INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE INGENIERÍA MECÁNICA Y ELÉCTRICA
UNIDAD CULHUACÁN

"DISEÑO E IMPLEMENTACIÓN DE UN DISPOSITIVO PORTATIL DE ALMACENAMIENTO DE ARCHIVOS Y MULTIMEDIA QUE GENERE RESPALDOS AUTOMÁTICOS EN DIRECCIONES IP DE REDES ESPECÍFICAS"

ALEJANDRO JOSHUA CASTILLO GÓMEZ

ASESOR

ING. ORLANDO BELTRÁN NAVARRO

DESARROLLO PROSPECTIVO DE PROYECTOS

GRUPO 9EV21

PLANTEAMIENTO DEL PROBLEMA

En la actualidad, el almacenamiento de archivos y multimedia personal y profesional como como documentos importantes, fotos, videos, archivos de trabajo y configuraciones personalizadas en dispositivos electrónicos es esencial. Sin embargo, este aumento en la dependencia de los dispositivos electrónicos también ha llevado a un riesgo significativo de pérdida de datos debido a diversas razones, como errores humanos, fallas de hardware, ataques de malware o incluso robos.

El problema clave radica en la falta de una solución de respaldo efectiva y conveniente. A menudo, los usuarios se ven obligados a depender de prácticas manuales de respaldo, que son propensas a olvidos y retrasos, o bien, se ven limitados por la falta de automatización y recursos. Además, las soluciones de respaldo tradicionales, como los servicios en la nube, pueden plantear costos sostenidos a largo plazo y preocupaciones de privacidad.

Sumado a que según el sitio *Checkpoint.com* "*El 94 % de las organizaciones está moderada a extremadamente preocupada por la seguridad en la nube. Cuando se consultó sobre cuáles son las amenazas de seguridad más grandes que enfrentan las nubes públicas, las organizaciones calificaron la configuración incorrecta (68 %) como la principal, seguida del acceso no autorizado (58 %), las interfaces no seguras (52 %) y el secuestro de cuentas (50 %). Aquí analizaremos las principales amenazas y preocupaciones de seguridad en la nube en el mercado actual."*

Y estudiado por el Instituto Federal de Telecomunicaciones en México (IFT) " en su apartado 6.2, páginas 72 y 73 del estudio de Cloud Computing en México en Julio del 2020 "*Hoy en día las organizaciones del sector público siguen teniendo una preocupación legítima por la seguridad de sus datos, por lo que algunos gobiernos han llegado a establecer a los Proveedores de Servicios de Comunicaciones (PSC) como requisito de que todo contenido de los clientes procesado y almacenado en un sistema de TI, permanezca dentro de las fronteras del país, con el afán de brindar una capa adicional de seguridad. La residencia de los datos es una combinación de problemas asociados principalmente a riesgos de seguridad percibidos (y en algunos casos reales), relacionados con el acceso de terceros a la información, incluidas las autoridades competentes extranjeras. Los clientes del sector público desean tener la certeza de que sus datos están protegidos frente al acceso no deseado de atacantes maliciosos o de otros gobiernos. Tomar una postura rigurosa sobre la residencia de los datos a veces restringe el uso del Cloud Computing. La preocupación general en torno a la ciberseguridad, así como la posible extralimitación de la vigilancia gubernamental por parte de algunos países ha contribuido a que el debate se centre continuamente en mantener los datos en el país. No obstante, esta restricción puede ser contraproducente para el objetivo de proteger de manera efectiva datos sensibles respecto a diversos ámbitos del sector público. Los niveles más altos de protección sólo se pueden garantizar en el entorno y hábitat donde se ubica la plataforma del proveedor de Cloud Computing ya que se puede mantener la soberanía reglamentaria de*

nación-estado.”

JUSTIFICACIÓN

Para abordar la creciente necesidad de gestionar de manera eficiente los respaldos de datos en dispositivos portátiles, se propone la implementación de un dispositivo innovador que ofrezca soluciones integrales a los desafíos actuales. Este proyecto busca desarrollar un dispositivo de almacenamiento portátil con respaldo automático que cumpla con los siguientes objetivos fundamentales:

Automatización Eficiente:

El dispositivo realizará respaldos automáticos programados a intervalos regulares, eliminando la necesidad de intervención manual por parte del usuario. Esta automatización garantizará que los archivos y multimedia estén siempre actualizados y protegidos, abordando así la limitación común asociada con la realización manual de respaldos.

Accesibilidad:

Con el objetivo de proporcionar una solución verdaderamente accesible, el dispositivo será diseñado para ser fácilmente transportable y permitirá a los usuarios recuperar sus archivos y multimedia en cualquier momento y lugar. Esto se logrará sin depender de una conexión a Internet, mejorando la accesibilidad y la conveniencia para los usuarios.

Seguridad de Datos:

El proyecto se centrará en implementar un sistema de respaldo confiable que proteja los archivos y multimedia contra pérdidas, ya sea debido a errores humanos, fallos de hardware, ciberataques u otros desastres digitales. Esta función crítica aborda la necesidad imperante de salvaguardar la integridad de los datos almacenados.

Flexibilidad y Personalización:

La capacidad de personalizar y adaptar la solución a las necesidades individuales será una característica distintiva. Los usuarios podrán expandir la capacidad de almacenamiento según sea necesario y disfrutar de un diseño de código que no solo genere confianza, sino que también sea intuitivo, facilitando una experiencia de usuario personalizada y cómoda.

Desde una perspectiva social, este proyecto adquiere una relevancia destacada al empoderar a los usuarios mediante la entrega de una solución tecnológica avanzada y accesible. En el ámbito empresarial, la implementación de respaldos automáticos en dispositivos portátiles puede tener un impacto significativo en la productividad y la continuidad del negocio, ya que los datos críticos se respaldan de manera más eficiente y segura.

Este proyecto representa una innovación tecnológica al fusionar la conectividad

inalámbrica, la detección de redes específicas y la automatización de respaldos. La detección contextual de redes permite que el dispositivo actúe de manera proactiva, marcando un avance significativo en comparación con las soluciones tradicionales. Esta convergencia tecnológica no solo mejora la eficiencia del respaldo, sino que también establece un precedente para el desarrollo futuro de dispositivos portátiles inteligentes y autónomos.

OBJETIVO GENERAL

Diseñar e implementar un dispositivo portátil de almacenamiento de archivos y multimedia que realice respaldos automáticos al detectar una red específica y almacene los archivos informáticos respaldados en un dispositivo receptor con base en una red de área local administrada por una IP estática designada, para comodidad y organización del usuario al ahorrar tiempo de forma eficiente y confiable.

OBJETIVOS ESPECÍFICOS

- Diseñar el dispositivo de almacenamiento portátil con una unidad de almacenamiento de estado sólido y un controlador basado en un microprocesador Raspberry Pi 4B 8gb.
- Desarrollar un script controlador en Python o MicroPython, para el dispositivo y su acceso a la red (o redes) especificadas.
- Implementar un control de acceso para cada usuario específico y un registro de uso.
- Diseñar una interfaz de usuario intuitiva y de fácil acceso para la gestión de los respaldos y de redes.
- Integrar los anteriores objetivos garantizar el diseño, desarrollo e implementación exitosa del dispositivo de almacenamiento portátil.

CAPÍTULO I.
ESTADO DEL ARTE

ESTADO DEL ARTE

El dispositivo de almacenamiento forma parte de una evolución constante de la tecnología, fusionando la funcionalidad práctica con la ingeniería. A través de la historia del arte en el campo de la tecnología y dispositivos de almacenamiento, se evidencia una progresión continua hacia soluciones más eficientes y convenientes para el usuario.

1. Antecedentes Tecnológicos

1.1 Dispositivos Precursores

Los primeros antecedentes de almacenamiento de datos se remontan a las décadas de 1950 y 1960, donde se utilizaban cintas y tambores magnéticos como principales medios de almacenamiento en sistemas informáticos primitivos. Estos dispositivos eran limitados en capacidad y velocidad.

1.2 Nacimiento del Disco Duro

El nacimiento del disco duro como lo conocemos hoy se remonta a la década de 1950. En 1956, IBM introdujo el primer disco duro comercial, el IBM 305 RAMAC, que tenía una capacidad asombrosa de 5 megabytes distribuidos en 50 discos magnéticos.

1.3 Evolución de la Tecnología de Grabación

A lo largo de los años 60 y 70, los discos duros experimentaron mejoras significativas en la densidad de grabación magnética. La transición de la grabación de película delgada a la grabación de película gruesa permitió aumentar la capacidad de almacenamiento.

1.4 Introducción de la Interfaz IDE

En la década de 1980, la introducción de la interfaz de disco de tipo IDE (“Integrated Drive Electronics”) marcó un hito al permitir que el controlador del disco se integrara directamente en el propio disco duro. Esto simplificó la conexión y mejoró la compatibilidad con diferentes sistemas.

1.5 Ascenso de las Unidades de Estado Sólido (SSD)

A fines de la década de 2000 y principios de la de 2010, las unidades de estado sólido (SSD) emergieron como una alternativa a los discos duros tradicionales. Estas unidades, basadas en memoria flash, ofrecían velocidades de lectura y escritura significativamente más rápidas y mayor durabilidad, aunque inicialmente a un costo más elevado.

1.6 Aumento Exponencial de la Capacidad

En las últimas décadas, la capacidad de almacenamiento de los discos duros ha experimentado un crecimiento exponencial. Se ha pasado de discos duros con capacidades de megabytes y gigabytes en los primeros días a terabytes y, más recientemente, a discos duros de múltiples terabytes.

1.7 Tecnologías Emergentes

En la actualidad, se están explorando tecnologías emergentes para el futuro de los discos duros. Esto incluye avances en la grabación magnética asistida por calor (HAMR), que busca aumentar aún más la densidad de almacenamiento, y la grabación magnética asistida por corriente (CAMR).

1.8 Integración de Tecnologías en la Nube

La evolución de la tecnología de almacenamiento no solo se limita a dispositivos físicos. La integración de servicios en la nube ha cambiado la forma en que se accede y almacena la información, complementando y, en algunos casos, compitiendo con las soluciones de almacenamiento local.

1.9 Desarrollos Actuales y Futuros

En la actualidad, los discos duros continúan evolucionando con la introducción de tecnologías como la grabación magnética perpendicular (PMR) y el almacenamiento en capas (SMR). Además, se exploran enfoques más allá de los discos duros convencionales, como discos duros híbridos (HDD + SSD) y tecnologías de almacenamiento basadas en memorias no volátiles (NVM).

2. Conectividad Inalámbrica

El auge de la conectividad inalámbrica en computadoras y redes tiene sus raíces en la necesidad de superar las limitaciones de las conexiones por cable. Aunque las conexiones cableadas eran predominantes en los primeros días de las redes informáticas, surgieron las primeras formas de conectividad inalámbrica en la década de 1970 con experimentos que utilizaron microondas para transmitir datos.

La incorporación de la conectividad inalámbrica en dispositivos de almacenamiento ha sido una tendencia clave en la historia reciente. El proyecto se inspira en la demanda de soluciones que permitan la transferencia de datos de manera más flexible y sin las limitaciones de cables, aprovechando las redes Wi-Fi para facilitar los respaldos automáticos.

2.1 La Revolución Wi-Fi

La década de 1990 marcó un hito con el desarrollo y la estandarización de la tecnología Wi-Fi. El lanzamiento del estándar IEEE 802.11 en 1997 allanó el camino para la popularización de la conectividad inalámbrica en entornos domésticos y empresariales. La capacidad de acceder a la red sin depender de cables físicos cambió fundamentalmente la forma en que las personas interactúan con la tecnología.

2.2 Impacto en la Movilidad y la Flexibilidad

El auge de la conectividad inalámbrica ha transformado la forma en que las personas trabajan y se comunican. La movilidad y la flexibilidad se han vuelto esenciales en entornos laborales y personales. Los usuarios pueden acceder a la información y compartir archivos en tiempo real sin estar vinculados por cables.

3. Internet de las Cosas (IoT)

La integración de la detección de redes específicas para iniciar automáticamente respaldos se alinea con la filosofía de la Internet de las Cosas. El dispositivo no solo actúa como un medio de almacenamiento, sino como un elemento inteligente que responde de manera proactiva a su entorno, mejorando la experiencia del usuario.

3.1 Precursor

El arte de la interconexión de dispositivos electrónicos y objetos cotidianos comenzó a vislumbrarse en la década de 1980. La idea de conectar objetos a la red para permitir la comunicación entre ellos fue conceptualizada en un laboratorio de investigación de Carnegie Mellon, donde se crearon los primeros dispositivos capaces de compartir información entre sí.

3.2 Terminología y Conceptualización

La expresión "Internet de las Cosas" fue acuñada en 1999 por el investigador británico Kevin Ashton. Ashton propuso esta terminología para describir la conexión de objetos físicos a internet para recopilar y compartir datos. Esta conceptualización marcó el inicio formal del IoT como una disciplina en sí misma.

3.3 Desarrollo Tecnológico

A medida que avanzaba la década de 2000, los avances tecnológicos en sensores, actuadores y sistemas de comunicación permitieron una mayor integración de objetos en la red. La miniaturización de componentes electrónicos y la mejora de las tecnologías inalámbricas facilitaron la conexión de una amplia gama de dispositivos.

3.4 Crecimiento de Dispositivos Conectados

En la última década, hemos sido testigos de un rápido crecimiento en el número de dispositivos conectados. Desde electrodomésticos inteligentes y dispositivos de salud hasta sistemas de monitoreo industrial, la variedad de objetos integrados en el IoT ha ampliado enormemente, formando una red interconectada de "cosas" que recopilan y comparten datos.

4. Seguridad y Ciberseguridad

La gestión de archivos y datos en entornos en la nube demanda una atención especial para resguardar la información sensible ante amenazas tanto internas como externas. Para ello, se recurre a una variedad de medidas y tecnologías diseñadas para preservar la confidencialidad, integridad y disponibilidad de los datos.

Entre estas medidas se incluyen la autenticación multifactorial, la encriptación de extremo a extremo y la gestión de accesos basada en roles. Estas estrategias garantizan que únicamente usuarios autorizados accedan a la información y que los datos permanezcan protegidos tanto en reposo como en tránsito.

4.1 Autenticación Multifactorial

La autenticación multifactorial es un método de seguridad que requiere la verificación de la identidad del usuario a través de múltiples factores. Estos factores suelen clasificarse en tres categorías: algo que sabe el usuario (como una contraseña), algo que tiene (como un token o tarjeta de seguridad) y algo que es (como una huella dactilar o reconocimiento facial). Los métodos comunes de autenticación multifactorial incluyen el uso de contraseñas junto con códigos de verificación enviados a dispositivos móviles, el uso de tarjetas de seguridad que generan códigos únicos y la biometría, como el escaneo de huellas dactilares o reconocimiento facial.

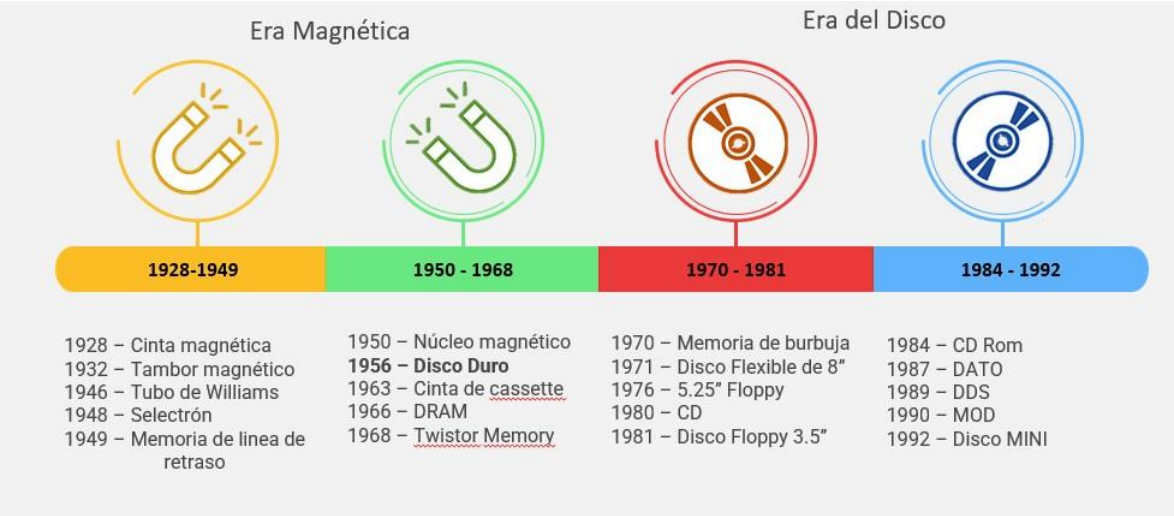
4.2 Encriptación de Extremo a Extremo

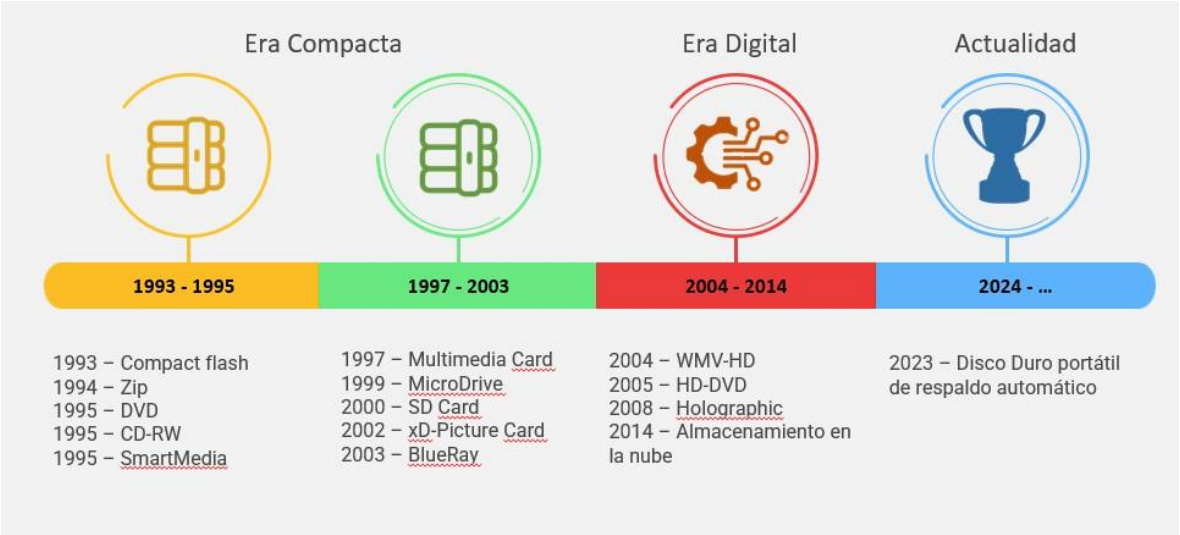
La encriptación de extremo a extremo es un método de seguridad que protege los datos durante su transmisión y almacenamiento, garantizando que solo los remitentes y destinatarios autorizados puedan acceder a la información. Los algoritmos de encriptación comunes utilizados en la encriptación de extremo a extremo incluyen AES (Advanced Encryption Standard) y RSA (Rivest-Shamir-Adleman). Estos algoritmos son altamente seguros y utilizan claves de encriptación únicas para proteger los datos.

4.3 Gestión de Accesos Basada en Roles

La gestión de accesos basada en roles (RBAC) es un enfoque para controlar el acceso a los recursos de la red según la función o el rol de un usuario dentro de una organización. La gestión de accesos basada en roles proporciona una manera eficiente y segura de administrar los privilegios de acceso, reduciendo el riesgo de accesos no autorizados y ayudando a garantizar la integridad y confidencialidad de los datos en la nube.

LÍNEA DEL TIEMPO





CAPÍTULO III.
DISEÑO E IMPLEMENTACIÓN

DIAGRAMA DE BLOQUES

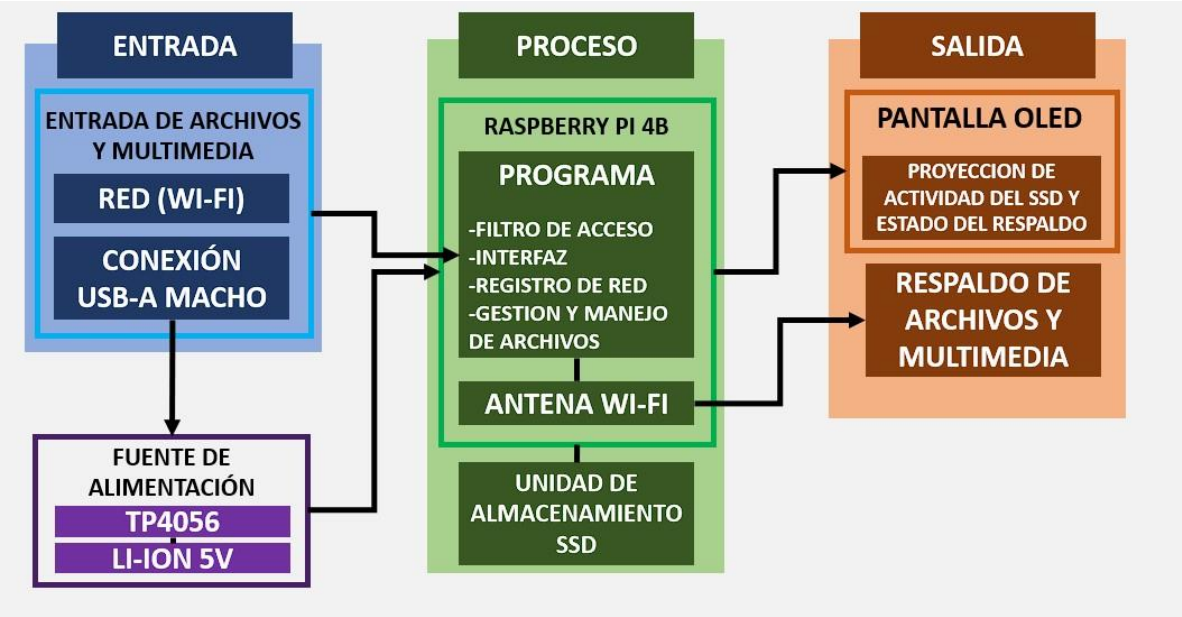


Ilustración 1: Diagrama a Bloques

En el diagrama a bloques se ven representadas tres etapas primordiales del proyecto, que son entrada, proceso y salida, así como un anexo para la etapa de alimentación.
Para la etapa de entrada se tienen descritos los dos procesos de entrada de datos, dónde se empezará la transferencia de estos, iniciando por la conexión de red WI-FI y la conexión física, USB-A Macho.

La etapa de alimentación nos permite tener en cuenta la manera en la que el dispositivo nos permite manejar sus pertinentes funciones de manera portátil, es decir, sin la necesidad de un cableado externo para su función, dicha etapa está conformada por un arreglo mixto “serie-paralelo” de baterías 18650 controlando su salida a los 5v de trabajo del dispositivo por medio de un TP5056 que funciona como controlador de carga para baterías LI-ION.

Pasando por el procesos, dónde nuestro núcleo de dicho sector es el microprocesador Raspberry pi 4B, que centraliza la operación del proyecto, concentrando en sí, el código pertinente para el desarrollo del filtro de acceso que permite saber y obtener el registro de los usuarios de los respaldos y archivos, la interfaz que permite conectar de manera más directa y amigable el dispositivo a la red pertinente, así como a la dirección IP dónde los respaldos serán depositados de manera fija, el registro de la red dónde se estaría conectando el dispositivo y la gestión de archivos a respaldar.

Por último, en la etapa de salida, a modo de proyección del proceso funciona un display oled SSD1306 que nos permite saber en tiempo real los procedimientos llevados a cabo por el dispositivo, incluyendo también una proyección de informes para errores, tanto de conexión, como de error de respaldo.

DIAGRAMA DE FLUJO

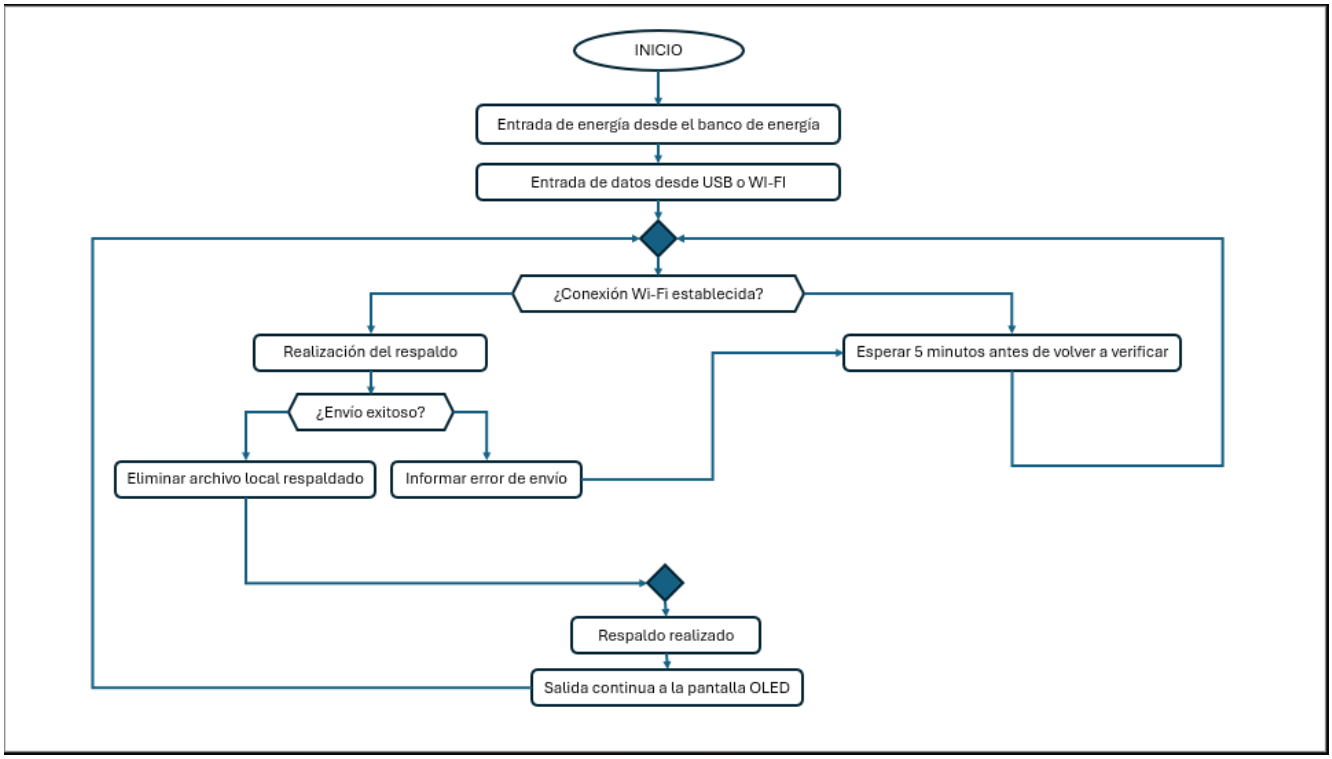


Diagrama 1: Diagrama de Flujo

DIAGRAMA DE CONEXIÓN

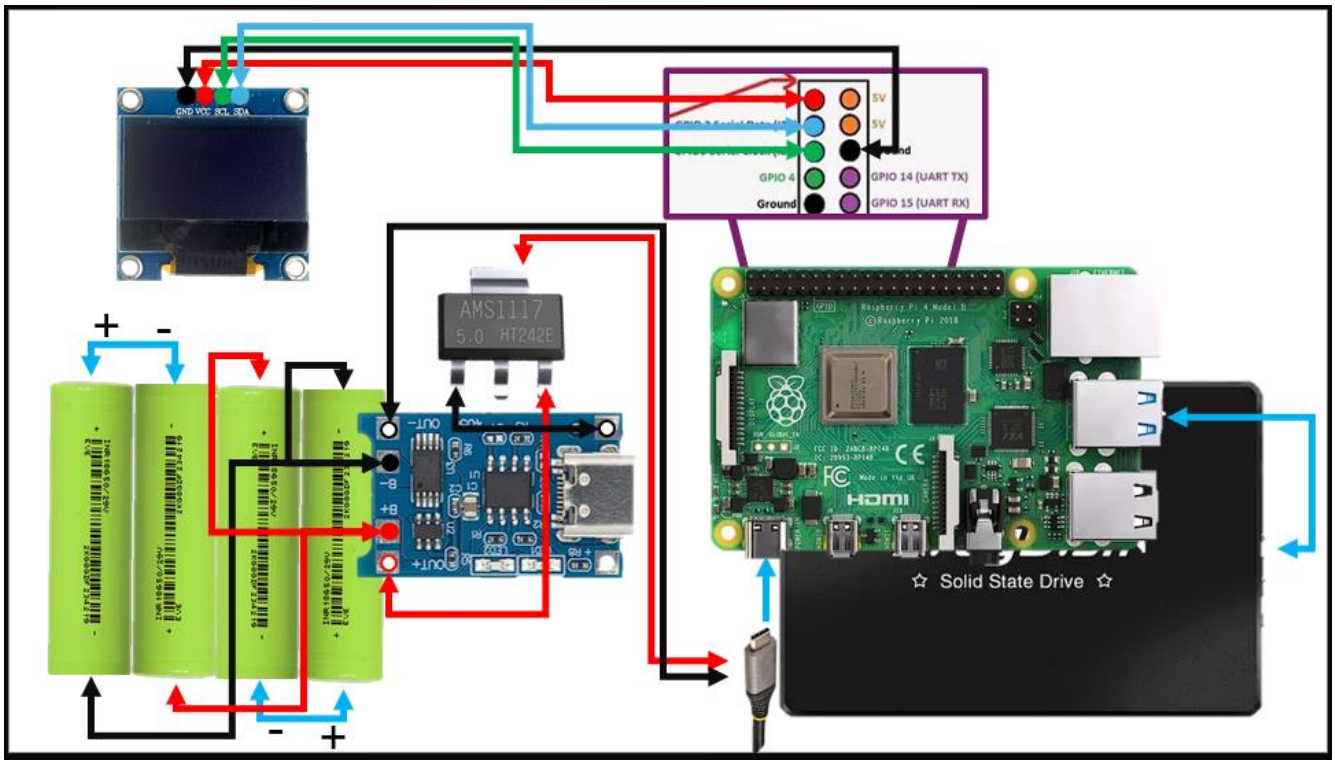


Diagrama 2: Diagrama de conexión

DIAGRAMA ESQUEMÁTICO

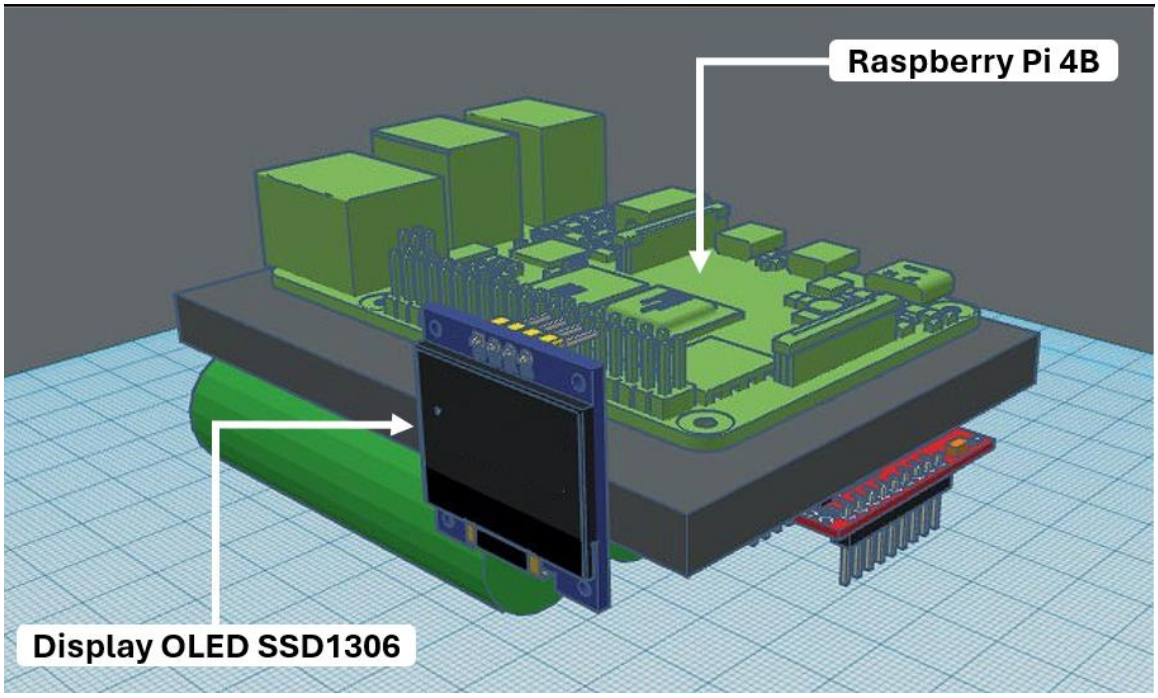


Diagrama 3: Vista 1 del Diagrama esquemático 3D

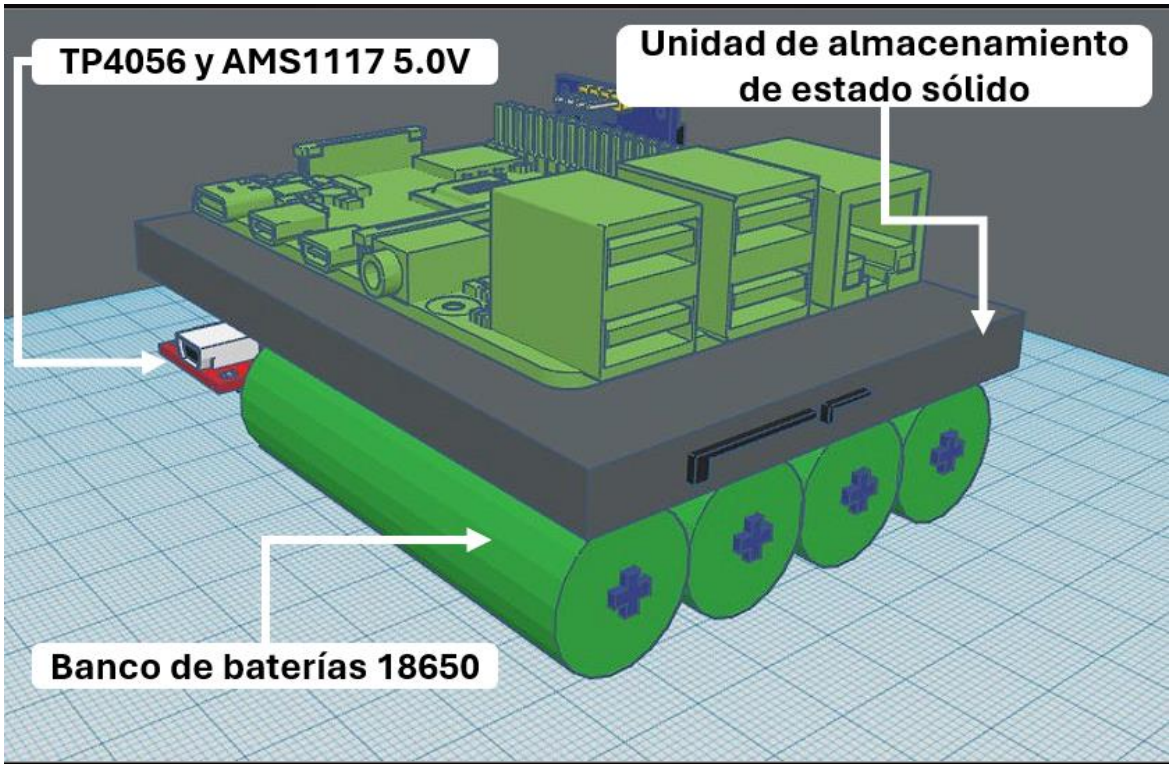


Diagrama 4: Vista 2 del Diagrama esquemático 3D

CÁLCULOS DE DISEÑO

Se mostrarán los cálculos pertinentes para el funcionamiento específico del banco de batería, que permitirá la portabilidad del dispositivo.

Primero, necesitaremos estimar el consumo total de energía por hora de todos los dispositivos. Asumiendo que la Raspberry Pi 4B consume alrededor de 3-5W, el SSD y el display OLED también tienen sus propios consumos de energía, pero para simplificar, tomemos un consumo total de 10W para todos los dispositivos juntos.

El cálculo para el consumo de energía por hora sería:

$$Consumo\ total\ = 10\ W$$

Para convertir este consumo de energía a miliamperios (mA), usaremos la fórmula:

$$Consumo\ total\ (mA) = \frac{Consumo\ total\ (W)}{Tensión\ de\ las\ beterías (V)}$$

La Raspberry Pi 4B generalmente funciona con 5V, por lo que podemos usar 5V como la tensión de las baterías para este cálculo.

$$Consumo\ total\ (mA) = \frac{10\ W}{5\ V} = 2000\ mA$$

Ahora que sabemos que el consumo total de energía por hora es de aproximadamente 2000mAh, podemos calcular una estimación del tiempo de funcionamiento dividiendo la capacidad total de las baterías por el consumo total de energía por hora:

$$Tiempo\ de\ funcionamiento\ (Horas) = \frac{2100\ mAh}{2000\ mAh} = 1.05\ Horas$$

Al conectar dos baterías en serie y luego ese par en paralelo con otro par en serie, estás aumentando tanto la capacidad total como el voltaje total suministrado.

En el arreglo serie-paralelo, el voltaje total sería la suma del voltaje de las dos baterías en serie. Sin embargo, la capacidad total sería la suma de las capacidades de las dos baterías en serie, multiplicada por dos (debido a la conexión en paralelo).

Si cada batería tiene una capacidad de 2100mAh, en el arreglo serie-paralelo tendríamos una capacidad total de 4200mAh debido a la conexión en paralelo.

Entonces, usando la misma fórmula utilizada anteriormente para calcular el tiempo de funcionamiento con la nueva capacidad total de las baterías (4200mAh), el tiempo de funcionamiento sería:

$$Tiempo\ de\ funcionamiento\ (Horas) = \frac{4200\ mAh}{2000\ mAh} = 2.1\ Horas$$

CÓDIGO DEL PROGRAMA

```python
import os
import subprocess
import time
from luma.core.interface.serial import i2c
from luma.core.render import canvas
from luma.oled.device import ssd1306

# Direccion IP especifica a la que se enviaron los respaldos
ip_destino = "192.168.1.253"

# Ruta del directorio que deseas respaldar
directorio_respaldar = "/home/pi/media/joshux07/Rasp/Respaldos"

# Inicialización de la pantalla OLED SSD1306
serial = i2c(port=1, address=0x3C)
device = ssd1306(serial)

# Comprobacion de la conexion a la red WiFi
def esta_conectado():
    try:
        output = subprocess.check_output(["iwgetid"]).decode("utf-8")
        if "ESSID" in output:
            return True
        else:
            return False
    except Exception as e:
        print("Error al comprobar la conexion WiFi:", e)
        return False

# Realizacion del respaldo
def realizar_respaldo():
    try:
        # Nombre del archivo de respaldo con la fecha y hora actual
        nombre_archivo = time.strftime("%Y%m%d-%H%M%S") + "_respaldo.tar.gz"

        # Comando para crear el archivo de respaldo
        comando_respaldo = "tar -czf {} {}".format(nombre_archivo, directorio_respaldar)

        # Ejecutar el comando de respaldo
        os.system(comando_respaldo)

        # Enviar el archivo de respaldo a la IP de destino
        comando_envio = "scp {} usuario@{}:/ruta/destino".format(nombre_archivo, ip_destino)
        os.system(comando_envio)

        # Eliminar el archivo de respaldo localmente
        os.remove(nombre_archivo)

        # Mostrar mensaje en la pantalla OLED
        with canvas(device) as draw:
            draw.text((0, 0), "Respaldo hecho", fill="white")

```

```
53          print("Respaldo realizado y enviado exitosamente a", ip_destino)
54      except Exception as e:
55          # Mostrar mensaje de error en la pantalla OLED
56          with canvas(device) as draw:
57              draw.text((0, 0), "Error en el respaldo", fill="white")
58
59          print("Error al realizar el respaldo:", e)
60
61  # Bucle principal
62  while True:
63      if esta_conectado():
64          print("Conectado a la red WiFi.")
65          realizar_respaldo()
66      else:
67          print("No se detecto conexion a la red WiFi.")
68      # Esperar 5 minutos antes de volver a comprobar
69      time.sleep(300)  # 300 segundos = 5 minutos
70
```

Bloque de importaciones:

```
1  import os
2  import subprocess
3  import time
4  from luma.core.interface.serial import i2c
5  from luma.core.render import canvas
6  from luma.oled.device import ssd1306
```

En este bloque, importamos los módulos necesarios para nuestro programa:

os: Proporciona funciones para interactuar con el sistema operativo, como ejecutar comandos del sistema y manipular archivos.
subprocess: Permite crear procesos, ejecutar comandos y comunicarse con ellos.
time: Proporciona funciones relacionadas con el tiempo, como obtener la hora actual y pausar la ejecución del programa.
i2c, canvas y ssd1306 son clases y métodos proporcionados por la biblioteca luma.oled, que nos permitirá interactuar con la pantalla OLED SSD1306.

Definición de variables:

```
8  # Direccion IP especifica a la que se enviaron los respaldos
9  ip_destino = "192.168.1.253"
10
11 # Ruta del directorio que deseas respaldar
12 directorio_respaldar = "/home/pi/media/joshux07/Rasp/Respaldos"
```

Aquí establecemos la dirección IP a la que se enviarán los respaldos (ip_destino) y la ruta del directorio que se respaldará (directorio_respaldar).

Inicialización de la pantalla OLED:

```
14  # Inicialización de la pantalla OLED SSD1306
15  serial = i2c(port=1, address=0x3C)
16  device = ssd1306(serial)
```

Creamos una instancia de la clase i2c para establecer la comunicación a través del bus I2C en el puerto 1 y con la dirección 0x3C (dirección predeterminada para la pantalla OLED SSD1306). Luego, creamos un objeto device de la clase ssd1306 para interactuar con la pantalla.

Función para comprobar la conexión WiFi:

```
19  def esta_conectado():
20      try:
21          output = subprocess.check_output(["iwgetid"]).decode("utf-8")
22          if "ESSID" in output:
23              return True
24          else:
25              return False
26      except Exception as e:
27          print("Error al comprobar la conexion WiFi:", e)
28          return False
```

Esta función (esta_conectado) verifica si la Raspberry Pi está conectada a una red WiFi. Utiliza el comando iwgetid para obtener información sobre la conexión WiFi. Si la conexión está activa, devuelve True; de lo contrario, devuelve False.

subprocess.check_output(["iwgetid"]): Esta parte ejecuta el comando iwgetid, que normalmente se utiliza para obtener información sobre la red inalámbrica a la que está conectado el dispositivo. Este comando puede proporcionar información como el SSID (nombre de la red) a la que está conectado el dispositivo.

.decode("utf-8"): La función .decode() se utiliza para decodificar una secuencia de bytes en una cadena Unicode utilizando el formato especificado. En este caso, estamos decodificando la salida del comando iwgetid utilizando UTF-8, que es un formato de codificación de caracteres ampliamente utilizado y compatible con una amplia gama de caracteres y símbolos.

Función para realizar el respaldo:

```python
def realizar_respaldo():
    try:
        # Nombre del archivo de respaldo con la fecha y hora actual
        nombre_archivo = time.strftime("%Y%m%d-%H%M%S") + "_respaldo.tar.gz"

        # Comando para crear el archivo de respaldo
        comando_respaldo = "tar -czf {} {}".format(nombre_archivo, directorio_respaldar)

        # Ejecutar el comando de respaldo
        os.system(comando_respaldo)

        # Enviar el archivo de respaldo a la IP de destino
        comando_envio = "scp {} usuario@{}:/ruta/destino".format(nombre_archivo, ip_destino)
        os.system(comando_envio)

        # Eliminar el archivo de respaldo localmente
        os.remove(nombre_archivo)

        # Mostrar mensaje en la pantalla OLED
        with canvas(device) as draw:
            draw.text((0, 0), "Respaldo hecho", fill="white")

        print("Respaldo realizado y enviado exitosamente a", ip_destino)
    except Exception as e:
        # Mostrar mensaje de error en la pantalla OLED
        with canvas(device) as draw:
            draw.text((0, 0), "Error en el respaldo", fill="white")

        print("Error al realizar el respaldo:", e)
```

Esta función (realizar_respaldo) realiza el respaldo del directorio especificado (directorio_respaldar). Primero, crea un nombre de archivo único para el respaldo basado en la fecha y hora actual. Luego, utiliza el comando tar para crear un archivo comprimido del directorio. A continuación, utiliza scp para enviar el archivo de respaldo a la dirección IP especificada (ip_destino). Si el respaldo se realiza correctamente, muestra "Respaldo hecho" en la pantalla OLED y lo imprime en la consola. En caso de error, muestra "Error en el respaldo" en la pantalla OLED y lo imprime en la consola.

nombre_archivo = time.strftime("%Y%m%d-%H%M%S") + "_respaldo.tar.gz":

time.strftime("%Y%m%d-%H%M%S"): Utiliza la función strftime del módulo time para formatear la hora actual según el formato especificado. %Y representa el año con cuatro dígitos, %m representa el mes, %d representa el día, %H representa la hora en formato 24 horas, %M representa los minutos y %S representa los segundos. Por ejemplo, si la hora actual es 14:30:15 del 1 de enero de 2024, esta parte del código produciría la cadena "20240101-143015".
"_respaldo.tar.gz": Es una cadena constante que se concatena con la cadena formateada anteriormente para crear el nombre completo del archivo de respaldo. En este caso, el nombre del archivo de respaldo será algo como "20240101-143015_respaldo.tar.gz". Esta convención de nombres es común para incluir la marca de tiempo en el nombre del archivo para que sea único y fácil de identificar.
comando_respaldo = "tar -czf {} {}".format(nombre_archivo, directorio_respaldar)":

"tar -czf {} {}".format(nombre_archivo, directorio_respaldar)": Este es un comando de shell que utiliza el programa tar para crear un archivo comprimido (gzip) que contiene los archivos del directorio especificado.

{}{}: En la cadena de formato, {} son marcadores de posición que se llenan con los valores proporcionados en el método format().
nombre_archivo: Este es el primer argumento pasado a format(), por lo que se sustituye en el primer {} en la cadena de formato. Esto proporciona el nombre del archivo que se creará.
directorio_respaldar: Este es el segundo argumento pasado a format(), por lo que se sustituye en el segundo {} en la cadena de formato. Esto proporciona la ruta del directorio que se comprimirá en el archivo de respaldo.
En resumen, esta línea de código construye un comando de shell que utiliza tar para comprimir el contenido del directorio directorio_respaldar en un archivo con nombre nombre_archivo.

Bucle principal:

```
62 while True:
63     if esta_conectado():
64         print("Conectado a la red WiFi.")
65         realizar_respaldo()
66     else:
67         print("No se detecto conexion a la red WiFi.")
68     # Esperar 5 minutos antes de volver a comprobar
69     time.sleep(300)  # 300 segundos = 5 minutos
```

Este bucle principal se ejecuta continuamente. En cada iteración, verifica si la Raspberry Pi está conectada a la red WiFi. Si está conectada, realiza el respaldo llamando a la función realizar_respaldo(). Si no está conectada, muestra un mensaje indicando que no se detectó conexión a la red WiFi. Después de cada iteración, el programa espera 5 minutos antes de volver a verificar la conexión WiFi.

CAPÍTULO IV.
PRUEBAS Y RESULTADOS

En el desarrollo del proyecto, nos encontramos con el primer obstáculo al interactuar con el monitor de la Raspberry Pi, la plataforma sobre la cual se ejecuta el código. RaspberryOS, siendo un sistema operativo relativamente novedoso en comparación con otros más establecidos, nos presentó un desafío inesperado y significativo. Es crucial destacar este punto dado que las soluciones disponibles para sistemas tan recientes son limitadas y, en muchos casos, se encuentran en un estado de continua evolución y desarrollo.

PRUEBAS Y ERRORES EN LA INTERFÁZ

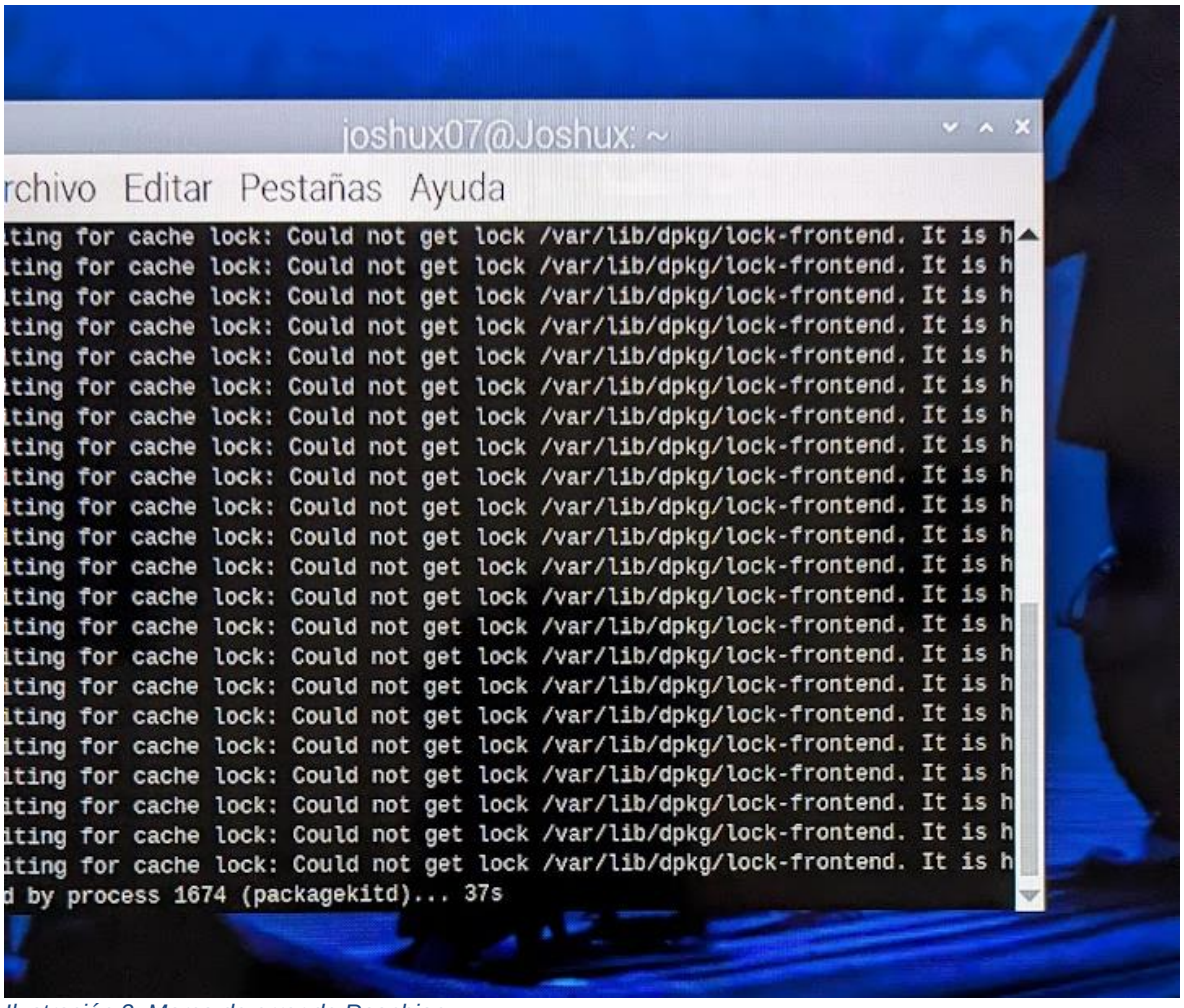


Ilustración 2: Marca de error de Raspbian:

El problema inicial surgió al tratar de integrar nuestro código con los controladores equivalentes disponibles para el sistema. Este proceso se reveló como un verdadero dolor de cabeza, ya que los controladores existentes no eran compatibles con el nuevo código que estábamos implementando para el arranque. Este inconveniente marcó únicamente el inicio de una serie de dificultades que tuvimos que enfrentar durante el desarrollo de nuestro proyecto.

Una solución temporal que encontramos para sortear este problema fue reiniciar forzosamente la Raspberry Pi. Si bien esta medida permitía continuar con el trabajo, era una solución poco práctica y, en última instancia, insostenible a largo plazo. Nos enfrentamos a la necesidad de encontrar una solución más definitiva y efectiva para garantizar el funcionamiento fluido de nuestro proyecto en la Raspberry Pi.

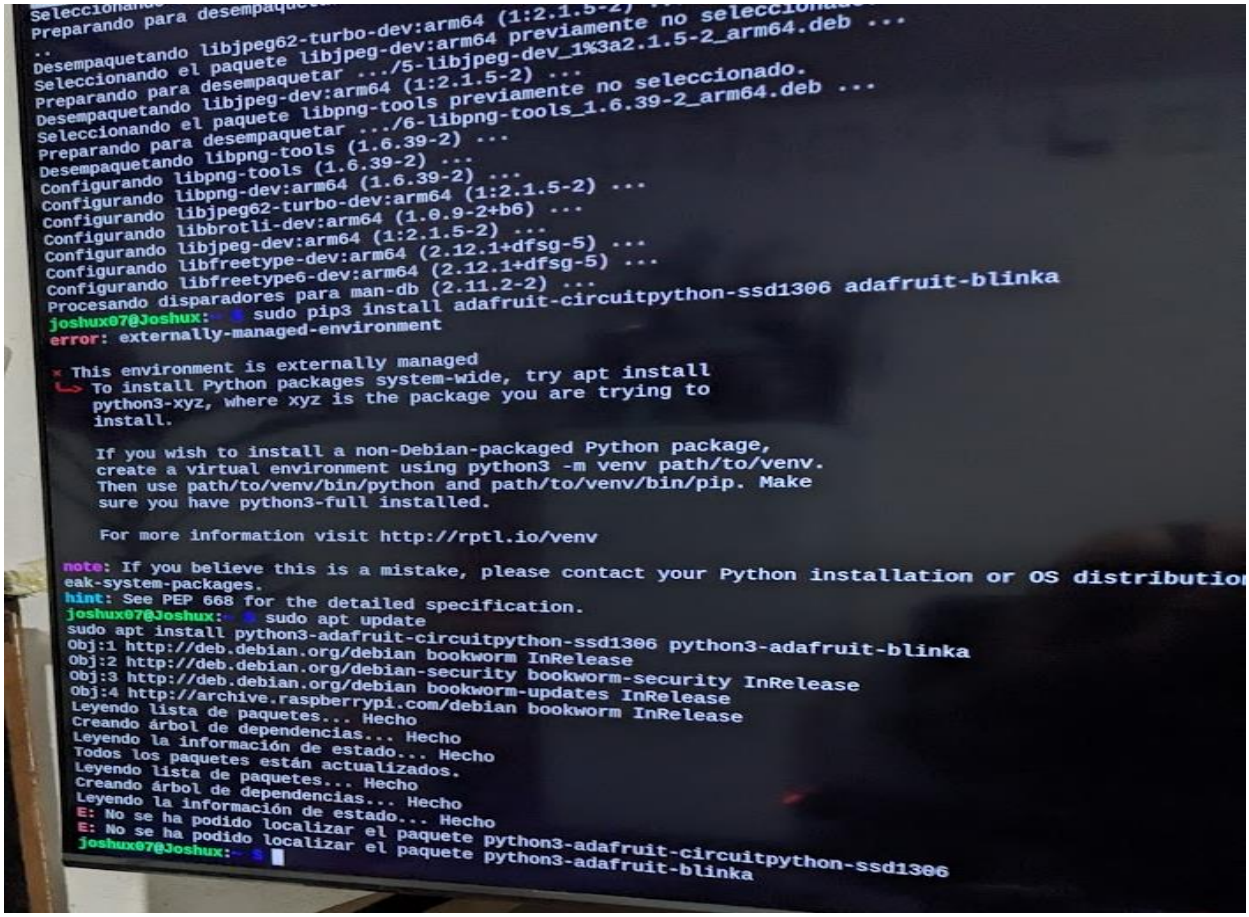


Ilustración 3: Etapa 2 del error

A medida que avanzábamos en nuestro proyecto, nos dimos cuenta de que el problema inicial no era un incidente aislado, sino más bien el preludio de un desafío mucho mayor. Con el tiempo, el error inicial comenzó a manifestarse de manera más pronunciada, hasta que finalmente se volvió completamente imposible iniciar la Raspberry Pi de manera tradicional.
Una vez más, el corazón del problema residía en la falta de soporte para el nuevo sistema operativo de Raspberry, ahora conocido como Raspberry OS (anteriormente Raspbian). Esta carencia de compatibilidad representaba una seria amenaza para el progreso de nuestro proyecto, ya que impedía que nuestra aplicación se ejecutara de manera adecuada en la plataforma designada.

Ante esta situación crítica, nos vimos obligados a tomar medidas drásticas. Decidimos crear una nueva partición en el disco de la Raspberry Pi en un intento desesperado por recuperar el archivo del código, que lamentablemente no habíamos respaldado adecuadamente. Este archivo era esencial para el funcionamiento de nuestra aplicación y su pérdida comprometía gravemente todo el trabajo realizado hasta el momento.

El proceso de recuperación del código fue arduo y desafiante, pero finalmente logramos recuperarlo, aunque no sin sacrificio. Una vez que aseguramos el archivo del código, el siguiente paso fue reinstalar completamente el sistema operativo. Esta tarea no solo consumió una cantidad considerable de tiempo y recursos, sino que también nos recordó la importancia de implementar procedimientos de respaldo robustos y efectivos en futuros proyectos para evitar la pérdida de datos críticos. En resumen, el incidente nos enseñó valiosas lecciones sobre la importancia de la planificación y la preparación meticulosa en el desarrollo de proyectos tecnológicos.

ERRORES EN LA CONEXIÓN WIFI

Ilustración 4:Modem de segunda mano

Para el apartado de pruebas, decidí adquirir un módem de segunda mano, específicamente uno proporcionado por TELMEX. Mi objetivo era configurarlo con un nombre de red y contraseña personalizados para adaptarlo a mis necesidades de respaldo de datos. Sin embargo, me encontré con una serie de inconvenientes que, en un principio, pasé por alto debido a que mi prioridad estaba en otras áreas del proyecto.

En un principio, ignoré las fallas de estabilidad que experimenté con el módem, ya que no las consideraba un problema crítico para el respaldo de datos en ese momento. Sin embargo, conforme avancé en el desarrollo y comencé a realizar pruebas más exhaustivas, se hizo evidente que estos problemas de estabilidad estaban teniendo un impacto directo en el funcionamiento de mi Raspberry Pi al ejecutar mi código.

Una de las manifestaciones más notorias de este problema fue la ocurrencia de un bucle infinito de transferencia de datos al intentar ejecutar mi código en la Raspberry Pi. Resultó que, debido a la naturaleza del código que estaba ejecutando, el cual funcionaba como un puerto de transferencia de datos en una red de área local, el módem no estaba recibiendo correctamente los datos transmitidos desde la Raspberry Pi.

Este malentendido en la transmisión de datos causó un efecto de "rebote" en los archivos que estaba intentando respaldar. Es decir, los datos que salían de la Raspberry Pi no lograban ingresar al puerto USB del módem de manera adecuada y, en lugar de ser almacenados, quedaban "en el aire", causando una interferencia significativa en la conexión Wi-Fi del dispositivo.

Este problema no solo afectó la estabilidad de la conexión Wi-Fi de mi Raspberry Pi, sino que

también comprometió la integridad de los datos que estaba intentando respaldar. Fue un recordatorio importante de la necesidad de prestar atención a todos los componentes y aspectos del sistema durante las pruebas, incluso aquellos que inicialmente pueden parecer menos críticos. En conclusión, esta experiencia me enseñó valiosas lecciones sobre la importancia de abordar y solucionar los problemas de manera proactiva durante el proceso de desarrollo de un proyecto tecnológico.

PRUEBAS SATISFACTORIAS

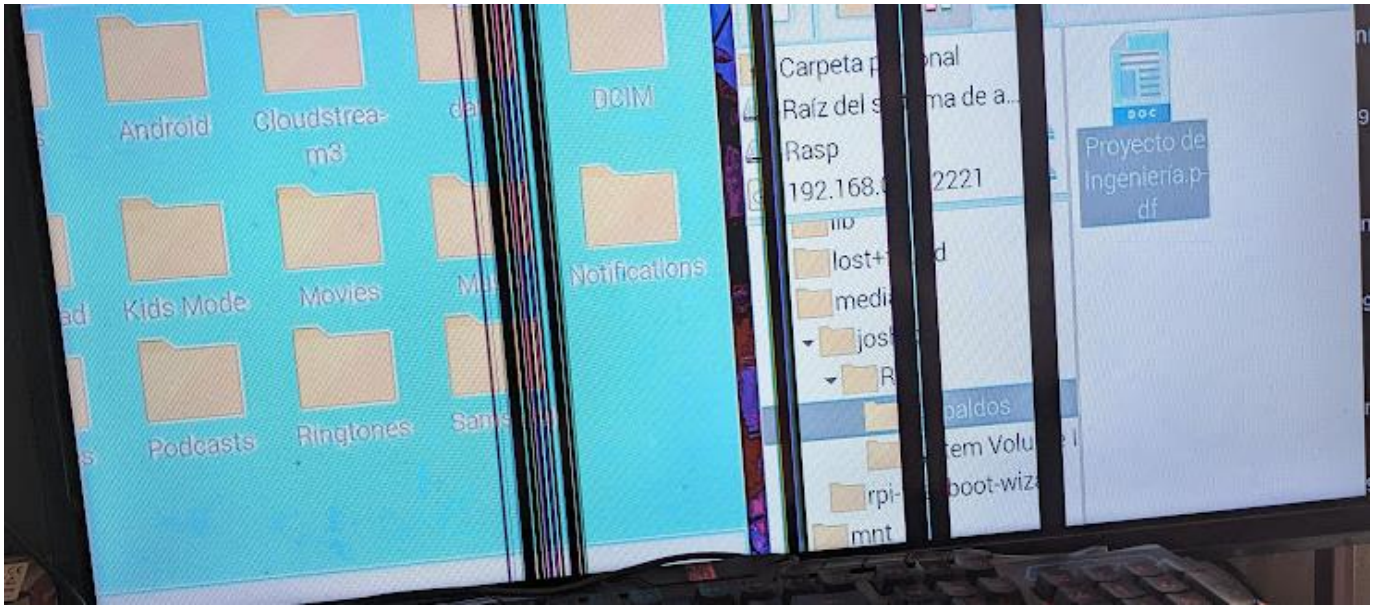


Ilustración 5: Fase 1 de la prueba satisfactoria

Para abordar el problema del módem, opté por una solución creativa: la creación de un servidor privado FTP en un dispositivo Android antiguo. Este dispositivo estaba conectado a la misma red Wi-Fi que mi Raspberry Pi, lo que me permitía replicar la funcionalidad que tenía la unidad USB conectada al módem. En la ilustración 5, se puede apreciar con claridad el apartado de archivos del servidor FTP Android en el lado izquierdo, que serviría como el destino final para los archivos respaldados. En el lado derecho, visualizo el archivo específico de la Raspberry Pi, denominado "Proyectos de Ingeniería", que deseaba transferir al servidor creado en el dispositivo Android.

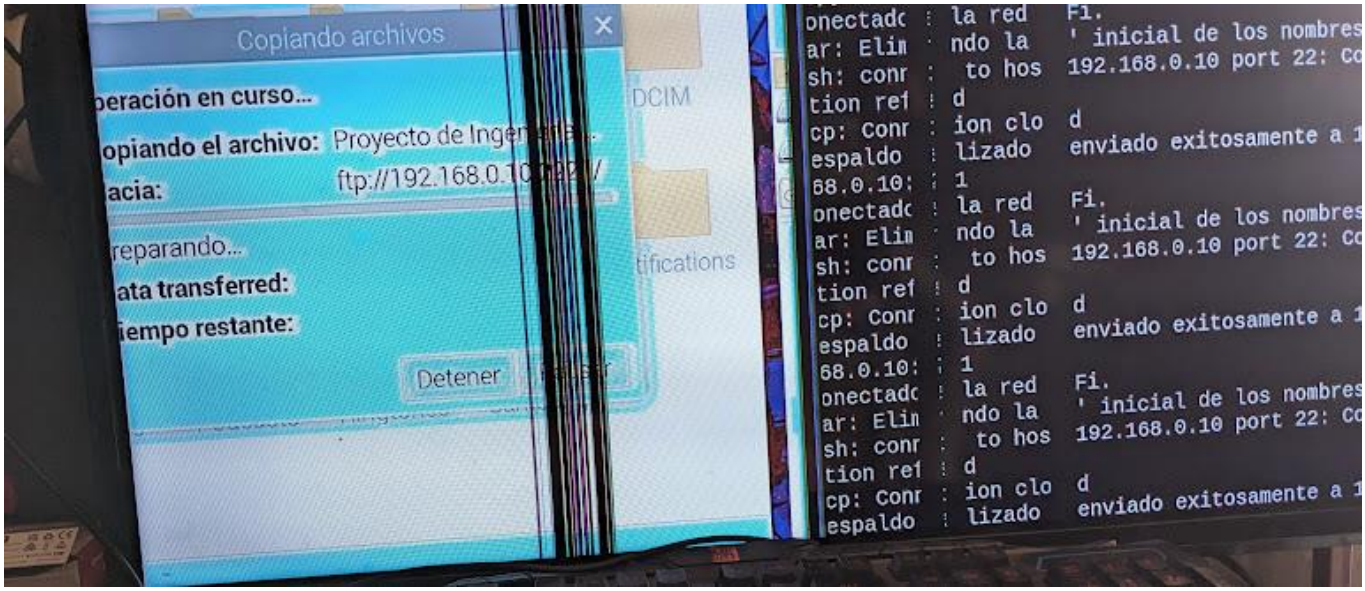


Ilustración 6:Fase 2 de la prueba satisfactoria

La ilustración 6 proporciona detalle del proceso en acción. En el lado derecho, observo cómo el código se está ejecutando y generando el respaldo de manera satisfactoria, lo cual es una señal alentadora de que la solución implementada está funcionando como se esperaba. Al mismo tiempo, en el lado izquierdo, puedo ver el proceso de copia del archivo desde su ubicación original en la Raspberry Pi hacia el servidor FTP que he configurado en el dispositivo Android. La dirección del servidor, ftp://192.168.0.10:2221, proporciona la ruta para la transferencia de archivos.

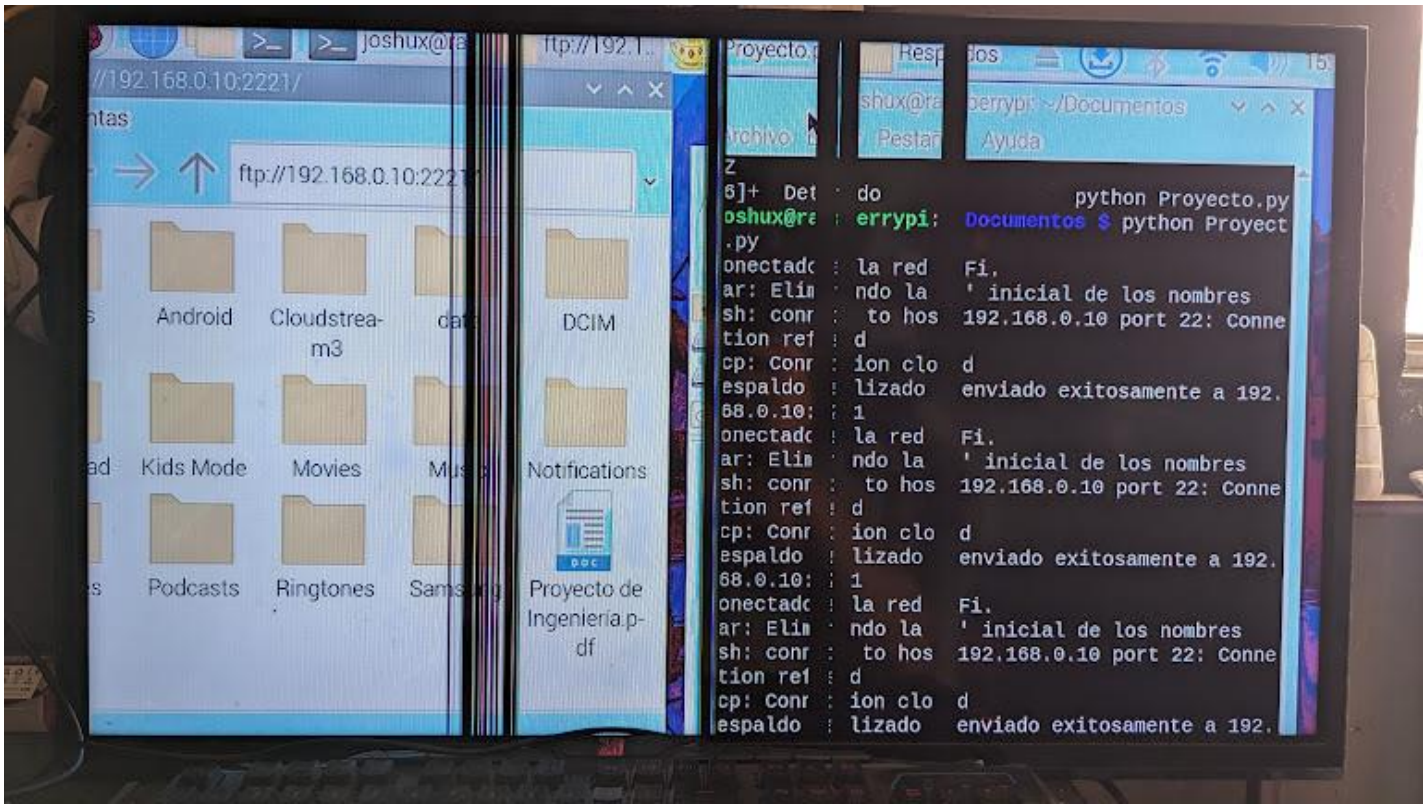


Ilustración 7: Fase final de la prueba satisfactoria

La ilustración presenta el resultado final. En el lado derecho, se muestra un mensaje que confirma que el respaldo se ha realizado exitosamente desde el código. Esta notificación representa un hito importante, indicando que la solución implementada ha sido efectiva en la tarea de respaldo de

datos. Por otro lado, en el lado izquierdo, puedo ver el archivo respaldado en el repositorio, con la carpeta abierta para su visualización. Este paso final confirma que el archivo deseado ha sido transferido con éxito al servidor FTP en el dispositivo Android, completando así el proceso de respaldo de datos de manera satisfactoria.