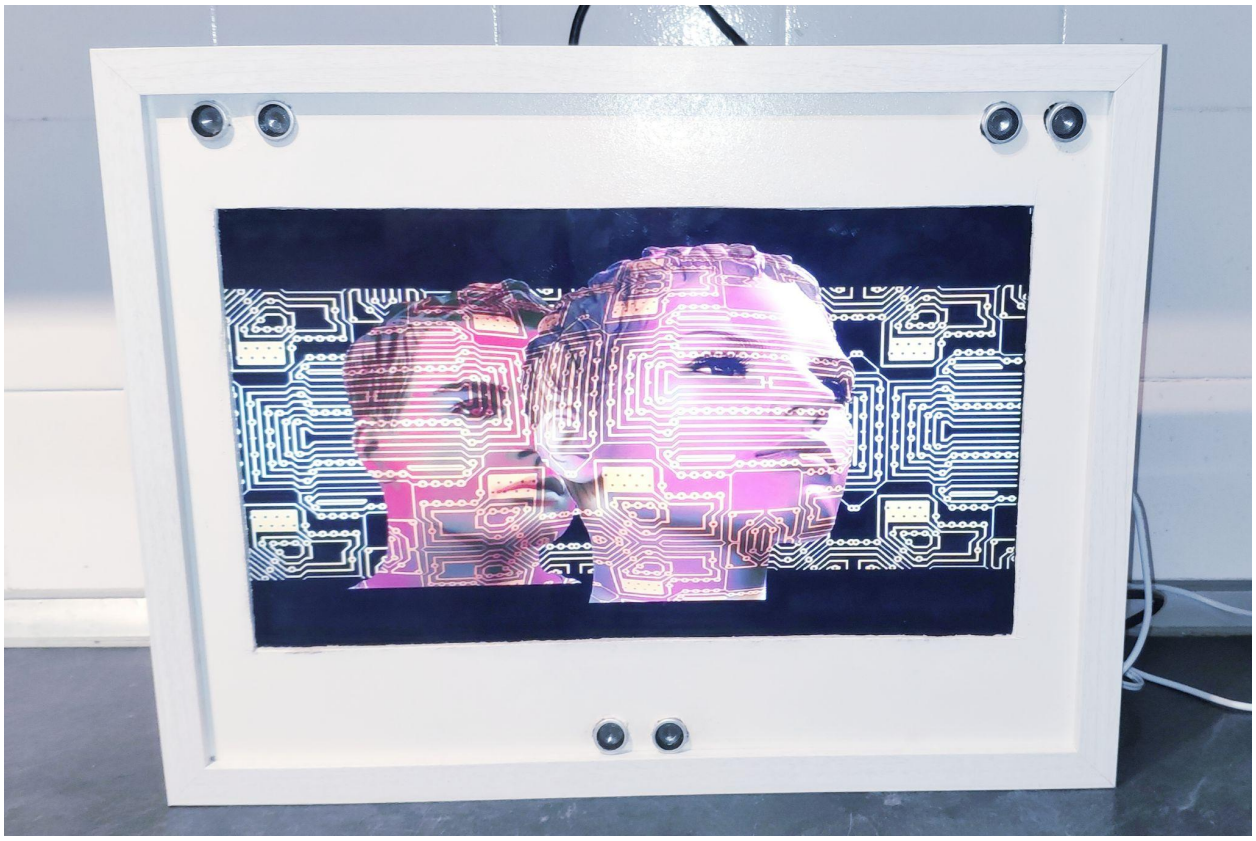


# Marco digital

09/11/2023

—

Agustín De León

# Descripción

El proyecto "Marco Digital" es una solución que combina la funcionalidad de un calendario digital y un marco de fotos para enriquecer la experiencia visual en el hogar u oficina. Este sistema utiliza una Raspberry Pi como núcleo y sensores de ultrasonidos HC-SR04 para interactuar de manera intuitiva con el usuario.

El modo "Calendario" brinda a los usuarios la capacidad de visualizar sus eventos diarios, semanales y mensuales en una interfaz de calendario atractiva. Además, se destacan eventos específicos en el calendario cuando corresponde.

Por otro lado, el modo "Fotos" ofrece la posibilidad de disfrutar de una presentación de diapositivas de imágenes y fotos almacenadas en una carpeta local. Los sensores de proximidad permiten avanzar o retroceder en la presentación con gestos simples, lo que añade una dimensión interactiva a la experiencia visual.

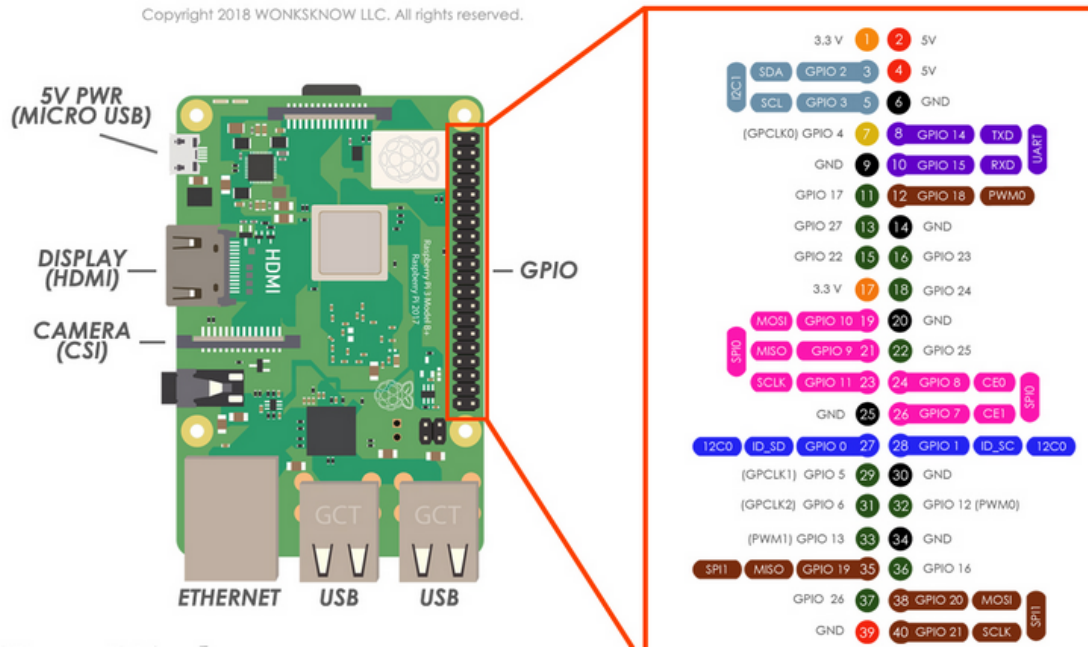
# Objetivos

1. Crear una interfaz de usuario intuitiva que permita a los usuarios cambiar entre el modo "Calendario" y el modo "Fotos" de manera sencilla y sin problemas.
2. Integración con Google Drive y permitir a los usuarios acceder y mostrar archivos almacenados en su cuenta de Google Drive directamente desde el "Marco Digital".
3. Integración con Google Calendar que permita a los usuarios conectarse a su cuenta de Google Calendar y visualizar eventos y citas de forma clara y organizada.
4. Utilizar sensores de ultrasonidos HC-SR04 para detectar gestos o proximidad del usuario y permitir la interacción sin contacto físico.

# Especificaciones

## Hardware

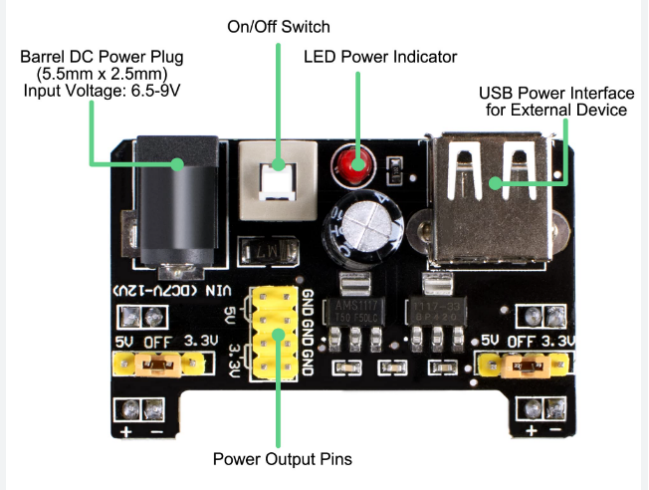
### Placa Raspberry Pi3




### Tarjeta MicroSD


### Cargador Raspberry

Fuente de alimentación

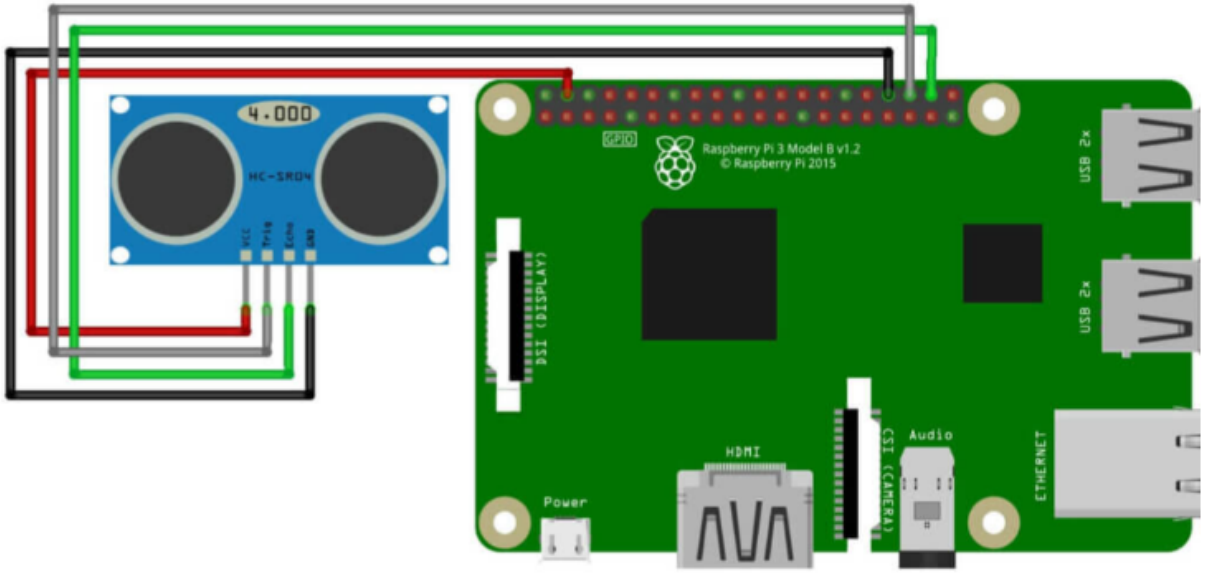



Pantalla 21”

Cable HDMI

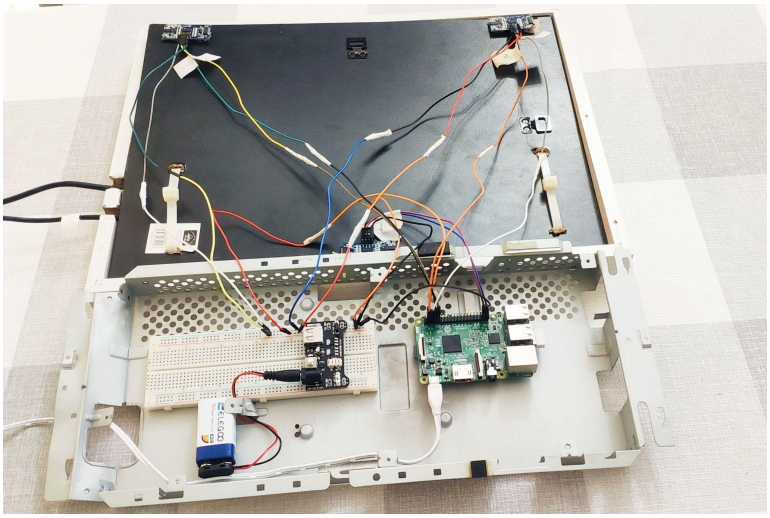
Sensores Ultrasónicos HC-SR04

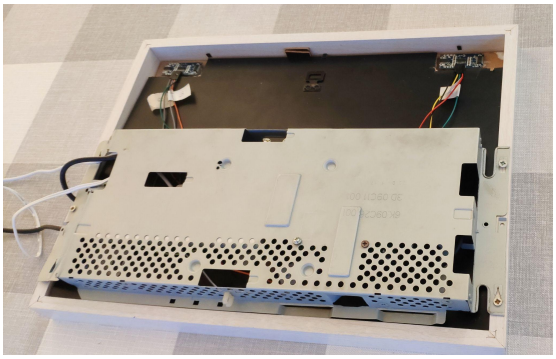
Diagrama de conección de los sensores con la raspberry




```
# HC-SR04 GPIO pins

TRIG_PIN_1 = 2

ECHO_PIN_1 = 3

TRIG_PIN_2 = 14

ECHO_PIN_2 = 15

TRIG_PIN_3 = 20

ECHO_PIN_3 = 21
```

## Ensamblaje de hardware





## Software y configuración

### Sistema Operativo

Raspberry Pi OS 32-bit.
Se instala el sistema operativo desde Windows. En primer lugar, se debe preparar una tarjeta MicroSD con suficiente capacidad y formatearla en FAT32. Luego, se debe quemar la imagen del sistema operativo en la tarjeta SD utilizando Linux File System for Windows. Una vez que la imagen está en la tarjeta SD, esta se puede insertar en la Raspberry Pi y encenderla.
Configuramos la conexión a internet con nuestros datos en el archivo /etc/dhcpcd.conf

### Libraries & Tools

- Pygame

  *pip install pygame*

- Rclone

  *sudo -v ; curl https://rclone.org/install.sh | sudo bash*

- Google API Client.

  *pip install google-auth google-api-python-client*

### Sincronización con Google Calendar

Para habilitar la sincronización con Google Calendar, el proceso comienza con la obtención de las credenciales de la API de Google Calendar, utilizando el tipo de autenticación OAuth 2.0.
En primer lugar, se debe acceder al Google Developers Console de nuestra cuenta de Google, donde se crea un nuevo proyecto y se habilita la API de Google Calendar. Una vez habilitada, se generan credenciales específicas para la aplicación y se descargan en forma de un archivo JSON.

Para configurar la autenticación OAuth 2.0, la aplicación utiliza el archivo JSON de credenciales para autenticarse ante los servicios de Google Calendar. Esto asegura que la aplicación tenga permiso para acceder a los datos del calendario de Google y realizar operaciones relevantes.

Una vez autenticada, la aplicación utiliza la biblioteca google-api-python-client para acceder al calendario de Google. Esta biblioteca facilita la comunicación con la API de Google Calendar y proporciona las funciones necesarias para obtener información sobre los eventos programados.

## Sincronización con Google Drive

La sincronización con Google Drive se logra utilizando la herramienta rclone, que permite gestionar y transferir archivos entre el sistema de archivos local y servicios de almacenamiento en la nube, como Google Drive.
Pero antes tenemos que efectuar los pasos de configuración de la API de google Drive con los pasos mencionados en el punto anterior.

Luego se configura rclone para facilitar la conexión y sincronización entre el sistema de archivos local y la nube de Google Drive.
*rclone config*

Se crea una carpeta local designada para la sincronización de archivos
*mkdir directorio_local/Fotos*

Se crea un archivo para sincronizar las fotos

*nano sync_fotos.sh*

Dentro de este archivo se escribe el siguiente script

```
#!/bin/bash
rclone sync gDrive:Fotos /directorio_local/Fotos
```

Con esta configuración se procede a programar una tarea **cron** para automatizar la sincronización a intervalos de 30 segundos. Se edita el archivo crontab.

*crontab -e*

Se define la frecuencia de la sincronización de nuestro archivo a 30 segundos

** * * * * /usr/bin/rclone sync -v /directorio_local/ remote:directorio_en_drive*

Teniendo todo lo previo configurado se procede con la creación del archivo .py para ejecutar nuestra aplicación.

## Lenguaje de Programación y Código de la aplicación

Python.

NombreArchivo.py

```
import calendar
from datetime import date
import os
import pygame
import time
import RPi.GPIO as GPIO
import json
from googleapiclient.discovery import build
from google.oauth2.credentials import Credentials

# Initialize Pygame
pygame.init()

# Set the display to full-screen
screen = pygame.display.set_mode((0, 0), pygame.FULLSCREEN)

# Path to the folder containing your photos
photo_folder = '/home/mirror/Desktop/Fotos'

# Get a list of image files in the folder
photo_files = [f for f in os.listdir(photo_folder) if f.lower().endswith(('.png', '.jpg', '.jpeg', '.gif', '.bmp'))]

# Create a clock to control the slide show timing
clock = pygame.time.Clock()
photo_index = 0
running = True

# HC-SR04 GPIO pins
TRIG_PIN_1 = 2
ECHO_PIN_1 = 3
TRIG_PIN_2 = 14
ECHO_PIN_2 = 15
TRIG_PIN_3 = 20
ECHO_PIN_3 = 21

# Set GPIO mode and setup
```

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(TRIG_PIN_1, GPIO.OUT)
GPIO.setup(ECHO_PIN_1, GPIO.IN)
GPIO.setup(TRIG_PIN_2, GPIO.OUT)
GPIO.setup(ECHO_PIN_2, GPIO.IN)
GPIO.setup(TRIG_PIN_3, GPIO.OUT)
GPIO.setup(ECHO_PIN_3, GPIO.IN)

# Function to measure distance1
def measure_distance1():
    # Ensure the trigger pin is low
    GPIO.output(TRIG_PIN_1, False)
    time.sleep(0.5)  # Wait for sensor to settle
    # Generate a short trigger pulse
    GPIO.output(TRIG_PIN_1, True)
    time.sleep(0.00001)
    GPIO.output(TRIG_PIN_1, False)
    # Wait for the echo to start
    while GPIO.input(ECHO_PIN_1) == 0:
        pulse_start1 = time.time()
    # Wait for the echo to end
    while GPIO.input(ECHO_PIN_1) == 1:
        pulse_end1 = time.time()
    # Calculate the duration of the echo pulse
    pulse_duration1 = pulse_end1 - pulse_start1
    # Calculate distance1 in centimeters
    distance1 = (pulse_duration1 * 34300) / 2
    return distance1

# Function to measure distance2
def measure_distance2():
    # Ensure the trigger pin is low
    GPIO.output(TRIG_PIN_2, False)
    time.sleep(0.2)  # Wait for sensor to settle
    # Generate a short trigger pulse
    GPIO.output(TRIG_PIN_2, True)
    time.sleep(0.00001)
    GPIO.output(TRIG_PIN_2, False)
    # Wait for the echo to start
    while GPIO.input(ECHO_PIN_2) == 0:
        pulse_start = time.time()
    # Wait for the echo to end
    while GPIO.input(ECHO_PIN_2) == 1:
        pulse_end = time.time()
    # Calculate the duration of the echo pulse
    pulse_duration = pulse_end - pulse_start
    # Calculate distance2 in centimeters
    distance2 = (pulse_duration * 34300) / 2
    return distance2

# Function to measure distance3
def measure_distance3():
    # Ensure the trigger pin is low
    GPIO.output(TRIG_PIN_3, False)
    time.sleep(0.2)  # Wait for sensor to settle
    # Generate a short trigger pulse
    GPIO.output(TRIG_PIN_3, True)
    time.sleep(0.00001)
    GPIO.output(TRIG_PIN_3, False)
    # Wait for the echo to start
    while GPIO.input(ECHO_PIN_3) == 0:
        pulse_start = time.time()
    # Wait for the echo to end
    while GPIO.input(ECHO_PIN_3) == 1:
        pulse_end = time.time()
    # Calculate the duration of the echo pulse
```

```
    pulse_duration = pulse_end - pulse_start
    # Calculate distance3 in centimeters
    distance3 = (pulse_duration * 34300) / 2
    return distance3

class CalendarDisplay:
    # Define the required scopes for accessing Google Calendar
    SCOPES = ['https://www.googleapis.com/auth/calendar.readonly']

    def __init__(self):
        # Initialize Pygame
        pygame.init()

        # Set up display dimensions and set it to fullscreen
        self.screen = pygame.display.set_mode((0, 0), pygame.FULLSCREEN)
        pygame.display.set_caption('Google Calendar Display')

        # Colors
        self.white = (255, 255, 255)
        self.black = (0, 0, 0)
        self.transparent = (255, 255, 255, 128)

        # Create a clock to control the frame rate
        self.clock = pygame.time.Clock()

        # Initialize sensor distances
        self.sensor1_distance = 0
        self.sensor2_distance = 0

        # Get the current month and year
        today = date.today()
        self.current_year = today.year
        self.current_month = today.month

        # Load credentials
        self.load_credentials()

    def load_credentials(self):
        # Load credentials from the JSON file
        CLIENT_CREDS_PATH = '/home/mirror/PhotoFrame/gCalendar/cred.json'
        with open(CLIENT_CREDS_PATH, 'r') as creds_file:
            creds_data = json.load(creds_file)

        # Create a credentials object using the loaded data
        self.creds = Credentials.from_authorized_user_info(creds_data, scopes=self.SCOPES)

        # Create a service for the Calendar API
        self.service = build('calendar', 'v3', credentials=self.creds)

    def render_calendar(self, current_year, current_month, events):

        # Get the days of the month as a 2D list (grid)
        cal = calendar.monthcalendar(current_year, current_month)

        # Get the name of the current month
        month_name = date(current_year, current_month, 1).strftime("%B %Y")

        # Define the day labels
        day_labels = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"]

        # Render the current month at the top of the screen
        font = pygame.font.Font(None, 50)
        text = font.render(month_name, True, self.black)
        x = self.screen.get_width() // 2 - text.get_width() // 2
        y = 40
        self.screen.blit(text, (x, y))
```

```
        # Render the day labels
        font = pygame.font.Font(None, 30)
        for i, label in enumerate(day_labels):
            text = font.render(label, True, self.black)
            x = i * (self.screen.get_width() // 7) + (self.screen.get_width() // 14) - 15
            y = 140
            self.screen.blit(text, (x, y))

        # Render the calendar grid
        x = 0
        y = 180
        cell_width = self.screen.get_width() // 7
        cell_height = self.screen.get_height() // 7

        for week in cal:
            for day in week:
                if day != 0:
                    # Render square with a black border
                    pygame.draw.rect(self.screen, self.white, (x, y, cell_width, cell_height), 1)

                    # Render day number
                    font = pygame.font.Font(None, 30)
                    text = font.render(str(day), True, self.black)
                    text_rect = text.get_rect(center=(x + cell_width // 2, y + cell_height // 5))
                    self.screen.blit(text, text_rect)

                    # Check if there are events for the day
                    events_for_day = [event for event in events if event['start'].get('date') == f'{current_year:04d}-{current_month:02d}-{day:02d}']

                    if events_for_day:
                        event_x = x + cell_width // 4
                        event_y = y + cell_height // 4

                        for event in events_for_day:
                            summary = event.get("summary", "")
                            font = pygame.font.Font(None, 19)  # Set font size to 19px

                            # Render square for each event
                            event_rect = pygame.draw.rect(self.screen, self.transparent, (event_x, event_y, cell_width - 10, cell_height - 4), 1)
                            text = font.render(summary, True, self.black)
                            text_rect = text.get_rect(center=event_rect.center)
                            self.screen.blit(text, text_rect)
                            event_y += cell_height

                x += cell_width
            x = 0
            y += cell_height

    def run(self):
        running = True
        while running:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    running = False
            last_day = calendar.monthrange(self.current_year, self.current_month)[1]

            # Fetch events for the current month
            start_date = f'{self.current_year}-{self.current_month:02d}-01T00:00:00Z'
            end_date = f'{self.current_year}-{self.current_month:02d}-{last_day:02d}T23:59:59Z'
            events = self.fetch_events(start_date, end_date)

            # Clear the screen
            self.screen.fill(self.white)

            # Render the calendar with events
```

```
        self.render_calendar(self.current_year, self.current_month, events)

        pygame.display.update()
        self.clock.tick(30)

        # Check for sensor input to change the month
        self.sensor1_distance = measure_distance1()
        self.sensor2_distance = measure_distance2()
        self.sensor3_distance = measure_distance3()

        if self.sensor3_distance < 10:
            # Switch to photo display
            displayPhotos = PhotosDisplay()
            displayPhotos.run()

        if self.sensor1_distance < 20 or self.sensor2_distance < 20:
            # Move to the previous or next month
            if self.sensor2_distance < 20:
                self.current_month = self.current_month + 1 if self.current_month < 12 else 1
                self.current_year = self.current_year + 1 if self.current_month == 1 else self.current_year
            else:
                self.current_month = self.current_month - 1 if self.current_month > 1 else 12
                self.current_year = self.current_year - 1 if self.current_month == 12 else self.current_year

    # Quit Pygame
    pygame.quit()

  def fetch_events(self, start_date, end_date):
    last_day = calendar.monthrange(self.current_year, self.current_month)[1]

    # Fetch events from the Google Calendar API for a given date range
    end_date = f'{self.current_year}-{self.current_month:02d}-{last_day:02d}T23:59:59Z'
    end_date = f'{self.current_year}-{self.current_month:02d}-28T23:59:59Z'
    events_result = self.service.events().list(
        calendarId='primary',
        timeMin=start_date,
        timeMax=end_date,
        maxResults=10,
        singleEvents=True,
        orderBy='startTime'
    ).execute()
    return events_result.get('items', [])

class PhotosDisplay:
  def __init__(self):
    self.photo_index = 0
    self.running = True  # Initialize the running variable for the class

  def run(self):
    while self.running:  # Use self.running to access the global variable
        for event in pygame.event.get():
            if event.type == pygame.QUIT or event.type == pygame.KEYDOWN:
                self.running = False  # Update self.running

        distance1 = measure_distance1()
        if distance1 < 30:
            self.photo_index = (self.photo_index - 1) % len(photo_files)
            time.sleep(1)

        distance2 = measure_distance2()
        if distance2 < 30:
            self.photo_index = (self.photo_index + 1) % len(photo_files)
            time.sleep(0.5)

        current_photo = pygame.image.load(os.path.join(photo_folder, photo_files[self.photo_index]))
        current_photo = pygame.transform.scale(current_photo, screen.get_size())
```

```
        screen.blit(current_photo, (0, 0))
        pygame.display.flip()
        clock.tick()

        distance3 = measure_distance3()
        if distance3 < 10:
        # Switch to Calendar display
            displayCalendar = CalendarDisplay()
            displayCalendar.run()

    GPIO.cleanup()
    pygame.quit()

if __name__ == '__main__':
    # Create an instance of the CalendarDisplay class
    display = CalendarDisplay()
    # Start the main loop
    display.run()
```

# Conclusiones

## Lecciones Aprendidas:

En el proceso de selección de software, se realizaron pruebas con diferentes alternativas para mejorar la interfaz gráfica del calendario. Sin embargo, se descubrió que algunas de estas opciones no eran factibles debido a las limitaciones de recursos de hardware disponibles. Por ejemplo, se evaluó el uso de Selenium para controlar el navegador en el que se ejecuta el calendario de Google, pero se encontró que esta solución provocaba una notable lentitud en el rendimiento.

## Recomendaciones para Futuros Proyectos:

Dedicar más tiempo a la investigación de software, elección del lenguaje de programación y descubrimiento de accesorios antes de iniciar el proyecto.

# Bibliografía


I. Raspberry Pi

https://www.raspberrypi.com/

II. Google Calendar API

https://developers.google.com/calendar/api/guides/overview?hl=es-419

III. Google Drive API

https://developers.google.com/drive/api/guides/about-sdk?hl=es-419

IV. rClone

https://rclone.org/

V. Cron

https://es.wikipedia.org/wiki/Cron_(Unix)

VI. ChatGPT

Para el desarrollo de este proyecto, se emplearon recursos de procesamiento de lenguaje natural, incluyendo el modelo de lenguaje GPT-3.5 desarrollado por OpenAI. ChatGPT se utilizó como una herramienta valiosa para generar contenido, responder preguntas y ofrecer asistencia, lo que contribuyó significativamente