UNIVERSIDAD NACIONAL DE SAN LUIS

FACULTAD DE CIENCIAS FISICO MATEMATICAS Y NATURALES

INGENIERÍA EN COMPUTACIÓN

PROYECTO FINAL

“KeyAid: Una plataforma para el mantenimiento remoto de dispositivos”

Merenda, Franco Nicolas

Mg. Abdelahad, Corina Natalia

Directora

Dr. Piccoli, María Fabiana

Co-Directora

San Luis - Argentina

-2022-

Índice

Agradecimientos **7**

Capítulo I - "Problemática actual y objetivos" **8**

1.1. Problemática Actual 9

1.2. Objetivos 10

1.3. Organización del Informe 11

Capítulo II - "Herramientas y Tecnologías" **12**

2.1 Herramientas de Software 13

2.1.1 Flutter 13

2.1.2 Dart 15

2.1.3 TinyPilot 16

2.1.4 uStreamer 16

2.1.5 Flask, Flask-SocketIO 18

2.1.6 Ventoy 18

2.2 Herramientas de Hardware 19

2.2.1 Raspberry Pi 20

2.3 Emulación de Dispositivos 21

2.3.1 Especificación USB On-The-Go (OTG) 21

2.3.2 Linux USB-Gadget API 23

2.3.3 Estructura de los Controladores Gadget 24

2.3.4 USB Gadget - ConfigFS 27
2.3.5 ConfigFS - Funcionamiento 28

Capítulo III - "KeyAid: Diseño e Implementación" 31
3.1 KeyAid: Características Generales 32
3.2 Plataforma Web - KeyAid 33
3.2.1 Módulo Front-End 34
3.2.2 Módulo Back-end 35
3.3 Módulo Acceso Remoto 36
3.4 Módulo Emular dispositivos USB 37
3.5 Módulo Conexión 39
3.5.1 Eventos de teclado/mouse 39
3.5.2 Captura de señal vídeo de la máquina del cliente 41
3.5.3 Consideraciones de seguridad de la conexión entre técnico-cliente 42

Capítulo IV - "Perfiles y Usos de KeyAid" 44
4.1 Idea básica del funcionamiento 45
4.2 Ejemplo del funcionamiento 46
4.2.1 Escenario - Primera vez en la Plataforma 46
4.2.2 Escenario de usuario cliente - Crear un trabajo 46
4.2.3 Escenario de usuario técnico - Postular a un trabajo 52

Capítulo V - "Evaluación Experimental" 56
5.1 Características Generales de la Experimentación 57
5.2 Escenario 1 - PC de escritorio 58
5.3 Escenario 2 - Netbook 60
5.4 Escenario 3 - MacBook (Mod. 2019) 61

Capítulo VI - "Conclusiones y Trabajos Futuros" 63
6.1. Conclusiones 64

6.2. Trabajos Futuros 66

Bibliografía **69**

Anexo I: Modelos de KeyAid **73**

Anexo II: Manual del Usuario **76**

Índice de Figuras

Capítulo II - "Herramientas y Tecnologías"

2.1 Arquitectura en capas de Flutter 14
2.2 Comunicación entre dispositivos de multimedia 17
2.3 Ejemplo de menú provisto por Ventoy 19
2.4 Ejemplo de sistema de archivos 28
2.5 Estructura de datos de ejemplo representando el sistema de archivos 29

Capítulo III - "KeyAid: Diseño e Implementación"

3.1 Modelo de componentes de "KeyAid" 32
3.2 Configuración del dispositivo emulado con ConfigFS 38
3.3 Configuración de lugar de almacenamiento para el dispositivo USB emulado 38
3.4 Conexión de Raspberry Pi 4b utilizando el "Splitter" 40
3.5 Ejemplo del esquema de aislamiento eléctrico de datos/corriente 41
3.6 Capturadora de Vídeo 1080p/30fps - HDMI a USB 2.0 41

Capítulo IV - "Perfiles y Usos de KeyAid"

4.1 Formulario de registro para la plataforma *KeyAid* 46
4.2 Pantalla de selección el tipo de trabajo a crear por el *cliente* 47
4.3 Pantalla de creación de trabajo para usuarios avanzados 47
4.4 Pantalla de creación de trabajo para usuarios novatos 48
4.5 Ejemplo de un trabajo simple y un trabajo avanzado 48
4.6 Ejemplo de trabajo de cliente con 3 postulaciones para el mismo 49
4.7 Ejemplo de vista de cliente para confirmar postulación para el trabajo 49

4.8 Trabajo con postulación confirmada 50
4.9 Pantalla de trabajo que fue confirmada por el cliente 50
4.10 Pantalla de trabajo donde ya las tareas fueron realizadas por el técnico 51
4.11 Pantalla de realización de pago por parte del cliente 51
4.12 Pantalla de reseña para el técnico, luego de haber realizado el pago 52
4.13 Ejemplo de pantalla de inicio de un usuario técnico 52
4.14 Ejemplo de postulación realizada por el técnico a un trabajo 53
4.15 Postulación confirmada por el cliente con acceso remoto para comenzar a trabajar 53
4.16 Pantalla mostrando la BIOS/UEFI del usuario cliente 54
4.17 Pantalla mostrando el “Escritorio” del sistema operativo Windows 10 del cliente 54
4.18 Pantalla con el botón para marcarle al usuario cliente que el trabajo ha sido finalizado 55

Capítulo V - “Evaluación Experimental”

5.1 Pantalla mostrando la BIOS/UEFI del usuario cliente. 59
5.2 Pantalla mostrando el “Escritorio” del sistema operativo Windows 10 del cliente. 59

Anexo I

A1.1 Modelo de Dominio - *KeyAid* (Usuarios/Trabajos/Postulaciones) 74
A1.2. Modelo de Actividad - Proceso - Solicitar un trabajo 75

Anexo II

A2.1 - Página de Registro 78
A2.2 - Pantalla de Inicio - Botón para crear trabajo 79
A2.3 - Tipos de trabajo a crear - Soy Novato/a (Simple) o Tengo Conocimientos (Avanzado) 80
A2.4 - Opción “Soy Novato/a (Simple)” - Paso 1 80
A2.5 - Opción “Soy Novato/a (Simple)” - Paso 2 81
A2.6 - Opción “Tengo conocimientos (Avanzado)” - Paso 1 82
A2.7 - Opción “Tengo conocimientos (Avanzado)” - Paso 2 82
A2.8 - Opción “Tengo conocimientos (Avanzado)” - Paso 3 83

A2.9 - Opción “Tengo conocimientos (Avanzado)” - Paso 4 83
A2.10 - Botón para crear trabajo en la plataforma 84
A2.11 - Trabajo con una postulación recibida 85
A2.12 - Postulación realizada por un técnico especificando costos y tiempos. 85
A2.13 - Pantalla de trabajo - Botón para realizar el pago 86
A2.14 - Pagar por el trabajo - Formulario de tarjeta de crédito/débito 87
A2.15 - Último paso (Opcional) - Dejar una breve reseña y calificación para el técnico 87
A2.16 - Pantalla de Inicio con trabajos - Técnico 88
A2.17 - Pantalla de postularse a un trabajo - Técnico 88
A2.18 - Pantalla de postulación - Indica la espera para postulación del *cliente* 89
A2.19 - Postulación confirmada - Botón para el acceso remoto a la PC/Servidor del cliente 90
A2.20 - Trabajo con todas las tareas finalizadas 90

Agradecimientos

Primeramente quisiera decir que este trabajo no hubiera sido posible sin el constante apoyo de mi familia, Antonino, Marisa y Luciano, quienes para poder realizar los estudios a distancia, siempre nos han apoyado en todos los proyectos que hemos empezado. Les agradezco por haber estado siempre, por su paciencia, su esfuerzo constante y el creer en nosotros. A ellos el mayor agradecimiento de todos.

A la vez mi mayor agradecimiento a Megan, quien ha sido la persona que ha estado todos los días al lado impulsándome a concretarlo, ayudándome desde la simpleza de prepararme un café hasta sus palabras de aliento. A vos mi mayor agradecimiento.

Gracias a poder haber vivido estos años de estudio, me pude rodear de personas con las mismas ganas y energías de crecer, construir y seguir estudiando. Entre ellos me toca destacar a mi grupo de amigos/as de cursada, con los que pudimos realizar proyectos juntos, tomar mates para hacer frente a las largas horas de cursada, por esas mañanas y tardes de mates y tortitas.

Seguido al párrafo anterior, también me gustaría destacar a la primera persona con la que tuve contacto desde la F.C.F.M.y.N, la cual desde el primer día, creyó en mí y siempre se interesó en todos los proyectos en los que estuve involucrado. Muchísimas gracias por ese impulso inicial *Dra. Nora Reyes* y las enseñanzas recibidas.

Agradecer también a la universidad, porque me dio la posibilidad de poder viajar a Brasil por una beca de intercambio, que si bien no pudo finalizar en término, respecto a fines académicos, me dio la oportunidad de poder haber vivido otra cultura y llevarme una experiencia inolvidable gracias a las personas con las que tuve la oportunidad de compartir.

Finalmente, quiero expresar mi gran agradecimiento a *Mg. Abdelahad, Corina Natalia*, por haberme ayudado a poder definir metas, límites y plazos realistas para el proyecto. Asimismo, quisiera agradecer a *Mg. Corina Natalia Abdelahad* y la Dra. *Fabiana Piccoli*, por las constantes correcciones y seguimiento del proyecto. Sin la ayuda de ambas, el proyecto no hubiese sido posible presentarlo en tiempo y forma.

Capítulo I

Problemática Actual y Objetivos

Hoy en día, realizar tareas como, la reinstalación de un sistema operativo, configuración de BIOS/UEFI, entre otras, no es posible mediante el acceso remoto a dispositivos del tipo PCs o Servidores. Si bien existen herramientas para control remoto de las computadoras y/o servidores, todas necesitan de un sistema operativo *host* o están orientados a usuarios con mayor conocimiento en el dominio de la informática. Por ello, en este trabajo se propone desarrollar una herramienta que permita realizar el mantenimiento a nuestros dispositivos de manera remota. La misma se encargará de la gestión de los trabajos, contemplando planificación de las tareas a realizar, tiempos estimados de trabajo, un portal para el acceso remoto y además, gestión de pago por el trabajo realizado.

En este capítulo se detallan la motivación del trabajo propuesto, su objetivo y la organización del presente informe.

1.1. Problemática Actual

En la actualidad existen diferentes soluciones para realizar determinadas tareas remotas, tales como, arreglar un programa, asistir a una persona con alguna dificultad en algún programa, instalar diferentes tipos de programas, realizar configuraciones en el sistema operativo, realizar transferencia de archivos, entre otras. Algunas de las soluciones existentes son:

- TeamViewer [1].
- AnyDesk [2].
- PiKVM [3].
- TinyPilotKVM [4].

Todas ellas pueden considerarse parciales, es decir no tienen en cuenta todos los aspectos implicados o derivados de las necesidades que surgen del usuario a la hora de requerir un mantenimiento/arreglo a su sistema. Por ejemplo, por un lado, *TeamViewer* y *AnyDesk* son soluciones que requieren de un sistema operativo *host* para lograr el acceso remoto, lo que implica que dichos programas no se encuentran ejecutando en el sistema durante el encendido o apagado del mismo. Tareas como la reinstalación de un sistema operativo, configuraciones en la BIOS/UEFI, entre otras, no son posibles mediante estas dos herramientas. En el caso de *PiKVM* y *TinyPilotKVM* están orientadas a un usuario con experiencia y/o conocimientos previos en el área, no son accesibles a usuarios con poco o sin conocimiento disciplinar.

Todas las soluciones antes mencionadas son parciales, no contemplan una plataforma donde unifique, y a su vez, simplifique el acceso a este tipo de tecnología para el usuario final. Actualmente existen plataformas de trabajo remoto como *Workana* [5], *UpWork* [6] y *Fiverr* [7], las cuales son plataformas web donde los usuarios disponen posibles trabajos a realizar y usuarios, dentro de la misma plataforma, pueden ofrecerse a realizarlo. Los tipos de trabajo generalmente están referidos al ámbito de diseño, programación, edición de videos, traducciones, entre otros.

Por todo lo mencionado, es posible determinar la necesidad de contar con una herramienta integral, la cual permita realizar trabajos de mantenimiento remoto a través de una plataforma

web simple, tanto para utilizar como acceder por parte de los *técnicos* y los *usuarios finales* o *clientes*.

1.2. Objetivos

Cada vez más, nuevas metodologías de trabajo son posibles debido a la mejora en infraestructura de conectividad y acceso a la tecnología. Por ello, muchas de las tareas antes realizadas de forma presencial, hoy pueden desarrollarse igual o mejor de manera remota.

El objetivo de este proyecto es el desarrollo de la plataforma web llamada ***KeyAid,*** la cual nos propone una nueva manera de realizar mantenimiento a nuestros dispositivos y en forma remota.

KeyAid es una plataforma web que vincula técnicos en Tecnología de Información (IT) con personas que necesiten asistencia con su dispositivo, ya sea, por requerir actualizaciones o por alguna falla técnica. Ésta consta de dos partes principales:

- Plataforma Web: Donde *técnicos* y *clientes* interactúan para poder describir el trabajo, realizarlo y que el *usuario final* pueda pagar y evaluar el trabajo realizado.
- Dispositivo de control (*Raspberry Pi*): Dispositivo utilizado para poder controlar la computadora destino del *usuario final*. El envío de los comandos al dispositivo destino se logra utilizando el software TinyPilot [4].

TinyPilot [4] es un software Open Source desarrollado por Michael Lynch, el cual es instalado en el Sistema Operativo Raspbian 32 bits - basado en la distribución Debian de GNU/Linux.

Sin importar la ubicación geográfica del *usuario/cliente*, mediante *KeyAid*, éste tendrá la posibilidad de realizar mantenimiento a sus dispositivos desde la comodidad de su casa, eligiendo el *técnico* que considere adecuado, coordinando tiempos de trabajo y costos.

1.3. Organización del Informe

El presente informe consta de seis capítulos, donde el Capítulo I consta de la problemática actual y cuál es la propuesta para la solución de la misma.

En el Capítulo II se detallan las múltiples tecnologías utilizadas para alcanzar el objetivo de este trabajo, el cual implica el desarrollo de la plataforma web para el mantenimiento remoto de equipamiento, el dispositivo para el acceso y control a los mismos, y su configuración para el desarrollo de las tareas de comunicación y mantenimiento.

El Capítulo III describe una herramienta integral desarrollada a fin de proveer servicios de mantenimiento remoto a equipamiento informático. Para ello se propone, mediante el uso de un dispositivo externo que actúa como un controlador de la PC del usuario, *cliente, y* una plataforma web de interacción entre éste (quien necesita el arreglo o mantenimiento) y el *técnico* (responsable de realizar el trabajo), gestionar los tiempos para realizar el trabajo, las tareas, costos y pagos.

En el Capítulo IV se explican los diferentes perfiles: *técnico* y *cliente*, y el uso de *KeyAid* con cada uno de ellos. La descripción se centra en este caso en la plataforma web, mostrando las diferentes interfaces gráficas desde el punto de vista del *cliente* como del punto de vista del *técnico*.

En el Capítulo V se describe la evaluación experimental en distintos escenarios. Los escenarios se caracterizan por tener diferentes particularidades tanto de hardware: Computadora Personal, Netbook y Notebook, como de software, distintos sistemas operativos: Windows, Linux y macOS. A través de los diferentes escenarios de experimentación, se muestran distintas funcionalidades de *KeyAid*, principalmente de la interacción *usuario/cliente* con el *técnico*.

Finalmente, en el Capítulo VI se detallan las conclusiones y las posibles líneas de trabajo a seguir a fin de ampliar y/o mejorar las funcionalidades y prestaciones de *KeyAid*.

Capítulo II

Herramientas y Tecnologías

Años atrás, cuando se quería realizar mantenimiento o corregir errores en las computadoras, era necesario llevar la máquina al técnico y por ende quedarse sin ella por varios días. En la actualidad, existen aplicaciones que permiten resolver problemas de software conectando remotamente al *usuario final* con el *técnico*. Sin embargo, estas requieren de un sistema operativo host para poder lograrlo.

Emular distintos periféricos USB tales como teclados (entradas de datos), mouse (entradas de datos), almacenamiento USB, entre otros, es posible gracias a una especificación llamada USB On-The-Go. El hecho de poder realizar diversas funcionalidades desde un mismo dispositivo y una única conexión USB, permite a un dispositivo como la Raspberry Pi poder enviar eventos de teclado/mouse o inclusive comportarse como una unidad flash USB de manera remota. Es decir, se hace innecesario el requerimiento de un sistema operativo con un software instalado para recibir estos eventos de manera remota; la Raspberry Pi recibe los eventos y los envía a través del puerto USB a la computadora/servidor del *usuario final*.

En este capítulo se detallan las múltiples tecnologías utilizadas para lograr el objetivo de este trabajo, el cual consta del desarrollo de la plataforma web para el mantenimiento remoto de dispositivos llamado *KeyAid*.

2.1 Herramientas de Software

Para el desarrollo de la plataforma web ***KeyAid*** fue necesario la utilización de diferentes tecnologías relacionadas al software y al hardware. En la actual sección se procede a explicar las tecnologías utilizadas en el proyecto relacionadas al Software: *Flutter* [9][10][11], *Dart* [12], *TinyPilot* [4], *uStreamer* [13][14], *Flask* [15][16], *Flask-SocketIO* [17][18], y *Ventoy* [19].

2.1.1 Flutter

Flutter [9][10][11] es un proyecto de código abierto de Google para crear aplicaciones multiplataforma compiladas de forma nativa a partir de una única base de código. Hoy en día, permite compilar de manera nativa para varias plataformas, entre ellas, Android, iOS, Linux, macOS, Windows, Google Fuchsia y además, para la web, utilizando el mismo código escrito en el lenguaje *Dart*.
Flutter está diseñado como un sistema extensible en capas (ver Figura 2.1). Existe un conjunto de bibliotecas independientes, cada una de las cuales depende de la capa subyacente. Cada capa del framework se ha diseñado para ser opcional y reemplazable, y donde ninguna capa puede acceder de manera privilegiada, es decir haga uso y/o tenga conocimiento de la capa inferior o superior inmediata.

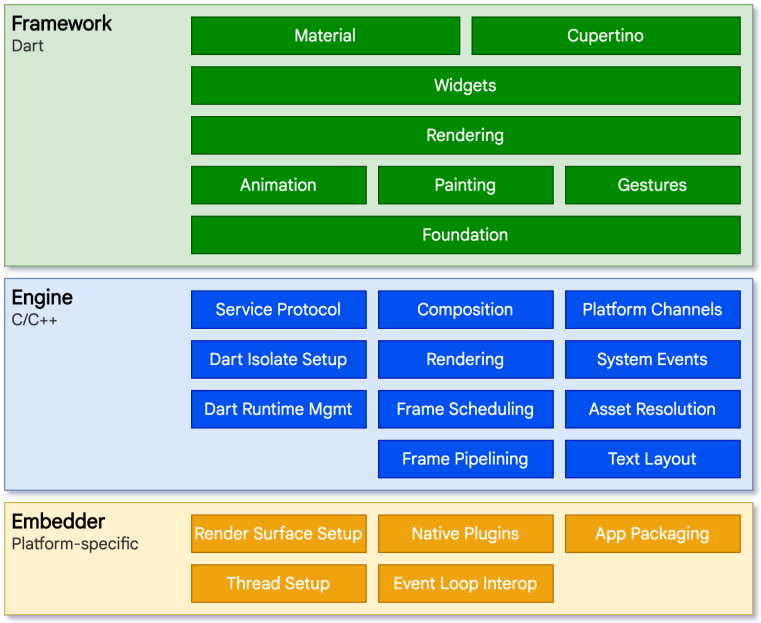

Figura 2.1. Arquitectura en capas de Flutter[20].

La forma en que las aplicaciones realizadas con Flutter son empaquetadas, es igual a cualquier otra aplicación nativa. Para cada plataforma, existe un empaquetador específico que proporciona un punto de entrada, coordina con el sistema operativo para acceder a servicios como el renderizado, accesibilidad, entrada de datos; y gestiona el bucle de eventos. Este empaquetador está escrito en un lenguaje apropiado para cada plataforma, entre ellas actualmente se encuentran:

- Android: Java and C++
- iOS/macOS: Objective-C/Objective-C++
- Linux/Windows: C++

El motor de Flutter, escrito principalmente en C++, contiene todo el código necesario para poder interactuar con cualquier aplicación desarrollada con el mismo. Este motor es el responsable principalmente de renderizar las escenas (conjunto de widgets), entre otras tareas. Además, proporciona la implementación de bajo nivel de la API de Flutter, la cual incluye gráficos (a través de Skia [21][22], librería de código abierto para gráficos 2D), disposición de los textos, Entrada/Salida (E/S) de archivos y redes, soporte de accesibilidad, arquitectura de complementos (plugins) y un conjunto de herramientas *Dart* en tiempo de ejecución y compilación.

El programador, generalmente interactúa con la primera capa: Flutter Framework, quién provee un framework moderno y reactivo escrito enteramente en *Dart*.

2.1.2 Dart

Dart[12] es un lenguaje optimizado para el lado del "cliente", permite desarrollar aplicaciones de manera rápida para cualquier plataforma ofreciendo un lenguaje más productivo que los actuales, así como Javascript lo es para la web, Java lo es para Android o Swift lo es para iOS, Dart lo es para las aplicaciones. Este lenguaje permite el desarrollo multiplataforma junto a una plataforma de tiempo de ejecución flexible para los frameworks como *Flutter*.

Prioriza tanto la productividad en código, con herramientas como *hot-reload*, y experiencias de usuario de alta calidad, a nivel de producción en distintas plataformas como (web, móvil y escritorio).

La tecnología de compilación de *Dart* le permite ejecutar código de diferentes maneras, ellas son:

- Plataforma nativa: para aplicaciones dirigidas a dispositivos móviles y de escritorio, Dart incluye una Máquina virtual (VM), la cual provee un ambiente de ejecución para el lenguaje Dart, con compilación justo a tiempo (JIT) y un compilador adelantado (AOT) para producir código de máquina.
- Plataforma web: Destinada a aplicaciones en la web. En este caso *Dart* puede compilar con fines de desarrollo o producción. El compilador web traduce *Dart* a *JavaScript*.

Gracias al compilador JIT, permite recompilar de manera incremental. En *Dart* y en *Flutter*, a esta característica se la conoce como *hot-reload*. Esto permite ir visualizando los cambios que se van realizando en el código de manera instantánea, disminuyendo tiempos de desarrollo.

Dart es parte de *Flutter*, proporcionando el lenguaje y los tiempos de ejecución que impulsan sus aplicaciones. Dart también es compatible con muchas tareas básicas del desarrollador, como formatear, analizar y probar código.

Hace chequeo estático de tipos para asegurar que el valor de la variable siempre sea de tipo estático. Si bien los tipos son obligatorios, declararlos es opcional ya que Dart soporta inferencia de tipos. Además, ofrece *seguridad de nulos* (*sound null safety)* [23], lo que asegura que los valores no pueden ser *null* a menos que explícitamente se indique que puede serlo. Esta propiedad

permite detectar excepciones por valores *null* en el momento de análisis estático del código antes del tiempo de ejecución.

2.1.3 TinyPilot

Tinypilot [4] es un proyecto de código abierto que proporciona acceso remoto a través del navegador web al ser instalado en una *Raspberry Pi.* Éste la convierte en una pieza de hardware con acceso a una PC en forma remota de manera completa. Esto significa que el acceso al teclado y al vídeo funciona incluso antes de que la máquina host se inicie, por lo cual incluso cambiar algo como la configuración del BIOS es posible.
Éste logra emular distintos periféricos USB tales como, teclados, mouse, almacenamiento USB, todo gracias a una especificación llamada USB On-The-Go, la cual es soportada por el procesador (Broadcom BCM2711) que lleva la Raspberry Pi Mod 4b. Éste va instalado sobre el sistema operativo GNU/Linux Raspbian que está instalado en la Raspberry Pi. El kernel del sistema operativo Raspbian, Linux, expone funcionalidades para emular periféricos USB a través de una funcionalidad llamada ConfigFS, que será explicada al final de este capítulo.
Finalmente, el envío y la captura de la señal de vídeo del dispositivo del usuario final, es posible gracias a un proyecto de código abierto llamado *uStreamer*, junto al uso de una capturadora de vídeo, la cual recibe el vídeo y se lo entrega a la Raspberry Pi a través de uno de sus puertos USB.

2.1.4 uStreamer

µStreamer [13][14] es un servidor rápido para transmitir vídeo en formato MJPEG (Motion JPEG) desde cualquier dispositivo que sea soportado por V4L2 [24] (Vídeo para Linux), a la red. Todos los navegadores modernos tienen compatibilidad nativa con este formato de vídeo, así como con la mayoría de los reproductores de vídeo, como *mplayer*, *VLC*, entre otros.
uStreamer hace que el stream del vídeo se realice de una manera eficiente y simple a través del framework Flask utilizado por *TinyPilot*. La latencia del vídeo se encuentra entre unos 200-300ms lo cual es suficientemente rápido para lo requerido por la plataforma a desarrollar.

La API, Vídeo para Linux (V4L2), es un framework que simplifica el desarrollo de drivers para dispositivos de vídeo. Éstos pueden llegar a ser muy difíciles de comprender por la complejidad en el hardware, por lo que se vuelve un requisito lograr una abstracción para este tipo de dispositivos multimedia. La mayoría de éstos contienen múltiples circuitos integrados y disponen de múltiples nodos creados en */dev* (Directorio donde se encuentran los dispositivos reconocidos por el kernel de Linux).

El hecho de que los drivers de V4L2 tengan que dar soporte a múltiples circuitos integrados (para realizar la decodificación/codificación/multiplexación de audio/vídeo), los hace más complejos respecto a otro tipo de drivers. Usualmente, estos circuitos se encuentran conectados a través de uno o más buses I2C, u otro tipo de bus (ver Figura 2.2). Estos dispositivos son conocidos como sub-dispositivos. Por lo tanto, este framework establece los componentes básicos que necesitan todos los controladores y tiene el fin de facilitar la refactorización del código común en funciones de utilidad compartidas por todos los controladores. Además, éste permite que distintos dispositivos capturadores de vídeo sean reconocidos por el kernel de Linux y entreguen el vídeo en un formato que es reconocido por múltiples navegadores, como es el caso de MJPEG. No se limita a este formato únicamente, sino que provee todas las herramientas necesarias para permitir este comportamiento y hacer uso de él de una manera simple.

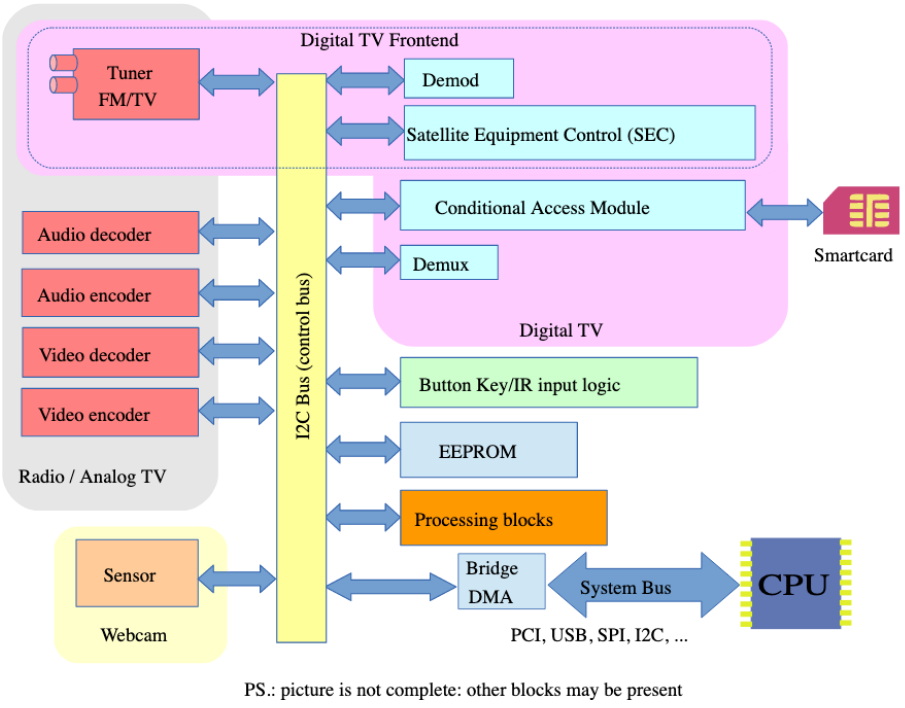

Figura 2.2. Comunicación entre dispositivos de multimedia [25].

2.1.5 Flask, Flask-SocketIO

Flask [15][16] es un framework para el desarrollo de servidores web. Éste está compuesto por *Werkzeug* [26] y *Jinja* [27], los cuales se encargan de manejar la lógica para las solicitudes http/https, y las plantillas web respectivamente.
Flask ofrece sugerencias, pero no aplica ninguna dependencia o diseño de proyecto. Depende del desarrollador elegir las herramientas y bibliotecas que desea utilizar. Además, existen muchas extensiones proporcionadas por la comunidad que facilitan la adición de nuevas funciones.
Flask es utilizado por *TinyPilot* para poder proveer diferentes endpoints para, el streaming del vídeo del dispositivo del cliente, el envío de eventos de teclado y mouse, y a su vez, brindar la interfaz web para interactuar con la máquina destino.

2.1.6 Ventoy

Ventoy [19] es una herramienta de código abierto para crear una unidad USB de arranque con archivos ISO/WIM/IMG/VHD(x)/EFI. No es necesario formatear el disco una y otra vez, sólo se necesita copiar los archivos ISO/WIM/IMG/VHD(x)/EFI a la unidad USB y arrancarlos directamente.
Se pueden copiar uno o más archivos y luego *Ventoy* provee un menú de inicio para seleccionar uno de todos ellos al momento de inicio (Ver Figura 3). Este es compatible con la mayoría de los tipos de sistemas operativos (Windows/WinPE/Linux/ChromeOS/Unix/VMware/Xen...)

Figura 2.3. Ejemplo de menú provisto por Ventoy.

Debido a la posibilidad de emular un dispositivo de almacenamiento USB a través de ConfigFS, se hace posible utilizar Ventoy sobre ese dispositivo emulado. Esto permite que agregar/eliminar sistemas operativos desde el lado del técnico se pueda realizar de una manera simple. Con el hecho de descargar un archivo ISO del sistema operativo deseado y cargarlo en dicho almacenamiento, es suficiente para poder hacer uso de él y tener un USB de arranque emulado por la Raspberry Pi.

2.2 Herramientas de Hardware

Para lograr la conexión de forma remota entre el *técnico* y el *usuario final* se requiere de un dispositivo que actúe como controlador de la máquina del *usuario final*, el cual es controlado, valga la redundancia, por el *técnico.* A continuación, se procede a explicar las tecnologías relacionadas al hardware utilizadas en el proyecto, *Raspberry Pi*.

2.2.1 Raspberry Pi

La Raspberry Pi es una computadora de bajo costo que tiene el tamaño de una tarjeta de crédito, se conecta a un monitor de computadora o televisor, y utiliza un teclado y un mouse estándar. Es un pequeño dispositivo que permite a personas de todas las edades explorar la informática y aprender a programar en lenguajes como Scratch [28] y Python [29].

Dentro de sus capacidades incluye navegar por Internet, reproducir vídeos de alta definición, trabajar con hojas de cálculo, procesamiento de textos y juegos, entre otras. Una de las grandes ventajas es la capacidad de interactuar con el mundo exterior a través de sus pines de entrada/salida, permitiendo trabajar en diferentes tipos de proyectos, desde estaciones meteorológicas, máquinas de música, armado de clusters con múltiples Raspberry Pi e inclusive sismógrafos [30].

Para el caso del desarrollo propuesto en este trabajo, se ha hecho uso del modelo 4B, el cual cuenta con las siguientes especificaciones:

- Broadcom BCM2711, Quad Core - Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz,
- 4GB LPDDR4-3200 SDRAM,
- 2.4 GHz y 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE,
- Gigabit Ethernet,
- 2 Puertos USB 3.0; 2 Puertos USB 2.0,
- 40 pin GPIO header,
- 2 × micro-HDMI ports (up to 4kp60 supported),
- 2-lane MIPI DSI display port,
- 2-lane MIPI CSI camera port,
- 4-pole stereo audio and composite vídeo port,
- H.265 (4kp60 para decodificar), H264 (1080p60 para decodificar, 1080p30 para codificar),
- Soporte para OpenGL ES 3.1, Vulkan 1.0,
- Slot para tarjeta Micro-SD para carga del sistema operativo y datos,
- 5V DC via conector USB-C (mínimo 3A*),
- 5V DC via pin GPIO (mínimo 3A*).

El uso de una Raspberry Pi se debe a su capacidad de extender funcionalidades a través de sus pines de entrada/salida, así como, el poder de cómputo que permite realizar streaming de vídeo y el envío de los datos suficientes para lograr el control remoto de otro dispositivo [31].
Respecto al software, en la Raspberry Pi, se instala el sistema Operativo Raspbian 32 bits, en el cual va instalado y configurado el proyecto *TinyPilot* para proveer el manejo remoto de los dispositivos, necesario para este proyecto.
El procesador de la Raspberry Pi cuenta con soporte para la especificación USB On-The-Go, explicado a continuación, la cual está expuesta a través del sistema operativo Raspbian, el cual cuenta con un kernel Linux dónde está habilitado el soporte para la misma. Si bien para el usuario final es transparente, es gracias a esta especificación que es posible enviar eventos de teclado y mouse a la PC que se controla de manera remota.

* Una fuente de alimentación de 2.5 A puede ser utilizada si los periféricos USB consumen menos de 500mA en total.

2.3 Emulación de Dispositivos

En la presente sección se explica en detalle las tecnologías que permiten la posibilidad de emular dispositivos como el teclado, mouse o un dispositivo de almacenamiento masivo, desde la Raspberry Pi, de manera remota. Las tecnologías utilizadas por el proyecto son: *Especificación USB On The Go (OTG)* [32], *Linux USB-Gadget API* [34], *Estructura de los Controladores Gadget* [35], *ConfigFS* [33].

2.3.1 Especificación USB On-The-Go (OTG)

Por definición, la comunicación USB ocurre entre un *host* y un *periférico.* La intención de la comunicación USB es colocar la carga de trabajo más pesada en la PC (host) y permitir que los periféricos USB sean más simples. [32]
En consecuencia, la especificación USB requiere que las PCs tengan la capacidad de:

- Proveer energía a los periféricos.

- Ser compatible con todas las velocidades definidas en la especificación.
- Ser compatible con todos los tipos de flujo de datos definidos en la especificación USB (*control*, *bulk*, *interrupt* y *isochronous*).

A medida que ha pasado el tiempo, los recursos de computación han ido disminuyendo su costo, volviéndose bastante económicos, por lo que la línea divisoria entre PCs y otros productos ya no es tan clara. Dichos productos contienen la capacidad de cómputo suficiente para manejar la función *host* de USB, funcionando de manera distinta a la forma convencional que lo hacen las PCs.

A pesar de que ellos proveen capacidad para trabajar como *hosts*, no hay razón para requerir que soporten todos los tipos de periféricos USB. Por ejemplo, el hecho de que a una impresora se le pueda conectar una cámara, es intuitivo pensar que obedece al hecho de querer imprimir fotos, no habría ninguna razón para que una impresora diera soporte a un dispositivo USB de GPS, ya que el GPS no se relaciona en ningún punto con las funciones de la impresora.

Targeted Hosts, es una especificación para definir a éstos dispositivos no-PCs. Un *targeted host* es un USB *host* con la capacidad de una serie limitada de periféricos. El desarrollador de cada *Targeted host* define qué conjunto de periféricos va soportar especificandolos en una lista llamada Targeted Peripheral List (TPL). Un *targeted host* necesita proveer la corriente, la velocidad de bus, el tipo de flujo de datos, etc, a los periféricos que se encuentren en la lista TPL.

Hay dos tipos de *Targeted Hosts* [32], estos son*:*

- *Hosts embebidos (EH)*: Un host embebido es un producto con la funcionalidad de *Targeted Host* a través de uno o más conectores Standard-A o micro-AB.
- *On-The-Go (OTG)*: Es un dispositivo portable que usa un único conector Micro-AB para operar algunas veces como *USB Targeted Host* y otras veces como un *Periférico USB*. Los dispositivos OTG siempre deben trabajar como un periférico estándar cuando son conectados a un host USB estándar.

En el caso de los dispositivos OTG, la especificación USB On-The-Go (OTG) permite a los dispositivos con dicha especificación comportarse como "*host* a *periférico*" o de "*periférico a host*". Esto significa que OTG permite que dos dispositivos con OTG puedan conectarse entre sí y ser capaces de comunicarse.

La especificación hace que la Raspberry Pi pueda comportarse como un teclado/mouse y almacenamiento masivo utilizando solamente un puerto micro-USB. La implementación de dicha especificación está soportada por el kernel de Linux y lo hace simple de utilizar a través de un sistema de archivos llamado *ConfigFS* que en sí, termina configurando dispositivos emulados haciendo uso de la API llamada *USB Gadget API* [34]. En las siguientes subsecciones del presente capítulo se procede a explicar ambos conceptos con mayor detalle.

2.3.2 Linux USB-Gadget API

Linux USB Gadget es una API provista por el kernel de Linux. Éste tiene las siguientes características:

- Da soporte USB 2.0, dispositivos de alta velocidad que pueden enviar datos a una velocidad de hasta 60 Mbyte/s.
- Es lo suficientemente flexible para exponer funcionalidades complejas de USB así como múltiples configuraciones, múltiples interfaces, dispositivos compuestos y alternar configuraciones de la interfaz.
- Da soporte USB OTG, en conjunto a las actualizaciones a la API Linux-USB (host).
- Comparte la estructura de datos y modelos con la API Linux-USB (host). Esto quiere decir, que a nivel de desarrollo, utiliza las mismas estructuras, permitiendo usar el mismo modelo de E/S tanto por el controlador de host y el periférico.
- Es minimalista. Es decir, es más fácil dar soporte a nuevos controladores de hardware. El procesamiento de E/S no implica largas demandas de memoria o recursos de CPU.

Ambas, API Linux-USB y Linux USB-Gadget API, comparten definiciones comunes para los mensajes, estructuras y constantes del estándar USB. Además, ambas APIs vinculan y desvinculan los controladores de los dispositivos. Difieren en detalle, ya que el framework USB actual del lado del host expone una serie de detalles de implementación y suposiciones inapropiados para la API Linux USB Gadget.

Si bien, el modelo para las transferencias de control y la administración de la configuración es diferente (un lado es un maestro neutral de hardware, el otro es un esclavo consciente del

hardware), la API para el endpoint de E/S usado aquí debería ser también compatible con la API Linux USB [34].

2.3.3 Estructura de los Controladores Gadget

Un sistema corriendo dentro de un periférico USB normalmente tiene al menos tres capas dentro del kernel para manejar el procesamiento del protocolo USB y además, puede tener capas adicionales en el código del espacio de usuario. La API Gadget es usada por la capa media que interactúa con la capa de más bajo nivel (la cual interactúa directamente con el hardware).
En Linux, de abajo hacia arriba, estas capas son:

1. USB Controller Driver

   Es el nivel de software más bajo. Es la única capa que se comunica con el hardware, a través de registros, fifos, dma, irqs y similares. El hardware se expone a través de objetos de punto final (endpoints), que aceptan flujos de buffer (espacio de memoria temporal que se utiliza para almacenar datos antes de que se transfieran o procesen en otro lugar) de entrada/salida, y a través de callbacks (función que se pasa como argumento a otra función y se invoca después de que se haya completado la operación de la función principal) que interactúan con los controladores de gadget. Dado que los dispositivos USB normales tienen un puerto ascendente, solamente tienen uno de estos controladores. El controlador del controlador puede admitir cualquier número de controladores de gadget diferentes, pero sólo se puede usar uno de ellos a la vez. Ejemplos de este tipo de hardware de controlador son el controlador de alta velocidad NetChip 2280 USB 2.0 basado en PCI, el UDC SA-11x0 o PXA-25x (que se encuentra en muchas PDA) y una variedad de otros productos [32].

2. Controlador Gadget

   El límite inferior de este controlador implementa funciones USB independientes del hardware mediante llamadas al driver del controlador USB. Debido a que dicho hardware varía ampliamente en capacidades y restricciones, y se utiliza en sistemas

embebidos donde el espacio es escaso, el controlador del gadget a menudo se configura en tiempo de compilación para trabajar con endpoints compatibles con un controlador en particular. Los "Controladores Gadget" pueden ser portables a varios tipos de controladores diferentes, utilizando compilación condicional, dependiendo de la plataforma destino (los kernels recientes simplifican sustancialmente el trabajo involucrado en el soporte de nuevo hardware, al configurar los endpoints automáticamente para muchos controladores). Las responsabilidades del controlador de gadgets incluyen:

- Gestionar las solicitudes de instalación (respuestas del protocolo EP0), posiblemente incluyendo la funcionalidad específica de la clase.
- Retornar descriptores de configuración y cadena.
- Restablecer configuraciones y altsettings (configuraciones específicas relacionadas al protocolo USB) interfaz, incluida la habilitación y configuración de endpoints.
- Controlar eventos del ciclo de vida, como la administración de enlaces a hardware, suspensión/reanudación de USB, activación remota y desconexión del host USB.
- Administrar transferencias de E/S en todos los endpoints habilitados actualmente.

Los controladores de gadget pueden consistir de módulos de código propietario, aunque ese enfoque se desaconseja en la comunidad Linux [34].

3. Capa superior

La mayoría de los controladores de gadgets tienen un límite superior que se conecta a algún controlador en Linux o Framework en Linux. En este límite fluyen los datos que el controlador del gadget produce y/o consume a través de comunicaciones utilizando el protocolo USB. Algunos ejemplos son:

- Código de modo de usuario, utilizando archivos genéricos (gadgetfs) o específicos de la aplicación en /dev.
- Subsistema de red (para gadgets de red, como el controlador de gadget CDC Ethernet Model).
- Controladores de captura de datos, tal vez V4L2 o un controlador de escáner; o hardware de prueba y medición.

- Subsistema de entrada (para gadgets HID).
- Subsistema de sonido (para gadgets de audio).
- Sistema de archivos (para gadgets PTP).
- Subsistema de E/S de bloque (para dispositivos de almacenamiento USB), entre otros.

4. Capas adicionales

   Pueden existir otras capas, por ejemplo, incluir capas de kernel, como pilas de protocolos de red, así como aplicaciones en modo de usuario que se basan en la API de llamadas al sistema POSIX estándar como open(), close(), read() y write(). En los sistemas más nuevos, las llamadas de E/S asincrónicas POSIX pueden ser una opción. Dicho código en modo de usuario no estará necesariamente sujeto a la Licencia Pública General de GNU (GPL) [34].

Además de lo antes expuesto, las consideraciones generales a tener en cuenta para dispositivos OTG son: Los sistemas aptos para OTG deben incluir una pila estándar Linux-USB del lado del host, con *usbcore (*módulo que provee soporte USB para diferentes tipos de dispositivos USB), uno o más controladores de host (HCDs), controladores de dispositivos USB para soportar la "lista de periféricos dirigidos" de OTG, etc. Además, habrá un controlador OTG visible para los desarrolladores de dispositivos y gadgets sólo indirectamente. Esto ayuda a los controladores USB del lado del host y del dispositivo a implementar los dos nuevos protocolos OTG (HNP y SRP) [34], por un lado HNP (Host Negotiation Protocol), o protocolo de negociación con *host,* permite intercambiar entre ser *host o periférico* en una misma conexión; se implementa a través de puertos micro-AB en un dispositivo. Por otro lado, el protocolo SRP (Session Request Protocol), o protocolo de pedido de sesión, permite enviar un pedido de activación del bus USB y alimentación a un dispositivo conectado pero parcialmente apagado. El fin de este protocolo es el ahorro de energía cuando no existen comunicaciones activas.

2.3.4 USB Gadget - ConfigFS

Para generar USB Gadgets con las diferentes funcionalidades, el kernel de Linux provee un módulo que permite crearlos mediante la utilización de un sistema de archivos llamado ConfigFS [36].

ConfigFS es un sistema de archivos basado en la memoria ram que proporciona lo contrario de la funcionalidad de SysFS [35]. SysFS permite *visualizar/acceder* a los objetos del kernel a través de un sistema de archivos (filesystem), provee una manera de exportar datos desde las estructuras de datos que almacena el kernel, sus atributos y sus vínculos entre ellos con el espacio de usuario. ConfigFS es un *administrador* de objetos del kernel (*config_items).*

Con SysFS, se crea un objeto en el kernel (por ejemplo, cuando se descubre un dispositivo) y se registra en el sistema utilizando el mismo sistema de archivos (SysFS). Los atributos de ese objeto, aparecen en el sistema de archivos SysFS, que permite en el espacio de usuario leer los atributos a través de *readdir/read* (funcionalidades del kernel). Puede permitir que algunos atributos se modifiquen a través de *write* (funcionalidad del kernel). El punto importante es que el objeto se crea y se destruye en el kernel, el kernel controla el ciclo de vida de la representación de SysFS el cual permite acceder a todos estos objetos en el kernel.

Un *config_item* de ConfigFS se crea a través de una operación de espacio de usuario explícita: *mkdir* (funcionalidad del kernel). Se destruye mediante *rmdir* (funcionalidad del kernel). Los atributos aparecen en el momento de *mkdir* y se pueden leer o modificar a través de *read* y *write*. Al igual que con SysFS, *readdir* consulta la lista de elementos y/o atributos.

A diferencia de SysFS, el tiempo de vida de la representación depende completamente del espacio de usuario, por lo que los módulos del núcleo que respaldan los elementos creados con ConfigFS, cumplen el tiempo de vida que cumplen en el espacio de usuario. Tanto SysFS como ConfigFS pueden y deben existir juntos en el mismo sistema. Uno no es un reemplazo para el otro [35].

2.3.5 ConfigFS - Funcionamiento

A continuación, se presenta la idea general de cómo funciona ConfigFS [36]. En ConfigFS existen *elementos* y *grupos*, ambos representados como directorios. La diferencia entre un *elemento* y un *grupo* es que un *grupo* puede contener otros grupos. En la Figura 2.4 sólo se muestra un *elemento (*Ver Figura 2.4, elemento *cs).* Tanto los elementos como los *grupos* pueden tener atributos, que se representan como archivos.

El usuario puede crear y eliminar directorios, pero no puede eliminar archivos, los cuales pueden ser de sólo lectura o de lectura y escritura, según lo que representen. La parte del sistema de archivos de ConfigFS opera en *config_items/groups* y *configfs_attributes* que son genéricos y del mismo tipo para todos los elementos configurados. Sin embargo, están incrustados en estructuras más grandes específicas del uso. En la Figura 2.5, se muestra la vista del sistema de archivos, en ella hay un "cs" que contiene un *config_item* y un "sa" que contiene un *configfs_attribute.*

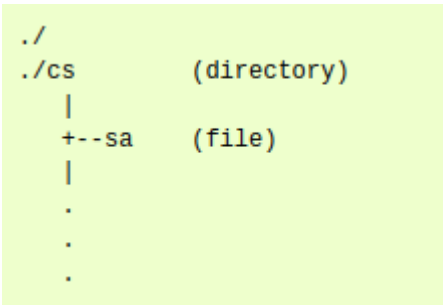

Figura 2.4. Ejemplo de sistema de archivos

Cada vez que un usuario lee/escribe el archivo *"sa"*, se llama a una función que acepta una estructura *config_item* y una *configfs_attribute* (ver Figura 2.5). En dicha función, "*cs*" y "*sa*" se recuperan utilizando la conocida técnica *container_of (*macro para obtener la dirección de memoria de estructuras de datos a partir de un atributo de la misma) y se llama a una función de *sa* apropiada (“*mostrar*” o “*almacenar*”) y se pasa el "*cs*" y un buffer de caracteres.

La función *"mostrar"* es para mostrar el contenido del archivo (copia datos del *"cs"* al buffer), mientras que el *"almacenar"* es para modificar el contenido del archivo (copia datos del buffer al *"cs"*), pero depende del implementador de las dos funciones para decidir lo que realmente hacen.

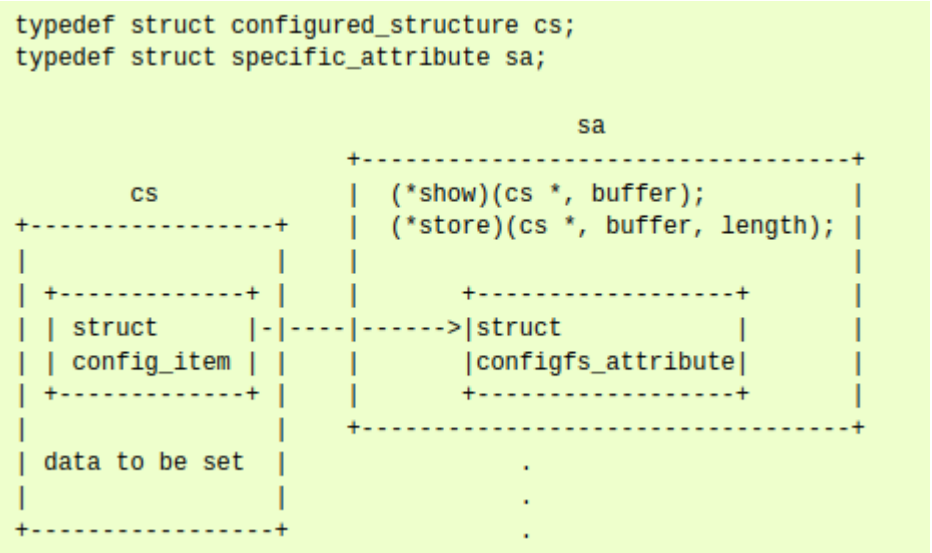

Figura 2.5. Estructura de datos de ejemplo representando el sistema de archivos en Figura 2.4.

Los nombres de los archivos se deciden por el diseñador de la configuración de item/grupo, mientras que los directorios en general se pueden nombrar a voluntad. Un grupo puede tener varios de sus subgrupos predeterminados creados automáticamente.

En resumen, los USB Gadgets creados utilizando ConfigFS están constituidos de la siguiente forma:

1. Un gadget tiene su grupo de configuración, que tiene algunos atributos (*idVendor*, *idProduct*, *etc.*) y subgrupos predeterminados (*configuraciones*, *funciones*, *strings*). Escribir en los atributos hace que la información se almacene en las ubicaciones adecuadas. En los subgrupos de configuraciones, funciones y strings, un usuario puede crear sus subgrupos para representar configuraciones, funciones y grupos de strings en un idioma determinado.

2. El usuario crea *configuraciones* y *funciones*. En las *configuraciones* se crean enlaces simbólicos a *funciones*. Esta información se usa cuando se escribe en el atributo UDC (Controlador de dispositivo USB) USB del gadget, lo que significa vincular el gadget al UDC. El código en

drivers/usb/gadget/configfs.c itera sobre todas las configuraciones, y en cada configuración itera sobre todas las funciones y las vincula. De esta manera, todo el dispositivo está vinculado.

3. El archivo *drivers/usb/gadget/configfs.c* contiene código para:

- *config_group* del gadget
- grupos predeterminados del gadget (configuraciones, funciones, string)
- asociar funciones con configuraciones (enlaces simbólicos)

4. Cada función USB naturalmente tiene su propia vista de lo que quiere configurar, por lo que los *config_groups* para funciones particulares se definen en los archivos de implementación de funciones *drivers/usb/gadget/f_*.c*.

5. El código de la función está escrito de tal manera que utiliza *usb_get_function_instance*() que, a su vez, llama a *request_module*. Entonces, siempre que *modprobe* (programa de Linux que permite cargar/remover un módulo cargable del kernel en tiempo de ejecución) funcione, los módulos para funciones particulares se cargan automáticamente.

Es a través del uso de ConfigFS que *TinyPilot* y *KeyAid* emulan los dispositivos dentro de la *Raspberry Pi*. *TinyPilot*, que se encuentra instalado en la Raspberry Pi, recibe los eventos a través de su interfaz web, los procesa y los comunica mediante USB a la máquina destino. Si bien se mencionan conceptos como USB Gadget, USB Gadget API, son todos conceptos para dar contexto de que, gracias a las APIs y controladores, ConfigFS logra ser una interfaz a nivel de usuario que permite generar y comunicarse con estos dispositivos de una manera relativamente simple.

Capítulo III

KeyAid: Diseño e Implementación

Las tareas de mantenimiento como acceder a configuraciones de la BIOS/UEFI, instalar un sistema operativo de forma remota, entre otras, se las puede considerar tareas no comunes a diferencia de las más habituales como es el caso de instalar programas, ayudar con alguna aplicación en particular, etc. Su resolución implica conocimiento técnico específico. Contar con una herramienta que permita realizarlo en forma remota, implica una mejora sustancial en la solución del problema: el personal técnico no necesita trasladarse y los tiempos se reducirían sustancialmente.

El propósito del proyecto es proveer una nueva forma de interacción con el servicio técnico a la hora de necesitar un arreglo/mantenimiento. El dispositivo externo, una Raspberry Pi, permite al *técnico* tener un control total de la PC/Servidor del *cliente*, y la plataforma web es la responsable de proveer el medio de interacción entre los técnicos y clientes, sin necesidad de conocimiento mutuo y cercanía espacial. Además, le permite al *cliente* seleccionar la mejor oferta para la resolución de su problema.

En este capítulo describimos una herramienta para llevar a cabo esta tarea. Se propone, mediante el uso de un dispositivo externo que actúa como un controlador de la PC del usuario, *cliente, y* una plataforma web de interacción entre éste, quien necesita el arreglo o mantenimiento y el *técnico,*

coordinar los tiempos donde el técnico va realizar el trabajo, las tareas a desarrollar, el costo y la gestión de pago.

3.1. KeyAid: Características Generales

KeyAid es una herramienta integral, la cual le permite al usuario, *cliente*, solicitar y resolver problemas comunes y no tan comunes en su equipo de hardware mediante un servicio técnico. Al *técnico*, el *cliente*, lo puede seleccionar entre los que le ofrecieron sus servicios y establecieron sus condiciones. No importa la ubicación geográfica del *cliente, técnico*, equipamiento, etc., *KeyAid* permite la resolución remota del problema en los momentos pactados.

Como *KeyAid* es una herramienta integral, está compuesta de varios módulos. En la Fig. 3.1. Se muestra un esquema del sistema completo con cada una de sus componentes. Para mayor información, ver Figura A1.1 y A1.2 en el Anexo I, donde se visualiza el modelo de dominio y de actividad respectivamente.

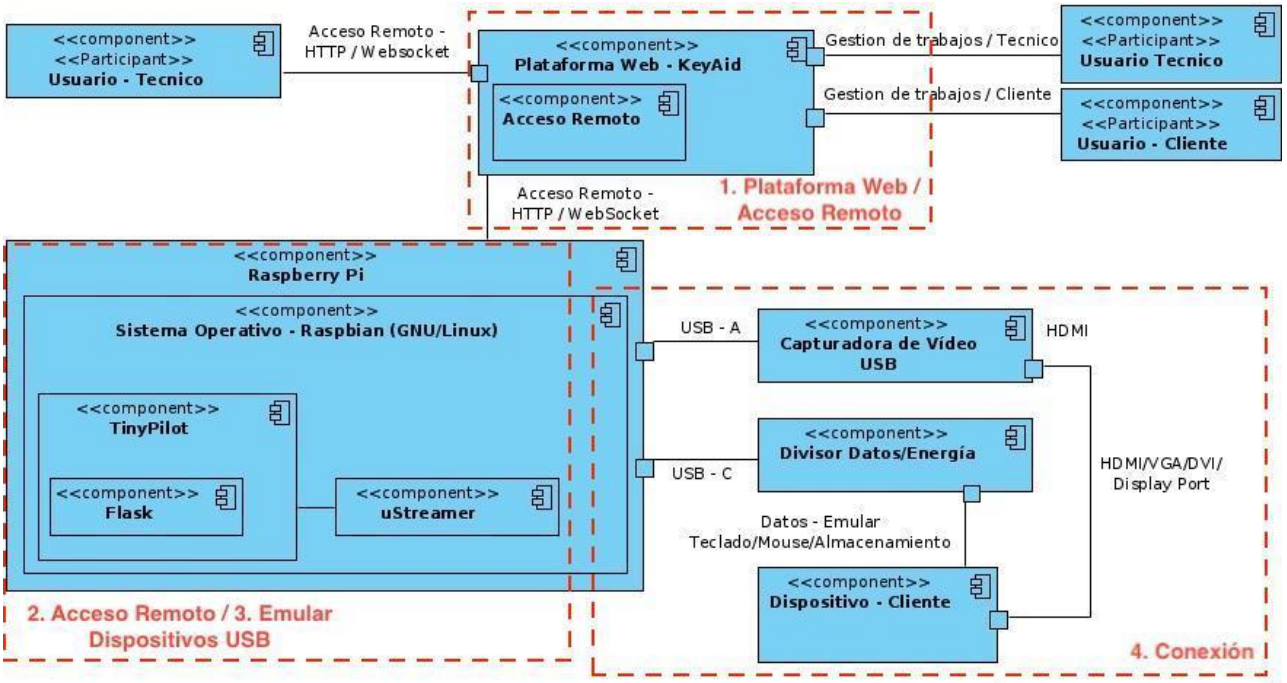

Figura 3.1. Modelo de componentes de "*KeyAid*"

Se puede visualizar todos los componentes involucrados, ellos son:

1. *Plataforma Web - KeyAid,* para la comunicación *Cliente-Técnico*.
2. Módulo *Acceso Remoto*, donde se integra *TinyPilot* en la plataforma web.
3. Módulo *Emular dispositivos USB,* donde se desarrolla la extensión en *TinyPilot* para tal fin.
4. Módulo *Conexión,* responsable de llevar a cabo la conexión entre la Raspberry Pi y la máquina del usuario final.

En las siguientes secciones se procede a explicar detalles de diseño e implementación de cada uno de ellos.

3.2. Plataforma Web - KeyAid

Para el desarrollo de la plataforma web se formularon las funcionalidades mínimas que la plataforma debía contemplar:

- Permitir a los usuarios *clientes* crear trabajos para los cuales los *técnicos* que estén interesados en resolverlo puedan postularse.
- Permitir a *técnicos* postularse a dichos trabajos y gestionar el cobro por el trabajo realizado mediante la plataforma.

A su vez, se definieron las funcionalidades para cada tipo de usuario:

- Cliente - (Usuario que necesita mantenimiento en su dispositivo)
    - Crear trabajo con la información necesaria para que el técnico defina las tareas a ser realizadas.
    - Revisar las postulaciones para un trabajo dado donde se visualicen tareas y costo total por el trabajo.
    - Poder seleccionar una de las postulaciones para asignarle el trabajo al técnico postulado.
    - Envío de dinero al técnico mediante servicios de pagos de terceros, ej. MercadoPago.
- Técnico - (Usuario que realiza tareas de mantenimiento a clientes)
    - Realizar postulación a un trabajo dado especificando tareas y el costo total por el trabajo.
    - Poder administrar el dispositivo del *usuario final* de manera remota y completa.

- Recibir el pago por el trabajo realizado

El desarrollo de la plataforma web involucró el desarrollo del *front-end* y el desarrollo del *back-end.*

3.2.1 Módulo Front-End

En el desarrollo del *front-end* se hizo uso del framework *Flutter* debido a que tiene la posibilidad que desde un único código fuente (utilizando el lenguaje *Dart*), puede generar los binarios para las diferentes plataformas (Windows, Linux, Android, iOS, Web). Para el presente trabajo, solamente se realizó el despliegue de la versión web del sistema.

Con *Flutter*, la manera de manejar el estado de la aplicación (que se refiere a cómo se comparte el estado entre las diferentes *vistas*) puede realizarse haciendo uso de diferentes librerías provistas por *Flutter* y otros proyectos de código abierto como *GetX*, *Redux*, *BloC*, *Provider*, entre otros. Se optó por elegir *GetX*, ya que permite realizar la codificación del manejo de estados de una manera simple, junto al manejo de navegación entre pantallas y la internacionalización, por lo cual redujo la cantidad de código producido.

La implementación se desarrolló siguiendo el patrón de diseño MVC (Modelo-Vista-Controlador), el cual es un patrón que divide a la aplicación en tres componentes, el *modelo* que lleva los datos y la *vista* y el *controlador*, que juntos comprenden la interfaz de usuario. Cada vez que hay un cambio en los datos, el controlador recibe el evento y actualiza los datos mostrados en la vista, manteniendo consistencia entre los datos y lo que recibe/visualiza el usuario. La razón de usar un patrón como MVC, es que permite independizar los componentes entre sí. Los cambios realizados a la interfaz de usuario no deben impactar en los otros componentes, así como si hubiesen cambios al resto de los componentes. Si bien existe cierta dependencia entre las capas, permite que uno pueda modificar la lógica de negocio sin alterar la *vista*. Lo mismo sucede en el caso de la parte visual (*vista/controlador)*, si uno cambiara la estética de botones, ventanas, entre otros, no tiene porqué afectar a la lógica de negocio.

3.2.2 Módulo Back-end

Para el back-end se optó por utilizar una PaaS (Platform as Service) llamado *Firebase* dado que provee un conjunto de servicios que permite acelerar el desarrollo. Dentro de los servicios que brinda *Firebase,* se hizo uso de:

- *Cloud Storage* [37]: Cloud Storage permite almacenar archivos en un bucket de Google Cloud Storage y los hace accesibles a través de Firebase y Google Cloud. Esto permite tener la flexibilidad para subir y descargar archivos de clientes móviles a través de los SDK de Firebase para Cloud Storage.
- *Cloud Firestore* [38]: es una base de datos NoSQL orientada a documentos. A diferencia de una base de datos SQL, no hay tablas ni filas. En su lugar, los datos se almacenan en *documentos*, que se organizan en *colecciones*.

  Cada *documento* contiene un conjunto de pares clave-valor. Firestore está optimizado para almacenar grandes colecciones de documentos pequeños. Todos los documentos se deben almacenar en colecciones. Pueden contener *subcolecciones* y objetos anidados, que pueden incluir campos primitivos, como strings, o tipos de objetos complejos, como listas.

  Las colecciones y los documentos se crean de manera implícita en Firestore; sólo se debe asignar datos a un documento dentro de una colección. Si la colección o el documento no existe, Firestore los crea.
- *Firebase Authentication* [39]: El servicio Firebase Authentication proporciona el SDK y bibliotecas de interfaz de usuario fáciles de usar para autenticar a los usuarios en su aplicación. Incluso maneja tareas como la fusión de cuentas, que si se hace manualmente puede ser compleja. A su vez permite el logueo por parte del usuario mediante diferentes plataformas como *Google*, *Facebook*, *Github*, entre otros.

Firebase Authentication brindó la facilidad de manejar la autenticación de manera simple y segura, así como acceder a los datos sólo si el usuario se encuentra con la sesión iniciada en el sistema con sus permisos correspondientes.

Cloud Firestore se utilizó para el almacenamiento de los datos de la aplicación de manera escalable. Entre la información almacenada de la aplicación se encuentran:

- *Postulaciones* de los técnicos a distintos trabajos.

- *Trabajos* requeridos por los usuarios finales.
- *Reviews* de los técnicos por trabajos realizados.
- *Habilidades* que pueden tener los técnicos.
- *Tipos de tareas* que pueden ser requeridas para los trabajos.
- Información relacionada a *usuarios.*

Cloud Storage fue utilizado para el almacenamiento de imágenes en caso que el usuario necesite enviar información adicional referida a un trabajo para poder dar una mejor descripción al técnico.

Para el procesamiento de pagos realizados por parte del usuario *cliente,* en el trabajo se hizo uso de *MercadoPago* [42] para tercerizar el tratamiento de dicha información. En la implementación de la plataforma solamente se hace uso de *MercadoPago* para poder administrar los pagos a través de la plataforma de KeyAid y que sea simple desde el punto de vista del usuario realizarla. La razón de porqué tercerizar el tratamiento es que ya existen empresas que han solucionado los problemas de seguridad que eso puede presentar, así como la encriptación de datos, que se cumplan las normativas del país donde ocurra el pago, entre otras posibles consideraciones.

3.3. Módulo Acceso Remoto

Para poder brindar el acceso remoto de manera completa al *técnico* y brindar a su vez, una experiencia de usuario agradable para el *cliente*, se necesitaba la integración de *TinyPilot* dentro de la plataforma web desarrollada en *Flutter.*

Para lograr dicha integración, se cuenta con la *Raspberry Pi* del lado del *cliente* donde *TinyPilot* se encuentra instalado, y éste provee un servidor web, el cual, es posible ser accedido en la dirección IP de la *Raspberry Pi* del *cliente.*

Para el *cliente,* con tener la *Raspberry Pi* conectada a su PC y a la fuente de alimentación, es suficiente para que el técnico pueda acceder a su PC.

Para el *técnico,* se requirió agregar un submódulo a la plataforma web para acceder al servicio web provisto por *TinyPilot,* y así poder realizar las tareas especificadas para el *cliente*. Éste consta de un navegador integrado provisto por *Flutter* que accede a la dirección IP de la *Raspberry Pi*.

3.4. Módulo Emular dispositivos USB

TinyPilot en su versión paga ofrece la emulación de dispositivos USB ya que, el poder emular un dispositivo de almacenamiento masivo permite ofrecerle al técnico, entre otras posibilidades:

- Elegir un sistema operativo y cargarlo en el almacenamiento para posteriormente instalarlo.
- Usar el almacenamiento para almacenar datos de forma temporal en forma de resguardo.

Para el alcance del proyecto, se optó por modificar el proyecto original de *TinyPilot*, logrando el objetivo de poder emular el dispositivo de almacenamiento USB. Se logró haciendo uso de los conocimientos sobre *ConfigFS*, explicado anteriormente en el Capítulo II, lo que permite cargar/instalar un sistema operativo (Ubuntu Linux, Windows, entre otros). En la Figura 3.2 se muestra como están configurados los dispositivos emulados utilizando ConfigFS, en la línea 6 a 8 se define, siguiendo la documentación de ConfigFS, los directorios para poder empezar a configurar los dispositivos tales como el teclado, mouse y el almacenamiento USB. En la línea 10 a 14, se definen los directorios donde se van a configurar las funcionalidades para cada uno de los dispositivos emulados (*functions/<dispositivo>*). En la línea 16 a 19, están definidos los directorios donde se definen las configuraciones generales para el dispositivo USB que emula las tres funcionalidades (mouse, teclado y almacenamiento USB). Luego en la Figura 3.3, desde la línea 159 a 160, se define el directorio en que se almacenarán los datos dentro de la Raspberry Pi, los cuales son aquellos que se almacenarán en el dispositivo de almacenamiento por el *técnico* (Sistemas operativos, archivos de resguardo, entre otros). Ambas figuras, 3.2 y 3.3, son una muestra parcial del código que logra la configuración de estos dispositivos, pero en éstas se

visualizan las líneas esenciales que permiten la configuración del dispositivo de almacenamiento USB.

Para lograr una experiencia de usuario que sea placentera para el técnico a la hora de necesitar descargar e instalar sistemas operativos, *Ventoy* permite agregarlos en modo de archivos en formato ISO/WIM/IMG/VHD(x)/EFI en el almacenamiento USB. El mismo provee de un menú de inicio con los sistemas operativos cargados en el almacenamiento USB. Esto hace que sea tan simple como copiar y pegar archivos dentro del almacenamiento USB. Además de ello, KeyAid, dentro de sus trabajos futuros, agregará una interfaz para dicha funcionalidad, para mejorar dicha experiencia.

A su vez, los sistemas operativos o datos que hayan quedado almacenados en el almacenamiento USB quedarán almacenados para un próximo uso, por lo que permite al usuario y al técnico ahorrar el tiempo de descarga para un posible futuro.

```
6   export readonly USB_DEVICE_DIR="g1"
7   export readonly USB_GADGET_PATH="/sys/kernel/config/usb_gadget"
8   export readonly USB_DEVICE_PATH="${USB_GADGET_PATH}/${USB_DEVICE_DIR}"
9
10  export readonly USB_STRINGS_DIR="strings/0x409"
11  export readonly USB_KEYBOARD_FUNCTIONS_DIR="functions/hid.keyboard"
12  export readonly USB_MOUSE_FUNCTIONS_DIR="functions/hid.mouse"
13  export readonly USB_MASS_STORAGE_NAME="mass_storage.0"
14  export readonly USB_MASS_STORAGE_FUNCTIONS_DIR="functions/${USB_MASS_STORAGE_NAME}"
15
16  export readonly USB_CONFIG_INDEX=1
17  export readonly USB_CONFIG_DIR="configs/c.${USB_CONFIG_INDEX}"
18  export readonly USB_ALL_CONFIGS_DIR="configs/*"
19  export readonly USB_ALL_FUNCTIONS_DIR="functions/*"
```

Figura 3.2. Configuración del dispositivo emulado con ConfigFS

```
158  # Mass Storage
159  mkdir -p "${USB_MASS_STORAGE_FUNCTIONS_DIR}"
160  echo "/mass_storage/data.bin" > "${USB_MASS_STORAGE_FUNCTIONS_DIR}/lun.0/file"
```

Figura 3.3. Configuración de lugar de almacenamiento para el dispositivo USB emulado.

3.5. Módulo Conexión

Para poder controlar la máquina del usuario *cliente* de manera remota, haciendo uso de la Raspberry Pi, es necesario la transferencia de:

1. Eventos de teclado/mouse.
2. Salida de vídeo de la máquina del *cliente.*

En las siguientes secciones se detallan las características principales de cada una.

3.5.1 Eventos de teclado/mouse

El envío de eventos de teclado/mouse desde la Raspberry Pi hacia la máquina del usuario *cliente*, es posible gracias al protocolo OTG, soportado por el procesador Broadcom BCM2711. La comunicación entre la máquina del usuario *cliente* y la Raspberry Pi con soporte OTG, es plausible a través de su puerto USB-C.
Una de las problemáticas que esto presenta, es que, por dicho puerto USB-C, también se debe alimentar energéticamente a la Raspberry Pi. Por ello es necesario dividir, los datos que son enviados desde la Raspberry Pi hacia la máquina del usuario *cliente,* y la energía que va hacia la Raspberry Pi. Para lograr dicha división, fue necesario un "*Splitter*" o divisor de corriente/datos, de manera que independice eléctricamente [40] a la Raspberry Pi de la máquina usuario *cliente* pero con la posibilidad de enviar los datos a la máquina cliente, ver Figura 3.4.

En la Figura 3.4 se pueden visualizar las conexiones con la Raspberry Pi. El cable que provee a la misma de alimentación y la comunicación de datos, es el que referencia el punto 2. El cable que realiza la alimentación energética a la Raspberry Pi, es a través del cable "Energía" (punto 4) y finalmente, el cable que se encuentra sin conexión (punto 2), es el cable que comunica los datos a la PC/Servidor del cliente, utilizando un conector USB-A.

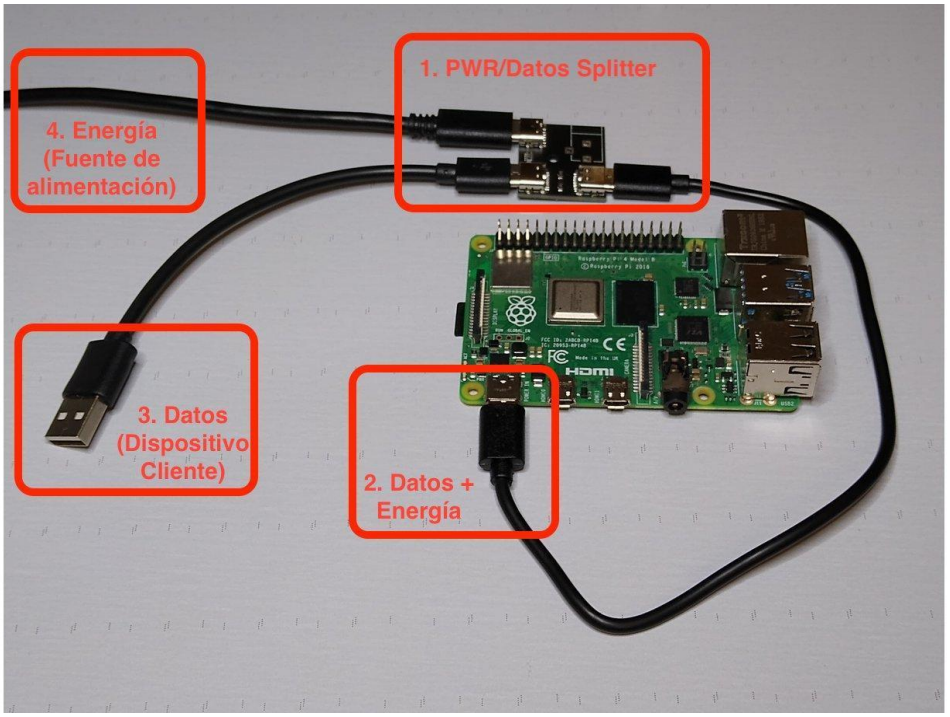

Figura 3.4. Conexión de Raspberry Pi 4b utilizando el “Splitter”.

Esto permite hacer uso de fuentes de alimentación externas sin peligro de dañar componentes ya sea en la máquina destino, o en la Raspberry Pi, cuando se encuentran conectadas entre sí. Para aislar eléctricamente tanto al Host como al Periférico, hay distintas maneras de lograrlo [40]. En el caso concreto del *Splitter*, si bien no hace uso del circuito integrado descrito en [40], se procede a explicar consideraciones necesarias en la implementación del aislamiento eléctrico.

Un ejemplo de cómo se realiza la implementación del aislamiento se puede visualizar en la Figura 3.5, donde la línea discontinua muestra el aislamiento que divide conceptualmente el cable USB. La información sobre el estado de *D+* y *D-* puede cruzar la barrera, pero la corriente no. *GND1* (la referencia de tierra del lado del *Host*) ahora es un nodo separado de *GND2* (la referencia de tierra del lado del *periférico*).

Desafortunadamente, el aislamiento evita que el *host* "vea" la resistencia *pull-up* del *periférico*, y el *periférico* no puede "ver" las resistencias *pull-down* del *host*. Por lo tanto, se necesitan algunas resistencias adicionales, como se muestra en la Figura 3.5, para imitar la conexión de sus contrapartes a través del aislamiento.

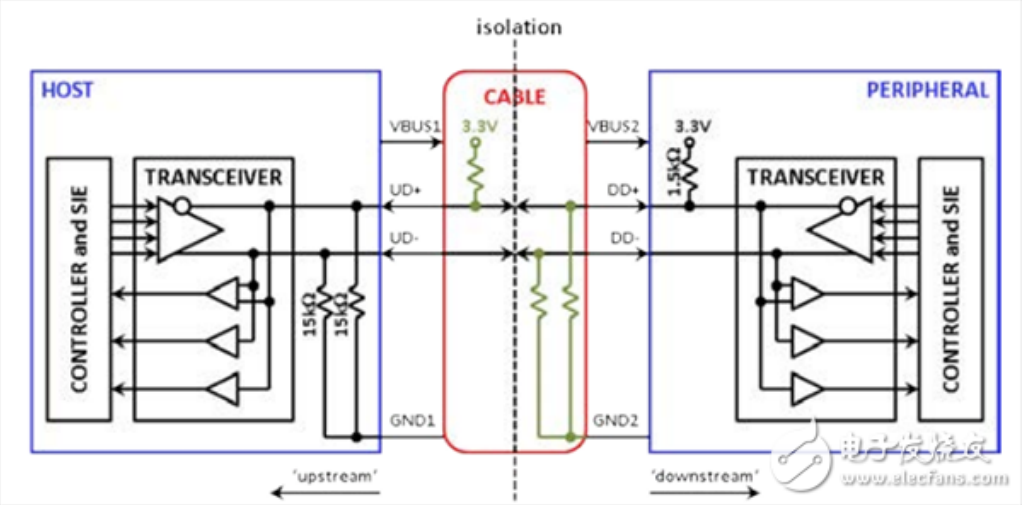

Figura 3.5. Ejemplo del esquema de aislamiento eléctrico de datos/corriente.

3.5.2 Captura de señal vídeo de la máquina del cliente

Para el envío de la señal vídeo de la máquina cliente, primero es necesario realizar captura del mismo, procesar los datos en la Raspberry Pi, para luego ser enviados al técnico.

Para la captura del vídeo se hace uso de una capturadora USB/HDMI de 1080p/30fps, la cual se conecta a la Raspberry Pi a través de uno de sus puertos USB-A, y recibe una conexión HDMI. Ver Figura 3.6. El chip interno usado por este tipo de capturadoras, entre otras genéricas, es el *MacroSilicon MS2109*. La ventaja principal de esta capturadora, entre otras, es que es soportada por V4L (Vídeo for Linux), lo cual permite el soporte para vídeo en tiempo real. Por ello, se puede hacer uso de él para diversos propósitos, como en este caso, redireccionar los datos de vídeo a través de la plataforma web hacia el técnico.

Figura 3.6. Capturadora de Vídeo 1080p/30fps - HDMI a USB 2.0

El redireccionamiento de los datos de vídeo desde la capturadora hacia la plataforma web, se logra mediante *TinyPilot* y principalmente al proyecto de código abierto *uStreamer*, el cual corre su propio servidor HTTP que brinda el vídeo en MJPEG, que es un formato que cualquier navegador lo puede reproducir de manera nativa. *TinyPilot*, hace utilidad de ello y expone el vídeo capturado en MJPEG para ser consumido por el navegador cliente. Además de ello, TinyPilot agregó una optimización a *uStreamer*, donde uStreamer, lo que hace es decodificar el vídeo de entrada y luego codifica el mismo en el formato de vídeo especificado, en este caso MJPEG. La optimización que se le realizó fue no realizar la decodificación, ya que, el formato de salida de vídeo de la capturadora USB es MJPEG. Dicha optimización redujo la latencia entre lo que visualiza el técnico y lo que ocurre en la máquina destino de forma significativa.

En el próximo capítulo, se describirán los resultados de las pruebas realizadas utilizando la plataforma web KeyAid junto a la Raspberry Pi (Dispositivo KeyAid). Como se describió a lo largo del presente capítulo, los principales puntos que pueden fallar son, las conexiones entre la Raspberry Pi y el dispositivo del cliente, así como, el direccionamiento del vídeo por parte de la PC/Servidor del cliente. Por direccionamiento del vídeo se refiere a que la mayoría de netbooks/notebooks, apenas son encendidas, antes de la carga del sistema operativo, dirigen el vídeo hacia el monitor integrado, una vez que el sistema operativo es cargado, en caso que detecten una conexión activa en el puerto de vídeo, redireccionan el vídeo por dicho puerto. Un escenario que fuerza al redireccionamiento de vídeo por dicho puerto, es cerrando la tapa de la netbook/notebook, haciendo que la única forma de salida de vídeo sea por ese puerto. El único problema que presenta dicho enfoque es que requiere de que el cliente cierre la tapa una vez encendida, y que la BIOS/UEFI de la netbook/notebook lo soporte. En base a lo descrito en este párrafo, se definieron los criterios a utilizar para la evaluación experimental.

3.5.3 Consideraciones de seguridad de la conexión entre *técnico-cliente*

La conexión entre el *técnico* y el *cliente*, ocurre entre el dispositivo del cliente y la Raspberry Pi. El acceso a dicha conexión, es una potencial vulnerabilidad que implicaría el acceso total al dispositivo del cliente por parte de entidades no autorizadas. Para minimizar los riesgos de

accesos no autorizados se detallan a continuación consideraciones/políticas para la conexión entre ambos dispositivos:

1. La conexión debería ser gestionada por la plataforma y no por el técnico.
    a. Se debe a que en caso de una actitud sospechosa, la plataforma/cliente, tengan la posibilidad de desvincular al *técnico.*
2. La conexión debe ser cifrada, se podría hacer uso de VPNs (Redes Privadas Virtuales), para mejorar la seguridad en este aspecto.
3. Siempre que la conexión se encuentre activa, el cliente debe ser notificado y tener la capacidad en todo momento de cancelar la conexión.
4. La gestión de las credenciales siempre debe ser gestionada por la plataforma. El técnico nunca debería poder tener acceso a dichas credenciales.

**En el presente trabajo, no se profundice en el detalle de cómo implementar cada una de estas consideraciones.

Para el presente trabajo, para lograr una conexión segura entre la Raspberry Pi y el *técnico*, se utilizó un servicio llamado *Tailscale* [41], el cual es un servicio que genera VPNs (Redes Privadas Virtuales) entre dispositivos asociados a una cuenta.

Capítulo IV

Perfiles y Usos de *KeyAid*

En este capítulo se explican los diferentes casos de uso, tanto por parte del *técnico* y del *cliente*, para mostrar cómo sería un caso donde un *cliente* crea un trabajo para una dificultad técnica que esté experimentando con su computador, y cómo el *técnico* se postula al mismo, y comienza a trabajar en ello. Dichos casos de uso, serán posibles de ver a través de imágenes mostrando la interfaz gráfica de la plataforma web, tanto desde el punto de vista del *cliente* como del punto de vista del *técnico*.

4.1. Idea Básica del Funcionamiento

KeyAid se trata de una plataforma web que logra conectar usuarios que necesitan resolver algún problema relacionado a su dispositivo (*clientes)* con usuarios calificados para poder resolverlos (*técnicos*). A su vez, consigue simplificar el acceso remoto y la gestión del trabajo realizado, tal como, describir al *cliente* las tareas a realizar, mostrar el costo total por el trabajo, como así también permitirle al *cliente* pagar desde cualquier parte con métodos de pago electrónicos. A continuación, se describe una breve idea de cómo interactúan ambos tipos de usuarios en la plataforma:

1. El *Usuario cliente* crea un *trabajo* describiendo las necesidades o problemas que pueda estar teniendo su dispositivo en la plataforma.
2. El *Usuario técnico* puede postularse a los trabajos especificando *tareas a realizar, fechas de inicio/final de trabajo, horario de trabajo* y *costo* de las tareas para que el *usuario* pueda tener conocimiento de lo que se va a hacer en su dispositivo y el costo a pagar del trabajo. A su vez, los horarios especifican los momentos del día donde el *técnico* necesita que el dispositivo esté conectado para poder tener acceso remoto al dispositivo del *usuario.*

A su vez, la plataforma web ofrece un portal para que el *técnico* pueda acceder al dispositivo del *cliente* de forma remota, y desde dicho portal, es que el técnico puede comenzar a trabajar en las tareas que especificó en la postulación al trabajo.

¿Cómo es que el *técnico* dentro de ***KeyAid*** puede controlar remotamente la PC del *cliente*? El dispositivo externo (Raspberry Pi) permite que esto sea posible porque dispone de un servidor web donde recibe los eventos de teclado/mouse, entre otros, de manera remota y los emula en la máquina destino gracias al uso de la especificación ***OTG (USB On-The-Go)*** [32] a través de una de las funcionalidades que provee el kernel de Linux (ConfigFS). A su vez, envía el vídeo de la pantalla del dispositivo al *técnico* para tener acceso a la salida de vídeo del *cliente* a través de una capturadora de vídeo.

Para el proyecto se utilizó la *Raspberry Pi 4* [8] como *"Proof of concept"*, pero en un futuro como posible mejora, se puede simplificar esta interfaz y lograr el mismo resultado.

4.2. Ejemplo del Funcionamiento

A continuación, se procede a explicar los principales flujos o escenarios de cómo los *técnicos* y *clientes* interactúan a través de la plataforma web *"KeyAid"*. Para mayor detalle, puede referirse al Anexo II, para ver la guía de manual de usuario.

4.2.1. Escenario - Primera vez en la Plataforma

En este caso, el usuario es la primera vez que ingresa a la Plataforma. Para ello debe:

1. Usuario se registra en plataforma como *técnico* o como *cliente* (usuario final). Ver Figura 4.1.

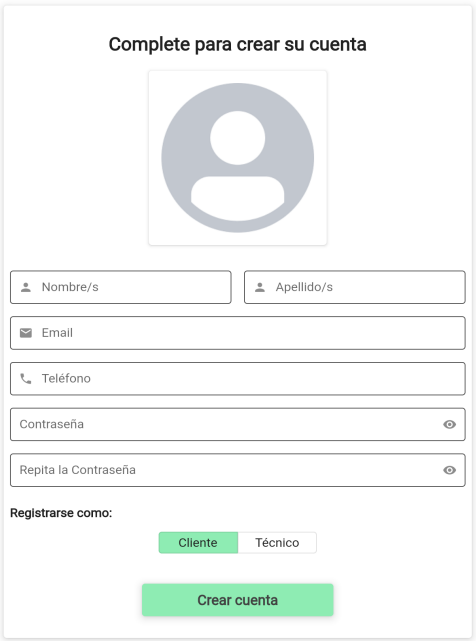

Figura 4.1. Formulario de registro para la plataforma *KeyAid*

4.2.2. Escenario de Usuario *cliente* - Crear un trabajo

En este escenario se muestra cómo un usuario *cliente* puede crear un trabajo en la plataforma para que distintos *técnicos* puedan postularse al mismo.

1. El *cliente* crea un trabajo Simple o Avanzado (Ver Figura 4.2).

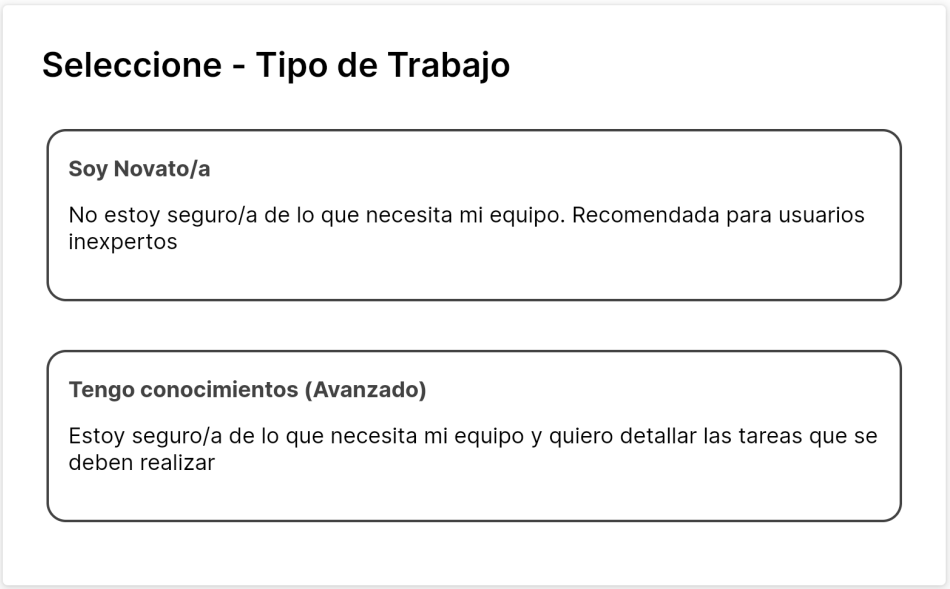

Figura 4.2. Pantalla de selección el tipo de trabajo a crear por el *cliente*

Considerando que los *clientes* en la plataforma pueden ser usuarios experimentados o usuarios con poco conocimiento, éstos pueden especificar de forma detallada cada una de las tareas a realizar (Ver Figura 4.3) o en el caso de los usuarios con poca experiencia, sólo detallan la información mínima necesaria que el *técnico* necesita (Ver Figura 4.4).

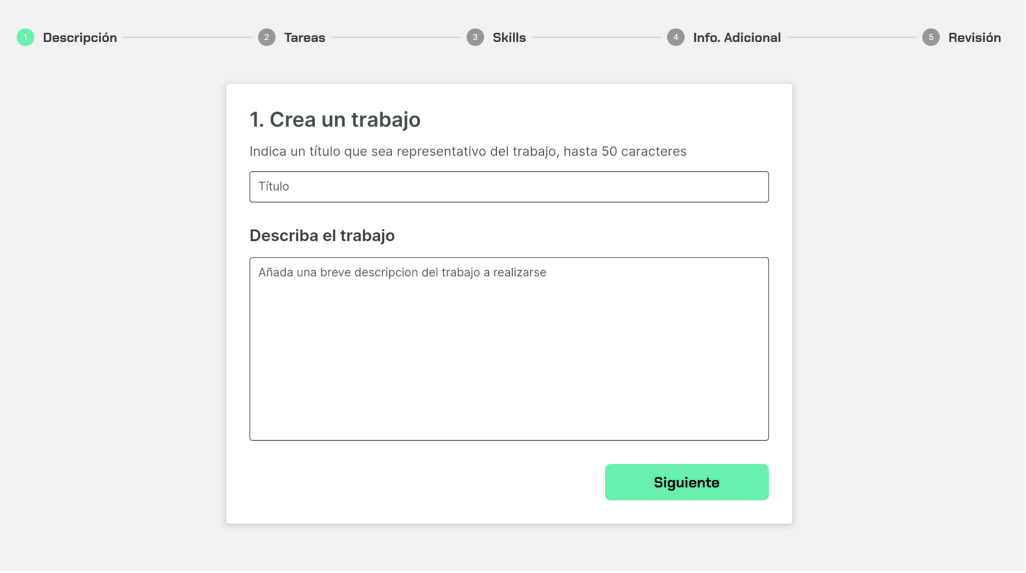

Figura 4.3. Pantalla de creación de trabajo para usuarios avanzados.

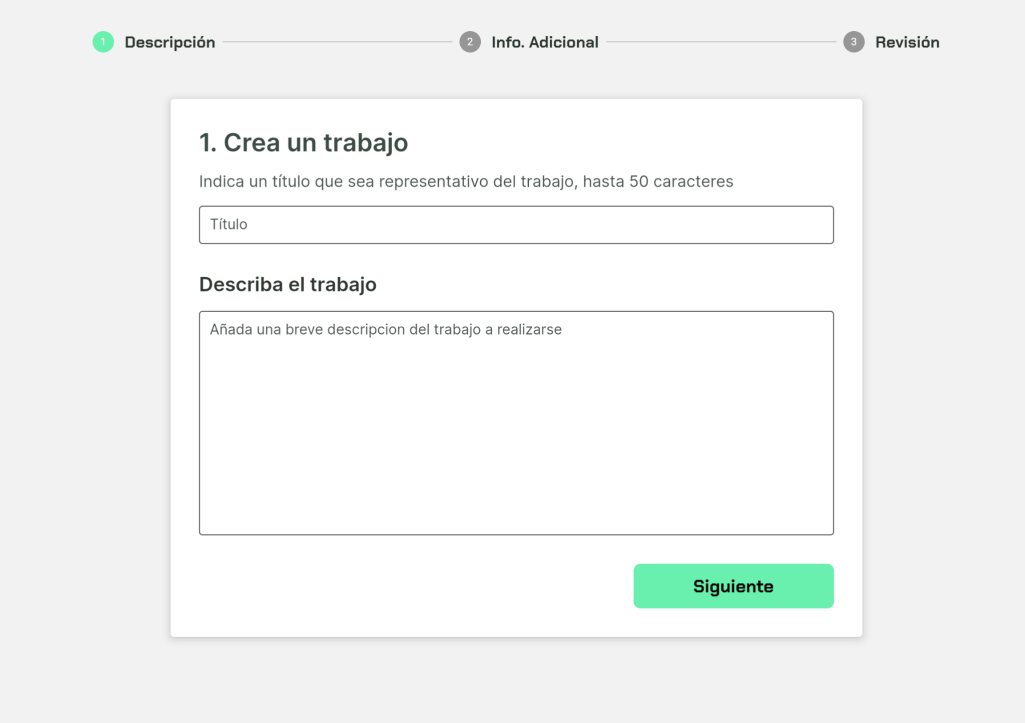

Figura 4.4. Pantalla de creación de trabajo para usuarios novatos.

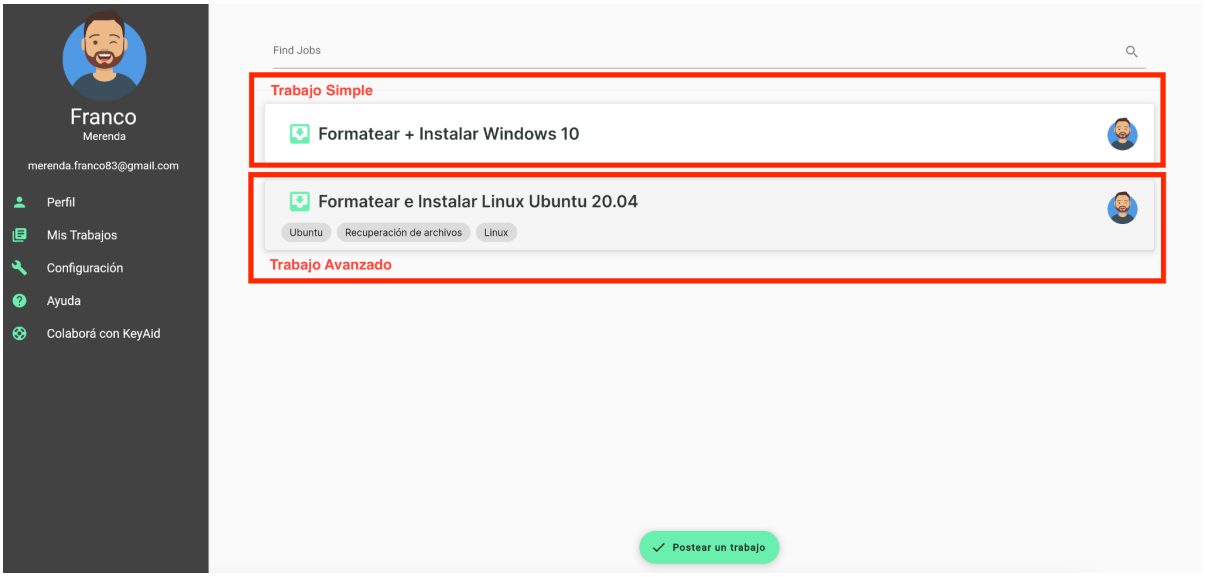

Figura 4.5. Ejemplo de un trabajo *simple* y un trabajo *avanzado*, creado por el mismo *cliente*.

2. El *cliente* espera por postulaciones de técnicos en la plataforma para evaluar el costo y el trabajo que se realizará (Ver Figura 4.6).

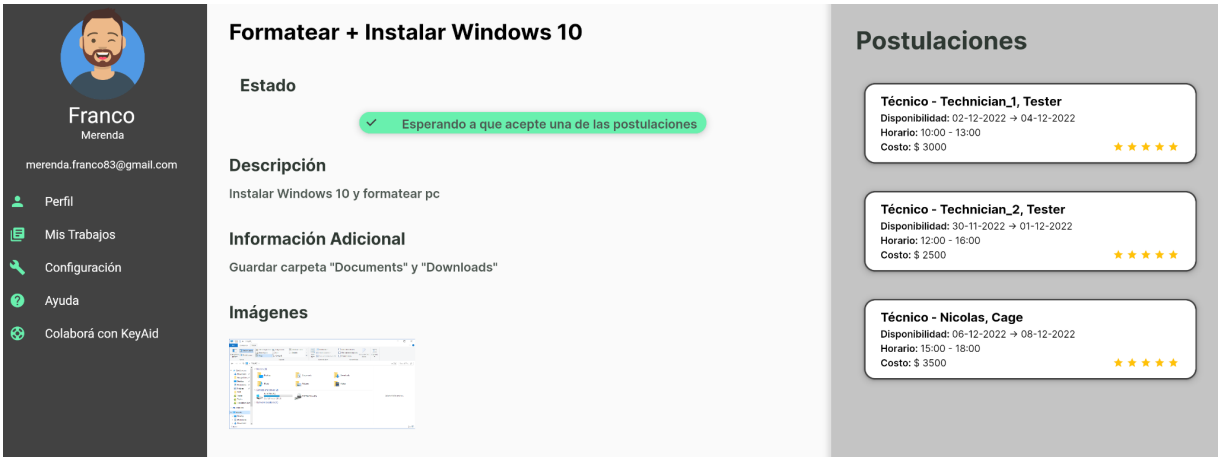

Figura 4.6. Ejemplo de trabajo de *cliente* con 3 postulaciones para el mismo.

3. El *cliente* elige una de las postulaciones y la confirma. Ver Figura 4.7 y 4.8.
    a. En la Figura 4.7 se muestra una postulación para el trabajo, donde el *cliente* puede ver las tareas realiza el técnico, el costo total por el mismo, y la disponibilidad del técnico para poder realizarlo.
    b. En la Figura 4.8 se muestra una postulación que fue confirmada para el trabajo.

Figura 4.7. Ejemplo de vista de *cliente* para confirmar postulación para el trabajo.

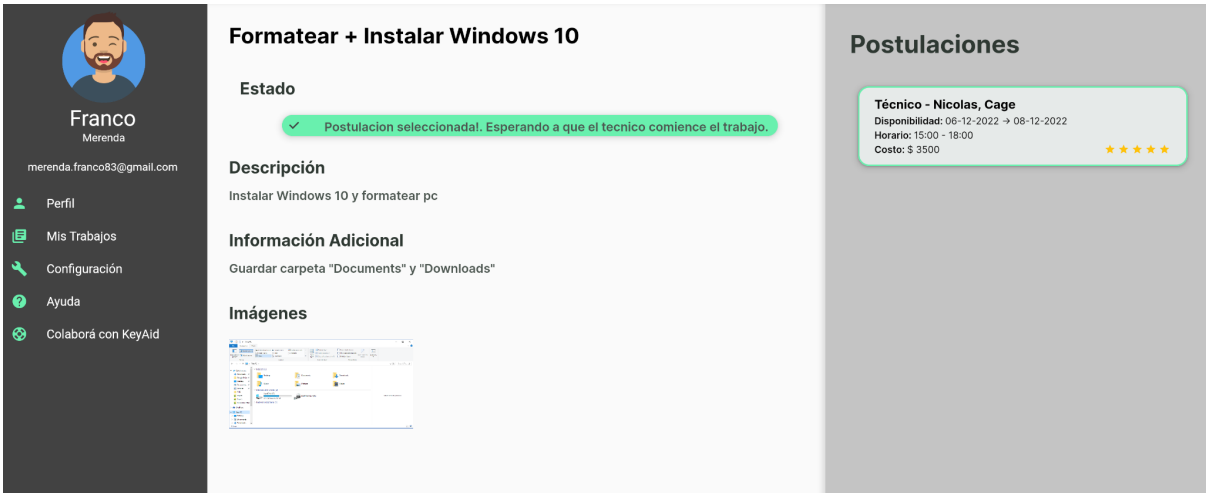

Figura 4.8. Trabajo con postulación confirmada

4. El *cliente* se debe asegurar de que la Raspberry Pi se encuentre conectada a su máquina durante los días especificados por el técnico.
5. Espera a que el técnico termine el trabajo. Ver Figura 4.9 y 4.10.

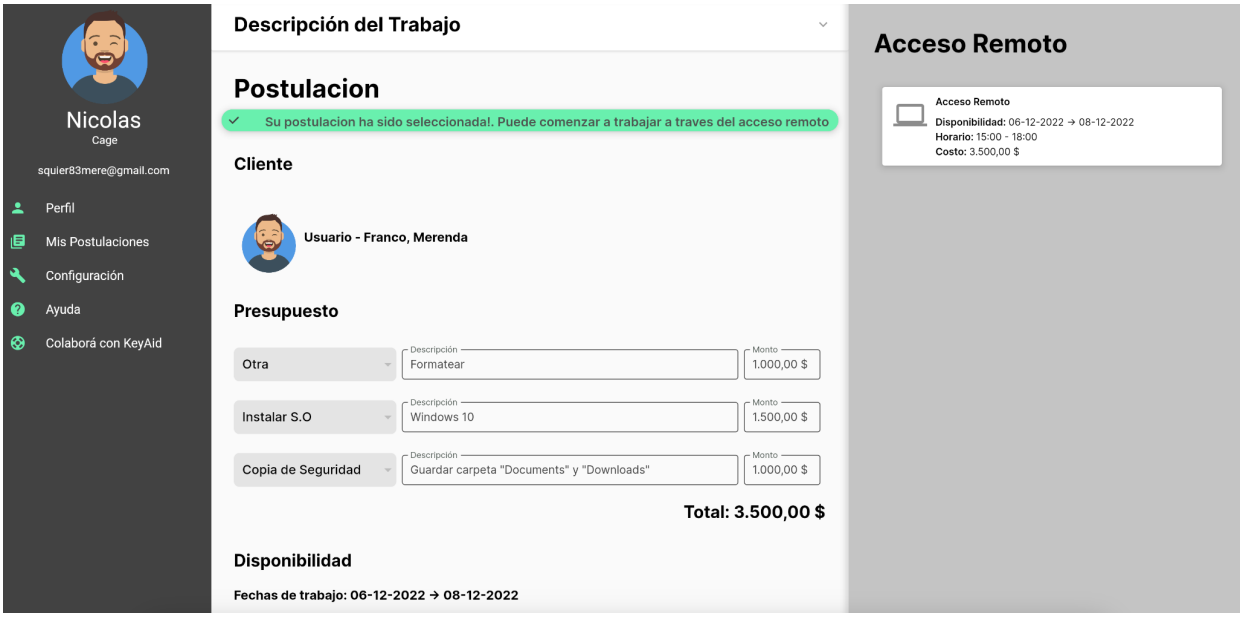

Figura 4.9. Pantalla de trabajo que fue confirmada por el *cliente*

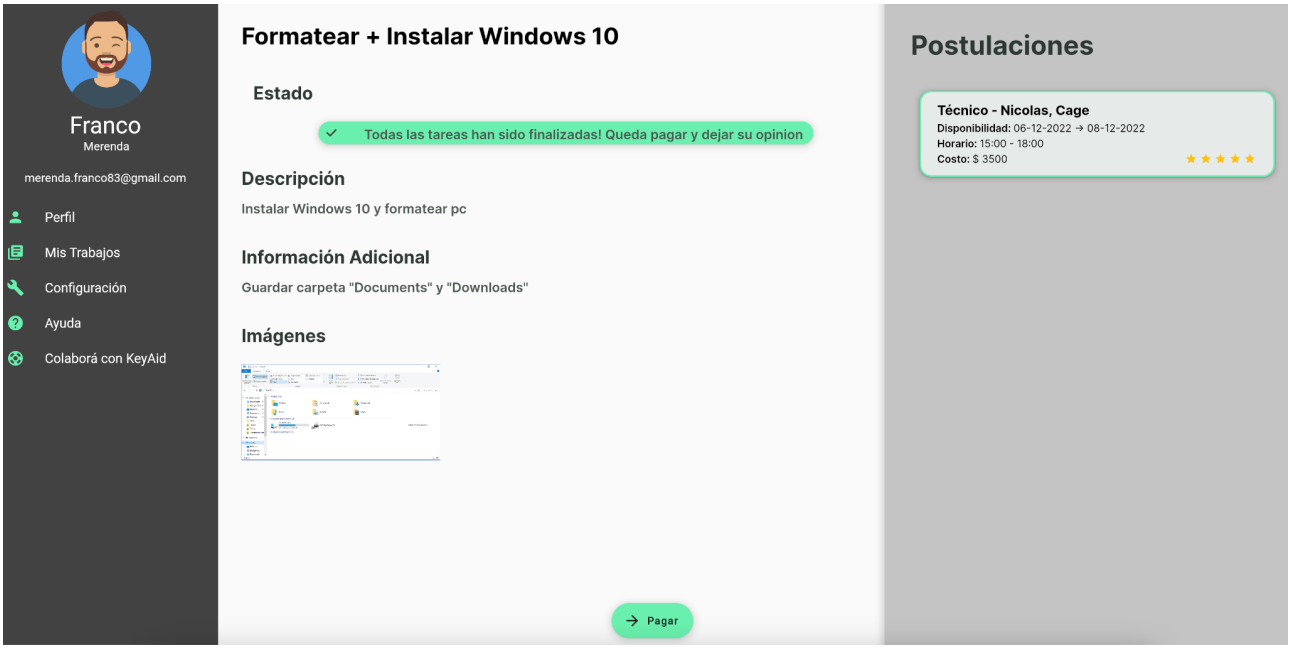

Figura 4.10. Pantalla de trabajo donde ya las tareas fueron realizadas por el *técnico.*

6. Una vez que el *técnico* marca el trabajo como finalizado, el *cliente* realiza el pago del trabajo a través de la plataforma y puede dejar una revisión para el *técnico*, para futuros trabajos. Ver Figura 4.11 y 4.12.

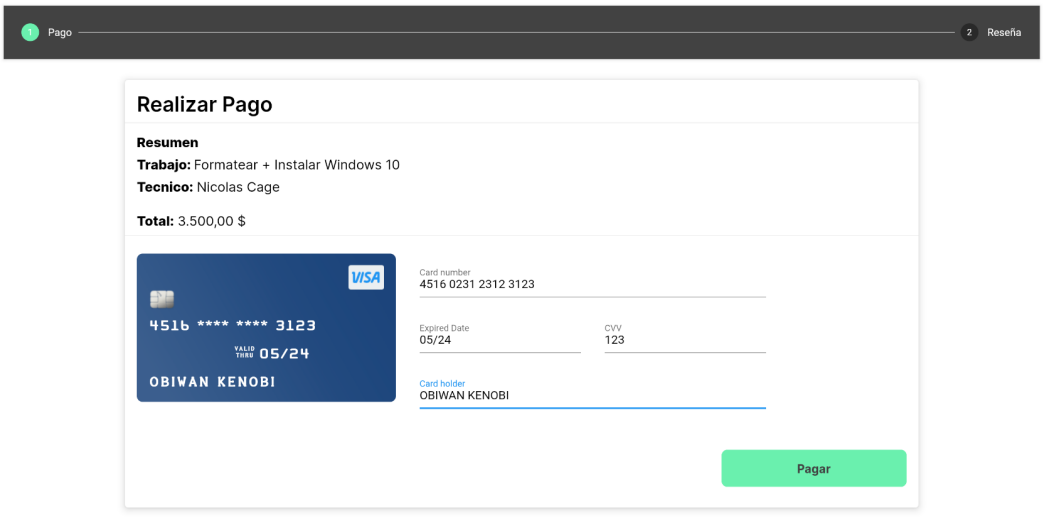

Figura 4.11. Pantalla de realización de pago por parte del cliente.

Figura 4.12. Pantalla de reseña para el técnico, luego de haber realizado el pago.

4.2.3 Escenario de Usuario técnico - Postular a un trabajo

En este escenario se muestra cómo un usuario *técnico* puede postularse para un trabajo en la plataforma y cómo es que accede remotamente a la máquina del usuario *cliente.*

1. El *técnico* ingresa a la plataforma.
2. Espera/revisa los distintos trabajos creados por los *clientes* en la plataforma. Ver Figura 4.13.

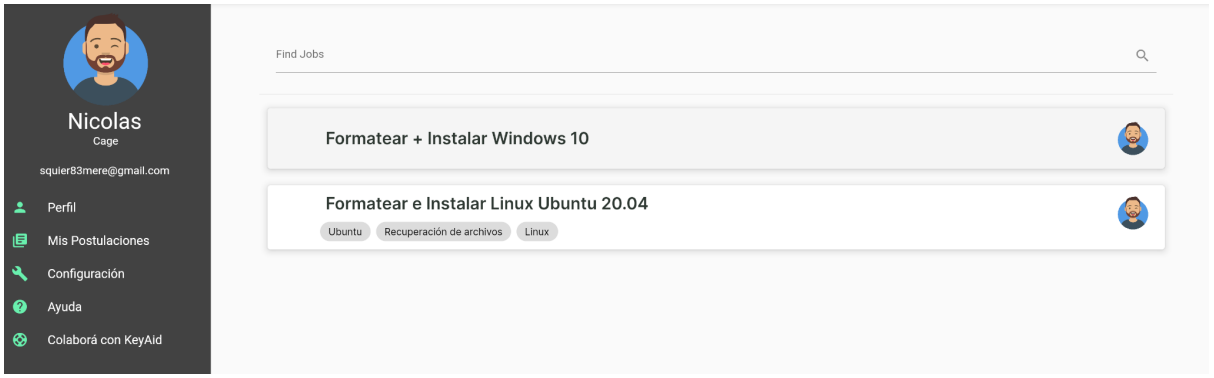

Figura 4.13. Ejemplo de pantalla de inicio de un usuario técnico.

3. Se postula a uno o más trabajos especificando tareas a realizar, costo total del trabajo y la disponibilidad en hora y días. Ver Figura 4.14.

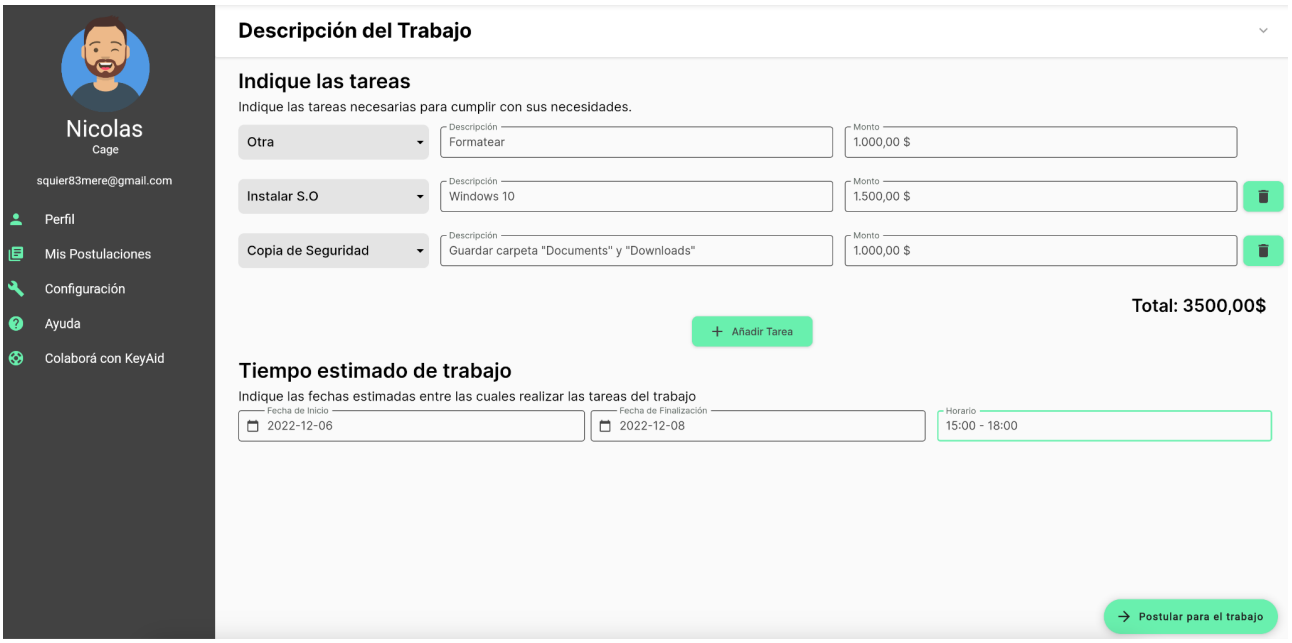

Figura 4.14. Ejemplo de *postulación* realizada por el *técnico* a un trabajo.

4. Espera a que una o más de las postulaciones realizadas sea aceptada por algún *cliente.* Ver Figura 4.15.

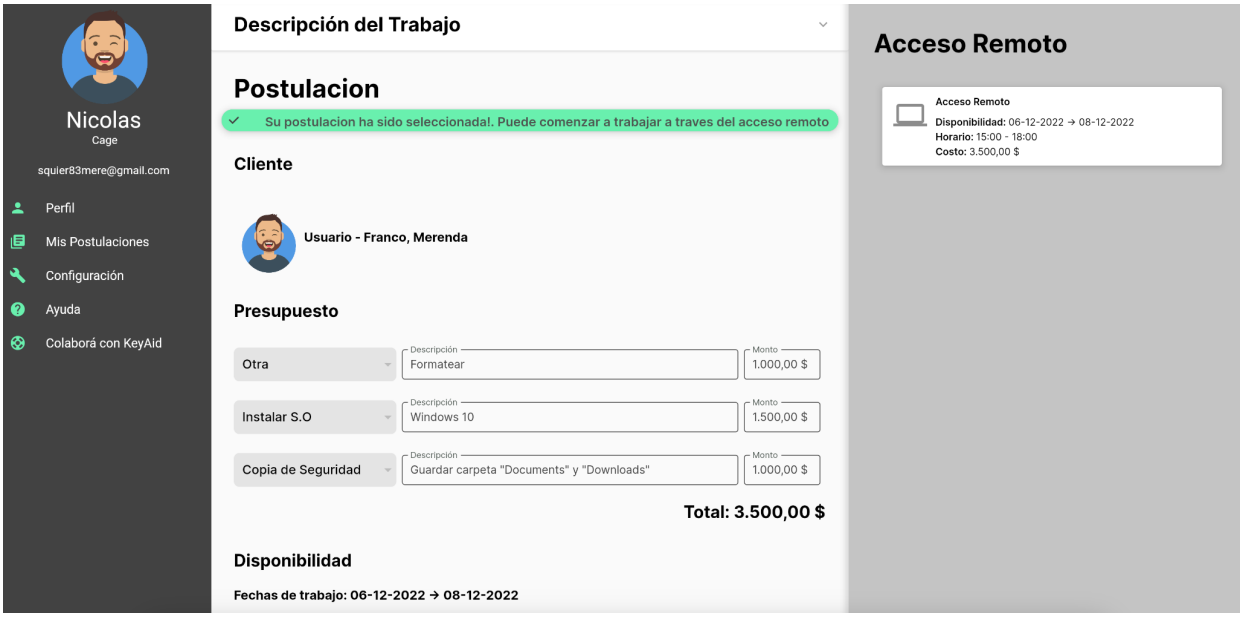

Figura 4.15. Postulación confirmada por el *cliente* con acceso remoto para comenzar a trabajar.

5. Acceder a la PC del usuario *cliente* a través de la plataforma para realizar el trabajo. Ver Figura 4.16 y Fig. 4.17.

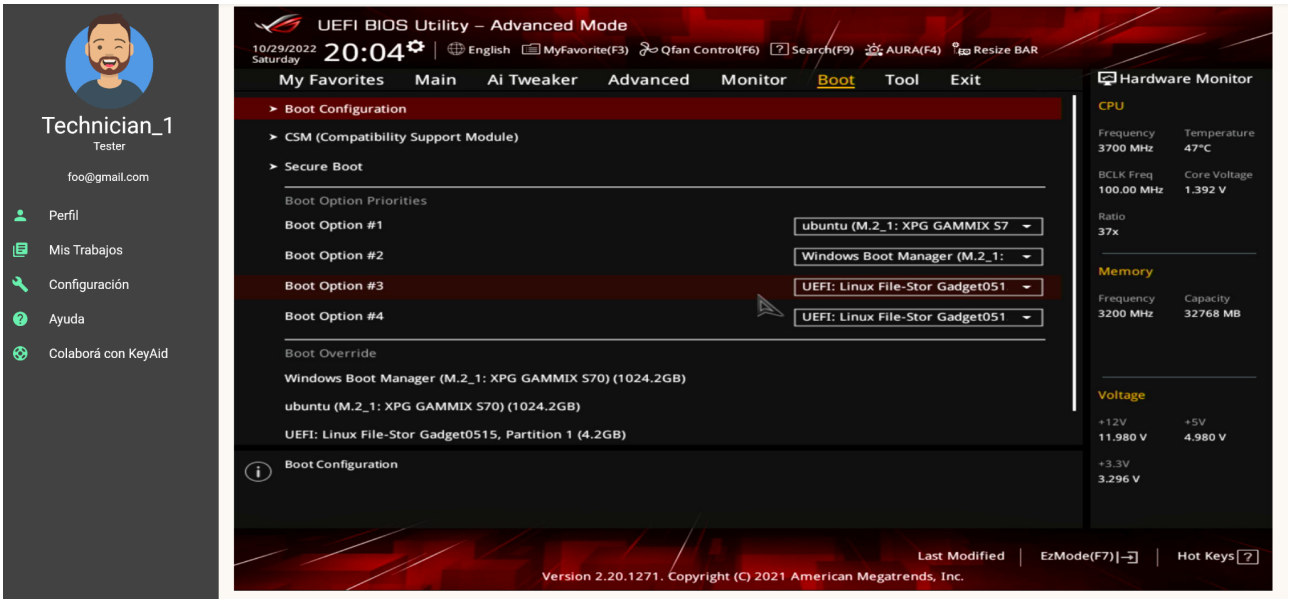

Figura 4.16. Pantalla mostrando la BIOS/UEFI del usuario *cliente*.

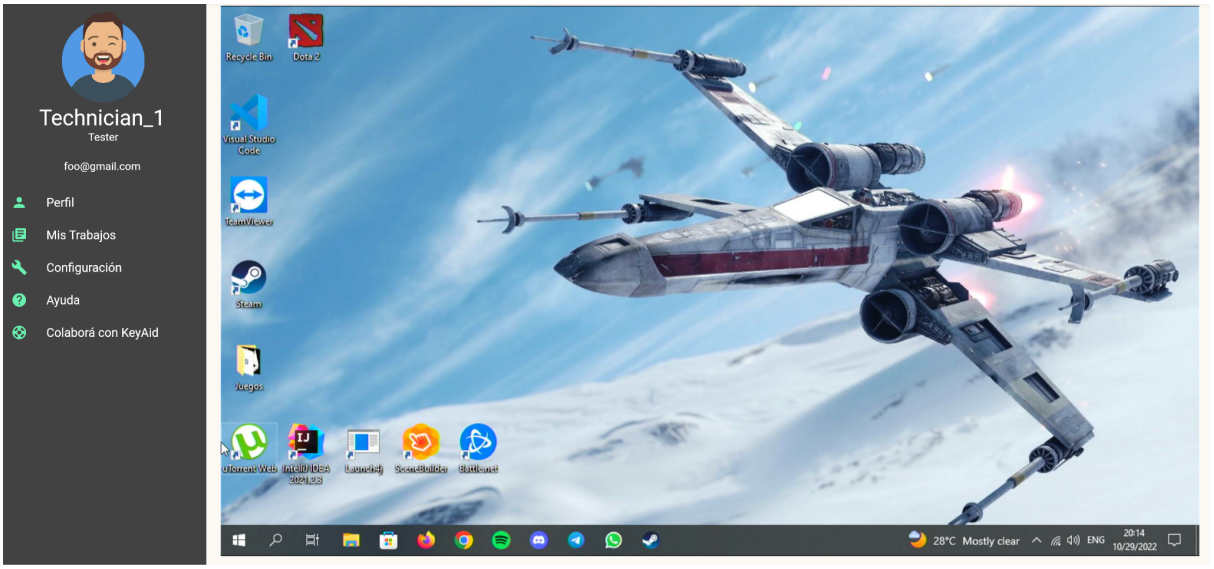

Figura 4.17. Pantalla mostrando el "Escritorio" del sistema operativo Windows 10 del *cliente.*

6. Confirmar que el trabajo ha finalizado, haciendo click en el Botón *Marcar trabajo como Finalizado*. Ver Figura 4.18.

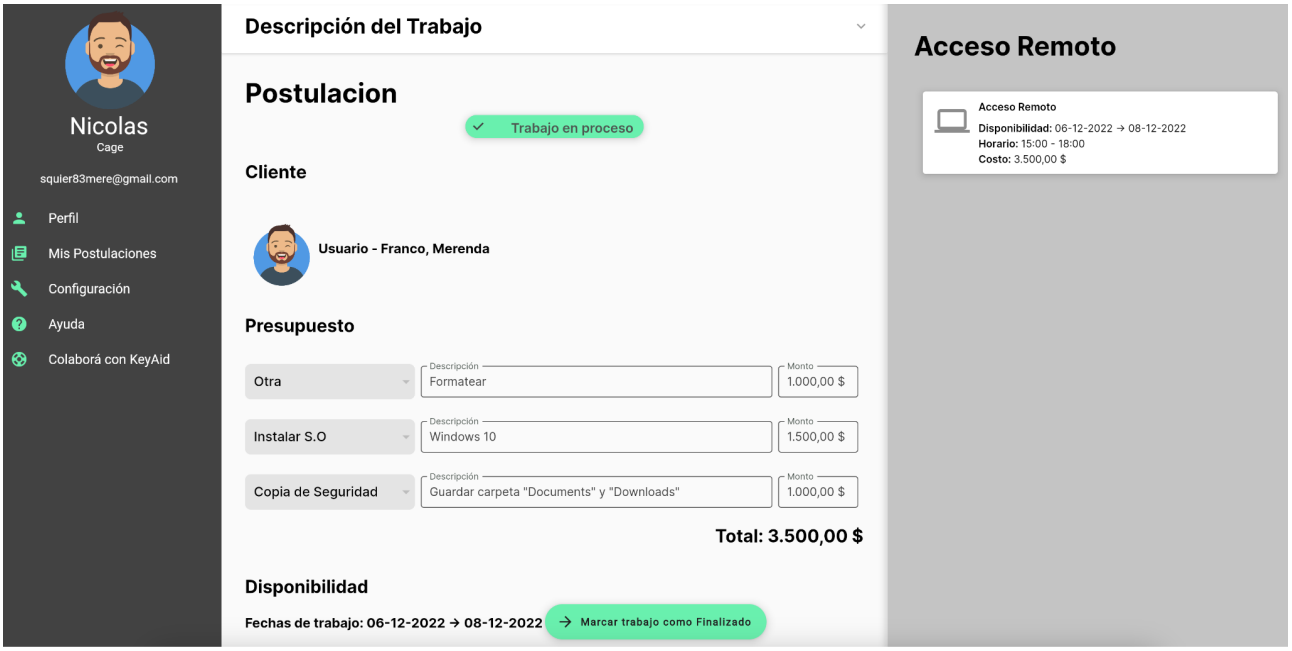

Figura 4.18. Pantalla con el botón para marcarle al usuario *cliente* que el trabajo ha sido finalizado.

7. Finalmente, queda a la espera por el pago realizado por el *cliente* y puede proceder a repetir desde el paso 2.

Capítulo V

Evaluación Experimental

Una parte importante en todo desarrollo son las evaluaciones, ya que éstas garantizan el funcionamiento correcto y esperado de cualquier desarrollo. En este caso particular, hemos considerado diferentes escenarios de experimentación a fin de mostrar distintas funcionalidades de KeyAid, principalmente de la interacción *usuario/cliente* con el *técnico*.

En este capítulo se describe la evaluación experimental en cada uno de los escenarios propuestos. Los escenarios se caracterizan por tener diferentes particularidades tanto de hardware como de software.

5.1. Características Generales de la Experimentación

Para realizar la experimentación se consideraron diferentes modelos de computadoras: PCs, Notebooks y Netbooks, con el objetivo de visualizar el funcionamiento de *KeyAid*, en cada una de ellas. En este caso la experimentación se basó principalmente en el acceso remoto y su funcionalidad respecto al control de cada uno de los equipos.

Para las pruebas se utilizaron dos computadoras como máquinas del *cliente* y la Raspberry Pi 4 - 4GB para realizar el control remoto de las dos.

Los criterios utilizados para evaluar la experimentación son Eventos de teclado, Eventos de mouse, Dispositivo de almacenamiento USB, todos reconocidos, Vídeo capturado antes de entrar al sistema operativo, y Vídeo capturado luego de entrar al sistema operativo. La razón de por qué evaluar cada uno de estos criterios está en:

- *Eventos de Teclado y Mouse:* A su vez, el hecho de definir si el sistema reconoce los eventos de teclado, mouse, es para poder determinar la compatibilidad con distintos tipos de hardware. En caso de que éstos los reciban, indica que es posible controlarlos remotamente, ya que el teclado y mouse, nos permiten movernos e ingresar datos al sistema.
- *Eventos de Almacenamiento USB:* Además de poder determinar la compatibilidad del hardware y la posibilidad de control remoto como en los casos del teclado y mouse, este tipo de almacenamiento USB nos permite, ya sea, almacenar datos o utilizarlo para instalar nuevos sistemas operativos, entre otros posibles usos.
- *Captura de Vídeo:* Si el vídeo capturado fue exitoso antes y después de entrar al sistema operativo, es porque depende de como inicializan los controladores de vídeo. Define por donde está direccionando el vídeo de salida, si por el puerto HDMI o si por la pantalla integrada en caso de Notebook/Netbooks.

En las siguientes secciones se detallan las diferentes pruebas realizadas en diversos dispositivos con diferentes sistemas operativos siguiendo el criterio descrito anteriormente junto a los resultados obtenidos.

5.2. Escenario 1 - PC de escritorio

A continuación, se detallan las especificaciones de los sistemas utilizados en la prueba.

Hardware utilizado para PC cliente:

- Procesador: AMD Ryzen 5600X - 6 nucleos @ 3.7GHz (64 bit).
- Tarjeta Gráfica: NVidia GeForce 3070 Ti.
- Memoria Ram: 32 GB - DDR4.
- Sistema Operativo: Windows 10 y Ubuntu 22.04.

Hardware utilizado para el control remoto de la PC del cliente:

- Raspberry Pi 4 Mod. B - 4GB.
- Procesador: Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz.
- Memoria Ram: 4GB LPDDR4-3200 SDRAM.
- 2 Puertos USB 3.0; 2 Puertos USB 2.0.
- 40 Pines de entrada/salida de propósito general.
- Conector USB-C 5V CC (minimo 3A).

Resultados:

- ✅ Eventos de teclado reconocidos con éxito
- ✅ Eventos de mouse reconocidos con éxito
- ✅ Dispositivo de almacenamiento USB reconocido con éxito
- ✅ Vídeo capturado antes de entrar al sistema operativo con éxito (ver Figura 5.1)
- ✅ Vídeo capturado luego de entrar al sistema operativo con éxito (ver Figura 5.2)

En la Figura 5.1 se puede visualizar un ejemplo del usuario *técnico* controlando de manera remota a la máquina del usuario *cliente* accediendo a configuraciones de la BIOS/UEFI, pudiendo

configurar parámetros del sistema como qué dispositivo usar para iniciar el sistema, configurar velocidades de ventiladores de acuerdo a la temperatura del sistema, frecuencias de la memoria o CPU, en caso que la BIOS/UEFI lo soporte, entre otras.

En la Figura 5.2, se puede visualizar un ejemplo de un usuario *técnico* accediendo al sistema del usuario *cliente.* Aquí el *técnico* ya puede instalar programas, configurar/instalar controladores para el sistema, entre otras.

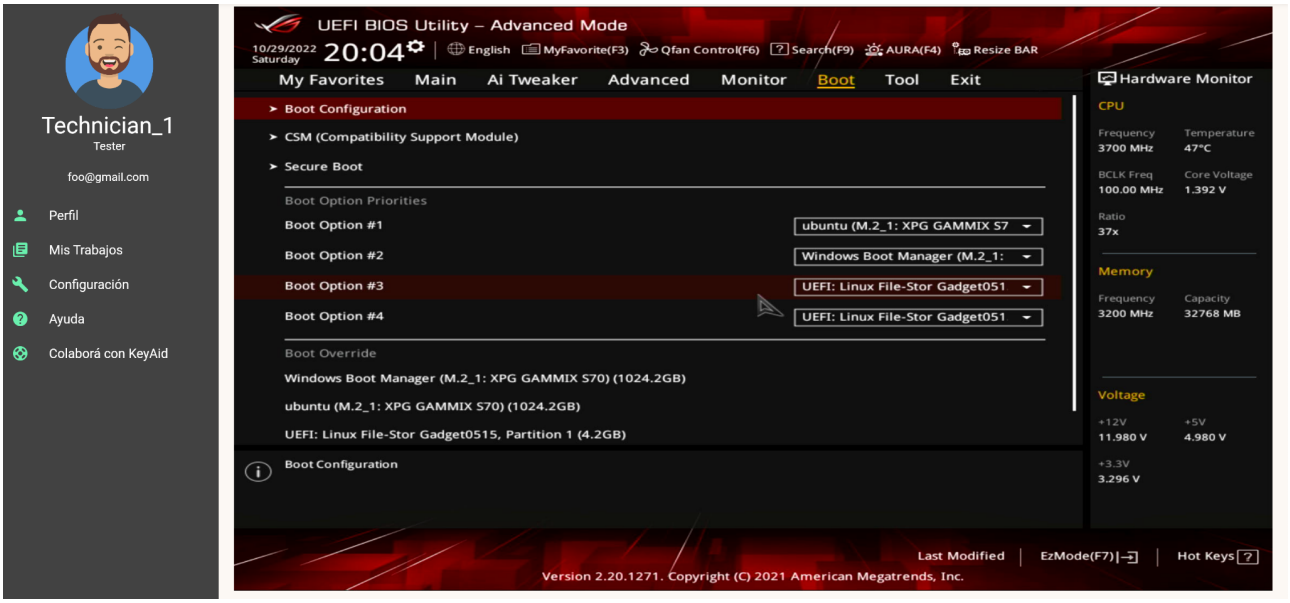

Figura 5.1. Pantalla mostrando la BIOS/UEFI del usuario *cliente.*

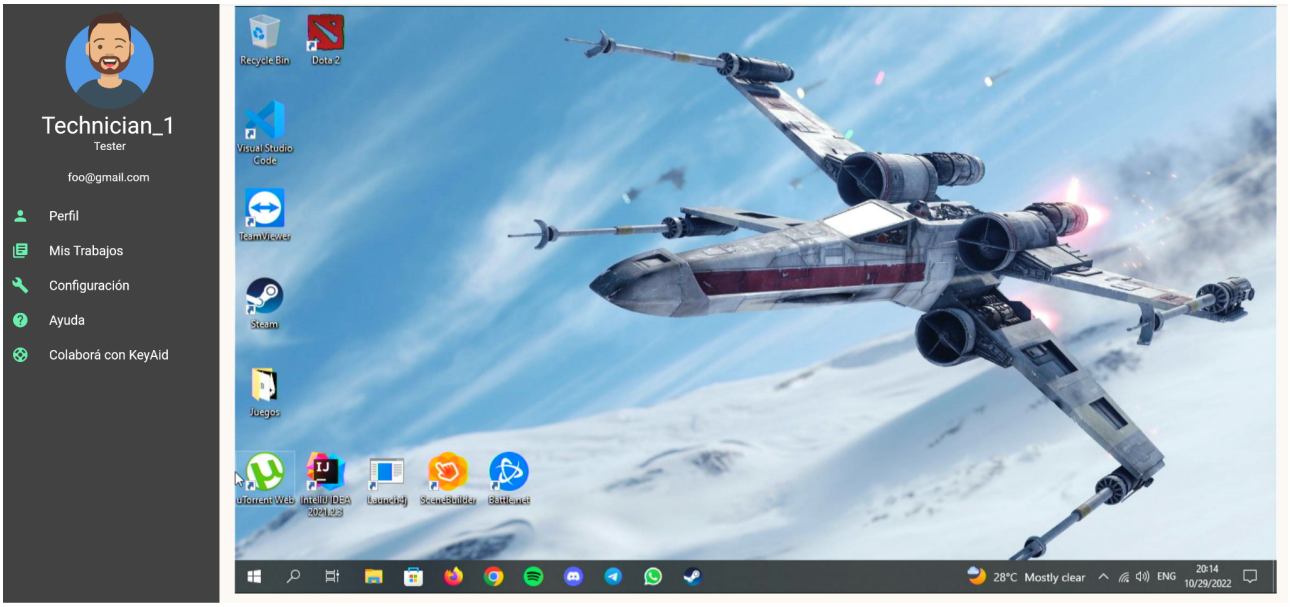

Figura 5.2. Pantalla mostrando el “Escritorio” del sistema operativo Windows 10 del *cliente.*

A partir de los resultados obtenidos para la PC de escritorio, se ve que el caso de uso es exitoso debido a que, la salida de vídeo se realiza por el puerto de vídeo de su tarjeta gráfica dedicada, o el de su tarjeta madre. Por lo que en todo momento, el *técnico* puede visualizar el vídeo del *cliente*.

5.3. Escenario 2 - Netbook

El hardware utilizado para el usuario *cliente* fue una netbook con las siguientes características:

- Procesador: Intel Atom N455 @ 1.66GHz
- Tarjeta Gráfica: Gráficos Integrados de Intel
- Memoria Ram: 2GB - DDR3
- Sistema Operativo: Debian 10 "*Buster*"

Hardware utilizado para el control de la netbook del usuario *cliente* de forma remota:

- Raspberry Pi 4 Mod. B - 4GB
- Procesador: Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
- Memoria Ram: 4GB LPDDR4-3200 SDRAM
- 2 Puertos USB 3.0; 2 Puertos USB 2.0
- 40 Pines de entrada/salida de propósito general.
- Conector USB-C 5V CC (minimo 3A)

Resultados:

- ✅ Eventos de teclado reconocidos con éxito
- ✅ Eventos de mouse reconocidos con éxito
- ✅ Dispositivo de almacenamiento USB reconocido con éxito
- ❌ Vídeo capturado antes de entrar al sistema operativo con éxito
- ✅ Vídeo capturado luego de entrar al sistema operativo con éxito

Como puede observarse, no se pudo capturar la señal del vídeo durante el inicio del sistema, sin embargo, este resultado no afecta ya que una de las posibles soluciones es forzar la salida del vídeo a través del puerto de salida de vídeo, lográndolo a través del cerrado de la tapa de la netbook. Otra posible solución para ello, es considerar el uso de una cámara externa para este tipo de dispositivos, obteniendo así la salida del vídeo en todo momento.

5.4. Escenario 3 - MacBook (Mod. 2019)

El hardware utilizado para el usuario *cliente* fue una macbook con las siguientes características:

- Procesador: Intel Core I9 - 8 núcleos @ 2.3 GHz
- Tarjeta Gráfica:
    - Integrada: Intel UHD Graphics 630
    - Dedicada: AMD Radeon Pro 5500M - 4GB GDDR6
- Memoria Ram: 16GB DDR4
- Sistema Operativo: MacOS Ventura 13.0

Hardware utilizado para el control remoto de la notebook del usuario *cliente*:

- Raspberry Pi 4 Mod. B - 4GB
- Procesador: Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
- Memoria Ram: 4GB LPDDR4-3200 SDRAM
- 2 Puertos USB 3.0; 2 Puertos USB 2.0
- 40 Pines de entrada/salida de propósito general.
- Conector USB-C 5V CC (mínimo 3A)

Resultados:

- ✅ Eventos de teclado reconocidos con éxito
- ✅ Eventos de mouse reconocidos con éxito
- ✅ Dispositivo de almacenamiento USB reconocido con éxito
- ❌ Vídeo capturado *antes* de entrar al sistema operativo con éxito
- ✅ Vídeo capturado *luego* de entrar al sistema operativo con éxito

Como puede observarse, aquí también no se pudo capturar la señal de vídeo durante el inicio del sistema, y las posibles soluciones al respecto se describieron en la prueba número 2. Lo que sí nos permite inferir, es que todo dispositivo que tenga un monitor integrado, aún con tarjetas gráficas dedicadas, tiene una gran probabilidad de tener este mismo comportamiento.

En las pruebas realizadas se observan los distintos puntos que se deben considerar a la hora de mejorar la experiencia del usuario, ya que, el proyecto está creado y diseñado con el propósito de permitir a usuarios resolver sus problemas técnicos con la mínima intervención posible por parte de ellos. Además, también visualizan lo que es posible de realizar desde el punto de vista del *técnico* de forma remota. En el próximo capítulo, se explica en mayor detalle el uso de la plataforma web y cómo es que el técnico accede de forma remota desde la misma.

Capítulo VI

Conclusiones y Trabajos Futuros

A partir de todo lo expuesto en el proyecto y los resultados de las pruebas en el capítulo IV, se puede considerar que el proyecto tiene la posibilidad de ser un competidor a las soluciones actuales.

A continuación, en la presente sección se presentan las conclusiones obtenidas luego de haber realizado el proyecto, y posibilidades de trabajos futuros qué podrían ayudar a que el proyecto pueda seguir creciendo.

6.1. Conclusiones

En este trabajo presentamos una herramienta, *KeyAid*, que vincula personas que necesitan soluciones en el dominio I.T. con sus dispositivos (*Clientes)*, y personas que tienen conocimientos y habilidades para poder resolverlos (*Técnicos*). El trabajo implicó el desarrollo de diferentes módulos, tales como la Plataforma Web, la modificación a *TinyPilot* para proveer el almacenamiento USB, y garantizar la conexión segura entre la Raspberry Pi y la máquina del *cliente*.

A *KeyAid* se le puede evaluar considerando dos aspectos principales: la facilidad de acceso para el usuario y su viabilidad económica, por ello la conclusión se enfoca en:

1. *Experiencia del usuario*
2. *Costo de acceso a la plataforma*

En cuanto a la *experiencia de usuario*, el proyecto presenta una interfaz intuitiva para la interacción entre *técnicos* y *clientes*, intentando replicar y mejorar como sería la interacción persona a persona, con la ventaja de poder documentar todo el proceso y que ambos tengan transparencia del mismo. Desde el punto de vista del *cliente*, en todo momento tiene en su poder su dispositivo con su información y el costo final que pagará por el trabajo realizado. Desde el punto vista del *técnico*, no tiene que preocuparse por realizar el recibo del pago, y disminuir el tiempo de atención al cliente, lo cual implica que podrá concentrarse en realizar sus tareas, mientras la plataforma realiza la gestión de sus trabajos.

Inclusive, el hecho de poder emular distintos dispositivos USB junto a tener disponibles pines de entrada/salida digitales, permitirá extender las funcionalidades del dispositivo *KeyAid*.

A medida que el proyecto alcance un mayor número de personas, el mismo podrá contar con más información respecto a las necesidades de los usuarios, permitiendo ir mejorando aún más la experiencia poco a poco.

En lo que respecta a los costos de la plataforma, el costo actual para adquirir el equipamiento necesario para poder acceder a *KeyAid*, implica una Raspberry Pi 4, su fuente de alimentación, la carcasa y los cables para las conexiones, capturadora de vídeo USB, junto a una placa derivadora de datos y alimentación (ver Figura 3.4).

El bajo costo de acceso a la plataforma brinda la posibilidad de contar con una amplia oferta de posibles *técnicos* a realizar el trabajo, y una importante reducción de los tiempos productivos, tanto de *clientes* como *técnicos*. Además, ofrece la posibilidad de generación de trabajo y de acceso a personal altamente calificado, independientemente de la ubicación geográfica.

Es posible reducir aún más los costos considerando alternativas del mercado, como por ejemplo, contar con agencias de alquiler, las cuales ofrecerían el equipamiento necesario para acceder a *KeyAid*. Se podría pensar en un "delivery" de dispositivos *KeyAid*, el cual le entregaría al usuario solicitante del servicio, un dispositivo *KeyAid (*Raspberry Pi configurada*)*, y, posteriormente, realizará el cobro por los días de uso. Estas alternativas, permitirán acceder a la plataforma de una forma muy económica sin tener que moverse de su casa ni esperar a que un técnico vaya a resolverle el problema.

Otra oportunidad para la reducción del costo es diseñar un hardware específico para dicho propósito a fin de minimizar los requerimientos y que se ajuste a las necesidades específicas.

Por todo lo expuesto, se puede afirmar que *KeyAid* tiene el potencial de competir con soluciones actuales. A su vez, visualiza las oportunidades que se pueden generar a partir del mismo, así como, construir una nueva forma de realizar el mantenimiento a los sistemas.

Este proyecto presentó una oportunidad para realizar un análisis de su viabilidad, definir plazos, objetivos y las funcionalidades del producto a entregar por cada etapa. Por cada iteración realizada a medida que fue desarrollado, nos permitió ir sumando nuevas funcionalidades y/o revisando las ya implementadas. Finalmente quedan muchas líneas abiertas para su mejora futura, algunas de las cuales detallamos en la próxima sección.

6.2. Trabajos Futuros

En esta sección, se dedicará a detallar posibilidades de trabajos futuros que podrían complementar a este proyecto, para lograr, ya sea, cubrir nuevas funcionalidades que mejoren la experiencia para el usuario o generar nuevas oportunidades:

- *Desarrollo de conector para encendido/apagado:*

  Actualmente el proyecto no tiene soporte para encender/apagar el computador de forma remota. Lo que se propone como trabajo futuro al respecto es que, dado que la Raspberry Pi tiene pines de entrada/salida digitales, es posible construir un mecanismo que se ubique sobre el botón de encendido de la máquina del cliente, y que su accionar haga presionar ese botón o no de forma remota.

  Algunas desventajas de este modelo, es que es impráctico para el usuario cliente ya que, aumentaría el costo del dispositivo y empeora la experiencia del usuario, porque implica que el cliente tenga que ubicar de manera correcta el mecanismo sobre el botón de su equipo.

  Otra alternativa es, que desde la tarjeta madre, el switch interno para encender/apagar la máquina (PWR ON), se exponga al exterior, de manera tal que el dispositivo *KeyAid* pudiera hacer uso de éste. En este caso, el usuario sólo verá un conector más, con el cual el dispositivo *KeyAid* sería compatible y evitaremos tener mecanismos físicos interactuando con el sistema con mayor posibilidad de fallar.

- *Solución temporaria para la salida de vídeo en notebooks/netbooks*

  A partir de las pruebas realizadas en el capítulo IV, se observó que para las notebooks/netbooks existe el problema de no poder acceder al vídeo antes de que el sistema operativo sea cargado por la misma. Una de las posibilidades para poder solucionar esto es encender el dispositivo y cerrar la tapa para forzarlo a que la salida de vídeo se produzca a través de sus salidas de vídeo externas (HDMI, VGA, DisplayPort), dependiendo la notebook/netbook.

  Otra solución alternativa, sería ofrecer al dispositivo *KeyAid* con dos opciones:

  A. Para el caso de los usuarios que tengan máquinas de escritorio, ofrecerlo con la capturadora de vídeo.

B. Para el caso de los usuarios con netbooks/notebooks, ofrecerlo con una cámara externa que se pueda ubicar en la notebook/netbook apuntando a la pantalla de ésta, logrando así conseguir el vídeo en cualquier momento.

Una de las desventajas de ofrecerlo de esta forma, es que, remueve un poco la generalidad de que el proyecto pueda funcionar en cualquier dispositivo, y requiere que el usuario tenga que posicionar la cámara de forma adecuada, de manera tal que el técnico tenga una buena visualización del vídeo de salida.

- *Chat dentro de la plataforma web*

  Para mejorar la comunicación entre el *técnico* y el *cliente*, se propone a futuro añadir un chat dentro de la plataforma web, de forma que ambos puedan interactuar de forma instantánea ya sea para discutir cualquier posible inconveniente ante una tarea en particular o para hablar sobre algún detalle en específico que el cliente o técnico requieran uno del otro.

- *Soporte para pagos utilizando criptomonedas*

  Hoy en día, se ve cada vez más el uso de criptomonedas en la sociedad como almacén de valor. Existen soluciones que permiten realizar compras de viajes, productos, entre otros, utilizando como método de pago las criptomonedas. Una de las principales ventajas del uso de éste tipo de método de pago, es permitir independizarse de la moneda local de un país, por lo que se vuelve una forma de pago global, donde desde cualquier parte del mundo uno puede pagar por ese servicio/producto.

  Por ello, es que se contempla a futuro agregar como forma de pagos algunas de las criptomonedas actuales como Bitcoin, Ethereum, o también, en criptomonedas basadas en monedas estables tales como USDC, DAI, etc, las cuales están basadas en el valor actual del dólar.

- *Automatización de tareas desde el dispositivo*

  Una posibilidad que el dispositivo *KeyAid* podría ofrecer, es que, para el caso de máquinas recién compradas o que quisieran una instalación desde cero, se podrían automatizar los procesos de instalación de sistemas operativos sin necesidad de un técnico intermediario. A través de análisis de imágenes del vídeo, se puede construir un flujo automatizado que realice la instalación de forma automática para distintos tipos de sistema operativos.

- *Solución para empresas*

  Otra oportunidad que se presenta, es que *KeyAid* podría ofrecer una solución para empresas, donde internamente se pueda tener acceso a los diferentes dispositivos desde una única ubicación, haciendo la tarea del servicio técnico mucho más rápida y efectiva, sin necesidad de realizar ningún movimiento físico del equipo.

- *Mobile App para configurar dispositivo KeyAid*

  Dado que la Raspberry Pi consta de conexión vía bluetooth, es posible realizar una aplicación para que el usuario cliente pueda realizar configuraciones, tales como, configuraciones de red, ver el estado de la plataforma, entre otras. Esto es necesario para lograr mayor transparencia para el usuario cliente y lograr dar versatilidad en caso que la conexión a la red por cable no fuese posible.

Bibliografía

[1]. Teamviewer documentation, *"TeamViewer: The Remote Desktop Software"*, https://www.teamviewer.com/en-us/.

[2]. AnyDesk, *"La aplicación de escritorio remoto rápido – AnyDesk."* https://anydesk.com/es.

[3]. PiKVM documentation, *"PiKVM - Open and cheap DIY IP-KVM on Raspberry Pi"*, https://pikvm.org.

[4]. Lynch, Michael, *"tiny-pilot/tinypilot: Use your Raspberry Pi as a browser-based KVM." GitHub, https://github.com/tiny-pilot/tinypilot.*

[5]. Workana, *"Find Freelancers & Freelance Jobs Online."*, https://www.workana.com/en.

[6]. Upwork documentation, *"Upwork | The World's Work Marketplace"*, https://www.upwork.com/.

[7]. Fiverr documentation, *"Fiverr - Freelance Services Marketplace"*, https://www.fiverr.com/.

[8]. Raspberry-Pi documentation, *"Raspberry-Pi 4 Model B - Raspberry Pi"*, https://www.raspberrypi.com/products/raspberry-pi-4-model-b/.

[9]. Flutter documentation, *"Flutter - Build apps for any screen"*, https://flutter.dev/.

[10]. Moroney, Laurence, *"The Definitive Guide to Firebase: Build Android Apps on Google's Mobile Platform, 1 Ed."*, 978-1484229422, Apress, 12 Noviembre 2017.

[11]. S. Faust, *"Using Google´s Flutter Framework for the Development of a Large-Scale Reference Application", Bachelor Thesis, https://nbn-resolving.org/urn:nbn:de:hbz:832-epub4-14989.*

[12]. Biessek, Alessandro, *"Flutter for Beginners: An Introductory Guide to Building Cross-platform Mobile Applications with Flutter and Dart 2"*, Packt Publishing, 12 Septiembre 2019.

[13]. *µStreamer* documentation, *"pikvm/ustreamer: µStreamer - Lightweight and fast MJPEG-HTTP streamer."* GitHub, https://github.com/pikvm/ustreamer

[14]. Linux Kernel, *"1.1. Introduction — The Linux Kernel documentation."* The Linux Kernel Archives, https://www.kernel.org/doc/html/v4.9/media/kapi/v4l2-intro.html.

[15]. Flask Documentation, *"Welcome to Flask — Flask Documentation (2.1.x)"*, https://flask.palletsprojects.com/en/2.1.x/.

[16]. Grinberg, Miguel, "*Flask Web Development: Developing Web Applications with Python, 2da Edicion*", 978-1491991732, O'Reilly Media, 24 Abril 2018.

[17]. Grinberg, Miguel. "*flask-socketio Documentation*", https://buildmedia.readthedocs.org/media/pdf/flask-socketio/stable/flask-socketio.pdf.

[18]. Socket documentation, "*Socket.IO*", https://socket.io/.

[19]. Ventoy, "*Get Started, Ventoy*", https://www.ventoy.net/en/doc_start.html.

[20]. Flutter documentation, *"Flutter architectural overview | Flutter."*, https://docs.flutter.dev/resources/architectural-overview#from-user-input-to-the-gpu.

[21]. Skia documentation, "*Skia*", https://skia.org/.

[22]. Xu, Hao, et al, *"A Framework to Run C/C++ Application on Web-Based OS."*, 2021 14th International Conference on Advanced Computer Theory and Engineering (ICACTE), 2021, https://doi.org/10.1109/ICACTE53799.2021.00020.

[23]. Dart documentation, *"Sound null safety."* , https://dart.dev/null-safety.

[24]. Linux Kernel, *"Part I - Video for Linux API — The Linux Kernel documentation."*, The Linux Kernel Archives, https://www.linuxtv.org/downloads/v4l-dvb-apis-new/userspace-api/v4l/v4l2.html.

[25]. Linux Kernel, *"Introduction The Linux Kernel documentation."*, LinuxTV.org, https://www.linuxtv.org/downloads/v4l-dvb-apis-new/userspace-api/intro.html

[26]. Werkzeug Documentation, *"Werkzeug — Werkzeug Documentation (2.1.x)"*, https://werkzeug.palletsprojects.com/en/2.1.x/.

[27]. Ronacher, Armin, *"Jinja2 documentation - Welcome to Jinja2—Jinja2 Documentation (2.8-dev)"*, 2008.

[28]. Scratch Documentation, *"Scratch - Imagine, Program, Share"*, https://scratch.mit.edu/.

[29]. Python Documentation, *"Welcome to Python.org"*, https://www.python.org/.

[30]. Raspberry Shake, *"Raspberry Shake: Earthquake & Earth Monitoring Solutions"*, https://raspberryshake.org/.

[31]. Raspberry Pi, *"What is a Raspberry Pi?"*, https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/.

[32]. USB.org, *"On-The-Go and Embedded Host Supplement to the USB Revision 3.0 Specification"*, 10 de Mayo 2012. https://www.usb.org/sites/default/files/documents/usb_otg_and_eh_3-0_release_1_1_10may2012.pdf.

[33]. Linux Kernel, *"Configfs - Userspace-driven Kernel Object Configuration — The Linux Kernel documentation."*, The Linux Kernel Archives, 31 de Marzo, 2005, https://www.kernel.org/doc/html/latest/filesystems/configfs.html.

[34]. Linux Kernel, Brownell, David. *"USB Gadget API for Linux — The Linux Kernel documentation."*, The Linux Kernel Archives, https://www.kernel.org/doc/html/latest/driver-api/usb/gadget.html.

[35]. Linux Kernel, *"sysfs - _The_ filesystem for exporting kernel objects — The Linux Kernel documentation."*, The Linux Kernel Archives, https://www.kernel.org/doc/html/latest/filesystems/sysfs.html.

[36]. Linux Kernel, *"Linux USB gadget configured through configfs — The Linux Kernel documentation."*, The Linux Kernel documentation, 25 de Abril, 2013, https://www.kernel.org/doc/html/latest/usb/gadget_configfs.html.

[37]. Firebase, *"Cloud Storage for Firebase | Store and serve content with ease."*, https://firebase.google.com/products/storage.

[38]. Firebase, *"Cloud Firestore | Store and sync app data at global scale."*, https://firebase.google.com/products/firestore.

[39]. Firebase, *"Firebase Authentication | Simple, no-cost multi-platform sign-in."*, https://firebase.google.com/products/auth.

[40]. Hauck, Lane, *"Isolating USB"*. EDN Magazine, Julio 2006.

[41]. Tailscale, "Tailscale", https://tailscale.com/.

[42]. MercadoPago, “Mercado Pago”, https://www.mercadopago.com.ar/

Anexo I

Modelos de KeyAid

A1.1. Modelo de Dominio

El modelo de dominio de la Figura A1.1, muestra las clases conceptuales: *técnicos, clientes, trabajos, postulaciones, tareas, skills*, entre otras, y la relación entre ellas, las cuales serán utilizadas en la plataforma. Este modelo fue utilizado como base para realizar la implementación de dicha plataforma.

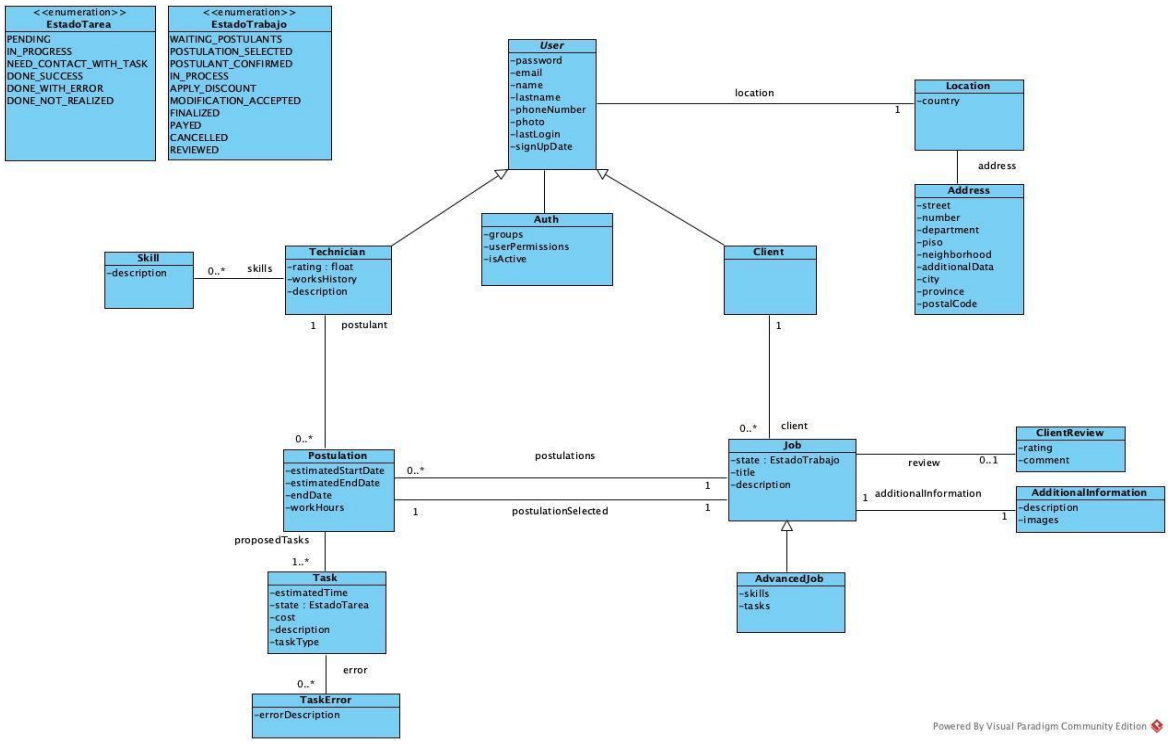

Figura A1.1. Modelo de Dominio - *KeyAid* (Usuarios/Trabajos/Postulaciones)

A1.2. Modelo de Actividad

El modelo de actividad de la Figura A2.2, visualiza el proceso desde que el trabajo es creado por el *cliente,* hasta que es finalizado por el *técnico* seleccionado. Este modelo ayuda a comprender cómo es el proceso al solicitar un trabajo.

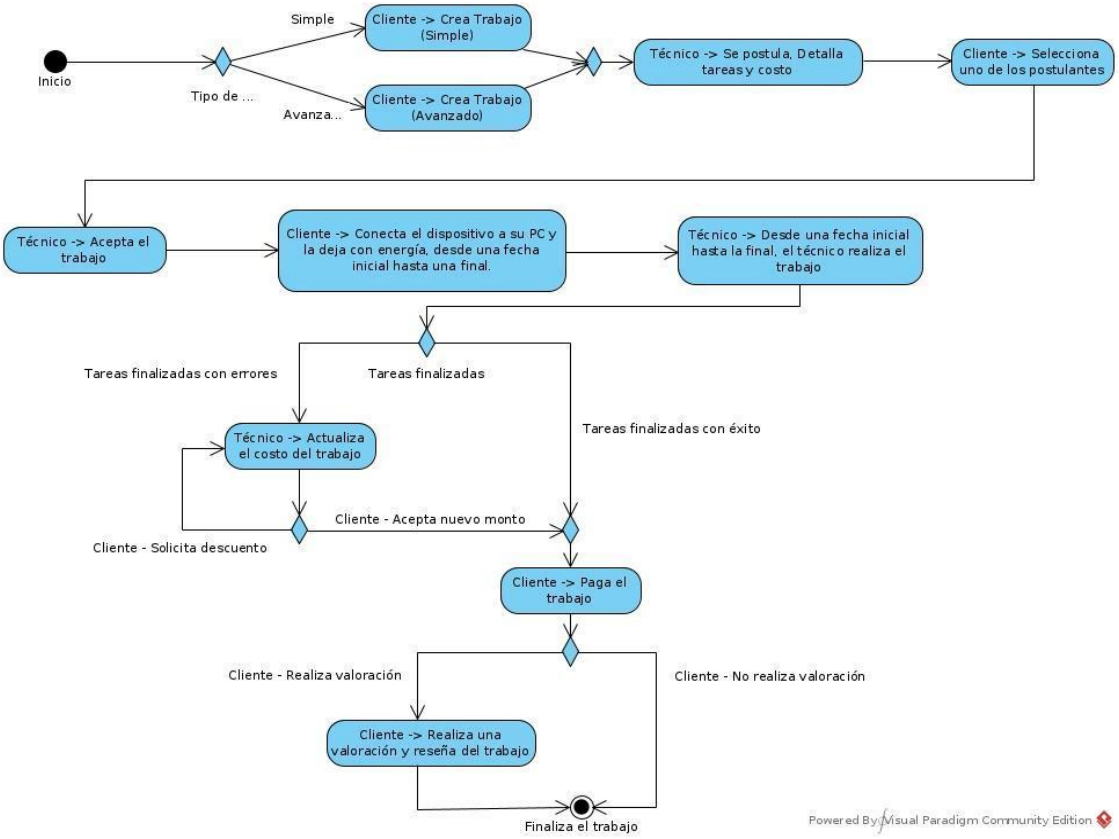

Figura A1.2. Modelo de Actividad - Proceso - Solicitar un trabajo

Anexo II

Manual de Usuario

A2.1 ¿Qué es y cómo funciona KeyAid?

KeyAid es una plataforma web que vincula Técnicos de IT con personas que necesiten asistencia con su dispositivo, ya sea, desde un problema como un programa hasta la reinstalación del sistema operativo o con alguna otra falla técnica.

Surgió de la necesidad de poder realizar el mantenimiento remoto a las PCs/Servidores que hoy en día no es posible realizar a menos que sea en persona. Si bien existen herramientas actuales para control remoto de los computadores/servidores, todas necesitan de un sistema operativo previamente instalado.

KeyAid logra el control remoto de las PCs/Servidores utilizando un dispositivo externo “KeyAid Device”, que permite desde la configuración de BIOS/UEFI, instalar sistemas operativos, hasta formatear el disco, todo vía remoto.

A medida que el proyecto siga creciendo y avanzando, nuevas funcionalidades se irán agregando, tales como, el apagado y prendido de forma remota, descarga de sistema operativos, backup de datos en la nube, entre otras.

A2.2. ¿Qué se necesita para acceder a la plataforma?

Para acceder a la plataforma, se requiere el dispositivo KeyAid en caso de ser *cliente* únicamente. En caso de ser técnico, sólo se necesita registrarse en la plataforma y tener conocimientos en IT.

Requerimientos para el *Cliente:*

- Paso 1: Registrarse en la plataforma como *Cliente*
- Paso 2: Adquirir o alquilar, si fuera posible, el dispositivo KeyAid para conectarlo a su PC/Servidor.

Requerimientos para el *Técnico*:

Si el usuario es un técnico en IT dispuesto a resolver problemas a la comunidad de KeyAid, lo único que va a necesitar es:

- Registrarse en la plataforma como *Técnico*.

A2.3. ¿Cómo realizar el registro en la plataforma?

En KeyAid existen dos tipos de usuarios, *Técnicos* y *Clientes* (Ver Figura A2.1):

- ***Técnicos***: Usuarios con experiencia en el mundo IT que arreglarán problemas o realizarán mantenimiento a los PCs/Servidores de los *Clientes*.
- ***Clientes***: Usuarios que necesitan algún arreglo o mantenimiento por parte de alguno de los/las *técnicos* en la plataforma.

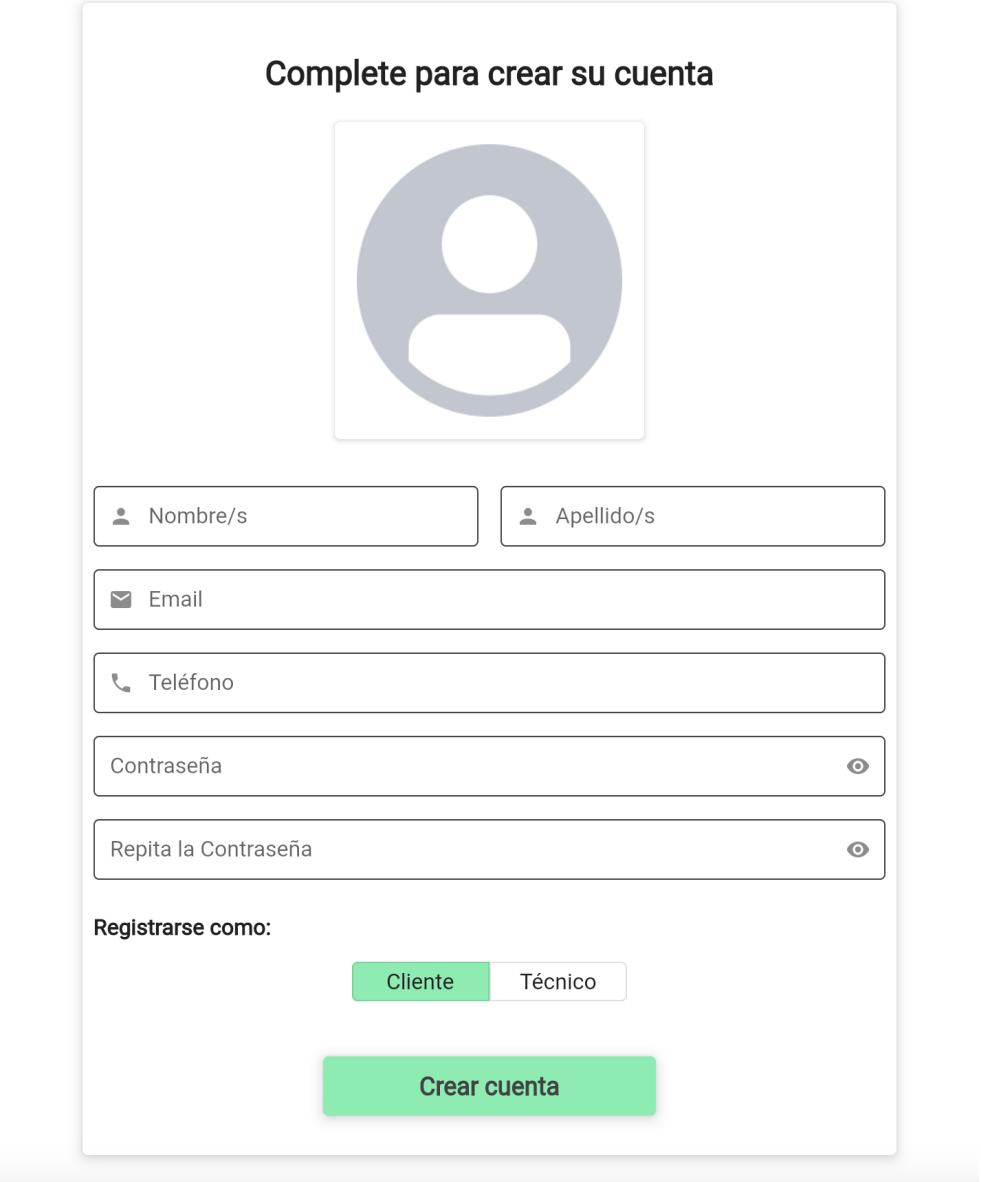

Figura A2.1 - Página de Registro

A2.4. Preguntas referidas a Usuarios Clientes

A2.4.1. El *Cliente* tiene un problema en su dispositivo, ¿Qué pasos debe seguir?

En caso de que el *cliente* tuviera un problema/error con su dispositivo, los pasos a seguir son los siguientes:

1. Ingresar a la plataforma KeyAid con el usuario con el que se registró en la plataforma.
2. Click en *"Postear un trabajo"* (Ver Figura A2.2)

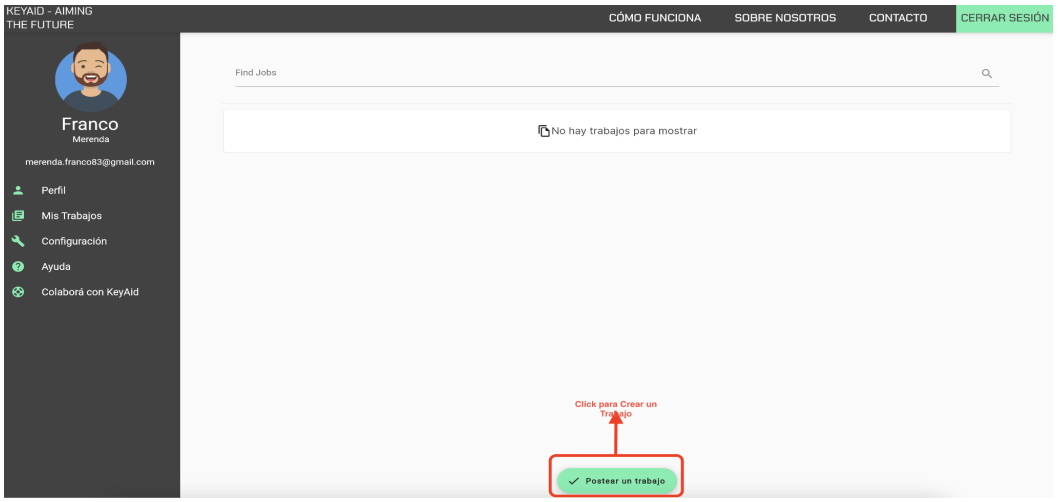

Figura A2.2 - Pantalla de Inicio - Botón para crear trabajo

3. Elegir tipo de trabajo a crear (ver Figura A2.3):
   A. En caso que sea el primer trabajo a crear en la plataforma, seleccione la opción **"Soy Novato/a"**. (Ver Figura A2.4)
   B. En caso que el usuario ya tenga un mayor conocimiento acerca de lo que necesita realizarle a su PC, seleccione **"Tengo conocimientos (Avanzado)"**. Útil para situaciones donde se desea tercerizar trabajos para otros técnicos con mayor experiencia. (Ver Figura A2.5)

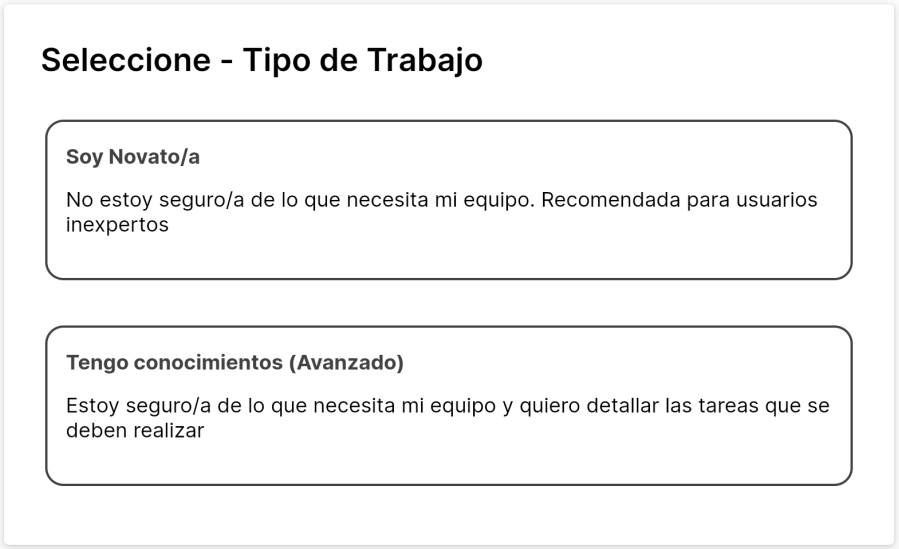

Figura A2.3 - Tipos de trabajo a crear - Soy Novato/a (Simple) o Tengo Conocimientos (Avanzado)

4. Siga los pasos indicados por la plataforma y complete la información solicitada.

- Pasos “**Soy Novato/a**”:

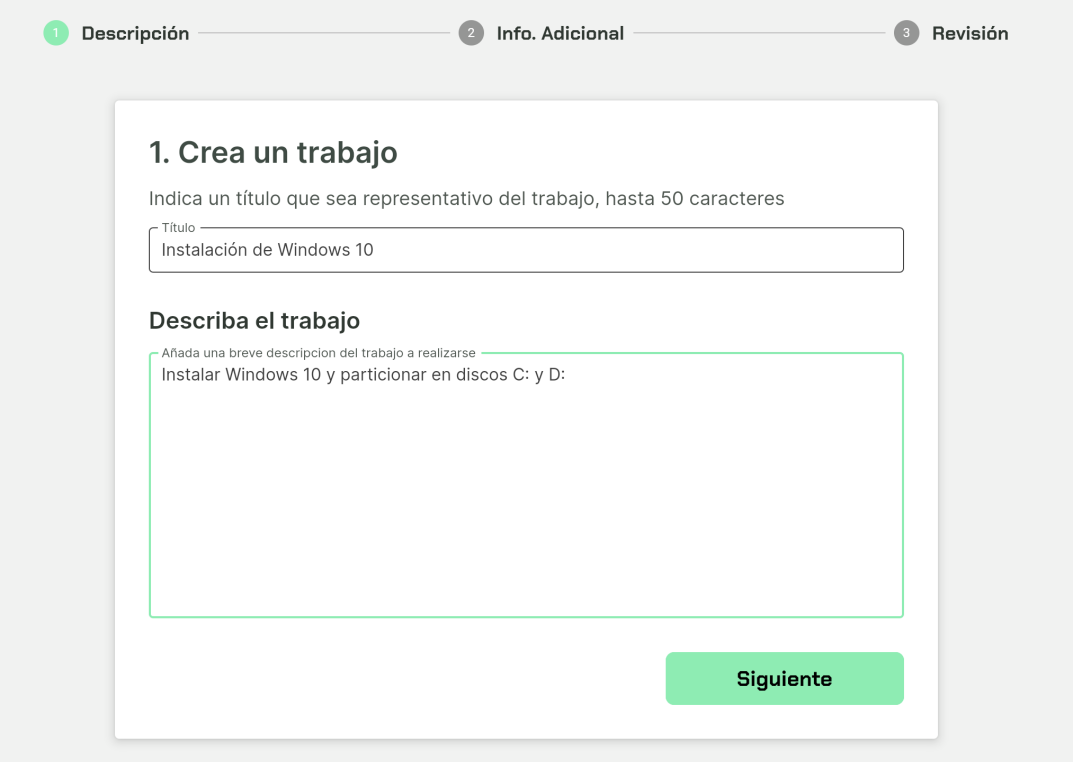

Figura A2.4 - Opción “Soy Novato/a (Simple)” - Paso 1.

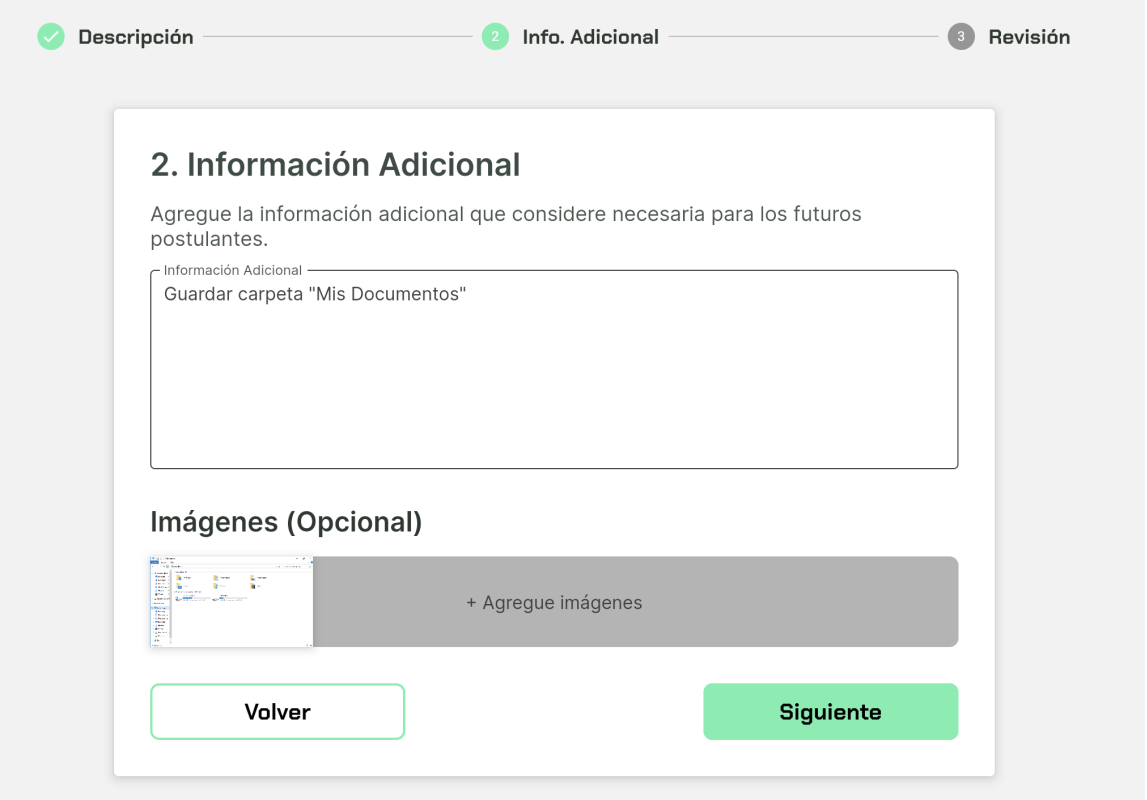

Figura A2.5 - Opción “Soy Novato/a (Simple)” - Paso 2.

- Pasos “**Tengo conocimientos (Avanzado)**”:

Figura A2.6 - Opción “Tengo conocimientos (Avanzado)” - Paso 1.

Figura A2.7 - Opción “Tengo conocimientos (Avanzado)” - Paso 2.

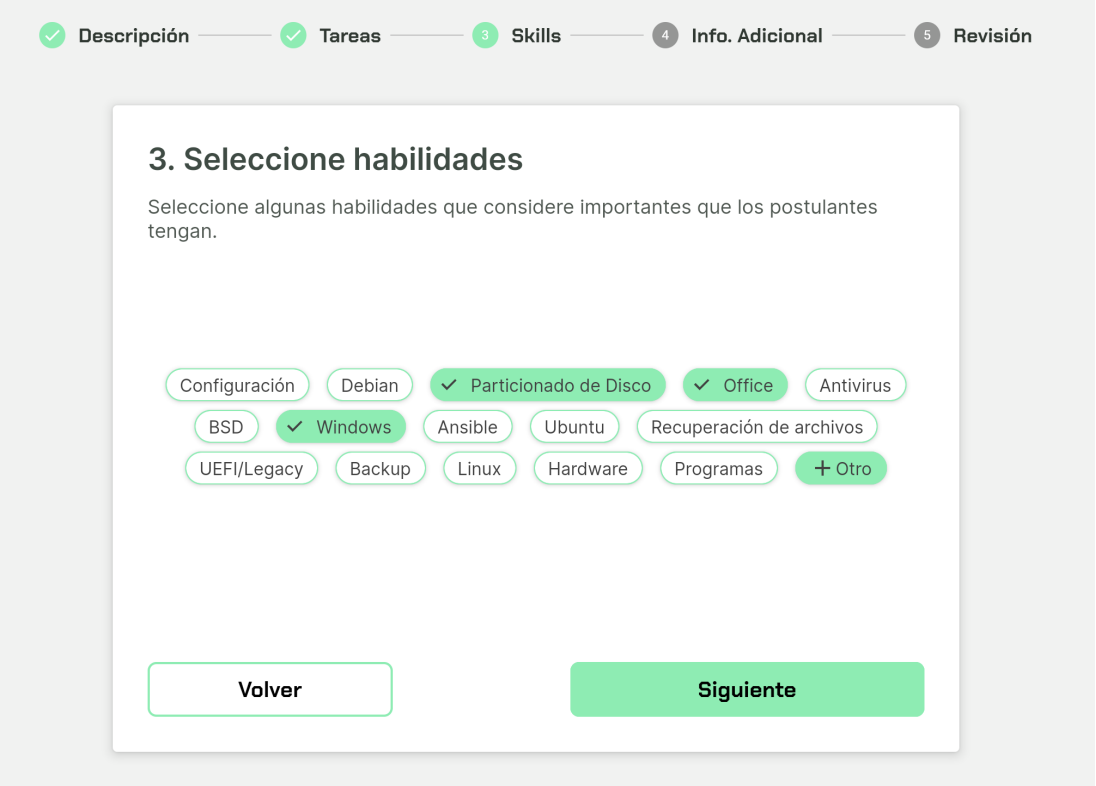

Figura A2.8 - Opción “Tengo conocimientos (Avanzado)” - Paso 3.

Figura A2.9 - Opción “Tengo conocimientos (Avanzado)” - Paso 4.

5. Una vez llegado al último paso, si el *cliente* desea, puede revisar la información de los pasos anteriores para validar información y luego, hacer click en “Crear Trabajo” para crear el mismo. (Ver Figura A2.10)

Figura A2.10 - Botón para crear trabajo en la plataforma.

6. Listo ! El trabajo ha sido creado y queda en espera de *técnicos* a postularse al mismo.

A2.4.2. Uno o más técnicos se postularon al trabajo, ¿Qué pasos debe seguir?

En caso que el trabajo creado haya recibido postulaciones por parte de diferentes técnicos, el *cliente* puede evaluar cada una de las postulaciones recibidas y aceptar una de ellas. Una vez que el *cliente* “Confirma” una de dichas postulaciones, el *técnico* que se postuló puede comenzar a trabajar en el horario estipulado.

Los pasos a seguir son:

1. Seleccionar una de las postulaciones recibidas
    a. Hacer click en la postulación y ver costos/fecha de trabajo. (Ver Figura A2.7)
    b. En caso de querer confirmar la postulación, hacer click en “Confirmar Postulación”. (Ver Figura A2.8)

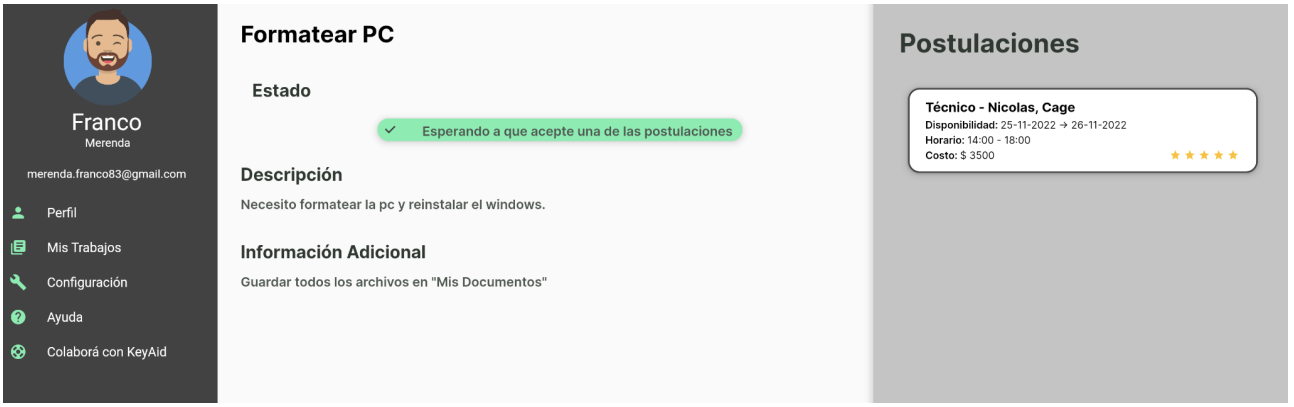

Figura A2.11 - Trabajo con una postulación recibida

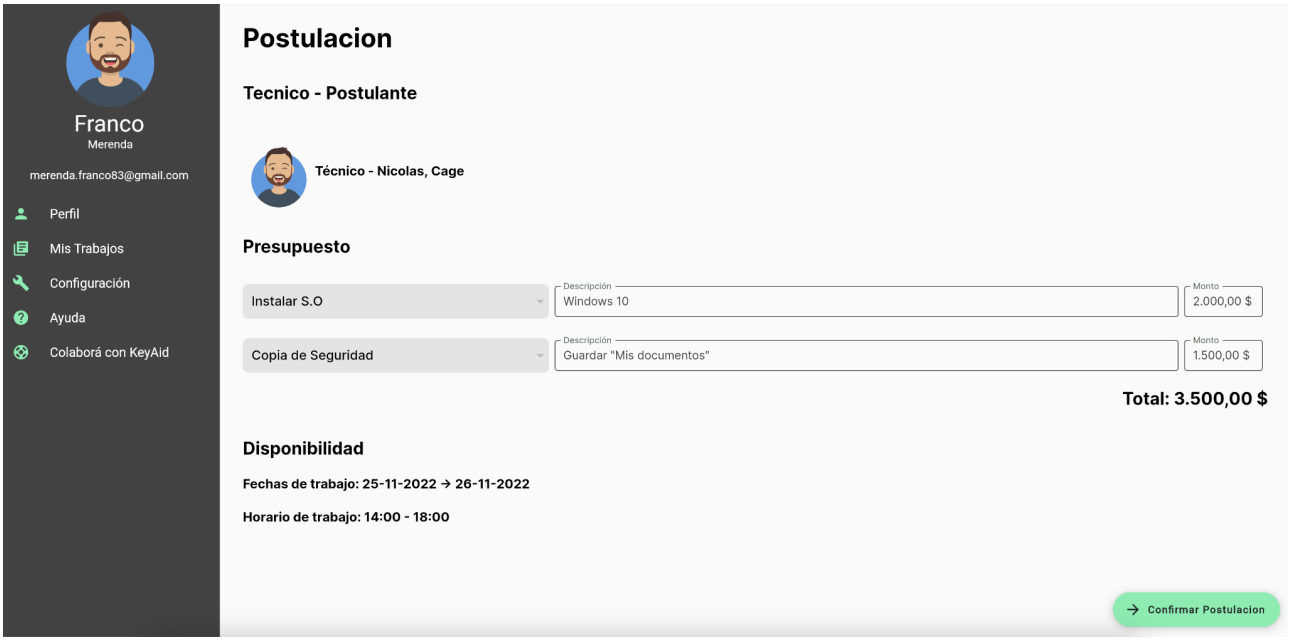

Figura A2.12 - Postulación realizada por un técnico especificando costos y tiempos.

2. El *cliente* debe conectar el dispositivo KeyAid a su PC/Servidor para que el *técnico* pueda tener acceso al dispositivo.

A2.4.3. ¿Cómo se conecta el dispositivo KeyAid al PC/Servidor?

En caso que necesite conectar el dispositivo KeyAid con su PC/Servidor/Notebook debe seguir los siguientes pasos:

1. Conectar la fuente de alimentación del dispositivo KeyAid a una toma de corriente.
2. Conectar el cable USB-C de la fuente de alimentación al puerto de alimentación del dispositivo KeyAid.
3. Conectar el cable USB-C de datos al dispositivo KeyAid, y el otro extremo, USB-A, conectarlo a uno de los puertos USB de sus PC/Servidor/Notebook.
4. Conectar Capturadora de Vídeo USB a la Raspberry PI en uno de sus puertos USB-A
5. Conectar el cable HDMI de su PC/Servidor/Notebook en la capturadora de Vídeo USB.
6. Conectar el cable Ethernet al puerto Ethernet del dispositivo KeyAid para conectarlo a Internet.

A2.4.4. El técnico finalizó el trabajo, ¿Qué debe hacer?

Una vez que el *técnico* marcó el trabajo como *finalizado*, el *cliente* debe revisar su sistema para validar que todo haya quedado de acuerdo a sus necesidades para luego enviar el pago al *técnico* por el trabajo realizado. Para realizar el pago, sólo queda seguir los siguientes pasos:

1. El *cliente* debe ir a la postulación finalizada, y hacer click en el botón “Pagar” (Ver Figura A2.9).
2. Realizar el pago a través de la plataforma utilizando alguno de los métodos de pago ofrecidos por la plataforma. Actualmente la plataforma cuenta con soporte para tarjetas de crédito (Ver Figura A2.10), próximamente, se agregarán nuevas formas de pago.

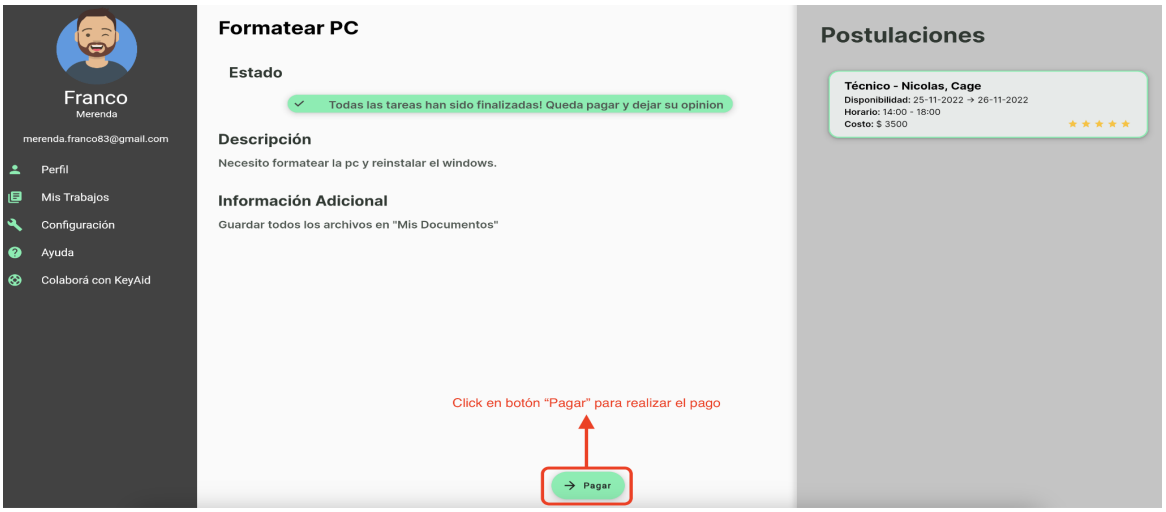

Figura A2.13 - Pantalla de trabajo - Botón para realizar el pago

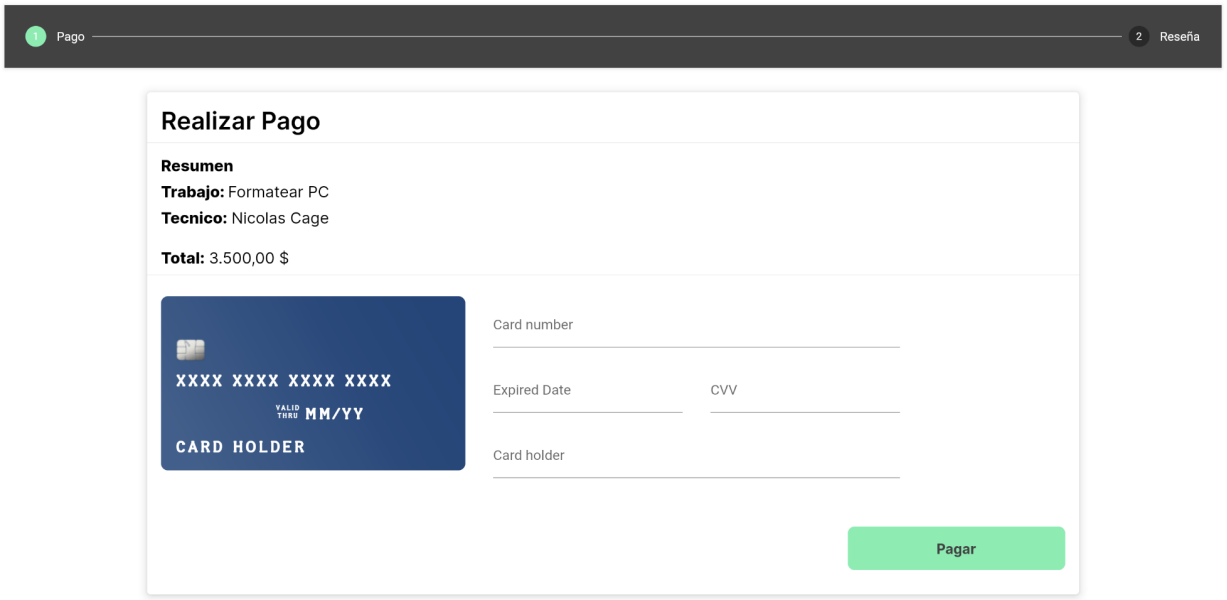

Figura A2.14 - Pagar por el trabajo - Formulario de tarjeta de crédito/débito.

3. (Opcional) El cliente puede dejar una breve reseña y calificación para que a otros usuarios en la plataforma les pueda servir a modo de referencia.

Figura A2.15 - Último paso (Opcional) - Dejar una breve reseña y calificación para el técnico.

A2.5. Preguntas referidas a Usuarios Técnicos

A2.5.1 El técnico ingresa a la plataforma, ¿Cómo comienza a trabajar?

Cuando el técnico ingresa a la plataforma, en la pantalla principal, tiene un listado con todos los trabajos disponibles para poder postularse a ellos. Los pasos a seguir son:

1. Elegir uno de los trabajos disponibles y hacer click. (Ver figura A2.12)
2. Realizar postulación al trabajo seleccionado.
    a. Indicar las *tareas* que va implicar el trabajo y su *costo*.
    b. Indicar el *tiempo* estimado y el *horario* estipulado donde se trabajará.
    c. Hacer click en **"Postular para el trabajo"**

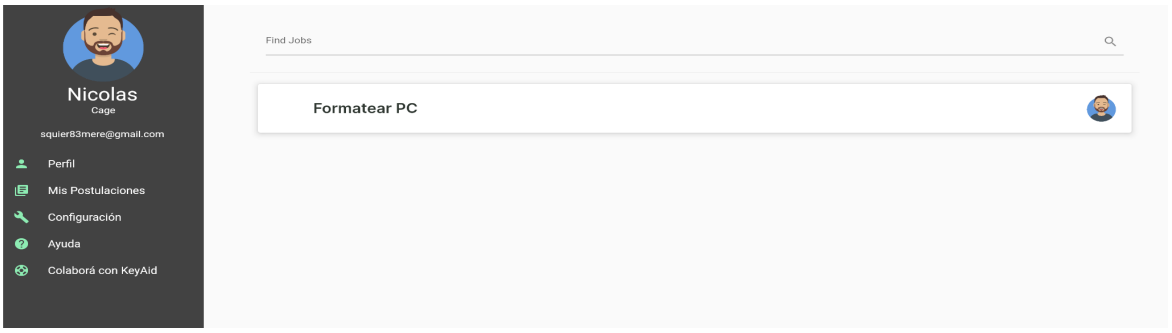

Figura A2.16 - Pantalla de Inicio con trabajos - Técnico

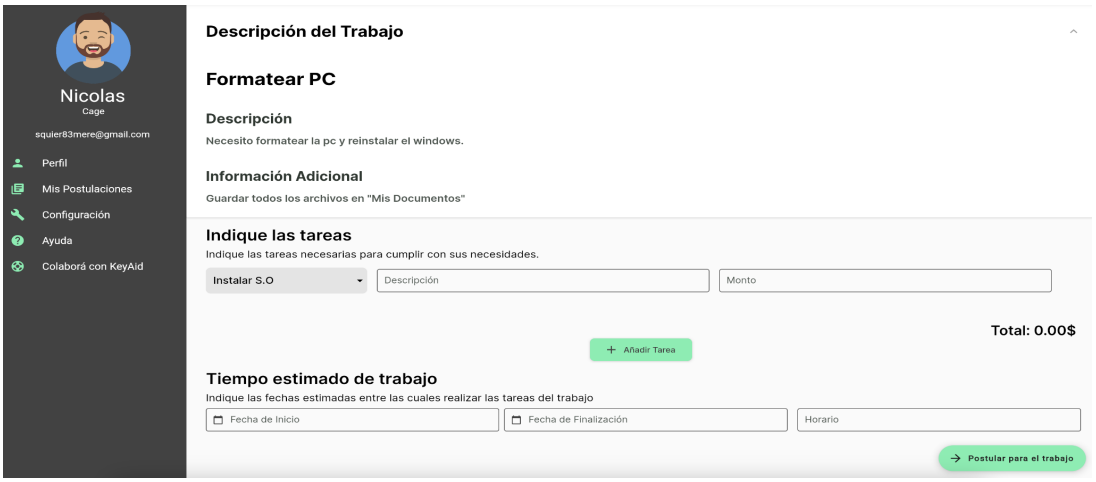

Figura A2.17 - Pantalla de postularse a un trabajo - Técnico

3. Esperar a que el *cliente* acepte la postulación realizada para poder comenzar a trabajar.

Figura A2.18 - Pantalla de postulación - Indica la espera para postulación del *cliente*.

A2.5.2. Cliente confirma la postulación, ¿Cómo acceder a la PC del cliente en forma remota?

Cuando la postulación haya sido confirmada por el *cliente*, puede acceder de forma remota a la PC del dispositivo del cliente a través del portal integrado en la plataforma. Para acceder a dicho portal, en el panel de la derecha, existe un botón (ver Figura A2.15) que lo llevará a conectarse de forma remota al PC del *cliente*.

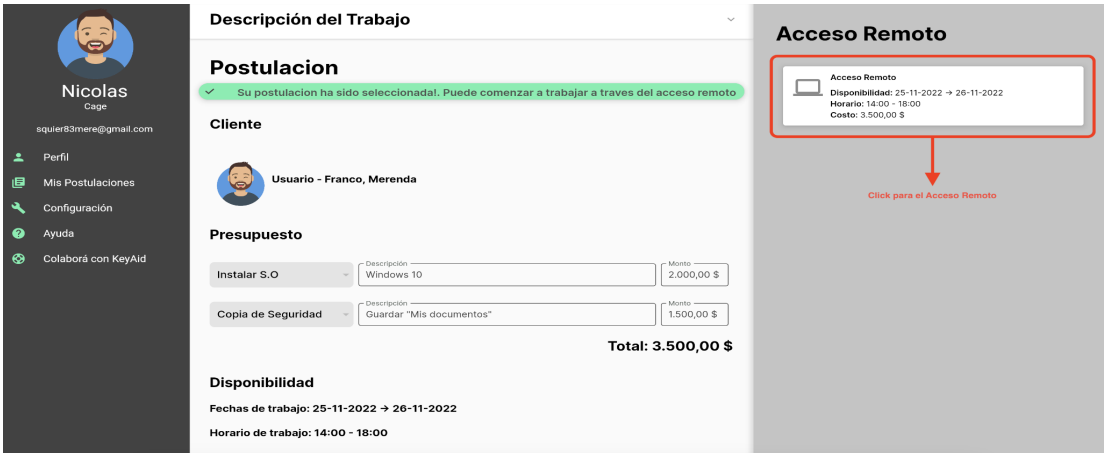

Figura A2.19 - Postulación confirmada - Botón para el acceso remoto a la PC/Servidor del cliente.

A2.5.3. El técnico finalizó con todas las tareas, ¿Cuáles son los pasos a seguir?

Cuando el *técnico* finaliza con todas las tareas que corresponden a la postulación del trabajo, el siguiente paso es marcar el trabajo como *finalizado*, para notificar al usuario *Cliente* que las tareas fueron finalizadas.

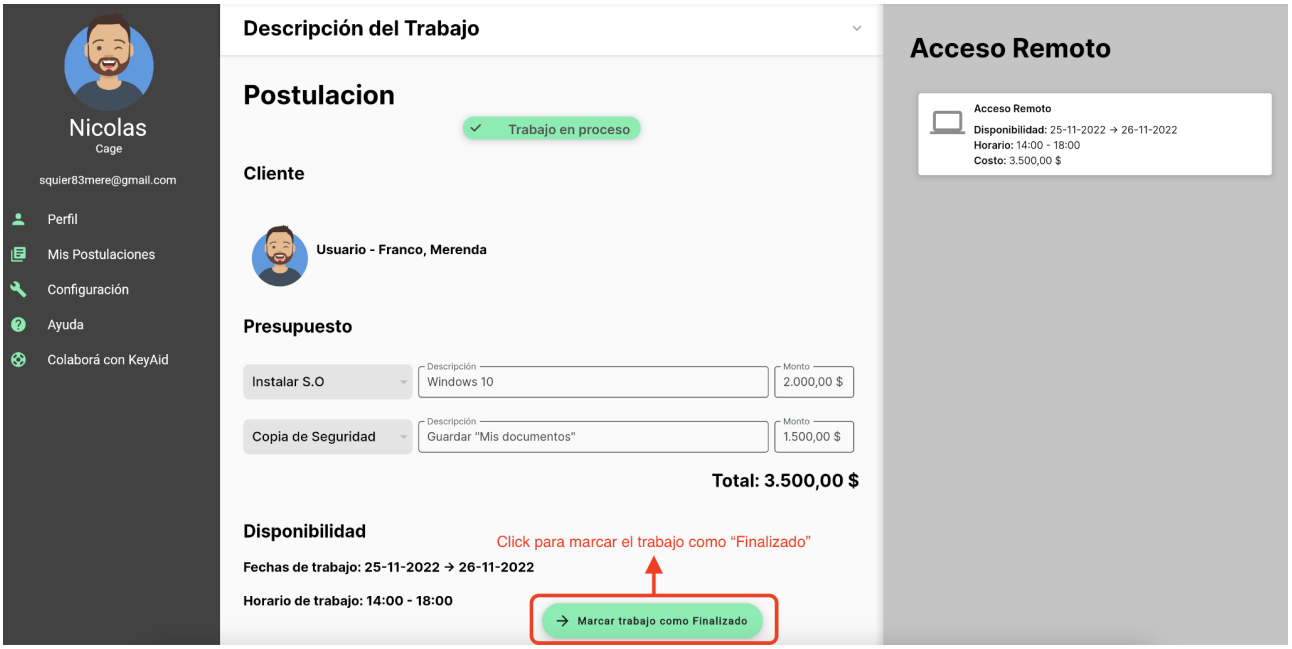

Figura A2.20 - Trabajo con todas las tareas finalizadas - Botón para marcar trabajo como finalizado.

Finalmente, sólo queda esperar por el pago y reseña por parte del usuario *Cliente*.