Universidad de Oviedo

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN.

GRADO EN INGENIERÍA INFORMÁTICA EN TECNOLOGÍAS DE LA INFORMACIÓN

ÁREA DE ARQUITECTURA Y TECNOLOGÍA DE COMPUTADORES

MONITORIZACIÓN DE LA COMUNICACIÓN VERBAL EN SESIONES PRESENCIALES DE TRABAJO EN EQUIPO

MEMORIA

D. DÍAZ RUÉNEZ, José Carlos
TUTOR: D. Francisco José Suárez Alonso

FECHA: Julio de 2021

ÍNDICE DE LA MEMORIA

1. Introducción.

En las sucesivas reuniones que se pueden dar durante la realización de un proyecto, independientemente del ámbito, es importante que todos los miembros que componen dicho equipo puedan dar su punto de vista y participar activamente en la misma medida, con el objetivo de que dichas ideas contribuyan a poder conseguir el propósito determinado inicialmente.

Sin embargo, en multitud de ocasiones, dicha premisa no se cumplirá, debido a que posiblemente exista una diferencia notable en el nivel de participación entre miembros del mismo equipo.

Esta situación no aplica únicamente al ámbito del desarrollo de proyectos. Esta circunstancia es perfectamente extrapolable a otros campos fuera del ámbito profesional. Por ejemplo, es posible ubicar esta problemática en el ámbito educativo, como pueden las reuniones grupales de alumnos. Y en un plano más abstracto todavía, esto es aplicable a cualquier reunión.

En el conjunto de situaciones en las que se dan dichas condiciones, se repiten dos conjuntos principales: en primer lugar, una o varias personas que asumen el rol de moderadores, cuyo deseo es poder medir la participación de los integrantes del equipo de trabajo; en segundo lugar, las personas que componen el equipo que será monitorizado.

La finalidad de este TFG es el desarrollo de una aplicación prototipo que permita poder monitorizar en tiempo real una sesión presencial de trabajo en equipo, con la posibilidad de poder analizar a posteriori la información que se recabe en esta, con el objetivo de determinar el grado de participación de cada uno de los miembros del equipo de trabajo.

2. Objetivos y alcance.

El propósito de este apartado es proporcionar una descripción de los objetivos y el alcance de este TFG. Dichos objetivos y alcance han sido establecidos como los siguientes.

2.1.- Monitorización de la participación en sesiones presenciales de trabajo en equipo.

El primer objetivo fundamental que se busca en este proyecto es la creación de una aplicación que permita recibir de cada uno de los miembros información acerca del estado de participación en la reunión.

La aplicación cliente se encontrará en todo momento capturando el posible sonido emitido, o lo que es lo mismo, si la persona asociada a dicho usuario está comunicándose o no. Posteriormente, será un servidor el que se encargue de, una vez recibida esa información, computarla de forma que se decida si se ha producido una comunicación o no. La persona responsable tendrá la capacidad de poder visualizar en tiempo real cómo se están desarrollando todas las reuniones que se estén dando.

2.2.- Capacidad de análisis a posteriori de la secuencia de intervenciones durante una sesión de trabajo.

El segundo de los objetivos fundamentales que se persigue en este proyecto es, que en base a la información recibida por cada uno de los miembros que componen los diferentes equipos de trabajo, se pueda procesar esta información a posteriori, a partir de un fichero que contendrá cómo se ha ido desarrollando la comunicación en dichos equipos.

La información representada en dicho fichero se presentará en formato binario, esto es, asociando un valor a cada una de las dos situaciones que pueden darse en dicha reunión para cada uno de los miembros del equipo. Dichos valores se establecerán como un “1” si

se ha detectado una comunicación asociada a dicha persona en un instante determinado de tiempo, o como “0” en caso contrario.

Finalmente, si bien es cierto que la persona que ha supervisado dicha sesión tiene en última instancia la capacidad de realizar el análisis que considere pertinente, la propia aplicación, una vez que finaliza la reunión, realiza un procesamiento mínimo de la información recibida. Dicha información se muestra al supervisor de forma estructurada por cada uno de los equipos monitorizados y, al mismo tiempo, también se genera un fichero por cada uno de los equipos con la misma información.

3. Estudios y análisis previos.

El primer paso es poder determinar si existen registros de trabajos realizados anteriormente cuya temática sean igual o similar al objeto de este TFG, que es poder estimar el porcentaje que una persona participa o no en una reunión cualquiera, en una primera investigación. En dicho proceso, si bien no se ha podido encontrar ningún desarrollo que fuera exacto al que se plantea en este proyecto, sí que se han encontrado estudios y aplicaciones que plantean ideas que han podido ayudar al desarrollo de esta idea.

3.1.- Detección, captura y procesamiento del audio.

En un estudio realizado en 2011 [1], se planteó el desarrollo de un sistema que permitiese la monitorización del habla en tiempo real con el fin de poder determinar el número de PPM (palabras por minuto) de una persona y en función de dicho parámetro, poder adecuar la velocidad de la conversación. El escenario planteado para usar dicho software era la de un Call Center. En última instancia, podría extrapolarse como un tipo de reunión. Dicho sistema dependía en primer lugar del número de sílabas por segundo y el factor de conversión (dependiente de cada idioma), durante un minuto. En función de estos dos parámetros, la aplicación daba indicaciones al usuario (en el caso de la investigación, a los operadores del Call Center) que usasen dicha aplicación, de cómo debía adecuar la velocidad del habla en la conversación.

En un segundo estudio realizado en 2013 [2], se tomó la idea planteada en 2011 para poder evolucionar la misma y que pudiera funcionar en dispositivos Android. En esta segunda publicación, mencionan que el algoritmo planteado anteriormente es computacionalmente exigente desde el punto de vista de que la aplicación se está ejecutando sobre un sistema operativo móvil y por tanto falla en procesarlo en tiempo real (únicamente se hace referencia a la frecuencia de reloj del dispositivo, 650 MHz). Si bien es cierto que uno de los objetivos de este TFG sería que la aplicación cliente pueda funcionar en dispositivos

Android, esta tarea a priori no debería suponer ningún problema considerando la capacidad computacional de los equipos móviles actuales.

Debido a que la una de las funcionalidades de la aplicación a desarrollar se basa en el audio captado del dispositivo de audio pertinente, es muy importante poder determinar aproximaciones o estudios realizados previamente con el fin de poder conseguir una eficiencia mayor a la hora de poder cuadrar la captura de audio y el envío al nodo donde se procesará la información, para poder construir así un sistema lo más cercano al tiempo real. En este caso, la librería de audio a emplear será *sounddevice*, que proporciona las funcionalidades necesarias para la realización de este proyecto. Dicho módulo de Python permite mediante diferentes rutinas, capturar el audio del dispositivo y transformarlo en información digital. Sin embargo, las rutinas de audio deben tener un tiempo prefijado, por lo que no es posible modificar in situ la duración que va a tener esta y, por ende, es un factor determinante a la hora de cómo poder diseñar el sistema cliente.

Como simplemente se desea capturar si existe o no sonido, no es necesario más que un canal de audio donde se recoja la información. La frecuencia de muestreo será de 8000 Hz, debido a que simplemente se desea saber si hay comunicación o no.

En primer lugar, se determinó que la forma de medir si un usuario estaba comunicándose o no, era recoger el último elemento del array que devolvía en cada instante la rutina de audio que captura la información del dispositivo de audio correspondiente. Posteriormente, esta información se envía al servidor, donde se forman cadenas binarias que contienen la información capturada por el cliente durante un segundo y es el servidor quien decide, en base a la información recibida por parte del cliente, si almacena ese instante como que ha habido comunicación o no.

3.2.- Envío de la información.

En este caso, el sistema a desarrollar no se trata de una aplicación VoIP, pues no existe una comunicación directa entre dos nodos cualesquiera y únicamente lo que se precisa es registrar la información proveniente de la fuente de sonido. Sin embargo, es muy

importante considerar factores como son la latencia o el periodo que tarda el cliente en el envío de la información al nodo servidor para que este pueda posteriormente procesarla. Con relación a esto último, se debe precisar el protocolo a emplear para el envío de información.

En este caso, se determinó que el protocolo a emplear debía ser TCP, aun considerando qué [3] "As TCP requires buffering to make sure that lost packets will have the chance to get retransmitted before an interruption of the stream occurs, delays will be imposed when using TCP" (C. Lauri, J. Malmgren, 2015, pp 5).

No obstante, como se mencionó anteriormente, el sistema a desarrollar no depende de qué captura el dispositivo de audio o no, sino del registro de si hay información o no. Por ende, es muy importante que no se produzcan pérdidas de paquetes en la comunicación, pues éstas podrían suponer la diferencia entre considerar que un usuario se esté comunicando o no. En el caso de que el sistema implementase un chat de voz de manera interna, similar a cualquier aplicación VoIP, y se precisase de la comunicación entre dos nodos cualesquiera y del envío de información al nodo principal para su procesamiento, entonces sí que se debería considerar el uso del protocolo UDP para minimizar los tiempos en las comunicaciones, aunque esto supusiera la pérdida de información puntualmente.

La tecnología Bluetooth de los dispositivos de captura de audio es recomendable que sea lo más reciente posible, a poder ser, una versión superior a 4.0 con el objetivo de que sea lo más eficiente respecto a la autonomía del dispositivo. Es obligatorio considerar que, así como existe una pequeña latencia en las comunicaciones, también existe entre el dispositivo de audio y el dispositivo objetivo, como podría ser un ordenador o un teléfono móvil. Sin embargo, se asumirá que la latencia existente entre estos dos nodos es siempre constante y lo suficientemente pequeña como para no afectar al rendimiento global del sistema. Únicamente se debe tener en cuenta que el nivel de audio que capture dicho dispositivo no depende de la aplicación que se pueda desarrollar, el nivel de sonido que determine la configuración de sonido del sistema operativo.

3.3.- Sincronización de relojes.

En una aplicación distribuida, cada uno de los diferentes nodos secundarios que la componen (entendiendo como nodo, cada equipo que es capaz de enviar la información capturada por el dispositivo de audio al nodo principal) pueden ser diferentes.

Cada uno de los nodos, consta internamente de un RTC (*Real Time Clock*), que se trata de un dispositivo que permite mantener la hora actual del sistema. La mayoría de los relojes de este tipo emplean un oscilador de cristal que es estabilizado, dando como valor de la frecuencia, la misma que se usa en los relojes de cuarzo, 32768 kHz. Dichos relojes tienen un consumo bajo de energía, no se ven afectados por las interrupciones del sistema y además son más precisos que otros métodos. En contraparte, dichos relojes se ven afectados fácilmente por factores como son la temperatura o si el cristal no está ajustado correctamente, incurriendo en que por cada oscilación se produzca un desfase.

Por tanto, para que todo el sistema pueda funcionar lo mejor posible, es necesario establecer fórmulas que permitan controlar el desfase que pueda existir entre dos dispositivos diferentes. Pero la sincronización de todos los nodos secundarios debería realizarse respecto del nodo principal, garantizando que cada uno de los diferentes equipos que compusieran la red, funcionan a la vez que el nodo principal, que sería el servidor donde está establecida la aplicación de monitorización. Por tanto, el objetivo que se persigue sería lograr una sincronización externa. Con el fin de lograr dicha tarea, se pueden emplear fórmulas que se basen en el empleo de protocolos de sincronización de tiempo, como pueden ser PTP (*Precision Time Protocol*) o NTP (*Network Time Protocol*).

- *Precision Time Protocol*: este protocolo es perfecto en situaciones en las que se requieran sistemas muy precisos, como pueden darse en entornos industriales. Por tanto, a diferencia de NTP, PTP es recomendado que se implemente mediante hardware de forma que se consiga el mejor rendimiento posible. En este protocolo, la selección del servidor que servirá como referencia se realiza mediante el algoritmo del mejor maestro, que se basa en qué si el nodo cliente es más preciso que los nodos ya disponibles, se convierte en un maestro y en caso contrario, en un esclavo. Además, quien inicia el flujo de mensajes es el maestro [4]:

1. El reloj maestro, que es quién inicia la comunicación, guarda el momento del envío T0 al enviar un mensaje de sincronización.
2. El cliente recibe el mensaje y guarda en T1 el momento en el que recibió dicho mensaje.
3. El reloj maestro reenvía un mensaje de seguimiento que contiene el valor T0. Internamente, el cliente realiza la operación:

$$\theta = T1 - T0 \tag{3.1}$$

Donde en la ecuación 3.1, T1 es el momento de recepción del mensaje por parte del cliente, y T0 el momento del envío del mensaje de sincronización por parte del servidor.

Dicho valor se suma al reloj interno, con el fin de ajustarlo. Ahora bien, no se tiene en cuenta la latencia existente debido a las comunicaciones de red, y por tanto es necesario calcularla.

4. Para calcular el tiempo que se ha perdido entre envío y recepción del mensaje, el cliente envía un mensaje de delay de petición al maestro, y guarda el momento, se supone T2. El maestro lo recibe en T3.
5. El maestro envía de vuelta al cliente un mensaje que contiene el instante T3. El cliente, una vez que recibe la respuesta al mensaje de retraso de petición, calcula el tiempo:

$$\theta = \frac{T2 + T1 - T0 - T3}{2} \tag{3.2}$$

Donde en la ecuación 3.2, T0 es el momento del envío del mensaje de sincronización por parte del servidor, T1 el momento de recepción del mensaje por parte del cliente, T2 el momento en el que el mensaje de delay es enviado desde el cliente y T3 el momento en el que el maestro recibe dicho mensaje.

- *Network Time Protocol*: este protocolo es el que usan la mayoría de los sistemas operativos hoy en día. Está basado en una arquitectura cliente – servidor. A diferencia de PTP, este protocolo está basado puramente en el software. Los servidores están organizados según quince niveles diferentes de estratos. En NTP versión 3, donde el 0 es un dispositivo de referencia directamente conectado y 16 el nivel más bajo y de peor calidad [5]. Un estrato representa el cómo de desfasado está el reloj interno de un dispositivo respecto a un reloj de referencia. El paso de mensajes es el siguiente:

  1. El cliente inicia la comunicación, pidiendo al servidor que le envíe una marca temporal en un instante T0. La marca temporal del cliente es almacenada.
  2. El servidor recibe la petición del cliente, en un tiempo T1, y en un instante T2 le envía la información de respuesta de vuelta al cliente que inició la comunicación.
  3. La información llega al cliente en un instante T3. Ahora que se dispone de cuatro marcas temporales diferentes, se procede a calcular el retardo entre ambos relojes siguiendo la ecuación 3.3:

$$\theta = \frac{(T1 - T0) + (T2 - T3)}{2} \tag{3.3}$$

Donde en la ecuación 3.3, T0 es el momento en el que el cliente requiere una marca de tiempo, T1 el momento en el que el servidor recibe dicho mensaje, T2 el momento en el que la respuesta se envía de vuelta al cliente y T3 el momento en el que dicho mensaje llega al cliente.

Dependiendo del valor obtenido, puede ser necesario o no realizar un sobreajuste en el reloj del cliente, dependiendo de cuál sea el baremo para considerar que hay demasiado desfase entre el reloj del cliente y el reloj del servidor.

Como la aplicación resultado del proyecto se desea que funcione en ordenadores además de en dispositivos móviles, se debe recalcar que no todos los sistemas operativos tienen el mismo ajuste y precisión. Por ejemplo, en operativos Windows 10 y recientes, la precisión que se logra es de 1 ms en el mejor de los casos [6].

Siguiendo las directrices el NIST (*National Institute of Standards and Technology*), podría establecerse que una precisión correcta para un proyecto de este alcance, podrían ser 50 ms o menos [7].

De manera adicional al protocolo de sincronización de relojes, se deben tener en cuenta dos factores extra que, en sistemas distribuidos y mayormente dependientes en el tiempo, pueden afectar al rendimiento. Dichas variables son el sesgo del reloj y la deriva del reloj. El sesgo de reloj se produce a raíz de que la señal del reloj no llega a todos los componentes a la vez. Esto puede deberse a los componentes empleados en la fabricación de los conductores o a la distancia que hay desde el RTC hasta cada uno de los diferentes componentes. Esto produce un desajuste entre dos relojes diferentes y da como resultado la pérdida de sincronización.

El segundo problema es la deriva de reloj, y se da cuando el reloj interno oscila a una frecuencia diferente de la de referencia. Este problema es importante que, conforme vaya aumentando el tiempo, se vaya corrigiendo periódicamente mediante mensajes de sincronización según el protocolo seleccionado, pues éste va degenerando con el paso del tiempo de modo que puede desembocar en que dos equipos cualesquiera, por ejemplo, se encuentren en una situación donde uno de ellos está adelantado en el tiempo respecto al otro. Adicionalmente, dos nodos cualesquiera no tienen por qué tener el mismo RTC, debido a que su oscilador interno puede ser de diferente tipo.

Si se supone un RTC cualquiera, C, en un momento determinado t, C(t) será el valor en dicho instante. Un reloj preciso cumplirá la expresión de la ecuación 3.4:

$$C(t) = \frac{dC}{dt} = 1 \tag{3.4}$$

Donde C(t) es el valor de dicho reloj en un momento determinado. Si dicho reloj estuviera adelantado, el valor sería > 1, y < 1 en caso de que estuviera atrasado, siendo ρ el desfase del reloj como puede apreciarse en la figura 3.1.

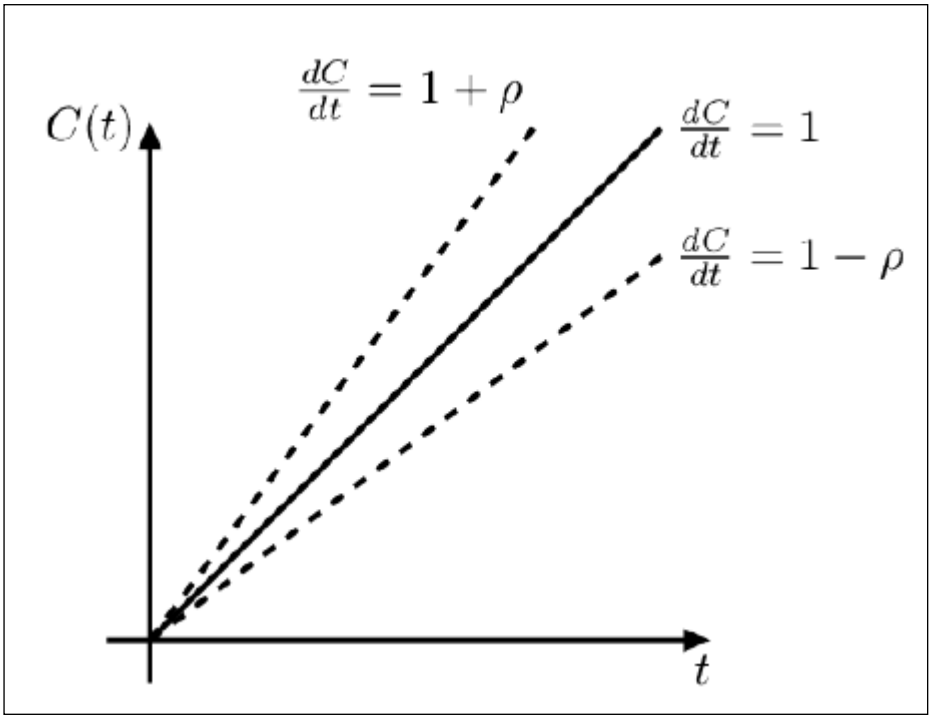

Figura 3.1.- Representación gráfica de la deriva de un reloj.

Para ejemplificar cómo afecta esto a sistemas cuya arquitectura se basa en el modelo cliente – servidor, es posible suponer una situación. Como se mencionó anteriormente, la oscilación de los RTC habitualmente está ajustada a 32768 kHz. En condiciones perfectas, dicho valor nominal no variaría, sin embargo, esto no es posible, y siempre existirá una desviación. Supongamos que dicha desviación sea 20 ppm (partes por millón), o lo que es equivalente, en un millón de ciclos del reloj, existirá un error de 20 unidades. Supuesto un día de 24 horas, si le aplicamos el desfase propuesto mediante la ecuación 3.5:

$$Desfase = 20 * 10^{-6} * 86400 = 1{,}728\ s \qquad (3.5)$$

O lo que es lo mismo, al cabo de 24 horas, el reloj interno tendrá un desfase de 1,728 segundos y por tanto podrían llegar a existir una diferencia de dos segundos entre nodos. Incluso en el peor de los casos, que el error del reloj fuera de 100 unidades como se indica en la ecuación 3.6:

$$Desfase = 100 * 10^{-6} * 86400 = 8{,}640\ s \quad (3.6)$$

Habiendo definido los factores de sincronización y de red, la mejor solución respecto a protocolos a usar sería hacer uso de NTP para la sincronización de los diferentes equipos en la red, pues no precisa de equipo hardware adicional y si bien es cierto que se pierde algo de precisión, ésta sigue siendo aceptable. Además, NTP funciona mediante software y por tanto es fácilmente portable. Si bien el servidor que se usaría como referencia a la hora de la sincronización es aquél donde se esté ejecutando la herramienta de monitorización, podría indicare cualquier otro servidor que estuviera en estratos superiores, por ejemplo, cualquiera de los servidores del NIST.

Sin embargo, teniendo en cuenta los resultados obtenidos en el estudio de cómo la deriva de reloj puede afectar al sistema y el objetivo final de este proyecto, la solución propuesta no precisa de la realización de una sincronización periódica. La sincronización inicial de todos los nodos de la red se realiza a través de evento de Python. El funcionamiento es simple, con dos estados como son Set() que indicaría que la sesión está activa o la negación de la misma para indicar que se ha finalizado.

En el caso de que el proyecto abarcase una red más extensa, como podría ser internet, podría surgir el problema de la latencia en las comunicaciones en red, provocando que un paquete llegase fuera de tiempo. Incluyendo una marca de tiempo asociada a cada paquete, es posible para el servidor poder gestionar de una forma más eficiente la información, dando la capacidad de recolocarla según esta. Independientemente de esto, también debe considerarse la posibilidad de emplear protocolos de sincronización, como podría ser NTP, para coordinar adecuadamente los diferentes nodos de la red.

4. Desarrollo del proyecto.

El objetivo de este apartado es describir algunas partes relevantes del desarrollo del proyecto.

4.1.- Captura de audio.

Una vez comenzado el desarrollo del proyecto, la funcionalidad de captura de audio estaba codificada según lo establecido en el apartado de estudios previos. Esto no es muy eficiente desde el punto de vista de que se registre si un usuario está comunicándose o no, debido a que en ningún momento se tomaba en consideración los decibelios que pudieran asociarse a la comunicación verbal.

Por tanto, finalmente se decidió que lo mejor era crear un sistema donde se tuviera en consideración la información global del array que devolvía la rutina de audio de *sounddevice* y transformarla en un valor que representase la relación entre la presión del sonido capturado por el dispositivo en un momento dado y la presión del aire. Es posible extrapolar la media cuadrática a la información recogida [8] en el array mencionado:

$$Presión_{sonido} = \sqrt{\frac{1}{N}(x_1^2 + x_2^2 + x_3^2 + \cdots + x_N^2)} \qquad (4.1)$$

Donde en la ecuación 4.1, x_N^2 un elemento elevado al cuadrado, de un conjunto de valores que representa la información del sonido tomada por la rutina de sonido de la aplicación cliente, y N el número total de muestras computables. Tomando como referencia, que la presión del aire equivale a $2 * 10^{-6}$ Pa [9]. La detección del audio se regirá por:

$$Detección\ de\ audio = 20 * \log\left(\frac{Presión_{sonido}}{Presión_{aire}}\right) \qquad (4.2)$$

En cualquier caso, el servidor procesará la información recibida durante dicho segundo para decidir si se trata de una comunicación o no, pero empleando la ecuación 4.1, permite descartar en mayor medida ruidos externos que sean captados por el dispositivo de audio y que pudieran suponer una falsa comunicación.

4.2.- Visualización de la información recibida en tiempo real.

Es necesario que el proceso entre la recepción de la información desde los clientes y la visualización por parte del usuario sea lo más rápida posible, de forma que exista la menor latencia entre llegada y dibujo.

En primera instancia durante el desarrollo, se consideró el uso de la librería de gráficos *Matplotlib* y en concreto, hacer uso de las funciones de animación disponibles en su API. Sin embargo, esta opción fue descartada, debido a qué, para 8 usuarios, el tiempo que el sistema necesitaba para poder actualizar todas las gráficas de todos los usuarios era de aproximadamente ≈0,74 segundos. Es decir, a un mayor número de usuarios, un mayor tiempo sería necesario para poder ir actualizando todas las gráficas. Además, por emplear dicha funcionalidad, el sistema en su totalidad pasaba a consumir una cantidad elevada de los recursos del equipo donde se realizaba tanto el desarrollo como las pruebas.
En base a esta información y estas limitaciones, se decidió hacer uso de otra librería de gráficos para Python: *pyqtgraph*. Dicha librería, si bien es mucho más simple de lo que puede ser *Matplotlib*, al ser más ligera, permite cumplir tanto con los requisitos de usuario como de sistema.

Esta estructura, permite tener un mejor control de cómo se van actualizando las diferentes gráficas. Además, el emplear dicha librería permite una ganancia de rendimiento considerable respecto a *Matplotlib*, pasando únicamente a necesitar ≈0,10 segundos en actualizar todas las gráficas, considerando una reunión de 8 personas.

La funcionalidad de poder monitorizar en tiempo real cómo se están desarrollando las reuniones, implica qué para cada uno de los equipos se visualizan tantas gráficas como usuarios tenga dicho equipo.

Por tanto, la ventana asociada a cada equipo, para un mejor manejo de estas, debe tratarse como un objeto. En este caso, la implementación seguida es la establecida según la POO (Programación Orientada a Objetos). Se estructura de la siguiente forma:

- Cada una de las diferentes ventanas asociadas a los diferentes equipos se trata como un objeto diferente, cada uno, identificado por el nombre del equipo. Se crean tantas gráficas como usuarios tenga el equipo, independientemente de que estén o no estén conectados al momento en el que se inicia la monitorización.
- Cada una de las gráficas dentro de una ventana cualquiera se trata como un objeto a su vez, identificada por el usuario correspondiente. En base a ello, cada gráfica obtiene de la estructura de almacenamiento correspondiente el último elemento recibido del usuario con dicho identificador. Ese elemento, se almacena de forma interna en el objeto mediante un array, que servirá para poder realizar la representación.
- De manera externa, no perteneciendo a ninguna de las dos clases expuestas anteriormente, una función se encarga de realizar la actualización de todas las gráficas, llamando a los métodos pertinentes de cada clase. Para ello, se basa en iniciar un cronómetro propio de la aplicación a la hora de comenzar la sesión, para que aproximadamente cada segundo, se actualice la información.

Esta misma metodología se emplea también, cuando a posteriori se realiza un procesamiento de la información recabada en un fichero de una reunión cualquiera. Esto es, realizado un procesamiento a partir de un fichero, se obtiene un fichero con extensión ".txt" donde aparece la información estadística obtenida a partir del mismo, un documento con extensión ".pdf" que muestra de forma gráfica la información contenida en el fichero y finalmente, también se despliega una ventana, idéntica a la misma que se usa en la monitorización en tiempo real, es decir, una ventana con tantos usuarios cómo existan en el fichero.

De esta forma, es posible tener las capacidades de zoom y desplazamiento horizontal a través de la gráfica de cada usuario, de forma que es posible estudiar cada caso de manera

concreta. Este método es más eficiente que el PDF, que simplemente es la representación gráfica de lo que se ve en las gráficas sin posibilidad de interacción con estas.

4.4.3.- Acceso a la información.

Uno de los problemas surgidos del desarrollo del proyecto es quizás dónde almacenar la información recibida de cada uno de los diferentes clientes. Para esto, es necesario conocer adecuadamente los tiempos de acceso de las estructuras de datos existentes en Python.

Para poder almacenar todos los usuarios, cuando se cargan por primera vez, se hace en un diccionario. Los diccionarios tienen de media un coste O(1) para operaciones básicas. Suponiendo que el número de usuarios a monitorizar fuera 50, podría considerarse aún que el coste de acceder a la posición 50 del diccionario sería O(1). Basándose en esta premisa, y en que el peor caso es O(n) pero con coste amortizado, es posible hacer eso de dicha estructura para almacenar en primera instancia toda la información de todos los usuarios. En Python, los tipos de estructuras de datos nativos son las listas, sets, tuplas, colas y diccionarios.

Emplear un set queda totalmente descartado, pues no permite elementos repetidos y, por tanto, no es posible almacenar en orden los elementos según se vayan recibiendo del cliente.

Una cola podría funcionar en primera instancia, pues el acceso al último elemento tiene coste lineal. También proporciona métodos para poder realizar la inserción si un paquete llega fuera de lugar. El problema reside en que las colas tienen un tamaño asignado y, por tanto, existen dos posibilidades, asignar un tamaño fijo al comienzo de la sesión, en base a la duración, o ir modificando el tamaño de la cola gradualmente, lo cual supone un coste extra.

Un diccionario, si bien tiene coste O(1), no parece encajar con el problema que queremos resolver. Esto es, considerando que la estructura de datos escogida fuese un diccionario, las marcas de tiempo podrían establecerse como las claves, y el valor asociado a cada clave, es

el valor. La inserción en un diccionario tiene coste O(1). No obstante, si un paquete llegase fuera de lugar, se debería calcular el retraso del mismo respecto a la hora actual del sistema. Suponiendo que llegase 2 segundos más tarde de lo que debería, se debería reinsertar la nueva información en la posición correspondiente del diccionario, y los pares de clave-valor posteriores, desplazarlos una posición. Si esto se repite comúnmente, es posible que el coste de emplear un diccionario crezca.

Por tanto, finalmente se tienen dos opciones disponibles: emplear una tupla, o emplear una lista. No obstante, las tuplas presentan un problema, y es que son inmutables, de forma que no se ofrecen operaciones que cambien la representación.

Por tanto, esto nos deja las listas como última opción para poder almacenar la información recibida de los clientes. El coste de añadir a una lista es O(1), y una inserción, en el peor de los casos es O(n). Como únicamente se añadiría la información conforme vaya llegando al cliente, y la inserción de elementos que hayan llegado fuera de tiempo es directa, pues simplemente se comprueba el retraso existente y se añade la información en la posición correspondiente de la lista, es posible asumir estos costes.

Por tanto, la representación escogida para almacenar la información es emplear el diccionario donde se cargan los usuarios, donde las claves son los identificadores, y dentro de cada entrada, el valor sea un diccionario que almacene el nombre y los apellidos del usuario, el socket a través del cual se efectúa la comunicación y una lista, que es donde se almacena la información proveniente de los usuarios a través de la aplicación cliente.

4.4.4.- Desarrollo en Android.

Uno de los objetivos planteados en este proyecto, era que se pudieran aprovechar los avances en los sistemas operativos móviles con el fin de poder ejecutar la aplicación cliente en un dispositivo que ejecutase Android.

Para poder llevar a cabo esta tarea, era necesario que se instalase cualquier aplicación que permitiera ejecutar scripts de Android. En este caso, se optó por emplear dos aplicaciones: *Termux*, y *Pydroid 3*.

Haciendo uso de *Termux* (habilita una consola de comandos como en cualquier sistema operativo), es necesario instalar Python en primer lugar y una vez instalada la aplicación, proceder con la instalación de las librerías de Python necesarias para poder ejecutar la aplicación cliente: *numpy* y *sounddevice*. En el caso de *numpy*, el tiempo de instalación necesario era bastante elevado, llegado a tomar más de 20 minutos aproximadamente para esta tarea. Posteriormente, ya era posible ejecutar la aplicación cliente, de forma exitosa, llegado a visualizar el menú contextual del mismo. Sin embargo, a la hora de ejecutar cualquier funcionalidad relacionada con capturar el sonido del micrófono, como podría ser realizar una prueba de sonido, las funciones implementadas haciendo uso de la librería *sounddevice* no son capaces de capturar el audio del micrófono, de forma que siempre arrojan un valor 0 como indicativo de que no hay sonido.

Una vez descartado el primer método, se intentó hacer uso de otra aplicación como es *Pydroid 3*. Dicha aplicación, a diferencia de *Termux*, consta de una ventana de comandos más limitada y un intérprete de Python. No obstante, mejora considerablemente la instalación de librerías no presentes por defecto en Python, posibilitando la instalación de *numpy* por ejemplo a través de un repositorio interno, acortando el tiempo de instalación. El problema presente en emplear esta aplicación es que no tiene forma de acceder a los recursos asociados al micrófono del dispositivo. Esto se representa mediante un error al cargar la librearía *sounddevice*, indicando que la librería *PortAudio* no se ha encontrado. Por tanto, a la vista de los resultados obtenidos, si bien es cierto que es posible ejecutar Python en dispositivos móviles, el alcance de este es más bien limitado. Se debe tener en cuenta que, para usar esta aplicación, debe obtenerse el permiso de usar el micrófono.

Por tanto, una de las posibles tareas a desarrollar en un futuro sería la investigación de cómo acceder a los recursos del sistema Android y en base a ello, conseguir una aplicación totalmente compatible con dispositivos móviles a partir de la aplicación cliente para ordenadores.

Considerando que los teléfonos móviles pueden actuar como nodos de la red de monitorización, otra opción a considerar podría ser la posibilidad de que la aplicación que estos empleen esté construida de forma nativa para ellos, es decir, mediante desarrollo en Java y emplear un entorno de desarrollo especializado para ello.

5. Planificación temporal.

El objetivo de este apartado es proporcionar una descripción de qué tareas se han realizado a lo largo de la planificación seguida para este proyecto.

El modelo escogido para la realización de este proyecto se basa en la metodología en cascada, con retroalimentación. Esto es, por ejemplo, la posibilidad de volver a una fase anterior si se detecta alguna incoherencia en el desarrollo o si por ejemplo los resultados de las pruebas no son aceptables.

La planificación está realizada considerando qué cada jornada consta de 8 horas de trabajo. El volumen total de horas obtenidas para la realización de este proyecto es de 544 horas. En la figura 5.1. se recoge la planificación temporal con las tareas globales, donde se indica el nombre de la tarea, su duración, las fechas de comienzo y fin, y la tarea predecesora (para una tarea cualquiera, indica la actividad o actividades que deben haber finalizado para que esta comience).

	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1	Estudios y análisis previos	8 días	lun 25/01/21	mié 03/02/2	
6	Análisis de requisitos y especificación	9 días	vie 05/02/21	mié 17/02/21	1
12	Diseño	7 días	jue 18/02/21	vie 26/02/21	6
16	Implementación	25 días	lun 01/03/21	vie 02/04/21	12
29	Verificación	5 días	lun 05/04/21	vie 09/04/21	16
35	Elaboración de la documentación	14 días	lun 12/04/21	jue 29/04/21	29

Figura 5.1.- Planificación temporal con las tareas globales.

5.1.- Estudios y análisis previos.

	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras	Nombres de los recursos
1	**Estudios y análisis previos**	**8 días**	**lun 25/01/21**	**mié 03/02/2**		
2	Determinar el objetivo del proyecto	2 días	lun 25/01/21	mar 26/01/21		Jefe del Proyecto
3	Determinar el alcance del proyecto	2 días	mié 27/01/21	jue 28/01/21	2	Jefe del Proyecto
4	Fase de estudio de soluciones similares	2 días	vie 29/01/21	lun 01/02/21	3	Ingeniero Software
5	Estudio de tecnologías	2 días	mar 02/02/2	mié 03/02/2	4	Ingeniero Software

Figura 5.2.- Planificación de la fase "Estudios y análisis previos".

5.2.- Análisis de requisitos y especificación.

	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras	Nombres de los recursos
6	**Análisis de requisitos y especificación**	**9 días**	**vie 05/02/21**	**mié 17/02/21**	**1**	
7	Definición de los requisitos de usuario	2 días	vie 05/02/21	lun 08/02/21	5	Ingeniero Software
8	Definición de requisitos del sistema	2 días	mar 09/02/21	mié 10/02/21	7	Ingeniero Software
9	Definición de los casos de uso	2 días	jue 11/02/21	vie 12/02/21	8	Ingeniero Software
10	Definición de las tecnologías a emplear	2 días	lun 15/02/21	mar 16/02/21	9	Ingeniero Software
11	Revisión por parte del cliente	1 día	mié 17/02/2	mié 17/02/2	10	Jefe del Proyecto

Figura 5.3.- Planificación de la fase "Análisis de requisitos y especificación".

5.3.- Diseño.

	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras	Nombres de los recursos
12	**Diseño**	**7 días**	**jue 18/02/21**	**vie 26/02/21**	**6**	
13	Diseño de la arquitectura del sistema	2 días	jue 18/02/21	vie 19/02/21	11	Ingeniero Software
14	Diseño de la aplicación cliente	2 días	lun 22/02/21	mar 23/02/2	13	Ingeniero Software
15	Diseño de la aplicación servidor	3 días	mié 24/02/2	vie 26/02/21	14	Ingeniero Software

Figura 5.4.- Planificación de la fase "Diseño".

5.4.- Implementación.

	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras	Nombres de los recursos
16	**Implementación**	**25 días**	**lun 01/03/21**	**vie 02/04/21**	**12**	
17	Construcción de la aplicación servidor	5 días	lun 01/03/21	vie 05/03/21	15	Programador
18	Construcción de la aplicación cliente	5 días	lun 08/03/21	vie 12/03/21	17	Programador
19	Pruebas unitarias de comunicación entre cliente y servidor	1 día	lun 15/03/21	lun 15/03/21	18	Ingeniero de Pruebas
20	Implementación del módulo de sonido en la aplicación cliente	2 días	mar 16/03/21	mié 17/03/21	19	Programador
21	Pruebas unitarias de sonido sobre el módulo de sonido en la aplicación cliente	1 día	jue 18/03/21	jue 18/03/21	20	Ingeniero de Pruebas
22	Construcción del módulo de escritura a fichero de la información de sesión	3 días	vie 19/03/21	mar 23/03/21	21	Programador
23	Pruebas unitarias sobre el módulo de escritura a fichero	1 día	mié 24/03/21	mié 24/03/21	22	Ingeniero de Pruebas
24	Construcción de los módulos de post procesamiento	3 días	jue 25/03/21	lun 29/03/21	23	Programador
25	Pruebas unitarias de los módulos de post procesamiento	1 día	mar 30/03/21	mar 30/03/21	24	Ingeniero de Pruebas
26	Rediseño de los menús en la aplicación cliente	1 día	mié 31/03/21	mié 31/03/21	25	Programador
27	Rediseño de los menú en la aplicación servidor	1 día	jue 01/04/21	jue 01/04/21	26	Programador
28	Pruebas unitarias interfaz gráfica	1 día	vie 02/04/21	vie 02/04/21	27	Ingeniero de Pruebas

Figura 5.5.- Planificación de la fase “Implementación”.

5.4.- Verificación.

	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras	Nombres de los recursos
29	**Verificación**	**5 días**	**lun 05/04/21**	**vie 09/04/21**	**16**	
30	Elaboración plan de pruebas	1 día	lun 05/04/21	lun 05/04/21	28	Ingeniero de Pruebas
31	Pruebas de integración sobre la aplicación cliente	1 día	mar 06/04/21	mar 06/04/21	30	Ingeniero de Pruebas
32	Pruebas de integración sobre la aplicación servidor	1 día	mié 07/04/21	mié 07/04/21	31	Ingeniero de Pruebas
33	Pruebas de sistema	1 día	jue 08/04/21	jue 08/04/21	32	Ingeniero de Pruebas
34	Pruebas de aceptación	1 día	vie 09/04/21	vie 09/04/21	33	Ingeniero de Pruebas; Jefe del Proyecto

Figura 5.6.- Planificación de la fase “Verificación”.

5.5.- Elaboración de la documentación.

	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras	Nombres de los recursos
35	**Elaboración de la documentación**	**14 días**	**lun 12/04/21**	**jue 29/04/21**	**29**	
36	Elaboración memoria	3 días	lun 12/04/21	mié 14/04/2:	34	Jefe del Proyecto
37	Elaboración pliego de prescripciones técnicas	2 días	jue 15/04/21	vie 16/04/21	36	Jefe del Proyecto
38	Elaboración documento de requisitos de usuario	2 días	lun 19/04/21	mar 20/04/21	37	Jefe del Proyecto
39	Elaboración documento de requisitos del sistema	2 días	mié 21/04/21	jue 22/04/21	38	Jefe del Proyecto
40	Elaboración documento de diseño del sistema	2 días	vie 23/04/21	lun 26/04/21	39	Jefe del Proyecto
41	Elaboración del documento de pruebas	1 día	mar 27/04/21	mar 27/04/21	40	Jefe del Proyecto
42	Elaboración manual de usuario	1 día	mié 28/04/2:	mié 28/04/2:	41	Jefe del Proyecto
43	Revisión final y corrección de fallos	1 día	jue 29/04/21	jue 29/04/21	42	Jefe del Proyecto

Figura 5.7.- Planificación de la fase “Elaboración de la documentación”.

5.6.- Tareas pendientes de realizar.

A continuación, se listan las tareas que quedan pendientes de su realización.

	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras	Nombres de los recursos
49	Compilar aplicación cliente en Android	5 días	lun 12/07/21	vie 16/07/21		Programador
50	Pruebas unitarias de la aplicación cliente en Android	1 día	lun 19/07/21	lun 19/07/21		Ingeniero de Pruebas
51	Crear aplicaciones ejecutables	7 días	mar 20/07/21	mié 28/07/21		Ingeniero Software; Programador
52	Mantenimiento de las aplicaciones	5 días	jue 29/07/21	mié 04/08/21		Ingeniero de Pruebas

Figura 5.8.- Tareas pendientes de realización.

5.7.- Diagrama Gantt del proyecto.

En la figura 5.9. se puede visualizar el diagrama Gantt correspondiente al proyecto, donde cada una de las tareas aparece representada con su identificador, y además también aparecen representadas las tareas globales.

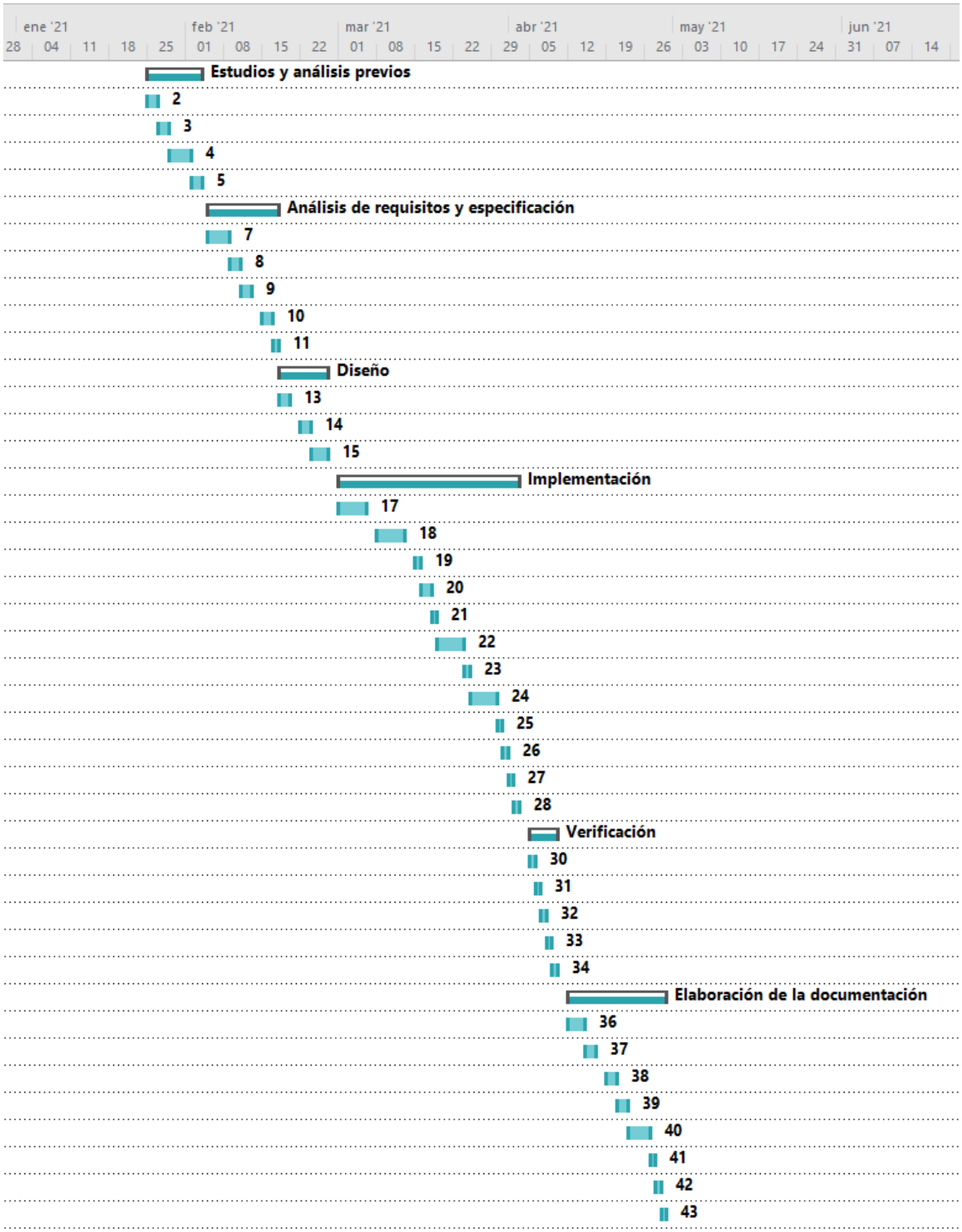

Figura 5.9.- Diagrama de Gantt del Proyecto

6. Conclusiones.

El desarrollo de este proyecto ha desembocado en un primer prototipo de una aplicación de monitorización que es totalmente funcional en un entorno de área local, como podría ser una oficina o una clase.

La simpleza de Python, acompañado de una gama amplia de librerías que permiten realizar múltiples tareas o manejar de forma más simple algunos aspectos, como es el acceso a periféricos o recursos del sistema, han permitido que el producto obtenido cumpla en mayor grado las expectativas planteadas al comienzo de este. También cabe mencionar, que el desarrollo del proyecto se efectuó de forma que se evitase en todo momento, librerías de terceros, excepto las de manejo de audio, de forma que la aplicación fuese compatible con todas las plataformas posibles o las de ayudan al procesamiento de la información.

Sin embargo, el ámbito final de este proyecto ha quedado limitado a que la ejecución de las aplicaciones solo puede efectuarse desde ordenadores, y no contar al menos con la opción de poder ejecutar la aplicación cliente desde un dispositivo móvil.

De cara a construir un sistema más completo que este primer prototipo, podrían proponerse otras ampliaciones como son la inclusión de una base de datos con la aplicación, de forma que no se requiera cargar un fichero que contenga todos los usuarios que disponen de acceso por cada sesión de monitorización. De esta forma, simplemente se necesitaría realizar una primera configuración, donde se creasen los usuarios, identificados por el correo electrónico. Esta construcción también habilitaría otra posibilidad, como es la eliminación de la contraseña para una sesión, sustituyéndose por una contraseña para cada usuario, autogenerada, o proporcionando a dicho usuario la posibilidad de asignar la que se desee.

Otra opción para considerar podría ser, en base a este prototipo, la construcción de una aplicación que funcionase desde cualquier lugar, es decir, que se apoyase en servicios web. No obstante, deberá considerarse la posibilidad de reconstruir la aplicación para poder dar

cabida a estos servicios. Además, sería necesaria la construcción de una API específica para ello.

Otra posible expansión podría ser la de crear aplicaciones que no fueran dependientes de tener instalado Python en el equipo ejecutor. De esta forma, si bien, habría que analizar la posibilidad de crear múltiples aplicaciones dependiendo del sistema operativo, se podría ofrecer una experiencia de uso superior a la actual.

Finalmente, otra ampliación sobre este proyecto sería investigar si es posible que la aplicación cliente obtenida, sea ejecutada correctamente en un dispositivo Android de forma que pueda ser totalmente funcional, eliminando la necesidad de que los usuarios dispongan de un equipo. En caso contrario, podría construirse una aplicación nativa sencilla que constase de las mismas opciones que la aplicación para ordenadores. Independientemente de la opción escogida, se debería apoyar sobre la API mencionada anteriormente, considerando que un teléfono móvil puede estar haciendo uso de la red de datos y por tanto no estar en la misma red que el servidor. Un ejemplo de una situación que podría apoyarse sobre la premisa mencionada anteriormente sería si se implementase el uso de una contraseña personalizada para cada usuario, sería necesario un servicio web para el acceso.

No obstante, a la vista de los resultados finales respecto a la capacidad de monitorización de la aplicación, y la capacidad a posteriori de la información obtenida mediante esta, se pueden considerar satisfactorios, y es posible que este proyecto pueda servir como base para poder desarrollar un sistema más eficiente y completo.

7. Documentación del proyecto.

El objetivo de este apartado es enumerar y describir brevemente, los documentos de los que se compone este proyecto. Este proyecto consta de los siguientes:

- Memoria: este documento describe un resumen de la realización del proyecto, desde los estudios previos al desarrollo, además de la planificación y sus conclusiones.
- Presupuesto: coste de realización del proyecto.
- Requisitos de usuario: describe las necesidades del sistema de forma que pueden ofrecer los servicios requeridos por el usuario.
- Análisis de requisitos del sistema: describe los aspectos de implementación que debe cumplir el sistema a construir.
- Diseño: describe los aspectos más relevantes a la hora del diseño del sistema.
- Pruebas: describe todas las pruebas realizadas sobre el producto obtenido.
- Manual de usuario: describe los pasos necesarios para poder configurar las aplicaciones para su uso, así como una guía de cómo funcionan las aplicaciones.
- Manual del programador: describe algunos aspectos relevantes de la programación de cara a un futuro mantenimiento o expansión de este.

8. Referencias.

[1] M. A. Pandharipande y S. K. Kopparapu. “Real time speaking rate monitoring system” presentado en *2011 IEEE Int. Conf. on Signal Processing, Communications and Computing (ICSPCC)*, Xi’an, China, sep. 2011, pp. 1-3. Doi: https://doi.org/10.1109/icspcc.2011.6061699 [junio de 2021]

[2] A. Imran, M. A. Pandharipande, S. K. Kopparapu. “SpeakRite: Monitoring Speaking Rate in Real Time on a Mobile Phone”, *Int. J. of Mob. Hum. Comp. Int.*, vol. 5, iss. 1, pp 62-69, 2013, doi: https://doi.org/10.4018/jmhci.2013010104 [junio de 2021]

[3] C. Lauri y J. Malmgren. “Synchronization of streamed audio between multiple playback devices over an unmanaged IP network”. Tesis de master, Dept. of Elect and Inf. Tech., Lund Univ., Lund, Suecia, 2015. [Online]. Disponible: https://www.eit.lth.se/sprapport.php?uid=894 [junio de 2021]

[4] INCIBE. “NTP, SNTP y PTP: ¿qué sincronización de tiempo necesito?”, INCIBE.es. https://www.incibe-cert.es/blog/ntp-sntp-y-ptp-sincronizacion-tiempo-necesito [junio de 2021]

[5] Fernando L. Romero. “Sincronización de Relojes en Ambientes Distribuidos”. Tesis de máster, Inst. de Inv. en Inf. LIDI (IIILIDI), Univ. Nacional de La Plata, La Plata, Argentina, 2009. [Online]. Disponible: http://sedici.unlp.edu.ar/bitstream/handle/10915/4157/Documento_completo.pdf?sequence=1&isAllowed=y [junio de 2021]

[6] D. M. Havey, E. Ross, P. Short y M. Blodgett. “Support boundary for high-accuracy time”. Docs.microsoft.com. Disponible: https://docs.microsoft.com/en-us/windows-server/networking/windows-time-service/support-boundary [junio 2021]

[7] NIST. “NIST Authenticated NTP Service”. NIST.gov. Disponible: https://www.nist.gov/pml/time-and-frequency-division/time-services/nist-authenticated-ntp-service [junio 2021]

[8] University of Rhode Island and Inner Space Center. *Discovery of Sound in the Sea*. USA. [Online]. Disponible: https://dosits.org/science/advanced-topics/introduction-to-signal-levels/ [junio de 2021]

[9] National Research Council. “Appendix A. Comparison of Sound-Pressure Reference Levels in Air and Water” en *Low-Frequency Sound and Marine Mammals: current Knowledge and Research Needs*, National Research Council, Washington (DC), USA, 1994. [Online]. Disponible: https://www.ncbi.nlm.nih.gov/books/NBK236684/ [junio de 2021]

9. Bibliografía.

Python Software Foundation. Python 3.8.11 documentation, version 3.8.11. Disponible: https://docs.python.org/3.8/

Matthias Geier, spatialaudio, Play and Record Sound with Python. Disponible: https://python-sounddevice.readthedocs.io/en/0.4.1/#