functions

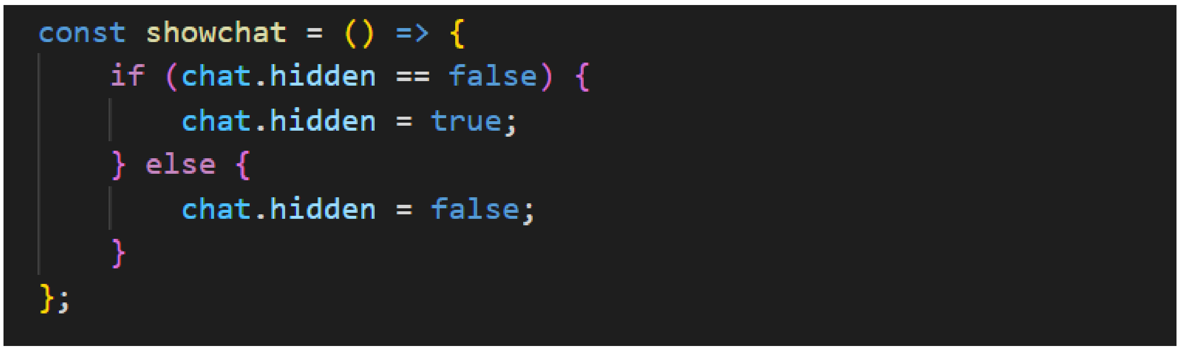
- **muteUnmute:** Controls the mute/unmute operation of the microphone.
- **VideomuteUnmute:** Controls the on/off operation of the camera.



```javascript
const muteUnmute = () => {
    const enabled = videoElement.getAudioTracks()[0].enabled;
    if (enabled) {
        videoElement.getAudioTracks()[0].enabled = false;
        document.getElementById("mic").style.color = "red";
    } else {
        document.getElementById("mic").style.color = "white";
        videoElement.getAudioTracks()[0].enabled = true;
    }
};

const VideomuteUnmute = () => {
    const enabled = videoElement.getVideoTracks()[0].enabled;
    if (enabled) {
        videoElement.getVideoTracks()[0].enabled = false;
        document.getElementById("video").style.color = "red";
    } else {
        document.getElementById("video").style.color = "white";
        videoElement.getVideoTracks()[0].enabled = true;
    }
};
```

Figure 7-18. “muteUnmute” and “VideomuteUnmute” functions

- **showChat:** Controls the chat button to show it on the screen or hide it (Figure 7.19).



```javascript
const showchat = () => {
    if (chat.hidden == false) {
        chat.hidden = true;
    } else {
        chat.hidden = false;
    }
};
```

Figure 7-19. “showchat” function

7.1.3.2 Signaling server

It has two parts:

1. It implements services associated with the patient and call management system. The communication is indirect through the use of common data from

the DB (query reservations, users) or through REST API. Here it’s important to control start of a call/access control to it/ and termination.

2. Allows communication through the use of websockets with the clients involved in the communication not only to indicate the creation or destruction of the room (beginning or end), but during the call the control part associated with the connection of new participants or abandonment of the same.

This is the most part of the core of the developed web application together with the main.js file already explained previously.

In this section we will start talking about the implementation of the services associated with and the patient and call management system and in the last part of the section will be dedicated to explain the functions related to the communication through websockets.

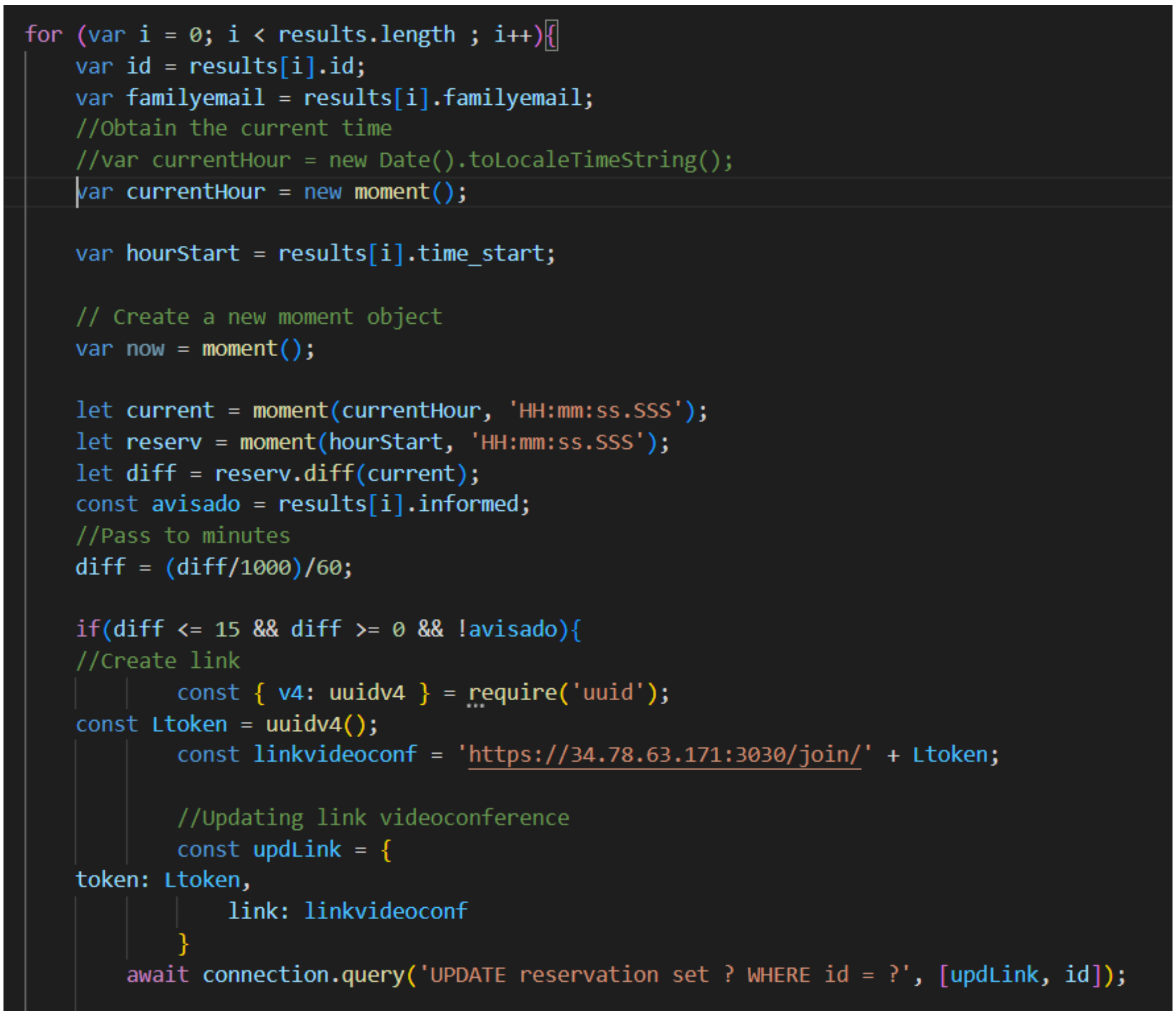
The file has been divided to keep it as organized as possible through comments. The initialization part refers to the call to the different node.js modules that the server will need, as well as defining the two template engines that will be used: HBS and EJS.

The next thing that has been implemented is the definition of the session, flash messages and login middleware together with the global variables (used for the pop-up messages) and the indication of the routers that it should use.

And from here the important functions will be discussed:

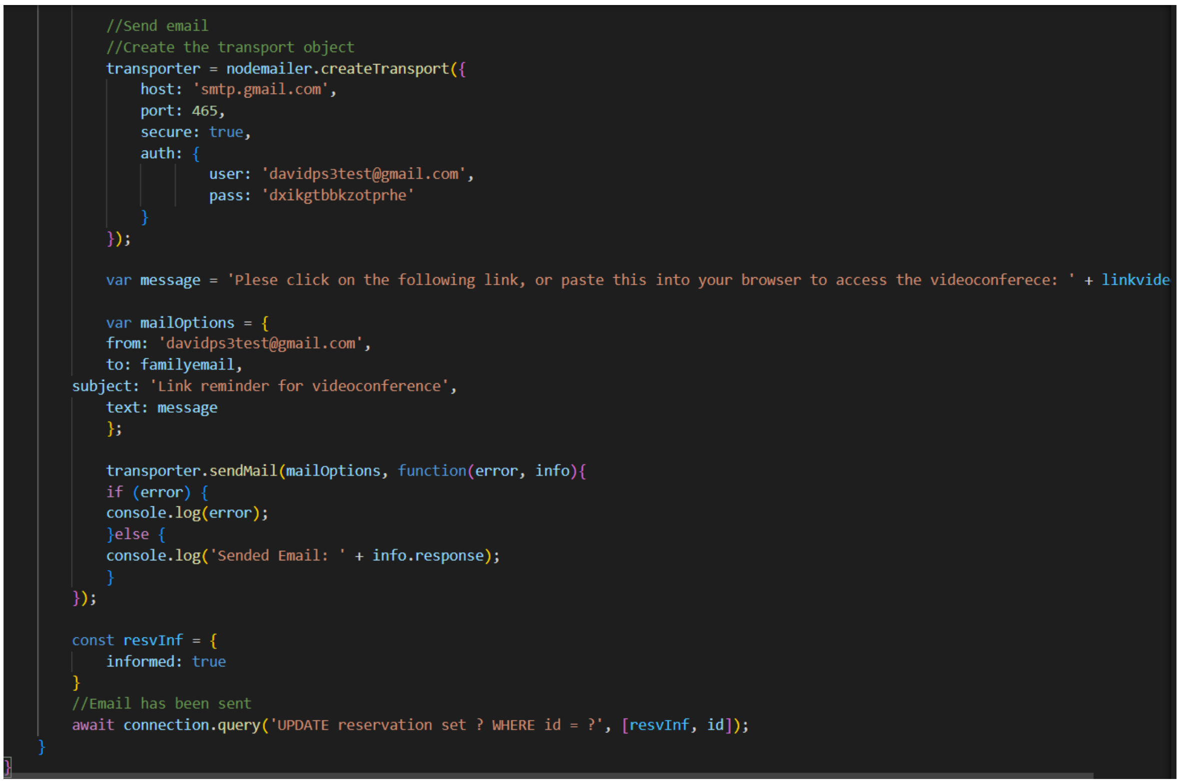
- **queryBBDD():** It is used to call the sendConsult() function every "x" time. This value is in milliseconds. It has been set by default to 5 minutes (300000 ms).
- **sendConsult():** It's a bit of a lengthy function because several checks need to be performed.

  First of all, it obtains the time and date of the moment of execution of the function. Once with this information, it queries the database to check if there are video calls scheduled for the same date. If there are, it obtains the data of each one of them and checks that there are less than 15 minutes left before the start of it and, furthermore, it has not been previously notified (Figure 7.20). If it meets these conditions, it creates a room ID and the video call link by adding it to the database. Then sends an email to the family member indicating its link and, before exiting, changes the boolean value of a variable that controls whether the familiar member has been notified by email (Figure 7.21).



```
for (var i = 0; i < results.length ; i++){
    var id = results[i].id;
    var familyemail = results[i].familyemail;
    //Obtain the current time
    //var currentHour = new Date().toLocaleTimeString();
    var currentHour = new moment();

    var hourStart = results[i].time_start;

    // Create a new moment object
    var now = moment();

    let current = moment(currentHour, 'HH:mm:ss.SSS');
    let reserv = moment(hourStart, 'HH:mm:ss.SSS');
    let diff = reserv.diff(current);
    const avisado = results[i].informed;
    //Pass to minutes
    diff = (diff/1000)/60;

    if(diff <= 15 && diff >= 0 && !avisado){
    //Create link
            const { v4: uuidv4 } = require('uuid');
    const Ltoken = uuidv4();
            const linkvideoconf = 'https://34.78.63.171:3030/join/' + Ltoken;

            //Updating link videoconference
            const updLink = {
    token: Ltoken,
                link: linkvideoconf
            }
        await connection.query('UPDATE reservation set ? WHERE id = ?', [updLink, id]);
```

Figure 7-20. Video call check in the day



```
            //Send email
            //Create the transport object
            transporter = nodemailer.createTransport({
                host: 'smtp.gmail.com',
                port: 465,
                secure: true,
                auth: {
                        user: 'davidps3test@gmail.com',
                        pass: 'dxikgtbbkzotprhe'
                }
            });

            var message = 'Plese click on the following link, or paste this into your browser to access the videoconferece: ' + linkvide

            var mailOptions = {
            from: 'davidps3test@gmail.com',
            to: familyemail,
        subject: 'Link reminder for videoconference',
            text: message
            };

            transporter.sendMail(mailOptions, function(error, info){
            if (error) {
            console.log(error);
            }else {
            console.log('Sended Email: ' + info.response);
            }
        });

        const resvInf = {
            informed: true
        }
        //Email has been sent
        await connection.query('UPDATE reservation set ? WHERE id = ?', [resvInf, id]);
    }
}
```

Figure 7-21. Sending email to familiar member

From here the signaling server implementation comes into play.

The last functions to be described are those related to communication through websockets. A variable has been defined that will contain the string of objects of each of the sessions to be carried out. This variable, called "stringObjects", is necessary in the event that more than one call coexists at the same time and therefore the signaling server knows how to differentiate the users of one from the users of another and thus treat them independently.

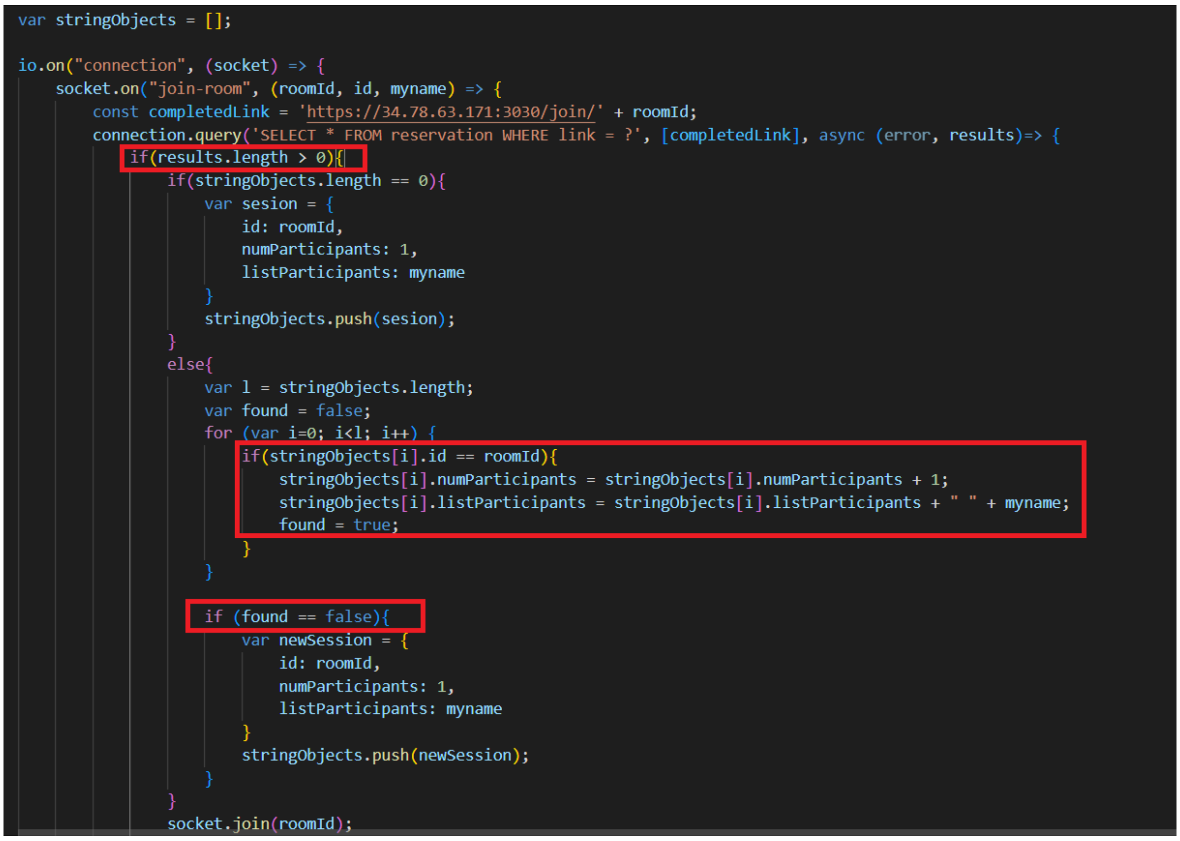
First, it opens the socket connection, and now it waits for messages from the different users that can connect.

Upon receiving the "join-room" message, the server queries the database to find out if that session exists and is valid (it has not ended). If the conditions are not met, the open connection is closed and the user is redirected to the application's home page. If you meet them, you must now control three different cases (Figure 7-22):

- The object string is empty: It means that the video call has not started, in this case we introduce the session in the “stringObjects” variable.
- The string is not empty and the session to which it is connected has already been entered in the “stringObjects” variable: It means that a new user wants to be added to a video call that has already started, therefore, we add the user's

nickname to the variable said session in the variable and we add the total number of participants by one to control the number of users in the video call.

- The string is not empty, and also that the session is not among the sessions that the stringObjects variable has: It means that a user is going to open a new session that has nothing to do with the ones that the signaling server already has. In this case, a new session is created inside the “stringObjects” variable.



```
var stringObjects = [];

io.on("connection", (socket) => {
    socket.on("join-room", (roomId, id, myname) => {
        const completedLink = 'https://34.78.63.171:3030/join/' + roomId;
        connection.query('SELECT * FROM reservation WHERE link = ?', [completedLink], async (error, results)=> {
            if(results.length > 0){
                if(stringObjects.length == 0){
                    var sesion = {
                        id: roomId,
                        numParticipants: 1,
                        listParticipants: myname
                    }
                    stringObjects.push(sesion);
                }
                else{
                    var l = stringObjects.length;
                    var found = false;
                    for (var i=0; i<l; i++) {
                        if(stringObjects[i].id == roomId){
                            stringObjects[i].numParticipants = stringObjects[i].numParticipants + 1;
                            stringObjects[i].listParticipants = stringObjects[i].listParticipants + " " + myname;
                            found = true;
                        }
                    }

                    if (found == false){
                        var newSession = {
                            id: roomId,
                            numParticipants: 1,
                            listParticipants: myname
                        }
                        stringObjects.push(newSession);
                    }
                }
                socket.join(roomId);
```

Figure 7-22. Session in variable “stringObjects”

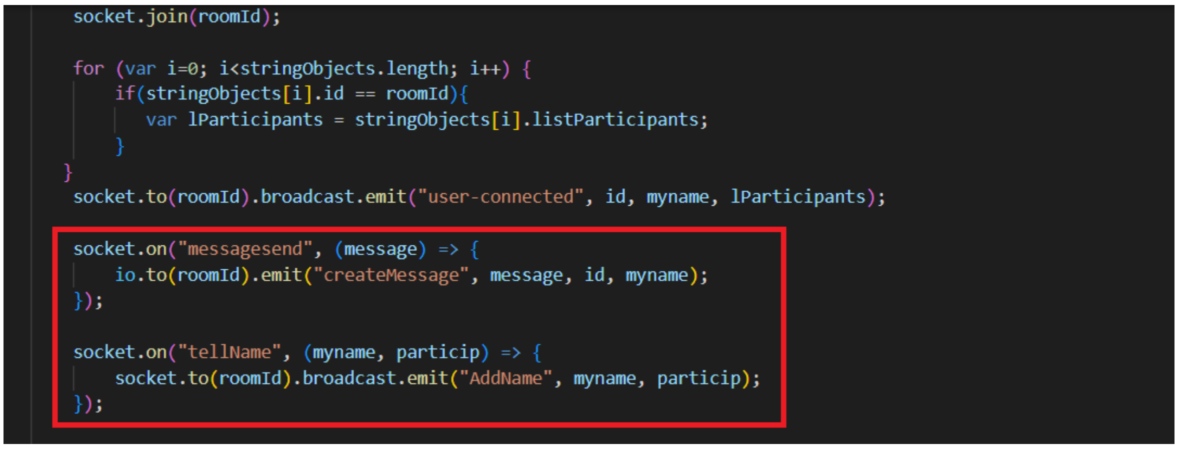
The video call is started and control messages are now needed to control the events that may occur.

When a user connects, it sends a broadcast message to all the other users that are in the call. In addition, some control parameters are passed that are needed in main.js in order to control the video call, such as the id, the name and the list of participants that are currently in the video call. This broadcast message is primarily responsible for the multi-user implementation of the web application.

To complete the operation of the video call (except ending), there are two more types of messages (Figure 7-23):

- **messagesend**: to send the message issued by the user to the chat.

- **tellName**: warns by emitting a broadcast message when a user connects to the other users of the video call so that they know the name of the person who has entered.

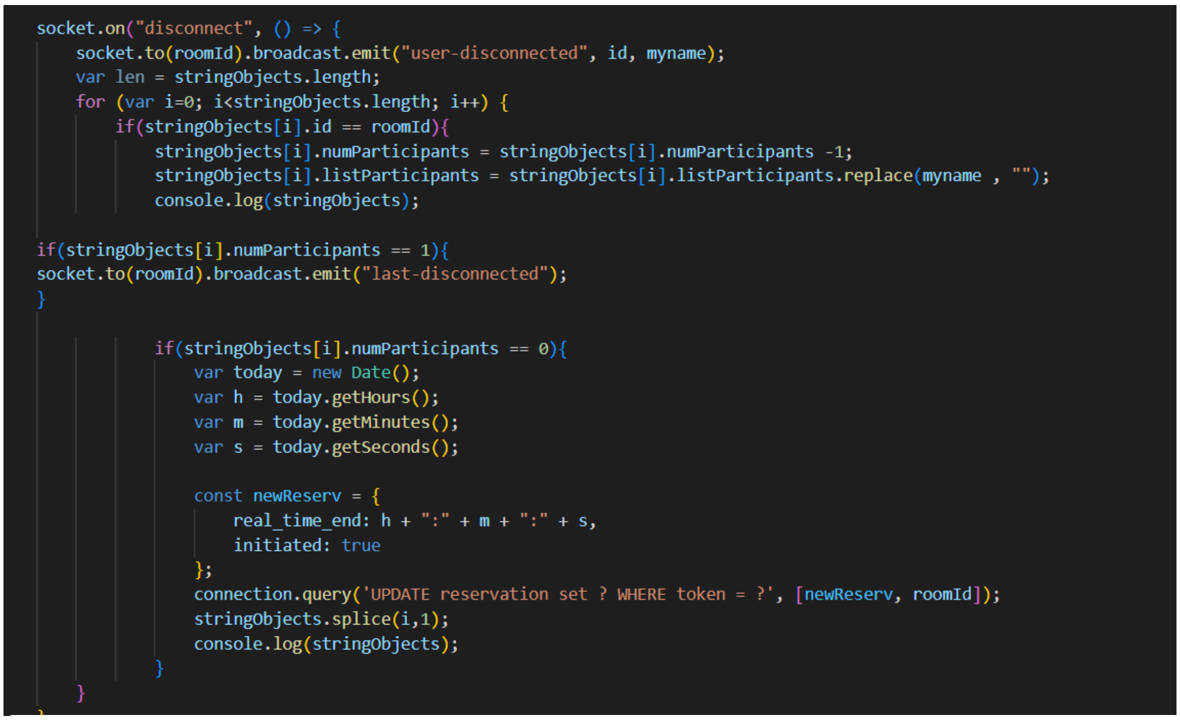


```
socket.join(roomId);

for (var i=0; i<stringObjects.length; i++) {
    if(stringObjects[i].id == roomId){
        var lParticipants = stringObjects[i].listParticipants;
    }
}
socket.to(roomId).broadcast.emit("user-connected", id, myname, lParticipants);

socket.on("messagesend", (message) => {
    io.to(roomId).emit("createMessage", message, id, myname);
});

socket.on("tellName", (myname, particip) => {
    socket.to(roomId).broadcast.emit("AddName", myname, particip);
});
```

Figure 7-23. “mesagesend” and “tellName” functions

The completion of calls is somewhat special since when the "disconnect" message is received indicating that the user is going to be disconnected, the connection with the rest of the users, the variable “stringObjects” and the number of participants in the call must be managed. This is achieved in next steps:

- First, a disconnect message is broadcasted to the other users so that they can close the connection at their end against us.
- Next, the video call object in the “stringObjects” variable is modified to decrease the number of participants and delete our name from the list of participants.
- Them, if the number of participants is one, it means that there is only one person left in the video call (as a general rule it will be the robot) so it emits a "last-disconnected" message to indicate that it will make the automatic disconnection.
- When this happens, the server obtains the current date and time and enters it in the database along with a boolean variable to indicate that the video call has already ended, so from now on it does not appear as a valid video call.

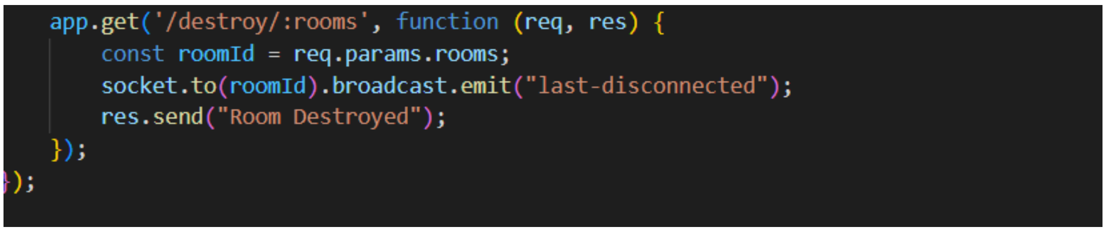


```
socket.on("disconnect", () => {
    socket.to(roomId).broadcast.emit("user-disconnected", id, myname);
    var len = stringObjects.length;
    for (var i=0; i<stringObjects.length; i++) {
        if(stringObjects[i].id == roomId){
            stringObjects[i].numParticipants = stringObjects[i].numParticipants -1;
            stringObjects[i].listParticipants = stringObjects[i].listParticipants.replace(myname , "");
            console.log(stringObjects);

if(stringObjects[i].numParticipants == 1){
socket.to(roomId).broadcast.emit("last-disconnected");
}

            if(stringObjects[i].numParticipants == 0){
                var today = new Date();
                var h = today.getHours();
                var m = today.getMinutes();
                var s = today.getSeconds();

                const newReserv = {
                    real_time_end: h + ":" + m + ":" + s,
                    initiated: true
                };
                connection.query('UPDATE reservation set ? WHERE token = ?', [newReserv, roomId]);
                stringObjects.splice(i,1);
                console.log(stringObjects);
            }
        }
```

Figure 7-24. Disconnection behavior

The last function that the server implements has to do with the FR7 discussed in section 6.4.2.1.

A security function is implemented to be able to destroy a video call at any time that is required. In this case, by means of a GET request, a broadcast message is issued to all the users of the video call who will automatically close the existing connection redirecting them to the home page and the administrator who made the request will be informed that the room has been destroyed.



```
    app.get('/destroy/:rooms', function (req, res) {
        const roomId = req.params.rooms;
        socket.to(roomId).broadcast.emit("last-disconnected");
        res.send("Room Destroyed");
    });
});
```

Figure 7-25. Security service for forced disconnection

7.1.4 Node.js modules used

The folder called “node_modules” contains all the Node.js modules that have been used for the development of this project and therefore for the operation of some functions that the web application has.

A module is a set of JavaScript objects and functions that can be used by external applications. The description of a piece of code as a module refers less to what the code is than to what it does; Any Node.js file or collection of files can be considered a module if its functions and data can be used in external programs [31].

The following modules have been used [32]:

- **bcryptjs:** It is a password hashing function designed by Niels Provos and David Maxieres, based on the Blowfish cipher. Allows saving passwords hashed instead of in plain text. In this case it has been used for the registration and login of users in the application.
- **connect-flash:** Allows developers to send a message every time a user is redirecting to a specific web page. It has been used to display information to the user in pop-up mode.
- **dotenv:** It is used to read the key and value pair from the .env and add it to the environment variable.
- **ejs:** It is a module that allows us to use the views in EJS format in our project.
- **express:** Allows the use of Express in the project. As described in section 6.1.3.
- **express-handlebars:** Allows the use of hbs views in our project. It is a Javascript template system based on Mustache Templates. Keep the code and the view separated. Allows generating HTML from objects with data in JSON format.
- **express-mysql-session:** It is a MySQL session store for express.js, and it is mainly used to upload session data to the MySQL database. This module creates a database table to save session data.
- **express-session:** The express-session middleware stores session data on the server; it only saves the session ID in the cookie itself, not the session data.
- **express-validator:** Validation in node.js can be easily done using the module. This module is popular for data validation.
- **moment:** It is a great help for managing dates in JavaScript.
- **morgan:** Morgan is an HTTP Request Layer Middleware. It's a great tool that logs requests along with some other information depending on your configuration and the default value used. It proves to be very useful while debugging and also if you want to create log files.
- **mysql:** A node.js driver for mysql.

- **nodemailer:** Nodemailer is the Node.js npm module that makes it easy to send emails. Used for sending emails to familiar members where the link of the video call to be used is indicated.
- **passport:** It is a Node.js module that uses the middleware design pattern for authentication requests.
- **passport-local:** It is used to implement a local authentication. This module allows to authenticate with a username and password in Node. js.
- **peer:** It helps establishing connections between PeerJS clients as described in Section 6.2.1. Data is not proxied through the server.
- **react-router-dom**: The react-router-dom package contains bindings for using React Router in web applications.
- **socket.io:** Socket.IO provides the ability to create applications that communicate in real time with a browser and mobile device, regardless of different communication mechanisms. as described in Section 6.2.1.
- **timeago.js**: is a nano library (less than 2 kb) used to format datetime with *** time ago statement. eg: '3 hours ago'.
- **util:** Provides utility functions for formatting strings, converting objects to strings, checking the type of objects, and performing synchronous writing to output streams, as well as some improvements to object inheritance.
- **uuid**: Used to generate a universally unique identifier. With this, unique and unequivocal rooms are created.
- **nodemon:** nodemon is a command line interface (CLI) utility that wraps a Node application, watches the file system, and automatically restarts the process.
- **react-error-overlay:** react-error-overlay is an overlay which displays when there is a runtime error.

7.1.5 Connection of the application with the database

We have two files to make the connection to our database:

- **db.js**

File for integration with the database. It includes a “connection” variable with the connection to the web application's database and a function to capture possible errors.

- **keys.js**

Contains an export module that allows to connect to the database and to save user sessions.

7.1.6 Certificates

Today's browsers, for security reasons, do not allow access to the camera and microphone when the web server is running over HTTP and the use of HTTPS is mandatory (except for the case of being the server at localhost).

Therefore, since our server is running at the cloud with a public IP address, for the deployment of this application we need to use HTTPS, which entails having security certificates for our website.

To obtain SSL certificates there are different options, in our case, zerossl.com has been chosen since it provides free certificates for three months that can be also renewed without cost.

Once a fixed public IP has been configured on virtual machine of Google Cloud Platform where our server is located, zerossl.com website has been used to obtain the certificates for our website after carrying out a series of steps and checks (a file generated by the platform must be made accessible on your server in a specific route to check the ownership of the server).

By this way three files are generated: ca_bundle.crt, certificate.crt, and private.key. Using an account in zerossl.com in free mode, you need to renew these files every 3 months, because they validity expires after 90 days.

7.2 VideoConf application design

Based on the use case identified in section 6.4.3, seven views have been designed.

For each of them, a wireframe or content diagram will be shown below:

- Administrator registration form
- Resident registration form
- Screen to show all residents of the system
- Screen to show the profile of the logged in user
- Form to schedule video calls
- Screen to show the different video calls of the specialist/doctor

As shown in Figure 7-26, the main page of the web application consistes of two main sections as the typical single-page websites. The navigation bar to access to all the functions implemented and the main window that adapts its view depending of the chosen functionality.

The web application has four well-differentiated section as can be seen in Figure 7-26, each of which will allow the user to access a specific function.

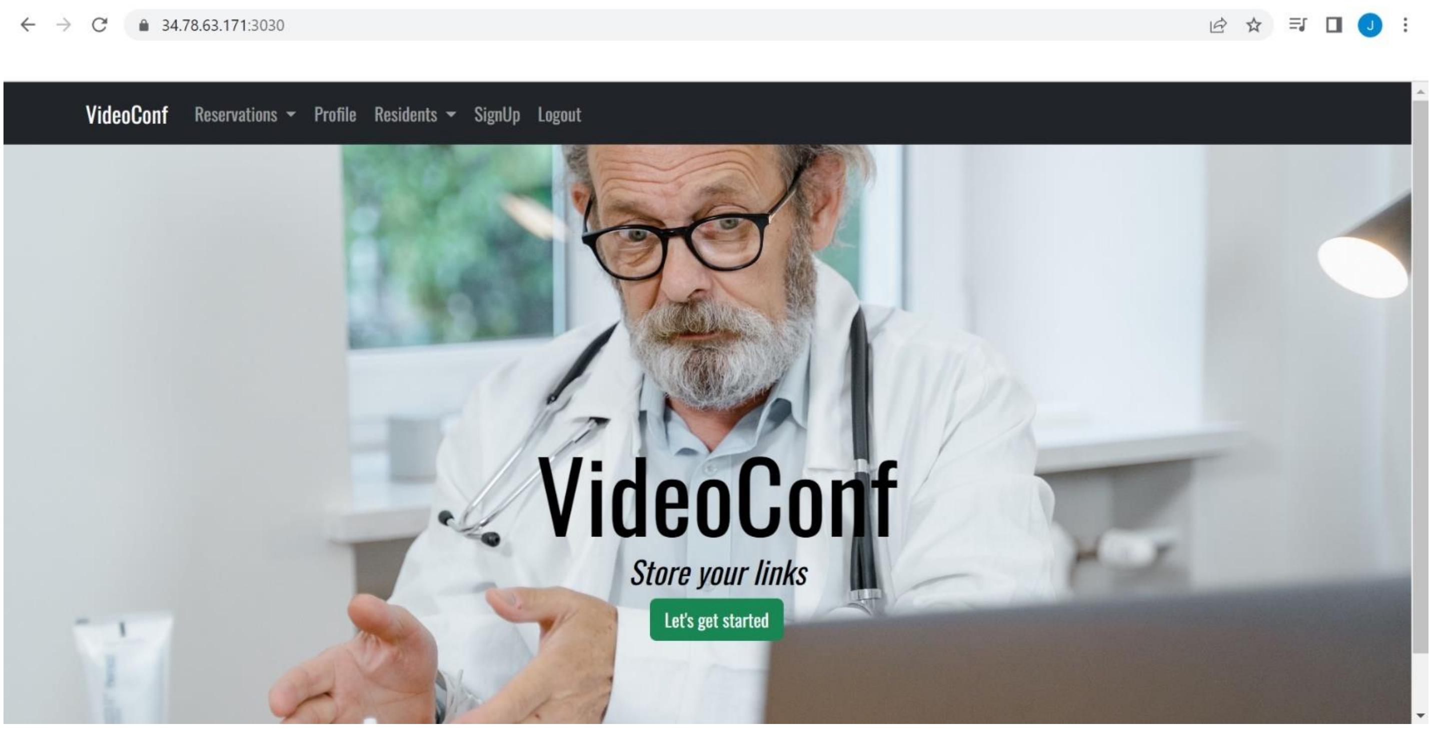


Figure 7-26. VideoConf application

The SingUp item in the bar is linked with the view of the administrator registration section (Figure 7-27), shows a form with fields associated to the administrator profile that with the will be saved in the DB, once a new user is registered. At the end the supervisor will be redirected to the view of his/her profile.

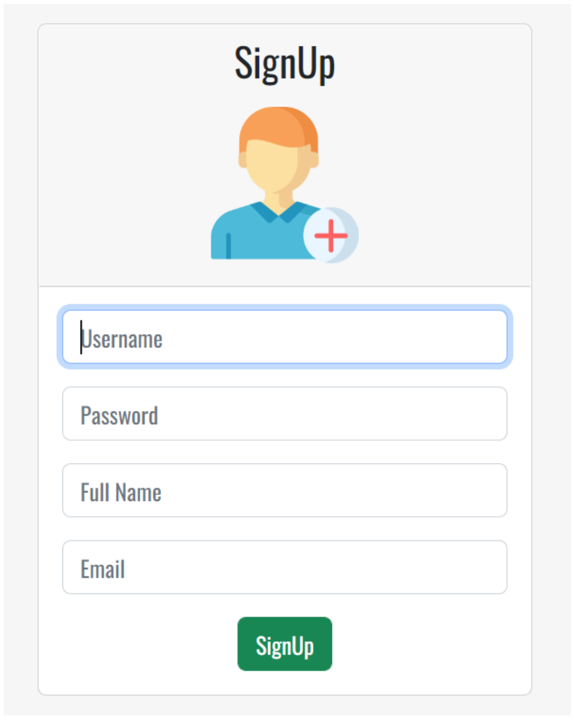


Figure 7-27. User form registration

The section associated with the registration of a resident (Figure 7-28) and accessed from Residents item in the navigation bar shows a form with the corresponding data to include in the DB. Once registration is complete, the system redirects the user to a view with all the residents registered in the system (Figure 7-29).

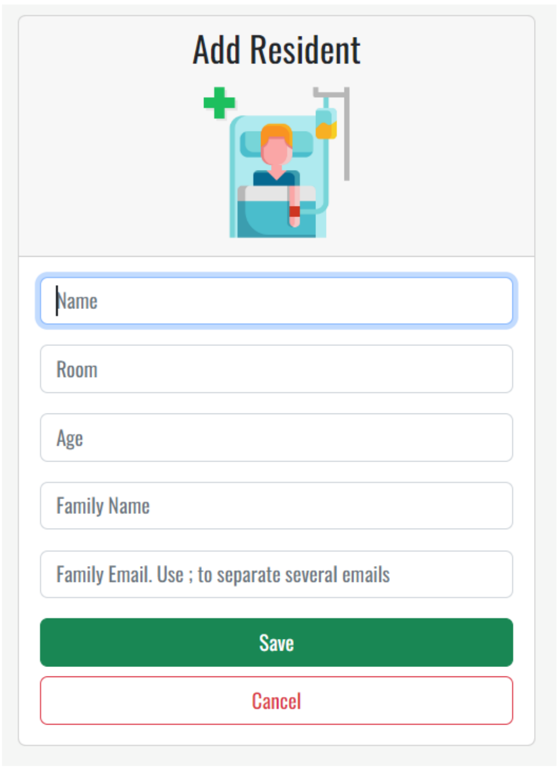
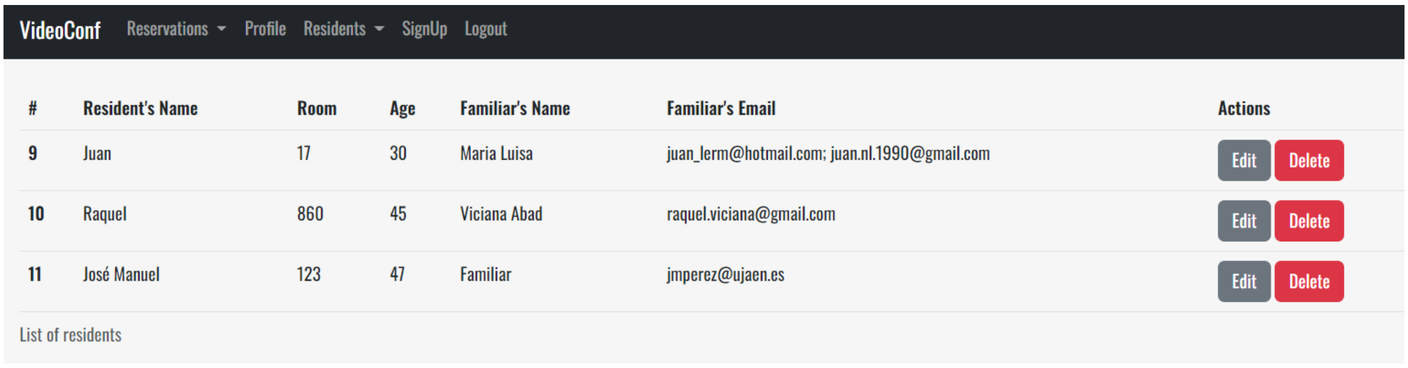


Figure 7-28. Resident form registration



#	Resident's Name	Room	Age	Familiar's Name	Familiar's Email	Actions
9	Juan	17	30	Maria Luisa	juan_lerm@hotmail.com; juan.nl.1990@gmail.com	Edit Delete
10	Raquel	860	45	Viciana Abad	raquel.viciana@gmail.com	Edit Delete
11	José Manuel	123	47	Familiar	jmperez@ujaen.es	Edit Delete

List of residents

Figure 7-29. Residents list

The user profile view (Figure 7-30) has two sections. The first contains a box where it shows the nickname and the name welcoming the specific administrator with a button that allows making a new video call reservations. The second part is a form where indicating the meeting ID of a session and our username allows the specialist to enter a valid video call.

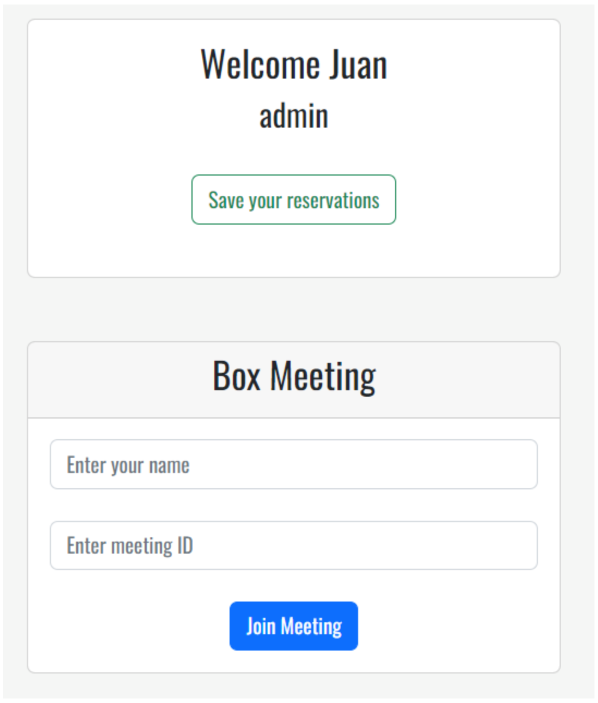
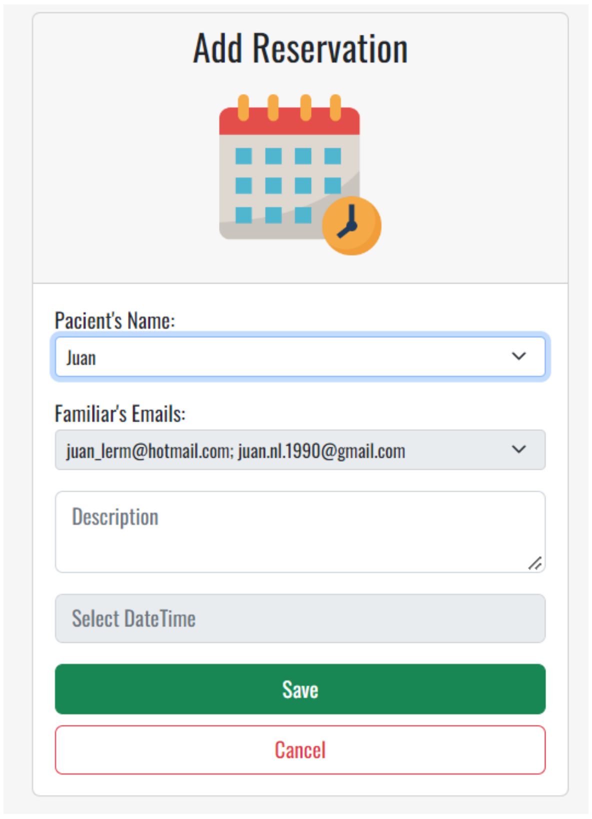


Figure 7-30. Profile view

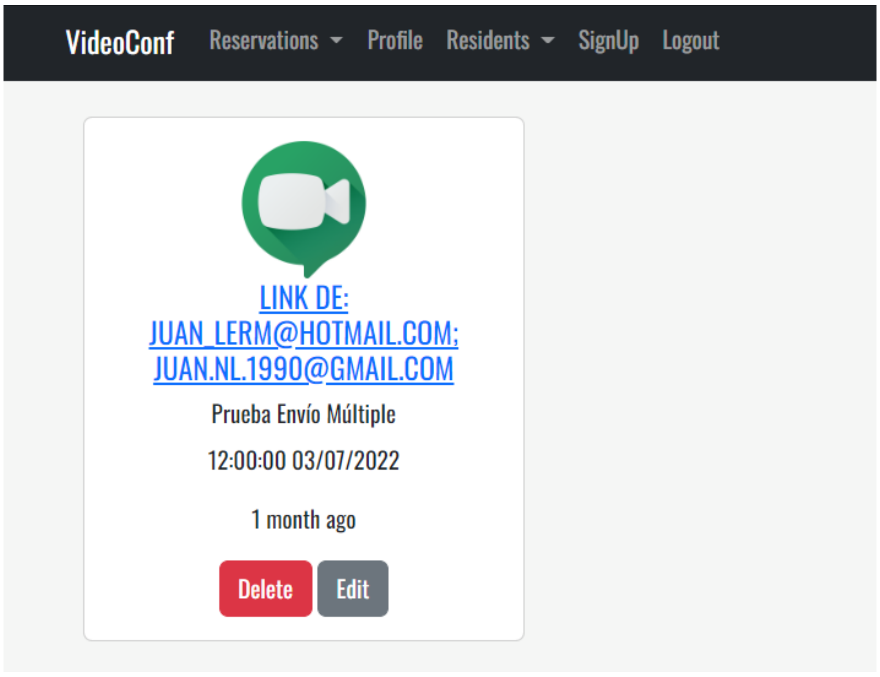
The screen to schedule a video call (Figure 7-31), will show a form with the necessary data of the video call to schedule, the email box appears in gray since it is disabled as it is filled automatically when the patient is selected. Once the registration is complete, the system redirects the user to a view with all the video calls registered by the specialist or doctor (Figure 7-32).


Add Reservation

Pacient's Name: Juan

Familiar's Emails: juan_lerm@hotmail.com; juan.nl.1990@gmail.com

Figure 7-31. Schedule form reservation



LINK DE: JUAN_LERM@HOTMAIL.COM; JUAN.NL.1990@GMAIL.COM

Prueba Envío Múltiple

12:00:00 03/07/2022

1 month ago

Figure 7-32. Videocalls list

The final view in which the video call can be made (Figure 7-33) consists of:

- Main window where the different cameras of the participants of the video call are shown.
- Text chat on the right side where participants can write.
- Mute/unmute button.
- Button to turn on/off the camera.
- Invite button where we will get the Room ID.
- Participants button where we can see the name of the participants of the video call.
- Chat button to hide/show text chat.
- Audio source button where the user can select the audio source.
- Video source button where the user can select the video source.

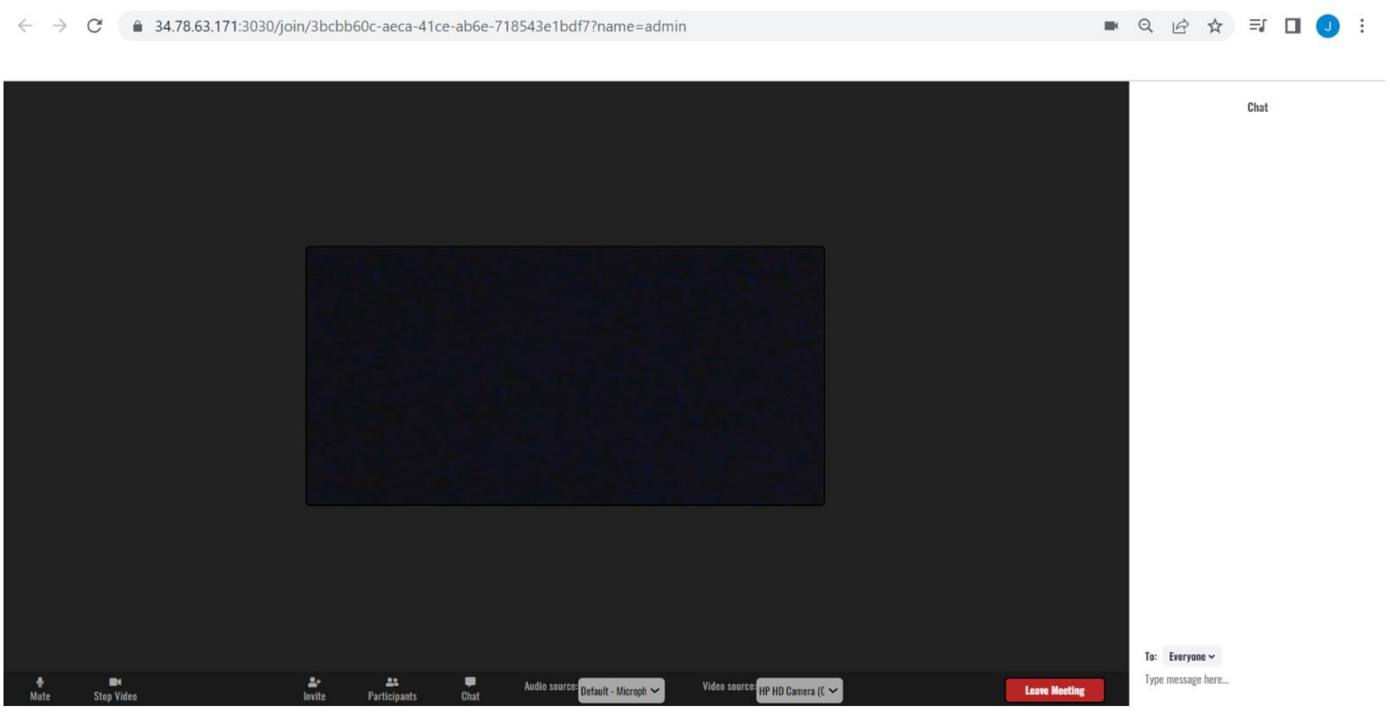

34.78.63.171:3030/join/3bcbb60c-aeca-41ce-ab6e-718543e1bdf7?name=admin

Figure 7-33. VideoCall view

7.3 Database structure

In the following sections, the structure of the database is described in terms of the entities that comprise it, as well as their attributes and data types.

7.3.1 Entities

The logic applied to the management of the information generated during the operation of the web application is resolved in a relational schema and is implemented on a MariaDB database. This database is running in the same virtual machine of Google Cloud Platform where the main server is located.

Figure 7-34 shows an entity diagram of the designed database.

• **login:** It is the entity where the information of the administrator users is represented.

• **reservation:** This entity stores all the information that has to do with video calls and their life cycle: ID, Meeting ID, date-time, user who created it, start time, real start time, real end time, link, email of the family member, if they have been informed by email, if the video call has ended and description.

• **residents:** Represents patient information relating them to their relatives in the system: ID, resident's name, room, age, relative's name, relative's email.

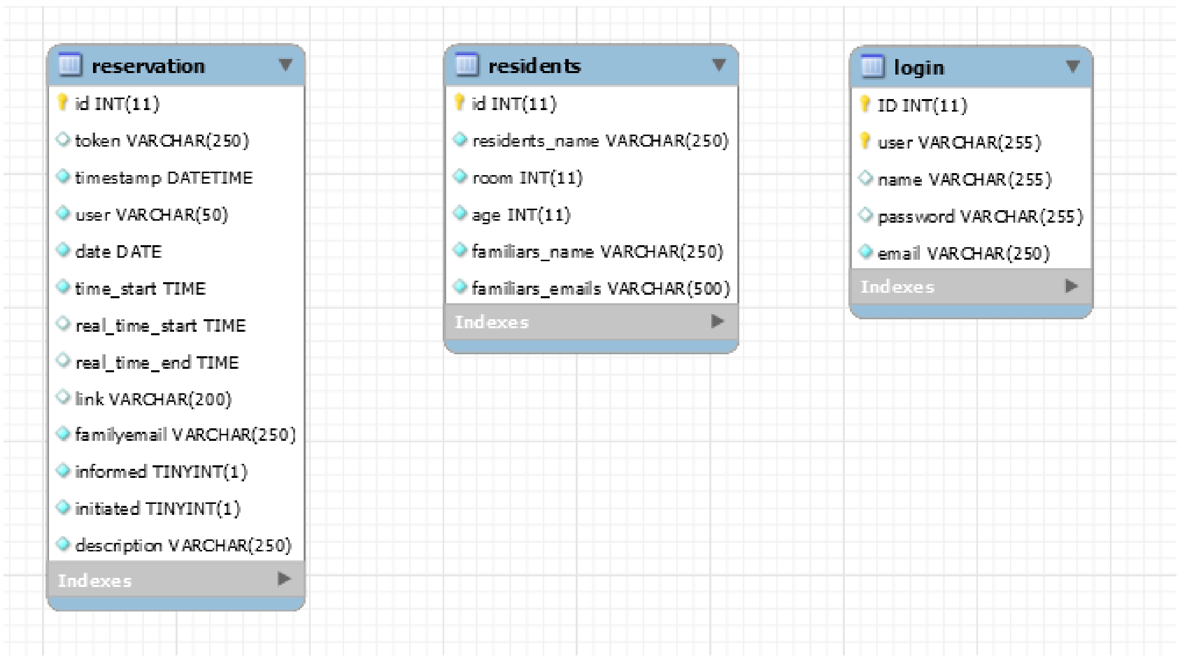


Figure 7-34. Entity diagram

7.3.2 Relational model

In the tables Table 7-1, Table 7-2 and Table 7-3 describe each of the above entities: their attributes, data type and their description.

- **reservation**

Attribute	Type	Length	Mandatory	Description
ID	INT	11	YES	Primary key of the entity. Unique identifier of each reservation.
TOKEN	VARCHAR	250	NO	Unique identifier of each of the video calls.
TIMESTAMP	DATETIME	-	YES	Reservation date and time
USER	VARCHAR	50	YES	User who schedules the video call
DATE	DATE	-	YES	Video call date
TIME_START	TIME	-	YES	Video call time
REAL_TIME_START	TIME	-	NO	Real start time of the video call
REAL_TIME_END	TIME	-	NO	Real end time of the video call
LINK	VARCHAR	200	NO	Video call link
FAMILYEMAIL	VARCHAR	250	YES	Family emails
INFORMED	TINYINT	1	YES	2 states: True or False
INITIATED	TINYINT	1	YES	2 states: True or False
DESCRIPTION	VARCHAR	250	YES	Description of the video call

Table 7-1. Reservation entity

- **residents**

Attribute	Type	Length	Mandatory	Description
ID	INT	11	YES	Primary key of the entity. Unique identifier of each resident.
RESIDENTS_NAME	VARCHAR	250	YES	Resident's name.
ROOM	INT	11	YES	Room number.
AGE	INT	11	YES	Resident's age.
FAMILIARS_NAME	VARCHAR	250	YES	Family Name.
FAMILIARS_EMAILS	VARCHAR	500	YES	Family Email.

Table 7-2. Residents entity

- **login**

Attribute	Type	Length	Mandatory	Description
ID	INT	11	YES	Primary key of the entity. Unique identifier of each user.
USER	VARCHAR	255	YES	Username.
NAME	VARCHAR	255	NO	Name.
PASSWORD	VARCHAR	255	YES	Password.
EMAIL	VARCHAR	250	YES	Email.

Table 7-3. Login entity

7.4 STUN/TURN Private server

As explained in sections 6.3.4 and 6.3.5, the STUN protocol allows other protocols to deal with NAT and TURN allows a client located behind a NAT or firewall to communicate directly with other clients using the service of an intermediate node acting as an intermediary.

These protocols are necessary to establish peer-to-peer communication in the video call.

In the early deployment of the web application, we started using public STUN and TURN servers:

```
REACT_APP_SIGNALING_SERVER = http://localhost:3030
REACT_APP_STUN_SERVERS = stun: stun4.l.google.com:19302
REACT_APP_TURN_SERVERS = turn: numb.viagenie.ca
# You can create your turn account here: http://numb.viagenie.ca/cgi-bin/numbacct
REACT_APP_TURN_USERNAME = [email]
REACT_APP_TURN_CREDENCIAL = [password]
```

To use the TURN server all we have to do is to create a user with his credentials on the web page shown in the table above.

After a thorough analysis of the environment where the application was going to be used and the data privacy policies that are normally in these places, it was decided that we should create our own dedicated STUN/TURN servers. In addition, with the implementation of our own STUN/TURN servers we avoid certain problems that may arise in the future such as:

- In the event of a change of address of the servers or if the servers stop working, the web application would stop working.
- If the web servers receive too many requests or receive a cyber-attack, they can collapse and stop offering services.

- Data privacy is not exposed as the traffic does not go to a third party.

To implement our own STUN / TURN server, we will rely on the Coturn project. Coturn is a free and open-source implementation of a TURN and STUN server for VoIP and WebRTC.

The installation of coturn on linux is quite simple. All we have to do is run this command:

```
sudo apt-get install coturn
```

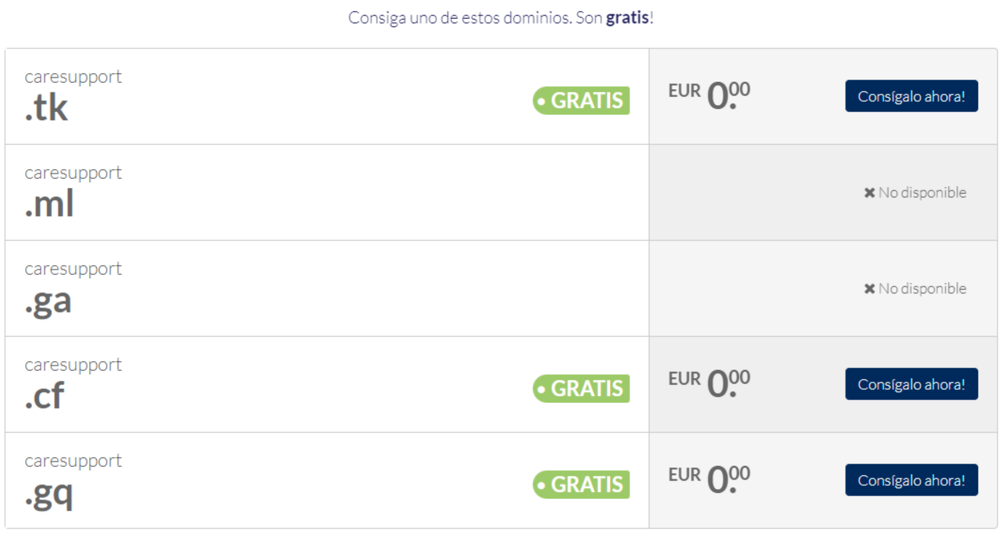
Be sure to stop the service after installing the package with the following command, as it will start automatically once the installation is finished and we need to configure Coturn.

```
systemctl stop coturn
```

After installation, edit the Coturn configuration file located in the path "etc/default/coturn" and uncomment the "TURNSERVER_ENABLED" property and set it to 1: TURNSERVER_ENABLED=1

A requirement of Coturn is that you must own some domain where the STUN / TURN server will be hosted. You have to create 2 new "A" records using TURN and STUN as host respectively pointing to the public IP of your server.

In our case we have used the free service offered by http://www.dot.tk/. From the page we can check the availability of domains and offer some free ones as shown in figure 7-35.



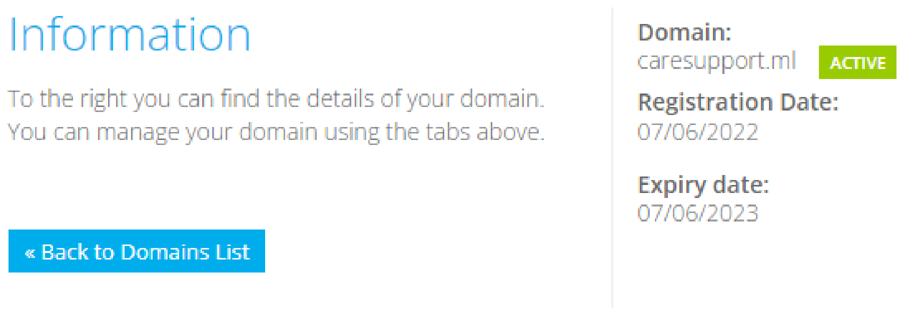
Consiga uno de estos dominios. Son **gratis**!

caresupport **.tk**	• GRATIS	EUR 0.00	Consígalo ahora!
caresupport **.ml**			✖ No disponible
caresupport **.ga**			✖ No disponible
caresupport **.cf**	• GRATIS	EUR 0.00	Consígalo ahora!
caresupport **.gq**	• GRATIS	EUR 0.00	Consígalo ahora!

Figure 7-35. Availability of caresupport domain

The domains caresupport.ml and caresupport.ga are not available because they are registered by us for the use of this application.

In Figure 7-36 we can see the domain information and in Figure 7-37 the configuration of the domain, the type and the IP address to which it points.



Information

To the right you can find the details of your domain.
You can manage your domain using the tabs above.

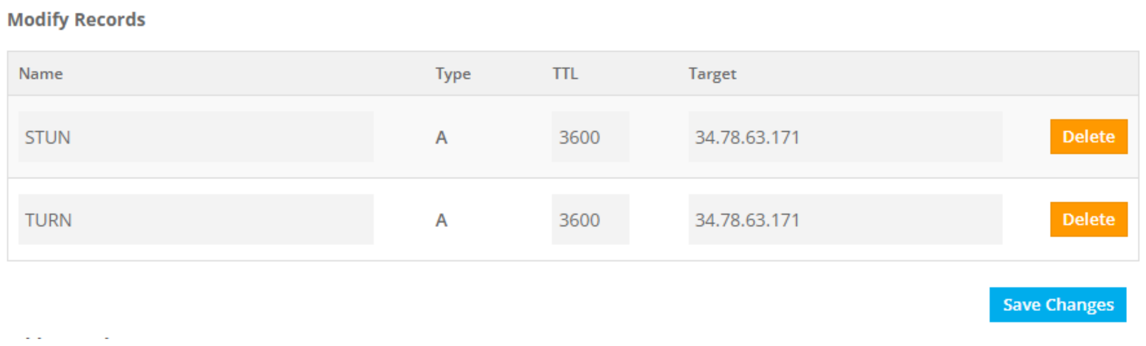
« Back to Domains List

Domain:
caresupport.ml ACTIVE

Registration Date:
07/06/2022

Expiry date:
07/06/2023

Figure 7-36. Information of caresupport domain



Modify Records

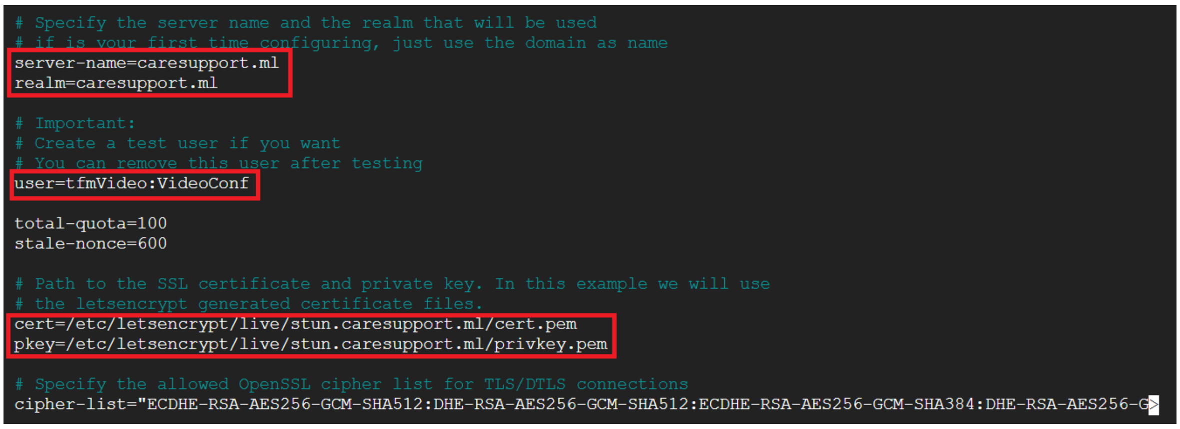
Name	Type	TTL	Target	
STUN	A	3600	34.78.63.171	Delete
TURN	A	3600	34.78.63.171	Delete

Save Changes

Figure 7-37. “caresupport” domain pointer

The last configuration step is the creation of a new configuration file in the path "/etc/turnserver.conf". The main configuration parameters of the file can be seen in Figure 7-38.

We must specify the domain of our STUN and TURN server, the user and credentials of TURN and the certificates.



```
# Specify the server name and the realm that will be used
# if is your first time configuring, just use the domain as name
server-name=caresupport.ml
realm=caresupport.ml

# Important:
# Create a test user if you want
# You can remove this user after testing
user=tfmVideo:VideoConf

total-quota=100
stale-nonce=600

# Path to the SSL certificate and private key. In this example we will use
# the letsencrypt generated certificate files.
cert=/etc/letsencrypt/live/stun.caresupport.ml/cert.pem
pkey=/etc/letsencrypt/live/stun.caresupport.ml/privkey.pem

# Specify the allowed OpenSSL cipher list for TLS/DTLS connections
cipher-list="ECDHE-RSA-AES256-GCM-SHA512:DHE-RSA-AES256-GCM-SHA512:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES256-G>
```

Figure 7-38. “turnserver.conf” file

Finally, we must start the coturn service on the server with the following command:

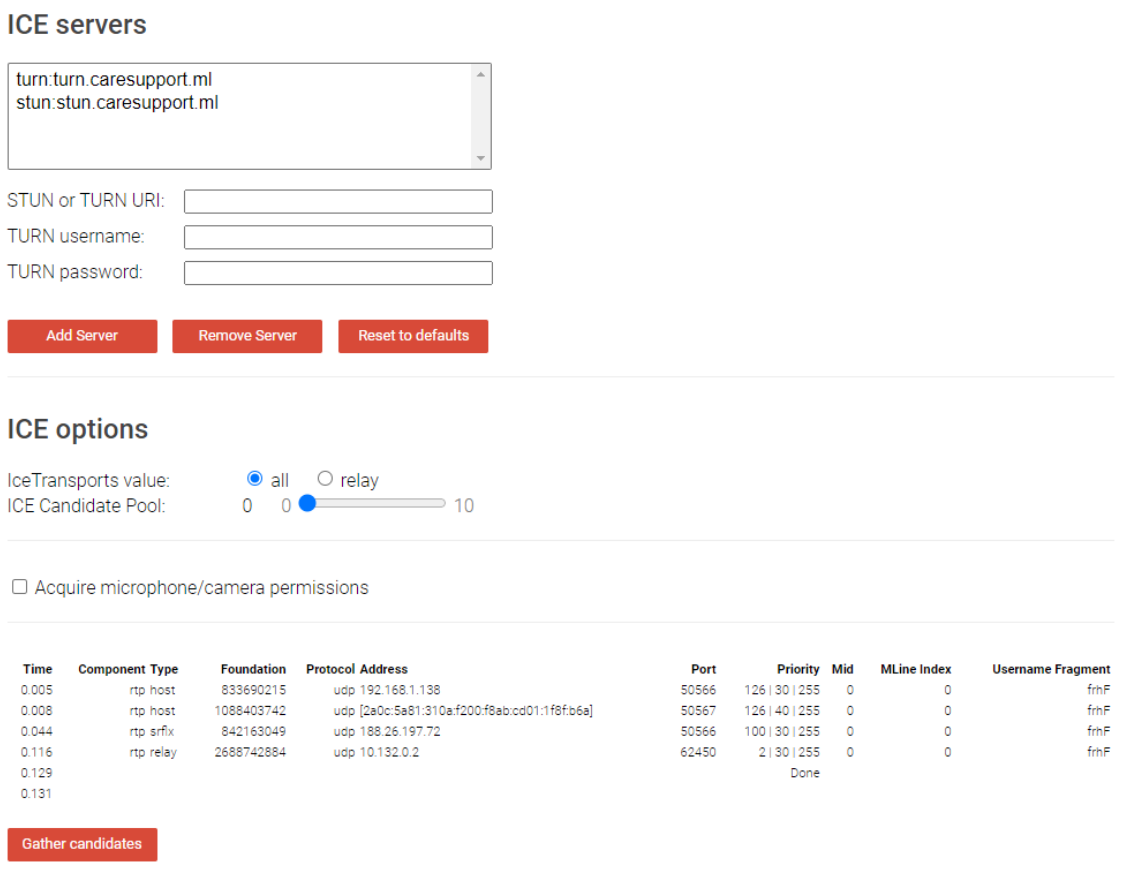
```
systemctl start coturn
```

To check the status of the coturn service we can run the following command:

```
systemctl status coturn
```

There is an online tool that allows to verify the functionality of STUN / TURN servers. This tool is Trickle ICE (https://webrtc.github.io/samples/src/content/peerconnection/trickle-ice/), a WebRTC project page that tests the ICE functionality on a regular WebRTC implementation. It creates a PeerConnection with the specified ICEServers (which will contain the information from our newly implemented server) and then starts collecting candidates for a session with a single audio stream. As the candidates are collected, they are displayed in the text box below, along with an indication when the candidate collection is complete [33].

In figure 7-39, we can see the test that has been performed against the STUN/TURN servers of the application of this project.



ICE servers

turn:turn.caresupport.ml
stun:stun.caresupport.ml

STUN or TURN URI:
TURN username:
TURN password:

ICE options

IceTransports value: [✓ all] relay
ICE Candidate Pool: 0 0 10

☐ Acquire microphone/camera permissions

Time	Component Type	Foundation	Protocol	Address	Port	Priority	Mid	MLine Index	Username Fragment
0.005	rtp host	833690215	udp	192.168.1.138	50566	126 | 30 | 255	0	0	frhF
0.008	rtp host	1088403742	udp	[2a0c:5a81:310a:f200:f8ab:cd01:1f8f:b6a]	50567	126 | 40 | 255	0	0	frhF
0.044	rtp srflx	842163049	udp	188.26.197.72	50566	100 | 30 | 255	0	0	frhF
0.116	rtp relay	2688742884	udp	10.132.0.2	62450	2 | 30 | 255	0	0	frhF
0.129						Done			
0.131									

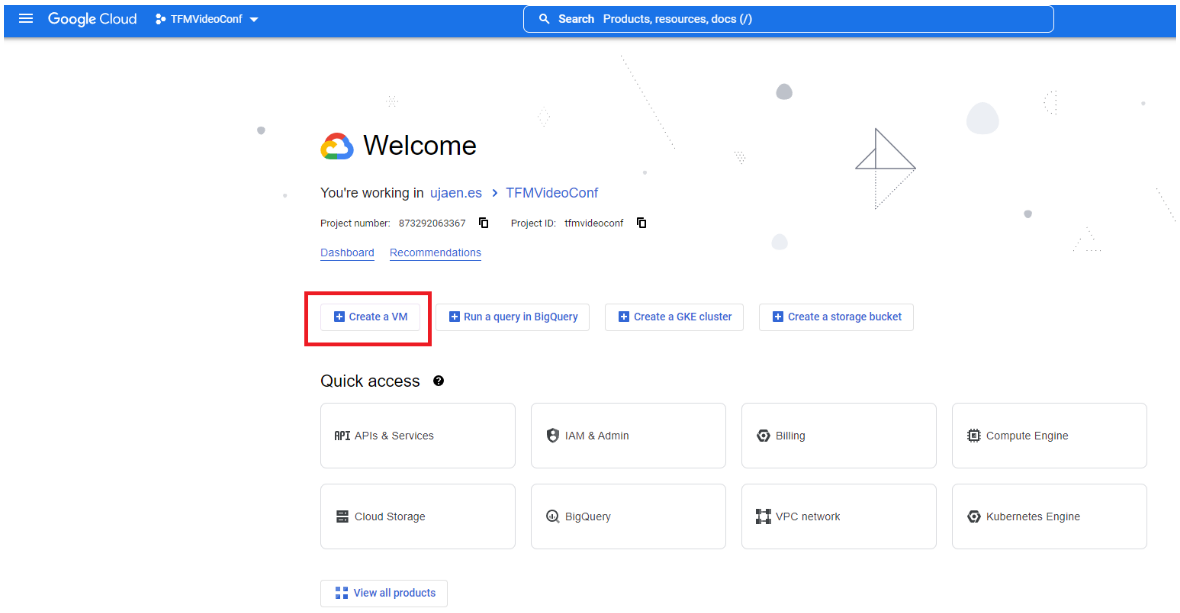
Figure 7-39. Verification of the Coturn server

7.5 Google Cloud Deployment

We have chosen to deploy this project in the Google cloud for all the reasons previously mentioned in chapter 6.5.

In view of this, the most relevant aspects of the deployment of this application on Google virtual machines will be described.

As a starting point we will have to create the virtual machine that will host our application. We enter the Google Cloud Platform page and log in with our user. On this screen we can directly see a button to create a virtual machine (Figure 7.40).



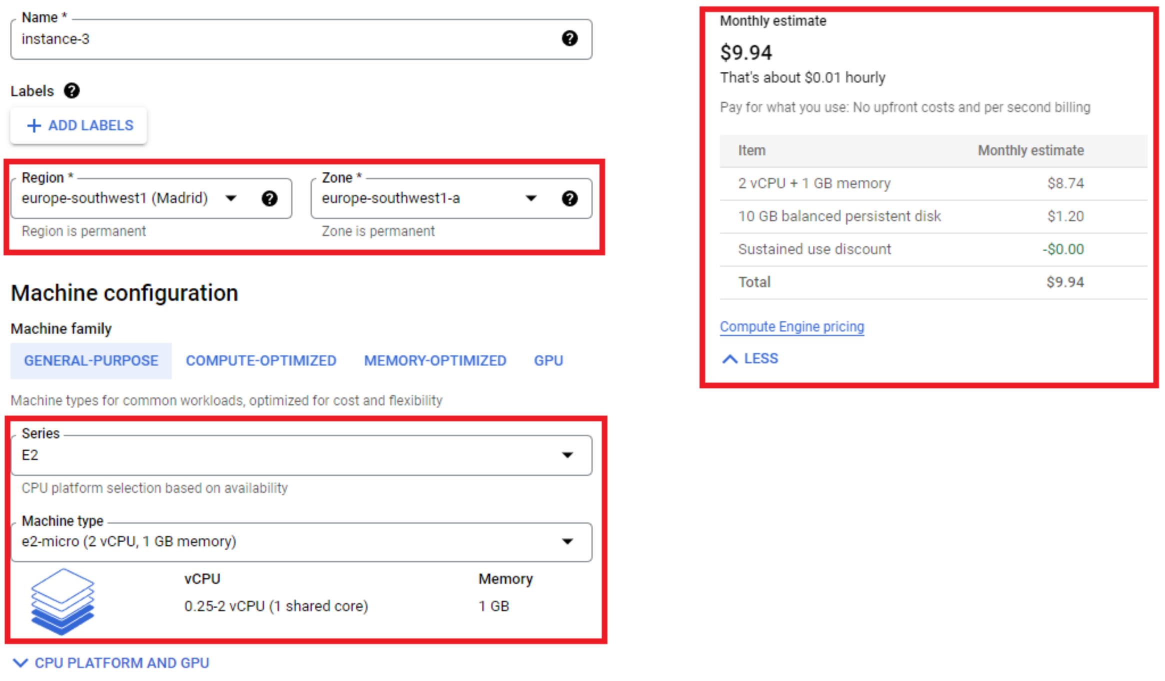
You're working in ujaen.es > TFMVideoConf

Project number: 873292063367 Project ID: tfmvideoconf

Figure 7-40. Google Cloud Platform homepage

We enter the part of selection of the characteristics of what is going to be the virtual machine. After choosing the name of the instance, we must select the region where the virtual machine will be located. It is advisable to locate it in the same country if possible, or if not, in a nearby country for latency reasons.

In the machine configuration we must choose the processor. In our case, since the application will not consume many resources, we are going to select the most basic one (e2-micro) which will be more than enough for the use we are going to give it and also, as you can see on the right side of the screen (Figure 7-41), it will reduce the cost of it.


Name *
instance-3

Labels
ADD LABELS

Region *
europe-southwest1 (Madrid)
Region is permanent

Zone *
europe-southwest1-a
Zone is permanent

Machine configuration

Machine family
GENERAL-PURPOSE COMPUTE-OPTIMIZED MEMORY-OPTIMIZED GPU

Machine types for common workloads, optimized for cost and flexibility

Series
E2
CPU platform selection based on availability

Machine type
e2-micro (2 vCPU, 1 GB memory)

vCPU	Memory
0.25-2 vCPU (1 shared core)	1 GB

CPU PLATFORM AND GPU

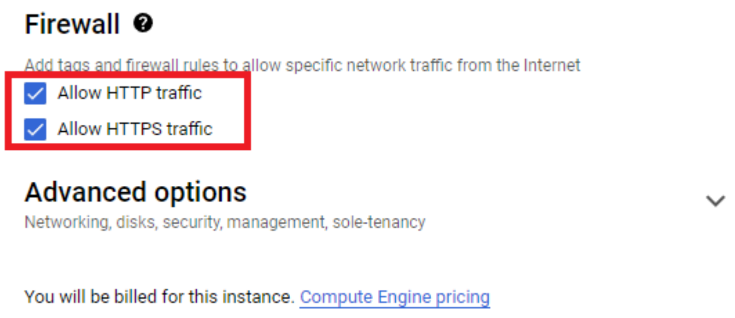
Monthly estimate
$9.94
That's about $0.01 hourly
Pay for what you use: No upfront costs and per second billing

Item	Monthly estimate
2 vCPU + 1 GB memory	$8.74
10 GB balanced persistent disk	$1.20
Sustained use discount	-$0.00
Total	$9.94

Compute Engine pricing
LESS

Figure 7-41. Choice of VM features

Another important part of the configuration is to select in the firewall to allow HTTP and HTTPS traffic (Figure 7-42).

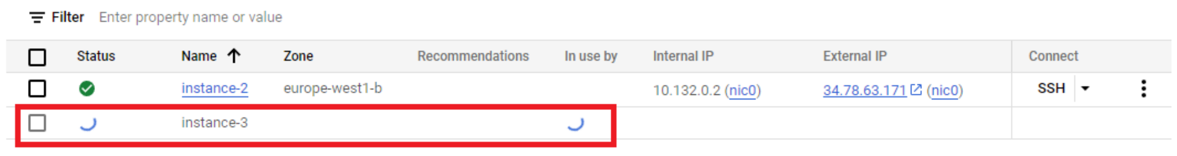

Firewall
Add tags and firewall rules to allow specific network traffic from the Internet
☑ Allow HTTP traffic
☑ Allow HTTPS traffic

Advanced options
Networking, disks, security, management, sole-tenancy

You will be billed for this instance. Compute Engine pricing

Figure 7-42. VM firewall options

Once the characteristics of the virtual machine have been selected, we proceed to click on the create button. This operation takes a couple of minutes (Figure 7-43).


Filter Enter property name or value

	Status	Name ↑	Zone	Recommendations	In use by	Internal IP	External IP	Connect
☐	✓	instance-2	europe-west1-b			10.132.0.2 (nic0)	34.78.63.171 (nic0)	SSH
☐		instance-3						

Figure 7-43. Creation of the instance

When the created instance appears, click on it and then click on the edit button (Figure 7-44) in the top bar to add the last configuration tips we need.

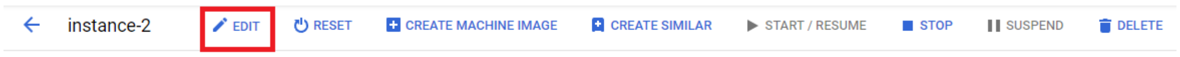


Figure 7-44. Edit button of the instance

We go down to the network options in order to set a fixed IP address. By default, the IP address that the virtual machine has is dynamic and when the server is restarted it changes. In the "External IPv4 address" box, select "CREATE IP ADDRESS" (Figure 7-45).

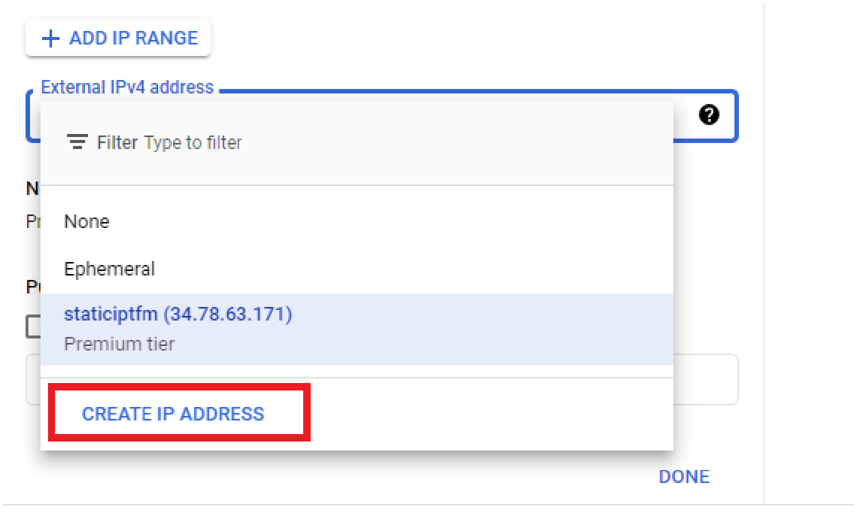
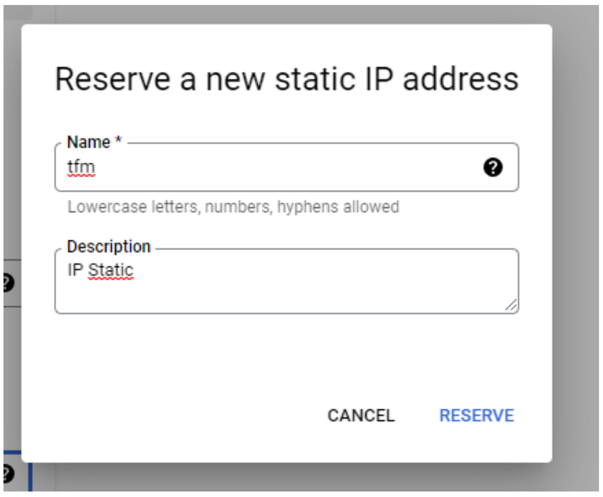


Figure 7-45. Static IP configuration

We have to define a name for this IP configuration and click on "RESERVE" (Figure 7-46), when we return to the menu, we will have already been assigned an IP address that we can see in brackets (Figure 7-47).



Reserve a new static IP address

Name * tfm

Lowercase letters, numbers, hyphens allowed

Description IP Static

Figure 7-46. Name static IP configuration

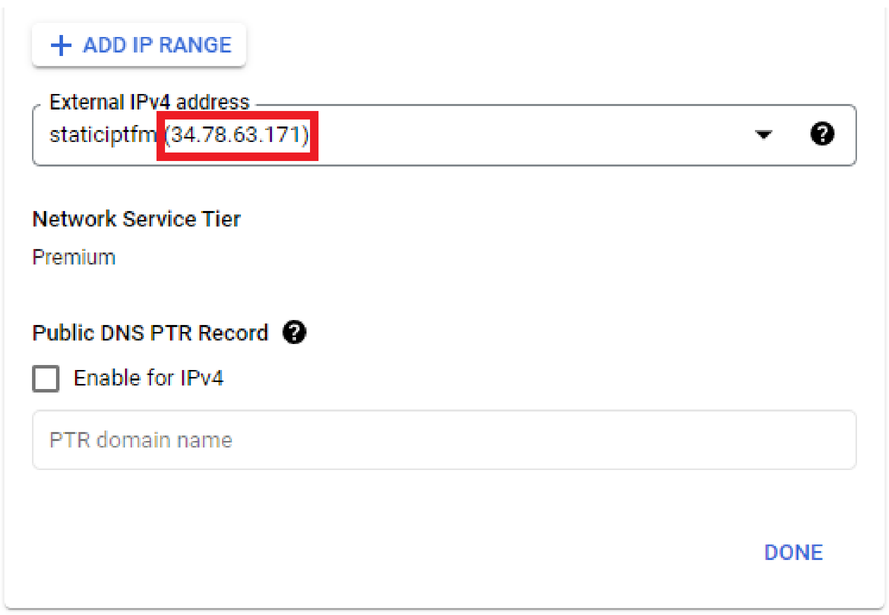
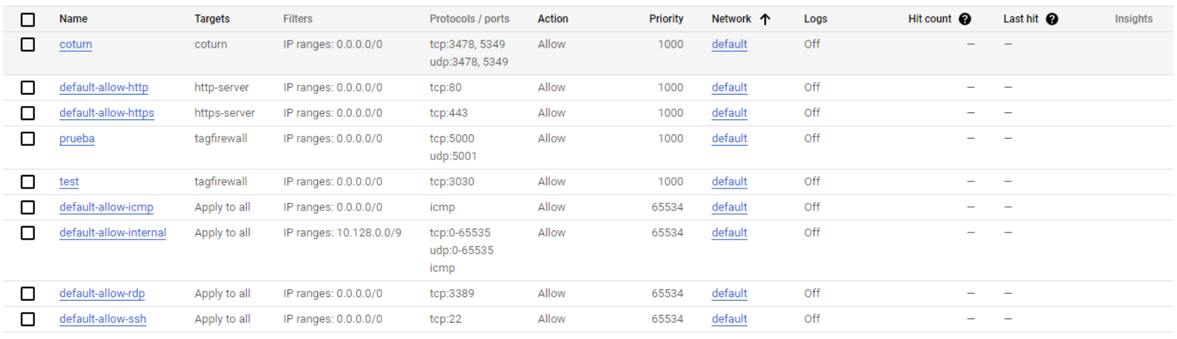


Figure 7-47. Static IP address

Once we have a static IP address, the next thing to do is to configure the firewall rules to allow traffic to our server on the ports we specify. The Coturn server, as seen in section 7.4, has been configured with ports 3478 and 5349, therefore, we have to configure a rule for this one and another one for the web server for which we will use 3030. These rules can be seen defined in figure 8-9.



Name	Targets	Filters	Protocols / ports	Action	Priority	Network	Logs	Hit count	Last hit	Insights
coturn	coturn	IP ranges: 0.0.0.0/0	tcp:3478, 5349 udp:3478, 5349	Allow	1000	default	Off	–	–	
default-allow-http	http-server	IP ranges: 0.0.0.0/0	tcp:80	Allow	1000	default	Off	–	–	
default-allow-https	https-server	IP ranges: 0.0.0.0/0	tcp:443	Allow	1000	default	Off	–	–	
prueba	tagfirewall	IP ranges: 0.0.0.0/0	tcp:5000 udp:5001	Allow	1000	default	Off	–	–	
test	tagfirewall	IP ranges: 0.0.0.0/0	tcp:3030	Allow	1000	default	Off	–	–	
default-allow-icmp	Apply to all	IP ranges: 0.0.0.0/0	icmp	Allow	65534	default	Off	–	–	
default-allow-internal	Apply to all	IP ranges: 10.128.0.0/9	tcp:0-65535 udp:0-65535 icmp	Allow	65534	default	Off	–	–	
default-allow-rdp	Apply to all	IP ranges: 0.0.0.0/0	tcp:3389	Allow	65534	default	Off	–	–	
default-allow-ssh	Apply to all	IP ranges: 0.0.0.0/0	tcp:22	Allow	65534	default	Off	–	–	

Figure 7-48. Firewall rules panel

To create these rules, simply click on "CREATE FIREWALL RULE" (Figure 8.8).

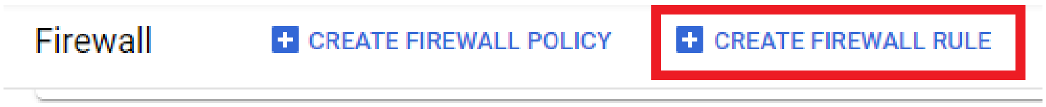
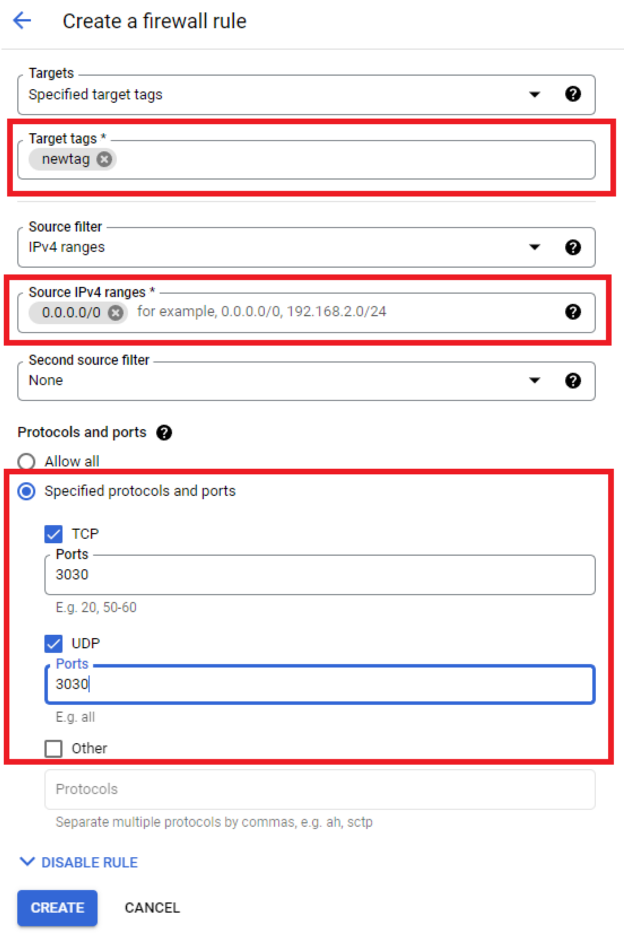


Figure 7-49. Create firewall rule button

In the form that is displayed we must select (Figure 7-50):

- A name for the rule, which we will later use so that the server knows which rules to use.
- Source addresses, which in this case will be all of them, since any user should be able to access.
- Ports of the TCP/UDP protocols that we are going to need.

Select "CREATE" and we can see the rules in the table shown above in Figure 7-48.


Create a firewall rule

Targets: Specified target tags

Target tags *: newtag

Source filter: IPv4 ranges

Source IPv4 ranges *: 0.0.0.0/0 for example, 0.0.0.0/0, 192.168.2.0/24

Second source filter: None

Protocols and ports

Allow all [✓ Specified protocols and ports]

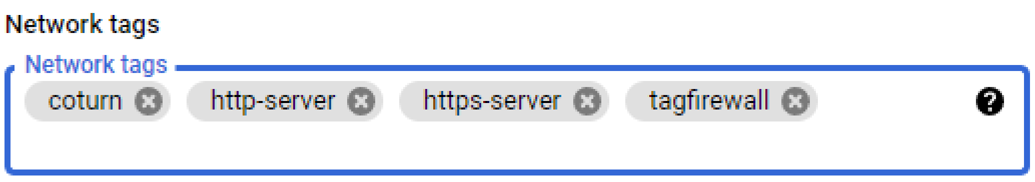
[✓] TCP
Ports: 3030
E.g. 20, 50-60

[✓] UDP
Ports: 3030
E.g. all

[ ] Other
Protocols
Separate multiple protocols by commas, e.g. ah, sctp

Figure 7-50. Form to create a firewall rule

With the firewall rules defined, we have to activate them in the virtual machine, for this we go down to the network options and in Network tags we write the name that we have given to the rules that we have created previously. For this application they have been called "coturn" and "tagfirewall" (Figure 7-51).

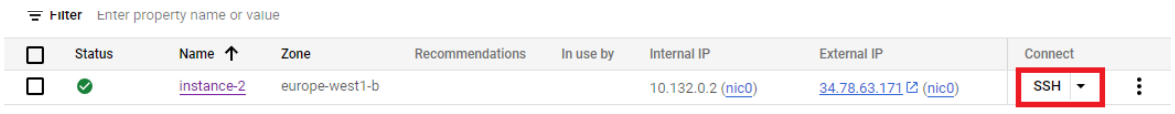

Network tags

Network tags

coturn http-server https-server tagfirewall

Figure 7-51. Firewall tags used

At this point most of the virtual machine configuration would be done. We proceed to connect via SSH to access the system terminal (Figure 7-52).

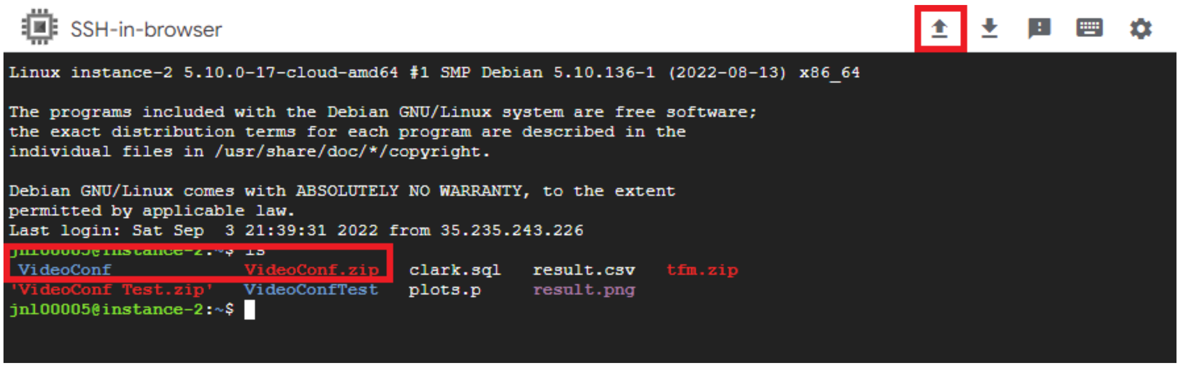

Filter Enter property name or value

	Status	Name ↑	Zone	Recommendations	In use by	Internal IP	External IP	Connect
☐	✓	instance-2	europe-west1-b			10.132.0.2 (nic0)	34.78.63.171 (nic0)	SSH

Figure 7-52. SSH button

Once the command console opens, as a recommendation, we must upload our project compressed, for example in .zip. For this we use the upload arrow located in the upper right corner. With the uploaded file in the virtual machine, we unzip it creating all the folders and files that conform the application, as we can see in figure 7-53.

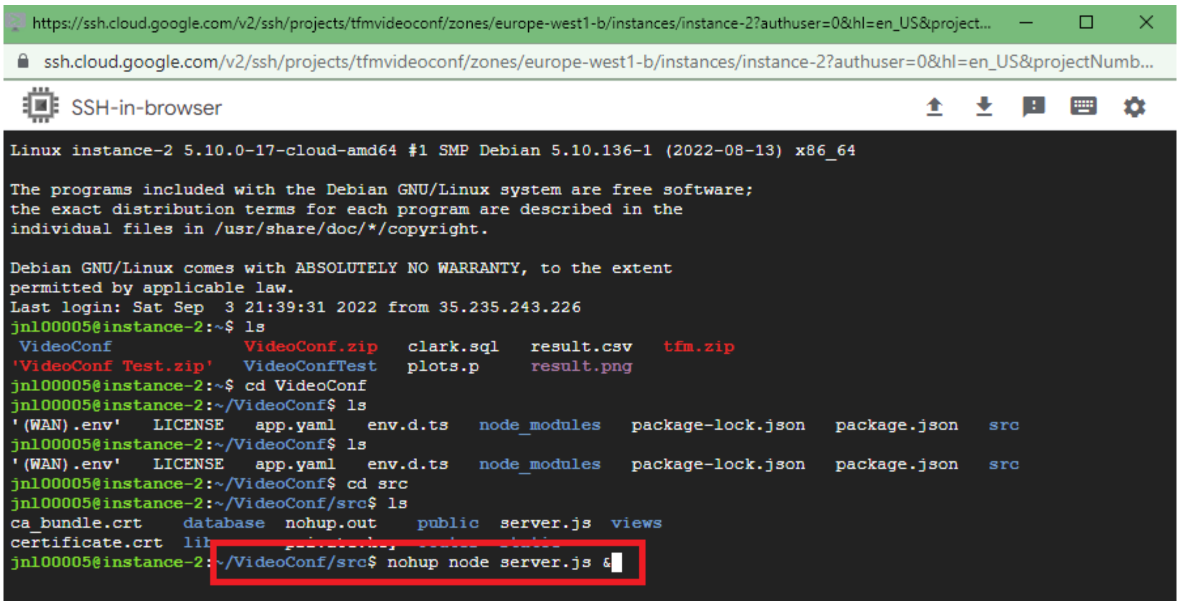
Last point we need to install Node.js in our system, this point we will treat it in Annex I not to make more extensive this section.



```
Linux instance-2 5.10.0-17-cloud-amd64 #1 SMP Debian 5.10.136-1 (2022-08-13) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sat Sep  3 21:39:31 2022 from 35.235.243.226
jnl00005@instance-2:~$ ls
 VideoConf               VideoConf.zip   clark.sql   result.csv   tfm.zip
'VideoConf Test.zip'     VideoConfTest   plots.p     result.png
jnl00005@instance-2:~$
```

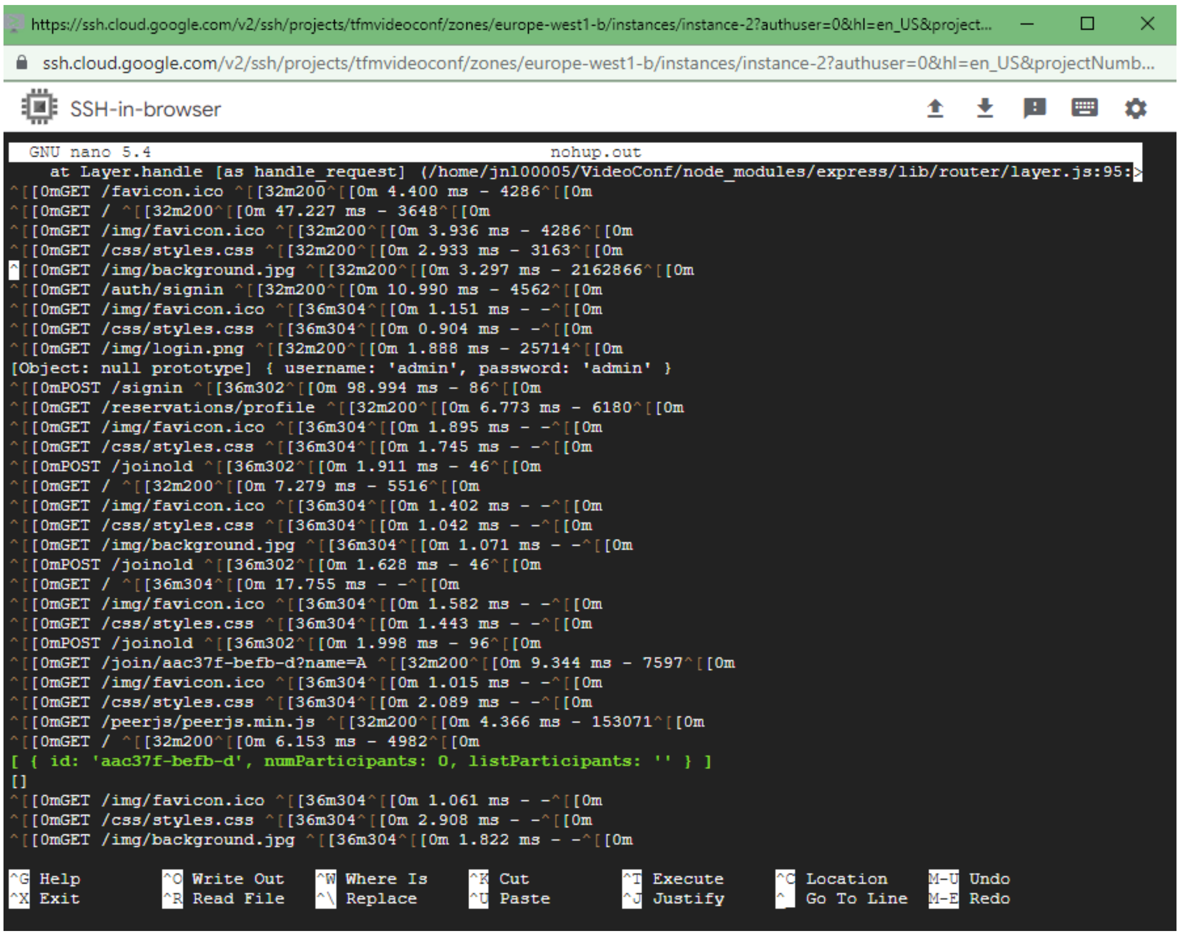
Figure 7-53. Files and upload button in SSH screen

To start the web and signaling server we are going to use the command "nohup node server.js &" (Figure 7-54) which leaves the server running in the background, allowing us to continue working with the command console elsewhere or close it without the process being interrupted. When we run the server in the background, a log called "nohup.out" is created that we can consult with any text editor (Figure 7-55).



```
Linux instance-2 5.10.0-17-cloud-amd64 #1 SMP Debian 5.10.136-1 (2022-08-13) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sat Sep  3 21:39:31 2022 from 35.235.243.226
jnl00005@instance-2:~$ ls
 VideoConf              VideoConf.zip    clark.sql    result.csv    tfm.zip
'VideoConf Test.zip'    VideoConfTest    plots.p      result.png
jnl00005@instance-2:~$ cd VideoConf
jnl00005@instance-2:~/VideoConf$ ls
'(WAN).env'   LICENSE   app.yaml   env.d.ts   node_modules   package-lock.json   package.json   src
jnl00005@instance-2:~/VideoConf$ ls
'(WAN).env'   LICENSE   app.yaml   env.d.ts   node_modules   package-lock.json   package.json   src
jnl00005@instance-2:~/VideoConf$ cd src
jnl00005@instance-2:~/VideoConf/src$ ls
ca_bundle.crt    database  nohup.out    public  server.js  views
certificate.crt  lib
jnl00005@instance-2:~/VideoConf/src$ nohup node server.js &
```

Figure 7-54. Run in background command



```
  GNU nano 5.4                                          nohup.out
    at Layer.handle [as handle_request] (/home/jnl00005/VideoConf/node_modules/express/lib/router/layer.js:95:>
^[[0mGET /favicon.ico ^[[32m200^[[0m 4.400 ms - 4286^[[0m
^[[0mGET / ^[[32m200^[[0m 47.227 ms - 3648^[[0m
^[[0mGET /img/favicon.ico ^[[32m200^[[0m 3.936 ms - 4286^[[0m
^[[0mGET /css/styles.css ^[[32m200^[[0m 2.933 ms - 3163^[[0m
^[[0mGET /img/background.jpg ^[[32m200^[[0m 3.297 ms - 2162866^[[0m
^[[0mGET /auth/signin ^[[32m200^[[0m 10.990 ms - 4562^[[0m
^[[0mGET /img/favicon.ico ^[[36m304^[[0m 1.151 ms - -^[[0m
^[[0mGET /css/styles.css ^[[36m304^[[0m 0.904 ms - -^[[0m
^[[0mGET /img/login.png ^[[32m200^[[0m 1.888 ms - 25714^[[0m
[Object: null prototype] { username: 'admin', password: 'admin' }
^[[0mPOST /signin ^[[36m302^[[0m 98.994 ms - 86^[[0m
^[[0mGET /reservations/profile ^[[32m200^[[0m 6.773 ms - 6180^[[0m
^[[0mGET /img/favicon.ico ^[[36m304^[[0m 1.895 ms - -^[[0m
^[[0mGET /css/styles.css ^[[36m304^[[0m 1.745 ms - -^[[0m
^[[0mPOST /joinold ^[[36m302^[[0m 1.911 ms - 46^[[0m
^[[0mGET / ^[[32m200^[[0m 7.279 ms - 5516^[[0m
^[[0mGET /img/favicon.ico ^[[36m304^[[0m 1.402 ms - -^[[0m
^[[0mGET /css/styles.css ^[[36m304^[[0m 1.042 ms - -^[[0m
^[[0mGET /img/background.jpg ^[[36m304^[[0m 1.071 ms - -^[[0m
^[[0mPOST /joinold ^[[36m302^[[0m 1.628 ms - 46^[[0m
^[[0mGET / ^[[36m304^[[0m 17.755 ms - -^[[0m
^[[0mGET /img/favicon.ico ^[[36m304^[[0m 1.582 ms - -^[[0m
^[[0mGET /css/styles.css ^[[36m304^[[0m 1.443 ms - -^[[0m
^[[0mPOST /joinold ^[[36m302^[[0m 1.998 ms - 96^[[0m
^[[0mGET /join/aac37f-befb-d?name=A ^[[32m200^[[0m 9.344 ms - 7597^[[0m
^[[0mGET /img/favicon.ico ^[[36m304^[[0m 1.015 ms - -^[[0m
^[[0mGET /css/styles.css ^[[36m304^[[0m 2.089 ms - -^[[0m
^[[0mGET /peerjs/peerjs.min.js ^[[32m200^[[0m 4.366 ms - 153071^[[0m
^[[0mGET / ^[[32m200^[[0m 6.153 ms - 4982^[[0m
[ { id: 'aac37f-befb-d', numParticipants: 0, listParticipants: '' } ]
[]
^[[0mGET /img/favicon.ico ^[[36m304^[[0m 1.061 ms - -^[[0m
^[[0mGET /css/styles.css ^[[36m304^[[0m 2.908 ms - -^[[0m
^[[0mGET /img/background.jpg ^[[36m304^[[0m 1.822 ms - -^[[0m

^G Help        ^O Write Out   ^W Where Is    ^K Cut         ^T Execute     ^C Location    M-U Undo
^X Exit        ^R Read File   ^\ Replace     ^U Paste       ^J Justify     ^_ Go To Line  M-E Redo
```

Figure 7-55. Nohup log

8 ANALYSIS AND DISCUSSION

The following pages detail the results obtained in the different tests and analyzes carried out on the deployed infrastructure with regard to traffic and load tests.

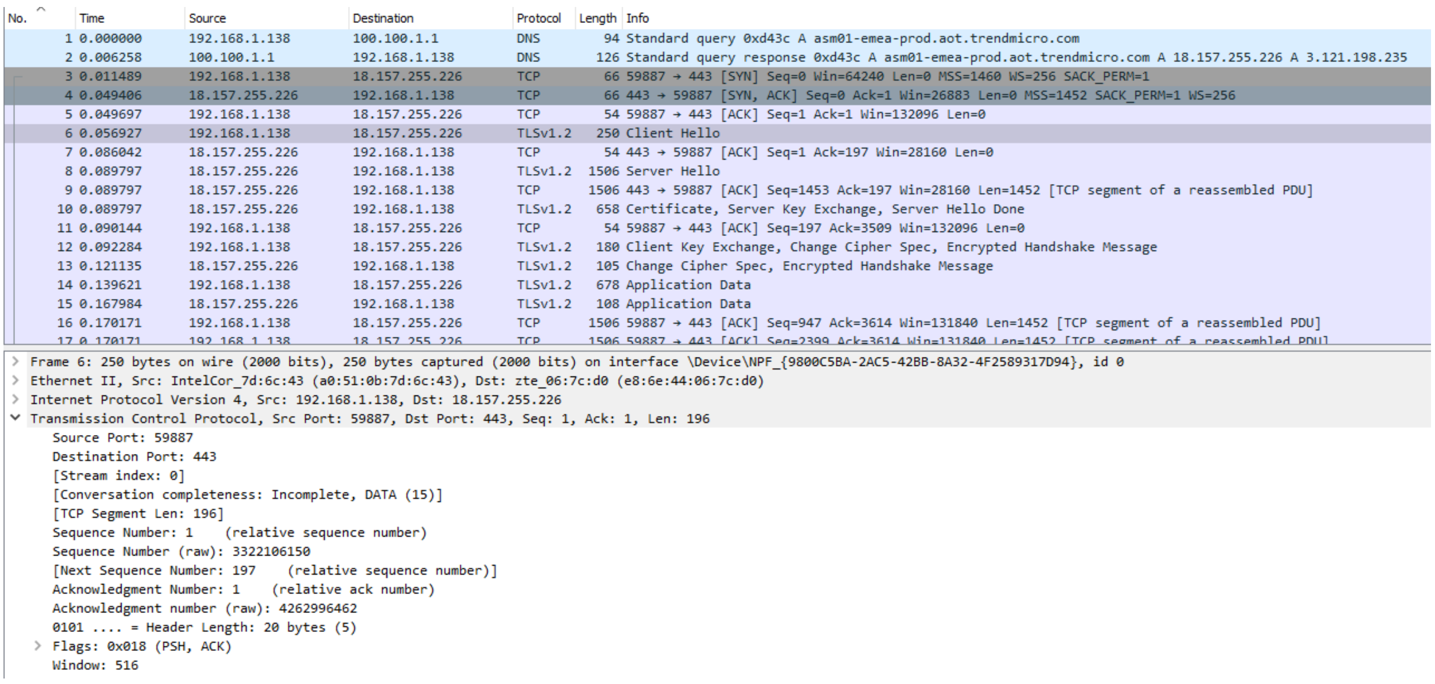
8.1 Traffic analysis

Situations of use of the web application will be defined. In this section, an analysis of the traffic generated in these different situations and communication environments will be carried out.

8.1.1 Test 1. Only three participants and everyone on the same network

In this situation, we proceed to analyze the traffic between three participants that are in the same network.

When using HTTPS, in the exchanged traffic we will not be able to see much information since everything will be encrypted (Figure 8-1). That is why the use of a secure protocol for web browsing with certain characteristics is highly recommended.

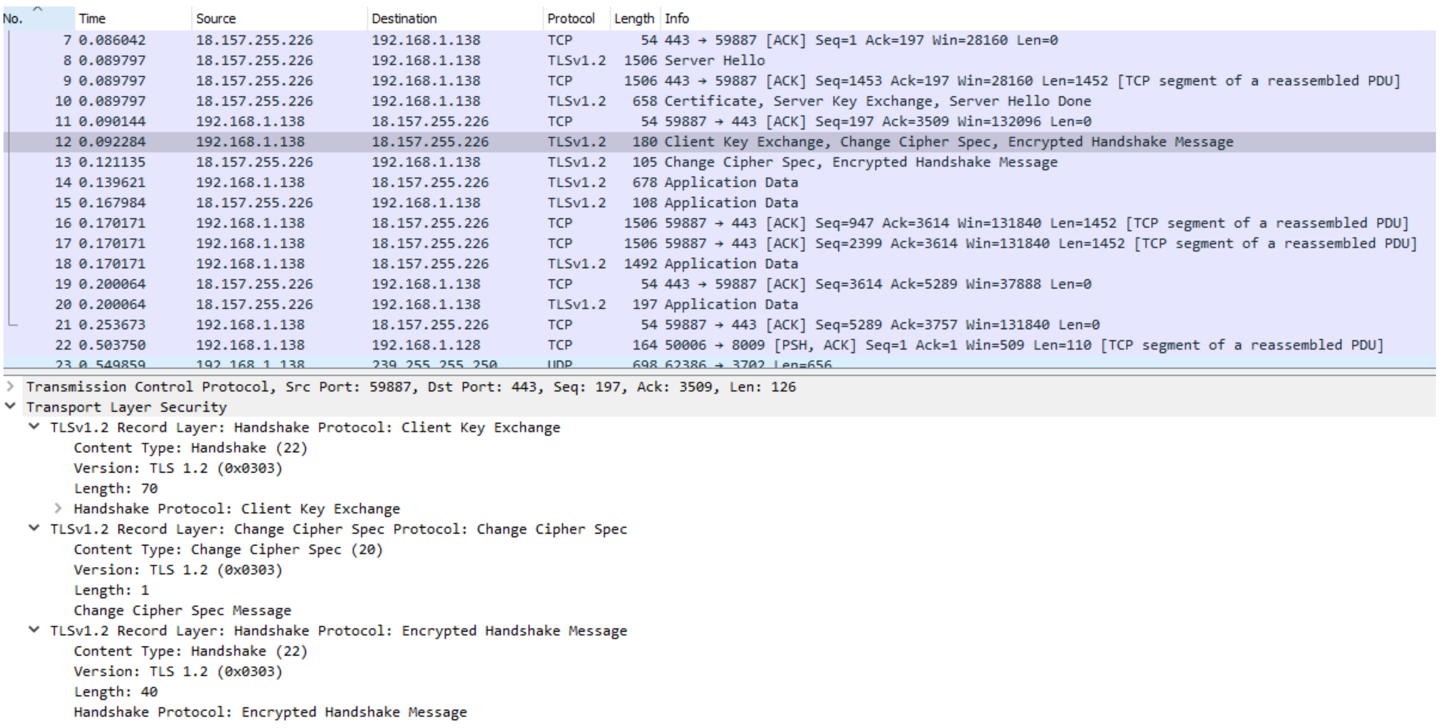


```
No.   Time       Source           Destination      Protocol Length Info
    1 0.000000   192.168.1.138    100.100.1.1      DNS        94 Standard query 0xd43c A asm01-emea-prod.aot.trendmicro.com
    2 0.006258   100.100.1.1      192.168.1.138    DNS       126 Standard query response 0xd43c A asm01-emea-prod.aot.trendmicro.com A 18.157.255.226 A 3.121.198.235
    3 0.011489   192.168.1.138    18.157.255.226   TCP        66 59887 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
    4 0.049406   18.157.255.226   192.168.1.138    TCP        66 443 → 59887 [SYN, ACK] Seq=0 Ack=1 Win=26883 Len=0 MSS=1452 SACK_PERM=1 WS=256
    5 0.049697   192.168.1.138    18.157.255.226   TCP        54 59887 → 443 [ACK] Seq=1 Ack=1 Win=132096 Len=0
    6 0.056927   192.168.1.138    18.157.255.226   TLSv1.2   250 Client Hello
    7 0.086042   18.157.255.226   192.168.1.138    TCP        54 443 → 59887 [ACK] Seq=1 Ack=197 Win=28160 Len=0
    8 0.089797   18.157.255.226   192.168.1.138    TLSv1.2  1506 Server Hello
    9 0.089797   18.157.255.226   192.168.1.138    TCP      1506 443 → 59887 [ACK] Seq=1453 Ack=197 Win=28160 Len=1452 [TCP segment of a reassembled PDU]
   10 0.089797   18.157.255.226   192.168.1.138    TLSv1.2   658 Certificate, Server Key Exchange, Server Hello Done
   11 0.090144   192.168.1.138    18.157.255.226   TCP        54 59887 → 443 [ACK] Seq=197 Ack=3509 Win=132096 Len=0
   12 0.092284   192.168.1.138    18.157.255.226   TLSv1.2   180 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
   13 0.121135   18.157.255.226   192.168.1.138    TLSv1.2   105 Change Cipher Spec, Encrypted Handshake Message
   14 0.139621   192.168.1.138    18.157.255.226   TLSv1.2   678 Application Data
   15 0.167984   18.157.255.226   192.168.1.138    TLSv1.2   108 Application Data
   16 0.170171   192.168.1.138    18.157.255.226   TCP      1506 59887 → 443 [ACK] Seq=947 Ack=3614 Win=131840 Len=1452 [TCP segment of a reassembled PDU]
   17 0.170171   192.168.1.138    18.157.255.226   TCP      1506 59887 → 443 [ACK] Seq=2399 Ack=3614 Win=131840 Len=1452 [TCP segment of a reassembled PDU]

> Frame 6: 250 bytes on wire (2000 bits), 250 bytes captured (2000 bits) on interface \Device\NPF_{9800C5BA-2AC5-42BB-8A32-4F2589317D94}, id 0
> Ethernet II, Src: IntelCor_7d:6c:43 (a0:51:0b:7d:6c:43), Dst: zte_06:7c:d0 (e8:6e:44:06:7c:d0)
> Internet Protocol Version 4, Src: 192.168.1.138, Dst: 18.157.255.226
v Transmission Control Protocol, Src Port: 59887, Dst Port: 443, Seq: 1, Ack: 1, Len: 196
    Source Port: 59887
    Destination Port: 443
    [Stream index: 0]
    [Conversation completeness: Incomplete, DATA (15)]
    [TCP Segment Len: 196]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 3322106150
    [Next Sequence Number: 197    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 4262996462
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x018 (PSH, ACK)
    Window: 516
```

Figure 8-1. HTTPS traffic

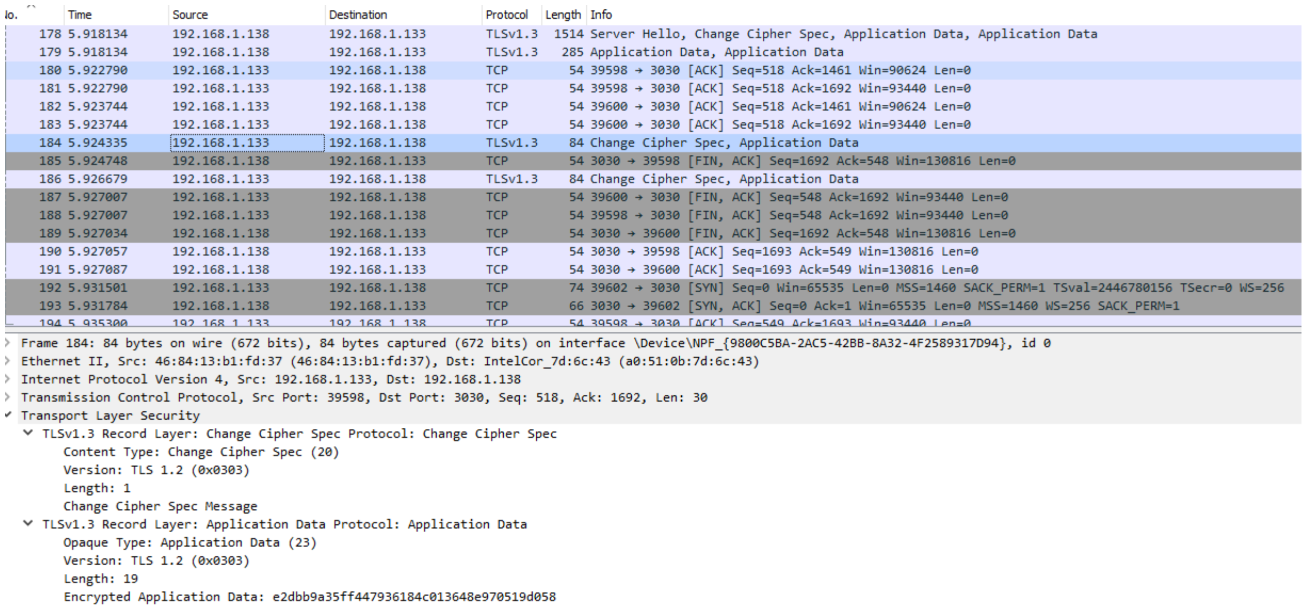
In this capture obtained when using the application at the local network level, it can be seen that it is using port 443 corresponding to HTTPS.

After this a series of messages follow with the aim of negotiating the key (Figure 8-2):



```
No.   Time       Source           Destination      Protocol Length Info
  7 0.086042     18.157.255.226   192.168.1.138    TCP        54 443 → 59887 [ACK] Seq=1 Ack=197 Win=28160 Len=0
  8 0.089797     18.157.255.226   192.168.1.138    TLSv1.2  1506 Server Hello
  9 0.089797     18.157.255.226   192.168.1.138    TCP      1506 443 → 59887 [ACK] Seq=1453 Ack=197 Win=28160 Len=1452 [TCP segment of a reassembled PDU]
 10 0.089797     18.157.255.226   192.168.1.138    TLSv1.2   658 Certificate, Server Key Exchange, Server Hello Done
 11 0.090144     192.168.1.138    18.157.255.226   TCP        54 59887 → 443 [ACK] Seq=197 Ack=3509 Win=132096 Len=0
 12 0.092284     192.168.1.138    18.157.255.226   TLSv1.2   180 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
 13 0.121135     18.157.255.226   192.168.1.138    TLSv1.2   105 Change Cipher Spec, Encrypted Handshake Message
 14 0.139621     192.168.1.138    18.157.255.226   TLSv1.2   678 Application Data
 15 0.167984     18.157.255.226   192.168.1.138    TLSv1.2   108 Application Data
 16 0.170171     192.168.1.138    18.157.255.226   TCP      1506 59887 → 443 [ACK] Seq=947 Ack=3614 Win=131840 Len=1452 [TCP segment of a reassembled PDU]
 17 0.170171     192.168.1.138    18.157.255.226   TCP      1506 59887 → 443 [ACK] Seq=2399 Ack=3614 Win=131840 Len=1452 [TCP segment of a reassembled PDU]
 18 0.170171     192.168.1.138    18.157.255.226   TLSv1.2  1492 Application Data
 19 0.200064     18.157.255.226   192.168.1.138    TCP        54 443 → 59887 [ACK] Seq=3614 Ack=5289 Win=37888 Len=0
 20 0.200064     18.157.255.226   192.168.1.138    TLSv1.2   197 Application Data
 21 0.253673     192.168.1.138    18.157.255.226   TCP        54 59887 → 443 [ACK] Seq=5289 Ack=3757 Win=131840 Len=0
 22 0.503750     192.168.1.138    192.168.1.128    TCP       164 50006 → 8009 [PSH, ACK] Seq=1 Ack=1 Win=509 Len=110 [TCP segment of a reassembled PDU]
 23 0.549859     192.168.1.138    239.255.255.250  UDP       698 62386 → 3702 Len=656
> Transmission Control Protocol, Src Port: 59887, Dst Port: 443, Seq: 197, Ack: 3509, Len: 126
v Transport Layer Security
  v TLSv1.2 Record Layer: Handshake Protocol: Client Key Exchange
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 70
    > Handshake Protocol: Client Key Exchange
  v TLSv1.2 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
      Content Type: Change Cipher Spec (20)
      Version: TLS 1.2 (0x0303)
      Length: 1
      Change Cipher Spec Message
  v TLSv1.2 Record Layer: Handshake Protocol: Encrypted Handshake Message
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 40
      Handshake Protocol: Encrypted Handshake Message
```

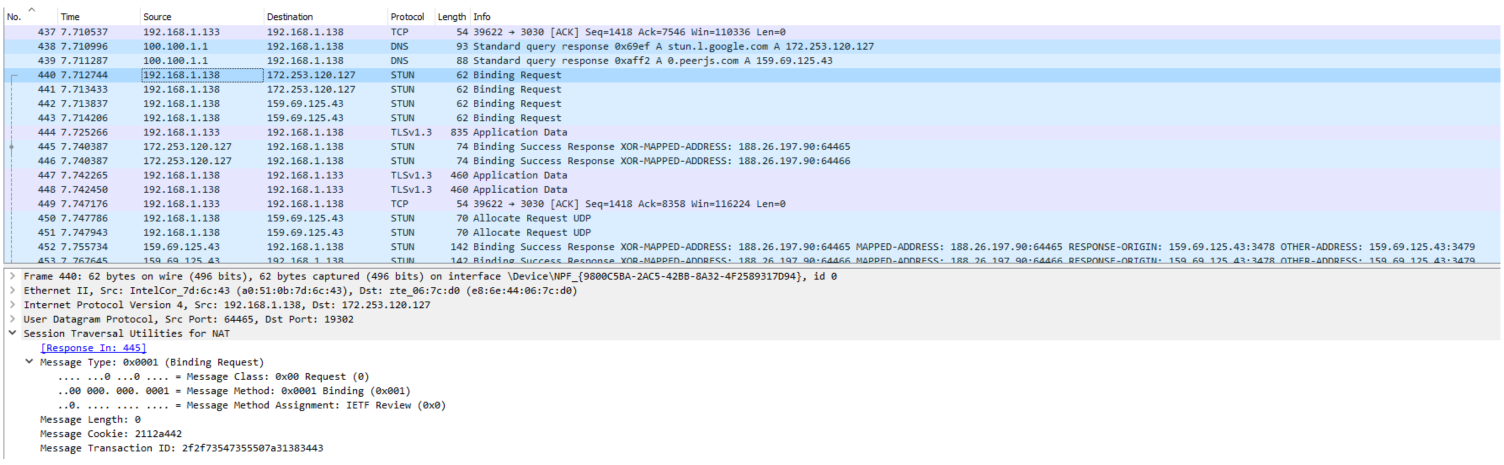
Figure 8-2. Negotiating key exchange



```
No.   Time       Source           Destination      Protocol Length Info
178 5.918134     192.168.1.138    192.168.1.133    TLSv1.3  1514 Server Hello, Change Cipher Spec, Application Data, Application Data
179 5.918134     192.168.1.138    192.168.1.133    TLSv1.3   285 Application Data, Application Data
180 5.922790     192.168.1.133    192.168.1.138    TCP        54 39598 → 3030 [ACK] Seq=518 Ack=1461 Win=90624 Len=0
181 5.922790     192.168.1.133    192.168.1.138    TCP        54 39598 → 3030 [ACK] Seq=518 Ack=1692 Win=93440 Len=0
182 5.923744     192.168.1.133    192.168.1.138    TCP        54 39600 → 3030 [ACK] Seq=518 Ack=1461 Win=90624 Len=0
183 5.923744     192.168.1.133    192.168.1.138    TCP        54 39600 → 3030 [ACK] Seq=518 Ack=1692 Win=93440 Len=0
184 5.924335     192.168.1.133    192.168.1.138    TLSv1.3    84 Change Cipher Spec, Application Data
185 5.924748     192.168.1.138    192.168.1.133    TCP        54 3030 → 39598 [FIN, ACK] Seq=1692 Ack=548 Win=130816 Len=0
186 5.926679     192.168.1.133    192.168.1.138    TLSv1.3    84 Change Cipher Spec, Application Data
187 5.927007     192.168.1.133    192.168.1.138    TCP        54 39600 → 3030 [FIN, ACK] Seq=548 Ack=1692 Win=93440 Len=0
188 5.927007     192.168.1.133    192.168.1.138    TCP        54 39598 → 3030 [FIN, ACK] Seq=548 Ack=1692 Win=93440 Len=0
189 5.927034     192.168.1.138    192.168.1.133    TCP        54 3030 → 39600 [FIN, ACK] Seq=1692 Ack=548 Win=130816 Len=0
190 5.927057     192.168.1.138    192.168.1.133    TCP        54 3030 → 39598 [ACK] Seq=1693 Ack=549 Win=130816 Len=0
191 5.927087     192.168.1.138    192.168.1.133    TCP        54 3030 → 39600 [ACK] Seq=1693 Ack=549 Win=130816 Len=0
192 5.931501     192.168.1.133    192.168.1.138    TCP        74 39602 → 3030 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1 TSval=2446780156 TSecr=0 WS=256
193 5.931784     192.168.1.138    192.168.1.133    TCP        66 3030 → 39602 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
194 5.935300     192.168.1.133    192.168.1.138    TCP        54 39598 → 3030 [ACK] Seq=549 Ack=1693 Win=93440 Len=0
> Frame 184: 84 bytes on wire (672 bits), 84 bytes captured (672 bits) on interface \Device\NPF_{9800C5BA-2AC5-42BB-8A32-4F2589317D94}, id 0
> Ethernet II, Src: 46:84:13:b1:fd:37 (46:84:13:b1:fd:37), Dst: IntelCor_7d:6c:43 (a0:51:0b:7d:6c:43)
> Internet Protocol Version 4, Src: 192.168.1.133, Dst: 192.168.1.138
> Transmission Control Protocol, Src Port: 39598, Dst Port: 3030, Seq: 518, Ack: 1692, Len: 30
v Transport Layer Security
  v TLSv1.3 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
      Content Type: Change Cipher Spec (20)
      Version: TLS 1.2 (0x0303)
      Length: 1
      Change Cipher Spec Message
  v TLSv1.3 Record Layer: Application Data Protocol: Application Data
      Opaque Type: Application Data (23)
      Version: TLS 1.2 (0x0303)
      Length: 19
      Encrypted Application Data: e2dbb9a35ff447936184c013648e970519d058
```

Figure 8-3. Establishing the connection

Once communication is established:



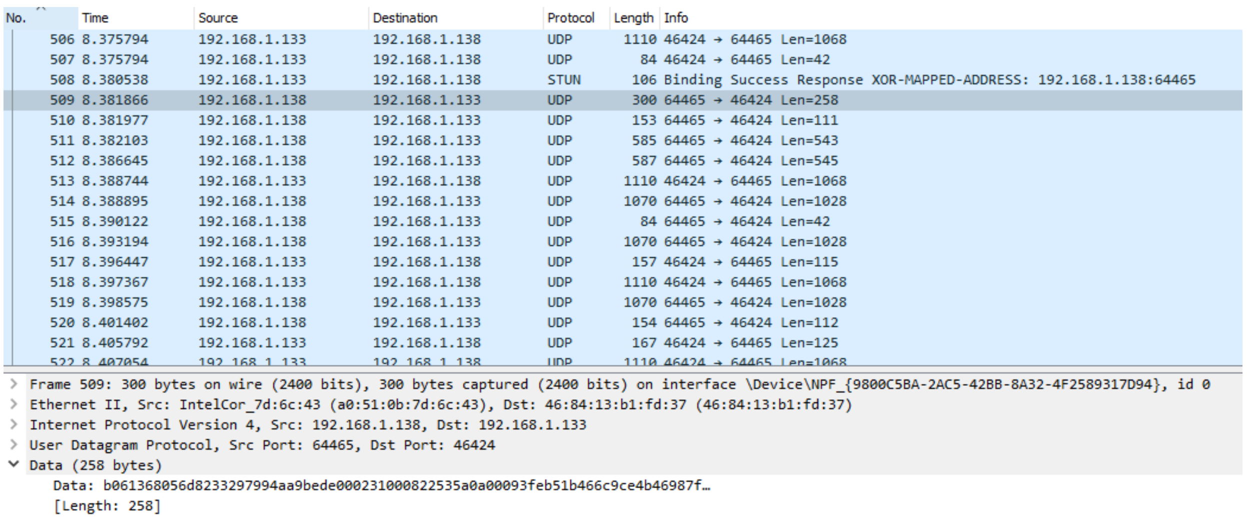
```
No.   Time       Source           Destination      Protocol Length Info
  437 7.710537   192.168.1.133    192.168.1.138    TCP        54 39622 → 3030 [ACK] Seq=1418 Ack=7546 Win=110336 Len=0
  438 7.710996   100.100.1.1      192.168.1.138    DNS        93 Standard query response 0x69ef A stun.l.google.com A 172.253.120.127
  439 7.711287   100.100.1.1      192.168.1.138    DNS        88 Standard query response 0xaff2 A 0.peerjs.com A 159.69.125.43
  440 7.712744   192.168.1.138    172.253.120.127  STUN       62 Binding Request
  441 7.713433   192.168.1.138    172.253.120.127  STUN       62 Binding Request
  442 7.713837   192.168.1.138    159.69.125.43    STUN       62 Binding Request
  443 7.714206   192.168.1.138    159.69.125.43    STUN       62 Binding Request
  444 7.725266   192.168.1.133    192.168.1.138    TLSv1.3   835 Application Data
  445 7.740387   172.253.120.127  192.168.1.138    STUN       74 Binding Success Response XOR-MAPPED-ADDRESS: 188.26.197.90:64465
  446 7.740387   172.253.120.127  192.168.1.138    STUN       74 Binding Success Response XOR-MAPPED-ADDRESS: 188.26.197.90:64466
  447 7.742265   192.168.1.138    192.168.1.133    TLSv1.3   460 Application Data
  448 7.742450   192.168.1.138    192.168.1.133    TLSv1.3   460 Application Data
  449 7.747176   192.168.1.133    192.168.1.138    TCP        54 39622 → 3030 [ACK] Seq=1418 Ack=8358 Win=116224 Len=0
  450 7.747786   192.168.1.138    159.69.125.43    STUN       70 Allocate Request UDP
  451 7.747943   192.168.1.138    159.69.125.43    STUN       70 Allocate Request UDP
  452 7.755734   159.69.125.43    192.168.1.138    STUN      142 Binding Success Response XOR-MAPPED-ADDRESS: 188.26.197.90:64465 MAPPED-ADDRESS: 188.26.197.90:64465 RESPONSE-ORIGIN: 159.69.125.43:3478 OTHER-ADDRESS: 159.69.125.43:3479
  453 7.767645   159.69.125.43    192.168.1.138    STUN      142 Binding Success Response XOR-MAPPED-ADDRESS: 188.26.197.90:64466 MAPPED-ADDRESS: 188.26.197.90:64466 RESPONSE-ORIGIN: 159.69.125.43:3478 OTHER-ADDRESS: 159.69.125.43:3479
```

```
> Frame 440: 62 bytes on wire (496 bits), 62 bytes captured (496 bits) on interface \Device\NPF_{9800C5BA-2AC5-42BB-8A32-4F2589317D94}, id 0
> Ethernet II, Src: IntelCor_7d:6c:43 (a0:51:0b:7d:6c:43), Dst: zte_06:7c:d0 (e8:6e:44:06:7c:d0)
> Internet Protocol Version 4, Src: 192.168.1.138, Dst: 172.253.120.127
> User Datagram Protocol, Src Port: 64465, Dst Port: 19302
v Session Traversal Utilities for NAT
    [Response In: 445]
  v Message Type: 0x0001 (Binding Request)
      .... ...0 ...0 .... = Message Class: 0x00 Request (0)
      ..00 000. 000. 0001 = Message Method: 0x0001 Binding (0x001)
      ..0. .... .... .... = Message Method Assignment: IETF Review (0x0)
    Message Length: 0
    Message Cookie: 2112a442
    Message Transaction ID: 2f2f73547355507a31383443
```

Figure 8-4. STUN IP discovery

We see how the STUN protocol begins to work until it obtains confirmation of the IP discovery from the other end (Figure 8-4):

And finally, we can see that all the traffic exchanged goes over UDP (Figure 8-5), and being in local network, the connection is direct between the two ends. You are also using the STUN server since it was configured in the development of the application.



```
No.   Time       Source           Destination      Protocol Length Info
  506 8.375794   192.168.1.133    192.168.1.138    UDP      1110 46424 → 64465 Len=1068
  507 8.375794   192.168.1.133    192.168.1.138    UDP        84 46424 → 64465 Len=42
  508 8.380538   192.168.1.133    192.168.1.138    STUN      106 Binding Success Response XOR-MAPPED-ADDRESS: 192.168.1.138:64465
  509 8.381866   192.168.1.138    192.168.1.133    UDP       300 64465 → 46424 Len=258
  510 8.381977   192.168.1.138    192.168.1.133    UDP       153 64465 → 46424 Len=111
  511 8.382103   192.168.1.138    192.168.1.133    UDP       585 64465 → 46424 Len=543
  512 8.386645   192.168.1.138    192.168.1.133    UDP       587 64465 → 46424 Len=545
  513 8.388744   192.168.1.133    192.168.1.138    UDP      1110 46424 → 64465 Len=1068
  514 8.388895   192.168.1.138    192.168.1.133    UDP      1070 64465 → 46424 Len=1028
  515 8.390122   192.168.1.138    192.168.1.133    UDP        84 64465 → 46424 Len=42
  516 8.393194   192.168.1.138    192.168.1.133    UDP      1070 64465 → 46424 Len=1028
  517 8.396447   192.168.1.133    192.168.1.138    UDP       157 46424 → 64465 Len=115
  518 8.397367   192.168.1.133    192.168.1.138    UDP      1110 46424 → 64465 Len=1068
  519 8.398575   192.168.1.138    192.168.1.133    UDP      1070 64465 → 46424 Len=1028
  520 8.401402   192.168.1.138    192.168.1.133    UDP       154 64465 → 46424 Len=112
  521 8.405792   192.168.1.133    192.168.1.138    UDP       167 46424 → 64465 Len=125
  522 8.407054   192.168.1.133    192.168.1.138    UDP      1110 46424 → 64465 Len=1068
```

```
> Frame 509: 300 bytes on wire (2400 bits), 300 bytes captured (2400 bits) on interface \Device\NPF_{9800C5BA-2AC5-42BB-8A32-4F2589317D94}, id 0
> Ethernet II, Src: IntelCor_7d:6c:43 (a0:51:0b:7d:6c:43), Dst: 46:84:13:b1:fd:37 (46:84:13:b1:fd:37)
> Internet Protocol Version 4, Src: 192.168.1.138, Dst: 192.168.1.133
> User Datagram Protocol, Src Port: 64465, Dst Port: 46424
v Data (258 bytes)
    Data: b061368056d8233297994aa9bede000231000822535a0a00093feb51b466c9ce4b46987f…
    [Length: 258]
```

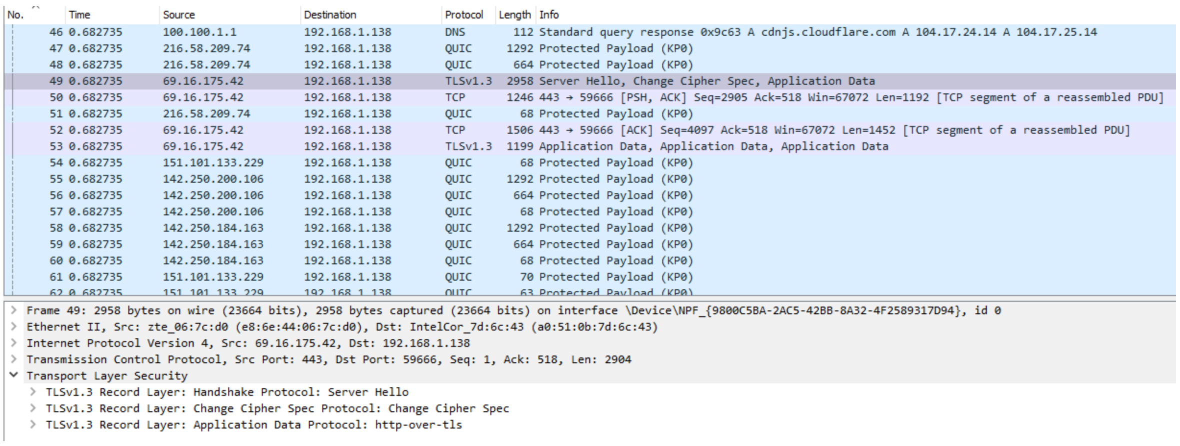
Figure 8-5. Data traffic over UDP

In this example, the peer.js constructor was configured with Google's STUN server. It can be seen in the figure 8-3.

8.1.2 Test 1. Only three participants but one requires the use of the TURN server

In two different LANs the scenario changes, in this case, we have a computer connected to a network with a restrictive policy, a computer connected to a router with internet access at home and a mobile phone with a 5G network as t the WAN to provide internet access.

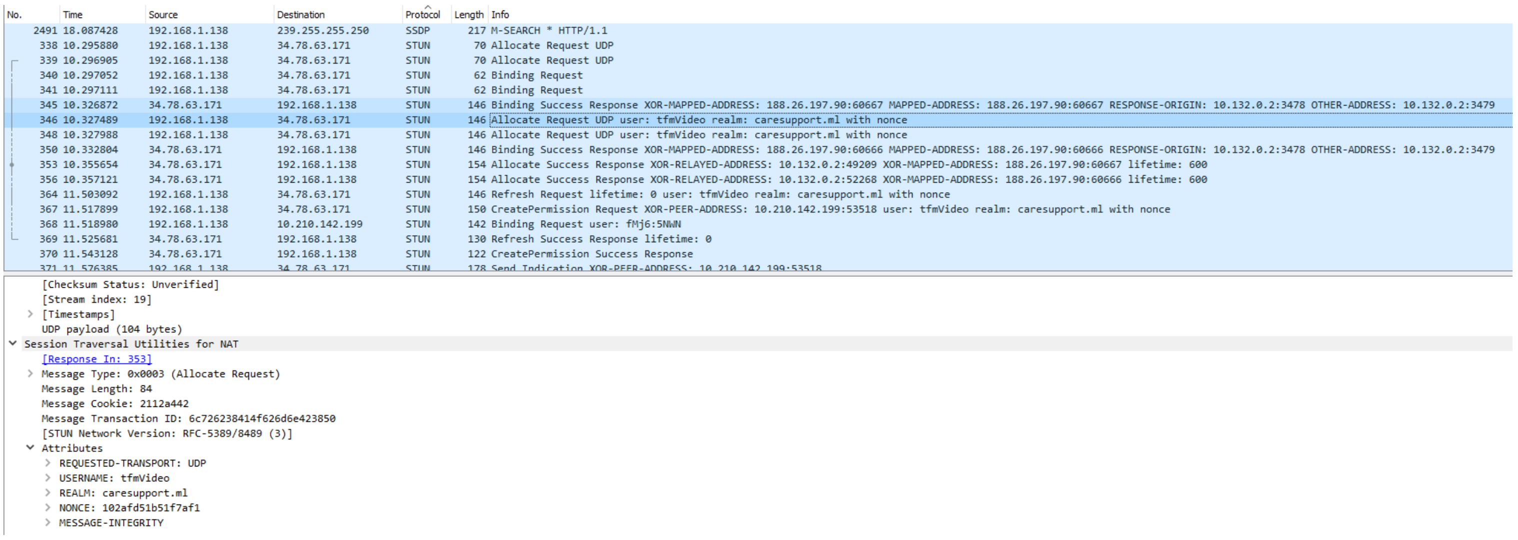
The part of negotiating the key does not change:



```
No.    Time      Source           Destination     Protocol Length Info
  46 0.682735   100.100.1.1      192.168.1.138   DNS       112 Standard query response 0x9c63 A cdnjs.cloudflare.com A 104.17.24.14 A 104.17.25.14
  47 0.682735   216.58.209.74    192.168.1.138   QUIC     1292 Protected Payload (KP0)
  48 0.682735   216.58.209.74    192.168.1.138   QUIC      664 Protected Payload (KP0)
  49 0.682735   69.16.175.42     192.168.1.138   TLSv1.3  2958 Server Hello, Change Cipher Spec, Application Data
  50 0.682735   69.16.175.42     192.168.1.138   TCP      1246 443 → 59666 [PSH, ACK] Seq=2905 Ack=518 Win=67072 Len=1192 [TCP segment of a reassembled PDU]
  51 0.682735   216.58.209.74    192.168.1.138   QUIC       68 Protected Payload (KP0)
  52 0.682735   69.16.175.42     192.168.1.138   TCP      1506 443 → 59666 [ACK] Seq=4097 Ack=518 Win=67072 Len=1452 [TCP segment of a reassembled PDU]
  53 0.682735   69.16.175.42     192.168.1.138   TLSv1.3  1199 Application Data, Application Data
  54 0.682735   151.101.133.229  192.168.1.138   QUIC       68 Protected Payload (KP0)
  55 0.682735   142.250.200.106  192.168.1.138   QUIC     1292 Protected Payload (KP0)
  56 0.682735   142.250.200.106  192.168.1.138   QUIC      664 Protected Payload (KP0)
  57 0.682735   142.250.200.106  192.168.1.138   QUIC       68 Protected Payload (KP0)
  58 0.682735   142.250.184.163  192.168.1.138   QUIC     1292 Protected Payload (KP0)
  59 0.682735   142.250.184.163  192.168.1.138   QUIC      664 Protected Payload (KP0)
  60 0.682735   142.250.184.163  192.168.1.138   QUIC       68 Protected Payload (KP0)
  61 0.682735   151.101.133.229  192.168.1.138   QUIC       70 Protected Payload (KP0)
  62 0.682735   151.101.133.229  192.168.1.138   QUIC       63 Protected Payload (KP0)

> Frame 49: 2958 bytes on wire (23664 bits), 2958 bytes captured (23664 bits) on interface \Device\NPF_{9800C5BA-2AC5-42BB-8A32-4F2589317D94}, id 0
> Ethernet II, Src: zte_06:7c:d0 (e8:6e:44:06:7c:d0), Dst: IntelCor_7d:6c:43 (a0:51:0b:7d:6c:43)
> Internet Protocol Version 4, Src: 69.16.175.42, Dst: 192.168.1.138
> Transmission Control Protocol, Src Port: 443, Dst Port: 59666, Seq: 1, Ack: 518, Len: 2904
v Transport Layer Security
   > TLSv1.3 Record Layer: Handshake Protocol: Server Hello
   > TLSv1.3 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
   > TLSv1.3 Record Layer: Application Data Protocol: http-over-tls
```

Figure 8-6. Negotiating key exchange

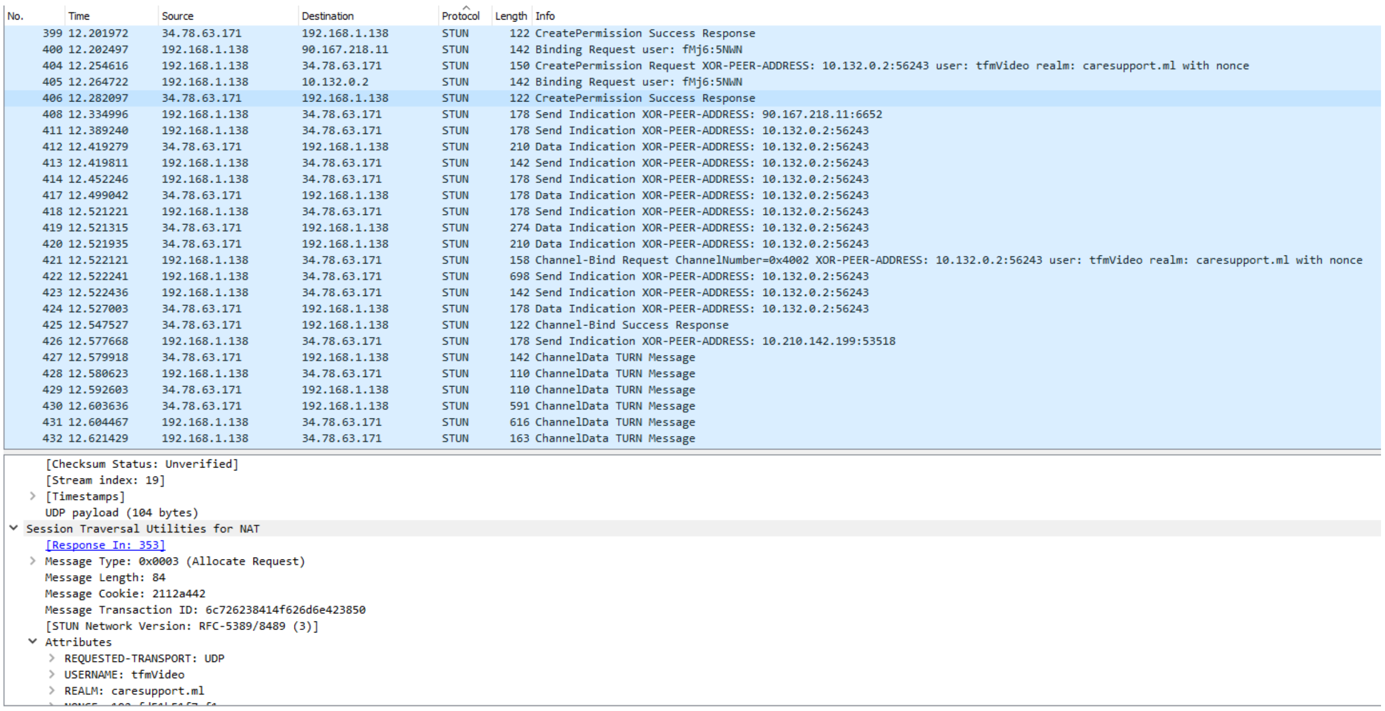
But now, STUN cannot discover the IP address of the ends and the appearance of TURN is necessary (Figure 8-7):



```
No.     Time       Source          Destination      Protocol Length Info
  2491 18.087428  192.168.1.138   239.255.255.250  SSDP      217 M-SEARCH * HTTP/1.1
   338 10.295880  192.168.1.138   34.78.63.171     STUN       70 Allocate Request UDP
   339 10.296905  192.168.1.138   34.78.63.171     STUN       70 Allocate Request UDP
   340 10.297052  192.168.1.138   34.78.63.171     STUN       62 Binding Request
   341 10.297111  192.168.1.138   34.78.63.171     STUN       62 Binding Request
   345 10.326872  34.78.63.171    192.168.1.138    STUN      146 Binding Success Response XOR-MAPPED-ADDRESS: 188.26.197.90:60667 MAPPED-ADDRESS: 188.26.197.90:60667 RESPONSE-ORIGIN: 10.132.0.2:3478 OTHER-ADDRESS: 10.132.0.2:3479
   346 10.327489  192.168.1.138   34.78.63.171     STUN      146 Allocate Request UDP user: tfmVideo realm: caresupport.ml with nonce
   348 10.327988  192.168.1.138   34.78.63.171     STUN      146 Allocate Request UDP user: tfmVideo realm: caresupport.ml with nonce
   350 10.332804  34.78.63.171    192.168.1.138    STUN      146 Binding Success Response XOR-MAPPED-ADDRESS: 188.26.197.90:60666 MAPPED-ADDRESS: 188.26.197.90:60666 RESPONSE-ORIGIN: 10.132.0.2:3478 OTHER-ADDRESS: 10.132.0.2:3479
   353 10.355654  34.78.63.171    192.168.1.138    STUN      154 Allocate Success Response XOR-RELAYED-ADDRESS: 10.132.0.2:49209 XOR-MAPPED-ADDRESS: 188.26.197.90:60667 lifetime: 600
   356 10.357121  34.78.63.171    192.168.1.138    STUN      154 Allocate Success Response XOR-RELAYED-ADDRESS: 10.132.0.2:52268 XOR-MAPPED-ADDRESS: 188.26.197.90:60666 lifetime: 600
   364 11.503092  192.168.1.138   34.78.63.171     STUN      146 Refresh Request lifetime: 0 user: tfmVideo realm: caresupport.ml with nonce
   367 11.517899  192.168.1.138   34.78.63.171     STUN      150 CreatePermission Request XOR-PEER-ADDRESS: 10.210.142.199:53518 user: tfmVideo realm: caresupport.ml with nonce
   368 11.518980  192.168.1.138   10.210.142.199   STUN      142 Binding Request user: fMj6:5NWN
   369 11.525681  34.78.63.171    192.168.1.138    STUN      130 Refresh Success Response lifetime: 0
   370 11.543128  34.78.63.171    192.168.1.138    STUN      122 CreatePermission Success Response
   371 11.576385  192.168.1.138   34.78.63.171     STUN      178 Send Indication XOR-PEER-ADDRESS: 10.210.142.199:53518

      [Checksum Status: Unverified]
      [Stream index: 19]
    > [Timestamps]
      UDP payload (104 bytes)
v Session Traversal Utilities for NAT
      [Response In: 353]
    > Message Type: 0x0003 (Allocate Request)
      Message Length: 84
      Message Cookie: 2112a442
      Message Transaction ID: 6c726238414f626d6e423850
      [STUN Network Version: RFC-5389/8489 (3)]
    v Attributes
       > REQUESTED-TRANSPORT: UDP
       > USERNAME: tfmVideo
       > REALM: caresupport.ml
       > NONCE: 102afd51b51f7af1
       > MESSAGE-INTEGRITY
```

Figure 8-7. TURN operation

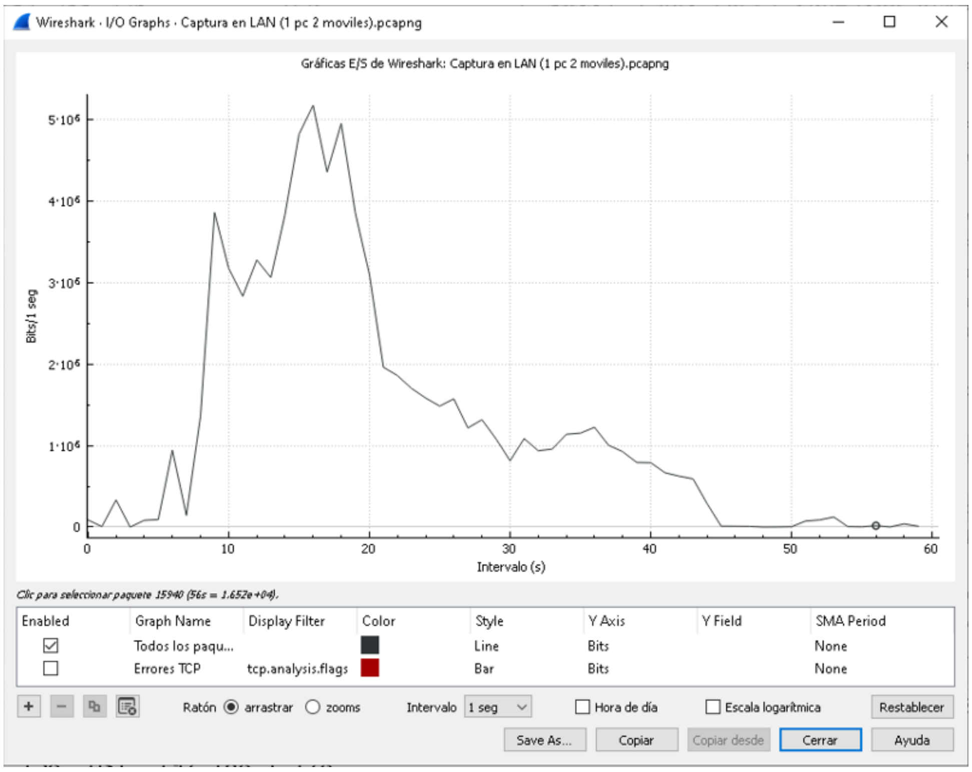
It is now when the TURN server creates a data channel with the user through which all the communication data between the ends will be transmitted (Figure 8-8).



```
No.   Time        Source          Destination     Protocol  Length Info
  399 12.201972   34.78.63.171    192.168.1.138   STUN      122 CreatePermission Success Response
  400 12.202497   192.168.1.138   90.167.218.11   STUN      142 Binding Request user: fMj6:5NWN
  404 12.254616   192.168.1.138   34.78.63.171    STUN      150 CreatePermission Request XOR-PEER-ADDRESS: 10.132.0.2:56243 user: tfmVideo realm: caresupport.ml with nonce
  405 12.264722   192.168.1.138   10.132.0.2      STUN      142 Binding Request user: fMj6:5NWN
  406 12.282097   34.78.63.171    192.168.1.138   STUN      122 CreatePermission Success Response
  408 12.334996   192.168.1.138   34.78.63.171    STUN      178 Send Indication XOR-PEER-ADDRESS: 90.167.218.11:6652
  411 12.389240   192.168.1.138   34.78.63.171    STUN      178 Send Indication XOR-PEER-ADDRESS: 10.132.0.2:56243
  412 12.419279   34.78.63.171    192.168.1.138   STUN      210 Data Indication XOR-PEER-ADDRESS: 10.132.0.2:56243
  413 12.419811   192.168.1.138   34.78.63.171    STUN      142 Send Indication XOR-PEER-ADDRESS: 10.132.0.2:56243
  414 12.452246   192.168.1.138   34.78.63.171    STUN      178 Send Indication XOR-PEER-ADDRESS: 10.132.0.2:56243
  417 12.499042   34.78.63.171    192.168.1.138   STUN      178 Data Indication XOR-PEER-ADDRESS: 10.132.0.2:56243
  418 12.521221   192.168.1.138   34.78.63.171    STUN      178 Send Indication XOR-PEER-ADDRESS: 10.132.0.2:56243
  419 12.521315   34.78.63.171    192.168.1.138   STUN      274 Data Indication XOR-PEER-ADDRESS: 10.132.0.2:56243
  420 12.521935   34.78.63.171    192.168.1.138   STUN      210 Data Indication XOR-PEER-ADDRESS: 10.132.0.2:56243
  421 12.522121   192.168.1.138   34.78.63.171    STUN      158 Channel-Bind Request ChannelNumber=0x4002 XOR-PEER-ADDRESS: 10.132.0.2:56243 user: tfmVideo realm: caresupport.ml with nonce
  422 12.522241   192.168.1.138   34.78.63.171    STUN      698 Send Indication XOR-PEER-ADDRESS: 10.132.0.2:56243
  423 12.522436   192.168.1.138   34.78.63.171    STUN      142 Send Indication XOR-PEER-ADDRESS: 10.132.0.2:56243
  424 12.527003   34.78.63.171    192.168.1.138   STUN      178 Data Indication XOR-PEER-ADDRESS: 10.132.0.2:56243
  425 12.547527   34.78.63.171    192.168.1.138   STUN      122 Channel-Bind Success Response
  426 12.577668   192.168.1.138   34.78.63.171    STUN      178 Send Indication XOR-PEER-ADDRESS: 10.210.142.199:53518
  427 12.579918   34.78.63.171    192.168.1.138   STUN      142 ChannelData TURN Message
  428 12.580623   192.168.1.138   34.78.63.171    STUN      110 ChannelData TURN Message
  429 12.592603   34.78.63.171    192.168.1.138   STUN      110 ChannelData TURN Message
  430 12.603636   34.78.63.171    192.168.1.138   STUN      591 ChannelData TURN Message
  431 12.604467   192.168.1.138   34.78.63.171    STUN      616 ChannelData TURN Message
  432 12.621429   192.168.1.138   34.78.63.171    STUN      163 ChannelData TURN Message

      [Checksum Status: Unverified]
      [Stream index: 19]
    > [Timestamps]
      UDP payload (104 bytes)
  v Session Traversal Utilities for NAT
      [Response In: 353]
    > Message Type: 0x0003 (Allocate Request)
      Message Length: 84
      Message Cookie: 2112a442
      Message Transaction ID: 6c726238414f626d6e423850
      [STUN Network Version: RFC-5389/8489 (3)]
    v Attributes
      > REQUESTED-TRANSPORT: UDP
      > USERNAME: tfmVideo
      > REALM: caresupport.ml
```

Figure 8-8. Data traveling through a data channel created by TURN

In Figure 8-9 it can be seen how the use of the bandwidth changes as a function of time and the development of the communication, the highest point corresponds when the three users are in the call and it is seen how it decreases depending on their disconnection.


Wireshark · I/O Graphs · Captura en LAN (1 pc 2 moviles).pcapng
Gráficas E/S de Wireshark: Captura en LAN (1 pc 2 moviles).pcapng
Bits/1 seg
5·10⁶
4·10⁶
3·10⁶
2·10⁶
1·10⁶
0
0
10
20
30
40
50
60
Intervalo (s)

Clic para seleccionar paquete 15940 (56s = 1.652e+04).

Enabled	Graph Name	Display Filter	Color	Style	Y Axis	Y Field	SMA Period
✓	Todos los paqu...		⬛	Line	Bits		None
☐	Errores TCP	tcp.analysis.flags	🟥	Bar	Bits		None

Ratón [✓ arrastrar] zooms Intervalo 1 seg ☐ Hora de día ☐ Escala logarítmica

Restablecer
Save As...
Copiar
Copiar desde
Cerrar
Ayuda

Figure 8-9. Bandwidth versus time throughout the video call

8.2 Network performance

To carry out this test, Iperf has been used. It is a tool used to test computer networks. The usual operation is to create TCP and UDP data streams and measure network performance.

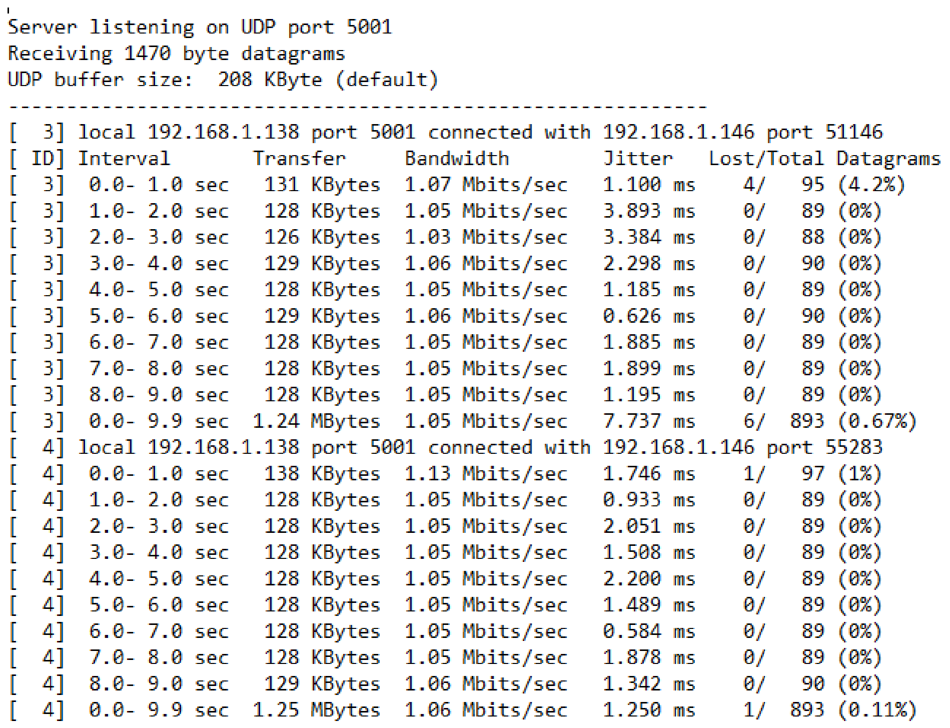

```
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 192.168.1.138 port 5001 connected with 192.168.1.146 port 51146
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0- 1.0 sec   131 KBytes  1.07 Mbits/sec   1.100 ms    4/   95 (4.2%)
[  3]  1.0- 2.0 sec   128 KBytes  1.05 Mbits/sec   3.893 ms    0/   89 (0%)
[  3]  2.0- 3.0 sec   126 KBytes  1.03 Mbits/sec   3.384 ms    0/   88 (0%)
[  3]  3.0- 4.0 sec   129 KBytes  1.06 Mbits/sec   2.298 ms    0/   90 (0%)
[  3]  4.0- 5.0 sec   128 KBytes  1.05 Mbits/sec   1.185 ms    0/   89 (0%)
[  3]  5.0- 6.0 sec   129 KBytes  1.06 Mbits/sec   0.626 ms    0/   90 (0%)
[  3]  6.0- 7.0 sec   128 KBytes  1.05 Mbits/sec   1.885 ms    0/   89 (0%)
[  3]  7.0- 8.0 sec   128 KBytes  1.05 Mbits/sec   1.899 ms    0/   89 (0%)
[  3]  8.0- 9.0 sec   128 KBytes  1.05 Mbits/sec   1.195 ms    0/   89 (0%)
[  3]  0.0- 9.9 sec  1.24 MBytes  1.05 Mbits/sec   7.737 ms    6/  893 (0.67%)
[  4] local 192.168.1.138 port 5001 connected with 192.168.1.146 port 55283
[  4]  0.0- 1.0 sec   138 KBytes  1.13 Mbits/sec   1.746 ms    1/   97 (1%)
[  4]  1.0- 2.0 sec   128 KBytes  1.05 Mbits/sec   0.933 ms    0/   89 (0%)
[  4]  2.0- 3.0 sec   128 KBytes  1.05 Mbits/sec   2.051 ms    0/   89 (0%)
[  4]  3.0- 4.0 sec   128 KBytes  1.05 Mbits/sec   1.508 ms    0/   89 (0%)
[  4]  4.0- 5.0 sec   128 KBytes  1.05 Mbits/sec   2.200 ms    0/   89 (0%)
[  4]  5.0- 6.0 sec   128 KBytes  1.05 Mbits/sec   1.489 ms    0/   89 (0%)
[  4]  6.0- 7.0 sec   128 KBytes  1.05 Mbits/sec   0.584 ms    0/   89 (0%)
[  4]  7.0- 8.0 sec   128 KBytes  1.05 Mbits/sec   1.878 ms    0/   89 (0%)
[  4]  8.0- 9.0 sec   129 KBytes  1.06 Mbits/sec   1.342 ms    0/   90 (0%)
[  4]  0.0- 9.9 sec  1.25 MBytes  1.06 Mbits/sec   1.250 ms    1/  893 (0.11%)
```

Figure 8-10. LAN Network performance test

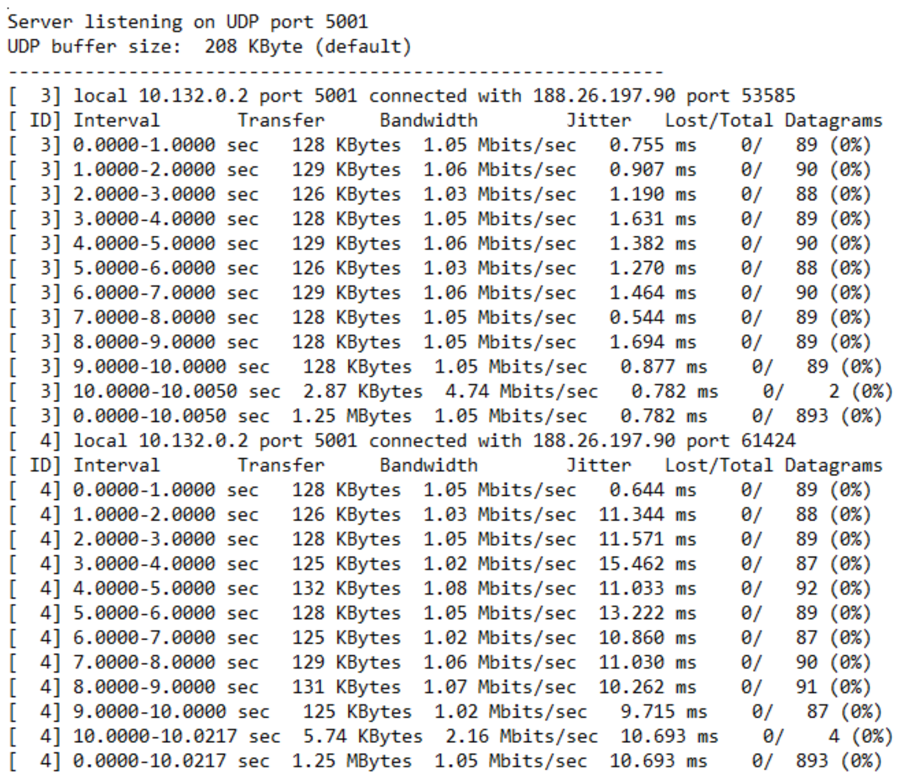

```
Server listening on UDP port 5001
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 10.132.0.2 port 5001 connected with 188.26.197.90 port 53585
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3] 0.0000-1.0000 sec   128 KBytes  1.05 Mbits/sec   0.755 ms    0/   89 (0%)
[  3] 1.0000-2.0000 sec   129 KBytes  1.06 Mbits/sec   0.907 ms    0/   90 (0%)
[  3] 2.0000-3.0000 sec   126 KBytes  1.03 Mbits/sec   1.190 ms    0/   88 (0%)
[  3] 3.0000-4.0000 sec   128 KBytes  1.05 Mbits/sec   1.631 ms    0/   89 (0%)
[  3] 4.0000-5.0000 sec   129 KBytes  1.06 Mbits/sec   1.382 ms    0/   90 (0%)
[  3] 5.0000-6.0000 sec   126 KBytes  1.03 Mbits/sec   1.270 ms    0/   88 (0%)
[  3] 6.0000-7.0000 sec   129 KBytes  1.06 Mbits/sec   1.464 ms    0/   90 (0%)
[  3] 7.0000-8.0000 sec   128 KBytes  1.05 Mbits/sec   0.544 ms    0/   89 (0%)
[  3] 8.0000-9.0000 sec   128 KBytes  1.05 Mbits/sec   1.694 ms    0/   89 (0%)
[  3] 9.0000-10.0000 sec   128 KBytes  1.05 Mbits/sec   0.877 ms    0/   89 (0%)
[  3] 10.0000-10.0050 sec  2.87 KBytes  4.74 Mbits/sec   0.782 ms    0/    2 (0%)
[  3] 0.0000-10.0050 sec  1.25 MBytes  1.05 Mbits/sec   0.782 ms    0/  893 (0%)
[  4] local 10.132.0.2 port 5001 connected with 188.26.197.90 port 61424
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  4] 0.0000-1.0000 sec   128 KBytes  1.05 Mbits/sec   0.644 ms    0/   89 (0%)
[  4] 1.0000-2.0000 sec   126 KBytes  1.03 Mbits/sec  11.344 ms    0/   88 (0%)
[  4] 2.0000-3.0000 sec   128 KBytes  1.05 Mbits/sec  11.571 ms    0/   89 (0%)
[  4] 3.0000-4.0000 sec   125 KBytes  1.02 Mbits/sec  15.462 ms    0/   87 (0%)
[  4] 4.0000-5.0000 sec   132 KBytes  1.08 Mbits/sec  11.033 ms    0/   92 (0%)
[  4] 5.0000-6.0000 sec   128 KBytes  1.05 Mbits/sec  13.222 ms    0/   89 (0%)
[  4] 6.0000-7.0000 sec   125 KBytes  1.02 Mbits/sec  10.860 ms    0/   87 (0%)
[  4] 7.0000-8.0000 sec   129 KBytes  1.06 Mbits/sec  11.030 ms    0/   90 (0%)
[  4] 8.0000-9.0000 sec   131 KBytes  1.07 Mbits/sec  10.262 ms    0/   91 (0%)
[  4] 9.0000-10.0000 sec   125 KBytes  1.02 Mbits/sec   9.715 ms    0/   87 (0%)
[  4] 10.0000-10.0217 sec  5.74 KBytes  2.16 Mbits/sec  10.693 ms    0/    4 (0%)
[  4] 0.0000-10.0217 sec  1.25 MBytes  1.05 Mbits/sec  10.693 ms    0/  893 (0%)
```

Figure 8-11. WAN Network performance test

In Figure 8-1, the test has been carried out on a local network with UDP traffic, simulating what could be the implementation of all the servers in the hospital's own network, while in Figure 8-2 the same test was simulated with the web application deployed on the Google Cloud virtual machine.

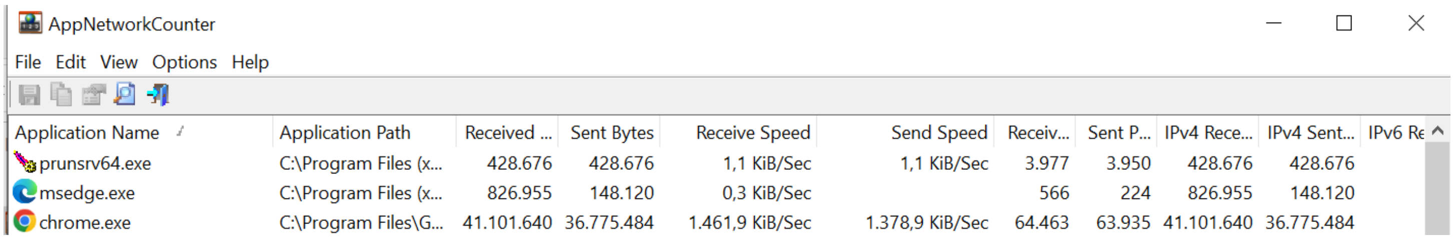
As can be seen in the results obtained, very similar results are obtained whether we deploy the server on a local machine and work from within the LAN network or if it is deployed in the Google cloud.

In fact, as can be seen in Figure 8-1, in the LAN test some datagrams have been lost while in the server hosted in the Google cloud they have not.

The jitter values obtained are also considered optimal since experts speak of 20 milliseconds as the ceiling for acceptable jitter.

To measure the bandwidth in a real way, another tool had to be used, in this case called AppNetworkCounter, since it must be measured in a typical video call situation where there are at least two participants.

In this case, a video call of three participants has been simulated obtaining the results shown in Figure 8.3.



Application Name	Application Path	Received ...	Sent Bytes	Receive Speed	Send Speed	Receiv...	Sent P...	IPv4 Rece...	IPv4 Sent...	IPv6 Re
prunsrv64.exe	C:\Program Files (x...	428.676	428.676	1,1 KiB/Sec	1,1 KiB/Sec	3.977	3.950	428.676	428.676	
msedge.exe	C:\Program Files (x...	826.955	148.120	0,3 KiB/Sec		566	224	826.955	148.120	
chrome.exe	C:\Program Files\G...	41.101.640	36.775.484	1.461,9 KiB/Sec	1.378,9 KiB/Sec	64.463	63.935	41.101.640	36.775.484	

Figure 8-12. Bandwidth test

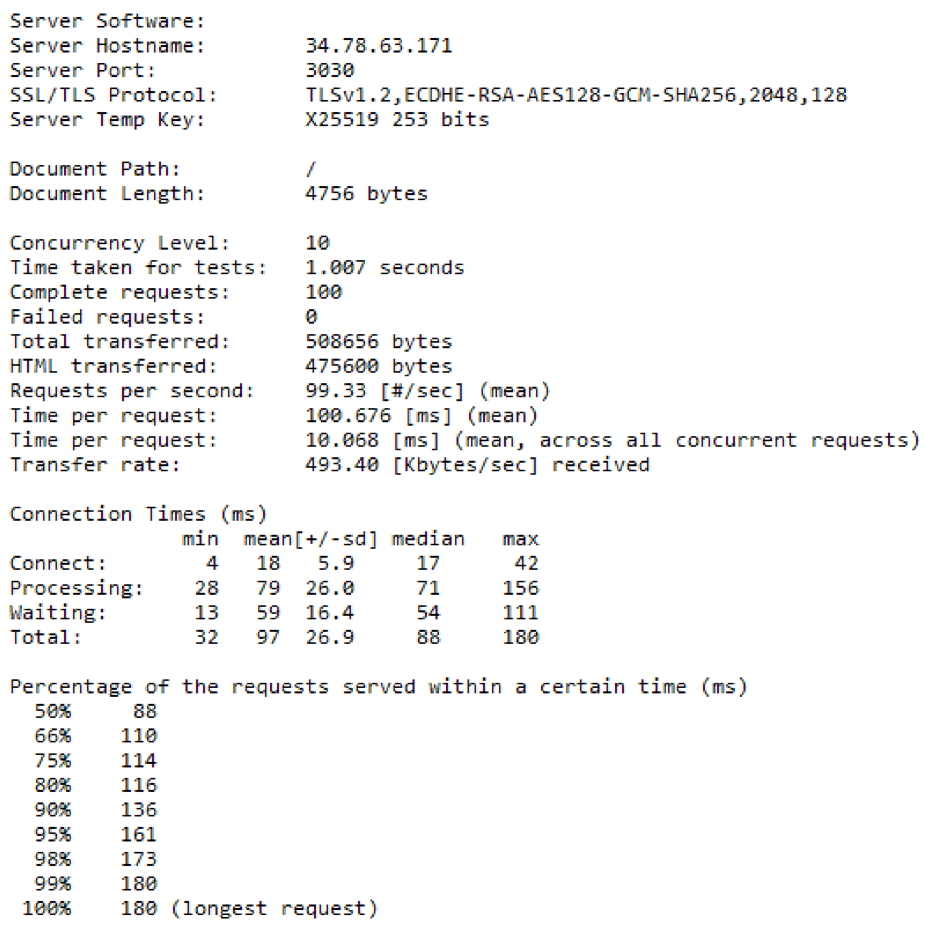
We get an average of 750 KB/Sec for each user. As there are three users, the download traffic is divided between two (we are the third user who acts as upload traffic).

In this case, it also depends on the device from which the video is obtained, since, for example, lower values have been obtained from a phone with more limited characteristics, around 100 KB/Sec per user.

8.3 Load test

To carry out this load test, the Apache Bench tool has been used. In this test, a total of 10,000 requests have been made, with 100 requests being made on a recurring basis.

As a result of this test (Figure 8-4) the following report has been obtained:



```
Server Software:
Server Hostname:        34.78.63.171
Server Port:            3030
SSL/TLS Protocol:       TLSv1.2,ECDHE-RSA-AES128-GCM-SHA256,2048,128
Server Temp Key:        X25519 253 bits

Document Path:          /
Document Length:        4756 bytes

Concurrency Level:      10
Time taken for tests:   1.007 seconds
Complete requests:      100
Failed requests:        0
Total transferred:      508656 bytes
HTML transferred:       475600 bytes
Requests per second:    99.33 [#/sec] (mean)
Time per request:       100.676 [ms] (mean)
Time per request:       10.068 [ms] (mean, across all concurrent requests)
Transfer rate:          493.40 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        4   18   5.9     17      42
Processing:    28   79  26.0     71     156
Waiting:       13   59  16.4     54     111
Total:         32   97  26.9     88     180

Percentage of the requests served within a certain time (ms)
  50%     88
  66%    110
  75%    114
  80%    116
  90%    136
  95%    161
  98%    173
  99%    180
 100%    180 (longest request)
```

Figure 8-13. Load test results

The most notable is the following:

- Requests per second: For every second, 99.33 requests are handled on average.
- Time per request (mean): The average time that the server has taken to attend to each block of 100 concurrent requests is 100.676 ms.
- Time per request (mean, across all concurrent requests): The time that the server has taken to attend to an individual request has been 10,068 ms.

The graphic representation of these results is shown in Figure 8-5. This figure shows how initially for the first requests as they are made, the response time increases quite significantly, but when reaching a certain number (around 20), the average response time of the server goes increasing almost linearly.

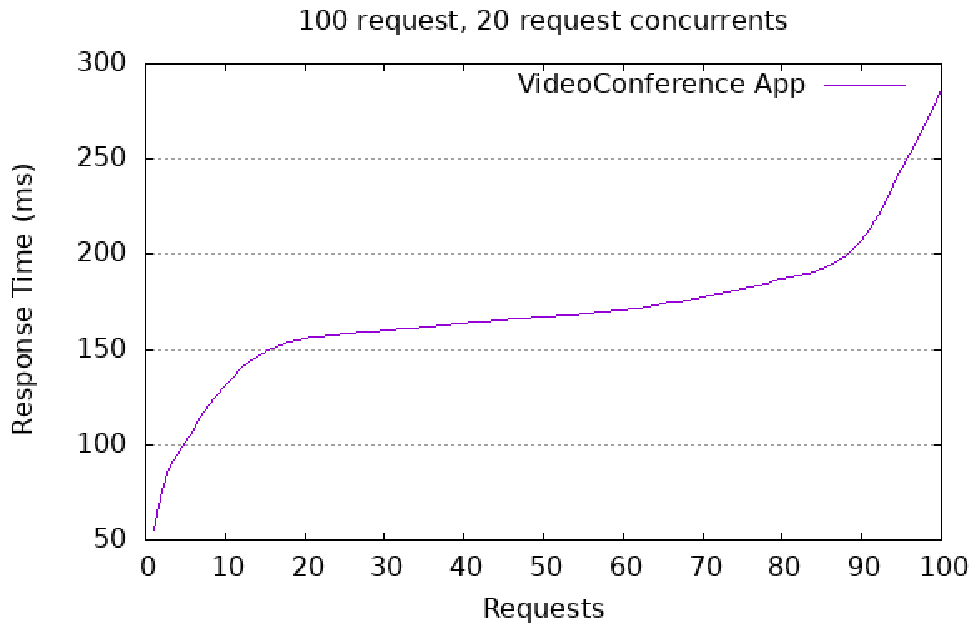


Figure 8-14. Load test tesults in graph

8.4 Stress test

8.4.1 Test 1. Stress tests with only one room open

Another stress test has been performed, regarding a massive case of users in a videoconference. The web application performs well in a room with up to 5 users. From this point on, it starts to present some problems such as video/audio freezes and automatic disconnections.

8.4.2 Test 2. Stress tests with three users connected per room

In this case the results before the stress test have been similar. With three users per room, the system is capable of keeping two simultaneous rooms running smoothly. From the third, and with the intervention of a seventh/eighth participant, the system begins to present some failures such as freezes in the audio/video, although in this case the automatic disconnections of the users are increased and much more appreciated.

9 RESULTS AND CONCLUSIONS

The result of this project has been the design and implementation of an infrastructure that consist of three servers (web / signaling / support) and a database that to provide support to a web application that allows the use of a multi-user video call system in a health environment.

It has been necessary to acquire a series of knowledge and skills to carry out the execution of all the proposed objectives, including familiarization and in-depth study of real-time communication protocols together with WebRTC technology.

Once the implementation phase of infrastructure has been completed, the system is flexible enough to be deployed even when one of the end-points of the communication it's the browser of a specific robot. Indeed, this system will be deployed in a specific real use-case where a robot is in charge of allowing remote communication among patients and care specialist.

It has also been necessary to analyze the benefits of using an open technology such as WebRTC to make your own videoconferencing system, and therefore, it can be adapted to the specific needs of a scenario in which a system of this type may be necessary, beyond the use of a proprietary video conferencing system. In this sense, the main limitation has been the need from the end of the robot to allow fixed configurations regarding the devices to use or how to initiate or accept a call without the intervention of a person.

With the development of this work have been put into practice many theoretical concepts acquired throughout both the degree of Telematics Engineering and the Master of Telecommunications Engineering that ends with the writing of this document, ranging from small theoretical, operation of different technologies and protocols, through implementation of different servers, different modes of communication, functions implemented under JavaScript, study and proposed solutions to the problem that arose in the beginning, to the final development of a web application that provides the user services that were intended to cover in the development of this project. To this end, it has been necessary to acquire a multitude of knowledge and combine it with the knowledge we already had in order to achieve the objectives set.

In this work we have explored a programming language for the development of web applications: JavaScript. The author had some knowledge of JavaScript but had to go much deeper in order to develop and address all the implementation requirements of this project.

Among the objectives that were pursued with the achievement of this project, the main one was the realization of a multi-user video call using WebRTC. This fundamental objective has been achieved.

The main objective of this work has been accomplished. This objective was the development of a system based on WebRTC technology that allows establishing a videoconference for multiple users. In addition, it was necessary to provide this system with different functionalities such as the existence of a text chat, the invitation of users via email, the reservation of video calls, etc.

The objectives of dividing the application into two distinct sections, one for specialists/doctors and the other for patients/families have also been achieved. In the specialist section, all the services associated with the management of patient and user video calls have been implemented, while the user section refers to the video call itself.

Two important problems arose that had to be solved while the project was being rolled out. Once everything was implemented and working, some problems arose such as:

- The system had been prepared for multiuser but not for multisession, that is, in the case of a hospital with two or more robots (due to the number of patients it has, for example) this application did not work since the signaling server only was able to hold a video call at a time without crossing the data. To solve this, an object variable was created in which the different sessions were dynamically saved and modified, thus allowing several simultaneous and multi-user video calls.
- In a specific situation, if in a video call of more than two people the person who entered the room before another, if he left the video call, the connection was closed but the camera stayed frozen on the window and did not disappear. To fix this, a disconnect handler had to be implemented at both ends of the video call.

From a technical point of view, this work has provided me with a great deal of depth and development in some technologies and protocols that have been studied throughout my master.

To conclude this chapter, it is worth mentioning that the system has been tested in a real scenario, behaving correctly and obtaining satisfactory results in different situations.

Finally, and leaving aside the purely implementation aspect, the aim has been to produce a report as complete and faithful as possible in relation to the work carried out.

10 FUTURE LINES

Although the implemented infrastructure meets the requirements established at the beginning of the work, several lines of work have been detected that could be developed in the future:

- Incorporation of a tool for evaluating the quality of video calls. It could collect statistics and data from WebRTC to store them and analyze the different behaviors of the tool in different situations. In addition, if a system is incorporated that is capable of self-assessing the quality of these, functions can also be implemented that make it possible to self-select an audio/video quality in these, typical of other systems such as YouTube.
- Implement additional functionalities, such as an individual chat over the general chat that exists, the possibility of sharing the screen in case of using telemedicine by the doctor or specialist, recording of the sessions, the implementation of a system of file transfer using the RTCDatachannels API, option to minimize or hide the different video windows or even the implementation of a virtual whiteboard.
- Develop the native application on Android and iOS pointing to our services to integrate it with the web version of this project.

11 ANNEXES

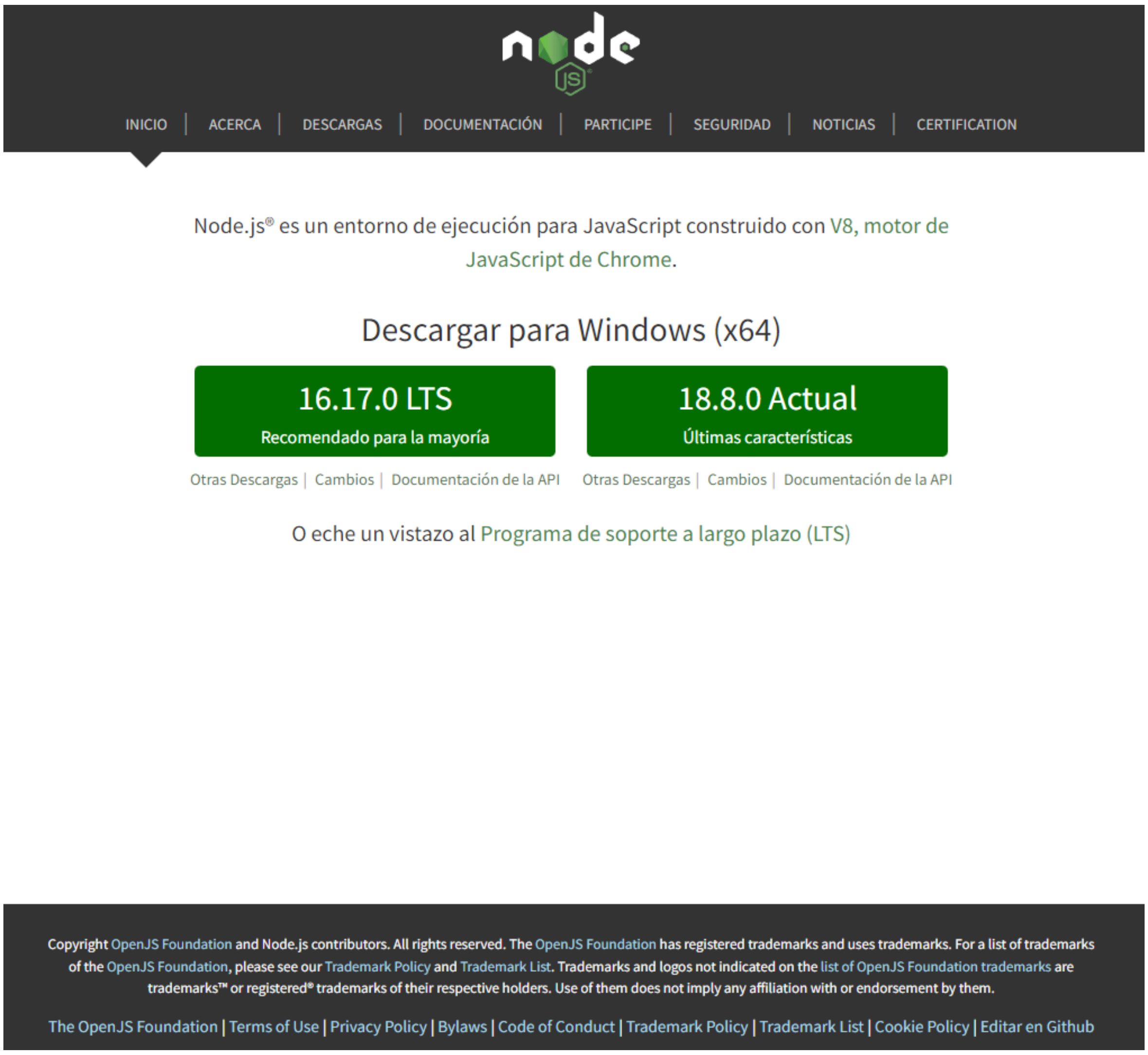
This chapter is made up of a series of annexes in which different aspects presented in previous chapters are developed. Thus, first of all, the reader is offered the installation of Node.js.

Next, an annex has been incorporated with the user manual of the developed web application.

11.1 Annex I. Installing Node.js

11.1.1 Node.js in windows

To download it, you only have to access the official website http://nodejs.org/es/ (Figure 11-1) and choose the desired download, generally the LTS (Long Term Support) version.


INICIO | ACERCA | DESCARGAS | DOCUMENTACIÓN | PARTICIPE | SEGURIDAD | NOTICIAS | CERTIFICATION
16.17.0 LTS
Recomendado para la mayoría
18.8.0 Actual
Últimas características
Otras Descargas | Cambios | Documentación de la API
Otras Descargas | Cambios | Documentación de la API
The OpenJS Foundation | Terms of Use | Privacy Policy | Bylaws | Code of Conduct | Trademark Policy | Trademark List | Cookie Policy | Editar en Github

Node.js® es un entorno de ejecución para JavaScript construido con V8, motor de JavaScript de Chrome.

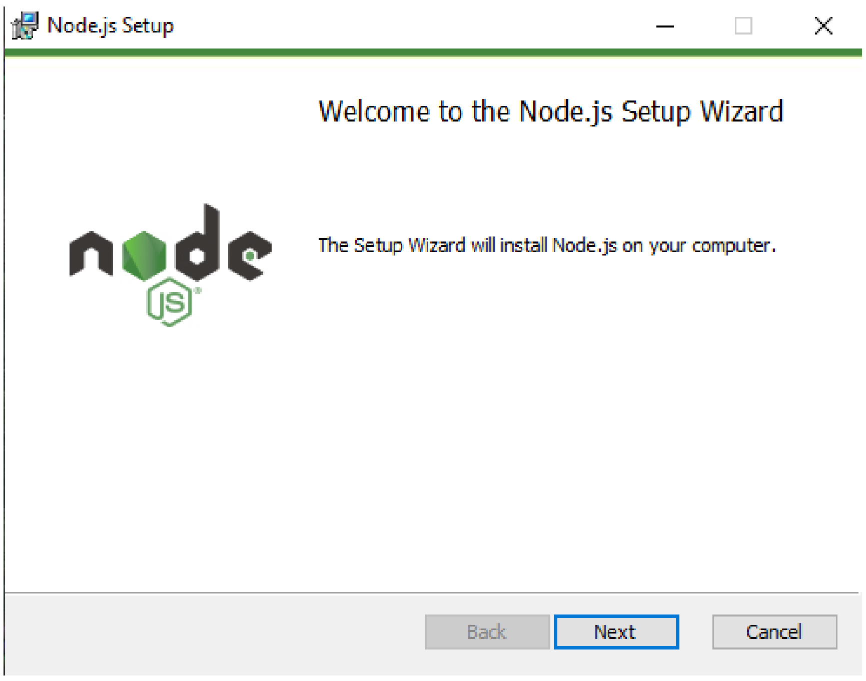
Descargar para Windows (x64)

O eche un vistazo al Programa de soporte a largo plazo (LTS)

Copyright OpenJS Foundation and Node.js contributors. All rights reserved. The OpenJS Foundation has registered trademarks and uses trademarks. For a list of trademarks of the OpenJS Foundation, please see our Trademark Policy and Trademark List. Trademarks and logos not indicated on the list of OpenJS Foundation trademarks are trademarks™ or registered® trademarks of their respective holders. Use of them does not imply any affiliation with or endorsement by them.

Figure 11-1. Node.js official website

Once the version is selected, the installer will automatically download. To proceed with its installation, it will open showing the following screen (Figure 11-2):

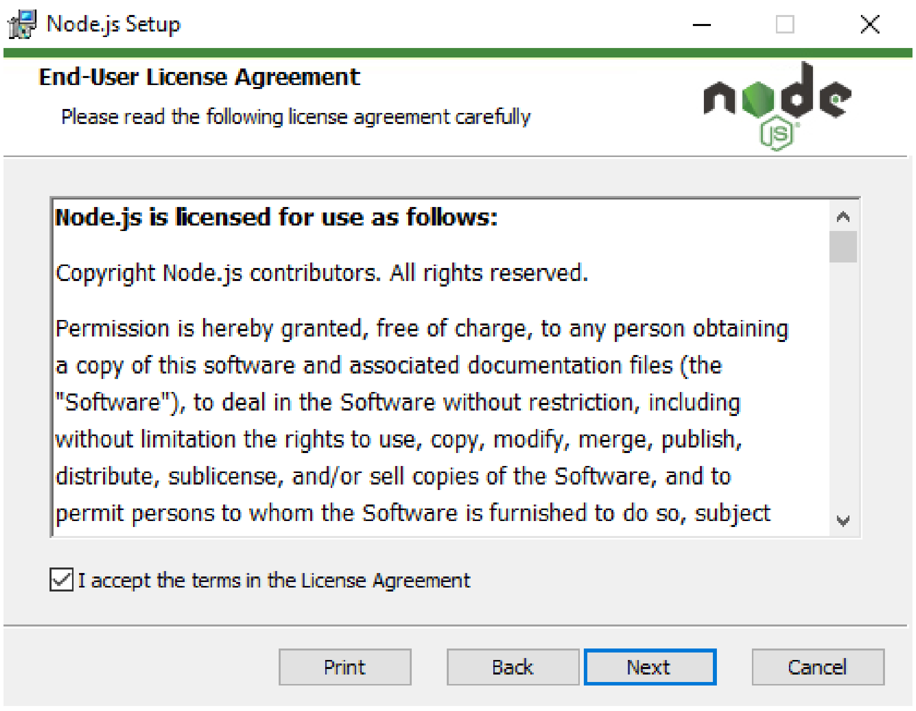

Node.js Setup

Welcome to the Node.js Setup Wizard

The Setup Wizard will install Node.js on your computer.

Figure 11-2. Node.js installer homepage

Click on next showing the terms and conditions of use (Figure 11-3) that must be accepted to continue:


Node.js Setup

End-User License Agreement

Please read the following license agreement carefully

Node.js is licensed for use as follows:

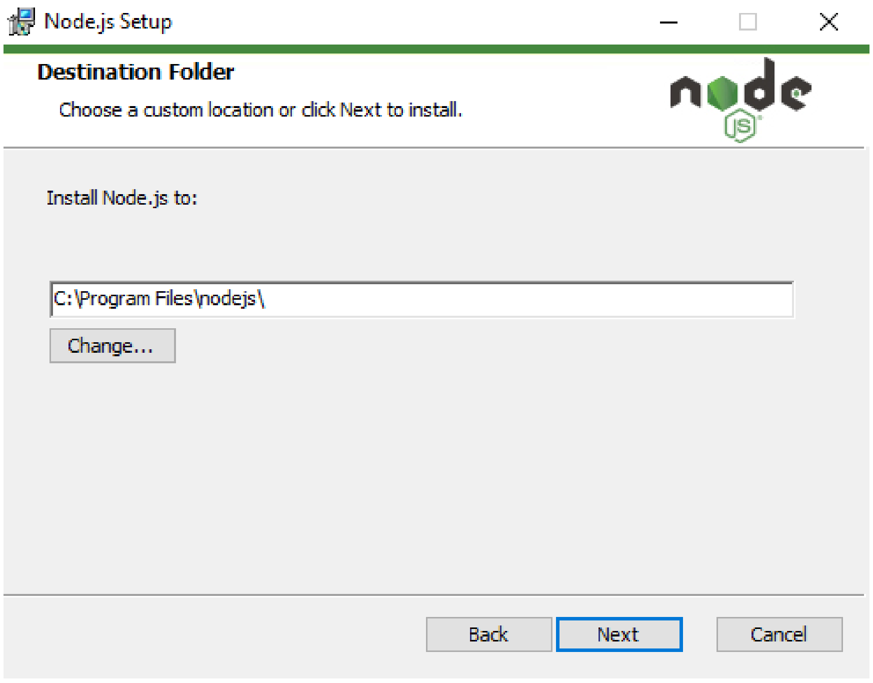
Copyright Node.js contributors. All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject

[✓] I accept the terms in the License Agreement

Figure 11-3. Node.js installer terms screen

The next window shown is the installation directory selection window (Figure 11-4):



Node.js Setup

Destination Folder

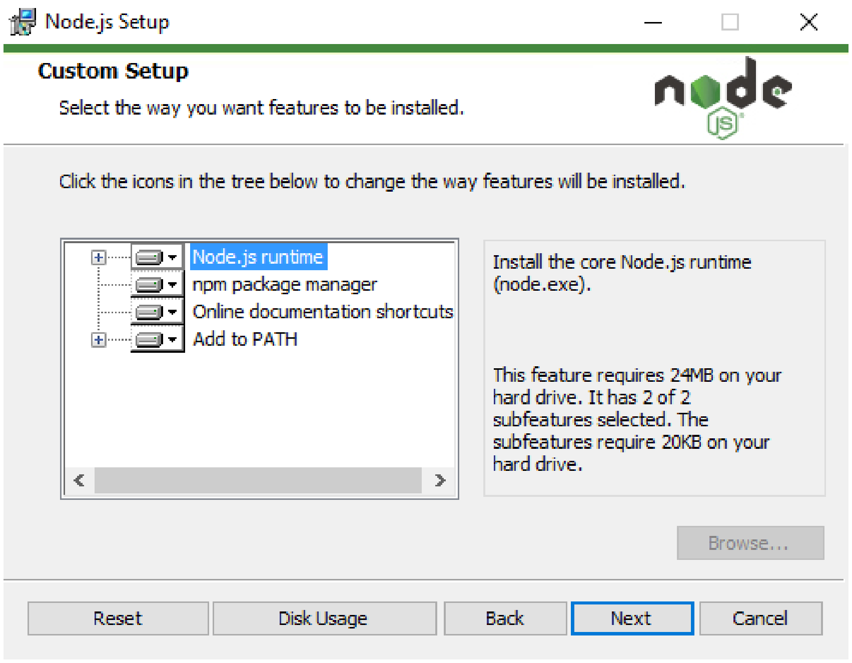
Choose a custom location or click Next to install.

Install Node.js to:

C:\Program Files\nodejs\

Figure 11-4. Node.js installation directory selection screen

Next, the screen shows the installation options (Figure 11-5), among which it is worth mentioning that in order to use the Node.js commands in the Windows terminal, the "Add to PATH" option must be active.



Node.js Setup

Custom Setup

Select the way you want features to be installed.

Click the icons in the tree below to change the way features will be installed.

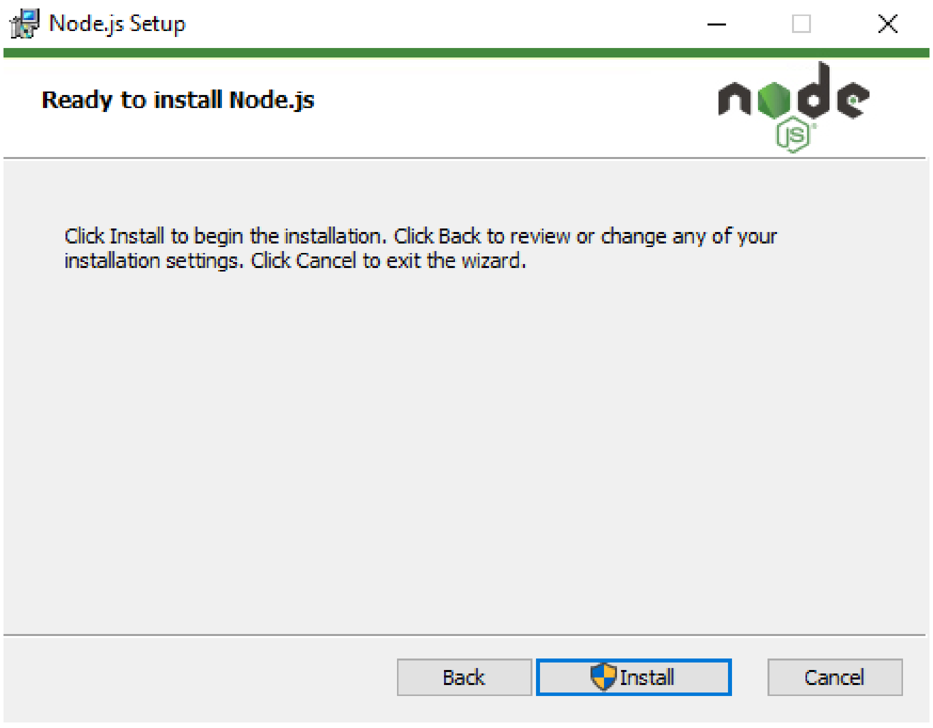
- Node.js runtime
- npm package manager
- Online documentation shortcuts
- Add to PATH

Install the core Node.js runtime (node.exe).

This feature requires 24MB on your hard drive. It has 2 of 2 subfeatures selected. The subfeatures require 20KB on your hard drive.

Figure 11-5. Node.js installation options screen

Finally, click on “Install” to proceed with its installation (Figure 11-6):

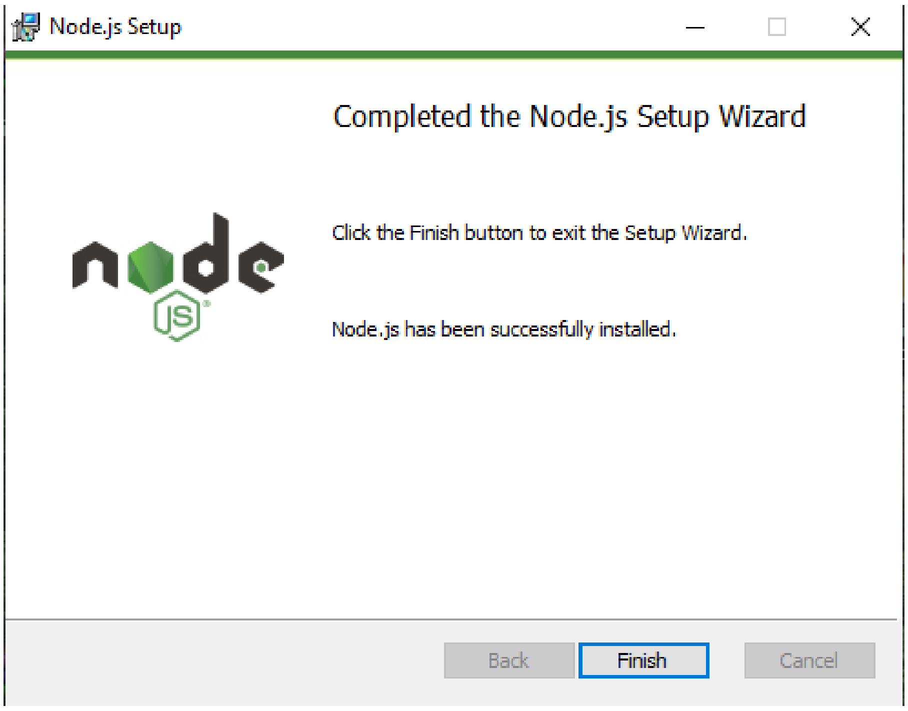

Node.js Setup

Ready to install Node.js

Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.

Figure 11-6. Node.js installation screen

And after the installation, the final window is shown (Figure 11-7) where the installation of Node.js would already have been carried out in a Windows OS (Operating System).


Node.js Setup

Completed the Node.js Setup Wizard

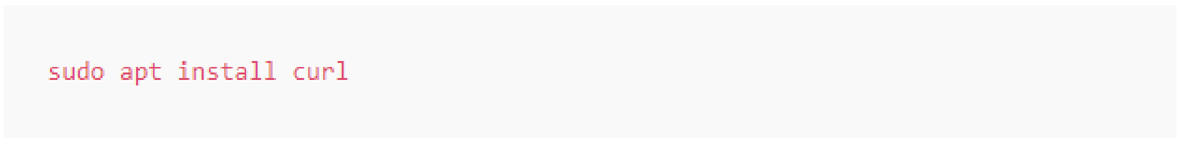
Click the Finish button to exit the Setup Wizard.

Node.js has been successfully installed.

Figure 11-7. Node.js installation completion screen

11.1.2 Node.js in linux

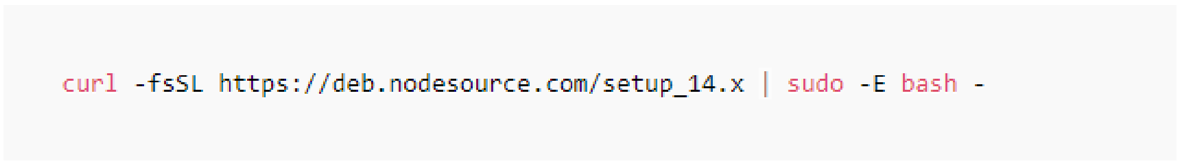
Before installing Node.js, you must have the curl command line utility installed on your system. If not, you should run:



```
sudo apt install curl
```

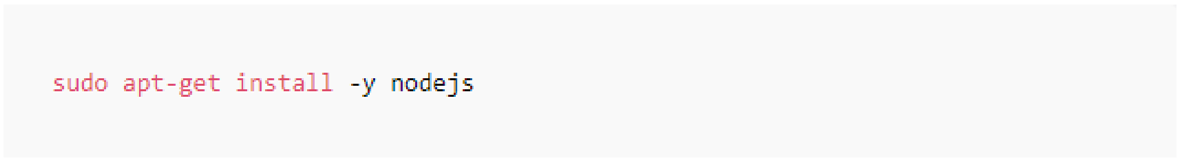
Figure 11-8. Curl install command

For example, here we will install Node.js v14.x. These are the installation commands for Ubuntu:



```
curl -fsSL https://deb.nodesource.com/setup_14.x | sudo -E bash -
```

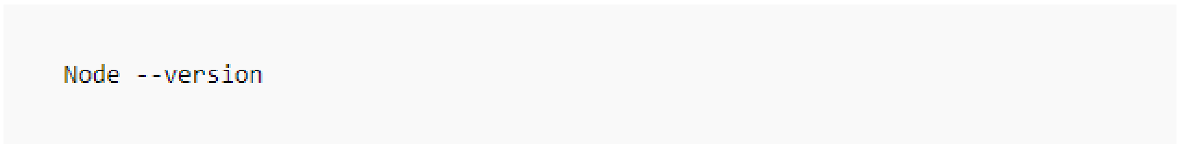
Figure 11-9. Node.js download command



```
sudo apt-get install -y nodejs
```

Figure 11-10. Node.js installation command

To check the version of Node.js installed we can execute the following command:



```
Node --version
```

Figure 11-11. Node.js version check command

11.2 Annex II. VideoConf Web Application User Manual

This document describes the management of a system in a hospital environment that allows for the booking of calls with relatives and/or doctors.
For security reasons, the system can only be accessed by hospital staff, who will be responsible for discharging patients and managing bookings.
There are two user roles that can use the system. The main administrator who can register other managers, e.g., nurses or healthcare professionals in the center and access the rest of the functionalities. Users registered by the administrator will be able to use these functionalities to register residents and make bookings for videoconferences with relatives or other specialists.

11.2.1 Homepage

The home screen of the application welcomes the user to the system and offers different alternatives. The available functionalities are defined in the following sections.

There are two main sections in the homepage:

- Navigation Bar: To do login, if you are an administrator user.
- Let’s get started button: Go to profile if you are logged in.

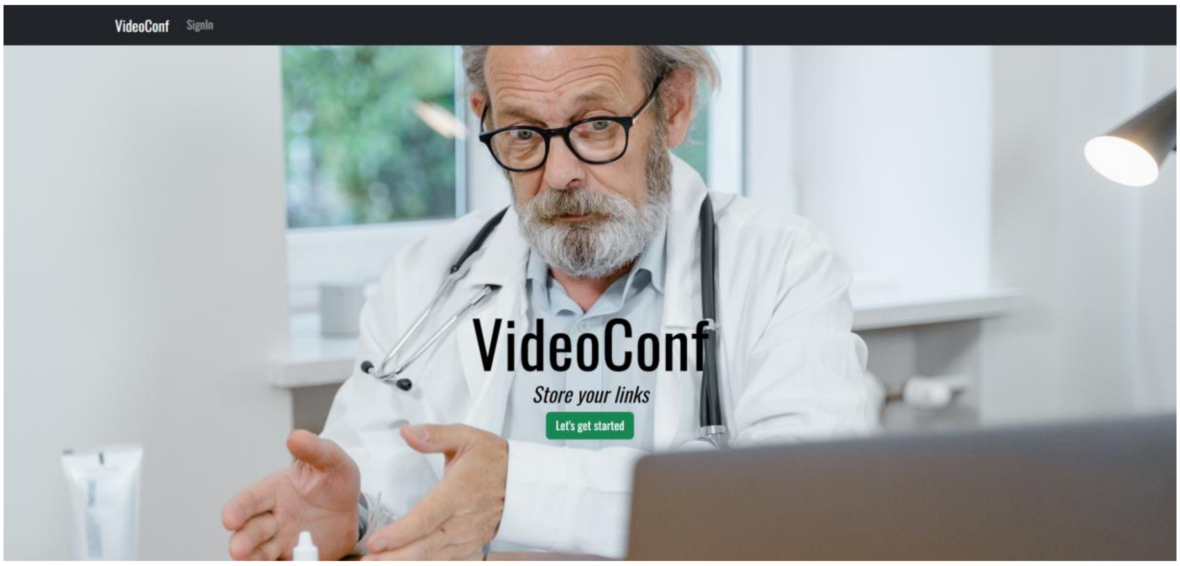


Figure 11-12. Screenshot of the home page

11.2.2 SignIn

This interface is for the exclusive use of the system administrator and the workers registered in the system.

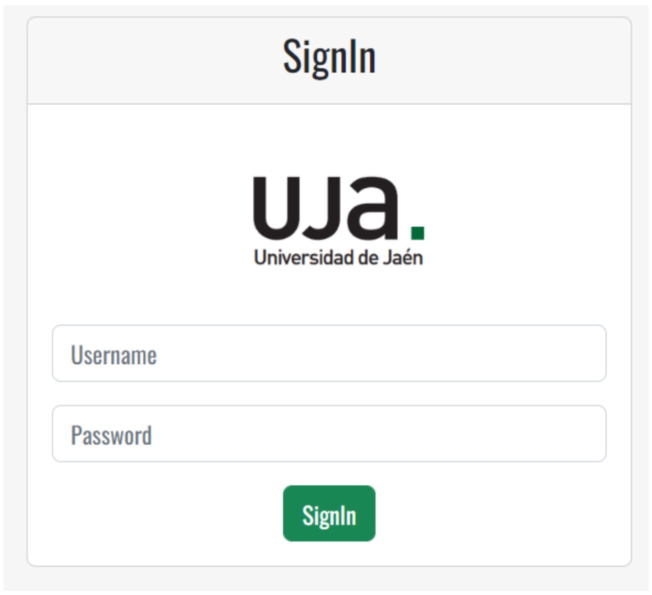


Figure 11-13. Access form

If the user has already been registered in the system, he/she will access this login interface shown in Figure 11-13 which allows him/her to enter his/her credentials.

The first interface after logging in is the one shown in Figure 11-14, which is associated with the user's profile. In addition, the navigation bar allows access, in the case of the administrator, to the screens that allow him/her to create and view videoconference reservations (Reservations), register residents and associate family and specialists' e-mails (Residents) and register managers (SignUp).

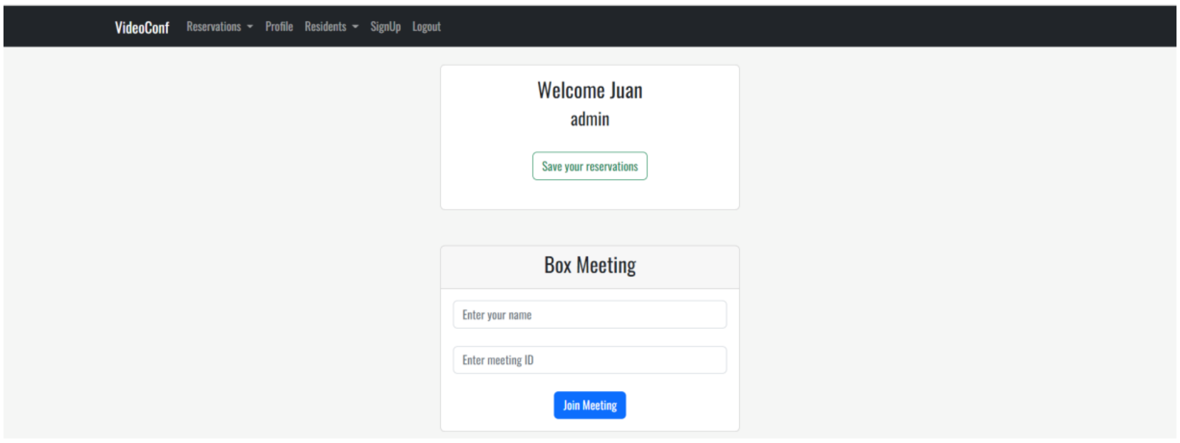


Figure 11-14. User's profile and box meeting

Also, in this part there is a box meeting where you can put a valid Room ID to enter in a videocall.

This form allows you to enter a user identifier for the videoconferencing system and the call identifier or token that will be automatically generated by the system and which, once active, will be accessible to the family member or remote specialist through a link sent by mail and to the manager/administrator through the corresponding reservation information.

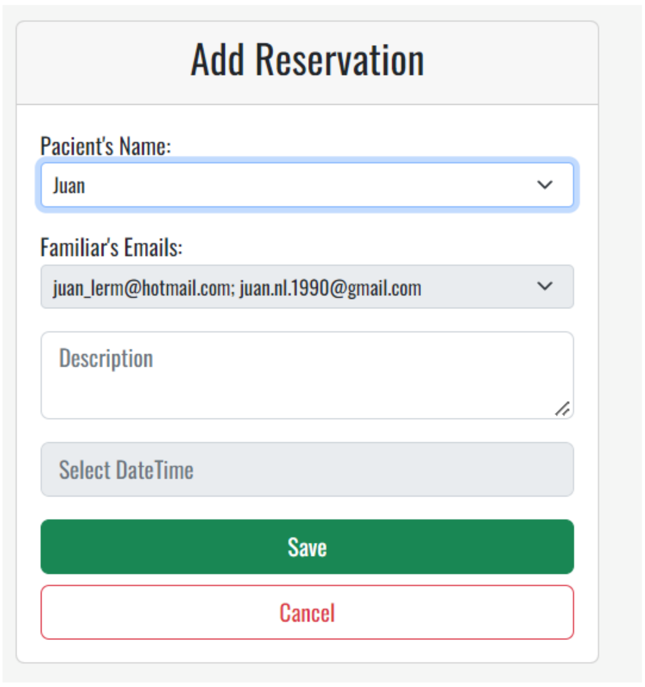
Once a valid session ID has been specified, the system will redirect to the video conference part.

11.2.2.1 Reservations

This application part aims to be able to manage all the reservations to make a videoconference that the administrator user in question has made.
If we click on “Save your reservations”, we access to a new page where we can create a new reservation.

11.2.2.1.1 Add reservations

It is necessary to specify the fields associated with the videoconference reservation.


Add Reservation

Pacient's Name: Juan

Familiar's Emails: juan_lerm@hotmail.com; juan.nl.1990@gmail.com

Description

Select DateTime

Figure 11-15. Reservation box

The family email box will be filled automatically once the patient is selected. The email field is associated with said patient when he is registered as a resident.

This part can also be accessed by clicking on the navigation bar: Reservations → Show Reservations

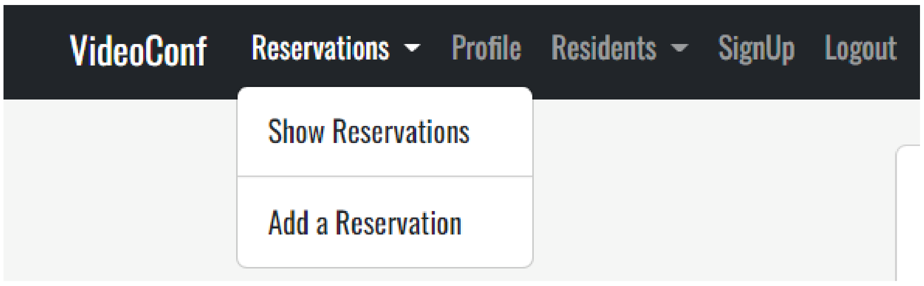


Figure 11-16. Reservation menu

11.2.2.1.2 Show reservations

This page shows all reservations associated with the administrator user.

The user can edit the different reservations and delete the ones he needs.

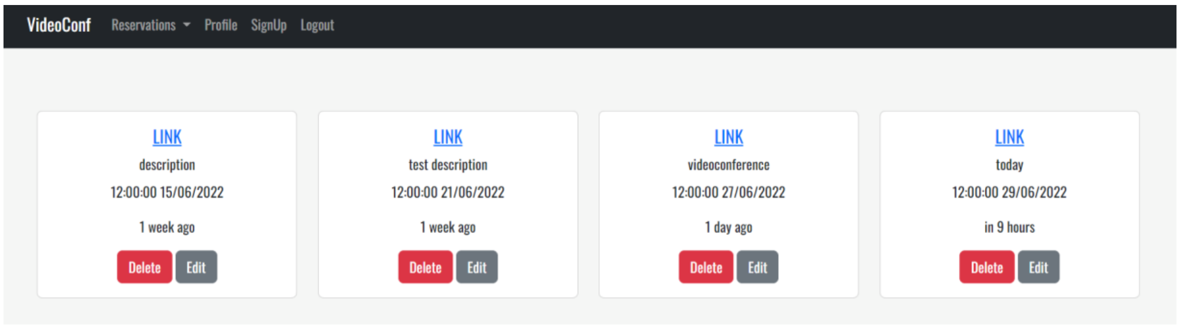
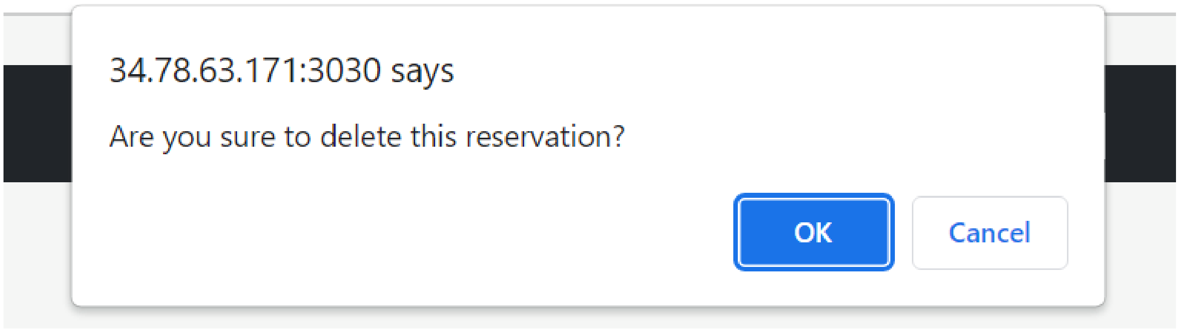


Figure 11-17. Show menu

- With the edit button, we can edit all the fields associated with the reservation.
- With the delete button, we can delete the reservation of the database.

It's necessary, for security, confirm the delete process on the alert message appear when you click on the button.


34.78.63.171:3030 says

Are you sure to delete this reservation?

Figure 11-18. Delete alert

11.2.2.2 Profile

From this page, as explained above, we can access our account data as well as add a new reservation. See image 11-14.

11.2.2.3 Residents

In this part of the web application, hospital residents will be discharged.

It is the part in charge of managing the data associated with patients.

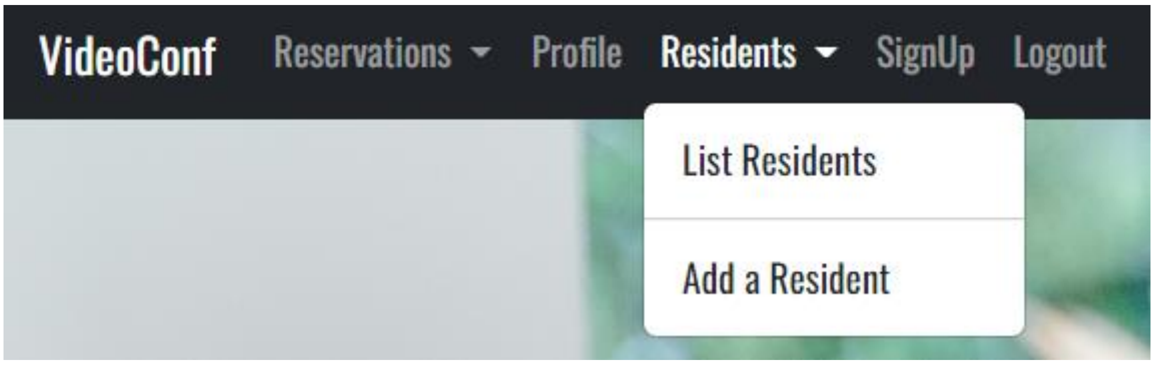
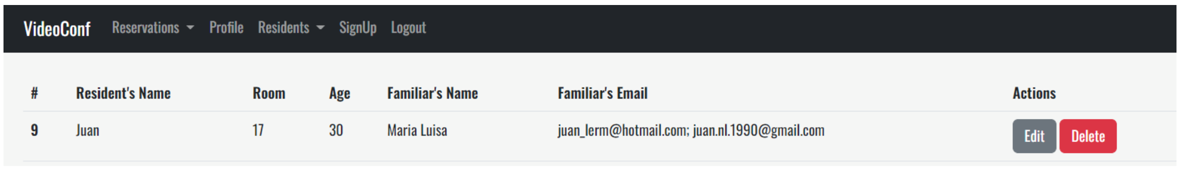


Figure 11-19. Residents menu

11.2.2.3.1 List residents

This page shows all residents registered in the hospital.

The user can edit the different residents and delete the ones he needs.



#	Resident's Name	Room	Age	Familiar's Name	Familiar's Email	Actions
9	Juan	17	30	Maria Luisa	juan_lerm@hotmail.com; juan.nl.1990@gmail.com	Edit Delete

Figure 11-20. List residents

- With the edit button, we can edit all the fields associated with the resident.
- With the delete button, we can delete the resident of the database.

It's necessary, for security, confirm the delete process on the alert message appear when you click on the button.

11.2.2.3.2 Add residents

It is necessary to specify the fields associated with the resident registration.

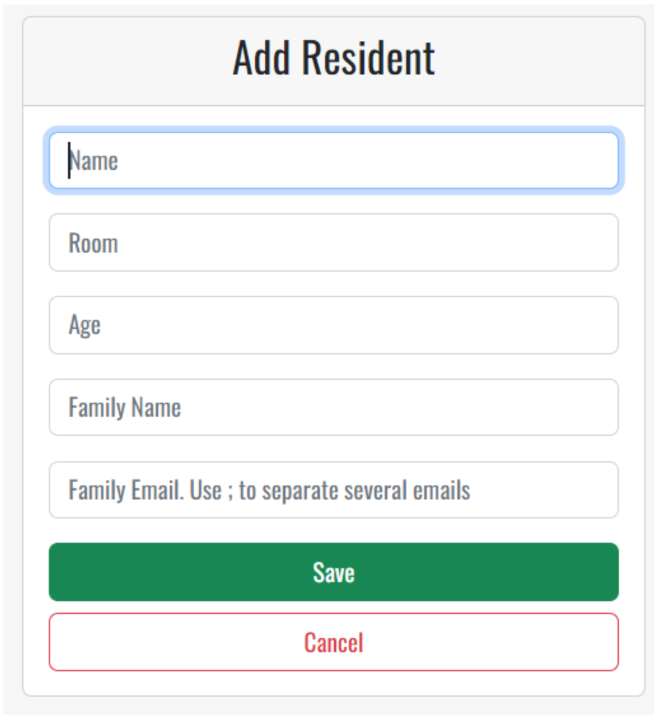


Figure 11-21. Residents box

When you add a new resident, the app redirects you to the residents list page where you can see all the residents.

11.2.2.4 SignUp

In this part it is possible to register new users for the administration of the system and reservations.

Simply specify the different data of the new user that you want to register.

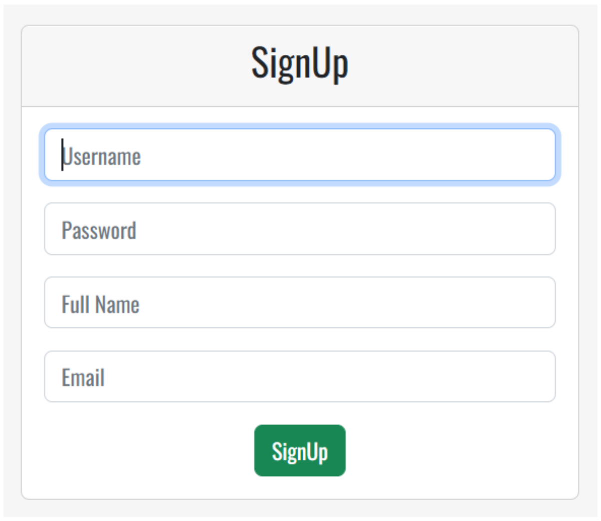


Figure 11-22. SignUp box

11.2.2.5 Logout

This button allows the user to log out of the application and automatically redirects the user to the login screen.

11.2.3 Room meeting

Once you have entered the room either through the email link or through the box meeting of a user logged, the system validates the session ID and redirect to the video conference part.

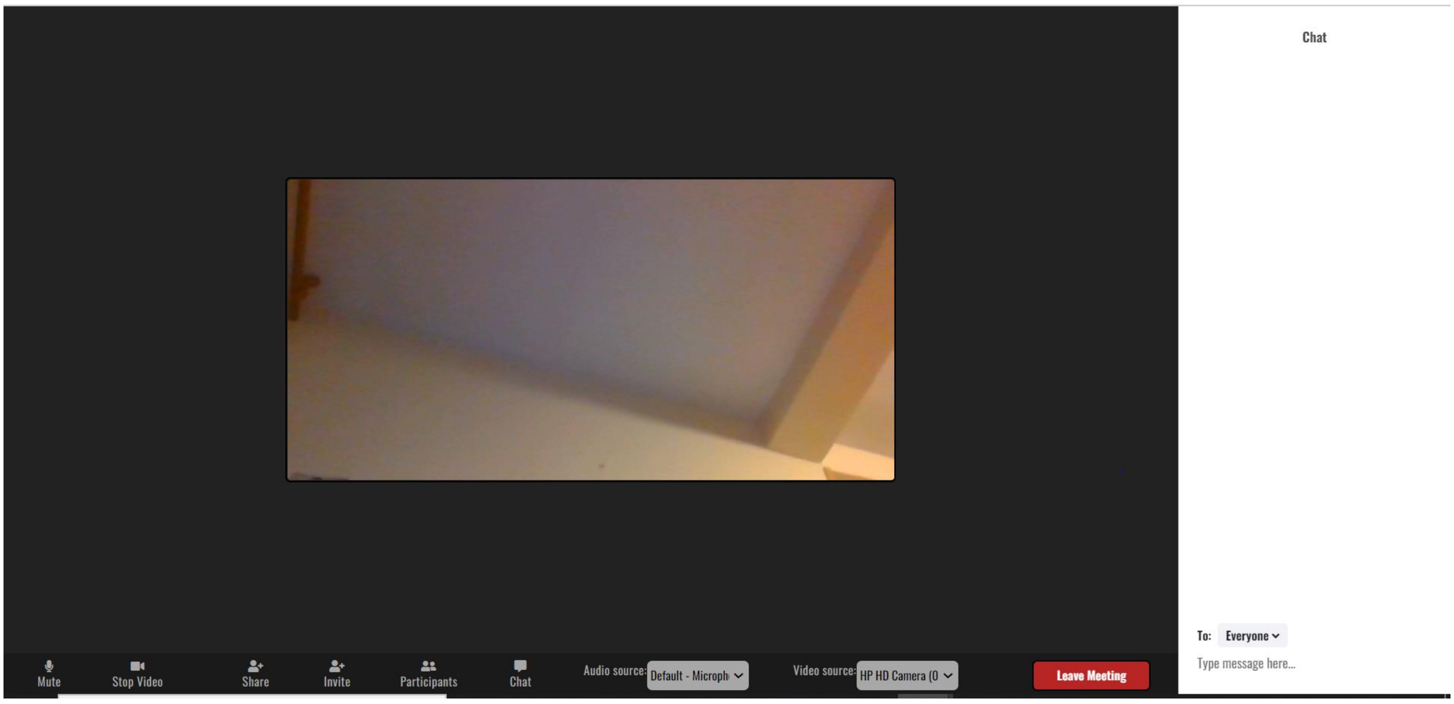


Figure 11-23. Videoconference system

The videoconference system has differents functionalities:

- Mute Button: Mute/Unmute device microphone.
- Stop Video: Turn off/on device video.
- Invite: Shows a full link of the video conference with the option to copy it.
- Participants: Shows the participants of the video conference.
- Chat: Hide/Show text chat.
- Audio Source: Select the audio source of the device.
- Video Source: Select the video source of the device.
- Leave Metting Button: Exit the video conference.

11.2.4 Let’s Get Started Button

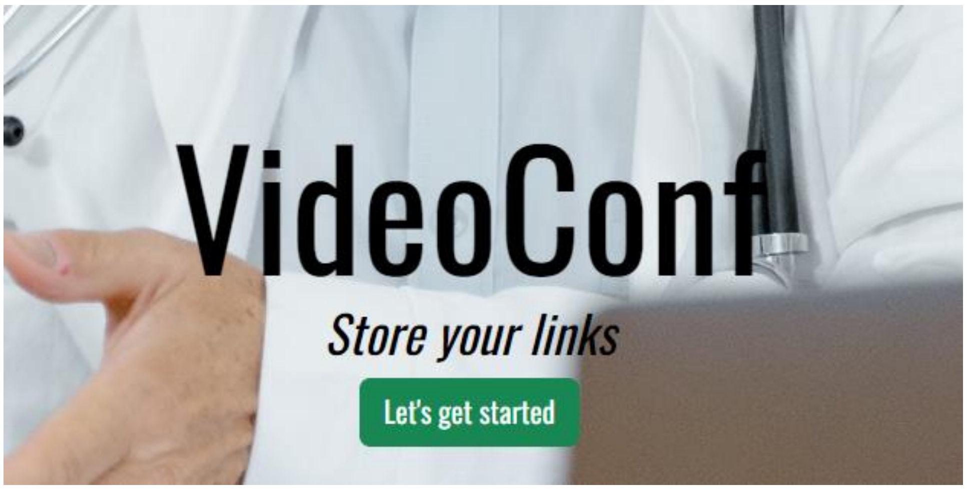


Figure 11-24. Let’s get started button

This button redirects to the login part of the application if the user is not logged in, or to his profile if the user is logged in.

REFERENCES

[1]. Luque, J., n.d. Videoconferencia. Acta.es. [en línea], [sin fecha]. [Consulta 15 agosto 2022]. Disponible en: https://www.acta.es/medios/articulos/comunicacion_e_informacion/057057.pdf

[2]. Weekly-geekly-es.imtqy.com. 2022. Una breve historia de la videoconferencia: desde el comienzo hasta el uso comercial completo. [en línea], [sin fecha]. [Consulta 15 agosto 2022]. Disponible en: https://weekly-geekly-es.imtqy.com/articles/465459/index.html

[3]. Baudot, O., 2018. Historia de la Videoconferencia. Linkedin.com. [en línea], [sin fecha]. [Consulta 15 agosto 2022]. Disponible en: https://www.linkedin.com/pulse/historia-de-la-videoconferencia-%C3%B3scar-b-/?originalSubdomain=eshttps://www.linkedin.com/pulse/historia-de-la-videoconferencia-%C3%B3scar-b-/?originalSubdomain=es

[4]. RDSI: Telefonía y Servicios Digitales. [en línea], [sin fecha]. [Consulta: 31 agosto 2022]. Disponible en: http://www.consulintel.es/Html/Tutoriales/Articulos/rdsi.html

[5]. Modo de transferencia asíncrona. [en línea], [sin fecha]. [Consulta: 31 agosto 2022]. Disponible en: https://es.wikipedia.org/wiki/Modo_de_transferencia_as%C3%ADncrona

[6]. Henning Schulzrinne y Stephen L. Casner. RTP: A Transport Protocol for Real-Time Applications. RFC 1889. 01-01-1996 [en línea]. [Consulta 31 agosto 2022]. Disponible en: https://www.rfc-editor.org/rfc/rfc1889

[7]. Mark Handley, Henning Schulzrinne, Eve Schooler y Jonathan Rosenberg. SIP: Session Initiation Protocol. RFC 2543. 01-03-1999. [en línea]. [Consulta 31 agosto 2022]. Disponible en: https://www.rfc-editor.org/rfc/rfc2543

[8]. Qué es Asterisk. Software de código abierto para telefonía IP. [en línea], [sin fecha]. [Consulta: 31 agosto 2022]. Disponible en: https://www.masip.es/blog/que-es-asterisk/

[9]. Linphone: Conoce esta aplicaciónde Vo-IP multiplataforma. [en línea], [sin fecha]. [Consulta: 31 agosto 2022]. Disponible en: https://www.redeszone.net/2012/09/01/linphone-conoce-esta-aplicacionde-vo-ip-multiplataforma/

[10]. Skype: | internet & marketing. [en línea], [sin fecha]. [Consulta: 31 agosto 2022]. Disponible en: https://www.icesi.edu.co/blogs_estudiantes/alejume7/2011/07/31/skype/

[11]. La WebRTC y sus principales beneficios empresariales. [en línea], [sin fecha]. [Consulta: 31 agosto 2022]. Disponible en: https://www.fonvirtual.com/blog/la-webrtc/

[12]. Historia de la telemedicina desde sus inicios hasta hoy. [en línea], [sin fecha]. [Consulta: 20 agosto 2022]. Disponible en: https://clinic-cloud.com/blog/historia-de-la-telemedicina/.

[13]. Preguntas frecuentes sobre Google Meet, Google Chat y Hangouts. [en línea], [sin fecha]. [Consulta: 21 agosto 2022]. Disponible en: https://support.google.com/a/users/answer/9845199?hl=es

[14]. El Gobierno de Castilla-La Mancha pone en marcha un procedimiento para facilitar la comunicación entre pacientes ingresados y sus familiares. [en línea], [sin fecha]. [Consulta: 21 agosto 2022]. Disponible en: https://sanidad.castillalamancha.es/saladeprensa/notas-de-prensa/el-gobierno-de-castilla-la-mancha-pone-en-marcha-un-procedimiento-para

[15]. COVID-19: Son Llàtzer pone en marcha el proyecto «Unidos» para comunicar a los pacientes aislados con sus familiares [en línea], [sin fecha]. [Consulta: 21 agosto 2022]. Disponible en: https://www.caib.es/pidip2front/jsp/es/ficha-convocatoria/9433120

[16]. Videollamadas para esquivar el aislamiento de los pacientes ingresados: una aplicación ofrece visitas virtuales a los familiares. [en línea], [sin fecha]. [Consulta: 21 agosto 2022]. Disponible en: https://www.elmundo.es/ciencia-y-salud/salud/2020/03/18/5e722fae21efa0a1178b458f.html

[17]. CÉSAR FERNÁNDEZ, J., 2017. Patrones de diseño en software (III) : MVC | Apple Coding. 19-01-2017 [en línea]. [Consulta: 27 agosto 2022]. Disponible en: https://applecoding.com/guias/patrones-diseno-software-mvc.

[18]. JavaScript. [en línea], [sin fecha]. [Consulta: 20 agosto 2022]. Disponible en: https://es.wikipedia.org/wiki/JavaScript.

[19]. 1.2. Breve historia (Introducción a JavaScript). [en línea], [sin fecha]. [Consulta: 20 agosto 2022]. Disponible en: https://uniwebsidad.com/libros/javascript/capitulo-1/breve-historia.

[20]. LÓPEZ SOSA, A., [sin fecha]. Introducción a express js. 23/05/2019 [en línea]. [Consulta: 20 agosto 2022]. Disponible en: https://medium.com/@aarnlpezsosa/introducción-a-express-js-a1ebe16dbcf4.

[21]. Comenzando con handlebars. Programación en Castellano. ICE [en línea]. [Consulta: 04 septiembre 2022]. Disponible en: https://programacion.net/articulo/comenzando_con_handlebars_1376

[22]. Node.js: ¿Qué es y para que sirve NodeJS? [en línea], [sin fecha]. [Consulta: 20 agosto 2022]. Disponible en: https://www.netconsulting.es/blog/nodejs/.

[23]. Servicios de cloud computing | Google Cloud [en línea], [sin fecha]. [Consulta: 28/08/2022]. Disponible en: https://cloud.google.com/?hl=es

[24]. ¿Qué es WebRTC? – Definición de TrueConf [en línea], [sin fecha]. [Consulta: 20 agosto 2022]. Disponible en: https://trueconf.com/es/webrtc.html.

[25]. WebSockets – Referencia de la API Web | WDM [en línea]. [Consulta: 04 septiembre 2022]. Disponible en: https://developer.mozilla.org/es/docs/Web/API/WebSockets_API

[26]. Henning Schulzrinne y col. RTP: A Transport Protocol for Real-Time Applications. RFC 3550. 01-07-2003 [en línea]. [Consulta: 21 agosto 2022]. Disponible en: https://rfc-editor.org/rfc/rfc3550.txt.

[27]. Ali C. Begen y col. SDP: Session Description Protocol. RFC 8866. 01-01-2021 [en línea]. [Consulta: 21 agosto 2022]. Disponible en: https://rfc-editor.org/rfc/rfc8866.txt.

[28]. Ari Keränen, Christer Holmberg y Jonathan Rosenberg. Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal. RFC 8445. 01-07-2021 [en línea]. [Consulta 21 agosto 2022]. Disponible en: https://rfc-editor.org/rfc/rfc8445.txt.

[29]. Marc Petit-Huguenin y col. Session Traversal Utilities for NAT (STUN). RFC 8489. 01-02-2020 [en línea]. [Consulta: 21 agosto 2022]. Disponible en: https://rfc-editor.org/rfc/rfc8489.txt.

[30]. Tirumaleswar Reddy.K y col. Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN). RFC 8656. 01-02-2020 [en línea]. [Consulta: 21 agosto 2022]. Disponible en: https://rfc-editor.org/rfc/rfc8656.txt.

[31]. Cómo crear un módulo Node.js [en línea]. [Consulta: 27 agosto 2022]. Disponible en: https://www.digitalocean.com/community/tutorials/how-to-create-a-node-js-module-es

[32]. npm [en línea]. [Consulta: 27 agosto 2022]. Disponible en: https://www.npmjs.com/

[33]. Trickle ICE [en línea]. [Consulta: 12 agosto 2022]. Disponible en: https://webrtc.github.io/samples/src/content/peerconnection/trickle-ice/